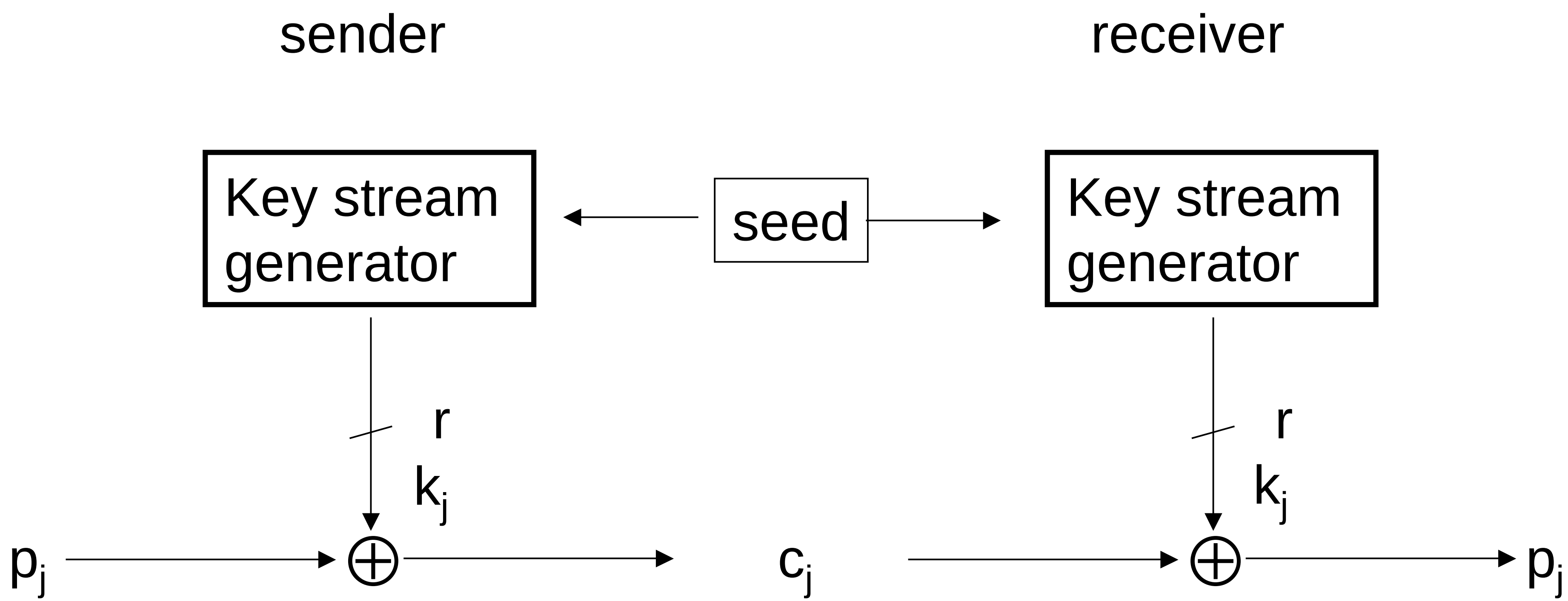
B.e) Stream Ciphers

B.125 Stream Ciphers

- Normally, stream ciphers are symmetric algorithms with encryption = decryption
- In this course we only consider symmetric stream ciphers.

B.126 Generic Design (Synchronous Stream Cipher)



$$\boxed{c_j = p_j \oplus k_j}$$

B.126 (continued)

- Both sender and receiver generate identical key stream sequences $k_1, k_2, ..$ (random numbers). The random numbers depend on the seed.
- The key stream is independent from plaintext and ciphertext.
- Encryption: $c_j = p_j \oplus k_j$
- Decryption: $p_j = c_j \oplus k_j$

Note: The *ciphertext digit* c_j depends on the plaintext p_j AND its position (= j) but not from any other plaintext digits.

B.127 General Remarks

- The key stream generator is a deterministic random number generator (pseudorandom number generator).
- The key stream is determined by the seed (to be kept secret !). The seed of the key stream generator is the pendant to the key of a block cipher.

Assumption: In the following we assume that the key stream generator generates r-bit strings (= random numbers, $r \geq 1$).

- Principally, a key stream generator may generate elements in any finite group. Then '$\oplus$' has to be replaced by the respective group operation.

B.127 (continued)

- Unlike the one-time pad cipher (cf. B.23) stream ciphers are not unconditionally secure against decryption attacks. (Why not?)
- Synchronous stream ciphers (cf. B.126) have some significant properties. In particular,
  - No error propagation, i.e. an altered ciphertext digit c_j does not affect the decryption of the remaining ciphertext.
  - The loss of a ciphertext digit c_j cannot be compensated.

B.127 (continued)

These properties imply:

- To guarantee data integrity further security mechanisms are needed (cf. also B.23)
- If some ciphertext digits got lost all at least from this step all ciphertext digits have to be transmitted once more.
- Alternatively, *self-synchronizing stream ciphers* could be applied (see B.141)

- In this section we restrict our attention to synchronous stream ciphers.

B.128 Decryption Attacks on Stream Ciphers

- In this section we restrict our attention to decryption attacks.
- Decryption Attacks on stream ciphers are typically known-plaintext attacks. Occasionally, even ciphertext-only attacks may be feasible.

Note: From the knowledge of some (plaintext, ciphertext) pairs $(p_{j\_1},c_{j\_1}),\ldots, (p_{j\_m},c_{j\_m})$ the adversary computes the corresponding random numbers $k_{j\_i} = c_{j\_i} \oplus p_{j\_i}$.

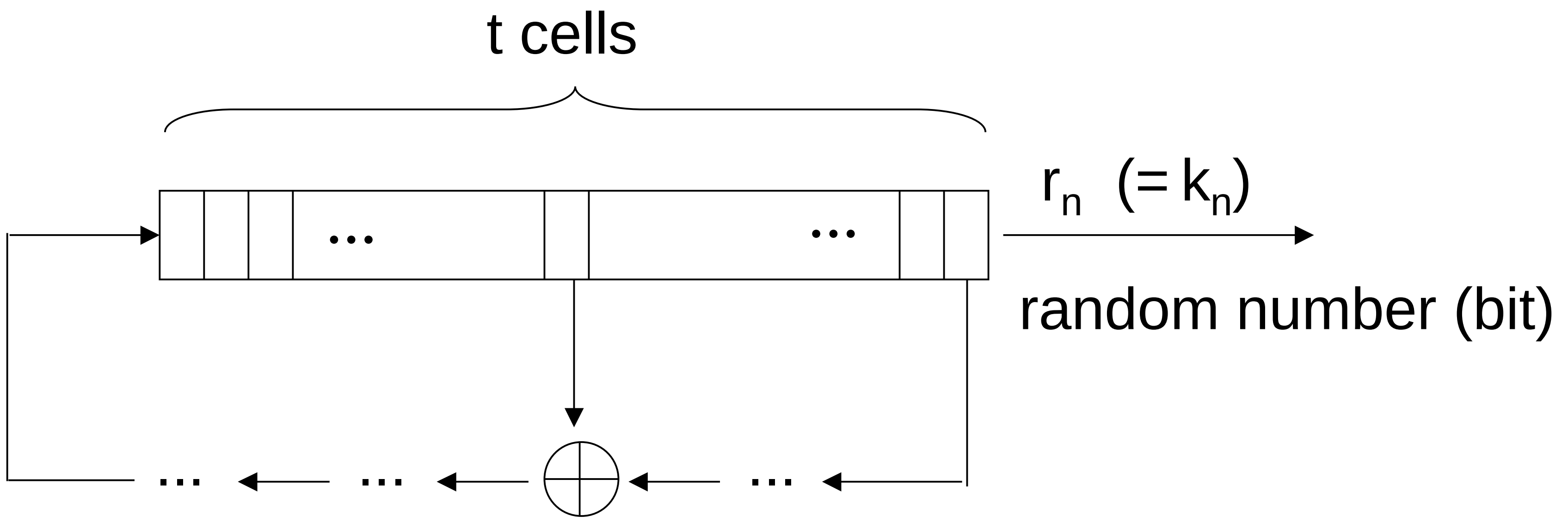
- Since the key stream is independent from the plaintext a chosen-plaintext attack does not improve the adversary’s chances of success compared to a known-plaintext attack.

B.129 The Key Stream Generator: Security Requirements

- It shall not be feasible to find the seed by exhaustive search. Hence the seed must be sufficiently long.
- The random numbers should assume all possible values with identical probability.
- The knowledge of some random numbers $k_{j\_1},\ldots,k_{j\_m}$ shall not allow an adversary to determine or to guess any further random numbers with non-negligibly higher probability than without the knowledge of $k_{j\_1},\ldots,k_{j\_m}$. The preferred goal, of course, is the seed as it allows the easy computation of all random numbers.

B.130 Example (Key Stream Generator)

Linear feedback shift register (LFSR) over GF(2)



Each cell stores a single bit. Content of the LFSR (= *internal state*) at time n from left to right: $r_{n+t},\ldots,r_{n+1}$

B.130 (continued)

1. The feedback value is computed (= XOR sum of particular cells (‘*taps*’)).
2. The content of all cells is shifted by one position to the right.
   - The feedback value is written into the left-most cell
   - The value that has been shifted over the right “border” of the LFSR is output (random bit)

B.130 (continued)

Note: If the cells $1 = s\_1 < \ldots < s\_m \leq t$ (labelled from the right to the left, beginning with '1') are taps then

$r_{n+t+1} = r_{n+s\_m} \oplus \ldots \oplus r_{n+s\_1}$ (recursion formula)

Fact: There is a correspondence between recursion formulae and polynomials over GF(2). More precisely,

$$r_{n+t+1} = r_{n+s\_m} \oplus \ldots \oplus r_{n+s\_1}$$

corresponds to the feedback polynomial

$$f(X) = X^t + X^{t+1-s\_2} + \ldots + X^{t+1-s\_m} + 1 \in GF(2)[X]$$

B.130 (continued)

Observation: The current internal state determines all following random numbers.

Consequence: At least from a certain step

- the internal state
- and hence the output sequence

are periodic.

Fact:

(i) The zero state (0,..,0) generates the constant output sequence 0,0,…

(ii) The period length $2^t - 1$ can be obtained ($\rightarrow$ primitive feedback polynomials).

Details: Blackboard

B.130 (continued)

Example: (t = 10) : The feedback polynomial

$f(X) = X^{10} + X^3 + 1$ is primitive.

Hence $r_{n+11} = r_{n+1} \oplus r_{n+8}$

provides a bit sequence with maximum period length $2^{10} - 1$ iff the initial state of the LFSR $\neq (0,\ldots,0)$.

B.131 Remark

- Due to their outstanding practical relevance we only consider LFSRs over GF(2) in this course.
- We mention that LFSRs can be defined over any finite field and over finite rings (e.g. over Z_n).

B.132 To Example B.130: Security

- The seed $r_1, r_2, \ldots, r_t$ determines the whole output sequence.
- Any random bit r_j can be written as a sum of the seed bits $r_1, r_2, \ldots, r_t$.
- Assume that the adversary knows m random bits bits $r_{i1}, r_{i2}, \ldots, r_{im}$. Let $\mathbf{s} := (r_1, r_2, \ldots, r_t)^T$ (seed!) and $\mathbf{z} := (r_{i1}, r_{i2}, \ldots, r_{im})^T$ then

$$A\mathbf{s} = \mathbf{z}$$

  where A is an $(m \times t)$-matrix A over GF(2).
- The seed **s** is a solution of the above equation. If rank(A) = t then **s** is the unique solution.

B.132 (continued)

Consequence: It is sufficient to know ≈ t random bits to recover the seed **s**.

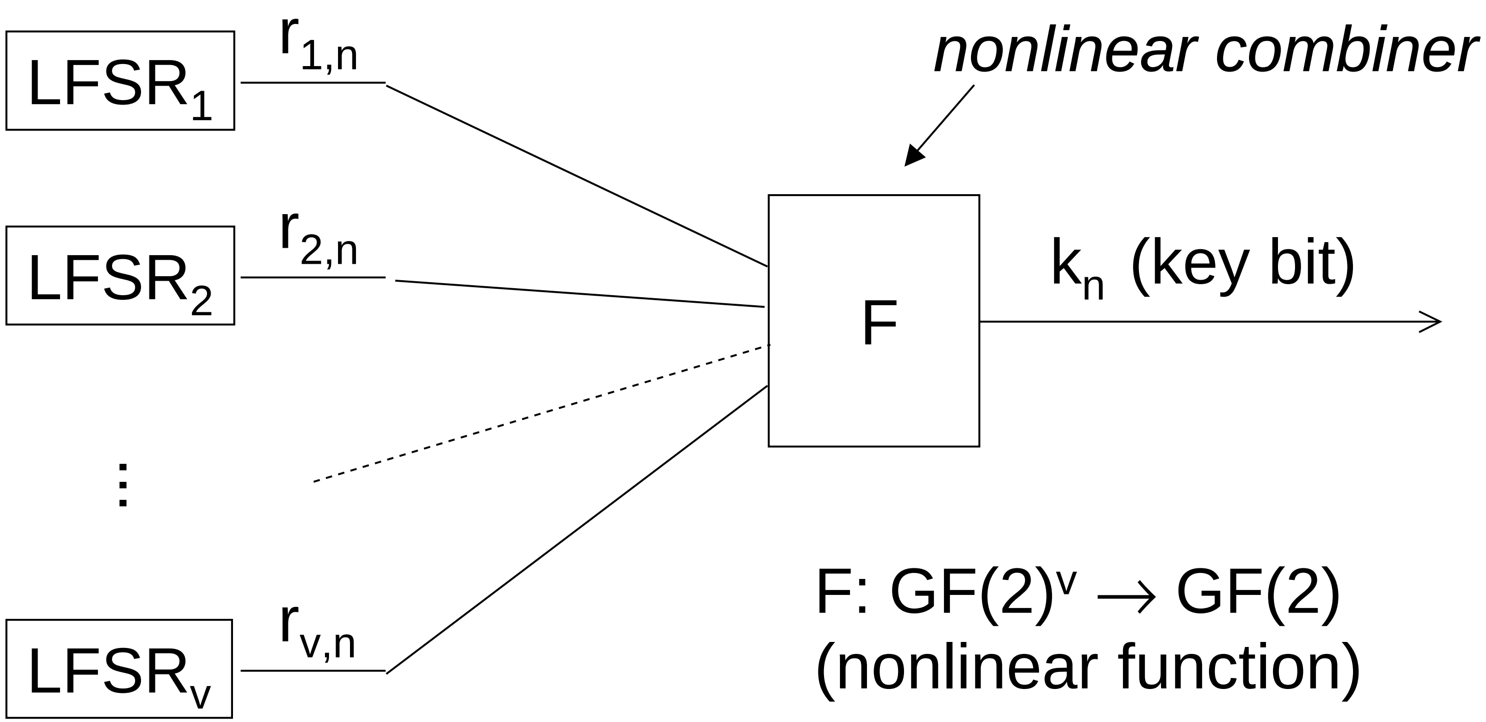
Fact: Even if the adversary does not know the taps the knowledge of ≈ 2t random bits is sufficient to recover the seed **s** (→ Berlekamp-Massey algorithm).

The key stream generator from Example B.130 (LFSR) is completely insecure.

Details: Blackboard

B.133 Example (Key Stream Generator)

Several LFSRs with a nonlinear combiner



F: $GF(2)^v \rightarrow GF(2)$
(nonlinear function)

B.133 (continued)

Observation:

- If $LFSR_j$ has length t_j, if all feedback polynomials are primitive and all LFSR seeds are non-zero (i.e., $\neq$ (0,…,0)) then $(r_{1,1}, r_{2,1}, \ldots, r_{v,1})$, $(r_{1,2}, r_{2,2}, \ldots, r_{v,2})$, …has period $p := \mathrm{lcm}(2^{t\_1}-1, 2^{t\_2}-1, \ldots, 2^{t\_v}-1)$
- The period of $k_1, k_2, \ldots$ divides p (usually it equals p)

B.133 (continued)

Assumption: The adversary knows a part of the key stream sequence.

Straight-forward attack (exhaustive seed search):

- The adversary computes the key stream sequences for all possible seeds (= $2^{t\_1+t\_2+\ldots+t\_v}$) and compares it with the known random numbers.
- If the computed key stream sequence differs from the known random numbers the assumed seed candidate is definitely false.
- If the attacker knows sufficiently many random numbers only the correct seed should remain.

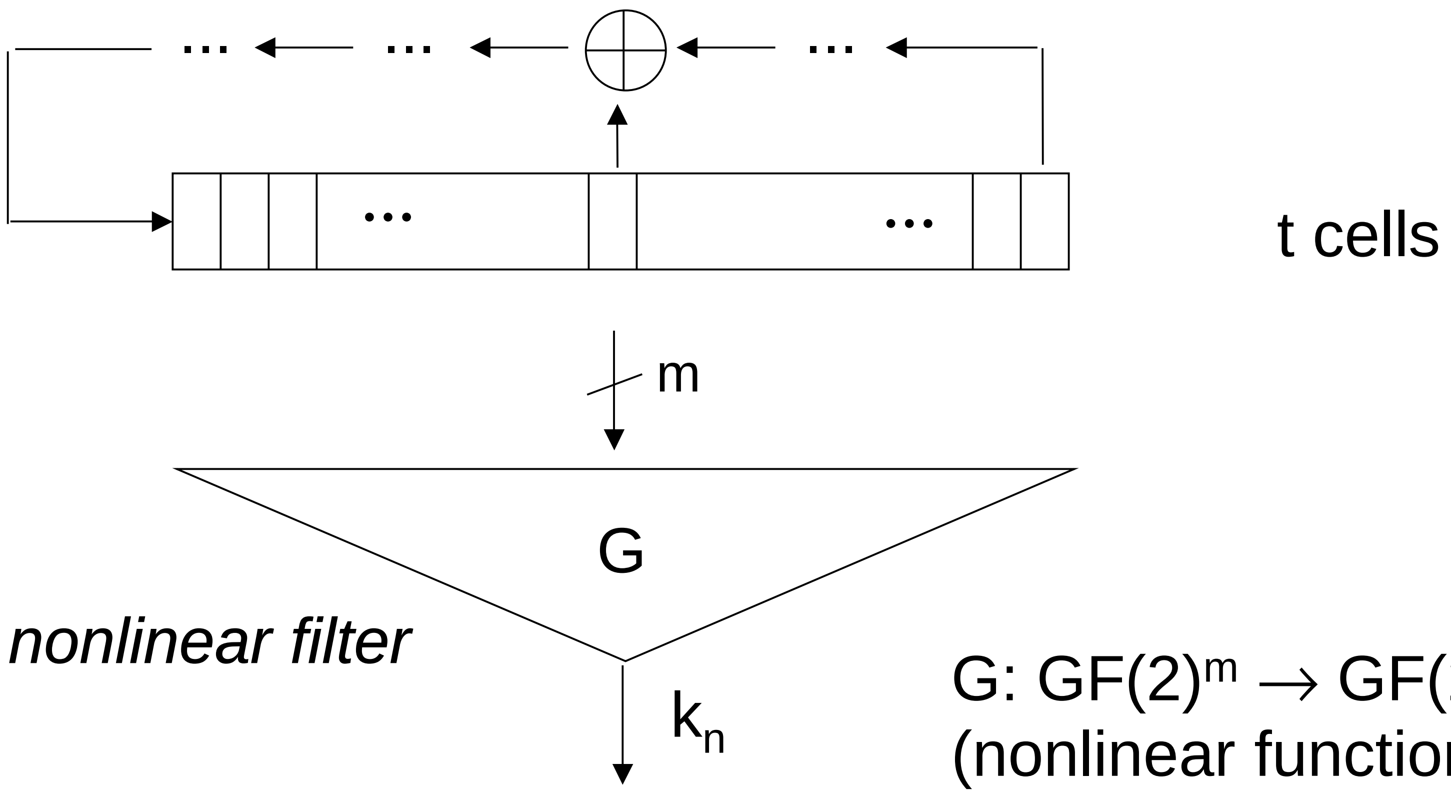
B.133 (continued)

Assessment: Principally, the straight-forward attack works. If $2^{t\_1+t\_2+\ldots+t\_v}$ is sufficiently large it is yet not practically feasible.

Remark: Many research work has been devoted to find more efficient attacks. At the end of this section we describe Siegenthaler's attack (cf. B.142f.), maybe the most elementary non-trivial attack.

B.134 Example (Key Stream Generator)

LFSR with a nonlinear filter



G: $GF(2)^m \rightarrow GF(2)$
(nonlinear function;
input = m internal state bits)

B.135 Example (Key Stream Generator)

Block cipher in OFB mode ($\rightarrow$ B.36)

Security: depends on the block cipher Enc

Note: Assume that an adversary knows the random numbers $r_i,\ldots,r_{i+j}$. Finding r_{i+j+1} or r_{i-1} is at least as difficult as a chosen-plaintext, resp. a chosen-ciphertext attack, on the block cipher Enc.

Proof: Exercise

B.136 Typical Applications

- Typically, stream ciphers are used by applications that meet at least some of the following assumptions:
  - The device has restricted computational resources.
  - Many random numbers have to computed in real-time.
  - Single plaintext bits or short bit sequences have to be processed immediately.
  - (At least to a certain extent) altered ciphertext digits are tolerable but these errors should not propagate.

B.136 (continued)

- Typical applications that use stream ciphers are mobile communication, wireless short range communication, WLANs etc.
- Well-known stream cipher algorithms: A5 (several variants) and f8 (mobile communication (GSM, resp. UMTS)), E0 (Bluetooth), RC4 (WLAN, WEP protocol), SEAL, …
- The goal of the eSTREAM project (organized by the EU ECRYPT network) is “to identify new stream ciphers that might become suitable for widespread adoption”.

B.137 Remark

- Principally, any pseudorandom number generator *that is suitable for cryptographic applications* may be used as a key stream generator.
- Note: Besides statistical properties (uniform distribution, …) it must in particular practically infeasible to find predecessors and successors of known subsequences with non-negligible probability.

B.137 (continued)

- Key stream generators with high throughput are of particular interest if they need only little resources (computation time, memory).
- For this reason various constructions using LFSRs have intensively been investigated.
- We do not deepen this topic in this course.
- Note: Since the key stream is independent from plaintext and ciphertext it can be pre-computed in idle time.

B.138 Random Number Generators (RNGs) for Cryptographic Applications

- Apart from stream ciphers a large number of cryptographic primitives and protocols need random number generators (RNGs).
- RNGs are needed, for instance, for the generation of
  - session keys
  - challenges (cf. B.30)
  - signature parameters ($\rightarrow$ Chap. C)
  - ephemeral keys ($\rightarrow$ Chap. C)
  - …

B.139 Remark

- Roughly speaking, RNGs can be divided into *true* and *deterministic* (*pseudorandom*) RNGs.
- The class of true RNGs itself falls into two subclasses containing *physical* RNGs (using dedicated hardware) and *non-physical* RNGs (using non-deterministic system data and / or user’s interaction).
- Combinations of the basic types are possible (hybrid RNGs).

<u>Details:</u> Blackboard

B.139 (continued)

- The international ISO norm 18031 “Random Bit Generation” provides examples and design principles for deterministic and true RNGs.
- Examples for deterministic RNGs can also be found in the “Handbook of Applied Cryptography”, for instance.
- In Germany the evaluation guidances AIS 20 and AIS 31 are mandatory if an internationally recognized IT security certificate (according to the so-called “Common Criteria”) is applied for. These guidances describe requirements on the RNG and the applicant’s and the evaluator’s tasks.

B.140 Warning

- Random numbers are also needed for stochastic simulations and Monte-Carlo integrations which play an important role e.g. in several fields of applied mathematics, computer science and applied sciences.
- Unlike for cryptographic applications (cf. B.129 and B.138, for instance) it is fully sufficient if these random numbers behave statistically inconspicuously.

B.140 (continued)

- Pseudorandom generators that are appropriate for stochastic simulations or Monte Carlo integrations may be totally unsuitable for cryptographic applications!
- Not everyone is aware of this fact, which has caused a lot of confusion.

B.141 Self-Synchronizing Stream Ciphers

- For self-synchronizing stream ciphers the key stream depends on a key and on some previous ciphertext digits.
- Roughly speaking, the general design of self-synchronizing stream ciphers is like the CFB mode for block ciphers (Example!).
- In particular, self-synchronizing stream ciphers can compensate the loss of ciphertext digits. (Depending on the application it may not be necessary to repeat the transmission.)
- On the negative side the key stream cannot be precomputed.

B.142 Siegenthaler's Attack

- We end this section with a well-known attack, which was introduced by Siegenthaler in 1984.
- Scenario: LFSRs with a nonlinear combiner (cf. B.133)
- Example: v=3, $F(x,y,z) := xy \oplus xz \oplus yz$;

  LFSR lengths: $t_1 = 29$, $t_2 = 31$, $t_3 = 33$;

  The attacker knows $k_{j\_1},\ldots,k_{j\_m}$

  Straight-forward attack (cf. B.133): requires the check of $2^{29+31+33} = 2^{93}$ seed candidates for $(LFSR_1, LFSR_2, LFSR_3)$, which is practically infeasible.

B.142 (continued)

x	y	z	F(x,y,z)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

F is balanced (four „0“s, four „1“s). But ...

B.142 (continued)

x	y	z	F(x,y,z)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

B.142 (continued)

- Observation: Assume that X,Y,Z are independent random variables that are uniformly distributed on {0,1}, i.e. Prob(X = 0) = … = Prob(Z = 1) = 0.5

  Then

  - Prob(X = F(X,Y,Z)) = 0.75

  and, similarly,

  - Prob(Y = F(X,Y,Z)) = 0.75
  - Prob(Z = F(X,Y,Z)) = 0.75

- Conclusion: We may expect that for about 75 % of the sub-indices $i \in \{1,\ldots,m\}$ we have $r_{1,j\_i} = k_{j\_l}$.

B.142 (continued)

Siegenthaler's Attack:

For each possible seed candidate s_1' for $LFSR_1$ do {

- compute the output sequence of $LFSR_1$ until index j_m
- determine the fraction $n(s_1')$ of the bits $r_{1,,j\_1}, r_{1,,j\_2},\ldots,r_{1,,j\_m}$ that are identical with the known part of the key stream sequences

Note:

(i) For the correct seed s_1 we may expect $n(s_1) \approx 0.75$.

(ii) For any false seed s_1' we may expect $n(s_1') \approx 0.5$.

(iii) Unless m is large the value $n(s_1')$ of some false seed candidates may exceed 0.5 considerably.

B.142 (continued)

Siegenthaler's Attack:

For each possible seed candidate s_1' for $LFSR_1$ do {

- compute the output sequence of $LFSR_1$ until index j_m
- determine the fraction $n(s_1')$ of the bits $r_{1,,j\_1}$, $r_{1,,j\_2}, \ldots, r_{1,,j\_m}$ that are identical with the known part of the key stream sequences
- add s_1' to a set S_1 of 'likely' seeds if $n(s_1') > th_1$ where $th_1 \in (0.5,0.75)$ is a suitably selected threshold

}

B.142 (continued)

- The attacker performs the same procedure for $LFSR_2$ and $LFSR_3$, too, obtaining three sets S_1, S_2, S_3 of 'likely' seeds of the particular LFSRs.
- The attacker checks all triples $(s_1', s_2', s_3') \in S_1 \times S_2 \times S_3$ (comparison of the generated output sequences at the positions $j_1, \ldots, j_m$ with the known bits $k_{j\_1}, k_{j\_2}, \ldots, k_{j\_m}$).

B.142 (continued)

Note:

The threshold th_1 (resp. th_2, resp. th_3) should be selected that

- S_i of contains the true seed s_i with high probability
- $| S_i |$ is not too large

The choice of th_i should consider the parameters t_i and m (apply the Central Limit Theorem as if the output of the LFSRs and the key stream bits were truly random).

B.142 (continued)

Efficiency:

- Siegenthaler’s attack is much more efficient than the straight-forward attack because the attacker determines the seeds of all LFSRs independently.
- The workloads for the individual LFSRs essentially add up whereas in the straight-forward attack these workloads multiply!
- In our example finding the seed of $LFSR_3$ dominates the workload (2^{33} seed candidates vs. 2^{93} in the straight-forward attack).

Note: The number m of known random numbers must be larger than in the straight-forward attack.

B.143 Remark

- Siegenthaler pointed out that his attack even works as a ciphertext-only attack (due to the non-uniformity of the plaintext).

- Source of Siegenthaler's attack:

  The correlation of the function value $F(x,y,z)$ with x (resp. with y, resp. with z).

B.143 (continued)

Preventing Siegenthaler's attack:

- Let $F:GF(2)^v \rightarrow GF(2)$ and let $X_1,X_2,\dots,X_v$ denote independent random variables that are uniformly distributed on $\{0,1\}$.

  Assume further that $F(X_1,X_2,\dots,X_v)$ and $(X_{j\_1},X_{j\_2},\dots,X_{j\_d})$ are independent for any choice of indices $j_1,\dots,j_d \in \{1,\dots,v\}$. Then F is said to be *correlation-immune* of order d.
- Consequence: To perform Siegenthaler's attack then the seeds of at least (d+1) LFSRs have to be guessed simultaneously.

Details: Blackboard + Exercises