STREAM CIPHERS: AN OVERVIEW

Applied Cryptography

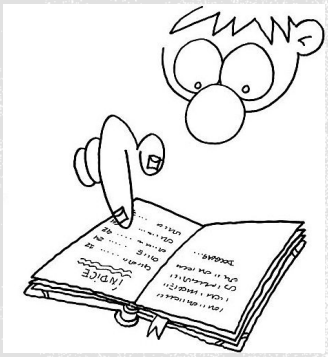
Silvio Ranise [silvio.ranise@unitn.it or *ranise@fbk.eu*]

UNIVERSITÀ DI TRENTO

FONDAZIONE BRUNO KESSLER

CONTENTS

- Stream ciphers
  - Linear Feedback Shift Registers (LFSR)
  - Connections of LFSRs, Finite Fields, and Linear Algebra
  - An example: DVD encryption
- Examples of stream ciphers
  - A family of ciphers: A5
    - GSM
  - E0
    - Bluetooth
  - RC4
    - WEP
    - WPA2 and the KRACK Attack



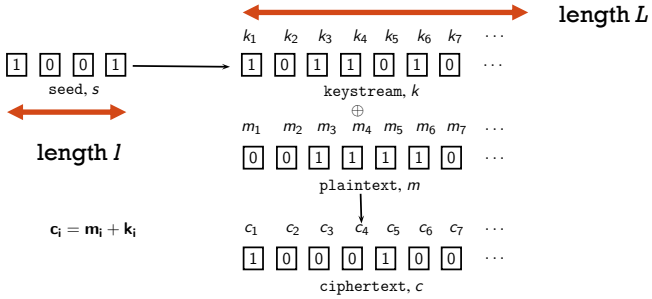
2 STREAM CIPHERS

HOW TO BUILD A STREAM CIPHER?

- The key must be randomly generated as long as possible (is it possible?)
- Alice and Bob know the encryption key (how can we do that??)
- Only Alice and Bob (and, of course, who generates the key) know the encryption key
- Must we always suppose that Eve knows the probability distribution of the plaintexts ??
- Is the One time pad used nowadays at all?

GENERAL STRUCTURE

- Recall that stream ciphers
  - take a seed (a small vector of a few random bits) that must be kept secret
  - **build a keystream (a very long sequence of pseudorandom bits)**
  - xor the keystream with the plaintext bitwise to calculate the ciphertext



DESIDERATA FOR KEYSTREAM GENERATORS

- They should be **fast**
  - I.e. computable in polynomial time as function of the number l of bits in the seed
- The should be **secure**
  - What does it mean for keystream generator to be secure?
  - Intuitively, **a string of L bits produced by a keystream generator should look random**
  - I.e. it should be impossible in a polynomial amount of time in l to distinguish between a truly random bit string of length L and a string of the same length returned by the keystream generator
  - This characterization can be turned into a set of **statistical tests** including
    - Does 1 appear in the output of the generator approximately as often as 0 does?
    - Does a 0 follow a 1 in the output of the generator approximately as often as a 1 follows a 1?
    - Does the string 000 occur in the output of the generator approximately as often as the string 111?
    - …

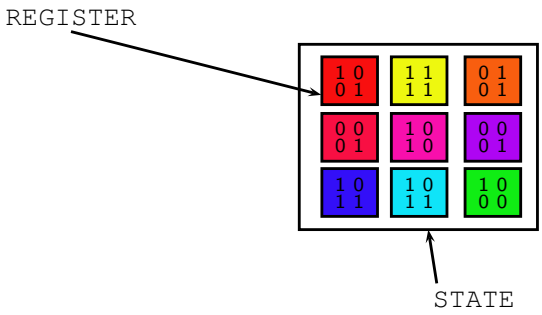
A keystream generator can produce two identical sequences of bits in two distinct locations when using the same seed!

MAIN COMPONENTS

- states
- key loading
- update function
- output function

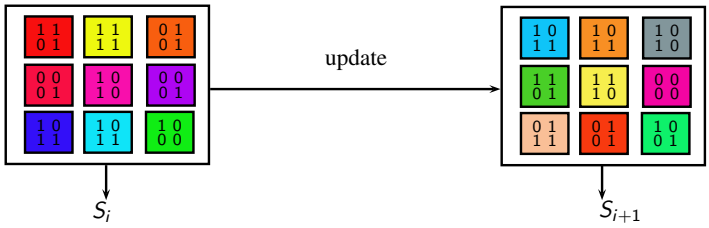
MAIN COMPONENTS

- **States** = vector of bits organized in registers
- key loading
- update function
- output function



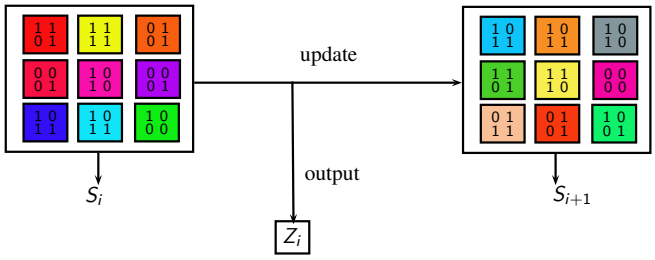
MAIN COMPONENTS

- **States** = vector of bits organized in registers
- key loading
- **update function** = function mapping a state to the next state (clock function)
- output function



MAIN COMPONENTS

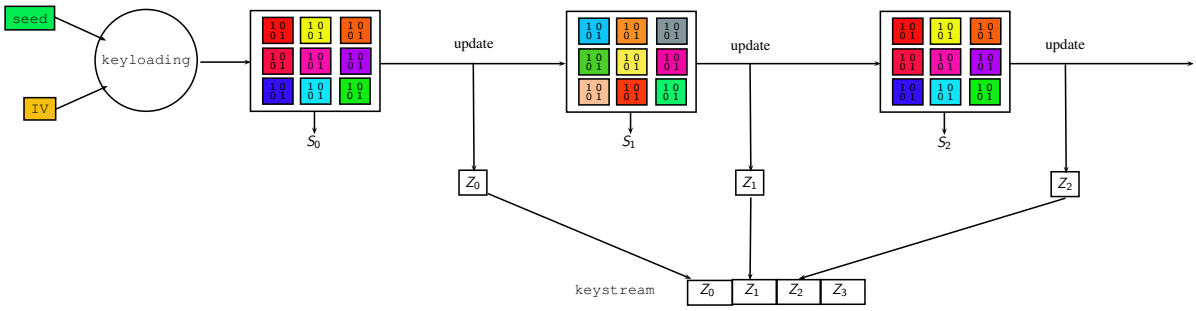
- **States** = vector of bits organized in registers
- key loading
- **update function** = function mapping a state to the next state (clock function)
- **output function** = function extracting a bit from a state
  - Concatenating all bits returned by this function, it is possible to obtain the **keystream**



MAIN COMPONENTS

- **States** = vector of bits organized in registers
- **key loading** = function that takes the **seed** (secret) and a (public) **Initialization Vector (IV)** to compute the initial state for the update function
- **update function** = function mapping a state to the next state (clock function)
- **output function** = function extracting a bit from a state (to compute keystream)

- An IV is an input to a cipher used to construct its initial
- An IV should be, at least, **unique**, i.e. it should not be reused to avoid creating the same initial state again and facilitate the task of an attacker
- In stream ciphers, encryption uniqueness is crucial otherwise plaintexts can be retrieved
  - We will see an example of this problem when discussing WEP…



PUTTING COMPONENTS TOGETHER

- If Eve is able to recover the states, then it breaks the cipher
- A good stream cipher does not allow Eve to find the states by knowing part of the keystream

WARM UP

- The first output bits strongly depend on the initial state
- Hence, it could be easy to recover the initial state by knowing
- To avoid potential problems, it is customary to run a **warm-up phase** before starting encryption
  - This preliminary phase consists in applying the update function several times without outputting any bits of keystream
- The warm-up phase allows for avoiding several practical attacks (e.g., RC4)
- It is a highly recommended security best practice

REMARKS (1)

- If
$$\text{update}(S_i) = S_i$$
- Then
$$S_i = S_{i+1} = S_{i+2} = \ldots$$
- Indeed, it is desirable to avoid as much as possible this situation, as Eve can easily break the cipher and recover all plaintexts

REMARKS (2)

- **Given any initial state, the states are periodic**, since they are in a finite number and at some point we will obtain again one of the previous states
- The keystream is also periodic… this is impossible to avoid
- The smallest number *i* such that

  $$update(\dots (update(S)) \dots) = S$$

  is called the **period of the keystream** (it depends on the initial state)

- Requirement
  - **The period of the keystream shall be quite large**, regardless of the initial state
- This can be achieved by a suitable design of the update function

UPDATE FUNCTION: AN EXAMPLE (1)

- Consider states of 3 bits
- We want to define an update function *f* from 3 bits states to 3 bits states
- We consider the following

$$(x, y, z) \mapsto (y + z, x, y)$$

x	y	z	$\xrightarrow{f}$	$y+z$	x	y
1	0	1		1	1	0

STATES
(0 0 0)
(1 0 0)
(0 1 0)
(0 0 1)
(1 1 0)
(0 1 1)
(1 0 1)
(1 1 1)

UPDATE FUNCTION: AN EXAMPLE (2)

- One can see that the repeated application of *f* on the initial state *(101)* yields

$$(101) \rightarrow (110) \rightarrow (111) \rightarrow (011) \rightarrow (001) \rightarrow (100) \rightarrow (010) \rightarrow (101)$$

with period equal to *7*

- However, if we choose *(000)* as the initial state we obtain

$$(000) \rightarrow (000)$$

and this leads us to discard *(000)* from the set of possible initial states

- **Note**: it is not possible to obtain *(000)* from the repeated application of *f* to an initial state containing at least one occurrence of 1 *(why?)*

UPDATE FUNCTION: ANOTHER EXAMPLE (1)

- Consider states of 3 bits
- We want to define an update function *f* from 3 bits states to 3 bits states
- We consider the following

$$(x, y, z) \mapsto (z, x, y)$$

- The definition is quite similar to the previous but we only perform a shift without additional operation on the first bit…

STATES
(0 0 0)
(1 0 0)
(0 1 0)
(0 0 1)
(1 1 0)
(0 1 1)
(1 0 1)
(1 1 1)

UPDATE FUNCTION: ANOTHER EXAMPLE (1)

- Consider *(101)* as the initial state
- It is easy to see that the repeated application of the update function yields

$$(101) \rightarrow (110) \rightarrow (011) \rightarrow (101)$$

with a very short period, namely 3

- Additionally, observe that

$$(000) \rightarrow (000) \qquad (111) \rightarrow (111)$$

further restricting the usefulness of the this function

COMPARISON BETWEEN UPDATE FUNCTIONS

The first definition of the update function

$$(x, y, z) \mapsto (y + z, x, y)$$

is "better" than the second one

$$(\underline{x}, \underline{y}, z) \mapsto (z, \underline{x}, \underline{y})$$

20

LINEAR FEEDBACK SHIFT REGISTERS (LFSR)

Or how to build "good" update functions for stream ciphers

Definition

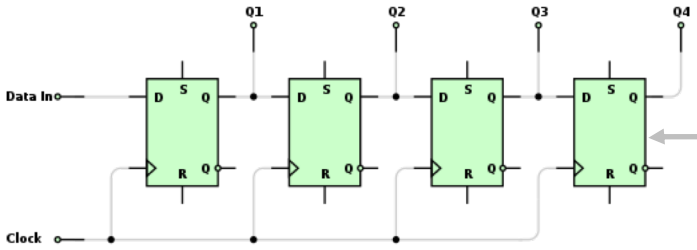
A `linear feedback shift register` *of length n is a shift register composed by n bits.*
At any clock the following operations are executed:

- *the last bit on the right is output and forms part of the keystream*
- *the other bits in the register are shifted to the right by one position*
- *the XOR of some bits of the register are put in the first position on the left.*

A **flip-flop** is a device which stores a single bit (binary digit) of data

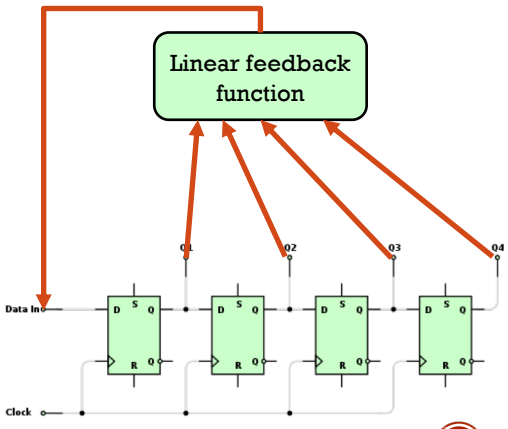
SHIFT REGISTER

- Type of digital circuit using a cascade of flip flops where the output of one flip-flop is connected to the input of the next
- Flip-flops share a single clock signal, which causes the data stored in the system to shift from one location to the next



LINEAR FEEDBACK SHIFT REGISTER (1)

- A **linear-feedback shift register** (**LFSR**) is a shift register whose input bit is a linear function of its previous state
- The most commonly used linear function of single bits is xor
- An LFSR is usually a shift register whose **input bit is driven by the xor of some bits of the overall shift register value**
- The initial value of the LFSR is called the seed
- Since the operation of the register is deterministic, the stream of values produced by the register is completely determined by its current (or previous) state



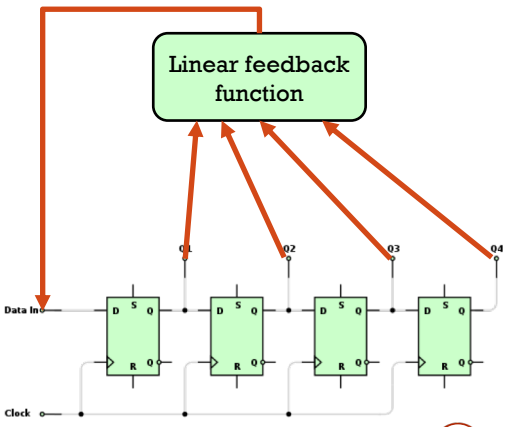
LINEAR FEEDBACK SHIFT REGISTER (2)

- Since the register has a finite number of possible states, it must eventually enter a repeating cycle
- An LFSR with a well-chosen feedback function can produce a sequence of bits that appears random and has a very long cycle
- Typically, the linear feedback function has the following form

$$c_1 q_1 + c_2 q_2 + \cdots + c_n q_n$$

where + denotes xor and c_i is 0 or 1

- The non null c_is are called **taps** and are those that have an influence on the output of the linear feedback function



ALREADY SEEN TWO EXAMPLES

$$(x, y, z) \mapsto (y + z, x, y)$$

Shift right and then overwrite first bit with the xor between first and last bits (all after the shift)

$$(\underline{x}, \underline{y}, z) \mapsto (z, \underline{x}, \underline{y})$$

Simply shift right or, equivalently, shift right and then overwrite first bit with the result of applying the identity function to it

Exercise write the linear feedback functions for the two cases above

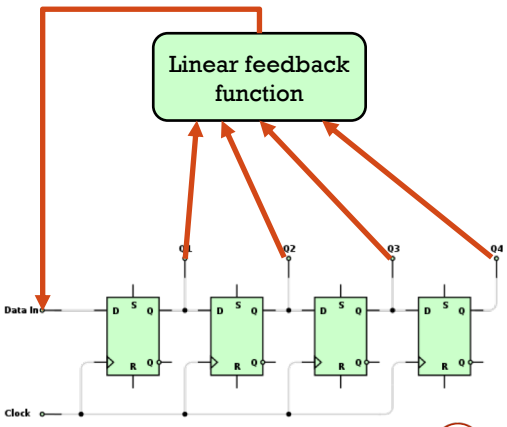
LINEAR FEEDBACK SHIFT REGISTER (3)

- There are only *2^n* possible states
- Assuming that the linear feedback function has the form

$$c_1 q_1 + c_2 q_2 + \cdots + c_n q_n$$

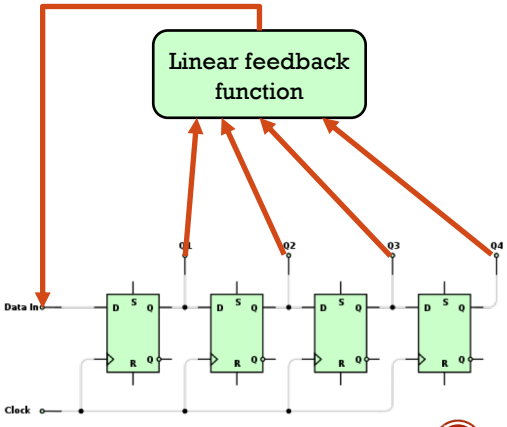
the LFSR must return to its initial state if $c_n = 1$

- If the initial state is the vector consisting of all s, then the machine will remain in that state forever, so we exclude it
- Hence, the maximum number of steps before the LFSR returns to its initial state is $2^n - 1$
- Given a seed *s*, the **period of *s*** is the number of steps the LFSR takes to return to *s*
- The **period of the LFSR** is the maximum period achieved for any seed



LINEAR FEEDBACK SHIFT REGISTER (4)

- If the period of *s* is $2^n - 1$, then the LFSR must visit every non-zero state, and so the period for any seed must be $2^n - 1$; we call such a LFSR *maximal*
- **For every *n*, a maximal LFSR exists**
- Of the $2^n - 1$ possible LFSRs, which taps correspond to maximal LFSRs?
- To answer this question, it is possible to use Linear Algebra or Finite Fields…



This is an informal introduction aiming to convey ideas with no attempt to mathematical rigor

LFSR AND FINITE FIELDS (1)

- To fix ideas, consider **addition modulo a certain number *N***
- This is an instance of the notion of **group**, namely
  - a set closed under a binary operation (addition modulo *N*)
  - the operation is associative (addition modulo *N* is so)
  - there is an identity (0)
  - each element has an inverse (e.g., 1 is the inverse of 11 when considering addition modulo *N*=12 as adding them gives 0)
- The **order of a group** is the number of elements in its set (in the case of addition modulo a number *N*, the order of the group is *N*)
- The **order of an element** *a* in a group is the number of times one needs to apply the operation to *a* to produce the identity
  - Example, the element 4 in the group modulo 12 has order 3 since 4+4+4=0

This is an informal introduction aiming to convey ideas with no attempt to mathematical rigor

LFSR AND FINITE FIELDS (2)

- Now, consider **multiplication modulo a certain number *N***
- It is interesting to consider the order of its various elements that may be regarded to form **cycles**, containing the elements obtained by repeatedly applying the operation to the initial element up to its order, covering all (in case the order of the element is *N)* or some of the *N* elements
  - Example when *N=13*
  - 3 has order 3 since 3*3 mod 13 = 9, 9*3 mod 13 = 1 and 1*3 mod 13 = 3
    - Cycle generated by 3 is the set { 3, 9, 1 }
  - 2 has order 12 since 2*2 mod 13 = 4, 4*2 mod 13 = 8, 8*2 mod 13 = 3, 3*2 mod 13 = 6, …
    - Cycle generated by 2 is the set { 2, 4, 8, 3, 6, 12, 11, 9, 5, 10, 7, 1 }
  - …
  - It is not difficult to see that there are other elements that can generate cycles containing all 12 elements of the group, namely 6, 7, and 11
  - Elements whose order is equal to the order of the group are called **generators**

This is an informal introduction aiming to convey ideas with no attempt to mathematical rigor

LFSR AND FINITE FIELDS (3)

- **Multiplication modulo a prime number** forms a group with similar properties to those discussed considering multiplication modulo 13
- The order of an element is always a divisor of the group order (**Lagrange's theorem**)
- Because some elements are generators of the group itself, it is called a **cyclic group**
- The number of generators of the "group multiplication modulo a prime number *p*" having the full order *p−1*, is $\varphi(p-1)$ where $\varphi(.)$ is **Euler's totient function**, namely

$$\varphi(n) = n\prod_{p|n}\left(1-\frac{1}{p}\right)$$

It counts the positive integers up to a given integer *n* that are relatively prime to *n*

- For *p=13* in the previous example, we have that $\varphi(12) = 12 * \frac{1}{2} * \frac{2}{3} = 4$ which corresponds to what we have observed, i.e. that the 4 elements 2, 6, 7, and 11 are generators

This is an informal introduction aiming to convey ideas with no attempt to mathematical rigor

LFSR AND FINITE FIELDS (4)

- Consider again a generator *g=2* of the group multiplication modulo the prime *p=13*
- We can write
  - g^0 to denote 1
  - g^1 to denote g
  - g^2 to denote g*g mod 13
  - …
- This allows us to list the elements of the group as powers of the generator, i.e.
  - 1 = g^0
  - 2 = g^1
  - 4 = g^2
  - 8 = g^3
  - …

This can be done for any group modulo a prime

This is an informal introduction aiming to convey ideas with no attempt to mathematical rigor

LFSR AND FINITE FIELDS (5)

- A **ring** is a group such that
  - the group binary operation is commutative
  - has another binary operation which is
    - closed over the ring's elements
    - associative
    - has an identity element
    - distributes over the group operation
- A **field** is a ring where both operations are commutative and have inverses
- Intuitively
  - a ring has addition, subtraction, and multiplication well-defined
  - a field adds division
- Integer arithmetic modulo a prime is a finite field

A finite field has
- a finite set of elements
- two binary operations, which are abstract analogues to addition and multiplication
- analogues to subtraction and division using additive and multiplicative inverses

Finite fields are also called Galois fields

This is an informal introduction aiming to convey ideas with no attempt to mathematical rigor

LFSR AND FINITE FIELDS (6)

- The simplest Galois field is GF(2)
  - it contains two elements: 0 and 1
  - the binary operations are addition and multiplication both modulo 2

+	0	1
0	0	1
1	1	0

×	0	1
0	0	0
1	0	1

- On to of GF(2), it is possible to construct
  - GF(2)[x] = the set of polynomials in x with only the elements of GF(2) allowed as coefficients
  - GF(2)[x]/p = the quotient of the set of polynomials by p
  - Example, consider the polynomial p(x)=x^4+x+1. In this case, GF(2)[x]/p contains all possible polynomials of degree less than 4, namely

$$\begin{array}{cccc} 0 & 1 & x & x+1 \\ x^2 & x^2+1 & x^2+x & x^2+x+1 \\ x^3 & x^3+1 & x^3+x & x^3+x+1 \\ x^3+x^2 & x^3+x^2+1 & x^3+x^2+x & x^3+x^2+x+1 \end{array}$$

The 16 elements are obtained by
- adding elements → add coefficients modulo 2
- multiplying elements → multiply polynomials and taking the remainder modulo p

This is an informal introduction aiming to convey ideas with no attempt to mathematical rigor

LFSR AND FINITE FIELDS (7)

- If we try to use the monomial x as generator, we obtain

$$
\begin{aligned}
x^0 &= 1 \\
x^1 &= x \\
x^2 &= x^2 \\
x^3 &= x^3 \\
x^4 &= x^4 - p(x) \\
&= x + 1 \\
x^5 &= x(x^4) \\
&= x^2 + x \\
x^6 &= x^3 + x^2 \\
x^7 &= x^4 + x^3 - p(x) \\
&= x^3 + x + 1 \\
x^8 &= x^4 + x^2 + x - p(x) \\
&= x^2 + 1 \\
x^9 &= x^3 + x
\end{aligned}
\qquad
\begin{aligned}
x^{10} &= x^4 + x^2 - p(x) \\
&= x^2 + x + 1 \\
x^{11} &= x^3 + x^2 + x \\
x^{12} &= x^4 + x^3 + x^2 - p(x) \\
&= x^3 + x^2 + x + 1 \\
x^{13} &= x^4 + x^3 + x^2 + x - p(x) \\
&= x^3 + x^2 + 1 \\
x^{14} &= x^4 + x^3 + x - p(x) \\
&= x^3 + 1 \\
x^{15} &= x^4 + x - p(x) \\
&= 1 = x^0
\end{aligned}
$$

- Except for 0, we have covered all elements in the group
- This implies that we can express all of the nonzero elements as powers of x

This is an informal introduction aiming to convey ideas with no attempt to mathematical rigor

LFSR AND FINITE FIELDS (8)

- Making x a generator depends on the polynomial p that is chosen
- For instance, consider the polynomial p(x)=x^4+1 then considering the elements generated we obtain

$$
\begin{aligned}
x^0 &= 1 \\
x^1 &= x \\
x^2 &= x^2 \\
x^3 &= x^3 \\
x^4 &= x^4 - p(x) \\
&= 1 = x^0
\end{aligned}
$$

- Indeed, the monomial x is not a generator of the group
- Fortunately, it is possible to characterize when a polynomial p allows for the monomial x to become a generator of GF(2)[x]/p…

This is an informal introduction aiming to convey ideas with no attempt to mathematical rigor

LFSR AND FINITE FIELDS (9)

- Polynomials p(x) such that x is a generator of GF(2)[x]/p(x) are called **primitive polynomials**
- In practice, this means that it is possible to start with 1 and, by using the procedure below, it is possible to generate all the elements of GF(2)[x]/p(x)
  - consider 1
  - multiply by x and if the result contains a term x^N with N = degree of p(x), then subtract p(x)
  - stop when the result is 1, this happens when the power of the generator is *2^N − 1*

This is an informal introduction aiming to convey ideas with no attempt to mathematical rigor

LFSR AND FINITE FIELDS (10)

- Primitive polynomials p(x) allow us to design maximal LFSRs by exploiting the following correspondence between these and GF(2)[x]/p(x)
  - State of an LFSR ⟷ Nonzero element of GF(2)[x]/p(x)
  - Period of an LFSR ⟷ Order of GF(2)[x]/p(x)
  - state update function (shift left, xor with taps) ⟷ generation procedure (multiply by x and subtract p(x) when necessary)
  - State of an LFSR ⟷ element x^k
  - Taps ⟷ Nonzero coefficients of p(x)

In conclusion, GF(2)[x]/p(x) "corresponds" to GF(2^N) when p(x) is a primitive polynomial of degree N

k	LFSR coefficients 10011	$GF(2)[x]/p(x)$, $p(x) = x^4 + x + 1$
0	$S[0]$ =0001	$x^0 = 0 + 0 + 0 + 1$
1	$S[1]$ =0010	$x^1 = 0 + 0 + x + 0$
2	$S[2]$ =0100	$x^2 = 0 + x^2 + 0 + 0$
3	$S[3]$ =1000	$x^3 = x^3 + 0 + 0 + 0$
4	$S[4]$ =0011	$x^4 = 0 + 0 + x + 1$
5	$S[5]$ =0110	$x^5 = 0 + x^2 + x + 0$
6	$S[6]$ =1100	$x^6 = x^3 + x^2 + 0 + 0$
7	$S[7]$ =1011	$x^7 = x^3 + 0 + x + 1$
8	$S[8]$ =0101	$x^8 = 0 + x^2 + 0 + 1$
9	$S[9]$ =1010	$x^9 = x^3 + 0 + x + 0$
10	$S[10]$ =0111	$x^{10} = 0 + x^2 + x + 1$
11	$S[11]$ =1110	$x^{11} = x^3 + x^2 + x + 0$
12	$S[12]$ =1111	$x^{12} = x^3 + x^2 + x + 1$
13	$S[13]$ =1101	$x^{13} = x^3 + x^2 + 0 + 1$
14	$S[14]$ =1001	$x^{14} = x^3 + 0 + 0 + 1$
15	$S[15]$ =0001	$x^{15} = 0 + 0 + 0 + 1$

Bits (n)	Feedback polynomial	Taps	Taps (hex)	Period ($2^n - 1$)
2	$x^2 + x + 1$	11	0x3	3
3	$x^3 + x^2 + 1$	110	0x6	7
4	$x^4 + x^3 + 1$	1100	0xC	15
5	$x^5 + x^3 + 1$	10100	0x14	31
6	$x^6 + x^5 + 1$	110000	0x30	63
7	$x^7 + x^6 + 1$	1100000	0x60	127
8	$x^8 + x^6 + x^5 + x^4 + 1$	10111000	0xB8	255
9	$x^9 + x^5 + 1$	100010000	0x110	511
10	$x^{10} + x^7 + 1$	1001000000	0x240	1,023
11	$x^{11} + x^9 + 1$	10100000000	0x500	2,047
12	$x^{12} + x^{11} + x^{10} + x^4 + 1$	111000001000	0xE08	4,095
13	$x^{13} + x^{12} + x^{11} + x^8 + 1$	1110010000000	0x1C80	8,191
14	$x^{14} + x^{13} + x^{12} + x^2 + 1$	11100000000010	0x3802	16,383
15	$x^{15} + x^{14} + 1$	110000000000000	0x6000	32,767
16	$x^{16} + x^{15} + x^{13} + x^4 + 1$	1101000000001000	0xD008	65,535
17	$x^{17} + x^{14} + 1$	10010000000000000	0x12000	131,071
18	$x^{18} + x^{11} + 1$	10000001000000000	0x20400	262,143
19	$x^{19} + x^{18} + x^{17} + x^{14} + 1$	11100100000000000	0x72000	524,287
20	$x^{20} + x^{17} + 1$	10010000000000000000	0x90000	1,048,575
21	$x^{21} + x^{19} + 1$	10100000000000000000	0x140000	2,097,151
22	$x^{22} + x^{21} + 1$	11000000000000000000	0x300000	4,194,303
23	$x^{23} + x^{18} + 1$	10000100000000000000000	0x420000	8,388,607
24	$x^{24} + x^{23} + x^{22} + x^{17} + 1$	11100001000000000000000	0xE10000	16,777,215

HOW TO FIND PRIMITIVE POLYNOMIALS

- https://en.wikipedia.org/wiki/Linear-feedback_shift_register
- Exercise, check that p(x)=x^3+x^2+1 is a primitive polynomial by applying the procedure to generate all elements of GF(2)[x]/p(x)
  - Recall that all operations on coefficients are to taken modulo 2

Fact

Given a maximum-length LFSR of n bits and reading $2^n - 1$ consecutive bits of the m-sequence that it produces, we have that:

1. *one half of the bits are 1 and one half are 0*
   (actually the 1's are one more than the 0's)
2. *there are $\mathbf{2^{n-1}}$ runs:*
   - *1/2 of the runs has length 1*
   - *1/4 of the runs has length 2*
   - *...*
   - *$1/2^i$ of the runs has length i (for $2 \leq i \leq n-2$)*
   - *there is only one run of $n-1$ zeros and none of the runs has $n-1$ ones*
   - *there is only one run of n ones and none of the runs has n zeros.*

LFSRS AS PSEUDORANDOM GENERATORS

- Any binary sequence contains runs of 0 and 1
  - Example: the sequence **000**11**0**111001 is composed by 6 runs of length 3, 2, 1, 3, 2, 1, respectively.
- **m-sequences** have been studied extensively in cryptography because they **simulate the behaviour of a random sequence** very well

Example

We use as feedback polynomial the primitive polynomial $x^5 + x^2 + 1$, *hence the period is* $2^5 - 1 = 31$ *bits and we expect* $2^{5-1} = \mathbf{16}$ *runs. Starting from the state* (10000), *the obtained m-sequence is*

0000 1 00 1 0 11 00 11111 000 11 0 111 0 1 0 1

There are indeed **16** *runs, and*

- 8 *runs have length 1*
- 4 *runs have length 2*
- 2 *runs have length 3*
- *There is only* 1 *run of* 4 *zeros and none of* 4 *ones*
- *There is only* 1 *run of* 5 *ones and none of* 5 *zeros*

LFSR AND LINEAR ALGEBRA (1)

- For analyzing LFSRs, it is useful to investigate their relationships with Linear Algebra
- For this, the key idea is the notion of **companion matrix**
  - Given the polynomial p(x) = c+ c1*x+ …+ cN*x^N in GF(2)[x]
  - Its companion matrix is

$$C(p) = \begin{bmatrix} 0 & 0 & \dots & 0 & c_0 \\ 1 & 0 & \dots & 0 & c_1 \\ 0 & 1 & \dots & 0 & c_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & c_{n-1} \end{bmatrix}$$

- It is possible to show that multiplication by C is equivalent to multiplication by x in GF(2)[x]/p(x)

LFSR AND LINEAR ALGEBRA (2)

- This connection allows us to consider the **state recovery problem**, i.e. predicting the state vector of the LFSR from its output bits
  - If we can solve this we can recover the initial state or, equivalently, the seed
- This problem can be solved by using powers of the companion matrix
  - This requires polynomial time computations
- Unfortunately, this leads to the following observation …

LFSRS: MAIN WEAKNESS (1)

- LFSRs cannot be used directly in cryptography, because of their linearity…
- Given any LFSR of length n, the j-th bit of the keystream, for $j \geq n + 1$, can be obtained as a linear combination of its previous n bits in the keystream
- Let the bits of the output sequence denoted by $a_0, \ldots, a_n$

  There exist n bits $\lambda_0, \ldots, \lambda_n$

  Such that $\sum_{i=0}^{n} \lambda_i a_i = 0, \quad \lambda_n = 1$
- **Note**: the values of $\lambda_0, \ldots, \lambda_n$ do not depend on when Eve starts intercepting the output sequence

LFSRS: MAIN WEAKNESS (2)

- Knowing the values of $\lambda_0, \ldots, \lambda_n$ is equivalent to knowing the feedback polynomial:

$$g = \sum_{i=0}^{n} \lambda_i x^i$$

- And thus to being able to predict the keystream!
- It is possible to use Linear Algebra methods or an appropriate algorithm (Berlekamp-Massey) to compute the desired values in polynomial time

LFSRS: MAIN WEAKNESS (3)

- Once Eve obtains the feedback polynomial, it is able to reproduce the whole keystream from any (short) subsequence of the keystream having length *n*
- Eve may obtain the required subsequence of the keystream mounting a known-plaintext or chosen-plaintext attack
- Hence LFSRs are vulnerable with respect to known-plaintext attacks and they must not be used as keystream generators
  - **Although they can be used as components**!
  - In particular when iterated and combined by using some kind of non-linear functions

46

AN EXAMPLE: DVD ENCRYPTION

A more robust design of a stream cipher by using a non linear function

EXAMPLE: DVD ENCRYPTION (1)

- The **Content Scrambling System** (**CSS**) is used for protecting movies on DVD disks
- It uses the CSS stream cipher to encrypt movie contents using a **40-bit secret key**
  - CSS was introduced in 1996
- Ciphers using 40-bit keys are insecure because of brute force attacks
- The CSS stream cipher is particularly weak as it can be broken in far less time than an exhaustive search over all *2^(40)* seeds
  - This is well within the reach of currently available computational power
  - On a Pentium II (microprocessor commercialized in 1997), the attack takes only 1 minute
  - Recall Moore's Law and imagine how much it can take on modern microprocessors… few seconds or even less*!*

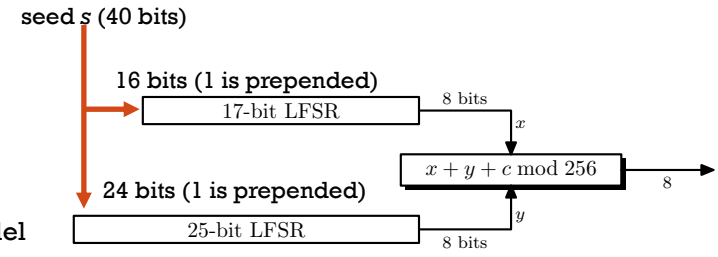
EXAMPLE: DVD ENCRYPTION (2)

- CSS uses LFSRs as hardware implementations are quite cheap
  - This makes LFSRs attractive for low-cost consumer electronics such as DVD players, cell phones, and Bluetooth devices
- To alleviate the problems deriving from the main weakness of LFSRs, the idea is to **run some LFSRs in parallel for some clock cycles and combine their results by means of some non-linear function**
  - The CSS stream cipher combines two LFSRs
  - The A5/1 stream cipher used to encrypt GSM cell phone traffic combines the outputs of three LFSRs
  - The Bluetooth E0 stream cipher combines four LFSRs
  - **All these algorithms have been shown to be insecure and should not be used**: recovering the plaintext takes far less time than an exhaustive search on the key space

EXAMPLE: DVD ENCRYPTION (3)

- Taps for the 17-bit LFSR
  - 14, 0
- Taps for the 25-bit LFSR
  - 12, 4, 3, 0
- The two LFSRs are run in parallel for 8 cycles and then the resulting bits are considered as integers and added modulo 256
  - c is the carry bit of the added

2^(40) attempts at most

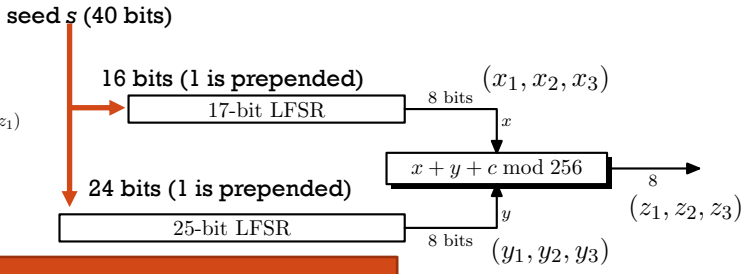


Attack (stupid brute force)
- Suppose Eve is given the first 100 bytes of the output sequence of additions
- Eve guesses a 40 bits seed
- Run for 100 iterations
- Compare results with those of the given sequence
- If equal, then guessed seed is correct, otherwise retry

EXAMPLE: DVD ENCRYPTION (4)

- Observation

$$(2^{16}x_3 + 2^8x_2 + x_1) + (2^{16}y_3 + 2^8y_2 + y_1) \equiv (2^{16}z_3 + 2^8z_2 + z_1)$$



Attack (smart brute force)

- Suppose Eve is given the first 100 bytes of the output sequence of additions and of the output sequence of the 17-bit LFSR
- Eve guesses a 16 bits seed and compute (x_1, x_2, x_3)
- By using the observation above, Eve computes (y_1, y_2, y_3) and is able to determine the seed for the 25 bits LFSR
- Run for 100 iterations
- Compare results with those of the given sequence
- If equal, then guessed seed is correct, otherwise retry

2^(16) attempts at most

MORE INFO ABOUT CSS & WEAKNESSES

- https://en.wikipedia.org/wiki/Content_Scramble_System
- https://www.cs.cmu.edu/~dst/DeCSS/Kesden/index.html

52 A FAMILY OF CIPHERS: A5

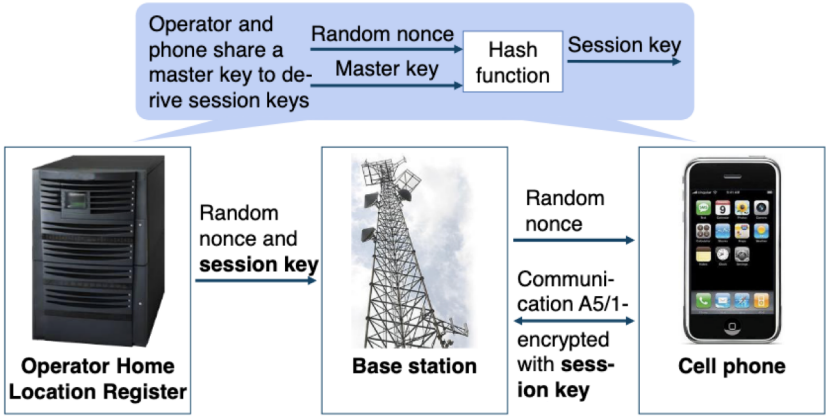
GSM

- ETSI = European Telecommunications Standards Institute (https://www.etsi.org/)
- Independent, not-for-profit, standardization organization

- GSM = Global System for Mobile communication (ETSI standard)
  - Protocols for 2G digital cellular mobile networks
  - GSM was the first near to global standard of wireless telecommunication
  - GSM provides services to more than 6 billion people worldwide (over 90 % market share)

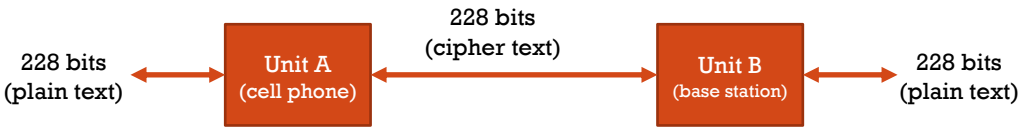
- When the GSM standard was designed from 1982-1991, the level of security specifications regarding both authentication and encryption were limited
  - Specific algorithms were never officially published (**security by obscurity?**)
  - Algorithms were **reverse-engineered** or **leaked**, leading to revelations of several possible attacks
  - Within a few months after the release, most of the cryptographic schemes had been compromised and some were even proven to be close to useless

GSM SETUP AND USE OF A5



AN INTRODUCTION TO A5 (1)

- GSM was originally designed for basic commodity phones and a security scheme could therefore be implemented on top of simple hardware components
- A5 was specified to encrypt over-the-air transmissions on the GSM network



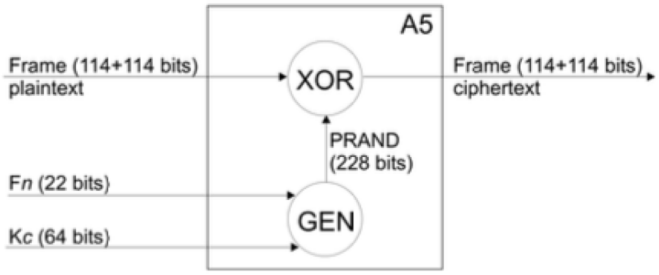
- 1 frame = 228 bits
- 28 frames per second
- 114 bits for **uplink** (mobile to base)
- 114 bits for **downlink** (base to mobile)

- Each frame is numbered by a **frame number**, obtained by a **frame counter** initialized with 0 at conversation-start and incremented by 1 (mod 2^22) with each frame sent

AN INTRODUCTION TO A5 (2)

- An algorithm of the A5 family **takes**
  - the session key *Kc* (symmetric) and
  - a frame counter *Fn*

  generates
  - 228 pseudo random bits (PRAND), called a key stream
- The key stream is then XORed with a 228 bit segment of plain text, yielding 228 bits of ciphertext



- Since encryption is very simple, what does make A5 secure?
- **Answer**: design/implementation of GEN
- Members of the A5 family differs on how GEN is implemented, thereby providing **different levels of security**

A5: SHORT OVERVIEW

- The GSM is de facto the standard protocol for 2G digital cellular networks used by mobile phones
- Data are transmitted in blocks of 228 bits
- First standard to introduce cryptographic algorithms for security
- It comes in 4 variants
  - A5/0: no encryption
  - **A5/1: stream cipher based on LFSRs**
  - **A5/2: stream cipher based on LFSRs**
  - A5/3: stream cipher based on the block cipher KASUMI

A5/0

- It is the weakest of the A5 versions as it **offers no encryption**
- It is a **no-operation cipher**, that generates the pseudo random bits by negating the input frame, thus leaving out the XOR function
- The result is an algorithm that outputs the plain text it received as an input
- This version is found in third world countries or countries with UN sanctions

59

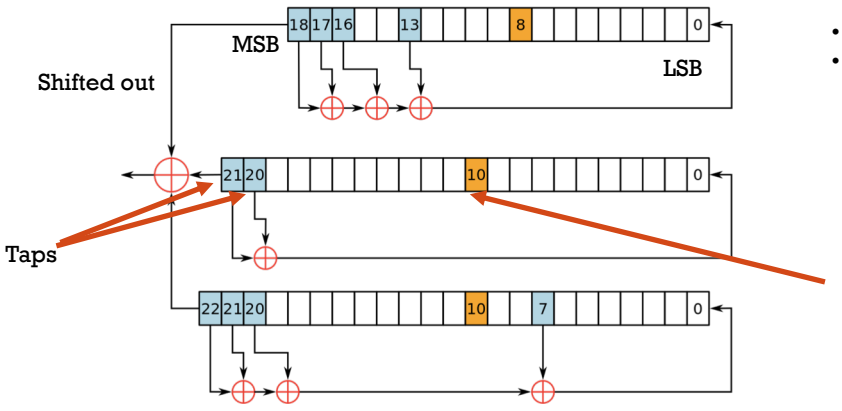
A5/1

A5/1: MAIN FEATURES

The three polynomials are primitives with periods:
- 2^(19)-1 = 524,287 (R1)
- 2^(22)-1 = 4,194,303 (R2)
- 2^(23)-1 = 8,388,607 (R3)

- year: 1987
- keystream length: 228 bits
- number of LFSRs: 3
- update of registers: majority function on the control bits
- Registers
  - lengths: 19, 22, 23 (total: 64 bits of state)
  - R1, R2, R3 are maximum-length LFSRs as the polynomials are primitive

LFSR number	Length in bits	Feedback polynomial	Clocking bit	Tapped bits
1	19	$x^{19} + x^{18} + x^{17} + x^{14} + 1$	8	13, 16, 17, 18
2	22	$x^{22} + x^{21} + 1$	10	20, 21
3	23	$x^{23} + x^{22} + x^{21} + x^{8} + 1$	10	7, 20, 21, 22

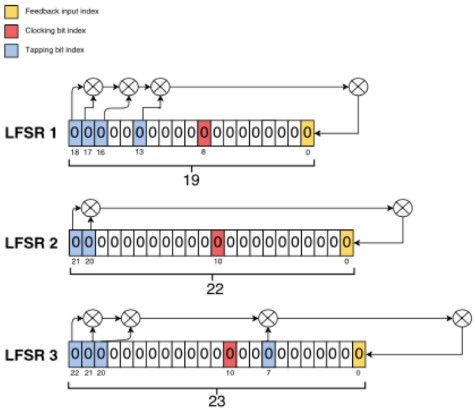


- MSB=Most Significant Bit
- LSB=Least Significant Bit

A5/1: ARCHITECTURE

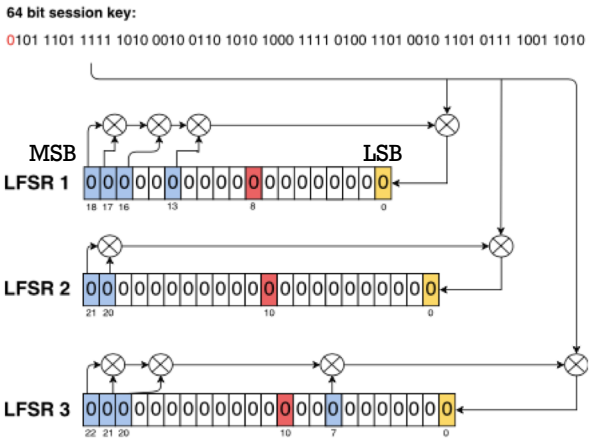
- When a register is clocked, its tapped bits are XORed and the result is stored in the registers LSB
- The registers MSB is shifted out of the register, and its value is forgotten

A5/1: STEP 1, INIZIALIZATION



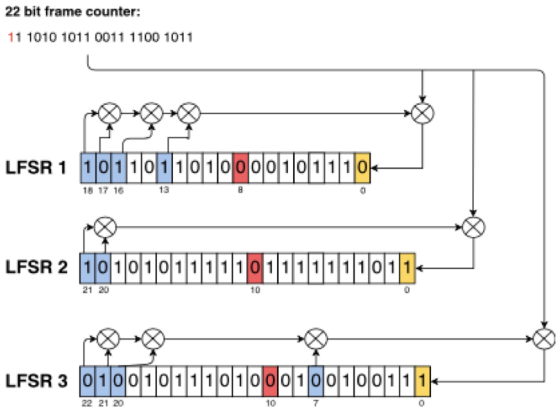
A5/1: STEP 2, CLOCKING WITH SESSION KEY

- After registers are initialized with 0s…
- … they are clocked 64 times
  - one time for each bit in the session key
- The bits of the 64 bit session key are consecutively XORed in parallel with the feedback of the register, and the result is fed into the LSB of the respective register
- First value of the session key is highlighted in red and is ready to be fed into the circuit
- Each cycle will result in a new (unique) state
- This step is finished when the last bit of the session key is fed into the circuit



A5/1: STEP 3, CLOCKING WITH FRAME COUNTER

- At the end of step 2, registers do not more hold only 0s
- This step is similar to previous
- Main difference: instead of session key, a (public) frame counter (Initialization Vector) is used
- The length of the frame counter is 22 bits
- The registers are thus clocked 22 times to inject all bits in the frame counter



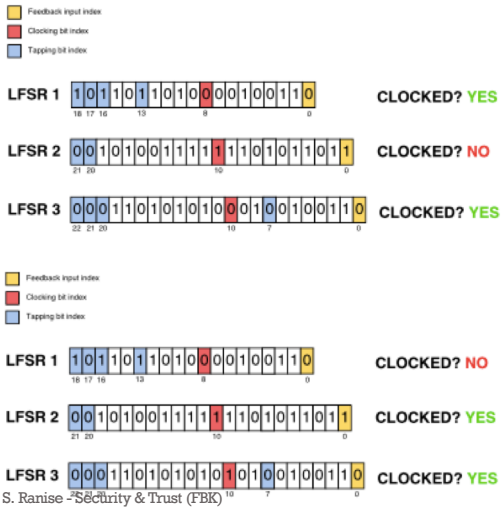
A5/1: STEP 4, CLOCKING WITH MAJORITY

- The registers are clocked 100 times with **irregular clocking**
  - This is where the clock bits come into play
- Irregular clocking follows the majority rule, a decision rule that selects alternatives that have a majority
  - The majority bit is determined by the **clocking bits** of the registers
    - LFSR 1 clocking bit: 8
    - LFSR 2 clocking bit: 10
    - LFSR 3 clocking bit: 10
  - **If a clocking bit of a register is equal to the majority bit, the register is clocked**
  - **Otherwise, the register is left unchanged**

EXAMPLES OF MAJORITY

CB of $R1$: $R1[8]$	0	0	0	0	1	1	1	1
CB of $R2$: $R2[10]$	0	0	1	1	0	0	1	1
CB of $R3$: $R3[10]$	0	1	0	1	0	1	0	1
R1 clocked?	√	√	√	–	–	√	√	√
R2 clocked?	√	√	–	√	√	–	√	√
R3 clocked?	√	–	√	√	√	√	–	√

CB = clocking bit



Maj(LFSR1[8],LFSR2[10],LFSR3[10])=Maj(0,1,0)=0

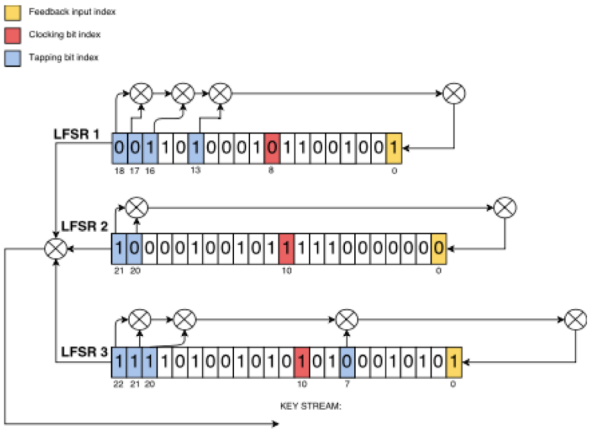
- LFSR1 is clocked as LFSR1[8]=0
- LFSR2 is not clocked as LFSR2[10] != 0
- LFSR3 is clocked as LFSR3[10]=0

Maj(LFSR1[8],LFSR2[10],LFSR3[10])=Maj(0,1,1)=1

- LFSR1 is not clocked as LFSR1[8] != 1
- LFSR2 is clocked as LFSR2[10] = 1
- LFSR3 is clocked as LFSR3[10]=1

A5/1: STEP 5, PRODUCING KEY STREAM

- The registers are **clocked 228 times with irregular clocking**
- For each cycle, the MSB of each register is used for a final computation
  - **Each register's MSB is XORed with one-another, and the result is added to the key stream**
- Since each cycle generates a key stream bit, and we cycle our circuit 228 times, the output key stream will consist of 228 bits



A5/1: STEP 6, CREATING THE CIPHERTEXT

- Now that we have a pseudo random generated key stream, we can encrypt plain texts
- Only 114 bits of the key stream can be used to encrypt our data, as 114 bits will be used to decrypt the incoming message
- To encrypt data, we take a 114 bit chunk of plain text and XOR it bitwise with the 114 bits of key stream dedicated for encryption
- This simple XOR between plain text and key stream creates the cipher text, which we can send **securely** over the GSM network

69 A5/1 IN 3 SLIDES

A5/1: KEY LOADING (STEPS 1-3)

- **Seed**: a (secret) vector *K* of 64 bits (session key)
- **Initialization Vector**: a public vector *IV* of 22 bits (frame counter)
- From *K* and *IV* we get the initial state (namely a vector of 64 bits) using a **key-loading function** *kl(K,IV)*
- The function *kl* is defined by iteration: initially the registers contain all zeros, the bits in *K* are injected by xor-ing them with other bits, and similarly for the bits of *IV*

A5/1: WARM UP (STEP 4)

- After key loading, the registers R1, R2, and R3 are **irregularly clocked** 100 times, without producing output
- The irregular clock works as follows:
  - in each step, the **clock bits**
  - Each one among R1, R2, and R3 is updated if the clock bits agree with the majority of the clock bits
- In this way, it is possible to show that each register clocks with a probability of 3/4
- At this point, the initialization of the registers is complete and the stream cipher is ready to output the key-stream (denoted with *Z*)

A5/1: UPDATE & OUTPUT (STEPS 5 AND 6)

- The key-stream is composed of 228 bits that are obtained in 228 steps
- At each clock tick, the functions are defined as follows
  - **Update**
    - The register R1 is updated if its clock bit agrees with the majority of the others
    - The register R2 is updated if its clock bit agrees with the majority of the others
    - The register R3 is updated if its clock bit agrees with the majority of the others
  - **Output**: the xor of the output of the three registers R1, R2, R3 is the output bit
  - After 228 bits of output, the key-loading phase is executed again

A5/1: SOME ATTACKS IN BRIEF (1)

- In GSM phones, the session keys are produced with another algorithm, which **might depend on the operator**
- Marc Briceno, Ian Goldberg and David Wagner published in 1999 the first complete description of A5/1 and claimed that all the implementations they checked used 54-bit session keys
  - I.e. 64-bit keys with 10 fixed bits set to 0s
- Although other operators could decide otherwise, it is probable that this convention will be maintained by operators in the name of backward compatibility

If you want to have a look at the code implementing A5/1, please, see at https://cryptome.org/jya/a51-pi.htm

From 2^64 to 2^54 key space… a reduction of more than 10^3

A5/1: SOME ATTACKS IN BRIEF (2)

- Anderson and Roe proposed an attack based on guessing the 41 bits in the shorter R1 and R2 registers, and deriving the 23 bits of the R3 register from the output
  - If we assume that a PC can test ten million guesses per second, Anderson and Roe's method would still need more than a month to find one key
- Golic designed a complex attack, where each step is based on the solution of a system of linear equations
  - Even though the attack requires fewer steps, each step is more complex, and does not run run faster than previous attacks on a regular computer
  - J. Golic. *Cryptanalysis of alleged A5 stream cipher*. Proceedings of EUROCRYPT'97, LNCS 1233, pp.239-255,1999.
- For more an almost real-time attack, see https://cryptome.org/a51-bsw.htm

75

A5/2

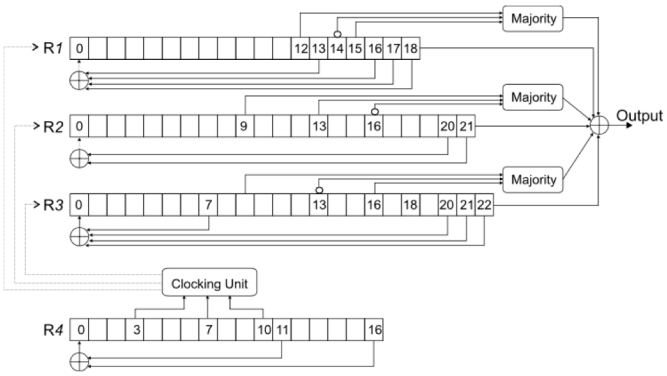
A5/2: MAIN FEATURES

- year: 1989
- keystream length: 228 bits
- Number of LFSRs: 4
- update of registers: majority function on the control bits in R4
- Registers
  - lengths: 19, 22, 23, 17 (total: 81 bits of state)
  - R1, R2, R3, R4 are maximum-length LFSRs as the polynomials are primitive

Register	Feedback Polynomial	Taps	Clock
R_1	$x^{19} + x^5 + x^2 + x + 1$	$0, 1, 2, 5$	
R_2	$x^{22} + x + 1$	$0, 1$	
R_3	$x^{23} + x^{15} + x^2 + x + 1$	$0, 1, 2, 15$	
R_4	$x^{17} + x^5 + 1$	$0, 5$	$6, 9, 13$

A5/2: ARCHITECTURE

- Main differences between A5/1 and A5/2
  - A fourth register
  - A slightly different initialization phase
  - The input bits for the Clocking Unit are taken from R4 only
  - The output is an XOR of
  1. the MSB of R1, R2 and R3
  2. the majority of 3 bits of each register with one of these 3 bits negated:



Register	MSB	Majority bits	Negated majority bit
R_1	$R_1[18]$	$R_1[12]$, $R_1[15]$	$R_1[14]$
R_2	$R_2[21]$	$R_2[9]$, $R_2[13]$	$R_2[16]$
R_3	$R_3[22]$	$R_3[16]$, $R_3[18]$	$R_3[13]$

A5/2: FIRST HALF STEPS

- All registers are set to 0s
- The registers are clocked for 64 cycles:
  - In each cycle i ($0 \leq i \leq 63$) the bit Kc[i] is XOR'ed with the LSB of the register and stored in the LSB of the same register
- The registers are clocked for 22 cycles:
  - In each cycle i ($0 \leq i \leq 21$) the bit Fn[i] is XOR'ed with the LSB of the register and stored in the LSB of the same register
- Set bits R1[15], R2[16], R3[18], R4[10] to 1

A5/2: SECOND HALF STEPS

- The next 99 cycles are run to diffuse Kc and Fn into the registers, discarding the output
  - Irregular clocking is applied: Whether a register is clocked or not is determined during each cycle by the Clocking Unit calculating the majority of all 3 clocking bits R4[3], R4[7], R4[10] - if the majority matches the clocking bit, the corresponding register is clocked
  - Register R4 is always clocked last in every cycle
- The next 228 cycles are again carried out with the same irregular clocking as in the previous step
  Each cycle i ($0 \leq i \leq 227$) the MSBs of all 3 registers are XOR'ed and the result is used as bit i of the output

$R_4[3]$	$R_4[7]$	$R_4[10]$	Majority	R_1	R_2	R_3
0	0	0	0	clock	clock	clock
1	0	0	0	clock		clock
0	1	0	0	clock	clock	
1	1	0	1		clock	clock
0	0	1	0		clock	clock
1	0	1	1	clock	clock	
0	1	1	1	clock		clock
1	1	1	1	clock	clock	clock

80

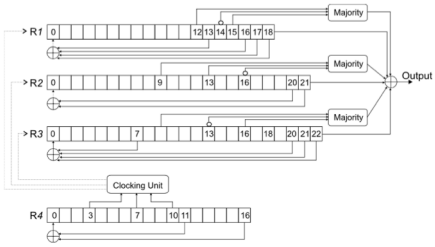
A5/2 IN 2 SLIDES

A5/2: KEY LOADING & WARM UP

- **Seed**: a (secret) vector *K* of 64 bits
- **Initialization Vector**: a public vector *IV* of 22 bits
- From *K* and *IV* we get the initial state (namely a vector of 64 bits) using a **key-loading function** *kl(K,IV)*
- The function *kl* is defined by iteration by using a similar approach to that of the A5/1 cipher
- Warm-up: defined in a way similar to that of A5/1

81

A5/2: UPDATE & OUTPUT



At each clock tick, the functions are defined as follows

- **Update**
  - R_1 is updated if $R_4[6] = maj(R_4[6], R_4[13], R_4[9])$
  - R_2 is updated if $R_4[13] = maj(R_4[6], R_4[13], R_4[9])$
  - R_3 is updated if $R_4[9] = maj(R_4[6], R_4[13], R_4[9])$
  - R_4 is always updated

- **Output**
  - A majority function is applied as non-linear filtering function to each of the first three registers:
    - for R we compute maj(R[3], R[4], R[6]) = a
    - for R we compute maj(R[5], R[8], R[12]) = a
    - for R we compute maj(R[4], R[6], R[9]) = a

$$z = (R_1[0] + R_2[0] + R_3[0]) + a_1 + a_2 + a_3$$

A5/2: CRYPTANALYSIS

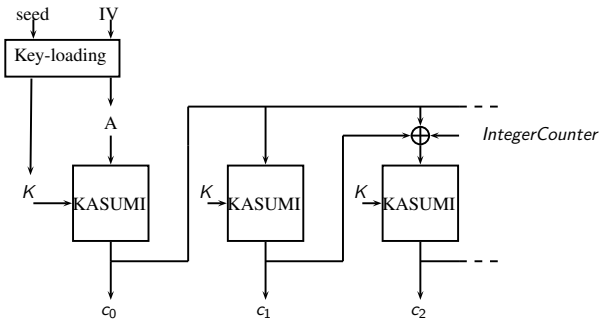
- In 1999, Ian Goldberg and David A. Wagner cryptanalyzed A5/2 in the same month it was published, and showed that it was **extremely weak**
  - Known plaintext attack
  - Low end equipment can probably break it in real time
- Known plaintext attack
  - Difference of two given (plaintext) frames which are roughly 6 seconds apart
  - The average computation cost is about 216 operations of 114-bit vectors
- Since July 1, 2006, the GSM Association mandated that GSM Mobile Phones should not support the A5/2 Cipher any longer

84

A5/3

A5/3: OVERVIEW

- A5/3 is the last stream cipher of the A5 family and provides users with a higher level of security than both A5/1 and A5/2
- Used in GSM, UMTS (f8-UEA1, f9-UIA1) and GPRS (GEA3) communications systems
- It is based on the **block cipher** KASUMI
  - Key space: 128 bits
  - Message space: 64 bits
    - Recall that there is one encryption function per key



A5/3: ARCHITECTURE

- We will see block ciphers in the next lecture

REMARKS ON THE A5 FAMILY (1)

The generated key (*Kc*) is the main flaw of the design:

- Maximal exhaustive search complexity is (only) 2^(64)
- Kc generated only once after the cell phone registers with the network and stays active for all communication, until the telco requests a new one or the cell phone deregisters
- *Kc* is artificially shortened in deployed systems when zeroing 10 bits
- Encryption is applied after error correction
  - Barkan, Biham and Keller proposed a very practical attack requiring only a few encrypted frames from a conversation to discover *Kc*
  - Started with known-plaintext attack based and then converted to a ciphertext-only attack by taking advantage of the error correction codes implemented in GSM data
  - http://www.cs.technion.ac.il/users/wwwb/cgi-bin/tr-get.cgi/2003/CS/CS-2003-05.pdf

REMARKS ON THE A5 FAMILY (2)

- **A5/2** is **extremely weak** and it can be broken in real time with inexpensive equipment; it is therefore **no longer supported** by new mobile phones
- **A5/1** is affected by a **number of serious weaknesses**, and its use is strongly **discouraged**, since there are practical attacks that can break the cipher
- **A5/3** is the **common standard for the new generation of mobile** and it is **considered secure**, even though there do exist practical attacks to KASUMI that suggest some significant weaknesses of the cipher
  - O. Dunkelman and N. Keller and A. Shamir. *A Practical-Time Attack on the A5/3 Cryptosystem Used in Third Generation GSM Telephony.* Cryptology ePrint Archive, Report 2010/013, 2010, http://eprint.iacr.org/

ON THE PRACTICALITY OF ATTACKING GSM COMMUNICATIONS

- In theory, the attacks are relatively simple
- In practice, a considerable amount of hardware is necessary to actually intercept GSM communications
- The hardware must at least consist of a radio receiver device which is capable of receiving and decoding digital data that is exchanged over-the-air by using one of the attacks reported above
- Hypothetically a simple GSM mobile phone already has all these capabilities (except the decrypting of an unknown A5 stream), so it might be possible to use such a phone for eavesdropping
- Nevertheless a huge amount of know-how, time and money is needed

90

EO

BLUETOOTH

- Bluetooth is a wireless technology standard invented by Ericsson in 1994 for exchanging data over short distances using short-wavelength UHF **radio** waves (Range: 2.4 to 2.485 GHz) from fixed and mobile devices
  - https://www.bluetooth.com/
- It allows devices to communicate with each other by splitting data into packets that are exchanged through one of 79 designated channels (each of which have 1 MHz in bandwidth)
- It was originally designed for continuous, streaming data applications, i.e. it is possible to exchange a lot of data at a close range: cellular phones, wireless headsets, printers, cars, and turnstiles
- The standard offers methods for generating keys, authenticating users, and **encrypting** data

BLUETOOTH LOW ENERGY (BLE)

- BLE was introduced in 2011 as Bluetooth 4.0
- Main difference between BLE and Bluetooth: power consumption
- In Machine 2 Machine (M2M) applications such as those supported by the Internet of Things (IoT), the use of BLW allows applications run on a small battery for 4/5 years!
- Although this is not ideal for talking on the phone, it is vital for **applications that only need to exchange small amounts of data periodically**
- BLE operates in the same radio frequency band of Bluetooth but remains in sleep mode constantly except for when a connection is initiated
- The actual connection times are only a few milliseconds
  - Bluetooth would take around 100 milliseconds
  - Data rates are very high at 1 Mb/s

BLE's M2M/IoT Applications

- Blood pressure monitors
- Fibit-like devices
- Industrial monitoring sensors
- Geography-based, targeted promotions (iBeacon)
- Public transportation apps

BLE uses the AES cipher with 128-bit key length to provide data encryption and integrity over the wireless link

E0

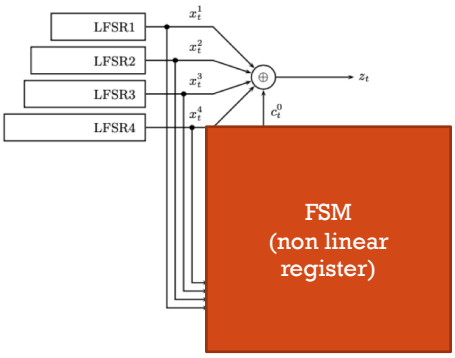
These are all primitive

- year: 1999
- used in Bluetooth standard
- number of LFSRs: 4 Linear registers:
- Linear registers
  - lengths: 25, 31, 33, 39 (total for the linear part: 128 bits)
- number of non-linear registers: 1
- Non-linear registers
  - length of the non-linear register: 4 bit (total: 132 bits)
- update functions: depend on the non-linear register

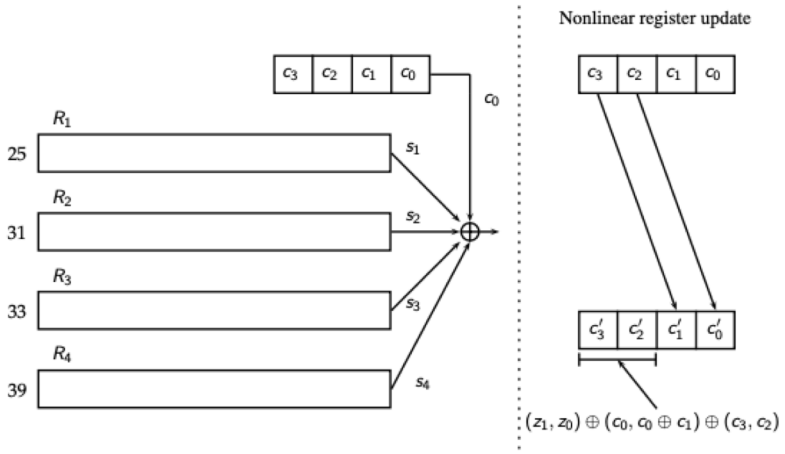
Register	Feedback Polynomial	Taps
R_1	$x^{25} + x^{20} + x^{12} + x^8 + 1$	0, 8, 12, 20
R_2	$x^{31} + x^{24} + x^{16} + x^{12} + 1$	0, 12, 16, 24
R_3	$x^{33} + x^{28} + x^{24} + x^4 + 1$	0, 4, 24, 28
R_4	$x^{39} + x^{36} + x^{28} + x^4 + 1$	0, 4, 28, 36

E0: OVERVIEW

- The key stream generator consists of 4 LFSRs whose outputs are combined by a simple finite state machine
- The LFSRs are all of different length and contain a total of 128 bits
- All 4 feedback polynomials are primitive, the LFSRs will generate maximum-length sequences
- The state of the FSM is determined by 4 bits, which are stored in a pair of 2-bit delay elements
- Encryption is performed by taking the xor of the output sequence and the bit sequence of a data packet
- The maximum size of such a packet is limited to 2745 bits.
  - When the transmission of this packet is completed, the cipher is reinitialised



E0: ARCHITECTURE



SOME PRELIMINARY OBSERVATIONS

- We will not use + for xor
- Instead, we use + as the usual addition over integers and $\oplus$ for the xor operation over bits
- A bit *x* can be also written as an integer, the value of which is *1* or *0*. In the same way, the integers *0* and *1* can be seen as bits
  - Example: $1+1=2$ and $1 \oplus 1=0$
- Given two bits *x* and *y*, we can compute the integer $2x + y$ which will be in the range $[0,3]$
- We write the integers *0, 1, 2,* and *3* as bit vectors of length *2: 00, 01, 10* and *11*

E0: OUTPUT & UPDATE (1)

- Output
  - To obtain a bit of keystream we XOR the output of each linear register and the output of the non-linear register: $x = s_1 \oplus s_2 \oplus s_3 \oplus s_4 \oplus c_0$

    where *s1, s2, s3* and *s4* indicate the output bits of the 4 LFSRs
- Update
  - The update of the 4 LFSRs is regular (without control bits)
  - To describe the update of the non-linear register, we define $z = \left\lfloor \frac{s_1 + s_2 + s_3 + s_4 + 2c_3 + c_2}{2} \right\rfloor$

    noting that $s_1 + s_2 + s_3 + s_4 + 2c_3 + c_2$ is an integer number (recall that + is the sum of integers)
  - Hence $0 \leq z \leq floor(7/2) = 3$ and we consider its binary representation (z_1, z_0)

E0: OUTPUT & UPDATE (2)

- The update of the non-linear register is given by

$$(c_3, c_2, c_1, c_0) \longmapsto (c_3', c_2', c_1', c_0')$$

where

  - $c_1' = c_3$ and $c_0' = c_2$, i.e. c_1 and c_2 are shifted of two places to the right
  - $(c_3', c_2') = (z_1, z_0) \oplus (c_0, c_0 \oplus c_1) \oplus (c_3, c_2)$.

Example

Let us consider the state of the non-linear register

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

and let $(s_1, s_2, s_3, s_4) = (1, 1, 0, 0)$ *be the output of the four linear registers.*

To update the non-linear register we first compute z*:*

$$z = \left\lfloor \frac{s_1 + s_2 + s_3 + s_4 + 2c_3 + c_2}{2} \right\rfloor = \left\lfloor \frac{1 + 1 + 0 + 0 + 2 \cdot 1 + 0}{2} \right\rfloor = 2$$

Hence the binary expression of z *is* $(z_1, z_0) = (1, 0)$.

Example (continued)

- $(c_3, c_2, c_1, c_0) = (1, 0, 1, 1)$
- $(z_1, z_0) = (1, 0)$

We recall how the non-linear register is updated:

- $c'_1 = c_3$ *and* $c'_0 = c_2$
- $(c'_3, c'_2) = (z_1, z_0) \oplus (c_0, c_0 \oplus c_1) \oplus (c_3, c_2)$.

Hence

$$(c'_3, c'_2) = (1, 0) \oplus (1, 1 \oplus 1) \oplus (1, 0) = (1, 0)$$

and therefore the updated non-linear register is

$$(c'_3, c'_2, c'_1, c'_0) = (1, 0, 1, 0)$$

REMARK ON NON-LINEARITY

- Note that the non-linear part is introduced in the definition of z, because

  summing vectors of GF(2^n) as integers is a non-linear operation on the vector space GF(2^n)

- This is a good technique to construct stream ciphers, since it allows to reach high non-linearity using easy and fast computations

- Why is this so*?*

LINEAR VS NON-LINEAR (1)

- Linearity is defined as maps between vector spaces
- Given two vector spaces U and V over a field F, a map $T:U \to V$ is linear if

$$T(\gamma 1 \odot u1 \oplus \gamma 2 \odot u2) = \gamma 1 \odot T(u1) \oplus \gamma 2 \odot T(u2)$$

where $\gamma 1, \gamma 2 \in F$ $\gamma 1, \gamma 2 \in$ F and $u1, u2 \in V$ u1, u2 ∈ V
  - **Notation**: $\oplus$ and $\odot$ denote the addition of vectors and their multiplication by a scalar (element of the field), respectively
- Take U to be the set of all 8-bit integers, i.e., the integers between 0 and 255
- Each such integer can be expressed as a string of exactly 8 bits
  - **Example**: 13 can be expressed as 0·2^7+0·2^6+0·2^5+0·2^4+1·2^3+1·2^2+0·2^1+1·2^0=8+4+1=13
- A way of looking at U is considering its elements to be vectors of {0,1}^8
  - Example: the number 13 becomes the vector (0,0,0,0,1,1,0,1)

LINEAR VS NON-LINEAR (2)

- The natural way of defining the sum of two vectors in this case is addition modulo 2, component by component, or equivalently bitwise xor
- This results in the sum being the xor of both addends
- When $F=\{0,1\}$, U becomes a vector space over F
- **Notice that $\oplus$ is xor or addition modulo 2, not addition modulo *n* for some *n>2***
- Vice versa, addition modulo $n>2$ can be described by a map over the field $\{0, \ldots, n-1\}$ but not a map over the finite field $\{0,1\}$
- In conclusion, if we consider bit vectors of length $n>2$ and manipulate them by performing operations modulo n, this operation cannot be represented as a linear map on the vector space (of dimension n) over $F=\{0,1\}$

ATTACKS ON E0 (1)

- Known plaintext attacks
  - An attacker is able to obtain a certain amount of decrypted text in one way or another
  - Goal: use availabe information to recover other (unknown) parts of the plaintext
- In the case of E0 (that is an additive stream cipher), the attack amounts to finding a way to predict the entire key stream given a limited number of key stream bits
- Two possible approaches
  - **Correlation** attacks
  - **Guess and determine** attacks

ATTACKS ON E0 (2)

- **Correlation**
  - Recover parts of the initial state of the generator by exploiting specific correlation properties of the FSM
  - Cannot be applied to the E0 algorithm, as it assumes sequences of consecutive key stream bits which are considerably longer than the maximum packet size
- **Guess and determine**
  - some parts of the initial state are guessed first and the observed key sequence is used to derive the remaining parts in a deterministic way afterwards
  - Example
    - The contents of a single LFSR can directly be derived from the contents of the three other LFSRs, the 4 bits of the FSM, and some known key stream
    - By guessing the initial state of the FSM and the contents of the three shortest LFSRs (4 + 25 + 31 + 33 = 93 bits), the attack succeeds in recovering the entire internal state in 292 steps

106

RC4

RC4: HISTORY

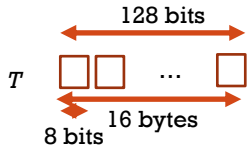
- It was designed by Ron Rivest of RSA Security in 1987
  - RC4 = Rivest Cipher 4
- Initially a trade secret, but in September 1994 a description of it was leaked
  - The name RC4 is trademarked
  - RSA Security has never officially released the algorithm
  - Rivest has confirmed the history of RC4 and its code in 2014
- RC4 became part of some commonly used encryption protocols and standard
  - WEP in 1997 and WPA in 2003/2004 for wireless cards
  - SSL in 1995 and its successor TLS in 1999
- The main factors in RC4's success over such a wide range of applications have been its **speed and simplicity**: efficient implementations were easy to develop

107

RC4: OVERVIEW

- RC4 generates a pseudorandom stream of bytes, the key stream
- To generate the keystream, the cipher makes use of a secret internal state which consists of two parts:
  - A permutation of all 256 possible bytes (denoted *S*)
  - Two 8-bit index-pointers (denoted *i* and *j*)
- The permutation is initialized with a **variable length key, typically between 40 and 2048 bits**, using the so-called **key-scheduling algorithm** (**KSA**)
- Once this has been completed, the stream of bits is generated using the pseudo-random generation algorithm (PRGA)
- While many stream ciphers are based on LFSRs (efficient in hardware but less so in software), the **design of RC4 avoids the use of LFSRs** and is **ideal for software implementation, as it requires only byte manipulations**

RC4: SETUP



- Fundamental to the RC4 algorithm is a 256 element array of 8-bit integers
  - It is called the state vector and denoted *S*
- The state vector is initialized with the encryption key
  - The initialization steps are as follows:
  1. *S* is initialized with entries from 0 to 255 in the ascending order
  2. *S* is further initialized with the help of a temporary 256-element vector denoted *T* that also holds 256 integers. The vector *T* is initialized with the encryption key as follows:
     a) Let *K* be the encryption key represented as a vector 8-bit integers of size 16 (in case of a 128-bit key), i.e. *K* stores 16 non-negative integers whose values will be between 0 and 255
     b) Initialize the 256-element vector *T* by placing in it as many repetitions of the key as necessary until T is full
  3. Use the 256-element vector *T* to produce the initial **permutation** of *S* as follows:

```
j  =  0
for i = 0 to 255
    j  =  ( j + S[i] + T[i] )  mod  256
    SWAP  S[i], S[j]
```

Key Scheduling Algorithm (KSA)

GENERATING THE PSEUDORANDOM BYTE STREAM

- **Idea**: as each byte of the plaintext becomes available, it is XORed with a byte of the pseudorandom byte stream
- **Procedure**

```
i, j  =  0
while ( true )
    i  =  ( i + 1 )  mod  256
    j  =  ( j + S[i] )  mod  256
    SWAP S[i], S[j]
    k  =  ( S[i]  +  S[j] )  mod  256
    output S[k]
```

- Note how the state vector S changes continuously because of swapping at each pass through the body of the loop
- The **state** of the generator **changes dynamically** as the the numbers are being generated

RC4: REMARKS

- Theoretical analysis shows that for a **128 bit key length**, the **period** of the pseudorandom sequence of bytes is likely to be **greater than 10^(100)**
- As all operations are at the byte level, the cipher possesses **fast software implementation**
  - For this reason, RC4 was the software stream cipher of choice for several years
  - More recently though, **RC4 was shown to be vulnerable to attacks** especially if the beginning portion of the output pseudorandom byte stream is not discarded
  - As a consequence, the use of RC4 in the SSL/TLS protocol is now prohibited (since 2015)
- WiFi security started with RC4 in the WEP protocol
- After it was discovered that the encryption key used in WEP could be acquired by an adversary in almost no time, WiFi security has now moved on to the WPA2 protocol that uses AES for encryption

RC4: SOME SECURITY CONSIDERATIONS

- Unlike modern stream ciphers, **RC4 does not take a separate nonce alongside the key**
  - If a single long-term key is to be used to securely encrypt multiple streams, the protocol must specify how to combine the nonce and the long-term key to generate the stream key for RC4
  - One approach to addressing this is to generate a "fresh" RC4 key by hashing a long-term key with a nonce
  - Unfortunately, many applications that use RC4 simply concatenate key and nonce; this gives rise to related key attacks that are famous for breaking the WEP standard
- Because RC4 is a stream cipher, it is **more malleable** than block ciphers
  - If not used together with other techniques (e.g., message authentication codes), then encryption is vulnerable to bit-flipping attacks

FINAL REMARKS ON STREAM CIPHERS

- Stream ciphers require a shared secret, a **seed**, which should be transmitted via a **secure channel**
  - Once Alice and Bob have the seed, they can exchange via an insecure channel other paramaters, like the IV and start encrypting/decrypting
- The use of **stream ciphers guarantees confidentiality**
  - Eve will not be able to understand what Alice and Bob are telling each other
- However, the **stream cipher** in itself **does not guarantee that the seed has been exchanged securely**, this has to be done in other ways
- On the other hand, the attacker might tamper with the ciphertext and Alice/Bob will not understand immediately that the messages have been corrupted
  - In other words, **stream ciphers provide neither authentication nor integrity**