Cryptography Lecture 3

Stream ciphers, random number generators

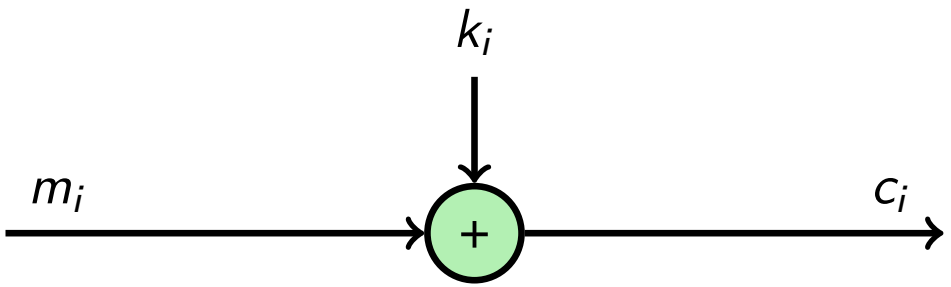
LINKÖPING UNIVERSITY

The One Time Pad is the only theoretically secure cipher

- **Theorem:** The one time pad has perfect secrecy
- You must generate a truly random key sequence equally long as the message, and then find a secure channel for transportation of that key to the intended message recipient
- Your problem is now instead to transfer large quantities of secure key

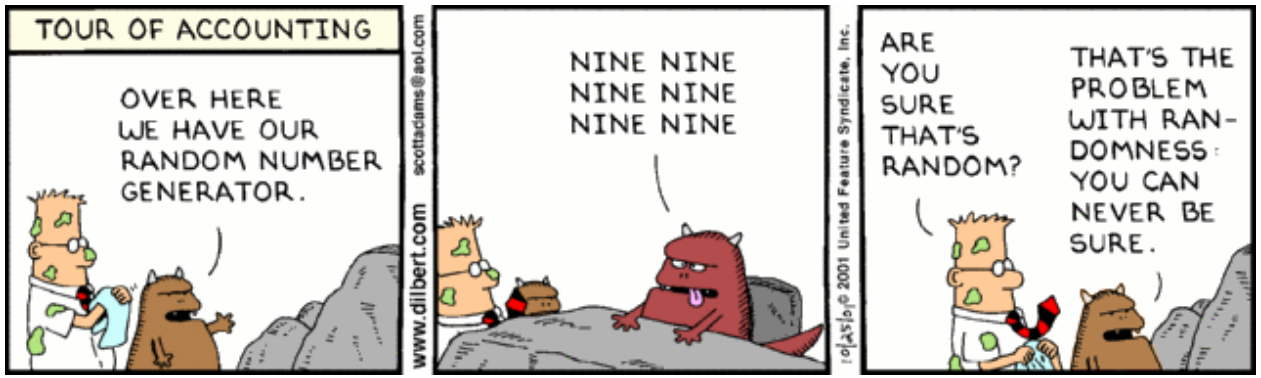
One-time-pad

- Useful for sending secret data without preserved data integrity
- Uses a long random bitsequence as key
- Adds this to the plaintext to form the next cryptogram
- No error propagation

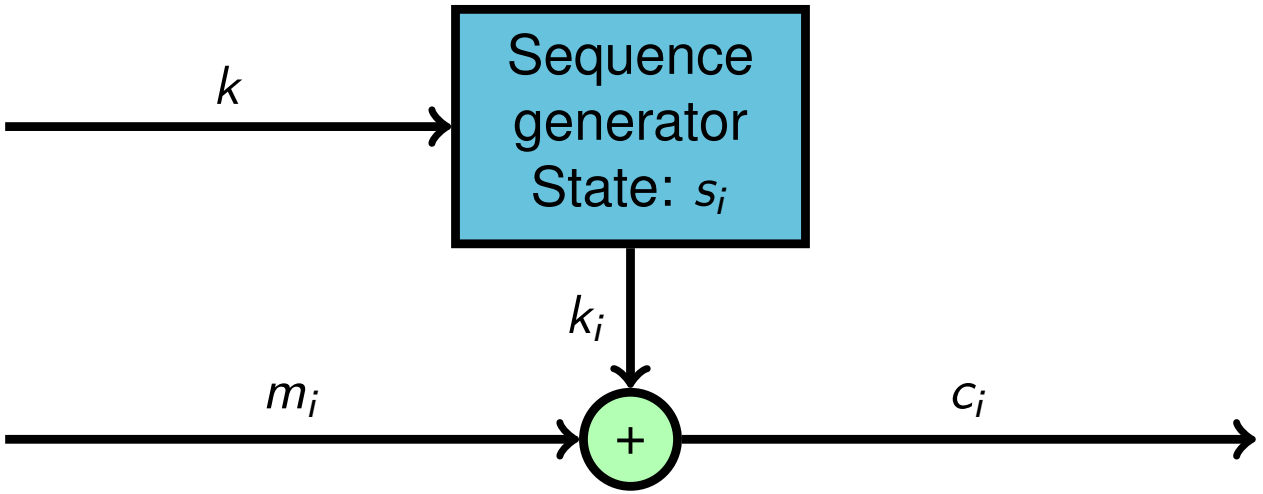


A more practical alternative is a stream cipher

- We generate a pseudorandom “key stream” from a seed, a “real key” much shorter than the full “key stream” added to the message
- We try to make the set of possible seeds, the real keys, so large that exhaustive search is impossible in practice
- We try to eliminate any shortcuts to finding this key from the “key stream”



Additive stream ciphers



$$s_i = f(k, s_{i-1})$$
$$k_i = g(s_i)$$
$$c_i = m_i \oplus k_i$$

The key, k, can be a parameter in f and/or the initial state s_0

Pseudo-Random Number Generator (PRNG)

- Sender and receiver must generate exactly the same key stream from the seed, so the generator is deterministic
- Real-world generators must be finite, having a fixed total number of states
- Sooner or later the generator is in a state it has been in before
- From then on, the internal states and the output repeats, exactly as the previous time the state was used
- Therefore, the output will be periodic, directly or after some initial output part

Stream ciphers are sometimes viewed as weak

- This is not generally true, they are useful and secure
  - Benefits: fast, no fixed block size, errors do not propagate
  - Drawbacks: No message integrity check, the devil is in the details
- They do need to be used with care
- If you don’t follow the algorithm to the letter, things can go very bad (we’ll see a few examples)

Stream ciphers and message integrity

- Imagine you are able to intercept the bank transfer that puts your salary into your bank account, and that it is encrypted with a stream cipher, and not authenticated
- You know what your salary is, and in what position it is on the bank transfer:

```
... CR LF SPC SPC SPC SPC SPC SPC  3  0  0  0  0 CR LF ...
... 13 10  32  32  32  32  32  32 51 48 48 48 48 13 10 ...
```

Stream ciphers and message integrity

- Imagine you are able to intercept the bank transfer that puts your salary into your bank account, and that it is encrypted with a stream cipher, and not authenticated
- You know what your salary is, and in what position it is on the bank transfer:

```
... CR LF SPC SPC SPC SPC SPC SPC  3  0  0  0  0 CR LF ...
... 13 10  32  32  32  32  32  32 51 48 48 48 48 13 10 ...
```

- You intercept the ciphertext

```
... 74 112 4 19 235 156 99 113 125 19 1 192 191 25 212 ...
```

Stream ciphers and message integrity

- Imagine you are able to intercept the bank transfer that puts your salary into your bank account, and that it is encrypted with a stream cipher, and not authenticated
- You know what your salary is, and in what position it is on the bank transfer:

```
... CR LF SPC SPC SPC SPC SPC SPC  3  0  0  0  0 CR LF ...
... 13 10  32  32  32  32  32  32 51 48 48 48 48 13 10 ...
```

- You intercept the ciphertext

```
... 74 112 4 19 235 156 99 113 125 19 1 192 191 25 212 ...
```

- The `19` in the ciphertext is at the same position as the first `48` in the cleartext

Stream ciphers and message integrity

- Imagine you are able to intercept the bank transfer that puts your salary into your bank account, and that it is encrypted with a stream cipher, and not authenticated
- You know what your salary is, and in what position it is on the bank transfer:

```
... CR LF SPC SPC SPC SPC SPC SPC 3  3  0  0  0 CR LF ...
... 13 10  32  32  32  32  32  32 51 51 48 48 48 13 10 ...
```

- You intercept the ciphertext

```
... 74 112 4 19 235 156 99 113 125 16 1 192 191 25 212 ...
```

- Changing `19` to `16` in the ciphertext changes the first `48` to `51` in the received cleartext. Noone will notice :)

Key stream generators

- You cannot use a true random sequence of sufficient length ($\sim$OTP), since this would require a(nother) secure channel
- Alternative: use a pseudorandom sequence, created through a known (deterministic) procedure that is “seeded” with a shorter key
- Since the generator is deterministic and finite, the sequence is periodic
- It is not enough that the output “seems” random. If the seed (key) is unknown, every output bit (byte, . . .) should be unpredictable given the already generated sequence

Cryptographic security of key stream generators

- A known plaintext attack will make part of the sequence known to the cryptanalyst
- It is not enough that the output “seems” random. If the seed (key) is unknown, every output bit (byte, . . .) should be unpredictable given the already generated sequence
- The sequence is periodic. A known plaintext attack then needs the period to be as long as possible, since the known part repeats after the period length

Linear congruential generators (pseudo-random, but bad for crypto)

$$X_n = aX_{n-1} + b \mod m$$

- The seed (key) is X_0
- Good properties:
  - The maximal sequence length is $m - 1$
  - Passes many tests for (pseudo-)randomness
- Bad properties
  - Predictable, as shown in the late 70's
  - Even polynomial congruential generators were broken in the early 80's

Linear feedback shift register

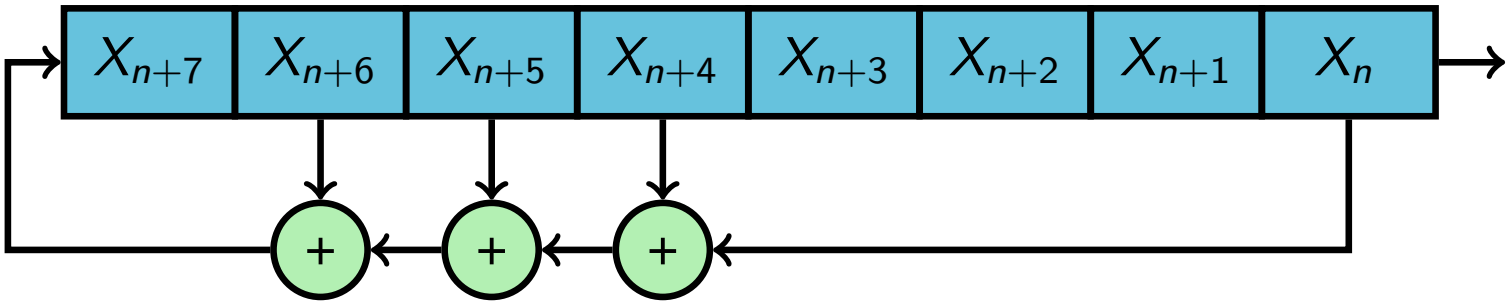
$$X_{n+m} = c_0 X_n + c_1 X_{n+1} + \ldots + c_{m-1} X_{n+m-1}$$

- Has been the workhorse for military encryption for as long as electronics (electromechanics) has existed
- Fast in hardware, but slow in software
- Except in CRAYs, of course, where the “population count” instruction exists

Linear feedback shift register

$$X_{n+m} = c_0 X_n + c_1 X_{n+1} + \ldots + c_{m-1} X_{n+m-1}$$

Example:



Period length: $2^8 - 1$

Linear feedback shift register

$$X_{n+m} = c_0 X_n + c_1 X_{n+1} + \ldots + c_{m-1} X_{n+m-1}$$

- The key is $c_0, \ldots, c_{m-1}$ and the starting sequence $X_1, \ldots, X_m$
- Good properties:
  - The maximal sequence length is $2^m - 1$
  - Passes many tests for (pseudo-)randomness
- Bad properties
  - Direct use of one LFSR enables a known plaintext attack, where a sequence of length $2m$ recovers the key (Berlekamp-Massey)

The Berlekamp-Massey algorithm

- Finds the shortest LFSR that gives the known sequence
- Needs only $2m$ bits to do this for a length-m LFSR
- Looks complicated, but isn't really
- The course book uses a simple, but similar technique

How to find an LFSR that outputs a given sequence

$$X_{n+m} = c_0 X_n + c_1 X_{n+1} + \ldots + c_{m-1} X_{n+m-1}$$

$$01101011110001001101011 1 \ldots$$

Try $m = 2$

$$\begin{cases} c_0 X_1 + c_1 X_2 = X_3 \\ c_0 X_2 + c_1 X_3 = X_4 \end{cases}$$

How to find an LFSR that outputs a given sequence

$$X_{n+m} = c_0 X_n + c_1 X_{n+1} + \ldots + c_{m-1} X_{n+m-1}$$

$$01101011110001001101011 1 \ldots$$

Try $m = 2$

$$\begin{cases} c_0 0 + c_1 1 = 1 \\ c_0 1 + c_1 1 = 0 \end{cases}$$

How to find an LFSR that outputs a given sequence

$$X_{n+m} = c_0 X_n + c_1 X_{n+1} + \ldots + c_{m-1} X_{n+m-1}$$

$$01101011110001001101011 1 \ldots$$

Try $m = 2$

$$\begin{cases} c_0 0 + c_1 1 = 1 \\ c_0 1 + c_1 1 = 0 \end{cases}$$

$$c_0 = 1,\ c_1 = 1$$

$$01101101101101101 1 \ldots$$

How to find an LFSR that outputs a given sequence

$$X_{n+m} = c_0 X_n + c_1 X_{n+1} + \ldots + c_{m-1} X_{n+m-1}$$

$$01101011110001001101011 1 \ldots$$

Try $m = 2$

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$c_0 = 1, \; c_1 = 1$$

$$01101101101101101 1 \ldots$$

How to find an LFSR that outputs a given sequence

$$X_{n+m} = c_0 X_n + c_1 X_{n+1} + \ldots + c_{m-1} X_{n+m-1}$$

$$01101011110001001101011 1 \ldots$$

Try $m = 3$

$$\begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

(det 0 = linearly dependent rows = no solution or more than one = linearly dependent columns)

How to find an LFSR that outputs a given sequence

$$X_{n+m} = c_0 X_n + c_1 X_{n+1} + \ldots + c_{m-1} X_{n+m-1}$$

$$01101011110001001101011 1 \ldots$$

Try $m = 4$

$$\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

How to find an LFSR that outputs a given sequence

$$X_{n+m} = c_0 X_n + c_1 X_{n+1} + \ldots + c_{m-1} X_{n+m-1}$$

$$01101011110001001101011 1 \ldots$$

Try $m = 4$

$2m = 8$

$$\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

$$c_0 = 1,\ c_1 = 1,\ c_2 = 0,\ c_3 = 0$$

How to find an LFSR that outputs a given sequence

$$X_{n+m} = c_0 X_n + c_1 X_{n+1} + \ldots + c_{m-1} X_{n+m-1}$$

$$01101011110001001101011 1 \ldots$$

For $m > 4$

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

det 0: columns are linearly dependent

How to find an LFSR that outputs a given sequence

$$X_{n+m} = c_0 X_n + c_1 X_{n+1} + \ldots + c_{m-1} X_{n+m-1}$$

$$01101011110001001101011 1 \ldots$$

Determinants:

$$0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, \ldots$$

Theorem: The length of the shortest possible LFSR tells you which is the last nonzero determinant

How to find an LFSR that outputs a given sequence

Theorem: The length of the shortest possible LFSR tells you which is the last nonzero determinant

Proof: Denote the shortest possible length m (assuming it is finite). Write the sequence as a diagonal-constant (Toeplitz) matrix n-by-n matrix

$$A = \begin{bmatrix} X_1 & X_2 & \dots & X_n \\ X_2 & X_3 & & \vdots \\ \vdots & & \ddots & \vdots \\ X_n & \dots & \dots & X_{2n-1} \end{bmatrix}$$

If $n > m$, the last row will depend linearly on the others (an LFSR of length m) and the determinant will be 0.

How to find an LFSR that outputs a given sequence

Theorem: The length of the shortest possible LFSR tells you which is the last nonzero determinant

Proof: (continued)

$$A = \begin{bmatrix} X_1 & X_2 & \dots & X_n \\ X_2 & X_3 & & \vdots \\ \vdots & & \ddots & \vdots \\ X_n & \dots & \dots & X_{2n-1} \end{bmatrix}$$

Assume that the determinant is zero when $n = m$. Then, some rows are linearly dependent; let k be the index of the first row in this linear dependence. When X_{m+k} is calculated, X_k can be eliminated because of the linear dependence. This gives a new, shorter LFSR from then on, so that m is not the minimum length (a zero determinant gives a contradiction). □

Mathematical representation

$$X_{n+m} = c_0 X_n + c_1 X_{n+1} + \ldots + c_{m-1} X_{n+m-1}$$

Taking the z-transform gives you the following equation:

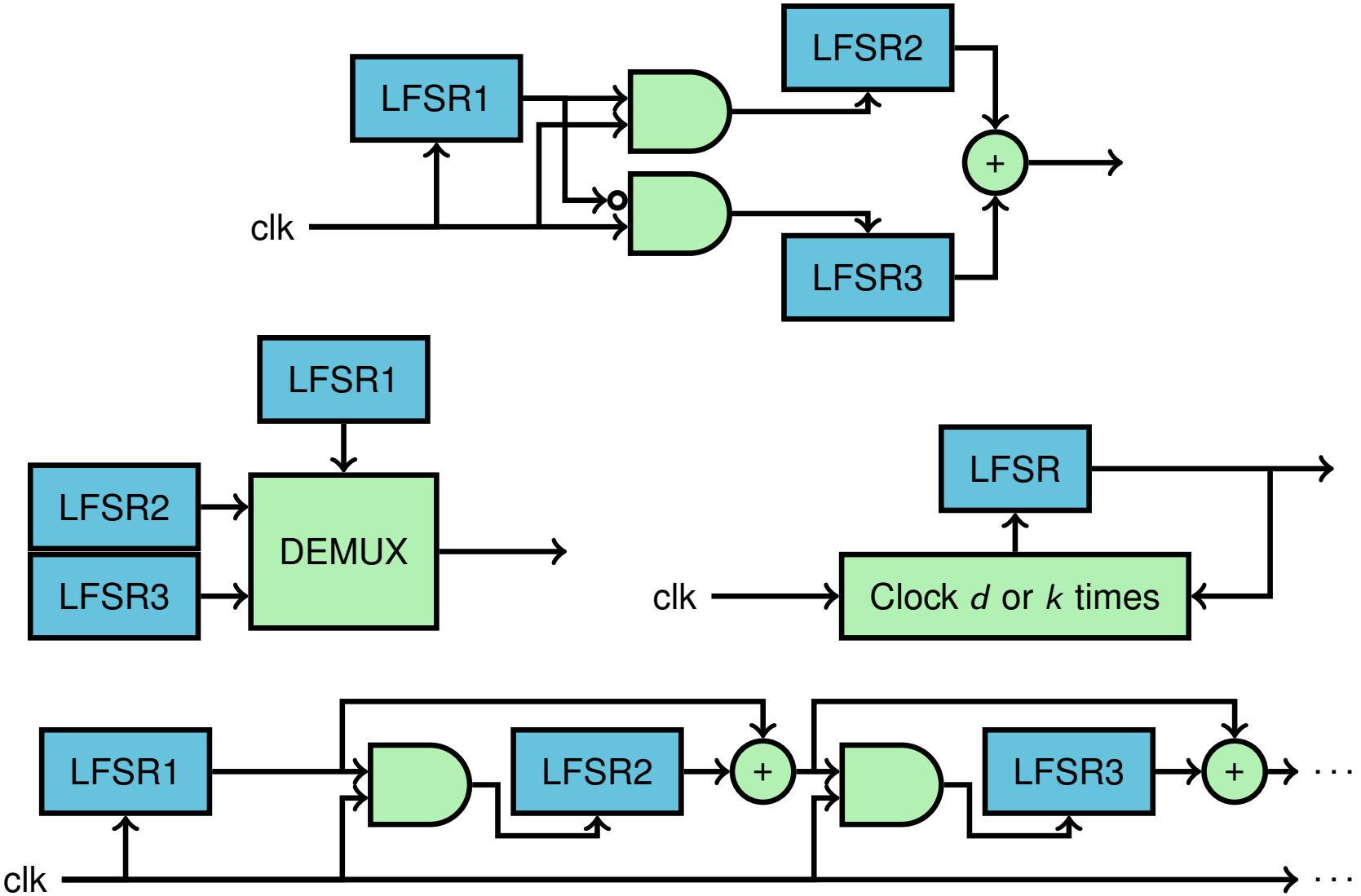
$$z^m X(z) = c_0 z^0 X(z) + c_1 z^1 X(z) + \ldots + c_{m-1} z^{m-1} X(z) + g(c_0, \ldots, c_{m-1}, X_0, \ldots, X_{m-1})$$

Solving for $X(z)$ gives

$$X(z) = \frac{g(c_0, \ldots, c_{m-1}, X_0, \ldots, X_{m-1})}{c_0 z^0 + c_1 z^1 + \ldots + c_{m-1} z^{m-1} + z^m}$$

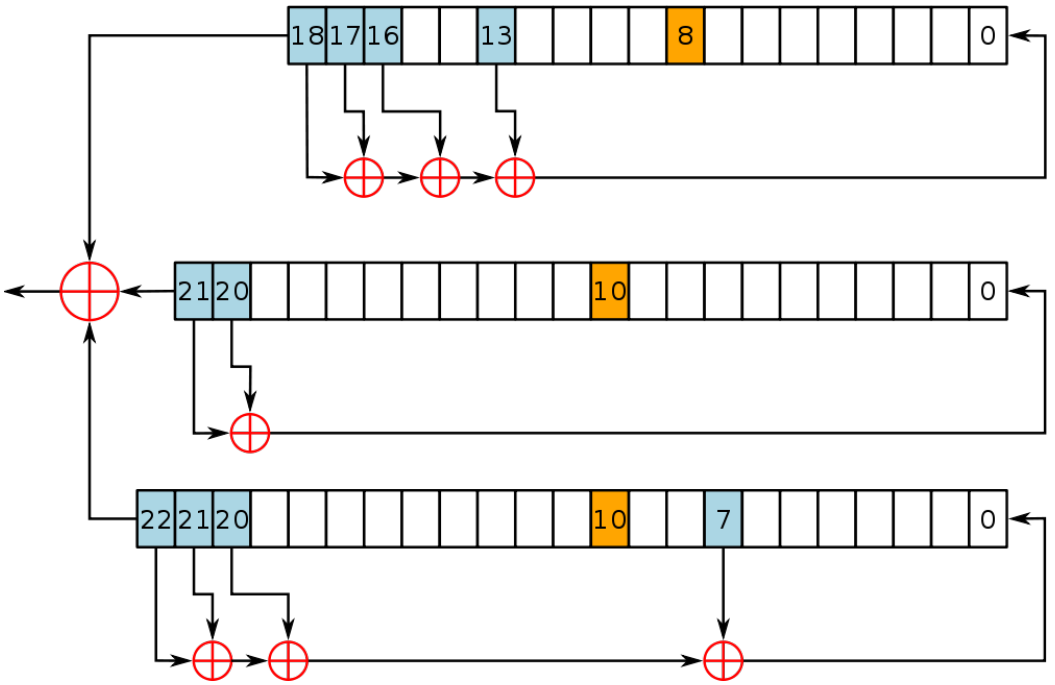
Theorem: To give the longest possible period $2^m - 1$, the polynomial in the denominator needs to be primitive (see the course book)

Using LFSRs as a basis for stream ciphers



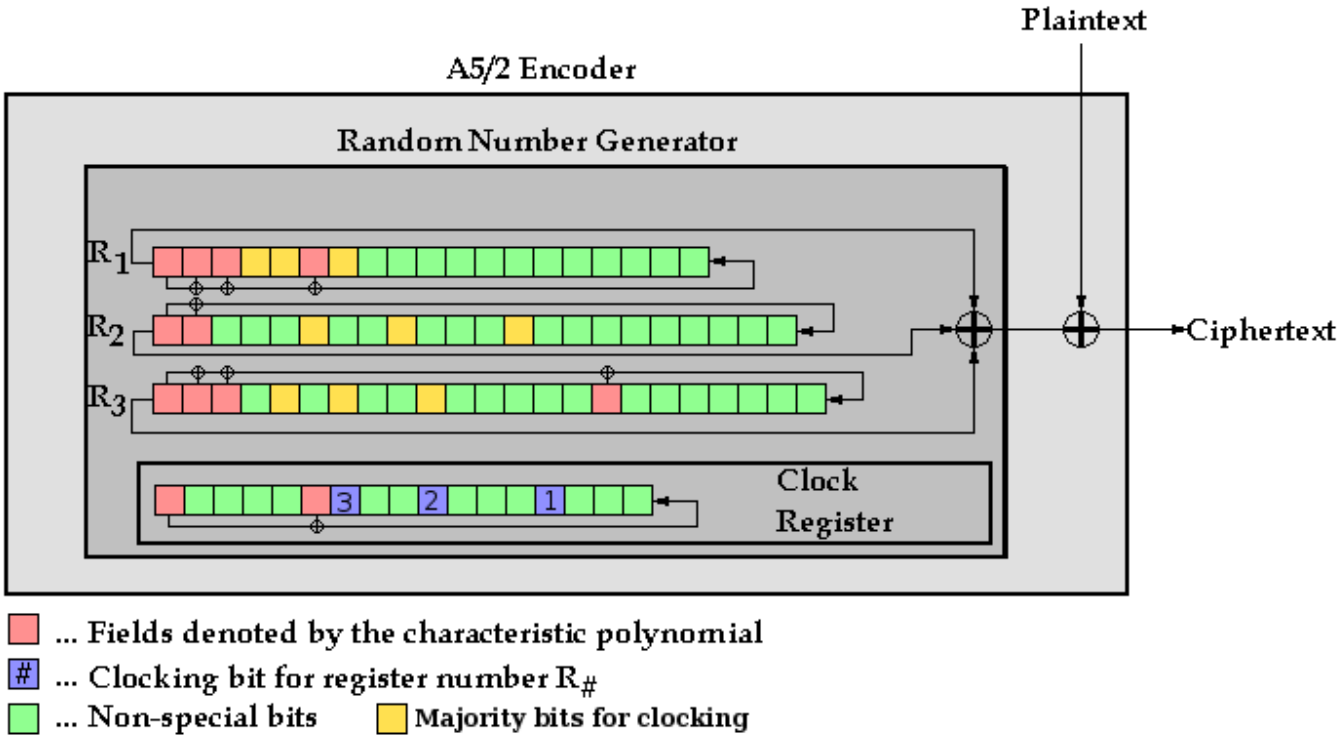
A5/1 (GSM)

- GSM A5/1 uses three LFSRs, only clocking those that agree in the clock bit (in orange)
- Only initial state is key, and sparse and rather short polynomials
- Broken:
  "A5/1 cracking project"
  (Dec 2009)

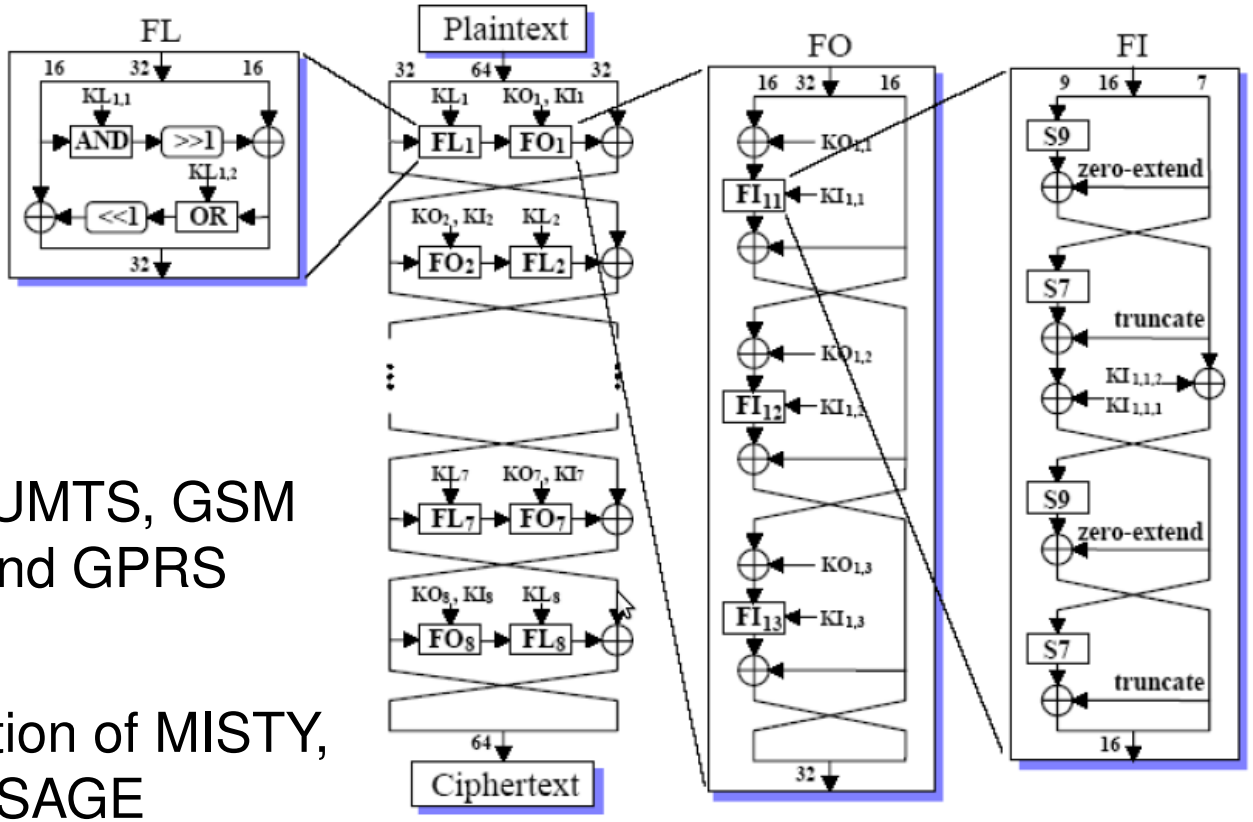


A5/2 (Weaker GSM)

- GSM A5/2 uses four LFSRs, only clock output LFSRs where the majority orange bits agree with the clock LFSR
- Broken the same month 1999 it was published
- Prohibited from implementation in mobiles since 2007



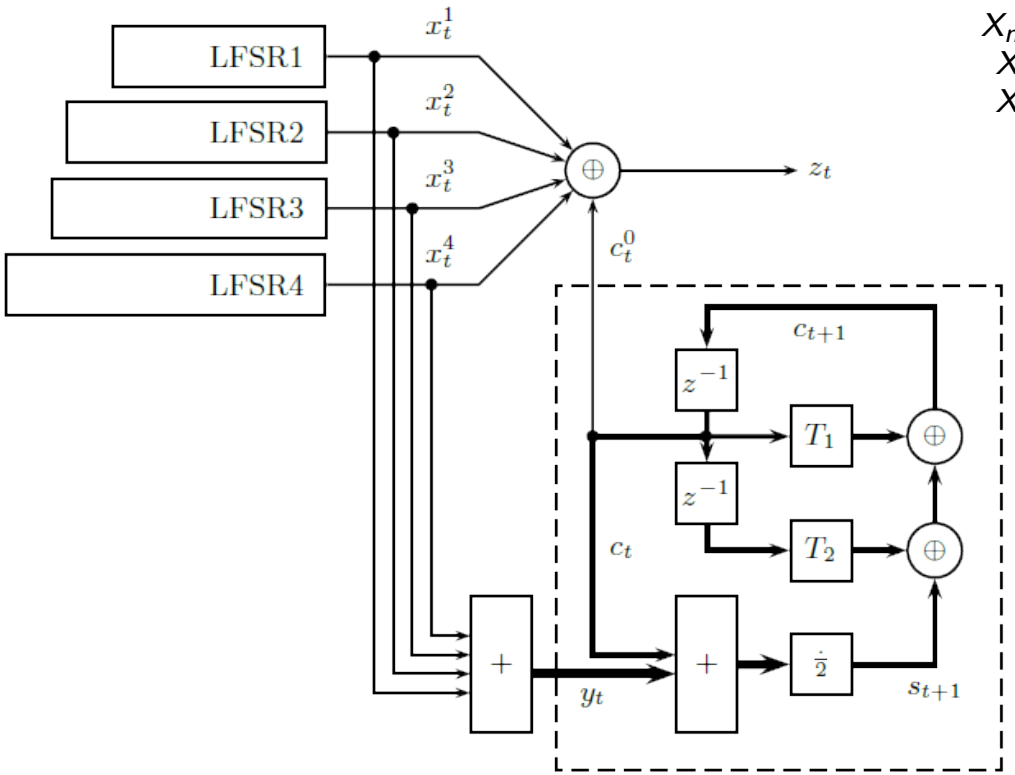
KASUMI



- Used in UMTS, GSM (A5/3), and GPRS (GEA3)
- Modification of MISTY, done by SAGE
- A related key attack breaks Kasumi with very modest computational resources
- Attack does not work on the designs used in UMTS, GSM, and GPRS

E0 (Bluetooth)

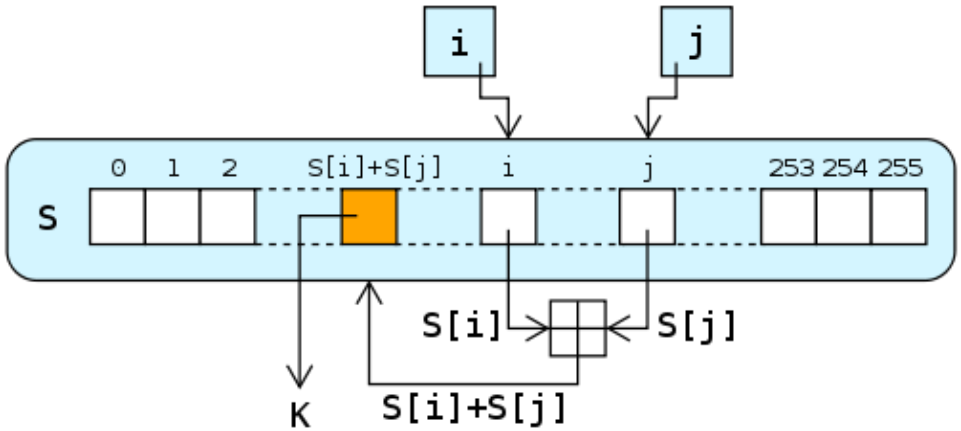
- Bluetooth E0 uses four LFSRs and two 2-bit internal states
- Known plain-text attacks exist of complexity 2^{38}



$$X_{n+26} = X_n + X_{n+8} + X_{n+12} + X_{n+20} + X_{n+25}$$
$$X_{n+32} = X_n + X_{n+12} + X_{n+16} + X_{n+24} + X_{n+31}$$
$$X_{n+34} = X_n + X_{n+4} + X_{n+24} + X_{n+28} + X_{n+33}$$
$$X_{n+40} = X_n + X_{n+4} + X_{n+28} + X_{n+36} + X_{n+39}$$

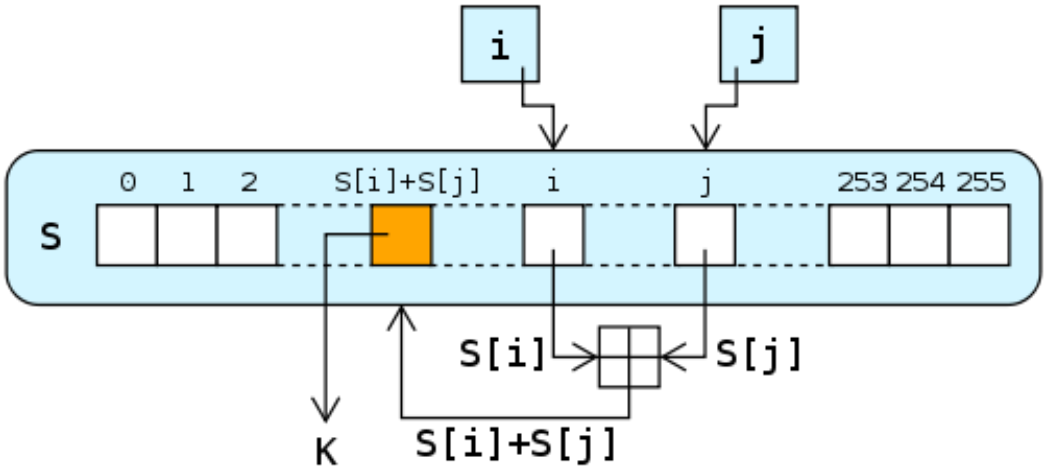
RC4 (ARCFOUR)

- Byte-based stream cipher with $\sim 2^{1700}$ internal states
- Init $S[k] = k$, for i in 0...255:$j+ = S[i] + k[i]$, $S[i] \leftrightarrow S[j]$ Key stream: $i++, j+ = S[i], S[i] \leftrightarrow S[j], K = S[S[i] + S[j]]$
- Officially known, but “Trade secret” (RSA labs)
- Considered broken since 2015, but previously used in
  - WEP, WPA (alt.: AES-CCMP), BitTorrent, MS P-t-P, Opera Mini, RDP, PDF, Skype (modified)
  - Optional in SSL, SSH, Kerberos, ...



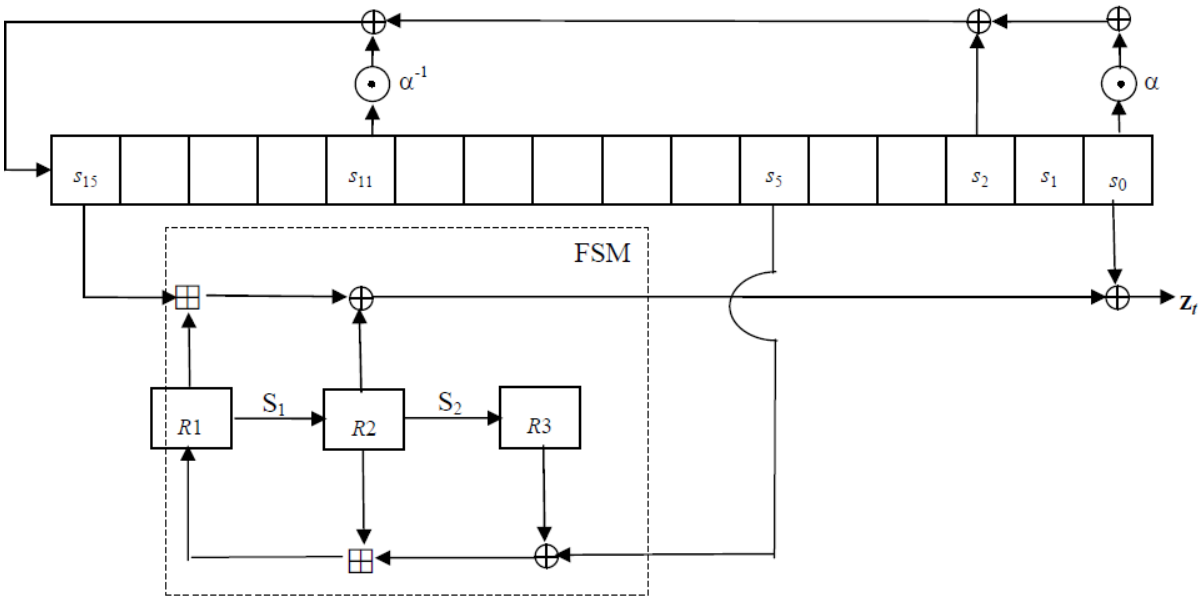
RC4 (ARCFOUR) weaknesses

- WEP uses a 40/104-bit key concatenated with a 24-bit IV (nonce), this is not large enough to avoid IV repetition
- RC4 is also vulnerable to related-key attacks, like when using non-random keys or badly designed key management, like in WEP
- The start of the key stream is correlated with the key and should be dropped, recommendations range between 512 and 3072 bytes



SNOW 3G

- Uses one LFSR and another more complicated Finite State Machine
- Registers are 32 bit
- Combines mod 2 and mod 2^{32} arithmetic
- Fast, secure (as of now), part of 3GPP standard



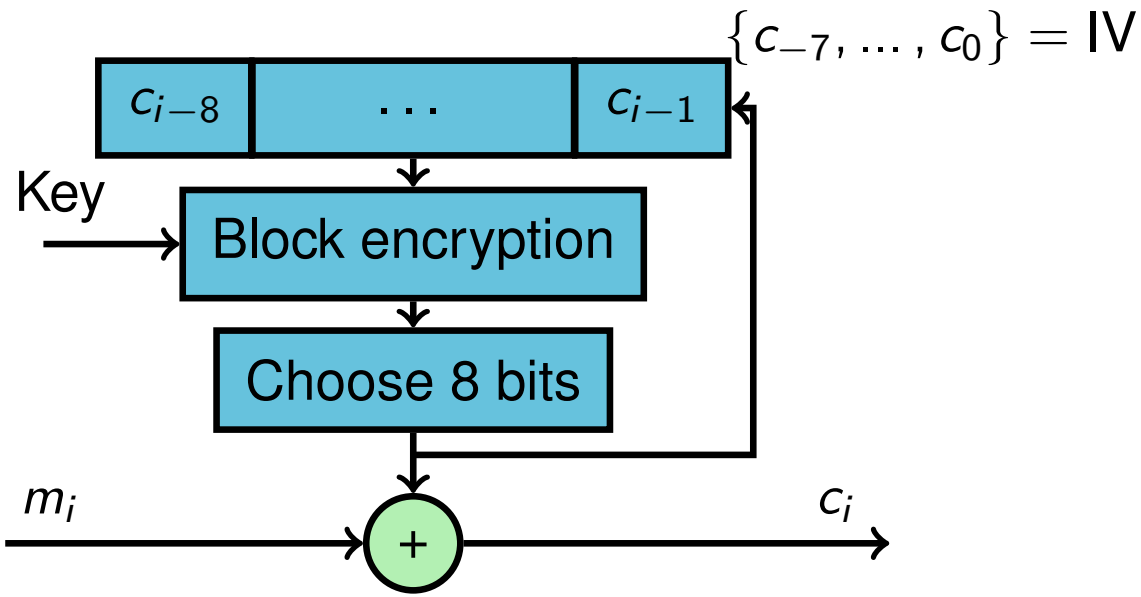
Many alternatives, many proposals

- Nonlinear Feedback Shift Register
  - used in RFID and smartcard applications
  - known to be more resistant to cryptanalytic attacks than LFSRs
  - construction of large NLFSRs with guaranteed long periods remains an open problem (done with brute force for $n \leq 25$ and $n = 27$)
- Feedback with Carry Shift Register
- Cipher feedback into registers (WAKE)
- ...

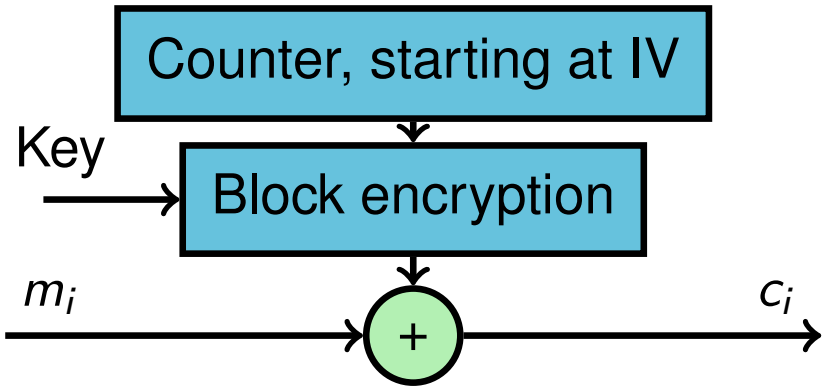
Problem: less theory available, so it is difficult to evaluate proposals

Stream ciphers from block ciphers

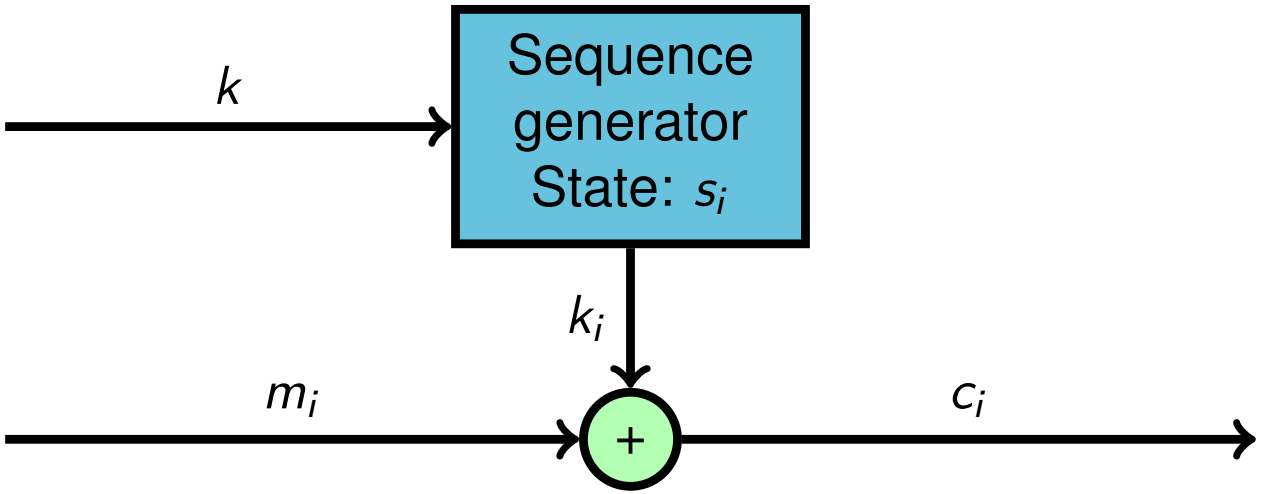
OFB



CTR



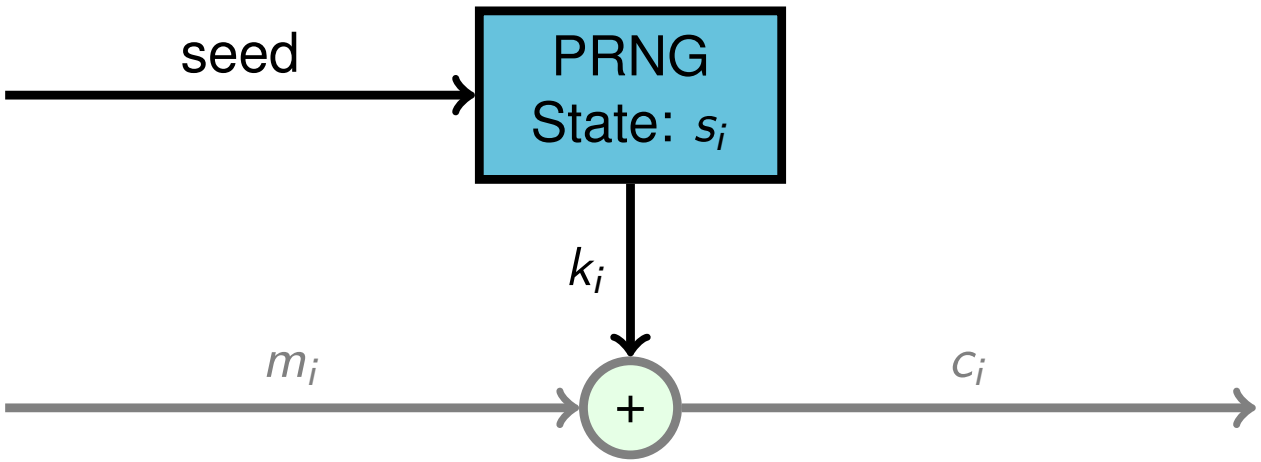
Additive stream ciphers



$$s_i = f(k, s_{i-1})$$
$$k_i = g(s_i)$$
$$c_i = m_i \oplus k_i$$

The key, k, can be a parameter in f and/or the initial state s_0

Random Number Generators



$$s_i = f(\text{seed}, s_{i-1})$$
$$k_i = g(s_i)$$

The seed can be a parameter in f and/or the initial state s_0
There are many other (cryptographic) uses for randomness

Random Number Generators

Random numbers are often used in cryptography, but also in games, scientific calculations (“Monte-Carlo” methods), . . .
Some examples:

- The Mersenne twister (Matlab, mathematica, . . .)
- The digits of π
- Blum-Blum-Shub (secure but slow)
- /dev/random for Linux (Ts’o, 1995)
- Yarrow for MacOs/iOS/FreeBSD (Apple Inc., 1999)
- Fortuna for Windows (Ferguson and Schneier, 2003)
- AES in OFB or CTR mode

RNG testing

The idea is to use a “hypothesis test,” where the hypothesis is that the tested RNG is good. The test looks for deviations that should not occur in the output of a good RNG. Examples

- Bias
- Correlation over distance
- Long strings of the same number
- Length of increasing sequences
- Output distribution from known functions of the output
- Poker hands
- Monkey-on-a-typewriter
- Length to repetition

RNG test suites

These tests have been collected into test suites with particular choices of the parameters. Examples

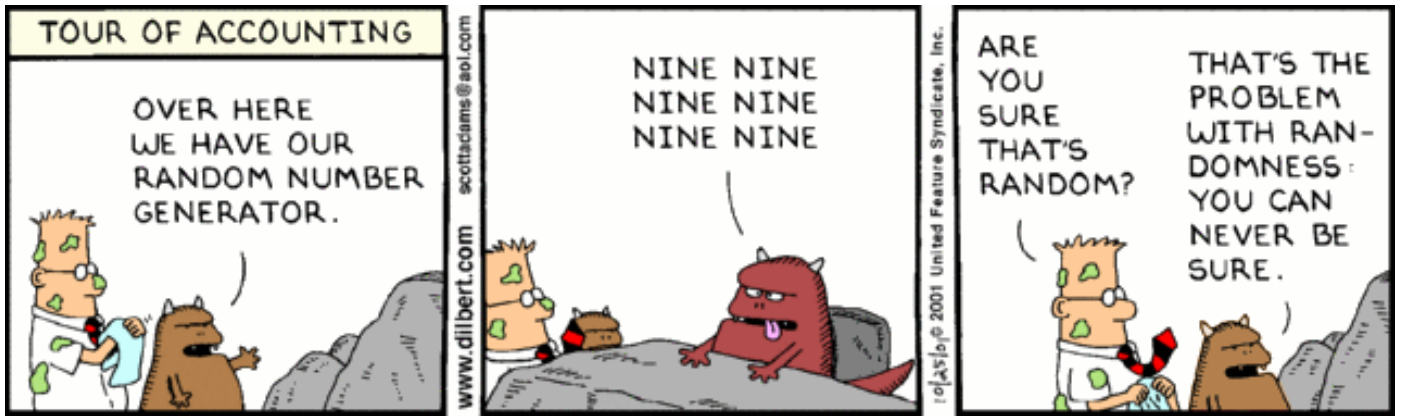
- Diehard (Marsaglia, 1995), good for simpler PRNGs, but not so well-documented
- NIST STS (-2010), well documented but from NIST
- ENT (Walker, 2008), simple teaching tool with limited documentation
- SPRNG (Mascagni, $\sim$2001), for parallell implementations
- Dieharder (Brown et al, -2013), easy to use and well documented
- TestU01 (L'Ecyuer, -2009), large and very well documented

Humans are bad random number generators (possibly, imps are better. . .)

A popular exercise

- Imagine and record 100 random coin flip results
- Flip a coin and record the result 100 times

There will be statistical anomalies (unless you plan carefully)

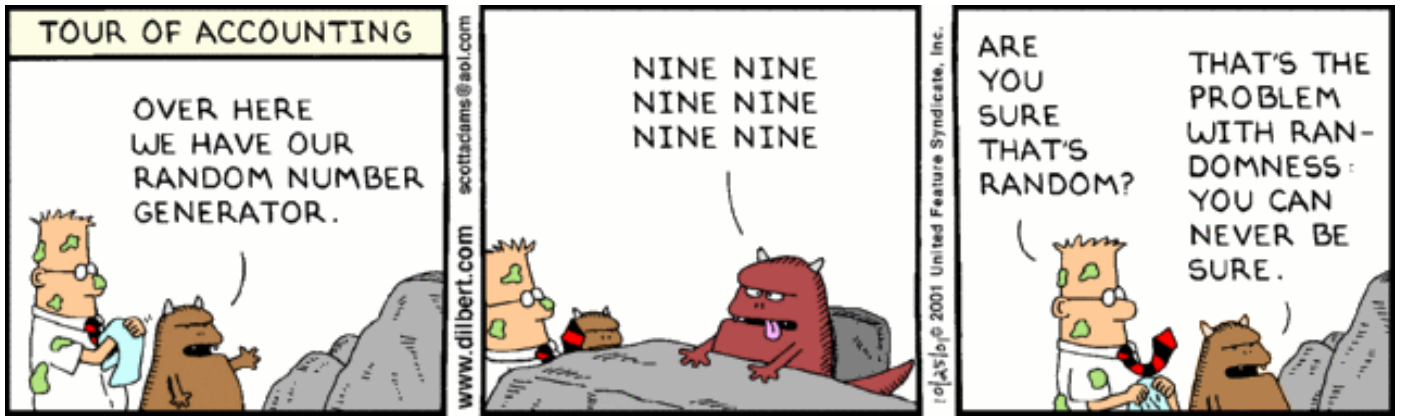


Humans are bad random number generators (possibly, imps are better. . .)

A popular exercise

- Imagine and record 100 random coin flip results
- Flip a coin and record the result 100 times

There will be statistical anomalies (unless you plan carefully)
People try to avoid patterns in a supposedly random output



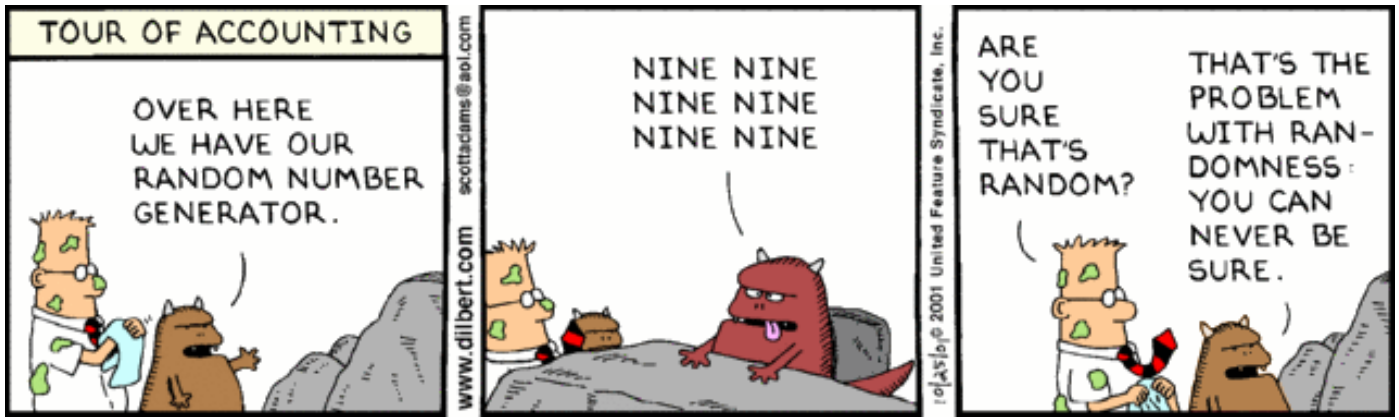
Random numbers

In principle, any stream cipher can function as a Pseudo Random Number Generator. The reverse is not always true
Some differences:

- Stream ciphers are used for speed, PRNGs can be slow
- Failing statistical tests is not always fatal for stream ciphers
- A PRNG may pass statistical tests, but still be reverse-engineered
- Re-keying a stream cipher is well-understood and used to avoid key-stream repetition, while re-seeding a PRNG is more of an art.

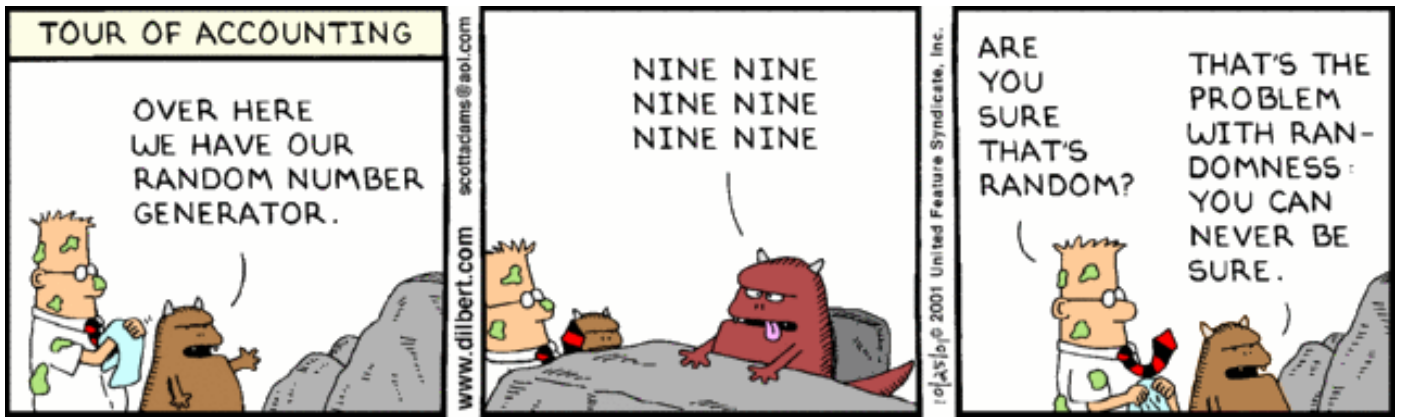
Not all PRNGs are cryptographically secure

- Example: the Mersenne Twister
- Very popular, implemented in many systems, period $2^{19337} - 1$, very good statistical properties
- Not suitable for cryptographic purposes (624 iterates makes it predictable)
- But there are modifications. . .



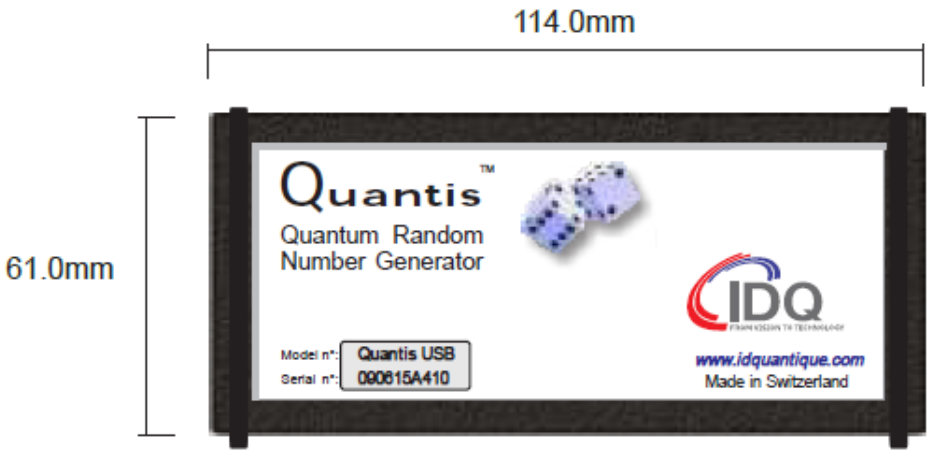
Base security on a hard mathematical problem: Blum-Blum-Shub

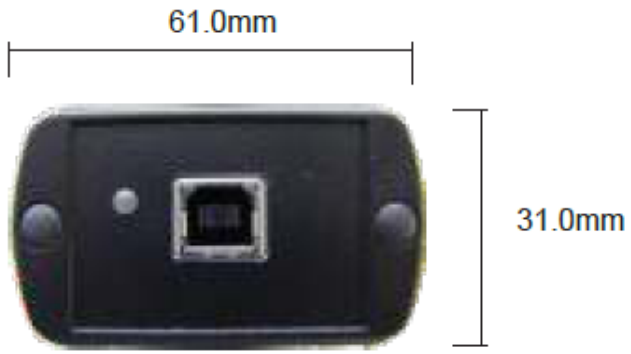
- Take two large primes p and q, both $= 3 \bmod 4$, set $n = pq$, choose random x relatively prime to n, and set $x_0 = x^2 \bmod n$
- Let $x_j = x_{j-1}^2 \bmod n$, output least significant bit
- Relatively slow, but secure as long as n cannot be factored (the proof of this appears later in the course)



Hardware RNGs

- Classical phenomena:
  - Atmospheric noise
  - Thermal noise in a resistor or Zener diode
- Quantum phenomena
  - Shot noise (photoelectric effect)
  - Nuclear decay
  - Electron tunneling in a reverse-biased transistor
  - Photon through half-silvered mirror





Stream ciphers, overview

Properties of stream ciphers:

- Benefits: fast, no fixed block size, errors do not propagate
- Drawbacks: No message integrity check, the devil is in the details

They do need to be used with care

If you don’t follow the algorithm to the letter, things can go very bad

Stream ciphers, overview

A stream cipher expands a short key into a long keystream, to be XORed with the plaintext

- LFSR-based: simple mathematical structure, needs nonlinear combination of several (A5/1, A5/2, E0, SNOW 3G)
- Many other variants, RC4 used to be popular but is now deprecated
- Another alternative is a block cipher in OFB or CTR mode (lecture 5)

Next lecture

- Block ciphers
- Feistel network
- DES
- Breaking DES (Differential cryptanalysis)