SLOVAK UNIVERSITY OF TECHNOLOGY

FACULTY OF ELECTRICAL ENGINEERING
AND INFORMATION TECHNOLOGY

Department of Mathematics

Ilkovičova 3, 812 19 Bratislava 1

Milan Vojvoda

STREAM CIPHERS AND HASH FUNCTIONS – ANALYSIS OF SOME NEW DESIGN APPROACHES

dissertation

Supervisor: Professor RNDr. Otokar Grošek, PhD.

Degree Course: 11-14-9 Applied Mathematics

July 2004
Bratislava

Statement

Hereby I state that this dissertation is the result of my own work and does not contain results, nor partial results, of other people unless it is explicitely stated. I state that I used only the literature listed in bibliography of this dissertation.

Bratislava, July 16, 2004 Milan Vojvoda

Acknowledgements

I would like to thank Professor Otokar Grošek for all the comments and suggestions on improving this thesis. Also I wish to thank all the staff of the Department of Mathematics of Faculty of Electrical Engineering and Information Technology for the friendly atmosphere they provided to work in.

Contents

Chapter 1

List of symbols

$\mathbb{Z}$	the ring of integers
$\mathbb{N}$	the set of integers $\{1, 2, 3, \dots\}$
$\mathbb{Z}_m$	the ring of integers reduced modulo m
$\mathbb{Z}_m^n$	the n-dimensional vector space over $\mathbb{Z}_m$
$GF(q)$	the finite field of order q
$GF(q)[X]$	the algebra of polynomials over $GF(q)$ (in indeterminate X)
H	Shannon's entropy
$I(X, Y)$	the amount of mutual information between X and Y
gcd	greatest common divisor
lcm	least common multiple
LSB(X)	the least significant bit of a binary representation of X
PRBG	Pseudo-Random Bit Generator
LFSR	Linear Feedback Shift Register
FCSR	Feedback with Carry Shift Register
NSG	Natural Sequence Generator

Chapter 2

Introduction

2.1 Motivation

Cryptography is nowadays an inseparable part of a large number of business processes and our everyday activities although in many cases we have no idea about it. Internet, bank transfers, mobile phones are common examples of the usage of cryptography.

The main goals of cryptography are to achieve privacy, data integrity, authenticity, and non-repudiation [54].

In this dissertation we will deal with some aspects of

- stream ciphers – encryption algorithms, used to ensure privacy, and
- hash functions – algorithms used to ensure data integrity.

A significant milestone in the area of stream ciphers was the year 1917 when G.Vernam invented his one-time pad. It was the first unconditionally secure cipher (according to the Shannon's definition) against the ciphertext-only attack. Stream ciphers are popular due to their high encryption/decryption speed. Their simple and cheap hardware design is often preferred in real-world applications. Probably the most widely used application of stream ciphers are mobile phones (GSM, UMTS).

Especially after the publication of the final report of the project NESSIE [68] one can get the feeling that stream ciphers are less secure than block ciphers since no stream cipher was recommended from the project proposals. That is probably not true but the design of block ciphers seems to be more sophisticated in the present time.

However there are still many open questions in the "classical" design of stream ciphers and new design trends (inspired by the design of block ciphers) appeared in recent years [71], [33], [13].

At the same time this was a new call for random and pseudorandom sequences which lasts up to these days. Modelling, simulation, and cryptography are probably the most important of them. The so-called *truly random sequences* are usually obtained from physical sources or processes with random behaviour. Emitters, and noise samplers in electrical circuits are common examples. However, slow speed of a truly random sequence production is a severe problem. Moreover a sample we get is finite and hence some other requirements are needed. This leads to an intensive research in the area of *pseudorandom sequences.* In spite of their deterministic production they possess (statistical) properties common to truly random sequences. Moreover they can be produced very fast, efficiently, and are reproducible when some initial seeds are known - what is an important feature in some applications.

In the area of cryptography pseudorandom sequences are of tremendous interest. Many items in cryptographic protocols (e.g. challenges), signature schemes and of course the keys for cryptosystems must be generated in a random fashion. Moreover pseudorandom sequences are the core of stream ciphers. Pseudorandom generators suitable for use in cryptographic applications may need to meet stronger requirements than those for other applications. In particular, their outputs must be unpredictable in the absence of knowledge of the inputs.

It can be simply said that hash functions "compress" (in a lossy manner) a string of an arbitrary length (a message) to a string of a fixed length (a message digest or hash value). The main goal is not to compress the message but to produce a message digest that in some sense represents this message. (An analogous example is the fingerprint of a human being.)

Hash functions have a large number of applications in computer science (optimized access to the stored data) and in cryptography (integrity protection of stored/transmitted data) as well. The uniform distribution of message digests is usually the most important requirement in computer science applications. However hash functions used in cryptographic applications may need to meet stronger requirements. First of all they must be one-way, i.e. given a message digest it must be "difficult" (computationally infeasible) to find a message that would hash to this given message digest. Moreover, given a message it must be "difficult" to produce another message such that these two messages have the same message digest.

Hash functions are used almost in all practical digital signature schemes. (Digital signature schemes are rather slow which makes it impractical to sign large messages. A common approach is to produce a message digest and then to sign it.)

Clearly, integrity protection via hash functions differs from the integrity protection via the well-known cyclic redundancy codes (CRCs) that enable the detection of errors that occur due the noise in the transmission channel. Analogy in hash functions is a message authentication code (MAC) that is a hash function with a secret key.

2.2 Research targets

In this dissertation there are properties of some specifically constructed pseudorandom sequences studied both from the point of view of cryptography and cryptanalysis. This dissertation deals also with a relatively new direction in cryptography – the usage of quasigroups in the design of stream ciphers and hash functions.

The research targets can be formulated as follows:

1. to introduce necessary basic notions concerning pseudorandom generators and hash functions in cryptography;
2. to study cryptographic properties of the concatenation of periods of several ml-pseudorandom sequences;
3. to cryptanalyse stream ciphers that are based on the concatenation of transformed runs of two ml-sequences;
4. to cryptanalyse one stream cipher based on a quasigroup, which was proposed in [59],
5. to study the security of a hash function based on a quasigroup, which was proposed in [21], [22].

The structure of this dissertation is as follows. Section 3 deals with the state of the art in stream ciphers and hash functions. Basic notions concerning stream ciphers are outlined in Section 3.1. Statistical tests of randomness of sequences are mentioned in Section

3.2. Design approaches for stream ciphers are described in Section 3.3 including several examples in Section 3.4. Basic attacks on stream ciphers are described in Section 3.5. Basic notions concerning hash functions are outlined in Section 3.6. Design approaches for hash functions are described in Section 3.7. Basic attacks on hash functions are described in Section 3.8. Finally, a short information about the projects NESSIE and CRYPTREC is given in Section 3.9 including the recommended stream ciphers and hash functions. The results of our research are presented in Section 4. This section is based on the author's papers [85], [86], [87], [88], [89], [90], [91], [92] and [93]. Cryptographic properties of the concatenation of periods of several ml-pseudorandom sequences are studied in Section 4.1. Section 4.2 deals with cryptanalysis of a stream cipher based on the concatenation of transformed runs of two ml-sequences. Several attacks on a stream cipher based on a quasigroup (proposed in [59]) are described in Section 4.3. The properties of a hash function based on a quasigroup (proposed in [21], [22]) are studied in Section 4.4. Conclusions are given in Section 5. A complete list of author's papers, conference presentations, and other related activities can be found in Section 6. Finally, Bibliography is to be found at the end of this dissertation.

Chapter 3

Stream ciphers and hash functions – state of the art

3.1 Stream ciphers

Stream ciphers form an important class of symmetric (classical, secret-key) cryptosystems.

Definition 3.1.1 *[32], [83] A cryptosystem is formally a 5-tuple* $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$*, where:*

- $\mathcal{P}$ *is a finite set of plaintexts,*
- $\mathcal{C}$ *is a finite set of ciphertexts,*
- $\mathcal{K}$ *is a finite set of keys,*
- $\mathcal{E}$ *is a finite set of encryption transformations* $e_{k_e} : \mathcal{P} \to \mathcal{C}$*, where the key* $k_e \in \mathcal{K}$ *is the parameter of the encryption transformation,*
- $\mathcal{D}$ *is a finite set of decryption transformations* $d_{k_d} : \mathcal{C} \to \mathcal{P}$*, where the key* $k_d \in \mathcal{K}$ *is the parameter of the decryption transformation. The following must hold:* $d_{k_d}(e_{k_e}(P)) = P, \forall P \in \mathcal{P}$.

Cryptosystems can be roughly subdivided into the following two groups according to the relation between the encryption and decryption keys:

symmetric (classical). Encryption key can be easily computed from the decryption key and vice versa. Since these keys are usually (specularly) identical, they are not considered separately. Security of symmetric cryptosystems is based on the secrecy of the key.

asymmetric (public). Different keys are used for encryption and decryption. Encryption (public) key can be made public. Decryption (private) key must be kept secret (the parameters used to calculate the decryption key must be kept secret, too). It is required that it is not possible to compute (in a real-time) the decryption key from the encryption key.

According to the encryption transformation cryptosystems can be subdivided as follows [72]:

block ciphers. They transform a plaintext block using a fixed encryption transformation, i.e. a plaintext block is substituted with another block. It is required that the block is large enough, to make the dictionary attack impossible. Typical block size nowadays is 128 or 256 bits. However in some applications the 64-bit block is still favourable.

stream ciphers. Individual blocks of plaintext (also called symbols, because they are much shorter than blocks used within block ciphers) are transformed using a time-varying encryption transformation that is dependent on the inner state of the stream cipher.

High encryption/decryption speed and simple hardware implementation are common and most favourable properties of stream ciphers. Moreover, their properties are usually analyzable and provable using algebraic techniques.

Stream ciphers can be subdivided into [72, pp.6–7]:

synchronous. The next inner state of the stream cipher depends only on its previous inner state and not on plaintext,

self-synchronizing. The next inner state of the stream cipher depends on its previous inner state and on a fixed number of previously encrypted symbols.

The description of its advantages, disadvantages and resistance against several kinds of attacks can be found in [78, pp.197–199, 202–203], [54, pp.193–195], [72, pp.14–15].

Remark 3.1.2 *One may construct a stream cipher also from a block cipher [78, pp.189–211]. Running a block cipher in the so called output feedback (OFB) mode or in the cipher feedback (CFB) mode, respectively we obtain a synchronous or a self-synchronizing stream cipher, respectively.*

An important position among the stream ciphers has the so-called binary additive stream cipher.

Definition 3.1.3 *Let $P = (p_0, p_1, \ldots, p_{N-1})$ denote the bits of the plaintext, $C = (c_0, c_1, \ldots, c_{N-1})$ denote the ciphertext bits, and $z = (z_0, z_1, \ldots, z_{N-1})$ denote the keystream bits. Binary additive stream cipher is a synchronous stream cipher. Its encryption transformation is given as:*

$$c_i = p_i \oplus z_i, \quad i = 0, 1, \ldots, N-1.$$

The symbol $\oplus$ denotes the addition modulo 2 or in other words it is the XOR operation. Decryption transformation is then:

$$p_i = c_i \oplus z_i, \quad i = 0, 1, \ldots, N-1.$$

If the keystream z is also the key k of the cryptosystem, this cipher is also known as the Vernam cipher. If the individual bits of the keystream z were produced randomly and independently, the Vernam cipher is absolutely secure (according to the Shannon's definition, see Section 3.5) against the ciphertext-only attack. Let H be the Shannon's entropy. Hence the following holds: $H(P/C) = H(P)$, i.e. the uncertainty about the plaintext if the ciphertext is known is the same as the uncertainty about the plaintext itself.

Shannon has proved, that the necessary condition to consider a symmetric cryptosystem as an absolutely secure one is $H(k) \geq H(P)$ (i.e. uncertainty about the key of the cryptosystem may not be smaller than the uncertainty about the plaintext). If the bits of the key were produced randomly and independently, then $H(k) = \|k\|$, where $\|k\|$ denotes the number of bits in the key k. Thus $\|k\| \geq H(P)$. As it can be seen, the Vernam cipher is the optimal solution from the point of view of the key length.

Two important problems arise when the Vernam cipher is used in real-world applications:

key generation. One of the basic security principles is that the key can be used only once. Thus a new key is needed for each message to be encrypted. Moreover the key must be as long as the message and the key bits must be generated randomly and independently. It can be done using a physical true random number generator. However such generators are very slow which makes them impossible to use when a heavy traffic must be encrypted.

key distribution. Both, a sender and a receiver must agree on a key. The key must be transmitted from the sender to the receiver. Thus a trusted and secure channel is necessary to be shared between the sender and receiver.

These were the reasons that has brought the pseudorandom bit generators into the centre of interest in the area of stream ciphers.

Definition 3.1.4 *[83, Chapter 12], [54, Chapter 5] Let $\|k\|, \|z\|$ be positive integers such that $\|z\| \geq \|k\| + 1$. A $(\|k\|, \|z\|)$-pseudorandom bit generator (more briefly, a $(\|k\|, \|z\|)$-PRBG) is a deterministic algorithm (running in polynomial time as a function of $\|k\|$) which, given a truly random binary sequence k of length $\|k\|$, outputs a binary sequence z of length $\|z\|$ which "appears" to be random. The input to the PRBG is called the seed, while the output of the PRBG is called a pseudorandom bit sequence. (PRBGs used in stream ciphers to produce keystreams are often referred to as keystream or running-key generators.)*

The design goal in stream ciphers is to efficiently produce pseudorandom sequences - keystreams, i.e. sequences that possess properties common to truly random sequences, that are unpredictable and in some sense "indistinguishable" from these sequences.

Definition 3.1.5 *[83, Chapter 12] Suppose p_0 and p_1 are two probability distributions on the set $Z_2^{\|z\|}$ of bit-strings of length $\|z\|$. Let $A : Z_2^{\|z\|} \to \{0,1\}$ be a probabilistic algorithm that runs in polynomial time (as a function of $\|z\|$). Let $\epsilon > 0$. For $j = 0, 1$, define*

$$E_A(p_j) = \\ = \sum_{(z_1,\ldots,z_{\|z\|}) \in Z_2^{\|z\|}} p_j(z_1, \ldots, z_{\|z\|}) p(A(z_1, \ldots, z_{\|z\|}) = 1 | (z_1, \ldots, z_{\|z\|})).$$

We say that A is an ϵ-distinguisher of p_0 and p_1 provided that $\|E_A(p_0) - E_A(p_1)\| \geq \epsilon$, and we say that p_0 and p_1 are ϵ-distinguishable if there exists an ϵ-distinguisher A of p_0 and p_1.

Definition 3.1.6 *[54, Chapter 5, p.171] A PRBG is said to pass all polynomial-time statistical tests if no polynomial-time algorithm can correctly distinguish between an output sequence of the generator and a truly random sequence of the same length with probability significantly greater than* 1/2.

Definition 3.1.7 *[83, Chapter 12] Let p_1 be the probability distribution on the set $Z_2^{\|z\|}$ of bit-strings of length $\|z\|$ induced by the $(\|k\|, \|z\|)$-PRBG. Then the probabilistic algorithm B_i, $1 < i \leq \|z\|$ is an ϵ-next bit predictor for $(\|k\|, \|l\|)$-PRBG if and only if*

$$\sum_{(z_1,\ldots,z_{i-1}) \in Z_2^{i-1}} p_1(z_1, \ldots, z_{i-1}) p(z_i = B_i | (z_1, \ldots, z_{i-1})) \geq \frac{1}{2} + \epsilon.$$

See [83, Chapter 12] for the connections between distinguishers and next-bit predictors.

Definition 3.1.8 *[54, Chapter 5, p.171] A PRBG is said to pass the next-bit test if there is no polynomial-time algorithm which, on input of the first l bits of an output sequence z, can predict the $(l+1)^{st}$ bit of z with probability significantly greater than* 1/2.

The importance of the next-bit predictors is expressed in the following Theorem.

Theorem 3.1.9 *[54, Chapter 5, p.171] A PRBG passes the next-bit test if and only if it passes all polynomial-time statistical tests.*

Studying properties and security of almost all practical designs of stream ciphers using the previously stated notions of unpredictability and indistinguishability is almost impossible. That is why other (weaker) measures are used in real world.

Notation 3.1.10 *Let an infinite sequence z of elements from a symbol set S be denoted as $z = z_0, z_1, z_2, \ldots,$ $z_i \in S$, $i \geq 0$. Further let the N-couple beginning of this sequence be denoted as $z^N = z_0, z_1, z_2, \ldots, z_{N-1}$.*

There are usually binary sequences used in cryptographic applications, i.e. $S = \{0, 1\}$. Since many of the used sequences are produced by some finite automata, it is natural to define the period of the sequence.

Definition 3.1.11 *Let $z = z_0, z_1, z_2, \ldots$ be an infinite sequence. If there exist $r, q \in \mathbb{N}$, $q \geq 0, r > 0$, such that $z_q = z_{q+r}$, $z_{q+1} = z_{q+r+1}, \ldots$, then the sequence z is called ultimately periodic. If $q = 0$, the sequence z is referred to as periodic. The smallest integer r for which the previous statements hold is called the period of the sequence z.*

Grouping identical adjacent elements in a sequence together leads to the notion of a run.

Definition 3.1.12 *Let $z^N = z_0, z_1, z_2, \ldots, z_{N-1}$ be an N-couple sequence. Then a subsequence $z_i, z_{i+1}, \ldots, z_{i+d-1}$, $0 \leq i \leq N-1$, $1 \leq d \leq N-i$ is called a run of length d, if $z_i = z_{i+1} = \cdots = z_{i+d-1}$ provided $z_{i-1} \neq z_i$ (if $i > 0$) and $z_{i+d-1} \neq z_{i+d}$ (if $i+d < N$). If the sequence z^N is binary then a run consisting of ones, resp. zeros is called a block, resp. a gap.*

Definition 3.1.13 *[4] Relation*

$$\begin{gathered} a_0 s_i + a_1 s_{i-1} + \cdots + a_m s_{i-m} = 0, \\ a_i, s_i \in GF(q), a_0 \neq 0, a_m \neq 0, \\ i = m, m+1, m+2, \ldots \end{gathered} \tag{3.1}$$

is called the mth-order linear recurring relation. Sequence $s_0, s_1, s_2, \ldots$ satisfying this relation is called the mth-order linear recurring sequence (or the solution of the linear recurring relation). Elements $s_0, s_1, \ldots, s_{m-1}$ are referred to as the initial values.

Definition 3.1.14 *[4] There are two important polynomials connected to the linear recurring relation (3.1):*

the left characteristic polynomial: $a(x) = a_0 + a_1 x + \cdots + a_m x^m$,

the right characteristic polynomial: $\bar{a}(x) = a_0 x^m + a_1 x^{m-1} + + \cdots + a_m$.

It holds that $a(x) = x^m \bar{a}(1/x)$.

If there are "many" coefficients a_i of the (left or right, respectively) characteristic polynomial equal to zero, the polynomial is said to be "sparse". Otherwise the polynomial is referred to as the "dense" one.

Definition 3.1.15 *[57] Polynomial of the smallest degree, which is characteristic polynomial of the sequence $s_0, s_1, s_2, \dots$ is called the minimal polynomial of this sequence.*

Definition 3.1.16 *[57] Linear complexity of a sequence is the degree of the minimal polynomial of this sequence.*

Definition 3.1.17 *Let us denote $s = s_0, s_1, \dots, s_{N-1}$ as $\{s_i\}_{i=0}^{N-1}$. Let us define the following set of sequences constructed from the sequence s: $J = \{\{s_i\}_{i=0}^{j}, j = 0, 1, \dots, N-1\}$. The linear complexity profile is a function $J \to \mathbb{N}$, which assigns to each sequence from J its linear complexity. (Note that the linear complexity profile is a nondecreasing function.)*

Any sequence produced by some generator must possess the following features in order to be useful in cryptography as a keystream sequence:

- long period, since only its part can be used as a keystream for encryption. Moreover long period is a basic condition for unpredictability of a sequence. Another important fact is that the period of a sequence represents an upper bound for its linear complexity.
- large linear complexity. It is well known that a sequence with linear complexity m can be completely reconstructed from its $2m$ consecutive bits. More precisely, it is possible to find the minimal polynomial of this sequence using the Berlekamp-Massey algorithm [51]. The complexity of this algorithm is roughly the square of the sequence length. If m is small and a ciphertext is given, only a short plaintext is required to calculate the corresponding part of the keystream and then to find the minimal polynomial of the keystream which yields in the break of the cipher. (If the short plaintext is not known, the attacker may try to guess probable words in the plaintext.) The complexities of higher orders (quadratic [11], cubic, etc.) can be defined too. However there is no known efficient algorithm

to calculate them. In past years also the 2-adic complexity was studied with the connection to feedback with carry shift registers (FCSRs) [15], [41], [42]. However for most current designs of stream ciphers (or keystream generators) it is impossible to give useful estimates of the 2-adic complexity of their keystreams.

- linear complexity profile, that is only a little bit different from the ideal one, which is represented by the $n/2$ line. Moreover the differences should be irregular. If a linear complexity is constant on a large segment under the $n/2$ line it is possible to approximate the keystream (or a part of it) using a linear recurring relation of a smaller order. A nice example is the sequence $0, 0, \ldots, 0, 1$. Its linear complexity is maximal, i.e. equal to the number of bits in this sequence. However this sequence can be very well approximated by the all-zero sequence.

- proper statistical properties. As it was said a (pseudorandom) keystream should possess features common to truly random sequences. How random a keystream appears can be checked when statistical tests are applied on it (see Section 3.2 for further details).

A commonly used approach in the keystream generator evaluation is a combination of theoretic and experimental approach. The period and linear complexity (or their lower and upper bounds) of keystreams produced by the studied generator are determined using algebraic techniques. Some basic statistical properties such as number of ones, frequency of pairs of bits, number of runs of a given length, etc. in a period of a keystream are sometimes studied analytically, too. However these results do not express statistical properties of a keystream in a sufficient manner. Then a battery of statistical tests (see [45], [23], [24], [76]) is applied on a number of keystreams. (One should realize that it is not possible to test all the keystreams and moreover only a small, negligible part of the period of the keystream is tested.) Finally the resistance of the keystream generator against the known attacks is studied.

3.2 Statistical tests of pseudorandom sequences

Any sequence must possess common features expected in random sequences in order to be considered pseudorandom.

Golomb was the first one who formulated necessary (but not sufficient) conditions a pseudorandom sequence has to satisfy in order to be considered random.

Let $z^N = z_0, z_1, ..., z_{N-1}$ be a binary sequence (for a periodic sequence let N be its period). Golomb's randomness postulates state:

P1: The number of ones and zeros in z^N differs at most by 1.

P2: In the sequence z^N, one half of runs has length 1, one fourth has length 2, one eighth has length 3, etc. Moreover, the number of blocks and the number of gaps of a given length is roughly equal in z^N.

P3: Autocorrelation function $C(t)$ is two-valued, i.e. there exists such a number $K \in \mathbb{Z}$, that the following holds:

$$NC(t) = \sum_{i=0}^{N-1} (2z_i - 1)(2z_{i+t} - 1) = \begin{cases} N, & \text{if } t = 0 \\ K, & \text{if } 1 \leq t \leq N-1. \end{cases}$$

However, these postulates are too strict for a practical evaluation of a keystream generator.

Let H_0 be the hypothesis that the studied sequence possesses features common to a random sequence. Alternative hypothesis (the studied sequence does not possess features common to a random sequence), is denoted H_1. Which one of these hypotheses will be accepted we decide after performing a statistical test. If we reject H_0 although it was correct, we make the Type-I of error. Probability of this type of an error is referred to as the size of the test and is denoted as α. A typical choice of the test size is $0.001 \leq \alpha \leq 0.05$. If we accept H_0 although H_1 is valid, we make the Type-II of error. Probability of this error is denoted as β. The number $1-\beta$ is referred to as the power of the test. It is necessary to consider the length of the sequence during the choice of parameters α or β. It is important to realize that power of the test is more important than its size since acceptance of a "bad" sequence (or a "bad" keystream generator) can cause a security incident whereas rejection of a "good" sequence can cause "only" inefficiency.

There are various statistical tests (theoretical or empirical) to be applied to check whether the studied sequence possesses some feature common to a random sequence. Usually a two-sided test is applied. A large number of statistical tests can be found e.g. in [45], however not all of them are meaningful to be applied to binary sequences.

FIPS 140-2

This standard for nonclassified data suggests 4 statistical tests a sequence should pass to be considered pseudorandom. Required number of bits in a sample of a sequence is 20 000. FIPS 140-2 ([24]) is the successor of FIPS 140-1 ([23]). A very important change has been done: the test size was set to $\alpha = 10^{-4}$, whereas in FIPS 140-1 it was $\alpha = 10^{-6}$. These tests are based on the law of large numbers and the χ^2-test.

1. **Monobit test**
   The number of occurences of element 1 in the sample (denoted as n_1) should pass the inequality $9\,725 < n_1 < 10\,275$.
2. **The poker test (equidistribution of quadruples)**
   The studied sample is divided into 5 000 consecutive non-overlapping quadruples. The number of occurences of individual quadruples, denoted as $o_i, i = 0, \ldots, 15$, is determined and the following value is calculated.

$$V = \frac{16}{5\,000} \sum_{i=0}^{15} o_i^2 - 5\,000.$$

   A sample passes this test if it holds that $2.16 < V < 46.17$.
3. **Run test**
   Let us denote B_i, resp. G_i the number of blocks, resp. gaps of length i in the sample. (Runs longer than 6 are counted together with runs of length 6.) The values B_i, resp. G_i should occur in intervals specified in the following table.

Length of run	Interval
1	2 343 – 2 657
2	1 135 – 1 365
3	542 – 708
4	251 – 373
5	111 – 201
6	111 – 201

Table 3.1: FIPS 140-2: Required number of runs

4. **Long run test**
   A sample passes this test if it does not contain runs of length 26 or longer.

Probably the only publication dealing with statistical tests (and containing a mathematical background) of pseudorandom sequences used in cryptography is [76] (see also corrections in [40]). There are 16 tests described (mathematical background as well as the implementation) in this book. Source codes for Linux are to be found at `http://csrc.nist.gov/rng/rng2.html`. Other software packages for statistical testing of pseudorandom sequences are e.g. DIEHARD or Crypt-X.

3.3 Design of stream ciphers

There are four different approaches to the design of stream ciphers according to Rueppel [74], [75]:

System-theoretic. A new keystream generator is designed using the best known design principles. Next the accomplishment of basic criteria (period, linear complexity, statistical properties, etc.) on keystream generators is checked. Finally, it is studied whether the new keystream generator is a difficult and unknown problem (from the point of view of possible attacks) for an attacker.

Information-theoretic. The idea is to keep the cryptanalyst in the dark about the plaintext. No matter how much work the cryptanalyst invests, he will never get a unique solution.

Complexity-theoretic. The goal is to base the cryptosystem on, or make it equivalent to, some known and difficult problem such as factoring or calculating discrete logarithms.

Randomized. The keystream generator designed using this approach is based on a large number of transformations. The idea is to force the cryptanalyst to examine lots of useless data in his attempts at cryptanalysis. (Note: Complexity and a number of transformations do not guarantee security. An example of a "super-random" generator can be found in [45, p.4, Algorithm K].)

The most commonly used design approach is the system-theoretic one. Almost all the designs based on linear feedback shift registers rank among this category (for further details see this Section hereafter). However, there are also several stream ciphers designed according to the complexity-theoretic approach (e.g. BBS).

Next we introduce several basic building blocks of stream ciphers. Several designs of stream ciphers are included as examples.

Linear congruential generator

One of the common ways of producing pseudorandom sequences (mostly used in modelling and simulation) is the linear congruential generator [45]. It is based on the recurring relation

$$z_i = (a.z_{i-1} + b) \bmod m,$$

where a, b, m are chosen constants (integers), an integer $z_0 \in \{0, 1, \ldots, (m-1)\}$ is called the seed, or the initial loading of the generator (secret key). A proper choice of the constants ensures the maximal period of the produced keystream (e.g. when b and m are relatively prime). It is well known (see e.g. [45]) that linear congruential generator passes many standard statistical tests. However, there are also tests [9], [47] that this generator fails.

The output of the linear congruential generator is easily predictable, thus it is not suitable for cryptographic applications.

It was suggested to extract a given number of most significant bits from each item of the sequence z. However, this transformation does not improve the security of this generator significantly [46].

It is possible to generalize the concept of a linear congruential generator to a polynomial one. Also these generators are not considered secure [47].

Linear feedback shift register (LFSR)

Linear feedback shift register L (see Fig. 3.1) represents a technical (hardware) implementation of a solution of a linear recurring relation (3.1). It is a common basic building block of keystream generators.

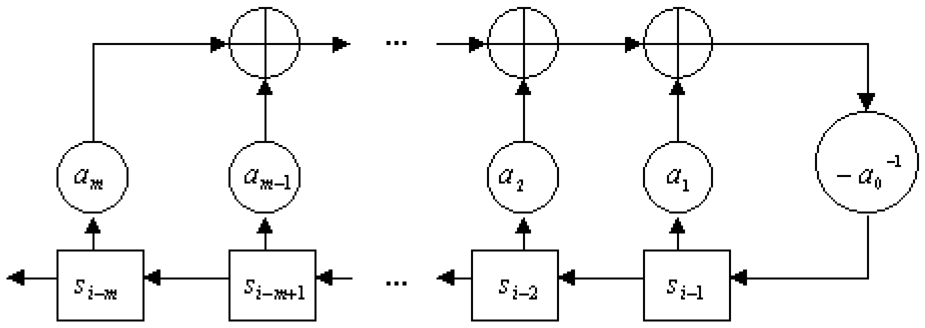

Figure 3.1: Linear feedback shift register

Below we show several possible representations of elements of a linear recurring sequence. Linear recurring sequences are studied e.g. in [38], [48], [57].

Theorem 3.3.1 *[57] Let $s_0, s_1, s_2, ...$ be a solution of the linear recurring relation (3.1) in the field $GF(q)$ with a left characteristic polynomial $a(x)$. If $\alpha_1, \alpha_2, \ldots, \alpha_m$ are mutually different roots of the polynomial $a(x)$, then $s_i = \sum_{j=1}^{m} \beta_j \alpha_j^i$, for $i = 0, 1, 2, \ldots$, where $\beta_1, \beta_2, \ldots, \beta_m$ are the elements of the extension field to the $GF(q)$ for the $a(x)$ polynomial, that are uniquely determined by the initial values $s_0, s_1, \ldots, s_{m-1}$.*

Definition 3.3.2 *[57] Let α be an element from $F = GF(q^m)$, that is an m-th order extension to the field $K = GF(q)$. Then the trace of the element α is defined as $Tr_{F/K}(\alpha) = \alpha + \alpha^q + \alpha^{q^2} + \cdots + \alpha^{q^{m-1}}$.*

The trace $Tr_{F/K}(\alpha)$ is a linear function from the field F onto the field K, provided both fields are understood as a linear (vector) spaces over the field K.

Theorem 3.3.3 *[57] Let $s_0, s_1, s_2, \ldots$ be a solution to the linear recurring relation (3.1) in $K = GF(q)$ with a left characteristic polynomial $a(x)$, that is irreducible over K. Let α be a root of $a(x)$ in the extension field $F = GF(q^m)$ of the field K. Then $s_i = Tr_{F/K}(\theta\alpha^i)$, for $i = 0, 1, 2, \ldots,$ where θ is a uniquely determined element from F.*

This representation of elements of linear recurring relations is of particular importance for the analysis of combination of several linear recurring relations.

As it was said before, the period of a keystream is a very important feature. Thus it is natural to ask how to construct an LFSR to obtain the maximal possible period of the produced keystream. It is quite easy to see that an m-bit long LFSR can produce only sequences with period at most $2^m - 1$.

Definition 3.3.4 *[4] A polynomial $a(x) \in GF(2)[X]$, $\deg a(x) = m$ is said to be primitive, if it holds that $a(x)$ divides $x^{2^m-1} - 1$ and does not divide any polynomial $x^t - 1$, where $t < 2^m - 1$.*

Theorem 3.3.5 *[4, p.350, Theorem 9] If the left characteristic polynomial $a(x) \in GF(2)[X]$, $\deg a(x) = m$ associated with an LFSR is primitive, then any sequence this LFSR produces from a non-zero inital loading has period $2^m - 1$. An LFSR with associated primitive polynomial is also called ml-LFSR and sequences it produces are called ml-sequences.*

ml-sequences are well known due to their nice statistical properties and they also pass the Golomb randomness postulates (see Section 3.2). The distribution of patterns in an ml-sequence is expressed in the following theorem.

Theorem 3.3.6 *[54, p.197,Fact 6.14] Let u be an ml-sequence generated by an ml-LFSR L, that is $\|L\|$ bits long. Let k be an integer, $1 \leq k \leq \|L\|$, and let $\bar{u}$ be any subsequence of u of length $2^{\|L\|}+k-2$. Then each non-zero sequence of length k appers exactly $2^{\|L\|-k}$ times as a subsequence of $\bar{u}$. Furthermore, the zero sequence of length k appears exactly $2^{\|L\|-k} - 1$ times as a subsequence of $\bar{u}$.*

It is also possible to associate a formal power series $s(x) = s_0 + s_1x + s_2x^2 + \dots$ to a binary sequence $s = s_0, s_1, s_2, \dots$ as its generating function. According to [4, p.340, Consequence of the Theorem 5] each solution to the linear recurring relation (3.1) has in $GF(2)[X]$ a generating function $s(x) = \frac{h(x)}{a(x)}$, $h(x), a(x) \in GF(2)[X]$, $\deg h(x) < \deg a(x)$ and $a(x)$ is the left characteristic polynomial of the linear recurring relation.

An LFSR cannot be used itself as a keystream generator. Its security weakness is the small linear complexity of the produced keystream.

J.L.Massey in [51] proved that Berlekamp algorithm for decoding BCH codes is a general solution to the problem of synthesis of the shortest LFSR that generates a given sequence. Let $s = s_0, s_1, s_2, \dots$ be a binary sequence and m be its linear complexity. Then it is possible to find uniquely the shortest LFSR that generates the sequence s using the Berlekamp-Massey algorithm from $2m$ consecutive bits of the sequence s. The complexity of this algorithm is roughly the square of the sequence length.

We will describe several modifications of an LFSR that enlarge the linear complexity of a produced sequence. These designs can be found in a large number of classical stream ciphers.

Filtered LFSR

Linear feedback shift register is in fact a finite automat. One of its possible modifications is to change the output function. Whereas the output of a "classical" LFSR is a single bit of its loading in the given time, the output of a filtered LFSR is a value of a Boolean function that takes as input all the bits of the LFSR loading in the given time. Properties of filtered LFSRs are studied in [72].

Let a sequence $s_0, s_1, s_2, ...$ be a solution to a linear recurring relation (3.1) in $GF(2)$ with a left characteristic polynomial $a(x)$. Further let $n \leq m$ of the LFSR bits be an input to a filter function $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$. The filter function must be nonlinear, otherwise it makes no sense to use it. Let $0 \leq j_1 < j_2 < \dots j_n < m$ be the indices of these bits. The output of this filtered LFSR in time i is $z_i = f(s_{i+j_1}, s_{i+j_2}, \dots, s_{i+j_n})$.

The next two Theorems characterize the important properties - period and linear complexity of a filtered LFSR.

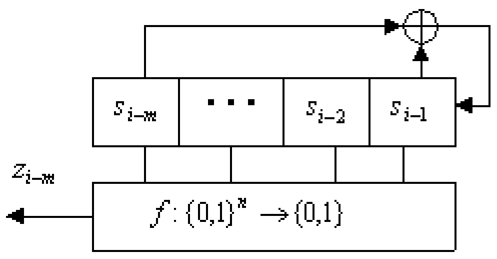

Figure 3.2: Filtered LFSR

Theorem 3.3.7 *[82] Let $a(x)$ be a left characteristic polynomial of a filtered LFSR with a non-zero initial loading. If the filter function f is balanced or ($2^{\deg a(x)} - 1$ is a prime number and f is not a constant function), then the period of this filtered LFSR is $2^{\deg a(x)} - 1$.*

Theorem 3.3.8 *[72] Let $a(x)$ be a primitive left characteristic polynomial of a filtered LFSR. Let $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$ be the filter function. Then the linear complexity of the output sequence z, denoted as $\Lambda(z)$, can be upperbounded by:*

$$\Lambda(z) \leq \sum_{i=1}^{n} \binom{\deg a(x)}{i}.$$

Clock-controlled LFSR

Let a sequence $s = s_0, s_1, s_2, ...$ be a solution to a linear recurring relation (3.1) in the field $GF(2)$ with a left characteristic polynomial $a(x)$. Let $z = z_0, z_1, z_2, ...$ be the output sequence from a clock-controlled LFSR.

One of the methods for increasing the linear complexity of a sequence z is to change the clocking of the register. Note, that in general it will be $s_i \neq z_i$. The change of the clocking of an LFSR also increases the resistance against the so-called correlation attacks (see Section 3.5 for further details).

The clock control of an LFSR can be in general understood as a selection of elements from its output sequence s driven by the so-called (periodic) decimation sequence $d = d_0, d_1, d_2, \ldots$, $0 \leq d_i \leq 2^{\deg a(x)} - 2$, $i = 0, 1, 2, \ldots$,

$$z_0 = s_0,$$
$$z_i = s_{\sum_{j=0}^{i-1} d_j}, \qquad i = 1, 2, \ldots.$$

LFSR with constant clocking

The simplest case is the selection of each t-th element, i.e. $z_i = s_{it}$. The next Theorem characterizes the important properties of an LFSR with a constant clocking, namely its period, and linear complexity.

Theorem 3.3.9 *[72] Let s be a linear recurring sequence over $GF(q)$ with period T and characteristic polynomial $a(x)$, $\deg a(x) = m$, that is irreducible over $GF(q)$. Let α be a root of $a(x)$. Then the sequence $z_i = s_{it}$ has the following properties:*

- *its characteristic polynomial is the minimal polynomial for the element α^t,*
- *its period $T^* = \frac{T}{\gcd(t,T)}$,*
- *its linear complexity is equal to the multiplicative order of q in Z_{T^*}.*

Moreover for all $t \in \{k, kq, kq^2, \ldots \pmod{T}\}$ is the output sequence z the same for a proper choice of the initial loading.

Self clock-controlled LFSR

Rueppel in [73] proposed a linear feedback shift register which is self clock-controlled.

Let a sequence $s = s_0, s_1, s_2, ...$ be a solution to a linear recurring relation (3.1) in $GF(2)$ with $a(x)$ being the left characteristic polynomial. Let $z = z_0, z_1, z_2, \ldots$ be the output sequence from the self clock-controlled LFSR. The clock-control works as follows:

- $z_0 = s_0$,
- if $z_i = 0$ then the register is clocked l-times (in other words if $z_i = s_j$ then $z_{i+1} = s_{j+l}$),
- if $z_i = 1$ then the register is clocked k-times,

where $l, k \in \mathbb{N}$ are arbitrarily chosen constants. The output sequence z is also called an $[l, k]$ self-decimated sequence.

If the polynomial $a(x)$ is primitive and the constants l, k are properly chosen, it is possible for the sequence z to achieve uniform distribution of patterns and also a maximal period equal to $\lfloor (2/3) * (2^{\deg a(x)} - 1) \rfloor$. Performed exhaustive search experiments show also a large linear complexity and an almost flat autocorrelation function.

If the constants l and k are known to the attacker, it is possible to find the initial loading of the register from the sequence z – it suffices to solve a system of linear equations. Thus a self clock-controlled LFSR is not resistant against the known plaintext attack (see Section 3.5). Hence a self clock-controlled LFSR should not be itself a keystream generator. Anyway it can be used as a building block for keystream generator.

The clock-control of an LFSR using the output of another LFSR is studied in [3]. The generalization of this idea, the so-called cascade of LFSRs, is studied in [29].

Other registers

Up to now we have discussed a linear feedback shift register and several of its modifications. It is also possible to build a shift register with a nonlinear feedback. However they are not favoured as building blocks for keystream generators. The main reason is the insufficient knowledge about their properties. On the other hand the not well developed mathematical methods concerning analysis of nonlinear feedback shift registers make the cryptanalysis much harder. However the designers of keystream generators favour well known and analyzed building blocks.

Feedback with carry shift register (FCSR) is another kind of register [41], [42], [15]. It is based on the theory of 2-adic numbers. It is known that for any periodic sequence there exists an FCSR that generates it. The mutual relation between LFSR and FCSR is unknown up to now [15]. An upper bound on the period of sequences produced by an FCSR has been determined. It is also known how to choose the parameters of an FCSR to obtain sequences with maximal period. The problem is that a sequence with maximal period is not obtained for all the initial loadings of an FCSR (with properly chosen parameters to obtain sequences with maximal period) [78], [15].

The only published keystream generators (known to me) based on FCSRs can be found in [78]. These generators are simple modifications of the well known keystream generators (usually one or more LFSRs are substituted by FCSRs). However their security is often an open problem. Cryptanalysis of two such designs - the parity and the threshold keystream generators was done in [84], [69].

Combination of LFSRs

One of the classical methods of building a keystream generator is to use several LFSRs and to combine their outputs using a (non-linear) combination function. The most simple case is the XOR of two linear recurring sequences. We define also a special operation - the bitwise multiplication of sequences.

Definition 3.3.10 *[57] Let $s_0, s_1, s_2, \dots$ and $s'_0, s'_1, s'_2, \dots$ be two linear recurring sequences over $GF(2)$ with associated left characteristic polynomials $a(x)$ and $a'(x)$. We define the sum of these sequences $s + s'$ to be the sequence $s_0 + s'_0, s_1 + s'_1, s_2 + s'_2, \dots,$ where $+$ denotes the addition in $GF(2)$ (in other words XOR). Next we define the (bitwise) product of these sequences $s.s'$ to be the sequence $s_0.s'_0, s_1.s'_1, s_2.s'_2, \dots,$ where $.$ denotes the multiplication in $GF(2)$ (in other words AND).*

The set of all linear recurring sequences, generated by a primitive polynomial $a(x)$, is closed with respect to the sum [4, p.351]. Moreover it is closed also with respect to the shift and scalar multiplication.

Linear complexity of a sequence produced as a polynomial combination of several (special) linear recurring sequences is determined in the following Theorem.

Theorem 3.3.11 *[57] Let F be a nonlinear function over $GF(2)$,*

$$F(s^{(1)}, s^{(2)}, \dots, s^{(N)}) = \\ = u_0 + \sum u_i s^{(i)} + \sum u_{ij} s^{(i)} s^{(j)} + \dots + u_{12\dots N} s^{(1)} s^{(2)} \dots s^{(N)},$$

where $u_i, u_{ij}, \dots, u_{12\dots N} \in GF(2)$. Let $s^{(1)}, s^{(2)}, \dots, s^{(N)}$ be linear recurring sequences over $GF(2)$ with minimal polynomials $m_{s^{(i)}}(x)$,

$\deg m_{s^{(i)}}(x) = M_i$. *Let us assume that each polynomial* $m_{s^{(i)}}$ *has only simple roots in* $GF(2^{M_i}) \setminus GF(2)$ *and none of the roots is a scalar multiple of another root. Let us further assume that the degrees of the minimal polynomials are pairwise relatively prime. Then a sequence* $z = F(s^{(1)}, s^{(2)}, \ldots, s^{(N)})$ *has a minimal polynomial* $m_z(x)$ *of degree* $M = F'(M_1, M_2, \ldots, M_N)$. *The polynomial* F' *is given by the same expression as the polynomial* F, *with (integer) coefficients* $u'_i, u'_{ij}, \ldots, u'_{12\ldots N}$, *that are equal to 0 or 1, respectively whenever* $u_i, u_{ij}, \ldots, u_{12\ldots N}$ *are equal or non-equal to zero, respectively. The polynomial* F' *is evaluated in integers and not in the field* $GF(2)$. *All the roots of the minimal polynomial* $m_z(x)$ *are simple and are from* $GF(2^m) \setminus GF(2)$, *where* $m = \prod_{i=1}^{N} M_i$.

Besides the nonlinearity of a combination function F (see e.g. [72], [57]), an important role plays also its correlation immunity (see e.g. [72], [79]). The correlation immunity characterizes the relation between the output of a function and its inputs.

Definition 3.3.12 *[72], [30] A function* $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$ *is said to be* k*-th order correlation immune,* $1 \leq k < n$, *if* $I(f(X), Y) = 0$ *for each* k*-dimensional subvector* Y *of a vector* X, *where* $I(U, V) = H(U) - H(U/V)$ *is the so-called amount of mutual information between* U *and* V.

The relation among the order of correlation immunity, algebraic order and number of variables of a Boolean function is determined in the next Theorem.

Theorem 3.3.13 *[79] If a function* $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$ *is* k*-th order correlation immune,* $1 \leq k < n$, *then each term in its algebraic normal form must have less than* $n - k + 1$ *variables.*

Construction of sequences with a given correlation immunity is studied in [79].

NSG: natural sequence generator

A special class of keystream generators - the so-called natural sequence generators (NSGs) is studied in [15]. From a design point of view there is an obvious similarity with a filtered LFSR. NSG is based on a counter and on an output (filter) function. The i-th keystream bit z_i is produced according to the rule $z_i = f(i + k \bmod N)$, where $f : \mathbb{Z}_N \to \mathbb{Z}_2$ is the output function, N is the period of the keystream and $k \in \mathbb{Z}_N$ is a secret key.

A quite special design of an NSG allows the application of many number theoretic results. It can be easily seen that the output function plays an important role in the security of the NSG. There are several constructions of the output function studied in [15]. It is important that under the proper choice of the NSG parameters one may obtain a keystream generator with high nonlinearity, linear, weight and sphere complexity. It is also resistant against the differential cryptanalysis. It is interesting that one of the parameters is the period of the keystream, which is usually chosen to be a prime number (sometimes a special form is required).

Although NSGs may possess many nice properties, they are very slow (about 4 kB per second) both in hardware and software. It is mostly due to the operations that NSGs use: multiplication and exponentiation are the common ones. Thus the practical usage of NSGs is rather restricted. One of the possible areas is the key generation.

Other designs

Some recent stream ciphers have been designed for efficient software implementation, and are not based on LFSRs [62]. Examples include the stream ciphers RC4, SEAL, Scream and the NESSIE submission LEVIATHAN. These ciphers are build upon the block cipher design ideas. In some cases, they are in fact a block cipher in a special mode of operation (e.g. the BMGL stream cipher, which is one of the submissions to NESSIE).

3.4 Examples of stream ciphers

There are several chosen stream ciphers presented in this Section.

Geffe's generator

Geffe's generator is based on a polynomial combination of outputs from three LFSRs $L1$, $L2$ and $L3$. Let $Li(t)$, $i = 1, 2, 3$ be the output from the LFSR Li in time t. The output of the generator in time t is then $z_t = (L1(t) \oplus 1).L2(t) \oplus L1(t).L3(t)$. The period of the keystream is the least common multiple of periods of sequences produced by the individual LFSRs. The linear complexity of the keystream (under a proper choice of LFSR polynomials) can be determined due to the Theorem 3.3.11. Note that $Pr(z_t = L2(t)) = Pr(z_t = L3(t)) = 0.75$, thus a correlation attack is possible to be performed. Hence the Geffe's generator is insecure. However it is a nice design for demonstrating divide-and-conquer and correlation attacks (see Section 3.5).

Generator LILI–128

LILI–128 [17] is one of the keystream generators submitted to the NESSIE project. Its design combines two principles - filter function and clock control. LILI–128 consists of two nonlinearly filtered LFSRs. The output the first register controls the clocking of the second register. The output of the keystream generator is the output from the second register. The produced keystream has a large period and also a large linear complexity. However there is an attack faster than exhaustive search against this generator and that was the reason why it was not included in the NESSIE portfolio of recommended stream ciphers.

BBS: Blum, Blum, Shub generator

BBS is a "number-theoretic" keystream generator designed according to the complexity-theoretic approach [7]. Its security is based on the problem of factoring integers. An important fact is that the keystream is unpredictable (neither to the left, nor to the right).

Algorithm of the BBS keystream generator:

1. Choose primes p and q such that $p \neq q$, $p \equiv 3 \bmod 4$, $q \equiv 3 \bmod 4$. Calculate $n = pq$.
2. Choose a random $w \in [1, n-1]$, such that $\gcd(w, n) = 1$. Calculate $x_0 \leftarrow w^2 \bmod n$.
3. Keystream $z_1, z_2, \ldots, z_N$ is produced as follows:
   $x_i \leftarrow x_{i-1}^2 \bmod n$.
   $z_i \leftarrow \mathrm{LSB}(x_i)$ for $i = 1, 2, \ldots, N$.
   $\mathrm{LSB}(x_i)$ is the least significant bit of a binary representation of x_i.

Due to the operations the BBS uses, it ranks among the slow generators. Thus its practical usage as a keystream generator is limited to applications, where the encryption speed does not play much a role or which require high security. For example, it can be used as a generator of random values (e.g. keys) for other cryptographic primitives.

A large number of keystream generators can be found e.g. in [78], [54].

3.5 Attacks on stream ciphers

The natural requirement on any cryptosystem is its security. There are also other important properties of cryptosystems, such as e.g. error propagation, key size, etc. (see Kerckhoffs's and Shannon's requirements on cryptosystems in [32, pp.40–41]).

A well known requirement, formulated by Kerckhoffs in the 19-th century, is that the security of any cryptosystem should be based on keeping the key secret and not on keeping the entire cryptosystem secret.

The famous notions of perfect and relative secrecy of cryptosystems were introduced by Shannon. A perfectly secure cryptosystem must have the following property: the amount of information about the plaintext and the key does not increase when a new ciphertext is obtained.

More formally, $H(P, K/C) = H(P, K)$, where $P \in \mathcal{P}$, $C \in \mathcal{C}$, $K \in \mathcal{K}$ and H is the entropy function (see e.g. [32, pp.81–82]). Assuming the mutual independency between the key and the plaintext one can write $H(P, K) = H(P) + H(K)$. Cryptosystems that are not perfectly secure can be only relatively secure.

A cryptosystem is said to be computationally secure, if it is not possible to perform an attack against it in a real time with really available computing power (number of processors, amount of memory and disc storage, etc.).

Moreover, Shannon introduced also the notion of ideal secrecy [1]. When the unicity distance is infinite, one speaks about ideal secrecy. The unicity distance determines the amount of a ciphertext needed

[1] An ideally secure cryptosystem does not have to be perfectly secure!

for finding the corresponding plaintext uniquely (see [78, pp.235–236]).

Basic assumptions for cryptanalysis were formulated by Kerckhoffs in the 19-th century. According to them a cryptanalyst has detailed knowledge about the cryptosystem, including all the details about the algorithm and also about its technical implementation. If the cryptanalyst, having this knowledge, is not able to break the cryptosystem, it is reasonable to assume that an attacker will not be able to break this cryptosystem without this knowledge.

The goal of an attack on a cryptosystem might be to find the decryption key, the plaintext, etc. Knudsen classified the following four categories of breaking an algorithm [43]:

total break. An attacker finds the decryption key.

global deduction. An attacker finds an algorithm for decrypting the ciphertext without knowing the decryption key.

instance (or local) deduction. An attacker finds the plaintext of an intercepted ciphertext.

information deduction. An attacker gains some information about the key or plaintext.

Attacks can be further subdivided into the following categories according the amount of information an attacker has:

ciphertext-only attack. An attacker knows only a ciphertext $C \in \mathcal{C}$.

known plaintext attack. An attacker is given a plaintext $P \in \mathcal{P}$ and the corresponding ciphertext $C \in \mathcal{C}$, $C = e_{k_e}(P)$ or several plaintext-ciphertext pairs.

chosen plaintext attack. An attacker can choose a set of plaintexts and obtain the corresponding ciphertexts.

Further details concerning the above mentioned attacks, including some other attacks, can be found in [78, pp.5–7].

It cannot be said that there is a standard set of attacks on a stream cipher as it is for a block cipher (differential and linear cryptanalysis, related keys attack, etc.). Neither there are no such important results relating the security of a stream cipher to some of its construction parameters as there is for a block cipher (e.g. the relation between the number of rounds and resistance against differential cryptanalysis or the notion of provable security against the differential and linear cryptanalysis, see e.g. the block cipher MISTY and the papers by M.Matsui). The attacks against stream ciphers usually exploit some specific weakness of the design.

In the following, we try to point out some general and most common attacks against stream ciphers.

Brute-force attack

Any relatively secure cryptosystem is vulnerable to a brute-force attack (sometimes also called exhaustive search). The idea behind this attack is simply to try all the keys. For each key a ciphertext is decrypted and the obtained plaintext is checked whether it is the "right" one. Usually some assumptions about the plaintext must be done, e.g. a language it is written in is known, which enables to search for words, or possibly some structure a plaintext message has is known, which might be the case of database records. Thus it is natural to require any cryptosystem to have a large keyspace and a flat probability distribution.

Divide-and-conquer attack

The idea of a divide-and-conquer attack is to divide the key into parts (not necessarily disjoint), perform attacks to gain these parts, put them together and conquer. Assume a keystream generator that consists of several LFSRs. Let the initial loadings of these LFSRs be the key. The divide-and-conquer strategy is to perform attacks against the individual LFSRs (see e.g. correlation attacks in this Section hereafter).

This attack is usually combined with other attacks or techniques. One of them is a guess-and-check technique. It is very common in the context of known (or chosen) plaintext attack. The idea is to divide the key into parts, choose some of them, calculate the remaining parts of the key and finally check the key, e.g. whether the obtained key produces the keystream an attacker has. A practical example of this approach is the attack on the stream cipher ORYX [94].

Time-memory tradeoff attack

The time-memory tradeoff on a stream cipher is in general an adaptation of the time-memory tradeoff developed by Hellman for block ciphers. Such attacks can be applied to almost any cryptosystem, but they are feasible only when the number of internal states is relatively small. The basic idea of the time-memory tradeoff is to keep a large set A of precomputed states on a hard disk, and to consider the large set B of states through which the algorithm progresses during the actual generation of output bits. Any intersection between A and B will enable us to identify an actual state of the algorithm from the stored information [5].

Time-memory tradeoff is one of the basic ideas of attacks [5], [6], [28] on the stream cipher A5/1 which is used in GSM.

Correlation attacks

Correlation attacks are studied mostly in connection to LFSRs. Let us have a keystream generator, which contains an LFSR as one of its building blocks. Let $s = s_0, s_1, \dots, s_{N-1}$ be the (unknown) output of this LFSR. Assume, the keystream bits $z = z_0, z_1, \dots, z_{N-1}$ and the value $p = Pr(z_i = s_i) > 1/2$ are known (a textbook example is the Geffe's generator, see Section 3.3). The idea behind the correlation attacks is to exploit the coincidence between the known keystream and the unknown output of the LFSR to find the initial loading of this LFSR.

Siegenthaler's correlation attack

The first published correlation attack [80] is based on an exhaustive search. The output of an attacked LFSR is produced for each of its possible initial loadings. The real coincidence between the known keystream and each of the produced output sequences from the LFSR is determined. The initial loading for which the difference between the real and theoretical coincidence is the smallest one, is the best candidate to be the key. This attack is infeasible when a sufficiently long LFSR (nowadays about 80 bits) is used.

Meier's-Staffelbach's fast correlation attacks

Meier's-Staffelbach's fast correlation attacks (algorithms A and B) [53] rank among the first fast correlation attacks that are not based on an exhaustive search in the keyspace. (Algorithm A is mentioned also in [30], where the description of several other attacks can be found.)

Recall that the sequence s is a solution to a linear recurring sequence given by a left characteristic polynomial $a(x)$, $\deg a(x) = m$. Let $(t+1)$ be the number of non-zero coefficients of $a(x)$. Each bit s_i of the sequence s (except several bits at the borders) may be written in the linear recurring relation at $t+1$ positions. In other words, $t+1$ relations may be written for each bit of s using the linear recurring relation. Recall that $a(x)^j = a(x^j)$, $i = 0, 1, 2, \dots$ holds for $j = 2^i$ (in fields of characteristic 2). This yields another relations. All these relations are also called the parity checks. Let us substitute the individual bits from the seqeunce s in parity checks by the corresponding bits from z. Not necessarily all of them will hold (or will be valid). This leads us to the idea of an iterative correlation attack, which can be informally described as follows:

1. find a set of (linearly independent) parity checks for each bit in the sequence z,
2. according to the number of parity checks that hold, decide whether $s_i = z_i$ or $s_i \neq z_i$,
3. if all the parity checks hold, stop the algorithm,
4. alter the bits in the sequence z for which the decision was $s_i \neq z_i$ and go to the step 2.

For further analysis we introduce a statistical model based on the set of (linearly independent) parity checks [53]

$$S \oplus B_{i1} \oplus B_{i2} \oplus \cdots \oplus B_{it} = 0, \quad i = 1, 2, \dots, o,$$

where S is a random variable corresponding to s_n, B_{ij} are random variables corresponding to those bits from s that appear in the i-th parity check for s_n. The average number of parity checks o for one bit will be discussed later.

Similarly for the keystream sequence:

$$L_i = Z \oplus Y_{i1} \oplus Y_{i2} \oplus \cdots \oplus Y_{it}, \quad i = 1, 2, \dots, o.$$

Let us assume that the used random variables are mutually independent and uniformly distributed. It follows that

$$\begin{aligned} Pr(Z = S) = Pr(B_{ij} = Y_{ij}) = p, \\ Pr(\bigoplus_{j=1}^{t} Y_{ij} = \bigoplus_{j=1}^{t} B_{ij}) = f(p,t), \\ f(p,t) = pf(p,t-1) + (1-p)(1-f(p,t-1)), \\ f(p,1) = p. \end{aligned}$$

Sometimes for simplicity we write only f instead of $f(p,t)$.

The probability that $z_i = s_i$ provided h of o parity checks hold (this condition is here denoted as *cond.*) is then

$$\begin{aligned} p^* = Pr(z_i = s_i/cond.) = \\ = \frac{pf(p,t)^h(1-f(p,t))^{o-h}}{pf(p,t)^h(1-f(p,t))^{o-h}+(1-p)(1-f(p,t))^h f(p,t)^{o-h}}. \end{aligned}$$

An average number of parity checks o for a single bit is

$$o = o(N,m,t) \approx \log_2(N/(2m))(t+1).$$

The probability that at least h of o parity checks hold for a chosen bit z_i is given by the following relation:

$$Q(p,o,h) = \sum_{i=h}^{o} \binom{o}{i} pf^i(1-f)^{o-i} + (1-p)(1-f)^i f^{o-i}.$$

Algorithm B

This algorithm is based on an iterative modification of the keystream sequence z, which yields the sequence s. The convergence of this process as well as the correctness of the solution is not guaranteed! However this attack works usually fine even when there are some linear dependencies in the set of the parity checks [10].

The probability that at most h of o parity checks hold for z_i is

$$U(p,o,h) = \sum_{i=0}^{h} \binom{o}{i} pf^i(1-f)^{o-i} + (1-p)(1-f)^i f^{o-i}.$$

Thus $U(p,o,h)N$ is then the average number of bits that will be altered.

The probability that $z_i = s_i$ and at most h of o parity checks hold for z_i is

$$V(p,o,h) = \sum_{i=0}^{h} \begin{pmatrix} o \\ i \end{pmatrix} pf^i (1-f)^{o-i}$$

and the probability that $z_i \neq s_i$ and at most h of o parity checks hold for z_i is

$$W(p,o,h) = \sum_{i=0}^{h} \begin{pmatrix} o \\ i \end{pmatrix} (1-p)(1-f)^i f^{o-i}.$$

$I(p,o,h)N = (W(p,o,h) - V(p,o,h))N$ is then the increase of correctly altered bits.

The algorithm works as follows:

1. Calculate o and find the value h (denoted as h_{max}) for which $I(p,o,h)$ is maximal.
2. Calculate the threshold probability $p_{thr} = (1/2)(p^*(p,o,h_{max}) + p^*(p,o,h_{max}+1))$ and the expected number of bits for which their $p^* < p_{thr}$ using the relation $N_{thr} = U(p,o,h_{max})N$. Choose γ - the maximal number of iterations in one round. (According to the performed experiments in [10], the best choice is $\gamma = 2$.)
3. Set the iteration counter; $I = 0$.
4. Calculate a new p^* for each bit of the analyzed sequence z. (These probabilities are stored and used in iterations when $I > 0$ in the relations for $f(p,t)$ which will be changed into $f(p_1^*, p_2^*, \ldots, p_t^*, t)$, see [53] for details.)
5. The algorithm terminates if all the parity checks hold. The obtained modified sequence z is the output from the attacked LFSR and its starting bits form its initial loading.
6. Determine N_w which is the number of bits in z with their $p^* < p_{thr}$. If ($N_w \geq N_{thr}$ or $I = \gamma$) then alter those bits z_i, for which their $p^* < p_{thr}$, forget the stored values p^* for all the bits in z (i.e. assume again that $Pr(z_i = s_i) = p$) and go to the step 3.
7. $I = I + 1$. Go to the step 4.

This attack is feasible when a sparse characteristic polynomial is used, i.e. $t \leq 10$. The length of the LFSR can be up to 1 000. (The largest succesfully attacked LFSR known to me was 9 689 bits long, $t = 2$, $p = 0.75$, $N = 700\,000$, see [10] for details.) Algorithm B works also when p is close to $1/2$, e.g. for values 0.55.

The following parameters influence the success of this algorithm: p, t, N/m. The estimates of complexity of this algorithm based on the values of the above mentioned parameters can be found in [53].

Comparison of chosen iterative correlation attacks

A comparison of several iterative correlation attacks based on a number of experimental attacks can be found in [56]. A large number of experimental results on various iterative correlation attacks can be found also in [10]. These attacks can be subdivided into the following three categories:

1. alternation of bits in the keystream z is based on the number of satisfied parity checks (e.g. linear syndrome attack [95], see also an improved version in [96]),
2. alternation of bits in the keystream z is based on the estimatation of the relevant àposterior probabilities (i.e. $Pr(z_i = s_i/cond.)$) obtained by using the average àposterior probability estimated in the previous iteration as the prior probability (i.e. $Pr(z_i = s_i)$) in the current iteration (e.g. simplified algorithm from [55]),
3. alternation of bits in the keystream z is based on the estimatation of the relevant àposterior probabilities (i.e. $Pr(z_i = s_i/cond.)$) obtained by using the àposterior probabilities estimated in the previous iteration as the prior probabilities (i.e. $Pr(z_i = s_i)$) in the current iteration (e.g. simplified algorithm B from [53], see also this Section above).

Attacks based on these principles were tested on a 10 000-bits long keystream, produced by an LFSR with associated left characteristic polynomial $1 + x^5 + x^{47}$, probabilities $Pr(z_i = s_i)$ were $p_1 = 0.6$, $p_2 = 0.575$, $p_3 = 0.565$. Algorithm based on the 1st principle succeeded in reconstruction of the output from the LFSR only for p_1. Algorithm that worked according to the 2nd principle was successful both for p_1 and p_2. Finally, the algorithm based on the 3rd principle succeeded in reconstruction of the output from the LFSR for all studied probabilities.

Based on the results from the experimental analysis of iterative correlation attacks, the authors in [56] suggest to use algorithms, based on the 1st or on the 2nd principle, for high probabilities $Pr(z_i = s_i)$. The main reasons are the higher speed of the algorithms and lower implementation costs. The algorithms based on the 3rd principle are suitable for probabilities $Pr(z_i = s_i)$ close to 1/2. However, it was suggested to perform only first few rounds of this algorithm and then to use algorithms based on the 1st or on the 2nd principle. Such a cooperative attack strategy was experimentally studied in [10].

In order to attain resistance against the described correlation attacks, sparse characteristic polynomials should be avoided and the probability that $s_i = z_i$ should be as close as possible to 1/2.

The impact of parity checks used in iterative correlation attacks was studied in [12]. Most appropriate ones are the parity checks with a small number of elements, preferably 4 or 5.

New correlation attacks [35], [37], [36] are applicable even though a dense characteristic polynomial is used and the probability of coincidence is very close to 1/2.

A correlation attack against a nonlinearly filtered LFSR was studied in [81], [77]. The idea behind this attack is to construct an equivalent keystream generator that is in fact a polynomial combination of several identical LFSRs. The attack does not require the knowledge of the filter function. However it is based on an exhaustive search through the all initial loadings of the attacked LFSR, which makes it infeasible for a sufficiently large LFSR.

Other attacks

There is no known fast correlation attack on a clock-controlled LFSR. This case is in general very complicated. However there were obtained some results when the decimation sequence has some special properties, see e.g. [97] (the complexity of this attack is exponential with respect to the length of the LFSR).

The differential cryptanalysis [15] is applicable on stream ciphers, too. This attack was studied mostly in connection to natural sequence generators (see Section 3.3 or [15]).

In past years there appeared the so called distinguishing attacks (e.g. against the stream cipher SEAL). Their objective is to disprove the assumption about the randomness of the keystream by distinguishing this keystream from a truly random sequence. Recall that there are also generic distinguishing attacks on block ciphers in OFB and Counter Mode. For a block cipher with block size m, $2^{m/2}$ blocks of a keystream are sufficient to distinguish this keystream from a truly random sequence. This is achieved by looking for repeated occurrences of blocks, which are not possible when the stream is generated by a block cipher in OFB or Counter Mode (unless the sequence has started to repeat itself) [62].

3.6 Hash functions

Hash functions compress a string of an arbitrary length to a string of a fixed length. They have a large number of applications in computer science (optimized/fast access to the stored data), and as well in cryptography (integrity protection of stored/transmitted data). However hash functions used in cryptographic applications may need to meet stronger requirements than those for other applications. Probably the best publication that deals with hash functions is [61].

According to the number of inputs one can subdivide hash functions into two classes [63]:

- one input (a message to be hashed). These are called Manipulation Detection Codes (MDCs), sometimes also cryptographic hash functions or just only hash functions.
- two inputs (a message to be hashed and a key). If the key is kept secret one calls them Message Authentication Codes (MACs). If the key is public one calls them Universal One-Way Hash Functions (UOWHFs).

Next we give an informal basic definition of a hash function and of a collision resistant hash function.

Definition 3.6.1 *([63], informal) A one-way hash function is a function $h : \{0,1\}^* \to \{0,1\}^m$, where $\{0,1\}^*$ is the set of all finite binary strings and m is a given integer[2], satisfying the following conditions:*

1. *the hash function must be one way in the sense that given a y in the image of h, it is "hard" (i.e. computationally infeasible in a real time) to find a message x such that $h(x) = y$ (preimage resistant);*
2. *given an x in the domain of h and $h(x)$, it is "hard" to find a message $x' \neq x$, such that $h(x') = h(x)$ (second preimage resistant).*

Definition 3.6.2 *([63], informal) A collision resistant hash function is a one-way hash function $h : \{0,1\}^* \to \{0,1\}^m$ for which it is "hard" to find two distinct messages x, x', such that $h(x) = h(x')$.*

Clearly, both Definitions given above can be extended for any alphabet Q.

However, only a few known results follow from collision resistance [1]. One of the most important ones is that this property is preserved under chaining [16]. In order to prove some security results for practical systems, one is usually forced to use other definitions, e.g. such as Okamoto's correlation free one-way hash functions.

Definition 3.6.3 *([1], informal) A function $h : \{0,1\}^* \to \{0,1\}^m$ is correlation free, if it is computationally infeasible to find $X, Y \in \{0,1\}^*$, such that the Hamming weight of $h(X) \oplus h(Y)$ is less than one would expect to get from random chance if we calculated $h(M)$ for a lot of $M \in \{0,1\}^*$.*

Intuitively, this definition means that as well as having no collisions, we get no near misses either.

3.7 Design of hash functions

Most known hash functions are based on a compression function with fixed size inputs [63]. Computation of the hash value can be described as follows:

[2]Nowadays commonly used values for m are 128, 160, 196, 256.

- A message to be hashed x is divided into blocks $x_1, x_2, \ldots, x_t$ of a fixed size. If the last block is shorter, it is padded using a padding rule to have a proper length.
- The hash value $h(x)$ is computed in an iterative way using the compression function f:
  $H_0 = IV$, $H_i = f(x_i, H_{i-1})$, $\qquad i = 1, 2, \ldots, t$,
  $h(x) = g(H_t)$.
  Here the IV is a given initial vector and g is the output function which is in many cases the identity function.

Both, IV and the padding rule, significantly affect the security of the hash function. IV is recommended to be a part of the hash function description. The padding rule should be designed in such a way that there do not exist two messages that will be padded to the same padded message.

A general model for MACs is similar to the previously described model for MDCs.

According to the construction of a compression function, hash function can be subdivided into the following categories:

MDCs based on a block cipher. An obvious motivation for such a construction is to adopt the knownledge on block ciphers due to the similarities between an iterative hash function and a block cipher. Moreover, it enables to reuse existing optimized designs and implementations which yields in the cost reduction of cryptographic hardware. Block ciphers are fast enough to provide sufficient speed for hashing. However, custom designed hash functions are usually much faster (realize that hash functions based on block ciphers require a key change after every encryption). One might naturally believe that the security of a block cipher will be handed over to the derived hash function. On the other hand, some weaknesses may appear due to the specific usage of a block cipher.

Various subcategories may be identified according to the relation among the hash value length, the key size, and the block size (see [63] for a detailed information).

MDCs based on algebraic structures and mathematical problems. These hash functions are based on known difficult mathematical problems, usually from number theory. This allows in many cases to prove security properties of a constructed hash function. The another design approach is to use operations from various algebraic structures. One of the main reasons for such an approach is the hardware reusability (e.g. modular arithmetic in RSA), since hash functions are typically used with signature schemes. See [63] for more detailed information. There were some designs also based on quasigroups published in recent years, e.g. [27], [50], [21], [22] (cryptanalysis of the last two designs can be found in Section 4.4). Finally, MAC based on a quasigroup can be found in [2].

Custom designed MDCs. This category covers the designs that are especially oriented on hashing. A common way is to use the so-called Davies-Meyer approach: the compression function is a block cipher, keyed by the text input x_i; the plaintext is the value H_{i-1}, which is also added to the ciphertext (feedforward) [63]. It must be said that almost all hash functions used in practice rank into this category.

Examples of custom designed hash functions are the well known algorithms such as MD4, MD5 designed by Rivest. A practical attack against the MD4 can be found in [20]. Several problems concerning collision resistance in MD5 were shown in [8], however they do not represent a real threat (see e.g. the evaluation of MD5 in the CRYPTREC Project). Many hash function designs were inspired by the MD4 hash function. We also speak about the MDx-family [63]. Another very popular hash function is SHA-1. It is an improved version of MD4, designed by NIST [25] (a newer version with 256, 384 and 512-bit hash value length is nowadays prepared). The "European" hash function, designed by Dobbertin et al. [19], is the RIPEMD-160. It is based on the design of MD4, too. We recall that SHA-1 and RIPEMD-160 are the only recommended hash functions in the Order of the National Security Authority of the Slovak Republic pursuant to the Electronic Signature Law [60]. A large number of MDCs and MACs can be found also in [78], [54].

Although almost all the practically used hash functions rank into the category of custom designs, a lot of work has been done in the area of the design of MDCs based on block ciphers and also on mathematical problems and algebraic structures. The study of these constructions helps us to better understand hash function as a cryptographic primitive.

3.8 Attacks on hash functions

In the following we present a taxonomy of attacks against MDCs, as it was described in [63].

Attacks independent of the algorithm

These attacks depend only on the size of the hash result (m bits) and do not exploit specific features of the hash algorithm.

Random (2nd) preimage attack. The attacker selects a random message and hopes that the given hash result will be hit. If the hash function has the required "random" behaviour, his probability of success equals $1/2^m$, where m is the number of bits of the hash result. In order to guarantee security for the next 15-20 years[3], m should be at least 80.

Birthday attack. The attacker generates r_1 variations on a bogus message and r_2 variations on a genuine message. The expected number of collisions equals $r_1 r_2/m$. The probability of finding a bogus message and a genuine message that hash to the same result is given by $1 - \exp(-r_1 r_2/2^m)$, which is about 63% when $r_1 = r_2 = 2^{m/2}$. References to several tricks that can be used to improve this attack in practice can be found in [63].

Attacks dependent on the chaining

Meet-in-the-middle attack. This attack is a variation on the birthday attack, but instead of comparing the hash result, one compares intermediate chaining variables. The attack enables an attacker to construct a (2nd) preimage, which is not possible for a simple birthday attack. The opponent generates r_1

[3]One has to realize that, according to the famous Moore's law, the speed of computers is multiplied by four every three years

variations on the first part of a bogus message and r_2 variations on the last part. Starting from the initial value and going backwards from the hash result, the probability for a matching intermediate variable is again $1 - \exp(-r_1 r_2 / 2^m)$. The only restriction that applies to the meeting point is that it cannot be the first or last value of the chaining variable.

Fixed point attack. The idea of this attack is to look for an H_{i-1} and x_i such that $f(x_i, H_{i-1}) = H_{i-1}$. If the chaining variable is equal to H_{i-1}, it is possible to insert an arbitrary number of blocks equal to x_i without modifying the hash result. Of course this attack can be extended to fixed points that occur after more than one iteration.

3.9 Projects NESSIE and CRYPTREC

NESSIE

New European Schemes for Signatures, Integrity and Encryption (NESSIE) was a 3-year research project within the Information Societies Technology (IST) Programme of the European Commission under the umbrella of the Fifth Framework Programme (FP5) [68], [64].

The main objective of the project was to put forward a portfolio of strong cryptographic primitives that had been obtained after an open call and had been evaluated using a transparent and open process. The project goal is to widely disseminate the project results and to build consensus based on these results using the appropriate fora (a project industry board, 5th Framework programme, and various standardization bodies). A final objective is to maintain the strong position of European research while strengthening the position of European industry in cryptography [68].

The project launched an open call (March 2000) for a broad set of primitives providing confidentiality, data integrity, and authentication. These primitives include block ciphers, stream ciphers, hash functions, MAC algorithms, digital signature schemes, and public-key encryption schemes. In addition, the NESSIE call asked also for evaluation methodologies for these primitives. The call also specified the main selection criteria: long-term security, market requirements, efficiency and flexibility.

As it can be seen, the scope of the NESSIE call was much wider than that of the AES call launched by NIST [58], which was restricted to 128-bit block ciphers. It is comparable to that of the RACE Project RIPE (RACE Integrity Primitives Evaluation, 1988-1992) [70] (confidentiality algorithms were excluded from RIPE for political reasons) and that of the Japanese CRYPTREC project (which also includes key establishment protocols and pseudo-random number generation) [65].

Another difference is that both AES [58] and CRYPTREC [65] intend to produce algorithms for government standards. The results of NESSIE will not be adopted by any government or by the European commission. However, the intention is that relevant standardization bodies will adopt these results.

There were all together 40 submissions to the NESSIE Project. Of course, there was an evaluation methodology (both for security and performance evaluation) and a software toolbox to support the evaluation (an improved version of the tools developed by the RIPE, but it is not publicly available) developed within the NESSIE project.

Due to the scope of this dissertation, we will mention only stream ciphers, hash functions and MACs.

The following synchronous stream ciphers were evaluated: BMGL, Leviathan, LILI–128, SNOW, SOBER-t16, and SOBER-t32.

The portfolio of suggested stream ciphers was empty!

The NESSIE portfolio of collision-resistant hash functions includes Whirpool, SHA-256, SHA-384, and SHA-512.

The NESSIE portfolio of MACs includes UMAC, TTMAC, EMAC, and HMAC.

See materials (e.g. final decision) in [68] for the explanation.

The relevant primitives from the NESSIE portfolia were suggested to be incorporated into e.g. ISO/IEC JTC 1/SC 27, IS 10118-3, ISO 18033.

A new project ECRYPT - European Network of Excellence in Cryptology started within the Information Societies Technology (IST) Programme of the European Commission under the umbrella of the Sixth Framework Programme (FP6) only a few months ago [67].

CRYPTREC

CRYPTREC is a project of the Japanese Information-Technology Promotion Agency (IPA). Its main objective is to prepare a list of cryptographic primitives and techniques available for use by the e-Government [65]. (Japan's e-Government project was set for inauguration by fiscal year 2003.)

The project launched an open call (June and July 2000) for a broad set of primitives. The categories were public-key cryptosystems, symmetric ciphers (stream ciphers, 64- and 128-bit block ciphers), hash functions and pseudo-random number generators. Besides the submitted algorithms also a large number of non-submitted algorithms were evaluated. Due to the national character of this project (e-Government), the evaluators were only from Japan.

These stream ciphers were evaluated in the CRYPTREC project: MULTI-S01, TOYOCRYPT-HS1, C4-1, FSAngo, MUGI, and RC4. Among them, MUGI, MULTI-S01, and RC4 were evaluated as "practically secure" and are the recommended ones (see [39] for further details).

The following hash functions were evaluated in the CRYPTREC project: MD5, RIPEMD-160, SHA-1, and draft SHA-256, 384, 512. All of them, except MD5, were evaluated as "practically secure" and are the recommended ones (see [39] for further details).

The final decision about other recommended cryptographic primitives can be found in [66].

Chapter 4

Results

The following results of the research are presented in this section:

- Cryptographic properties of the concatenation of periods of several ml-pseudorandom sequences are studied in Section 4.1. This section is based on [87], [88].
- Section 4.2 deals with cryptanalysis of one stream cipher based on the concatenation of transformed runs of two ml-sequences. It is based on [85], [86].
- Several attacks on one stream cipher based on a quasigroup (proposed in [59]) are described in Section 4.3. Main results of this section were published in [90], [91].
- The properties of one hash function based on a quasigroup (proposed in [21], [22]) are studied in Section 4.4. This section is based on the papers [92], [93].

4.1 A new construction of a completely equidistributed sequence

In the following we introduce necessary notions concerning completely equidistributed sequences (for more details see [44], [45]).

Definition 4.1.1 *b-ary sequence $X_1, X_2, X_3, \ldots$ is called k-distributed, if $Pr(X_{n+1} = a_1, X_{n+2} = a_2, \ldots, X_{n+k} = a_k) = 1/b^k$, for all ordered k-tuples $(a_1, a_2, \ldots, a_k)$, $a_i \in \{0, 1, \ldots, b-1\}$. A sequence is called completely equidistributed, if it is k-distributed for all k.*

A simple construction of a k-distributed b-ary sequence ($k \in \mathbb{N}$, $k \geq 2$) was proposed by Ford [26]. Choose $X_1 = X_2 = \cdots = X_k = 0$ and then choose X_{n+k} for $0 < n \leq b^k$ according to the following rule: $X_{n+k} = 0$ if and only if all k-tuples $(X_{n+1}, X_{n+2}, \ldots, X_{n+k-1}, j)$ have already appeared in the sequence for $1 \leq j < b$.

For example, if $b = k = 3$, and if we choose X_{n+k} to be the smallest value consistent with the above rule, we obtain 000111211012212010210022220200.

Now let $A(b,k)$ be the finite k-distributed b-ary sequence consisting of its first b^k terms, with each term divided by b. Thus, each element of $A(b,k)$ is a real number in $[0,1)$. Further, let $A(b,k)^n$ denote the sequence $A(b,k)$ repeated n times.

The first construction of a completely equidistributed sequence is known due to Knuth (see [44]) and is shown in the following theorem.

Theorem 4.1.2 *The sequence of real numbers*

$$A(2,1)^{1.2^2}, A(2^2,2)^{2.2^4}, A(2^3,3)^{3.2^6}, \ldots$$

is completely equidistributed.

We found that although Ford's and Knuth's sequences have uniform distribution of patterns, they posses several weaknesses (see [87] for details). Ford's k-distributed b-ary sequence appears to be vulnerable to differential cryptanalysis. It has a non-flat distribution of difference parameters (see [15]) with distances between peaks equal to k. Similar weaknesses appear in (a finite part of) Knuth's completely equidistributed sequence, too. From a practical point of view both sequences are difficult to be produced by a hardware-designed generator which makes them improper for a usage in real-world applications. The Knuth's construction of a completely equidistributed real-valued sequence leads to an infinite sequence and is based on the concatenation of an infinite number of k-distributed b-ary sequences.

These nice statistical properties of completely equidistributed sequences motivated us to find another (more practical) construction of such a sequence. Since in computer-world we have to deal with finite sequences only, we turn our attention to some finite part of a newly constructed sequence.

The new construction of a completely equidistributed real valued sequence is based on the concatenation of *ml*-sequences. *ml*-

sequences are almost k-distributed, only the all-zero k-tuple is missing in one period of this sequence. Proofs of the next results of this section are published in [88].

Theorem 4.1.3 *Let p be a prime, l an integer, and $b = p^l$. Further, let $ML'(b,k)$ be a finite b-ary ml-sequence consisting of the first $b^k - 1$ terms, generated by some primitive polynomial over $GF(b)$. Let $ML(b,k)$ be constructed from $ML'(b,k)$ by dividing each term by b. Real valued sequence*

$$ML(2,1)^{1.2^2}, ML(2^2,2)^{2.2^4}, ML(2^3,3)^{3.2^6}, \ldots$$

is completely equidistributed.

Sketch of the proof: The proof is similar to the proof of Theorem 3 in [44]. The missing k-tuple in the ml-sequence is ignored just as the effects that appear at the borders of the concatenated sequences. □

If we consider a deterministic hardware production of such a pseudorandom sequence we are constrained on building only (ultimately) periodic sequences consisting of terms from some finite set.

Thus we may use only a part of the above defined sequence. Moreover, we need to transform the terms of $ML'(b,k)$ sequences, e.g. into $GF(2)$.

Because of the practical point of view we studied the local properties, namely the concatenation of two (or more) ml-sequences over $GF(2)$. Our analysis, using extensive computer simulations, showed that the concatenation of two ml-sequences possesses a large linear complexity and moderate "out-of-phase" autocorrelation function magnitudes. Moreover, we found the period of such a sequence.

Theorem 4.1.4 *Let $u = u_0, u_1, \ldots, u_{2^{\deg c_1(x)}-2}$, and $v = v_0, v_1, \ldots, v_{2^{\deg c_2(x)}-2}$ be one period of a sequence produced by primitive polynomial $c_1(x)$, and $c_2(x) \in GF(2)[X]$, respectively, $\deg c_1(x)$, $\deg c_2(x) > 1$, $\deg c_1(x) \neq \deg c_2(x)$, $\deg c_1(x) \neq 2$ and $\deg c_2(x) \neq 3$, and vice versa. Then the period of a sequence $u, v, u, v, \ldots$ produced by concatenation of sequences u and v is equal to $(2^{\deg c_1(x)} - 1) + (2^{\deg c_2(x)} - 1)$.*

Proof: Assume that the period d of the sequence is one of the non-trivial divisors of $(2^{\deg c_1(x)} - 1) + (2^{\deg c_2(x)} - 1)$.

Let $MAX = \max\{\deg c_1(x), \deg c_2(x)\}$. It follows from the pattern distribution property of a ml-sequence that $2^{MAX} - 1 - MAX < d \leq (2^{\deg c_1(x)} - 1 + 2^{\deg c_2(x)} - 1)/2$. This inequality is a contradiction. In the special case, where $\deg c_1(x) = 2$, $\deg c_2(x) = 3$, the possible periods are $d = 5$ and $d = 10$. In all other cases the period of a sequence is $d = (2^{\deg c_1(x)} - 1) + (2^{\deg c_2(x)} - 1)$. $\square$

This theorem can be generalized as follows:

Theorem 4.1.5 *Let* $u^1 = u^1_0, u^1_1, \ldots, u^1_{2^{\deg c_1(x)}-2}$, $u^2 = u^2_0, u^2_1, \ldots u^2_{2^{\deg c_2(x)}-2}$, $\ldots$, $u^n = u^n_0, u^n_1, \ldots, u^n_{2^{\deg c_n(x)}-2}$ *be one period of a sequence produced by primitive polynomials* $c_1(x), c_2(x), \ldots, c_n(x) \in GF(2)[X]$, $3 < \deg c_1(x) < \deg c_2(x) < \cdots < \deg c_n(x)$ *or* $\deg c_1(x) > \deg c_2(x) > \cdots > \deg c_n(x) > 3$. *Then the period of a sequence* $u^1, u^2, \ldots, u^n, u^1, u^2, \ldots, u^n, \ldots$ *produced by concatenation of sequences* $u^1, u^2, \ldots, u^n$ *is* $\sum_{i=1}^{n}(2^{\deg c_i(x)} - 1)$.

Proof: The inequality used in the proof of the Theorem 4.1.4 must be slightly modified: $2^{MAX} - 1 - MAX < d \leq (\sum_{i=1}^{n}(2^{\deg c_i(x)} - 1))/w$, where $MAX = \max\{\deg c_1(x), \deg c_2(x), \ldots, \deg c_n(x)\}$, and $w = 2$ for n even, $w = 3$ for n odd. The rest of the proof is similar to the proof of the Theorem 4.1.4. $\square$

Conclusions

There are cryptographic properties of some specially constructed sequences studied in this section. The period of a sequence obtained by periodic concatenation of two or more ml-sequences is determined. Moreover, a new construction of a completely equidistributed real valued sequence based on the concatenation of ml-sequences is presented.

4.2 Concatenation of runs from two ml-sequences

This section deals with the cryptanalysis of one running key generator which combines the outputs of two asynchronously clocked LFSRs. Its keystream production could be characterized as concatenation of transformed runs of two ml-sequences. Computer simulations show a large linear complexity of the produced keystream sequence. The period of the keystream and several theorems concerning the number of runs in an ml-sequence are proved. Conditions for passing the Golomb's randomness postulates are proposed. Results of the performed statistical tests (FIPS 140-1, gap test, serial correlation test) are presented. Finally, a known plaintext attack against the studied running key generator is presented.

This section is based on author's papers [85] and [86].

Description of the running key generator

The generator G consists of two asynchronously clocked (in a stop-and-go fashion) LFSRs $L1$ and $L2$, respectively. The key of the generator is the initial state of the registers $L1$ and $L2$. Assume the polynomials $c_1(x), c_2(x) \in GF(2)[X]$ associated to the registers $L1$, $L2$ are primitive. Let $\tilde{a} = \tilde{a}_0, \tilde{a}_1, \ldots$, resp. $\tilde{b} = \tilde{b}_0, \tilde{b}_1, \ldots$ be the binary sequence produced by clock-controlled (as used in generator G) registers $L1$, resp. $L2$. Moreover, let $a = a_0, a_1, \ldots$, resp. $b = b_0, b_1, \ldots$ be the binary sequence produced by the regularly clocked registers $L1$, resp. $L2$.

Algorithm of the generator G:

1. Keystream bit production: $z_t = L1(t) \oplus L2(t) = \tilde{a}_t \oplus \tilde{b}_t$.
2. Next-state function: if $z_t = 1$, then $L1$ clocks, otherwise ($z_t = 0$) $L2$ clocks.

Example 4.2.1 *Assume the following realization of the generator G: $c_1(x) = 1+x+x^2$ and $c_2(x) = 1+x+x^3$. Let us look at the changes of the registers $L1$ and $L2$ states during the keystream generation. (Output bits $\tilde{a}_t$, resp. $\tilde{b}_t$ are the underlined bits of the $L1$, resp. $L2$ states. The state of a register that clocks at a given time t is bold typed. The underlined bits of bold typed states of register $L1$, resp. $L2$ form runs (either blocks B_i^a, resp. B_i^b or gaps G_i^a, resp. G_i^b) of the sequences a, and b, respectively.*

t	State of $L1$	State of $L2$	z_t	Runs of a	Runs of b
0	$\underline{0}1$	$\mathbf{\underline{0}01}$	0		G_0^b
1	$\underline{0}1$	$\mathbf{\underline{0}11}$	0		
2	$\mathbf{\underline{0}1}$	$\underline{1}11$	1	G_0^a	
3	$\underline{1}1$	$\mathbf{\underline{1}11}$	0		B_1^b
4	$\underline{1}1$	$\mathbf{\underline{1}10}$	0		
5	$\underline{1}1$	$\mathbf{\underline{1}01}$	0		
6	$\mathbf{\underline{1}1}$	$\underline{0}10$	1	B_1^a	
7	$\mathbf{\underline{1}0}$	$\underline{0}10$	1		
8	$\underline{0}1$	$\mathbf{\underline{0}10}$	0		G_2^b
9	$\mathbf{\underline{0}1}$	$\underline{1}00$	1	G_0^a	
10	$\underline{1}1$	$\mathbf{\underline{1}00}$	0		B_3^b
11	$\mathbf{\underline{1}1}$	$\underline{0}01$	1	B_1^a	
12	$\mathbf{\underline{1}0}$	$\underline{0}01$	1		
13	$\underline{0}1$	$\mathbf{\underline{0}01}$	0		

Table 4.1: Generation of a keystream of the generator G

Observation 4.2.2 *The keystream production could be characterized as joining transformed runs of sequences a and b (look at the relation among underlined bold typed bits, z_t, and runs of the sequences a, and b, respectively).*

Analysis of the keystream

Theorem 4.2.3 *Let u be an ml-sequence generated by an ml-LFSR with associated primitive polynomial $c(x)$, $\deg c(x) > 1$. Let $u_0 = u_1 = \cdots = u_{\deg c(x)-2} = 0, u_{\deg c(x)-1} = 1$. Then the number of runs in one period of the sequence u is even. Moreover, the number of blocks is equal to the number of gaps.*

This theorem follows from the fact $u_{2^{\deg c(x)}-2} = 1$.

Next, we determine the exact number of runs in one period of an ml-sequence.

Theorem 4.2.4 *Let u be a sequence generated by an ml-LFSR with associated primitive polynomial $c(x) \in GF(2)[X]$, $\deg c(x) > 1$. Assume $u_0 = 0, u_1 = 0, \ldots, u_{\deg c(x)-2} = 0, u_{\deg c(x)-1} = 1$. Then the number of runs in one period of the sequnce u is $2^{\deg c(x)-1}$.*

Proof: According to the Theorem 4.2.3, it is sufficient to prove that the number of blocks in one period of $u = u_0, u_1, \ldots, u_{2^{\deg c(x)}-2}$ is equal to $2^{\deg c(x)-2}$. Let us denote $B^u[i]$ the number of blocks of length $\deg c(x) - i$. It follows from the Theorem 3.3.6, that

$$B^u[i] = 2^{\deg c(x)-(\deg c(x)-i)} - \sum_{j=0}^{i-1} B^u[j](i-j+1).$$

$$\sum_{j=0}^{i} B^u[j] = 2^{i-1},\ 1 \leq i < \|L\|$$
$$B_0 = 1$$

□

Corollary 4.2.5 *(of the Theorem 4.2.4). Let u and v be ml-sequences generated by ml-LFSRs L_u and L_v with associated primitive polynomials $c_u(x), c_v(x) \in GF(2)[X]$, $\deg c_u(x) = \deg c_v(x)$. Let $u_0 = u_1 = \cdots = u_{\deg c_u(x)-2} = 0, u_{\deg c_u(x)-1} = 1$ and $v_0 = v_1 = \cdots = v_{\deg c_v(x)-2} = 0, v_{\deg c_v(x)-1} = 1$. Then the sequences u and v have the same number of blocks, and gaps of lengths $1, 2, ..., \deg c_u(x) = \deg c_v(x)$.*

Next, we generalize Theorem 4.2.4 for any non-zero initial state of the generating register.

Theorem 4.2.6 *Let u denote an ml-sequence generated by an ml-LFSR (from a non-zero initial state) with associated primitive polynomial $c(x) \in GF(2)[X]$, $\deg c(x) > 1$. Then the number of runs in one period of the sequence u is either $2^{\deg c(x)-1}$ or $2^{\deg c(x)-1} + 1$.*

Proof: Let us denote a sequence u that starts with $u_0 = u_1 = \cdots = u_{\deg c(x)-2} = 0, u_{\deg c(x)-1} = 1$ as w. Realize that any sequence u can be obtained from the sequence w by shifting [4, pp.350–351]. Thus the sequence w can be shifted to the beginning of a new run, which yields the number of runs equal to $2^{\deg c(x)-1}$ or somewhere inside a run, which yields the number of runs equal to $2^{\deg c(x)-1} + 1$. □

The following theorem concerning the period of the keystream of the generator G is based on the Observation 4.2.2 and on the Theorem 4.2.6. (A conjecture was presented in [86].)

Theorem 4.2.7 *Assume that the registers $L1$, resp. $L2$ with associated primitive polynomials $c_1(x)$, resp. $c_2(x)$, $\deg c_1(x)$, $\deg c_2(x) > 1$ are loaded with a non-zero initial state. Then the period of the keystream sequence z of the generator G is*

$$(2^{\max\{\deg c_1(x),\deg c_2(x)\}}-1)+2^{|\deg c_1(x)-\deg c_2(x)|}(2^{\min\{\deg c_1(x),\deg c_2(x)\}}-1). \tag{4.1}$$

Proof: First, assume that (4.1) is an integer multiple of the period of the keystream sequence z. There are 16 possibilities according to the start and end runs of one period of the sequences a and b. Look at one of them (the other possibilities can be analyzed in a similar way).

Assume that a starts with a gap and ends with a block, b starts and ends with a gap. (The notation of blocks and gaps is similar to that used in Example 4.2.1.)

$L1$		G_0^a		B_1^a	$\dots$	$B^a_{2^{\deg c_1(x)-1}}$			G_0^a
$L2$	G_0^b		B_1^b		$\dots$		$G^b_{2^{\deg c_2(x)-1}+1}$	G_0^b	
	$*$	$*$	$*$	$*$	$\dots$	$*$	$*$		
		○	○	○	$\dots$	○	○	○	

Table 4.2: Joining of runs during the production of the keystream

The *-denoted (as well as the ○-denoted, that have the start and end runs from different registers) runs clearly form an integer multiple of the period of the keystream sequence s.

Finally realize that the ○-denoted part of the keystream sequence contains exactly one block of length $\|L1\|$ (if $\|L1\| \geq \|L2\|$) or one gap of length $\|L2\|$ (if $\|L1\| \leq \|L2\|$). Thus the ○-denoted part of the keystream must form exactly one period.

The next theorem characterizes the basic balancedness of the keystream sequence.

Theorem 4.2.8 *Assume that the registers $L1$, resp. $L2$ with associated primitive polynomials $c_1(x)$, resp. $c_2(x)$, $\deg c_1(x)$, $\deg c_2(x) > 1$ are loaded with a non-zero initial state. Then the number of ones and zeros in one period of the generated keystream sequence is given as follows:*

1. *if* $\deg c_1(x) \geq \deg c_2(x)$ *then*
   number of ones is $2^{\deg c_1(x)} - 1$
   number of zeros is $2^{|\deg c_1(x) - \deg c_2(x)|}(2^{\deg c_2(x)} - 1)$*;*
2. *if* $\deg c_1(x) < \deg c_2(x)$ *then*
   number of ones is $2^{|\deg c_1(x) - \deg c_2(x)|}(2^{\deg c_1(x)} - 1)$
   number of zeros is $2^{\deg c_2(x)} - 1$*.*

Proof: Theorem 4.2.8 follows from the proof of the period and from the production of the keystream as joining transformed runs from sequences a and b. $\square$

The following theorem about passing the first and second Golomb's postulates follows.

Theorem 4.2.9 *Assume that the registers* $L1$*, resp.* $L2$ *with associated primitive polynomials* $c_1(x)$*, resp.* $c_2(x)$*,* $\deg c_1(x)$*,* $\deg c_2(x) > 1$*,* $|\deg c_1(x) - \deg c_2(x)| \leq 1$ *are loaded with a nonzero initial state. Then the generated keystream sequence passes the first Golomb's postulate. Moreover, if* $\deg c_1(x) = \deg c_2(x)$ *then the keystream sequence passes the second Golomb's postulate, too.*

Statistical tests - results

Our realization of the generator G was tested for: $c_1(x) = 1 + x + x^2 + x^5 + x^{19}$, $c_2(x) = 1 + x^3 + x^{31}$. The test set consisted of 1 000 keystream sequences (each 20 000 bits long) produced by this realization of the generator.

All of the tested sequences passed all tests given by FIPS 140-1 [23] (at the time of writing the papers [85] and [86], FIPS 140-2, which is the successor of FIPS 140-1, was not admitted), 95% of them passed the serial correlation test [45] and none of them passed the gap test [45].

Table 4.3 outlines the values of the serial correlation coefficient, the statistics for the poker test [54, p.182], and the number of ones in a keystream sequence for the monobit test [23]. The last row of the table shows the expected intervals.

#	Serial correlation test	Poker test (for quadruples)	Monobit test
0	0.002385	15.5264	9958
1	0.001986	6.08640	9734
2	-0.005419	9.92	10003
3	0.003574	10.8288	9750
4	0.007899	18.5472	9986
5	0.007897	16.6784	10156
6	0.004948	10.5088	10248
7	-0.002401	18.0352	10067
8	-0.000602	15.904	9973
9	-0.000602	16.6464	10224
	[-0.00068,0.00068]	[1.03,57.4]	[9654,10346]

Table 4.3: Results of statistical tests

# / run length	1	2	3	4	5	6
0	2459	1302	693	386	187	123
1	2642	1209	713	267	176	160
2	2340	1179	604	245	173	129
3	2400	1167	530	361	155	134
4	2589	1376	587	378	113	149
5	2512	1391	536	276	180	167
6	2680	1268	568	246	99	187
7	2540	1290	633	358	137	181
8	2397	1104	712	369	169	152
9	2645	1176	589	374	138	192
	[2267,2733]	[1079,1421]	[502,748]	[223,402]	[90,223]	[90,223]

Table 4.4: Run test - numbers of occurrences of runs with certain lengths

According to the results of the Maurer's universal statistical test [52] the keystream sequence is not significantly compressable (in Table 4.5, Q denotes the number of initial blocks and K denotes the number of tested blocks).

Theorem 4.2.10 *The keystream sequence produced by the generator G passes the long run test (FIPS 140-1) if* $1 < ||L1||, ||L2|| < 34$ *(and registers* $L1$*, resp.* $L2$ *are loaded with a non-zero initial state).*

Proof of this theorem follows from the fact that the longest run in the keystream sequence has $\max\{||L1||, ||L2||\}$ bits (see Example 4.2.1). $\square$

#	$Q = 2560$	$Q = 25600$
	$K = 256000$	$K = 2560000$
0	8.003677	8.002048
1	7.999273	8.000793
2	8.002426	7.999964
3	8.000400	8.000941
4	8.001049	7.999997
5	7.999030	8.002516
6	7.998300	8.001611
7	8.000900	8.001506
8	7.999110	7.999905
9	8.002141	8.001281

Table 4.5: Maurer's universal statistical test - entropy on the 8-bit block

The cryptanalysed generator is based on an alternating clocking of its registers. Thus the situations when only one register clocks should be avoided. This happens when one of the registers produces the sequence with period equal to 1. It follows that the use of the zero initial state and the use of the polynomials of degree equal to 1 should be avoided.

Linear complexity

One of the important properties of the keystream sequence is its linear complexity. Based on the computer simulations (small sample of them is shown in the Table 4.6) we conclude that the produced keystream sequence has a large linear complexity.

Theorem 4.2.11 puts important restrictions on the choice of the lengths of registers $L1$ and $L2$ (see Expression 4.1).

Theorem 4.2.11 *[15, p.52, Theorem 3.4.4] Let N be an odd prime and q be a primitive root modulo N such that* $\gcd(N, q) = 1$. *Then the linear complexity of any nonconstant sequence u of period N over $GF(2)$ is N or $N - 1$.*

Linear complexity profile

According to the performed simulations the linear complexity profile of the keystream sequence has no significant differences from the optimum. The small differences do not weaken the generator.

$L1$ polynomial	$L2$ polynomial	Linear complexity
x^2+x+1	x^2+x+1	7
x^2+x+1	x^3+x+1	13
x^2+x+1	x^4+x+1	25
x^2+x+1	x^5+x^2+1	55
x^2+x+1	x^6+x+1	109
x^2+x+1	x^7+x+1	223
x^2+x+1	x^7+x^3+1	223
x^3+x+1	x^2+x+1	13
x^3+x+1	x^3+x+1	14
x^3+x+1	x^4+x+1	28
x^3+x+1	x^5+x^2+1	58
x^3+x+1	x^6+x+1	115
x^3+x+1	x^7+x+1	238
x^3+x+1	x^7+x^3+1	238
x^4+x+1	x^2+x+1	25
x^4+x+1	x^3+x+1	29
x^4+x+1	x^4+x+1	30
x^4+x+1	x^5+x^2+1	60
x^4+x+1	x^6+x+1	120

Table 4.6: Linear complexity of the keystream sequence.

Attacks

The simplest attack against any cipher system is the brute-force attack. Its complexity depends on the size of the keyspace. The complexity of the brute-force attack against the cryptanalysed generator is given by the following formula (the zero initial states are excluded):

$$(2^{\|L1\|}-1)(2^{\|L2\|}-1).$$

The cryptanalysed running key generator is vulnerable to the known plaintext attack. We show how to find the key of the generator.

Assume the plaintext $P = p_0, p_1, \ldots, p_{N-1}$, and the corresponding ciphertext $C = c_0, c_1, \ldots, c_{N-1}$ are given. Applying the encryption formula for the stream cipher systems $c_t = p_t \oplus z_t$ we immediately calculate the keystream sequence z.

Realize that in one step only one register clocks. Thus we can build a simple algorithm for the initial state of the $L1$ and $L2$ registers reconstruction.

Algorithm 4.2.12 *(Reconstruction of initial states)*

```
1. if $s_0 = 1$ then
   $a_0^{(1)} \leftarrow 0, b_0^{(1)} \leftarrow 1$
   $a_0^{(2)} \leftarrow 1, b_0^{(2)} \leftarrow 0$
   else
   $a_0^{(1)} \leftarrow 0, b_0^{(1)} \leftarrow 0$
   $a_0^{(2)} \leftarrow 1, b_0^{(2)} \leftarrow 1$
2. $i \leftarrow 0$
   $j \leftarrow 0$
   $t \leftarrow 0$
3. if $i \geq \|L1\| - 1$ and $j \geq \|L2\| - 1$ then terminate; the initial
   states of $L1$ and $L2$ are reconstructed
4. if $t = N - 1$ then terminate; the initial states of $L1$ and $L2$ are
   partially reconstructed
5. $t \leftarrow t + 1$
6. if $s_{t-1} = 0$ then $j \leftarrow j + 1$
   else $i \leftarrow i + 1$
7. solve the equation $s_t = a_i \oplus b_j$
8. go to step 3
```

The number of found bits of the register $L1$, resp. $L2$ is determined by the number of ones, resp. zeros in the keystream sequence z.

There are two possible solutions after the algorithm terminates:

1. $a^{(1)} = a_0^{(1)}, a_1, \ldots, a_i$ $\qquad b^{(1)} = b_0^{(1)}, b_1, \ldots, b_j$
2. $a^{(2)} = a_0^{(2)}, a_1, \ldots, a_i$ $\qquad b^{(2)} = b_0^{(2)}, b_1, \ldots, b_j$

The correct solution could be found when keystreams produced from these two solutions are compared to the keystream z.

The complexity of the initial state reconstruction is of order $O(2N2^{m_1}2^{m_2})$, where m_1, resp. m_2 denote the number of missing bits of initial state of the registers $L1$, resp. $L2$. In an optimal case an $(\|L1\| + \|L2\| - 1)$ bits long keystream sequence is sufficient for finding the whole key with the complexity of order $O(2N)$.

A note on the security of the generator

Assume now that the generator G consists of two subgenerators $G1$ and $G2$, respectively.

Using the known plaintext attack presented above it is easy to find sequences a and b generated by these subgenerators $G1$ and $G2$. Thus the security of the whole generator against the known plaintext attack depends on the security of $G1$ and $G2$ against this kind of an attack.

Clearly, when using LFSRs $L1$ and $L2$ as the subgenerators $G1$ and $G2$, the key of the generator (the initial loading of the registers $L1$ and $L2$) is directly the beginning part of the sequences a and b.

Conclusions

There are several theorems determining the number of runs in an ml-sequence presented in this section. The period of the keystream sequence of the cryptanalysed generator is determined as well as its basic statistical properties. The keystream sequence possesses good cryptographic properties as long period and large linear complexity. The results of statistical tests are outlined. A known plaintext attack on the studied running key generator is proposed. The security of the generator against the known plaintext attack is generalized.

4.3 Attacks on one stream cipher based on a quasigroup

There are several attacks on a stream cipher, that was proposed in [59], presented in this section. Almost the same cipher was proposed also in [49]. The cryptanalysed stream cipher is based on a hidden quasigroup (that represents the key). It works in a self-synchronizing fashion and was suggested for the encryption of a file system. The cipher has a very large keyspace and was claimed to be resistant against any attack [59].

Main results of this section were published in [90] and [91].

Self-synchronizing stream cipher based on a quasigroup

Definition 4.3.1 *[18] The structure* $(Q, *)$, $Q = \{q_1, q_2, \ldots, q_n\}$, $\|Q\| = n$ *is called a finite quasigroup of order* n *if, when any two elements* $a, b \in Q$ *are given, the equations* $a * x = b$ *and* $y * a = b$ *each have exactly one solution. Thus the Caley table of a finite quasigroup of order* n *is a Latin square, i.e. an* $n \times n$ *array with the property that each row and each column contains the permutation of symbols from* Q*. The operation* $\backslash$ *is called the right inverse of* $*$ *if it holds that* $x\backslash(x * y) = y$*, and* $x * (x\backslash y) = y$*.*

Let $(Q, *)$ be a finite quasigroup. Let individual plaintext characters be represented by the elements of Q, i.e. $p_1, p_2, \ldots, p_k$, $p_i \in Q$, $1 \le i \le k$. Similarly let the ciphertext characters $c_1, c_2, \ldots, c_k$ be represented also by the elements of Q, i.e. $c_i \in Q$, $1 \le i \le k$. The key of the studied stream cipher is the definition of the operation $*$ on the set Q, i.e. the Caley table of this operation[1].

Encryption:

$encrypt(p_1, p_2, \ldots, p_k) = c_1, c_2, \ldots, c_k$.

$c_1 = l * p_1$, where l is a given "initial value".

$c_{i+1} = c_i * p_{i+1}$, $i = 1, 2, \ldots, k-1$.

Decryption:

$decrypt(c_1, c_2, \ldots, c_k) = p_1, p_2, \ldots, p_k$.

$p_1 = l\backslash c_1$.

$p_{i+1} = c_i\backslash c_{i+1}$, $i = 1, 2, \ldots, k-1$.

[1]It is a rather strange design concept. Nowadays ciphers do not use operations on sets as keys. The key is an item in some set, e.g. an element of Q that is kept secret.

Example 4.3.2 *Let* $Q = \{0, 1, 2\}$ *and let the quasigroups* $(Q, *)$, *resp.* $(Q, \backslash)$ *be defined by Table 4.7. Let* $l \in Q, l = 0$.

$*$	0	1	2
0	1	2	0
1	2	0	1
2	0	1	2

$\backslash$	0	1	2
0	2	0	1
1	1	2	0
2	0	1	2

Table 4.7: Caley tables of quasigroups $(Q, *)$ and $(Q, \backslash)$

$encrypt(1, 2, 0, 0, 0, 1, 1, 2, 0) = 2, 2, 0, 1, 2, 1, 0, 0, 1$
$c_1 = l * p_1 = 0 * 1 = 2$
$c_2 = c_1 * p_2 = 2 * 2 = 2$
$c_3 = c_2 * p_3 = 2 * 0 = 0$
$\vdots$

$decrypt(2, 2, 0, 1, 2, 1, 0, 0, 1) = 1, 2, 0, 0, 0, 1, 1, 2, 0$
$p_1 = l \backslash c_1 = 0 \backslash 2 = 1$
$p_2 = c_1 \backslash c_2 = 2 \backslash 2 = 2$
$p_3 = c_2 \backslash c_3 = 2 \backslash 0 = 0$
$\vdots$

There are at least $n!(n-1)!(n-2)! \ldots 2!$ Latin squares of order n. If we assume that $Q = \{0, 1, \ldots, 255\}$ (i.e. each data item is represented by 8 bits = 1 byte) then there are at least $10^{58\,000}$ quasigroups. The keyspace is enormously large. The cipher was claimed to be resistant against any attack [59] although the authors studied only resistance against brute force attack and performed some statistical tests on this cipher. From a point of view of cryptanalysis, a good cipher should be resistant against ciphertext-only attack, chosen/known ciphertext/plaintext attacks, as well. Some possible attacks are shown below.

Chosen ciphertext attack

Let $Q = \{q_1, q_2, \ldots, q_n\}$ and assume the cryptanalyst has access to the decryption device loaded with an unknown key. Then he/she can construct the following ciphertext:

$q_1, q_1, q_1, q_2, q_1, q_3, \ldots, q_1, q_n,$
$q_2, q_1, q_2, q_2, q_2, q_3, \ldots, q_2, q_n,$

$\vdots$

$q_n, q_1, q_n, q_2, q_n, q_3, \ldots, q_n, q_n$

and enter it into the decryption device. The decryption device gives the following plaintext:

$l\backslash q_1, q_1\backslash q_1, q_1\backslash q_1, q_1\backslash q_2, q_2\backslash q_1, q_1\backslash q_3, \ldots, q_1\backslash q_n,$

$\vdots$

$q_n\backslash q_n, q_n\backslash q_1, q_1\backslash q_n, q_n\backslash q_2, q_2\backslash q_n, q_n\backslash q_3, \ldots, q_n\backslash q_n.$

It is easy to see that the Caley table of the operation $\backslash$ defined on Q is completely found. The construction of the Caley table for the operation $*$ is straightforward. The ciphertext used in the attack consists of $2n^2$ characters. (Of course a shorter ciphertext can be constructed. The only requirement is that all the pairs of adjacent elements will appear in the ciphertext. The presented attack requires $2n^2$ operations $\backslash$.

One may also use the elementary fact that the last column and the last row of the Caley table of the operation defined on quasigroup is completely determined by previous rows, resp. columns. Generalization of this idea leads to the notion of critical sets of Latin squares. Recall that a partial Latin square of order n is an $n \times n$ array on a symbol set E, $\|E\| = n$, such that each cell is either empty or contains an element of E, and each element of E occurs in each row and in each column at most once. A critical set C of order n is a partial Latin square of order n which can be completed to a Latin square L in a unique way, and removing any element of C destroys that property. That is, C provides minimal information from which L can be reconstructed uniquely (see [18], [34]). Denote the minimum size of a critical set of order n by $M(n)$. In [14] it has been shown that $M(n) \leq \frac{n^2}{4}$, which is generally believed to be asymptotically the correct order of $M(n)$. The closest up to date lower bound on the size of the critical set of order n, $n \geq 8$ is $M(n) \geq \lfloor \frac{4n-8}{3} \rfloor$ (see [34]).

However, the implementation of the above mentioned facts on critical sets into the described attacks will lead to a problem of reconstruction of the Latin square from its critical set.

Chosen plaintext attack

Let $Q = \{q_1, q_2, \ldots, q_n\}$ and assume the cryptanalyst has access to the encryption device loaded with an unknown key. Then he/she can construct the following plaintexts:

$$\begin{array}{l}
q_1, q_1, \\
q_1, q_2, \\
\vdots \\
q_1, q_n, \\
q_2, q_1, \\
q_2, q_2, \\
\vdots \\
q_2, q_n, \\
\vdots \\
q_n, q_1, \\
q_n, q_2, \\
\vdots \\
q_n, q_n
\end{array}$$

and enter them into the encryption device. The following ciphertexts will be obtained:

$$\begin{array}{l}
l * q_1, (l * q_1) * q_1, \\
l * q_1, (l * q_1) * q_2, \\
\vdots \\
l * q_1, (l * q_1) * q_n, \\
l * q_2, (l * q_2) * q_1, \\
l * q_2, (l * q_2) * q_2, \\
\vdots \\
l * q_2, (l * q_2) * q_n, \\
\vdots \\
l * q_n, (l * q_n) * q_1, \\
l * q_n, (l * q_n) * q_2, \\
\vdots \\
l * q_n, (l * q_n) * q_n.
\end{array}$$

It is easy to see that the key, i.e. the Caley table of the operation

$*$ defined on Q is completely found. The presented chosen plaintext attack requires n^2 messages and each message consists of two characters. (Of course a smaller number of messages can be used. See the above described chosen ciphertext attack.) The presented attack requires $2n^2$ operations $*$.

It is also possible to built up a known plaintext attack. However it is not guaranteed that the whole key will be revealed.

Ciphertext-only attack

Let us assume that the plaintext message was written in a redundant language, e.g. Slovak, Czech, English, etc., i.e. the distribution of frequencies of occurences of individual characters is not uniform. Further let us assume that the language, the plaintext message was written in, is known and also that the cryptanalyst knows the distribution of frequencies of occurences of individual characters from the used language. Assume that each character from the plaintext message is represented by a single element from the quasigroup. Further assume that the order n of the quasigroup $(Q, *)$, where $Q = \{q_1, q_2, \dots, q_n\}$, is known. Let us denote the obtained ciphertext as $c_1, c_2, \dots, c_k$, $c_i \in Q$, $1 \leq i \leq k$.

For each i, $1 \leq i \leq n$, the cryptanalyst determines the number of occurences of pairs of elements $q_i q_j$, $1 \leq j \leq n$. If the ciphertext is large enough, for each q_i, $1 \leq i \leq n$, the obtained number of occurences of pairs of elements can be matched to the known frequencies of occurences of individual characters from the used language. Thus the cryptanalyst is able to construct the Caley table of the quasigroup $(Q, \backslash)$ and decrypt the message. The reconstruction of the key, i.e. of the Caley table of the quasigroup $(Q, *)$ from the quasigroup $(Q, \backslash)$ is straightforward.

However the matching of obtained number of occurences of pairs of elements to the known frequencies of occurences of individual characters from the used language can lead to some errors in the reconstruction of the quasigroup $(Q, \backslash)$, either due to the short length of the analyzed ciphertext or due to the specific properties of the used language. A better approach is to match only the obvious pairs of elements, then partially decrypt the ciphertext. From the partially decrypted message it is possible (or highly probable) to find some other cells in the Caley table of the quasigroup $(Q, \backslash)$. This leads to the iterated decryption, resp. iterated construction of the

Caley table of the quasigroup $(Q, \backslash)$. One can also use the known results on critical sets, resp. on completing Latin squares.

The previously described iterative ciphertext-only attack was successfully performed on a plaintext written in Slovak language (book "SLOVENSKO. Európske súvislosti ľudovej kultúry" by Rastislava Stoličná et al., VEDA Bratislava 1997). The plaintext was written in the enhanced telegraph alphabet, i.e. it consisted only of letters A,B, . . . ,Z and "space" and contained 291 041 characters. The quasigroup was of order 27.

Conclusions

There are three kinds of attacks against the self-synchronizing stream cipher (proposed in [59]) presented in this section. These attacks rank among the standard basic cryptanalyst techniques. Each of these attacks is much faster than the brute-force attack. We conclude that the cryptanalysed self-synchronizing stream cipher is insecure due to its vulnerability to the presented attacks.

4.4 Attacks on one hash function based on quasigroup

There are properties of a hash function based on a quasigroup (proposed in [21], [22]) studied in this section. An attack against this hash function for some special quasigroups is presented. Moreover, the modification of the studied hash function to a keyed hash function – the so called MAC is studied, too.

This section is based on author's papers [92], [93].

Description of the hash function based on a quasigroup

Construction 4.4.1 *(A new hash function [21], [22].) Let* $(Q, *)$ *be a finite quasigroup and* Q^* *be the set of all finite sequences of elements from* Q*. Let the message be a sequence of elements* $\{m_1, m_2, \dots, m_k\}$ *from the quasigroup* Q*. For a fixed* $a \in Q$ *let the hash function* $H_a : Q \times Q^* \to Q$ *be*

$$H_a(m_1, m_2, \dots, m_k) = ((\dots((a * m_1) * m_2) * \dots) * m_{k-1}) * m_k,$$

where $m_i \in Q$*,* $1 \leq i \leq k$*.*

Example 4.4.2 *Let* $Q = \{0, 1, 2, 3\}$ *and let the operation * on* Q *be defined by its Caley table, in Table 4.8.*

*	0	1	2	3
0	0	2	1	3
1	2	3	0	1
2	1	0	3	2
3	3	1	2	0

Table 4.8: Caley table of the operation * defined on Q

Let $a = 2$ *and let the message to be hashed be encoded as* $\{0, 0, 1, 3\}$*. Then the hash can be computed as*

$$H_2(0, 0, 1, 3) = (((2 * 0) * 0) * 1) * 3 = 3.$$

The usage of a general quasigroup in computation requires to store its Caley table, i.e. n^2 elements. The storage requirements are outlined in Table 4.9. (One can also take notice of a very short

$\|Q\|$	Hash value length	Storage requirements
2^{16}	16 bits	$16.2^{16}.2^{16} = 64$ GB
2^{18}	18 bits	$18.2^{18}.2^{18} > 1$ TB

Table 4.9: Storage requirements for the Caley table a general quasigroup

hash value length. Nowadays the hash value length considered to be secure is 160 - 256 bits.)

Several tricks can be used to overcome the storage requirements problem. They are connected to known results on critical sets in Latin squares (see Section 4.3). However, such an approach would significantly slow down the speed of computation of the hash value.

A better approach is to find a special large quasigroup $(Q, *)$. The operation $*$ in such a quasigroup should be given by some "easy to evaluate" expression, i.e. $a * b = f(a, b)$, $a, b \in Q$. One of the general security requirements that $f(a, b)$ has to satisfy is that given the value $f(a, b)$ and the element a, it should be computationally infeasible to find the element b, such that $a * b = f(a, b)$. In other words, it should be computationally infeasible to find the Caley table of the quasigroup $(Q, \backslash)$.

In order to overcome the storage requirements for the Caley table, a special quasigroup, namely the quasigroup of modular subtraction, was proposed in [21], [22] to be used. The operation * defined on Q is then given as

$$a * b = a + (n - b) \bmod n, \qquad n = \|Q\|.$$

*	0	1	2	3
0	0	3	2	1
1	1	0	3	2
2	2	1	0	3
3	3	2	1	0

Table 4.10: Multiplication table in the quasigroup of modular subtraction, $n = 4$

Usage of such an "easy to evaluate" expression for the definition of the operation * on quasigroup allows us to use quasigroups with a very large number of elements. Moreover, the isotopism of quasigroups gives us the power to use a large number of isotopic quasigroups where the computation of a hash value will be done almost only using the mentioned "easy to evaluate" expression. Later we show that it has also a severe impact on the security of the studied hash function.

Definition 4.4.3 *[18] Let $(G, .)$ and $(H, *)$ be two quasigroups. An ordered triple (θ, φ, ψ) of one-to-one mappings θ, φ, ψ of the set G onto the set H is called an isotopism of $(G, .)$ upon $(H, *)$ if $\theta(x) * \varphi(y) = \psi(x.y)$ for all x, y in G. The quasigroups $(G, .)$ and $(H, *)$ are then said to be isotopic.*

Definition 4.4.4 *Let $(G, .)$ and $(H, *)$ be two quasigroups. An ordered triple (θ, φ, ψ) of mappings θ, φ, ψ of the set G to the set H is called an homotopism of $(G, .)$ to $(H, *)$ if $\theta(x) * \varphi(y) = \psi(x.y)$ for all x, y in G. The quasigroups $(G, .)$ and $(H, *)$ are then said to be homotopic.*

Remark 4.4.5 *In [21], [22] the authors used a notion of homotopism of quasigroups, however in fact they used isotopism of quasigroups, because the mappings from one quasigroup to the another one (here denoted as θ, φ, ψ) were permutations.*

Attacks against the hash function

Let $a \in Q$ be the known parameter of the hash function and $\{m_1, m_2, \ldots m_k\}$, $m_i \in Q$, $1 \leq i \leq k$ be a message to be hashed. Let the hash value be

$$H_a(m_1, m_2, \ldots, m_k) = (((a * m_1) * m_2) * \ldots) * m_k = d.$$

Due to the simple construction of the hash function one can (in some cases easily) create false messages that hash to the same value. The false message can be constructed from the original message by adding prefix and/or suffix, changing some parts somewhere in the middle of the message, or it can be just a totally new message not based on the original message.

The false message created from the original one by adding prefix can be written as

$$p_1, p_2, \ldots, p_l, m_1, m_2, \ldots m_k, \qquad p_i \in Q, 1 \leq i \leq l.$$

Hence it must hold that $(((a * p_1) * p_2) * \ldots) * p_l = a$.

The false message created from the original one by adding suffix can be written as

$$m_1, m_2, \ldots m_k, s_1, s_2, ..., s_t \qquad s_i \in Q, 1 \leq i \leq t.$$

Hence it must hold that $(((d * s_1) * s_2) * \ldots) * s_t = d$.

Remark 4.4.6 *Only the last element of the last added, resp. changed part of the message has to be chosen in a proper way. It is important to mention that such an element always exists (because the Caley table of a quasigroup is a Latin square). All the other elements can be chosen arbitrarily, i.e. they can represent meaningful data.*

Due to the nice algebraic properties of the studied hash function it is possible to evaluate exactly the number of messages of a given length that hash to the same value.

Theorem 4.4.7 *Let $(Q, *)$ be a finite quasigroup and H_a be the hash function specified by the Construction 4.4.1. Then the number of messages $\{m_1, m_2, \ldots m_k\}$, $m_i \in Q$, $1 \leq i \leq k$ of length k that hash to the same value is $\|Q\|^{k-1}$.*

This Theorem can be prooved easily by induction. A straightforward consequence of this Theorem is the balancedness of the studied hash function.

Altering a part of the message or creating a new false message is similar to previous examples of false messages.

Remark 4.4.8 *While thinking about altering some parts of the original message one may ask how many elements may/must be changed in order to get the same hash value as the original message. It can be easily seen that changing a single element leads always to a different hash value from the hash value of the original message. Further it can be seen that changing more than one element in the original message always allows to reach the same hash value as the original message. Note that only the last element has to be chosen properly (see below).*

Remark 4.4.9 *Sketch of the proof of the preimage resistance of the studied hash fucntion was given in [22]. The problem of second preimage resistance and of collision resistance is in general of complexity at most $\|Q\|$ (see the construction of a false message given above, and also Remark 4.4.6).*

The question is how to find the last element of the last added/changed/created part of the false message? In other words, if we want to produce a false message by adding prefix to the original message, how to find p_l such that the following will hold $(((a * p_1) * p_2) * \dots) * p_l = a$?

For small instances of quasigroups with a "storable" Caley table it is possible to perform brute force attack (see Table 4.9). It is widely accepted that problems of complexity up to 2^{64} are nowadays solvable by exhaustive search. Thus the table implementation of the studied hash function is not secure.

In order to overcome the storage requirements for the Caley table, a special quasigroup, namely the quasigroup of modular subtraction, was proposed in [21], [22] to be used.

Lemma 4.4.10 *Quasigroup of modular subtraction contains a right unit 0.*

Corollary 4.4.11 *To construct a false message (if quasigroup of modular subtraction is used) one can insert an arbitrary number of 0s anywhere into the original message.*

Corollary 4.4.11 shows a trivial construction of false messages based on the insertion of a right unit. However, one can do much more.

Let we try to create a totally new message $x_1, x_2, \dots, x_v$, $x_i \in Q$, $1 \le i \le v$ that will hash to the value d. The elements $x_1, x_2, \dots, x_{v-1}$ can be chosen arbitrary. Let $d' = (((a*x_1)*x_2)\dots)*x_{v-1}$. It remains to find such a x_v that $d' * x_v = d$, which yields

$$x_v = d' + (n - d) \bmod n.$$

Theorem 4.4.12 *Hash function H_a with the quasigroup of modular subtraction is neither collision resistant, nor second preimage resistant.*

The attack may become much more difficult when a quasigroup isotopic to the quasigroup of modular subtraction is used for the hash function.

Example 4.4.13 *Let $(Q, *)$, $\|Q\| = 4$ be the quasigroup of modular subtraction with the Caley table given in Table 4.10. Let*

.	0	1	2	3
0	0	2	1	3
1	2	1	3	0
2	1	3	0	2
3	3	0	2	1

Table 4.11: Multiplication table of the quasigroup $(Q,.)$

$\theta = [1,2,3,0]$, $\varphi = [3,2,1,0]$ and $\psi = [2,0,3,1]$. The Caley table of the quasigroup $(Q,.)$ that is isotopic to the $(Q,*)$ is shown in Table 4.11.

For a fixed $a \in Q$ the hash value of a message $\{m_1, m_2, \dots, m_k\}$, $m_i \in Q$, $1 \leq i \leq k$ will now[2] be computed as $H_a(m_1, m_2, \dots, m_k) = ((\dots((a.m_1).m_2).\dots).m_{k-1}).m_k$.

A nice trick is that the quasigroup operation in $(Q,.)$ may also be written as

$$a.b = \psi^{-1}(\theta(a) + (n - \varphi(b)) \bmod n),$$

where $n = \|Q\|$, and ψ, θ, φ are the mappings that define the isotopism between the quasigroups $(Q,.)$ and $(Q,*)$.

In the previously described attacks, when a quasigroup of modular subtraction was used, the attacker was forced to solve the equation $a * b = d$ in a given quasigroup, where a and d are known, b is unknown. Formally, the solution can be written as $b = a \backslash d$, where $\backslash$ is the right inverse of $*$. For a quasigroup of modular subtraction one can write $b = a \backslash d = a * d$. When a quasigroup, isotopic to the quasigroup of modular subtraction, is used the attack leads to the equation $d = a.b = \psi^{-1}(\theta(a) + (n - \varphi(b)) \bmod n)$ for a given quasigroup, where a and d are known, and b is unknown. Hence, the security of the studied hash function, when a quasigroup isotopic to a quasigroup of modular subtraction is used, severely depends on the difficulty of inverting the mappings φ and ψ^{-1}. The mapping θ has no impact on the security of the studied hash function because in the previously described attacks the argument of this mapping is known.

Remark 4.4.14 *Note that any Latin square of prime power order is polynomial [31]. It is an open question if the results on polynomial*

[2]a quasigroup $(Q,.)$ isotopic to the quasigroup of modular subtraction $(Q,*)$ is used for the hash function

Latin squares or polynomial approximations of Latin squares can be used in attacks against the studied hash function.

There might be also another security problem. Let $(G,.)$ and $(H,*)$ be two isotopic quasigroups, i.e. there exist one-to-one mappings θ,φ,ψ of the set G onto the set H, such that $\theta(x)*\varphi(y)=\psi(x.y)$ for all x,y in G. However, there might exist other one-to-one mappings θ',φ',ψ' of the set G onto the set H, such that $\theta'(x)*\varphi'(y)=\psi'(x.y)$ for all x,y in G. For example, another triplet of mappings (θ,φ,ψ) that define isotopism between the quasigroups used in Example 4.4.13 is $\theta=[1,2,3,0]$, $\varphi=[2,1,0,3]$ and $\psi=[3,1,0,2]$. It might happen that even though the mappings φ and ψ^{-1} were "hard to invert", the mappings φ' and ψ'^{-1} were "easy to invert". However, finding such mappings θ',φ',ψ' may be difficult. Moreover, we tried to treat these mappings at a general level, i.e. we have not chosen any definite mappings. Neither in [21], [22] was the choice of the mappings θ,φ,ψ^{-1} studied.

We also performed exhaustive search experiments (for $\|Q\|=3,4,5,$ and 6) where we studied the number of one-to-one mappings θ,φ,ψ that define isotopism between the quasigroup of modular subtraction $(Q,\odot)$ and any quasigroup $(Q,.)$ isotopic to this quasigroup of modular subtraction. In all the experiments the number of one-to-one mappings θ,φ,ψ was $2\|Q\|^2$.

Modification to a keyed hash function – MAC

Assume, a in the H_a hash function is a secret key. H_a is then an MAC. Further assume that the quasigroup of modular subtraction was used. In the following we show how to create false messages, that will hash to the same value.

Let $m_1,m_2,\dots m_k$, $m_i\in Q$, $1\le i\le k$ be a message to be hashed. The hash value is then computed as follows:

$$H_a(m_1,m_2,\dots,m_k)=(((a*m_1)*m_2)*\dots)*m_k=d.$$

We can add suffix and construct such a message $\{m_1,m_2,\dots,m_k,s_1,s_2,\dots,s_u\}$, $s_i\in Q$, $1\le i\le u$ that will hash to the same result as the original message. Again the elements $s_1,s_2,\dots,s_{u-1}$ can be arbitrary (i.e. they can represent meaningful data), only s_u has to be calculated in a proper way. For the quasigroup of modular subtraction it is easy to do (see above presented construction of false messages).

It is possible to create the following false messages that will hash to the same value as the original message: take a new message, add suffix to the original message, or change some parts of the message. Adding only a prefix to the original message seems to be impossible due to the secret key a. However it is possible to add both a prefix and a suffix, or add a prefix and change some part of the original message.

Theorem 4.4.15 *Hash function H_a with the quasigroup of modular subtraction when used as MAC, with the secret key a, is neither collision resistant, nor second preimage resistant.*

A stronger result on the security of this MAC is as follows.

Theorem 4.4.16 *Construction of false messages for the hash function H_a when used as MAC, with the secret key a, is only as difficult as the construction of a false messages for the hash function H_a itself (a is public).*

Conclusions

There were some possible attacks against the hash function, proposed in [21],[22] shown in this section. Attacks were studied in a setting when a general (storable, i.e. small) quasigroup was used and also when a special (large) quasigroup, namely the quasigroup of modular subtraction was used. The security of the construction of a hash functions was studied both in the MDC and also in the MAC scenario. In all the cases it was possible to create false messages.

In order to make such a hash function useful in cryptology a very special quasigroup $(Q, *)$ has to be found. The multiplication in such a quasigroup should be given by some "easy to evaluate" expression $f(a, b)$, i.e. $a * b = f(a, b)$, $a, b \in Q$. Thus a "large" quasigroup could be used (without storing its multiplication table). Moreover, given a, resp. b and $f(a, b)$ it must be "difficult" (computationally infeasible) to find b, resp. a.

One of the ways how to achieve this is to use isotopic quasigroup to the quasigroup of modular subtraction, as it was proposed in [21], [22]. The security of the studied hash function then depends on the "difficulty" (i.e. computational infeasibility) of inverting the mappings φ and ψ^{-1} used in the isotopy, and is a topic for further research.

Chapter 5

Conclusion

We are convinced that the research targets assigned at the beginning of this dissertation were accomplished.

The state of the art in stream ciphers and hash fuctions is given in Section 3.

The results of the research are presented in Section 4. This section is based on the author's papers [85], [86], [87], [88], [89], [90], [91], [92] and [93].

Cryptographic properties of the concatenation of periods of several ml-pseudorandom sequences are studied in Section 4.1. The length of the period of a sequence obtained by periodic concatenation of two or more ml-sequences is determined. Moreover, a new construction of a completely equidistributed real valued sequence based on concatenation of ml-sequences is presented.

Section 4.2 deals with cryptanalysis of one stream cipher based on the concatenation of transformed runs of two ml-sequences. There are several theorems determining the number of runs in an ml-sequence presented in this section. The period of the keystream sequence of the cryptanalysed generator is determined as well as its basic statistical properties. The keystream sequence possesses good cryptographic properties such as long period and large linear complexity. The results of statistical tests are outlined. A known plaintext attack on the studied running key generator is proposed. The security of the generator against the known plaintext attack is generalized.

There are three successful attacks, namely chosen ciphertext, chosen plaintext and ciphertext-only attacks, against the self-synchronizing stream cipher (proposed in [59]) presented in Section 4.3. These attacks rank among the standard basic cryptanalyst techniques. Each of these attacks is much faster than the brute-force attack. We conclude that the cryptanalysed self-synchronizing stream cipher is insecure due to its vulnerability to the presented attacks.

The properties of one hash function based on a quasigroup (proposed in [21], [22]) are studied in Section 4.4. Some possible attacks against this hash function are presented. Attacks are studied in a setting when a general (storable, i.e. small) quasigroup is used and also when a special (large) quasigroup, namely the quasigroup of modular subtraction is used. The security of the construction of a hash functions is studied both in the MDC and also in the MAC scenario. In all the cases it was possible to create false messages. It was demostrated which mappings play an important role in the security of the studied hash function when a quasigroup isotopic to the quasigroup of modular subtraction is used. A possible weakness of isotopic mappings was found.

Chapter 6

List of author's publications, presentations, and other related activities

Scientific papers

- Vojvoda, M.: Cryptanalysis of a Clock-Controlled Running Key Generator, Journal of Electrical Engineering, Vol. 50 (1999), No. 10/s, pp.16-18.
- Vojvoda, M.: Enhanced Cryptanalysis of a Clock-Controlled Running Key Generator, Journal of Electrical Engineering, Vol. 51 (2000), No. 12/s, pp. 81-84.
- Vojvoda, M., Šimovcová, M.: Some Properties of Uniformly Distributed Sequences, Proceedings of abstracts from the conference Elitech 2000, (Vojvoda 50%, Šimovcová 50%).
- Vojvoda, M., Šimovcová, M.: On Concatenating Pseudorandom Sequences, Journal of Electrical Engineering, Vol. 52 (2001), No. 10/s, pp.36-37, (Vojvoda 70%, Šimovcová 30%).
- Vojvoda, M.: A Survey of Security Mechanisms in Mobile Communication Systems, Tatra Mountains Mathematical Publications, Vol. 25 (2002), pp. 101-117.
- Vojvoda, M.: A Probabilistic Approach to Weight Complexity of Binary Sequences, Proceedings of Elitech 2001, FEI STU, Bratislava, 2002, pp.91-92.

- Šimovcová, M., Vojvoda, M.: Symmetric and Complementary Boolean Functions, Proceedings of Elitech 2001, FEI STU, Bratislava, 2002, pp. 89-90, (Vojvoda 30%, Šimovcová 70%).
- Vojvoda, M.: Cryptanalysis of a File Encoding System Based on Quasigroup, Journal of Electrical Engineering, Vol.54 (2003), No.12/s, pp.69-71.
- Vojvoda, M.: Cryptanalysis of One Hash Function Based on Quasigroup, accepted for publication in Tatra Mountains Mathematical Publications.
- Vojvoda, M.: Attacks on a File Encryption System Based on Quasigroup, Proceedings of the 6th Scientific Conference on Electrical Engineering and Information Technology for PhD students - Elitech 2003, FEI-STU 2003, pp.54-56.
- Vojvoda, M.: On One Hash Function Based on Quasigroup, Proceedings of the Conference "Mikulášská kryptobesídka", ecom-monitor.com 2003, pp.23-28.

Lecture Notes

- Akantis,D., Grošek,O., Nemoga,K., Satko,L., Vojvoda,M.: CRYPTOLOGY: The Elements and Applications in Banking VIII., FEI-STU 2001, Lecture Notes, 86 pages.
- Grošek,O., Nemoga,K., Satko,L., Strnád,O., Šrámka,M., Vojvoda,M.: CRYPTOLOGY: The Elements and Applications in Banking IX., FEI-STU 2002, Lecture Notes, 152 pages.
- Grošek,O., Nemoga,K., Oravec,P., Satko,L., Šiška,J., Vávra,A., Vojvoda,M., Zanechal,M.: CRYPTOLOGY: The Elements and Applications in Banking X., FEI-STU 2003, Lecture Notes, 123 pages.

Grants, Technical Reports, and Research Projects

- Co-researcher of the grant "Methods and resources of obtaining, representing, presenting and searching of information and knowledge", VEGA 1/7611/20, Principal researcher: Professor Ing. Vladimír Vojtek, PhD. (years 2001 – 2002).

- Co-researcher of the grant "Information processing in the distributed environment of intelligent mobile agents", VEGA 1/0161/03, Principal researcher: Professor Ing. Vladimír Vojtek, PhD. (since 2003).
- Co-researcher of 9 research projects and co-author of 9 technical reports for the National Security Authority of the Slovak Republic (since 2001).

Conference Presentations and Seminar Lectures

- Cryptanalysis of a Clock-Controlled Running Key Generator, talk at SCAM 1999, Bratislava, Slovak Republic.
- Enhanced Cryptanalysis of a Clock-Controlled Running Key Generator, talk at SCAM 2000, FEI STU, Bratislava, Slovak Republic.
- Some Properties of Uniformly Distributed Sequences, joint work with M.Šimovcová, talk at ELITECH 2000, Bratislava, Slovak Republic.
- Decision Tree Attack, talk at the CRYPTO seminar, june 2000, FEI-STU, Bratislava, Slovak Republic.
- Some Properties of Uniformly Distributed Sequences, talk at the CRYPTO seminar, october 2000, FEI-STU, Bratislava, Slovak Republic.
- 2-adic Numbers and Sequences, 3 talks at the CRYPTO seminar, november 2000, FEI-STU, Bratislava, Slovak Republic.
- On Concatenating Pseudorandom Sequences, joint work with M.Šimovcová, talk at SCAM 2001, Bratislava, Slovak Republic.
- Stream Ciphers, 4 talks at the CRYPTO seminar, april 2001, FEI-STU, Bratislava, Slovak Republic.
- Mobile Communications and Security, talk at TATRACRYPT 2001, Liptovský Ján, Slovak Republic.
- Security in E–Business, talk at the BEST summer course, August 15, 2001, FEI STU, Bratislava, Slovak Republic.

- A Probabilistic Approach to Weight Complexity of Binary Sequences, talk at Elitech 2001, Bratislava, Slovak Republic.
- Identification Protocols Secure Against Reset Attacks, talk at the seminar "CRYPTOLOGY: The Elements and Applications in Banking VIII.", 2001, FEI-STU, Bratislava, Slovak Republic.
- Does Encryption With Redundancy Ensure Authenticity, talk at the seminar "CRYPTOLOGY: The Elements and Applications in Banking VIII.", 2001, FEI-STU, Bratislava, Slovak Republic.
- Cryptanalysis of the MD4 Hash Function, talk at the CRYPTO seminar, March 13, 2002, FEI-STU, Bratislava, Slovak Republic.
- Some Problems Concerning Latin Squares and Their Cryptographic Applications, talk at HAJDUCRYPT 2002,Debrecen, Hungary.
- Cryptanalysis of the Self-Shrinking Generator, talk at SCAM 2002, Bratislava, Slovak Republic.
- Digital Signatures, talk for the Lazar Consulting Company, October 2002, BCPB, Bratislava, Slovak Republic.
- Order of the National Security Authority of the Slovak Republic pursuant to the Electronic Signature Law, talk at the CRYPTO seminar, October 9, 2002, FEI-STU, Bratislava, Slovak Republic.
- New Stream Ciphers, talk at the seminar "CRYPTOLOGY: The Elements and Applications in Banking IX.", 2002, FEI-STU, Bratislava, Slovak Republic.
- Attacks on the A5 Stream Cipher, talk at the seminar "CRYPTOLOGY: The Elements and Applications in Banking IX.", 2002, FEI-STU, Bratislava, Slovak Republic.
- Cryptanalysis of a File Encoding System Based on Quasigroup, talk at ISCAM 2003, Bratislava, Slovak Republic.
- Cryptanalysis of One Hash Function Based on Quasigroup, talk at TATRACRYPT 2003, Bratislava, Slovak Republic.

- New Approaches to Digital Evidence, talk at the CRYPTO seminar, October 22, 2003, FEI-STU, Bratislava, Slovak Republic.
- Attacks on a File Encryption System Based on Quasigroup, talk at Elitech 2003, Bratislava, Slovak Republic.
- Using Hard Artificial Intelligence Problems in Security, 2 talks at the CRYPTO seminar, November 26, 2003, and December 3, 2003, FEI-STU, Bratislava, Slovak Republic.
- On One Hash Function Based on Quasigroup, talk at the Conference "Mikulášská kryptobesídka", Prague, Czech Republic.
- New Approach to Timestamping, talk at the seminar "CRYPTOLOGY: The Elements and Applications in Banking X.", 2003, FEI-STU, Bratislava, Slovak Republic.

Supervised theses and projects

- Jurenka, M.: Stream ciphers for software applications, Diploma thesis (2003/04), diploma project (2002-2003).
- Bálik, M.: Attacks on A5 algorithm, Diploma project (2003/04).
- Klenovič, L.: New attacks on stream ciphers, Diploma project (2003/04).
- Balvan, R.: Security of mobile agents, Diploma project (2002-03), Diploma thesis (2003).
- Bučka, A.: Attacks on stream ciphers, Diploma thesis (2003).
- Trgala, Š.: Usage of additive generators in cryptography, Bachelor's project (2002/03).
- Klenovič, L.: Security mechanisms in UMTS, Bachelor's project (2002/03).
- Zeman, J.: A note on row-complete latin squares, Students' scientific project - ŠVOČ (2001/02).
- Vadovič, P.: Cryptanalysis of the parity generator based on LFSRs and FCSRs, Diploma project (2000/01), Diploma thesis (2001).

- Repčík, P.: Cryptanalysis of the treshold generator based on LFSRs and FCSRs, Diploma project (2000/01), Diploma thesis (2001).

Bibliography

[1] Anderson, R.: The Classification of Hash Functions, Codes and Ciphers - Cryptography and Coding IV, 1995, pp.83–93.

[2] Bakhtiari, S., Safavi-Naini, R., Pieprzyk, J.: A Message Authentication Code Based on Latin Squares, Proceedings of ACISP '97, LNCS 1270, Springer-Verlag 1997, pp.194–203.

[3] Beth, T., Piper, F.C.: The Stop–and–go Generator, Advances in Cryptology – EUROCRYPT '84 Proceedings, LNCS 209, Springer–Verlag 1985, pp.88–92.

[4] Birkhoff, G., Bartee, T.C.: Applied Algebra (Aplikovaná algebra), Bratislava, Alfa 1981 (in Slovak).

[5] Biryukov, A., Shamir, A.: Real Time Cryptanalysis of the Alleged A5/1 on a PC, preliminary draft, December 9, 1999.

[6] Biryukov, A., Shamir, A., Wagner, D.: Real Time Cryptanalysis of A5/1 on PC, FSE 2000, LNCS 2365, Springer–Verlag, pp.1–18.

[7] Blum, L., Blum, M., Shub, M.: A Simple Unpredictable Pseudo-Random Number Generator, SIAM Journal on Computing, Vol.15 (1986), No.2, pp.364–383.

[8] den Boer, B., Bosselaers, A.: Collision for the Compression Function of MD5, Advances in Cryptology - EUROCRYPT '93 Proceedings, LNCS 765, Springer–Verlag 1994, pp.293–304.

[9] Boyar, J.: Inferring Sequences Produced by Pseudo-Random Number Generators, J.Assoc.Comput.Mach., Vol.36 (1989), pp.129–141.

[10] Bučka, A.: Attacks on stream ciphers, diploma thesis, FEI-STU 2003.

[11] Chan, A.H., Games, R.A.: On the Quadratic Spans of Periodic Sequences, Advances in Cryptology - CRYPTO '89 Proceedings, LNCS 435, Springer–Verlag 1990, pp.82–89.

[12] Chepyzhov, V., Smeets, B.: On a Fast Correlation Attack on Certain Stream Ciphers, Advances in Cryptology – EUROCRYPT '91 Proceedings, LNCS 547, Springer–Verlag 1991, pp.176–185.

[13] Coppersmith, D., Halevi, S., Jutla, C.: Cryptanalysis of Stream Ciphers with Linear Masking, `http://eprint.iacr.org/2002/020.ps`, 15.4.2004.

[14] Curran, D., van Rees, G.H.J.: Critical Sets in Latin Squares, Proc. Eight Manitoba Conf. on Numerical Math. and Comput., Congressus Numerantium, Vol.23 (1979), pp.165–168.

[15] Cusick, T.W., Ding, C., Renvall, A.: Stream Ciphers and Number Theory, Elsevier Science B.V. 1998.

[16] Damgård, I.B.: A Design Principle for Hash Functions, Advances in Cryptology - CRYPTO '89 Proceedings, LNCS 435, Springer–Verlag 1990, pp.416–427.

[17] Dawson, E., Clark, A., Golič, J., Millan, W., Penna, L., Simpson, L.: The LILI–128 Keystream Generator. NESSIE submission, Proc. 1st Open NESSIE Workshop (Leuven, November 2000), `http://cryptonessie.org/`.

[18] Dénes, J., Keedwell, A.D.: Latin Squares and Their Applications, Academic Press, NY 1974.

[19] Dobbertin, H., Bosselaers, A., Preneel, B.: RIPEMD-160: A Strengthened Version of RIPEMD, Fast Software Encryption, LNCS 1039, Springer-Verlag 1996, pp.71–82.

[20] Dobbertin, H.: Cryptanalysis of MD4, Journal of Cryptology, Vol.11 (1998), No.4, pp.253–271. See also Fast Software Encryption, LNCS 1039, Springer-Verlag 1996, pp.53–69.

[21] Dvorský, J., Ochodková, E., Snášel, V.: Hash Function Based on Quasigroups ("Hashovací funkce založená na kvazigrupách"), *Proc. of Mikulášská kryptobesídka*, Praha, pp. 27-36, 2001 (in Czech).

[22] Dvorský, J., Ochodková, E., Snášel, V.: Hash Functions Based on Large Quasigroups, *Proc. of Velikonoční kryptologie*, Brno, pp. 1-8, 2002.

[23] FIPS PUB 140–1, Federal Information Processing Standard Publication, Security Requirements for Cryptographic Modules, National Institute of Standards and Technology (NIST), U.S. Department of Commerce, Washington D.C.

[24] FIPS PUB 140–2, Federal Information Processing Standard Publication, Security Requirements for Cryptographic Modules, National Institute of Standards and Technology (NIST), U.S. Department of Commerce, Washington D.C., May 25, 2001.

[25] FIPS PUB 180–1, Federal Information Processing Standard (FIPS), Secure Hash Standard, National Institute of Standards and Technology (NIST), U.S. Department of Commerce, Washington D.C., April 17, 1995.

[26] Ford, L.R.Jr.: A Cyclic Arrangement of m-Tuples, Rand Corporation, Santa Monica, California, 1957, report 1071.

[27] Gligoroski, D., Markovski, S., Bakeva, V.: On Infinite Class of Strongly Collision Resistant Hash Functions "EDON-F" with Variable Length of Output, Proceedings of 1st International Conference On Mathematics and Informatics for Industry, April 2003, Thessaloniki, Greece.

[28] Golic, J.Dj.: Cryptanalysis of Alleged A5 Stream Cipher, Advances in Cryptology – EUROCRYPT '97 Proceedings, LNCS 1233, Springer–Verlag 1997, pp.239–255.

[29] Gollmann, D.: Pseudo Random Properties of Cascade Connections of Clock Controlled Shift Registers, Advances in Cryptology – EUROCRYPT '84 Proceedings, LNCS 209, Springer–Verlag 1985, pp.93–98.

[30] Grošek, O.: On the Stability of Stream Ciphers (O stabilite prúdových šifier), Proceedings of the conference "Jesenný seminár z kryptoanalýzy", Liptovský Mikuláš, October 9–11, 1996, pp.14–26 (in Slovak).

[31] Grošek, O., Horák, P., van Tran, T.: On Non-Polynomial Latin Squares, accepted for publication in Design, Codes and Cryptography, Kluwer Academic Publishers.

[32] Grošek, O., Porubský, Š.: Cryptology - Algorithms, Methods, Practice (Šifrovanie - algoritmy, metódy, prax), Praha, Grada 1992 (in Slovak).

[33] Halevi, S., Coppersmith, D., Jutla, C.: Scream: a Software-Efficient Stream Cipher, `http://eprint.iacr.org/2002/019.ps`, 15.4.2004.

[34] Horák, P., Aldred, R.E.L., Fleischner, H.: Completing Latin Squares: Critical Sets, Journal of Combinatorial Designs, Vol.10 (2002), pp. 419–432.

[35] Johansson, T., Jönsson, F.: Improved Fast Correlation Attacks on Stream Ciphers via Convolutional Codes, Advances in Cryptology – EUROCRYPT '99, LNCS 1592, Springer–Verlag 1999, pp.347–362.

[36] Johansson, T., Jönsson, F.: Fast Correlation Attacks Based on Turbo Code Techniques, Advances in Cryptology – CRYPTO '99, LNCS 1666, Springer–Verlag 1999, pp.181–197.

[37] Johansson, T., Jönsson, F.: Fast Correlation Attacks through Reconstruction of Linear Polynomials, Advances in Cryptology – CRYPT0 '2000, LNCS 1880, Springer–Verlag 2000, pp.300–315.

[38] Jungnickel, D.: Finite Fields. Structure and Arithmetics, B.I.Wissenschaftsverlag, 1992.

[39] Kaneko, T.: Report on Evaluation of Symmetric-Key Cryptographic Techniques, `http://www.ipa.go.jp/security/enc/CRYPTREC/fy15/doc/005_kaneko.pdf`, (April 15, 2004).

[40] Kim, S.-J., Umeno, K., Hasegawa, A.: Corrections of the NIST Statistical Test Suite for Randomness, `http://eprint.iacr.org/2004/018.ps`, (April 15,2004).

[41] Klapper, A.: Feedback with Carry Shift Registers over Finite Fields, K.U.Leuven Workshop on Cryptographic Algorithms, Springer-Verlag 1995, pp.170–178.

[42] Klapper, A., Goresky, M.: Large Period Nearly de Bruijn FCSR Sequences, Advances in Cryptology - EUROCRYPT '95 Proceedings, LNCS 921, Springer-Verlag 1995, pp.263–273.

[43] Knudsen, L.R.: Block Ciphers – Analysis, Design, Applications. Ph.D. dissertation, Aarhus University, November 1994.

[44] Knuth, D.E.: Construction of a Random Sequence, Nordisk tidskrift for informationbehandling, Vol.5 (1965), No.4, pp.246–250.

[45] Knuth, D.E.: The Art of Computer Programming, Vol.2 Seminumerical Algorithms, Addison-Wesley 1969.

[46] Knuth, D.E.: Deciphering a Linear Congruential Encryption, IEEE Transactions on Information Thoery, Vol.IT–31 (January 1985), No.1.

[47] Krawczyk, H.: How to Predict Congruential Generators, J.Algorithms, Vol.13 (1992), pp.527–545.

[48] Lidl, R., Niederreiter, H.: Introduction to Finite Fields and Their Applications, Cambridge University Press, 1994, Revised Edition.

[49] Markovski, S., Gligoroski, D., Andova, S.: Using Quasigroups for One–One Secure Encoding, Proceedings of LIRA '97 – Novi Sad Yugoslavia.

[50] Markovski, S., Gligoroski, D., Bakeva, V.: Quasigroups and Hash Functions, Proceedings of the 6th International Conference on Discrete Mathematics and Applications, Bansko, Bulgaria, South-West University, Blagoevgrad, Bulgaria.

[51] Massey, J.L.: Shift-Register Synthesis and BCH Decoding, IEEE Transactions on Information Theory, Vol.IT–15 (January 1969), No.1.

[52] Maurer, U.: An Universal Statistical Test for Random Bit Generators, Advances in Cryptology - CRYPTO '90 Proceedings, LNCS 537, Springer-Verlag 1991, pp.409–420.

[53] Meier, W., Staffelbach, O.: Fast Correlation Attacks on Certain Stream Ciphers, Journal of Cryptology, Vol.1 (1989), pp.159–176.

[54] Menezes, A., van Oorschot, P., Vanstone, S.: Handbook of Applied Cryptography, CRC Press 1996, `http://www.cacr.math.uwaterloo.ca/hac`.

[55] Mihajlević, M.J., Golić, J.D.: A Fast Iterative Algorithm for a Shift Register Initial State Reconstruction Given the Noisy Output Sequence, Advances in Cryptology – AUSCRYPT '90, LNCS 453, Springer–Verlag 1990, pp.165–175.

[56] Mihajlević, M.J., Golić, J.D.: A Comparison of Cryptanalytic Principles Based on Iterative Error–Correction, Advances in Cryptology – EUROCRYPT '91, LNCS 547, Springer–Verlag 1991, pp.527–531.

[57] Nemoga, K.: Linear Recurring Sequences (Lineárne rekurentné postupnosti), Proceedings of the conference "Jesenný seminár z kryptoanalýzy", Liptovský Mikuláš, October 9–11, 1996, pp.1–13 (in Slovak).

[58] NIST: AES Initiative, `http://www.nist.gov/aes` (May 25, 2001).

[59] Ochodková, E., Snášel, V.: Using Quasigroups for Secure Encoding of File System, Proceedings of the International Scientific NATO PfP/PWP Conference "Security and Information Protection 2001", May 9–11, 2001, Brno, Czech Republic, pp.175–181.

[60] Order of the National Security Authority of the Slovak Republic pursuant to the Electronic Signature Law No. 537, 2002.

[61] Preneel, B.: Analysis and Design of Cryptographic Hash Functions, Doctoral Dissertation, Katholieke Universiteit Leuven, 1993.

[62] Preneel, B., Rijmen, V., Bosselaers, A.: Recent Developments in the Design of Conventional Cryptographic Algorithms, Computer Security and Industrial Cryptography, State of the Art and Evolution, LNCS 1528, Springer-Verlag, 1998, pp.106–131.

[63] Preneel, B.: The State of Hash Functions, Cryptology and Information Security, Proceedings of VI RECSI, Teneriffe, Spain, September 2000, RA-MA, Madrid, 2000, pp. 3-27.

[64] Preneel, B.: New European Schemes for Signatures, Integrity and Encryption (NESSIE): A Status Report, Proceedings of Mikulášská kryptobesídka 2001, ecom-monitor.com 2001, pp.7–17.

[65] Project CRYPTREC, `http://www.ipa.go.jp/security/enc/CRYPTREC/index-e.html`, (April 19, 2004).

[66] Project CRYPTREC 2002-2003, `http://www.ipa.go.jp/security/enc/CRYPTREC/fy15/cryptrec20030620_repe.html`, (April 19, 2004).

[67] Project ECRYPT - European Network of Excellence in Cryptology, `http://www.ecrypt.eu.org`, (April 19, 2004).

[68] Project NESSIE, `http://www.cryptonessie.org`.

[69] Repčík, P.: Cryptanalysis of the treshold generator based on LFSRs and FCSRs, diploma thesis, FEI-STU 2001.

[70] RIPE: Integrity Primitives for Secure Information Systems, Final report of RACE Integrity Primitives Evaluation (RIPE-RACE 1040), LNCS 1007, Springer-Verlag, 1995.

[71] Rogaway, P., Coppersmith, D.: A Software-Optimized Encryption Algorithm, Fast Software Encryption Proceedings, LNCS 809, Springer–Verlag 1994, pp.56–63.

[72] Rueppel, R.A.: The Analysis and Design of Stream Ciphers, Berlin, Heidelberg, Springer–Verlag, 1986.

[73] Rueppel, R.A.: When a Shift Registers Clock Themselves, Advances in Cryptology – EUROCRYPT '87 Proceedings, LNCS 304, Springer–Verlag 1988, pp.53–64.

[74] Rueppel, R.A.: Security Models and Notions for Stream Ciphers, Cryptography and Coding II, C.Mitchell, ed., Oxford:Clarendon Press 1992, pp.213–230.

[75] Rueppel, R.A.: Stream Ciphers, Contemporary Cryptology: The Science of Information Integrity, Editor:G.J.Simmons, IEEE Press 1992, pp.65–134.

[76] Rukhin, A. et al.: A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. NIST Special Publication 800–22, May 15, 2001.

[77] Satko, L.: Correlation attack of Siegenthaler and Rueppel (Korelačný útok Siegenthalera a Rueppela), Proceedings of the conference "Jesenný seminár z kryptoanalýzy", Liptovský Mikuláš, October 9–11, 1996, pp.27–35 (in Slovak).

[78] Schneier, B.: Applied Cryptography. Protocols, Algorithms, and Source Code in C, John Wiley & Sons, Inc. 1996, Second Edition.

[79] Siegenthaler, T.: Correlation-Immunity of Nonlinear Combining Functions for Cryptographic Applications, IEEE Transactions on Information Theory, Vol.IT-30 (September 1984), No.5, pp.776–780.

[80] Siegenthaler, T.: Decrypting a Class of Stream Ciphers Using Ciphertext Only, IEEE Transactions on Computers, Vol.34 (January 1985), No.1, pp.81–85.

[81] Siegenthaler, T.: Cryptanalysts Representation of Nonlinearly Filtered ML-Sequences, Advances in Cryptology, Proc. of EUROCRYPT '85, LNCS 219, Springer-Verlag 1986, pp.103–110.

[82] Simpson, L.: Divide and Conquer Attacks on Shift Register Based Stream Ciphers, PhD thesis, Information Security Research Centre, Queensland University of Technology, Brisbane, Australia, November 1999.

[83] Stinson, D.R.: Cryptography: Theory and Practice, CRC Press 1995.

[84] Vadovič, P.: Cryptanalysis of the parity generator based on LFSRs and FCSRs, diploma thesis, FEI-STU 2001.

[85] Vojvoda, M.: Cryptanalysis of a Clock-Controlled Running Key Generator, Journal of Electrical Engineering, Vol.50 (1999), No.10/s, pp.16–18.

[86] Vojvoda, M.: Enhanced Cryptanalysis of a Clock-Controlled Running Key Generator, Journal of Electrical Engineering, Vol.51 (2000), No.12/s, pp.81–84.

[87] Vojvoda, M., Šimovcová, M.: Some Properties of Uniformly Distributed Sequences, Abstracts of the conference "Elitech 2000", FEI-STU, 2000.

[88] Vojvoda, M., Šimovcová, M.: On Concatenating Pseudorandom Sequences, Journal of Electrical Engineering, Vol.52 (2001), No.10/s, pp.36–37.

[89] Vojvoda, M.: A Probabilistic Approach to Weight Complexity of Binary Sequences, Proceedings of Elitech 2001, FEI STU, Bratislava, 2002, pp.91–92.

[90] Vojvoda, M.: Cryptanalysis of a File Encoding System Based on Quasigroup, Journal of Electrical Engineering, Vol.54 (2003), No.12/s, pp.69–71.

[91] Vojvoda, M.: Attacks on a File Encryption System Based on Quasigroup, Proceedings of the 6th Scientific Conference on Electrical Engineering and Information Technology for PhD students – Elitech 2003, FEI-STU 2003, pp.54–56.

[92] Vojvoda, M.: Cryptanalysis of One Hash Function Based on Quasigroup, accepted for publication in Tatra Mountains Mathematical Publications.

[93] Vojvoda, M.: On One Hash Function Based on Quasigroup, Proceedings of the Conference "Mikulášská kryptobesídka", ecom-monitor.com 2003, pp.23-28.

[94] Wagner, D., Simpson, L., Dawson, E., Kelsey, J., Millan, W., Schneier, B.: Cryptanalysis of ORYX, SAC '98, LNCS 1556, Springer–Verlag 1999, pp.296-305.

[95] Zeng, K.C., Huang, M.: On the Linear Syndrome Method in Cryptanalysis, Advances in Cryptology – CRYPTO '88, LNCS 403, Springer–Verlag 1990, pp.469–478.

[96] Zeng, K.C., Huang, M., Rao, T.R.N.: An Improved Linear Syndrome Algorithm in Cryptanalysis With Applications, Advances in Cryptology – CRYPTO '90, LNCS 537, Springer–Verlag 1991, pp.34–47.

[97] Živković, M.V.: An Algorithm for the Initial State Reconstruction of the Clock-Controlled Shift Register, IEEE Transactions on Information Theory, Vol.37 (September 1991), No.5, pp.1488–1490.