1 Stream Ciphers

Carl Schünemann (cas0597), Larysa Bondar (lab7449), Simon Thalmaier (sit7432)

Contents

1.1 Security of stream ciphers based on LFSRs

To evaluate the usability of encryption methods in real-world problems, multiple factors need to be analyzed, like the ease of implementation, performance and security. Based on the discussed technical realization, a pseudo-random bit stream can be generated. By applying the primitive polynomial to an LFSR, their output always has the largest possible period indifferent to their initial values of the memory cells. This allows for fast encryption of messages with unknown length [1, p. 181]. Further, the next bits of the keystream can be calculated in advance to improve processing speed [2, p. 3]. In computer hardware the LFSRs are efficiently implemented with shift registers [3]. These reasons established their wide usage in cryptographic contexts [4, p. 97]. For example, real-time audio and video data were encrypted with stream ciphers. Especially since there is no error propagation because if one bit of the stream differs from its correct value, the following bits are not defective [1, p. 181]. Nevertheless, the remaining main concern regarding stream ciphers is their security aspects.

1.1.1 Known-plaintext attack

In cryptanalysis, attacks can be categorized based on the data available to the adversary. Besides the ciphertext-only attacks and chosen-plaintext attacks, there is also the group of known-plaintext attacks. Here, the plaintext and its position of an encrypted sequence are laid open. Because of their linear nature, LFSR-based stream ciphers are prone to these known-plaintext attacks: Given the adversary has a segment of the encrypted message s and the corresponding plaintext p, the used keystream k can be reproduced by calculating $s_i \oplus p_i$. This is possible due to the mathematical laws of XOR like $b \oplus b = 0$ and $0 \oplus b = b$. The method can be especially abused for metadata like header fields since their structure and content are mostly known [5, p. 359]. Figure 1 illustrates the principle behind a basic known-plaintext attack.

$$
\begin{array}{lllll}
\textit{Given:} & s & = (s_0, s_1, ..., s_n) \in \mathbb{F}_2 & := \text{encrypted message} \\
& p & = (p_0, p_1, ..., p_n) \in \mathbb{F}_2 & := \text{plaintext} \\
& k & = (k_0, k_1, ..., k_n) \in \mathbb{F}_2 & := \text{keystream} \\
& f(p_i, k_i) & = k_i \oplus p_i = s_i & := \text{encryption function}
\end{array}
$$

$$
\textit{Attack:} \qquad s_i \oplus p_i = (k_i \oplus p_i) \oplus p_i = k_i \oplus 0 = k_i
$$

Figure 1: Basic known-plaintext attack on a stream cipher

If the period of a keystream is shorter than the gained segment of its plaintext, then the rest of the message can be decrypted [6, p. 9]. Therefore, a large period is necessary to diminish this threat [3, p. 83]. Even if it is not possible for the adversary to recreate the complete keystream period, the original data p can be replaced by malicious content p' of the same length. To demonstrate this, it is assumed that the position of the plaintext '10.000€' and its corresponding encrypted message s is known. The first digit is now replaced by a '9' in Figure 2. This is a clear security concern.

Sender (sends s):
$p = 00000001_{(2)} = 1_{(10)}$
$k = 10111110_{(2)}$
$s = k \oplus p = 10111110 \oplus 00000001 = 10110101$

Adversary (receives s, knows p and sends s'):
$$
\begin{aligned}
p' \quad & = 00001010_{(2)} = 9_{(10)} \\
k \oplus p' \quad & = s \oplus p \oplus p' \\
& = 10110101 \oplus 000000001 \oplus 00001010 \\
& = 10110100 \oplus 00001010 = 10111110 = s'
\end{aligned}
$$

Receiver (receives s'):
$k \oplus s' = 10111110 \oplus 101110101 = 00001001 = 9_{(10)}$

Figure 2: Replacing original data with modified text in a known-plaintext attack

1.1.2 Linear complexity and the Berlekamp-Massey algorithm

Besides the period of a sequence, linear complexity is also used as an indicator for the cryptographic usefulness of this sequence.

Definition: The linear complexity $L(s)$ of a finite binary sequence s is equal to the length and therefore degree of the shortest LFSR to generate s [1, p. 233]. L follows the properties [7, pp. 20-21]:

- s is the zero sequence with $(0, 0, ..., 0)$ $\Leftrightarrow$ $L(s) = 0$
- s is the zero sequence with $(0, 0, ..., 0)$ $\Leftrightarrow$ $L(s) = 0$
- s has length n with format $(0, 0, ..., 1)$ $\Leftrightarrow$ $L(s) = n$
- s cannot be generated by an LFSR $\Rightarrow$ $L(s) = \infty$
- s is periodic with period r $\Rightarrow$ $L(s) \leq r$
- s is the one-periodic sequence of a primitive feedback polynomial with degree n $\Rightarrow$ $L(s) = n$

The Berlekamp-Massey algorithm presented in the paper 'Shift-register synthesis and BCH decoding' can be used to calculate the linear complexity of a sequence and its corresponding shortest LFSR. Given a primitive polynomial has degree n and consequently, its generated period has linear complexity of L, if an adversary gains a sequence of the keystream $k \geq 2L$, the primitive polynomial can be determined [8, pp. 124-125]. The used LFSR can then be successfully asserted. So the linear complexity is directly connected to the required keystream sequence to crack the stream cipher.

Exploiting a known-plaintext attack, a finite sequence of the keystream can be obtained. This sequence can be used as input for the Berlekamp-Massey algorithm to try to recreate the LFSR generating the full period of the keystream [1, p. 232]. The algorithm has an efficient linear run time of $O(n)$ for a sequence with length n. Its structure is displayed in Figure 3.

s $= (s_0, s_1, ..., s_n) \in \mathbb{F}_2 :=$ keystream sequence of the LFSR and input for the algorithm
n $:=$ length of the input sequence
i $:=$ current index of the input sequence
i' $:=$ previous index since the last increment of the linear complexity
$C(x)$ $= 1 + c_1x^1 + c_2x^2 + ... + c_ix^i$ (mod 2)
$:=$ feedback connection polynomial of the minimal LFSR generating s
c_i $:=$ if tapped:$c_i = 1$else$c_i = 0$
$B(x)$ $:=$ previous connection polynomial since the last increment of the linear complexity
L $:=$ linear complexity of the minimal LFSR
d $:=$ discrepancy between the input and the output generated by $C(x)$

Berlekamp-Massey(s) :

$n = |s|$
$C(x) = B(x) = 1$
$L = i = 0$
$i' = -1$
while $i < n$:
    $d = s_i \oplus c_1s_{i-1} \oplus c_2s_{i-2} \oplus ... \oplus c_Ls_{i-L}$
    if $d == 1$:
        $C_{tmp}(x) = C(x)$
        $C(x) = C(x) + (B(x) * x^{i-i'})$
        if $L <= \frac{i}{2}$:
            $L = i + 1 - L$
            $i' = i$
            $B(x) = C_{tmp}(x)$
    $i = i + 1$
$return(L, C(x))$

Figure 3: Explanation and structure of the Berlekamp-Massey algorithm [8]

As a demonstration, the LFSR in Figure ? with characteristic polynomial $G(x) = 1 + x + x^4$ is uniquely determined by inputting its bit sequence into the Berlekamp-Massey algorithm. A characteristic polynomial $G(x)$ and connection polynomial $C(x)$ both represent the structure of an LFSR. However, they differ in their written mathematical form. The formula $G(x) = x^L * C(\frac{1}{x})$ describes their relation to each other. The expected output for the demonstration can be calculated by reversing the above equation, resulting in $C(x) = 1 + x^3 + x^4$. As input, the six-bit deciphered keystream $s = 110001$ is used which corresponds to the initial state $s_{12} = 12_{(10)} = 1100$ of the LFSR. The state and action of each iteration of the loop over the input sequence are presented in Figure 4.

Current values of attributes	Resulting attribute assignments
Input: $s = 110001$ $n = 6$	
$i = 0$ $i' = -1$ $L = 0$ $C(x) = 1$ $B(x) = 1$	d $\leftarrow 1$ $C(x)$ $\leftarrow 1 + 1 * x^1 = 1 + x$ L $\leftarrow 1$ $i' \leftarrow 0$ $B(x) \leftarrow 1$
$i = 1$ $i' = 0$ $L = 1$ $C(x) = 1 + x$ $B(x) = 1$	d $\leftarrow 0 \oplus 1 \odot 1 = 0$
$i = 2$ $i' = 0$ $L = 1$ $C(x) = 1 + x$ $B(x) = 1$	d $\leftarrow 0 \oplus 1 \odot 1 \oplus 1 \odot 0 = 1$ $C(x)$ $\leftarrow 1 + 1 * x^1 + (1 * x^{2-0}) = 1 + x + x^2$ L $\leftarrow 2$ $i' \leftarrow 2$ $B(x) \leftarrow 1 + x$
$i = 3$ $i' = 2$ $L = 2$ $C(x) = 1 + x + x^2$ $B(x) = 1 + x$	d $\leftarrow 0 \oplus 1 \odot 0 \oplus 1 \odot 1 \oplus 1 \odot 0 = 1$ $C(x)$ $\leftarrow 1 + x + x^2 + ((1 + x) * x^{3-2})$ $\leftarrow 1 + 2x + 2x^2 \pmod 2 = 1$
$i = 4$ $i' = 2$ $L = 2$ $C(x) = 1$ $B(x) = 1 + x$	d $\leftarrow 0$
$i = 5$ $i' = 2$ $L = 2$ $C(x) = 1$ $B(x) = 1 + x$	d $\leftarrow 1$ C(x) $\leftarrow 1 + ((1 + x) * x^{5-2}) = 1 + x^3 + x^4$ L $\leftarrow 5$ $i' \leftarrow 5$ $B(x) \leftarrow 1$
	Output: $C(x) = 1 + x^3 + x^4$ $L = 4$

Figure 4: Step-by-step execution of the Berlekamp-Massey algorithm

To again validate the gained connection polynomial, it is inserted into the equation $G(x) = x^L * C(\frac{1}{x}) = x^4 * (1 + \frac{1}{x^3} + \frac{1}{x^4}) = x^4 + x + 1$. The expected output was indeed computed correctly by the algorithm after six iterations. In the example, the adversary knew only six bits and recreated the LFSR successfully. In this case, she would have required at most $2 * L = 2 * 4 = 8$ bits for the algorithm to determine the correct connection polynomial. This can also be achieved with an equation as long as the length of the LFSR is known. To calculate which memory cell c_i is tapped, the following formula can be used [1, p. 232]:

$$\begin{pmatrix} s_{L-1} & s_{L-2} & \dots & s_1 & s_0 \\ s_L & s_{L-1} & \dots & s_2 & s_1 \\ \dots & \dots & \dots & \dots & \dots \\ s_{2L-3} & s_{2L-4} & \dots & s_{L-1} & s_{L-2} \\ s_{2L-2} & s_{2L-3} & \dots & s_L & s_{L-1} \end{pmatrix} * \begin{pmatrix} c_1 \\ c_2 \\ \dots \\ c_{L-1} \\ c_L \end{pmatrix} = \begin{pmatrix} s_L \\ s_{L+1} \\ \dots \\ s_{2L-2} \\ s_{2L-1} \end{pmatrix}$$

Since the degree of a primitive polynomial is equal to its linear complexity, even for an LFSR with a period of $2^{512} - 1$ only 1024 bits of the keystream are required to crack the stream cipher. Thus pure LFSRs are of no value as cryptographic tools due to their linear behavior [1, p. 231].

1.2 Increasing the cryptographic qualities of LFSRs

In an attempt to increase the security of LFSRs, several adjustments to their basic structure can be made [4, p. 97]. The discussed exploits mainly abuse their linear nature. Therefore, LFSRs can be combined with nonlinear transformations, for example, the multiplication of two bits by an AND function, to diminish their weak points. In this section, a few concepts based on this idea are expanded.

1.2.1 Nonlinear Feedback Shift Registers (NLFSR)

Instead of combining the keystream and plaintext linearly, NLFSRs utilize a nonlinear feedback function. This approach seems secure since there are 2^{2^n} possible Boolean functions for n bits. However, it is mathematically proven that every bit of a keystream with period r can be correctly determined after at most r bits. For example, the algorithm by Boyar and Krawczyk recursively computes the whole keystream after $n + m$ bits of plaintext, where n is the length of the NLFSR and m is the number of degrees of freedom of the feedback function. The required plaintext is generally much smaller than the period of the NLFSRs. There are even further restrictions that keep reducing their cryptographic values. To still guarantee the effective calculation of the nonlinear function, the amount of tapped cells is limited to a realistic sum. [9]

In this paper, the usage of pure NLFSRs is not further pursued because they cannot be securely used on their own. [4, p. 97]

1.2.2 Combining Linear Feedback Shift Registers with an output generator

Another solution is to combine the n output bits of multiple LFSRs based on a nonlinear function $f : \mathbb{F}_2^n \to \mathbb{F}_2$. A famous representative of this group is the *Geffe generator* which was developed by P.R. Geffe in 1973. It involves two LFSRs, whose outputs a and b are chained together by the nonlinear function $f(a_i, b_i, c_i) = k_i = a_i + c_i a_i + c_i b_i$, where c is produced by a third LFSR. The idea behind a Geffe generator is the selection of the keystream bit k_i based on c_i. If $c_i = 0$ then a_i is returned, else b_i is chosen. [10] Thus, the combination component is implemented as a *multiplexer* whose index value is c_i [2, p. 19]. The three LFSRs can also be of different length. This structure can be seen in Figure 5.

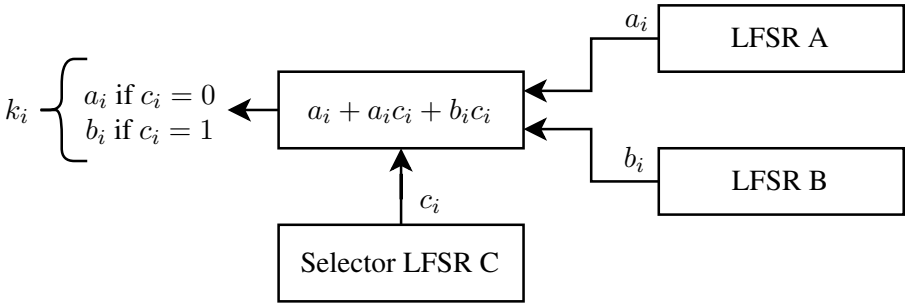

Figure 5: Graphic representation of a Geffe generator

This results in the following change in cryptographic metrics of the output sequence k: [1, p. 234]

$$\begin{aligned} \text{Linear Complexity } L_k &= L_A + (L_A * L_C) + (L_B * L_C) \\ \text{Period } r_k &= (2^{L_A} - 1) * (2^{L_B} - 1) * (2^{L_C} - 1) \end{aligned}$$

The increase in linear complexity and period improves the quality of the stream cipher. However, the Geffe generator has a statistical weakness that can be exploited. To prove this statement, its truth table is created in Figure 6.

a_i	b_i	c_i	k_i
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Figure 6: Truth table of a Geffe generator

It can be deducted, that an output bit k_i has the probability $p = \frac{3}{4}$ of being equal to the bit a_i of LFSR A. The same can be seen for LFSR B. Consequently, the Geffe generator can be easily broken with a correlation attack [4, p. 104].

A correlation attack is a brute-force plaintext attack in which the correlation between the final output keystream and the generated sequence of one of the LFSRs is exploited. First, a random initial state for LFSR A is produced. The resulting output of the system based on this guessed state is compared to the actual output stream. If the state is correct, then they should be equal in approximately $\frac{3}{4}$ths of their bits. If so, the next step is to calculate the initial state of LFSR B and afterwards repeated for LFSR C. The problem of estimating the initial states of all LFSRs at once is divided into determining only one at a time. This attack pattern is also called a divide-and-conquer attack. [2, p. 17] Without this approach the brute-force method requires roughly '$2^{L_A+L_B+L_C}$ times the total number of possible connection polynomials' operations to guess the three initial states. The first part of the equation is reduced to $2^{L_A} + 2^{L_B} + 2^{L_C}$, due to the separated calculation of the initial state by the

divide-and-conquer pattern. The probability of guessing the correct states for all LFSRs amplifies drastically. [1, p. 235]

To measure the qualities of a cipher concerning correlation attacks, a new term was introduced. A function is m^{th}-order correlation-immune when any subset of m input bits are uncorrelated to the output bit [11, p. 777]. In the case of a Geffe generator, the function is of 1st-order correlation-immune, since only the sequence of LFSR C has no influence on the output bits. Interestingly, high linear complexity L of the combination function results in low correlation-immunity m [11, p. 779].

1.2.3 Extending nonlinear output generators with memory cells

To counter the undesired effect between linear complexity and correlation-immunity, the combination function can be expanded with a memory component to achieve maximum linear complexity and correlation-immunity at the same time [2, p. 17]. The addition of memory turns the system into a nonlinear finite state machine [12, p. 209]. The combination function f is modified to not only produce the output bit k but also the next state of the machine σ.

The *summation combiner* uses this principle and allows for maximum linear complexity and correlation-immunity [13, p. 261]. The generated bits from two LFSRs are added together every tact and the resulting carry σ is saved in an one-bit memory, similar to an integer addition. This method proves to be highly nonlinear. The general structure is illustrated in Figure 7. [14, p. 70]

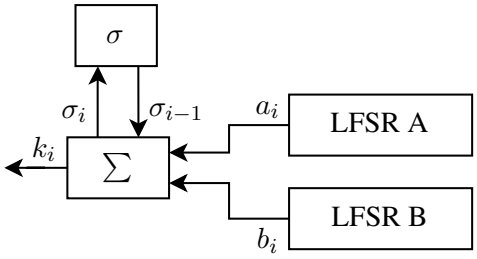

Figure 7: Basic overview of a summation combiner with two input LFSRs

Its combination functions are defined as:

$$\begin{aligned} f_k(a_i, b_i): \quad k_i &= a_i \oplus b_i \oplus \sigma_{i-1} && := \text{keystream bit} \\ f_\sigma(a_i, b_i): \quad \sigma_i &= a_i b_i \oplus a_i \sigma_{i-1} \oplus b_i \sigma_{i-1} && := \text{carry} \end{aligned}$$

The summation combiner accomplishes improvements in regards to the period r_k and linear complexity L_k of the produced keystream k. The period r_k is equal to $r_a * r_b$ for a integer addition of two sequences a and b with their corresponding periods r_a, r_b assuming $gcd(r_a, r_b) = 1$ [6, p. 220]. Also the gained linear complexity L_k is generally close to the period r_k with $L_k \leq r_a * r_b$ [6, p. 225]. In terms of the correlation between the state and its output, it still shows a slight bias which can be exploited by a correlation attack. This bias can be decreased by adding more LFSRs as input for the integer addition and increasing its memory [14, pp. 81-82].

1.2.4 Clock-Controlled Linear Feedback Shift Registers

Until this point in the paper, all registers were shifted on every clock tick. A different idea for refining the security characteristics of LFSRs is changing their tick rate based on the output of another register. This introduces a nonlinear nature to the keystream. The *Stop-and-Go generator* for example has two LFSRs A and B with their output a_i and b_i. The clock signal c_i is AND-combined with b_i before being connected to the LFSR A, so that the register is only shifted if $b_i = 1$. As long as $(b_i, b_{i+1}, ..., b_{i+n}) = (0, 0, ..., 0)$ the generator produces the unchanged bit a_i. [15, pp. 89-90] Since the Stop-and-Go generator suffers from a high correlation between b_i and a switch in the bits of the keystream, it is prone to correlation attacks [16, p. 156]. An improvement to this concept is the *shrinking generator*. Instead of outputting the same bit if $b_i = 0$, the LFSR A is shifted regardless but nothing is added to the keystream. This restricts the threat of correlation attacks [16, p. 159]. Even though the input clock signal is not directly tampered with, the shrinking generator counts towards the group of the clock-controlled stream ciphers [2, p. 23]. In Figure 8 the general structures of both generators are shown.

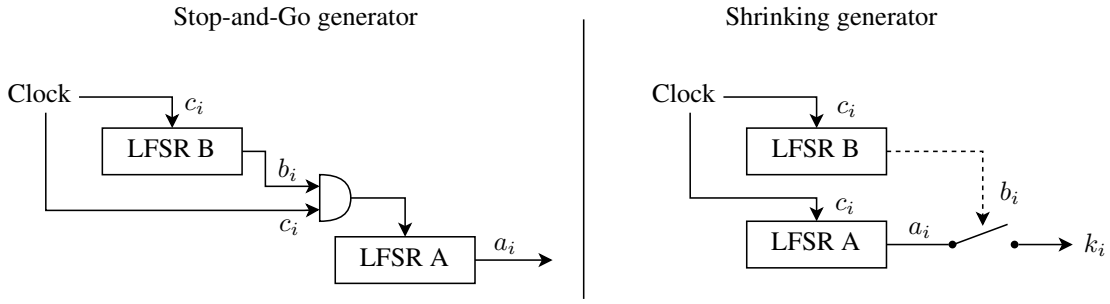

Figure 8: The Stop-and-Go [15, pp. 89-90] and shrinking generator [16, p. 159]

Let LFSR A have a maximal period of $r_A = 2^{L_A} - 1$ with linear complexity L_A and LFSR B has accordingly period $r_B = 2^{L_B} - 1$ with linear complexity L_B. On the condition that L_A and L_B are relative prime, meaning $gcd(L_A, L_B) = 1$, the period r_k of the output keystream k is equal to $r_k = 2^{L_B-1} * r_A = 2^{L_B-1} * (2^{L_A} - 1)$ [17, p. 25][16, p. 159]. Furthermore, the linear complexity of the output keystream L_k can be described as: $L_A * 2^{L_B-2} < L_k \leq L_A * 2^{L_A-1}$ [17, p. 25]. Because the shrinking generator only sends bits if $b_i = 1$, the time between the consecutive bits k_i and k_{i+1} can be measured. The stream of LFSR B can then be immediately reconstructed. These *timing attacks* are a specific variant of so called *side-channel attacks*. To lower the possibility of such attacks, dummy operations can be implemented or the actual bit b_i can be hidden by caching the output stream. [16, pp. 163-164]

The metrics to measure the cryptographic applicability of stream ciphers such as their period, linear complexity and correlation-immunity only indicate required properties to guarantee minimal security not their actual real-world usability [17, p. 24]. Over time a lot of stream ciphers have been cracked. For them to be reliable even today, more adjustments need to be made.

References

[1] Nigel P. Smart. *Cryptography made simple*. 1st edition. Information security and cryptography. Cham et al.: Springer, 2016. ISBN: 978-3-319-21936-3.

[2] Matthew JB Robshaw. "Stream ciphers". In: *RSA Labratories* 25 (1995). URL: http://www.networkdls.com/Articles/tr-701.pdf (visited on 04/29/2022).

[3] Mark Stamp and Richard M. Low. *Applied cryptanalysis: Breaking ciphers in the real world*. 1st edition. Hoboken, New Jersey: Wiley-Interscience a John Wiley & Sons Inc., 2007. ISBN: 978-0-470-1-1486-5.

[4] Klaus Pommerening. *Cryptology Part IV: Bitstream Ciphers*. 2000. URL: https://www.staff.uni-mainz.de/pommeren/Cryptology/Bitstream/ (visited on 05/13/2022).

[5] Claudia Eckert. *IT-Sicherheit: Konzepte - Verfahren - Protokolle*. 10th edition. Berlin/Boston: De Gruyter, 2018. ISBN: 978-3-11-055158-7. DOI: 10.1515/9783110563900.

[6] Rainer A. Rueppel. *Analysis and Design of Stream Ciphers*. 1st edition. Communications and Control Engineering Series. Berlin and Heidelberg: Springer, 1986. ISBN: 978-3-642-82865-2. DOI: 10.1007/978-3-642-82865-2.

[7] Thomas W. Cusick and Pantelimon Stănică. *Cryptographic Boolean functions and applications*. 1st ed. Amsterdam and Boston: Academic Press/Elsevier, 2009. ISBN: 978-0-08-095222-2. URL: http://site.ebrary.com/lib/alltitles/docDetail.action?docID=10286068.

[8] J. Massey. "Shift-register synthesis and BCH decoding". In: *IEEE Transactions on Information Theory* 15.1 (1969), pp. 122–127. ISSN: 0018-9448. DOI: 10.1109/TIT.1969.1054260.

[9] Klaus Pommerening. "Cryptanalysis of nonlinear feedback shift registers". In: *Cryptologia* 40.4 (2015), pp. 303–315. ISSN: 0161-1194. DOI: 10.1080/01611194.2015.1055385. (Visited on 05/13/2022).

[10] F. Handayani and N. P. R. Adiati. "Analysis of Geffe Generator LFSR properties on the application of algebraic attack". In: AIP Conference Proceedings. AIP Publishing, 2019, p. 020029. DOI: 10.1063/1.5132456.

[11] T. Siegenthaler. "Correlation-immunity of nonlinear combining functions for cryptographic applications (Corresp.)" In: *IEEE Transactions on Information Theory* 30.5 (1984), pp. 776–780. ISSN: 0018-9448. DOI: 10.1109/TIT.1984.1056949.

[12] Hongjun Wu. "Cryptanalysis and Design of Stream Ciphers". PhD thesis. 2008. URL: https://www.esat.kuleuven.be/cosic/publications/thesis-167.pdf (visited on 04/25/2022).

[13] Rainer A. Rueppel. "Correlation Immunity and the Summation Generator". In: *Advances in Cryptology — CRYPTO '85 Proceedings*. Ed. by Hugh C. Williams. Vol. 218. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 1986, pp. 260–272. ISBN: 978-3-540-16463-0. DOI: 10.1007/3-540-39799-X{\textunderscore}20. URL: https://link.springer.com/content/pdf/10.1007/3-540-39799-X_20.pdf (visited on 05/15/2022).

[14] Willi Meier and Othmar Staffelbach. "Correlation properties of combiners with memory in stream ciphers". In: *Journal of Cryptology* 5.1 (1992), pp. 67–86. ISSN: 0933-2790. DOI: 10.1007/BF00191322.

[15] T. Beth and F. C. Piper. “The Stop-and-Go-Generator”. In: *Advances in Cryptology*. Ed. by Thomas Beth, Norbert Cot, and Ingemar Ingemarsson. Vol. 209. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 1985, pp. 88–92. ISBN: 978-3-540-16076-2. DOI: 10.1007/3-540-39757-4{\textunderscore}9.

[16] Andreas Klein. *Stream ciphers*. London: Springer, 2013. ISBN: 9781447150787.

[17] Don Coppersmith, Hugo Krawczyk, and Yishay Mansour. “The shrinking generator”. In: *Annual International Cryptology Conference*. Springer. 1993, pp. 22–39.