METHODS AND TOOLS FOR ANALYSIS OF SYMMETRIC CRYPTOGRAPHIC PRIMITIVES

Oleksandr Kazymyrov

DISSERTATION FOR
THE DEGREE OF PHILOSOPHIAE DOCTOR

THE SELMER CENTER
DEPARTMENT OF INFORMATICS
UNIVERSITY OF BERGEN
NORWAY

DECEMBER 1, 2014

ACKNOWLEDGMENTS

It is impossible to thank all those who have, directly or indirectly, helped me with this thesis, giving of their time and experience. I wish to use this opportunity to thank some of them.

Foremost, I would like to express my very great appreciation to my main supervisor Tor Helleseth, who has shared his extensive knowledge and experience, and made warm conditions for the comfortable research in one of the rainiest cities in the world. I owe a great deal to Lilya Budaghyan, who was always ready to offer assistance and suggestions during my research. Advice given by Alexander Kholosha has been a great help in the early stages of my work on the thesis.

My grateful thanks are also extended to all my friends and colleagues at the Selmer Center for creating such a pleasant environment to work in. I am particularly grateful to Kjell Jørgen Hole, Matthew G. Parker and Håvard Raddum for the shared teaching experience they provided. Moreover, I am very grateful for the comments and propositions given by everyone who proofread my thesis. In addition, I would like to thank the administrative staff at the Department of Informatics for their immediate and exhaustive solutions of practical issues.

I wish to acknowledge the staff at the Department of Information Technologies Security, Kharkiv National University of Radioelectronics, Ukraine, especially Roman Oliynykov, Ivan Gorbenko, Viktor Dolgov and Oleksandr Kuznetsov for their patient guidance, enthusiastic encouragement and useful critiques.

I would like to offer my special thanks to my wife, who made invaluable contributions, including reading of the early versions of the thesis and making extremely beneficial and penetrating observations on the research results.

ABSTRACT

The development of modern cryptography is associated with the emergence of computing machines. Since specialized equipment for protection of sensitive information was initially implemented only in hardware, stream ciphers were widespread. Later, other areas of symmetric and asymmetric cryptography were established with the invention of general-purpose processors. In particular, such symmetric cryptographic primitives as block ciphers, message authentication codes (MACs), authenticated ciphers and others began to develop rapidly. Today various cryptographic algorithms are commonly used in everyday life to protect private data.

Design and analysis of advanced symmetric cryptographic primitives require a lot of time and resources. This is related to many factors, mainly to the cryptanalysis of prospective encryption algorithms under development. Every year new and modified attacks are published, leading to a rapid increase in the quantity of requirements and criteria imposed on cryptoprimitives.

Most of this thesis is devoted to analysis and improvement of cryptographic attacks and corresponding criteria for basic components. Almost all modern cryptoprimitives use nonlinear mappings for protection against advanced attacks. In connection with that a new method was proposed for the generation of random substitutions (S-boxes) with extreme cryptographic indicators that can be used in the next-generation ciphers to provide high and ultra-high security levels. In addition, several criteria imposed on S-boxes used in block ciphers were analyzed and their significance for block ciphers was proven. It is worth mentioning a practical method of testing two vectorial Boolean functions and a universal tool for checking properties of arbitrary binary nonlinear components presented in papers gathered in this thesis.

Another part of the thesis is dedicated to the cryptanalysis of hash functions as well as block and stream ciphers. To be more precise, an algebraic attack based on a binary decision diagram (BDD) was performed on the reduced Data Encryption Standard (DES), a scaled-down version of Advanced Encryption Standard (AES) and extended affine (EA) equivalence problem. Moreover, an algebraic approach was used to reconstruct an initial representation of the current Russian hash standard GOST 34.11-2012. Finally, a backward states tree method has been used to analyze stream ciphers based on the combination principle of linear and nonlinear feedback registers.

List of Papers

[I] Kazymyrov, O., Raddum, H.: Algebraic attacks using binary decision diagrams. *In Pre-proceedings of BalkanCryptSec 2014*, pp. 31–44, 2014.

[II] Eilertsen, A. M., Kazymyrov, O., Kazymyrova, V., Storetvedt, M.: A Sage library for analysis of nonlinear binary mappings. In *Pre-proceedings of Central European Conference on Cryptology (CECC14)*, pp. 69–78, 2014.

[III] Kazymyrov, O., Kazymyrova, V., Oliynykov, R.: A method for generation of high-nonlinear S-boxes based on gradient descent. In *Mathematical Aspects of Cryptography*, vol. 5, pp. 71–78. Steklov Mathematical Institute, 2014.

[IV] Kazymyrov, O., Kazymyrova, V.: Algebraic aspects of the Russian hash standard GOST R 34.11-2012. In *Pre-proceedings of 2nd Workshop on Current Trends in Cryptology (CTCrypt 2013)*, pp. 160–176, 2013.

[V] Kazymyrov, O., Kazymyrova, V.: Extended criterion for absence of fixed points. In *Pre-proceedings of 2nd Workshop on Current Trends in Cryptology (CTCrypt 2013)*, pp. 177–191, 2013.

[VI] Helleseth, T., Jansen, C.J.A., Kazymyrov, O., Kholosha, A.: State space cryptanalysis of the MICKEY cipher. In *Information Theory and Applications Workshop (ITA)*, pp. 1–10. Institute of Electrical and Electronics Engineers (IEEE), 2013.

[VII] Budaghyan, L., Kazymyrov, O.: Verification of restricted EA-equivalence for vectorial Boolean functions. In Özbudak, F., Rodríguez-Henríquez, F. (eds.), *Arithmetic of Finite Fields*, vol. 7369 of *Lecture Notes in Computer Science*, pp. 108–118. Springer Berlin Heidelberg, 2012.

Contents

Scientific Results

Introduction

1. The state-of-the-art of symmetric cryptology

One of the strategic priorities of any country is to adopt comprehensive measures to protect the national information space [1]. The main feature of this trend is to increase performance and to improve security in telecommunication systems. Fast and secure access to information and computing resources, most of which are a part of the Internet, may be regarded as one of the requirements of a developed country.

Information technologies (IT) are an essential part of our daily lives. Efficiency of application and operation of information systems depend on their security and reliability. There are many fields where unpredictable or abnormal operation of telecommunication systems may result in serious consequences. These include management and control systems of water, gas and energy supply; petroleum and nuclear industries; transport systems, etc. Over the past few decades the number of publications and projects related to different aspects of information security has considerably increased.

The emergence of new problems requires improved methods to solve them [2, 3]. Until recently, cryptographic tools were available only to special state authorities. Today they are used in everyday life in the process of creating, sending, receiving, processing, storing and destroying data [3, 4].

Block ciphers play an important role in complex information systems [3, 5]. They are widely used due to their high efficiency and low implementation complexity. In addition to providing confidentiality, block ciphers realize message authentication codes (MACs), hash functions, pseudorandom number generators (PRNG) and authentication protocols [3, 6]. Thus, block ciphers are used in most modern symmetric cryptographic primitives. Nonetheless, special algorithms have many advantages. For example, to provide secure high-speed transmission of information, especially when the data processing is in hardware, stream ciphers are used. Due to their structure they are optimized for hardware platforms by default. At the same time their performance can be ten times better than of block ciphers.

Many international competitions for choosing hash functions, block and stream ciphers have shown that the task of creating a secure cryptographic algorithm is rather complicated. For example, all stream ciphers submitted to the New European Schemes for Signatures, Integrity and Encryption (NESSIE) were theoretically broken [7]. At the same time, the role of the cryptographic community should not be underestimated. Every year more and more people invent new and advanced approaches to solve cryptographic problems.

In view of the above, the goal of this thesis is to improve the resistance of modern iterative cryptographic primitives to advanced attacks through the development of methods and tools of cryptanalysis.

1.1. Global development of symmetric cryptography

At the end of the 20th century a number of successful theoretical attacks allowed the block cipher DES to be broken [8]. A bit later practical implementations emerged to find the encryption key in a reasonable time [9]. As a consequence in the USA in 1997, the Advanced Encryption Standard (AES) competition was launched [10]. The main objective of the competition was the selection of a new generation block cipher as the standard. After several years of research the algorithm Rijndael was selected as a winner. This cipher was became the encryption standard FIPS-197, also known as AES [11]. Rijndael ranked first due to its high-level resistance against known attacks, simple description, and high performance on most platforms of that time.

A similar European open competition NESSIE was launched in February 2000 [5, 7]. The main task of the project was the selection of the best cryptographic primitives among submitted candidates from around the world. Security, performance, and flexibility were offered as the main criteria. After the competition a recommended list containing block ciphers, hash functions, MACs and digital signature algorithms for industrial usage was created [7].

Along with other cryptographic algorithms six stream ciphers were submitted to NESSIE [7]. All of them as mentioned above were theoretically broken. This led, in November 2004, to a separate project called eSTREAM, whose main task was to choose one or more stream ciphers for use in the business sector [12]. It should be mentioned that the stream ciphers were divided into two separate categories. While the first one consisted of software oriented primitives, another contained algorithms adapted for hardware applications. After four years of research, 4 ciphers were selected for each category. However, in 2008 the stream cipher F-FCSR-H v2 was excluded from the list because of vulnerabilities [13].

In parallel with NESSIE a similar research was carried out by the Japanese government under CRYPTREC [14]. As a result of this analysis the best algorithms were selected for data protection. As of today many cryptoprimitives have been recommended for use in both government (e.g., AES, Camellia, KCipher-2, ECDH, SHA-512, HMAC, etc.) and business (e.g., MISTY1, MUGI, SC2000, PC-MAC-AES, PSEC-KEM, etc.) sectors [15].

In post-Soviet states the block cipher GOST 28147 is used [16]. It was adopted in 1989 and has been outdone in performance, usability and other characteristics by modern ciphers, including AES. In the past few years theoretical attacks on this encryption algorithm have been successfully carried out. The complexity of finding the key was reduced from 2^{256} to 2^{225} [17, 18]. However, the complexity of 2^{225} is unachievable for modern computers so GOST 28147 remains practically secure[19].

However, long before the proposed attacks the cryptographic community and government agencies of these countries began to think about changing the encryption algorithm. Since 2003 Belarus has used a new standard for confidentiality and integrity [20]. It includes a block cipher and its modes of operation, a message authentication code and a hash function.

In order to find an alternative to GOST 28147 the State Service of Special Communication and Information Protection of Ukraine announced in 2006 an open competition to design a prototype of a block cipher for the new standard [21]. One of the main requirements for the prospective cipher was a high-level of resistance against known and promising types of cryptanalytic attacks. At the same time, it was necessary to achieve a performance not less than the previous standard. In practice the designers tried to beat AES. According to the results of the competition in 2009 the cipher Kalyna was allowed to be used for protection of nongovernmental information [3, 21]. This cipher with improvements is now undergoing a formal assessment, and is at the stage of adoption as the standard [22].

In November 2007 the National Institute of Standards and Technology (NIST) opened a competition to develop a hash function SHA-3, which would complement the existing two versions [23]. In analogy with AES, NIST teamed cryptanalysts and developers from around the world in order to select one or more additional hash algorithms. In October 2012 it was announced that SHA-3 will be based on the algorithm Keccak [24]. Two years later a draft version of a new standard was published [25].

Unlike the USA, Russia did not announce an open competition, and used the hash function Stribog (Streebog) as a prototype [26–28]. This algorithm is the only known version of the draft state standard. Since January 1st, 2013 GOST R 34.11-2012 came into effect, replacing the earlier version [29]. Further development of block ciphers in Russia was presented at CTCrypt'14 [30]. According to the article the current standard GOST 28147 will be used in hardware, and the new block cipher Kuznechik (Grasshopper) will target software.

A similar path has been chosen by Ukraine in the development of the hashing algorithm. Drawing on the experience gained from cryptanalysis of block ciphers and considering finished competitions, Grøstl was taken as a basis for the new hash function. Together with Keccak, Grøstl is one of five finalists of SHA-3 [24, 31]. The main difference of the Ukrainian hash function is the usage of Kalyna with 512-bit block and key length instead of AES in the functions P and Q [31]. As in the case of the block cipher, the hash function is at the final stage of the standardization procedure.

In recent years the question about improvement of methods providing security and integrity of transmitted data simultaneously has been increasingly raised. In connection with this, the Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR) was organized [32]. Over the next few years, cryptologists, and software and hardware specialists from all over the world will select a modern authenticated cipher.

1.2. General design ideas of cryptographic primitives

1.2.1. Block ciphers

Let $E : \{0,1\}^l \times \{0,1\}^k \mapsto \{0,1\}^l$ be a function which takes a key K of length k bits, an input message (plaintext) M of length l bits and returns an output message (ciphertext) $E(M,K)$. For each K let $E_K : \{0,1\}^l \mapsto \{0,1\}^l$ be a function defined by $E_K(M) = E(M,K)$. Then E is a block cipher if E_K and E_K^{-1} are efficiently computable and E_K is a permutation for every K.

Most modern block ciphers are iterative (Fig. 1). A round function is usually used multiple times with different parameters (round keys). An arbitrary iterative block cipher can be mathematically described as follows

$$E_K(M) = PW_{k_{r+1}} \circ \prod_{i=2}^{r} (R_{k_i}) \circ IW_{k_1}(M),$$

where R, IW and PW are a round routine, a prewhitening and a postwhitening routine, respectively. In Fig. 1 the key expansion is an algorithm that takes a master key K as input and produces the subkeys $k_1, k_2, \ldots, k_{r+1}$ for all stages of encryption.

A mixing key routine of a block cipher is an algorithm which injects a round key into an encryption routine. In the majority of modern block ciphers the mixing key function is implemented using the XOR operation because of its low-cost implementation.

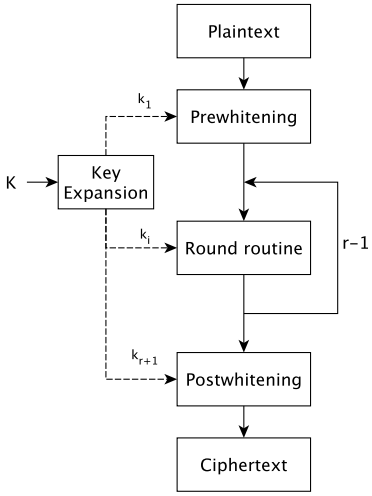


Fig. 1: *The general structure of an iterative block cipher*

To be an advanced algorithm, a modern block cipher should satisfy the following requirements [5]

- the complexity of the encryption and decryption has to be commensurate with the current standards;
- be protected against all known and prospective attacks;
- have high performance on widespread platforms.

It is quite challenging to satisfy the last point. Nevertheless, there are many publications regarding high performance implementations of AES. This is due to the fact that it is the most widespread block cipher and therefore the most optimized cryptographic algorithm for variety of platforms. However, getting into the Internet of things era, where devices communicate with each other via secure channels, it became necessary to have lightweight primitives. A lightweight cryptographic algorithm possesses a practical security level with enough performance in resource-limited settings: clock-cycles, area or energy [33].

1.2.2. AUTHENTICATED CIPHERS

There has been insufficient time to identify the generalized model of authenticated ciphers. There are only general structures such as encrypt-then-MAC, encrypt-and-MAC and MAC-then-encrypt [34]. Therefore, this section will focus on the general ideas and issues underlying these algorithms.

As mentioned earlier, in addition to confidentiality of transmitted information it is often required to ensure its integrity. Due to limitations of equipment and large amounts of processed data, the application of asymmetric cryptography for these purposes is not always possible. Therefore, an encrypted message is processed by a message authentication code to produce a tag [35]. Lots of modern MACs are based on block ciphers. These belong to the group of symmetric algorithms. Some of them are standardized and widely used in everyday life [36].

In general MACs only provide data integrity. Moreover, the complexities of the tag calculation and the message encryption are commensurable. In other words, to provide both confidentiality and integrity, two transformations of approximately equal complexity must be performed sequentially. In order to reduce the amount of transformations and system bandwidth, special algorithms have been developed [36]. The next generation authenticated cipher will be chosen after CAESAR.

Most authenticated schemes are nonce-based, i.e. an initialization vector (nonce) is transmitted together with data [37]. This solution helps to protect the algorithm against replay attacks and to use a pre-shared key for many messages. In addition, authenticated ciphers can operate in associated data mode [38]. This mode allows to encrypt only part of the data while the tag is generated for the entire message. This property is a useful addition in many situations where part of the message must be transmitted in plaintext. An Internet protocol (IP) packet is the most obvious example due to its widespread distribution. While the body of the packet can contain encrypted data, service information (e.g., data ports, IP addresses of sender and recipient, etc) has to be in clear to maximize data transfer speed.

From a security point of view the requirements imposed on authenticated ciphers include everything from block ciphers and message authentication codes [39]. Game theory is often used to prove the security of algorithms. However, all specific attacks applied to block ciphers and MACs can be easily adapted to authenticated ciphers (see Section 1.3). Security evaluation of

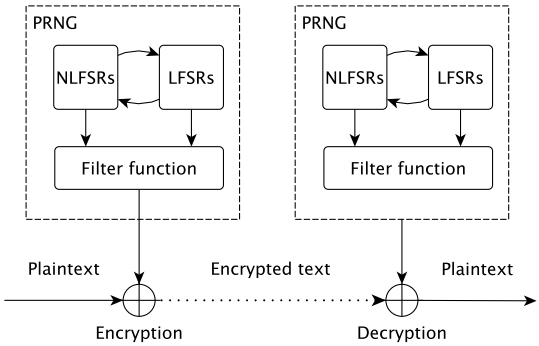


Fig. 2: *The overall structure of a stream cipher*

authenticated encryption algorithms therefore is more complex and consequently requires more resources.

1.2.3. STREAM CIPHERS

The main feature of stream ciphers is generation of random numbers (keystream) based on an initialization vector and key. Further, the plaintext is divided into chunks and added with the keystream using modulo operations to form the ciphertext. Since stream ciphers are typically targeted at hardware implementations, the XOR operation is often used instead of additional modulo [12, 40, 41].

Modern stream ciphers consist of linear and nonlinear feedback shift registers (LFSRs and NLFSRs), and a filter function to achieve maximum resistance against advanced attacks. Fig. 2 depicts the overall structure of a stream cipher.

In most cases, registers do not work independently, and operate in so-called mutual control mode. In other words, the states of the registers depend not only on their previous states, but also on other components of the cipher. If the keystream is generated randomly and without period, then an adversary cannot even theoretically recover the ciphertext [6, 42]. However, the practical application of such a scheme is too limited. Therefore, a key of fixed length is used to generate a pseudorandom sequence satisfying a number of criteria, including Golomb's randomness postulates [41].

Many designers of stream ciphers provide security proofs using a number of assumptions, which hypothetically could lead to vulnerabilities [12]. As a consequence the question regarding the theoretical proof of the security of NLFSRs-based ciphers remains open.

1.2.4. Cryptographic hash functions

A hash function is a method for mapping data of arbitrary size to a fixed-length value (hash code or hash value). Cryptographic hash functions are the subset of hash functions, which are resistant to at least 3 attacks: pre-image, second pre-image and collision [2, 6]. These criteria are classic and the most general, i.e. applicable for any cryptographic hash function. However, the practical application introduces it's own criteria for these cryptographic primitives. For example, performance and protection against all known attacks were the main criteria while selecting functions at the SHA-3 competition [23].

The existence of one-way hash functions has not been theoretically proven. It is assumed that the determination of the input message is a time consuming task. For example, the "birthday paradox" attack allows to find a collision after about $2^{\frac{n}{2}}$ calls of the hash function with an n-bit length hash code. Therefore, the hash function has resistance to the collision attack if and only if there is no algorithm with a complexity less than $2^{\frac{n}{2}}$ [2].

By default (sometimes used as a criterion) it is assumed that a slightest change (e.g., bit inverse) in the input message leads to significant changes in the hash value. This criterion is also known as the avalanche effect and plays a very important role when the hash function is used for generation of pseudorandom sequences [43].

Modern cryptographic hash functions have three main stages to compute the hash code (Fig. 3) [24]

- initialization based on IV (IS);
- partitioning the input message (M) into blocks and consistent application of a compression function (CF) to each of them;
- final transformations and generation of the output (FS).

Most modern hash functions were constructed using the Merkle-Damgård scheme [6, 44, 45]. In the last 10 years many undesirable features have been found in this approach, including the length extension attack [46]. During the

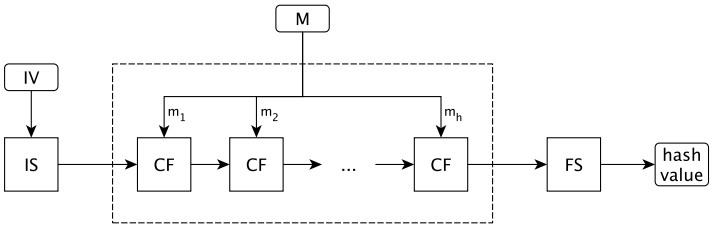


Fig. 3: *The high-level scheme of a hash function*

SHA-3 competition, a well-proven alternative construction called sponge was introduced [47]. It can be used to design authenticated and stream ciphers, message authentication codes, and other symmetric primitives. Moreover, this method of construction is the basis of the algorithm Keccak, which became the winner of SHA-3 [24].

1.3. Methods of cryptanalysis

1.3.1. Differential

Differential cryptanalysis implies the existence of ordered pairs (α, β) such that a randomly chosen plaintext M and the corresponding value $M - \alpha$ map to ciphertexts C and C', respectively [48]. Denote by $\beta = C - C'$ the difference between the ciphertexts, where "$-$" is the operation inverse to the mixing key routine. The ordered pair (α, β) is called the differential. The set of differentials at different rounds for a certain cipher is termed the differential characteristic [5, 48]. The attack is more effective for higher differential probability (at the same time not equal to 1). While the most general case is considered in [49, 50], in this section it is assumed that "$-$" is equivalent to XOR.

To apply the attack a difference distribution table is calculated for a given substitution. The maximum value of the table (MDT) excluding the value of the first row and first column, is calculated as follows [51]

$$\delta = \max_{\alpha \in \mathbb{F}_2^n, \alpha \neq 0, \beta \in \mathbb{F}_2^m} \#\{x \mid S(x) \oplus S(x \oplus \alpha) = \beta\},$$

where S is an S-box used in a cryptographic primitive.

During the differential attack an adversary learns how the difference of plaintexts affects the resulting difference (ciphertexts) [5]. The differential

propagated with the highest probability is used to find a round key. For the most modern block ciphers it is enough to break the entire encryption algorithm.

1.3.2. LINEAR

Linear cryptanalysis is based on the Piling-up lemma and was first applied to the block cipher FEAL [52]. Later Nyberg described the concept of the attack and Matsui has shown a practical example for the block cipher DES [53, 54]. The basic idea of linear cryptanalysis is based on the following statement. For randomly chosen bits of the key (k), plaintext (m) and ciphertext (c) the probability of the expression $\alpha \cdot m + \beta \cdot c + \gamma \cdot k$, where "$\cdot$" denotes the scalar product, differs from $\frac{1}{2}$ [5]. Let S be a substitution with n-bit input and m-bit output, and λ be the maximum value of an approximation table (excluding the value of the cell [0,0]) [51]. Then

$$\lambda = \max_{\alpha \in \mathbb{F}_2^n, \alpha \neq 0, \beta \in \mathbb{F}_2^m} \left| \#\left\{ x \mid \bigoplus_{s=0}^{n-1} (x_s \cdot \alpha_s) = \bigoplus_{t=0}^{m-1} (S[x]_t \cdot \beta_t) - 2^{n-1} \right\} \right|,$$

where γ_j is jth bit of γ. Linear cryptanalysis is more efficient for the greater value of λ [5].

1.3.3. ALGEBRAIC

Algebraic cryptanalysis exploits algebraic features of cryptographic algorithms. Whilst algebraic attacks on stream ciphers are well studied from both a theoretical and practical point of view [51, 55–57], for others cryptoprimitives the question remains open. In this connection, the following description will be based on known results for block ciphers. The same approach can be applied for other cryptographic primitives such as authenticated ciphers, hash function, etc.

During an algebraic attack the encryption algorithm is often represented as a system of equations. To obtain the key it is necessary to solve the system with respect to all variables. It is believed that the system with a lower degree is easier to solve [55]. To implement the attack, the following stages must be performed

- decompose the encryption algorithm into basic components;
- describe each of the elements algebraically;

- bind each of the output values to the input of other components.

Decomposition is a partition of the encryption algorithm into smaller pieces. By a basic component in modern ciphers linear and nonlinear transformations (layers) are understood [5]. An algebraic description is the conversion of the main elements into a system of equations that holds for all input and output values of the transformations. The output of these stages is the system of equations describing the entire encryption (decryption) algorithm including the key expansion routine.

To date there are many methods for solving systems of equations over $\mathbb{F}_2$ such as Gaussian elimination, XL, F4 and Gröbner basis [55, 57]. Moreover, the complexity of most methods depends on the density of the system. This allows one to conclude that the density of the system of equations describing the substitution affects the complexity of the final system.

This method was fist applied to block ciphers by Courtois in the early 2000s [58]. His approach is based on the principle stated above, that is the description of the substitution by the system of equations with the gradual expansion for the entire encryption algorithm. Application of this approach allows to describe AES with a system of equations of degree 2.

Later theoretical results of Courtois were implemented by Weinmann in practice [59]. He attacked a scale-down version of the AES cipher (MiniAES), and thus demonstrated the viability of the algebraic attack. Similar results were obtained by Kleiman in [60]. Unlike Weinmann, she presented a general algorithm based on a matrix approach for obtaining the system of equations describing a given S-box. However, to break 4 rounds of the 16-bit version of AES was not possible, even with enormous computing resources [60]. A few years ago it was announced that a special case of the Gröbner basis algorithm can break up to 10 rounds of scaled-down AES [61].

In 2006 Courtois demonstrated an attack on a full version of 6-round DES [62]. Only one plaintext/ciphertext pair was necessary to find a key (20 bits of which have been fixed) on a personal computer.

Application of the algebraic attack was also demonstrated for ciphers submitted to the Ukrainian competition [63, 64]. Many designers have used Nyberg's method, i.e. calculation of the inverse element in the field $\mathbb{F}_{2^n}$ followed by an affine transformation, to generate substitutions [65]. This approach allows to achieve the best known indicators for protection against differential and linear cryptanalysis. However like in AES, the entire cryptoprimitive can be described by a system of equations of degree 2 [55, 63]. This is an

undesirable property that may cause future attacks. As a consequence, the analysis showed that substitutions used in the ciphers Kalyna and Mukhomor had a number of advantages over other ciphers [21].

There are other trends in the solution of the system of equations such as conversion to the SAT problem [57]. Moreover, in Paper I an essentially different approach to the description of the cryptographic primitive is shown where the degree of the system does not affect the complexity.

1.3.4. Related-key

A related-key attack is a kind of cryptanalysis where an adversary can observe the input and output of a cipher under the influence of different keys. She only knows mathematical relations of the keys whilst the exact values are initially unknown [66].

During this attack it is assumed that the cryptanalyst has no direct access to the searched key (e.g., the key is stored in a hardware encryption unit). Nonetheless, the adversary can change in a certain way different pieces of the key. Due to these limitations, the related-key cryptanalysis is more theoretical than practical. Nevertheless, it allows one to find the key with the minimal known complexity [67].

It should be noted that one of the main components of the biclique attack on AES is a correlation of the round keys [68]. The biclique attack became widespread after the successful implementation on that cipher. The authors of [68] have theoretically proved that the encryption key can be found with a complexity less than exhaustive search.

1.3.5. Combination of the methods

Nowadays it is almost impossible to apply independent attacks against modern ciphers. This is due to the fact that the designers take into consideration all known attacks when a new cipher is created. Differential and linear cryptanalysis in the form which has been applied to DES is already ineffective against present-day ciphers. Thereby, modified or combined attacks have begun to develop rapidly.

Because of the simplicity of the description a lot of attacks based on the differential properties have been developed during the last 20 years. These include truncated differential, impossible differential, boomerang, higher order differential and others attacks [5, 10].

In 2011 the full version of the cipher GOST 28147 was firstly attacked by sequential application of fixed points, meet-in-the-middle and brute force attacks [17]. The same year the first attack on the full version of the cipher AES was published [68]. This attack consists of a combination of related-key and brute force attacks with the help of a complete bipartite graph.

Thus, the development and application of combined methods is a priority area of research in cryptology.

1.3.6. Other directions

It is assumed that even weak ciphers can become cryptographically strong when increasing the number of rounds. However, unlike the others, in slide attacks an adversary analyzes the key expansion rather than looks for vulnerabilities in the encryption routines [69]. This type of attack was firstly proposed by Wagner and Biryukov in [70]. It is mainly applied to iterative ciphers, a part of which (usually the round function) is applied sequentially by using only one key. The important thing in this attack is that the part must be identical and invertible. Thus, the number of cycles of the algorithm in this case does not affect the success of its breaking.

In recent years, the number of papers on cryptanalysis which do not consider the internal structure of the cipher is constantly increasing. For example, in [71, 72] it was shown that if an adversary has access to the session key management then she could restore a long-term key of the cipher GOST 28147 in a few minutes. Another example in this direction is Isobe's attack [73]. It is based on the ratio of the round key lengths to the block size while the round routine of the cipher is represented as a random function. More general theoretical results consist in finding distinguishers for universal schemes (Feistel, Lai-Massey, SPN and Sponge). The analysis shows advantages of one construction over another under the condition of the random or permutation round function. [74].

Side channel attacks should also be mentioned [75]. They use power or time fluctuations, leakage through electromagnetic or sound media, and other sources for obtaining information about the master key. Side channel attacks relate to attacks on implementation. Even theoretically secure encryption algorithms can be broken due to poor software or hardware implementation. However, practical experiments show that in some cases it is possible to create additional criteria to the basic components, thereby increasing the complexity of certain side channel attacks [76].

2. BINARY NONLINEAR MAPPINGS IN CRYPTOGRAPHY

Analysis of the latest solutions used in constructions of advanced cryptographic primitives allows to conclude that they largely inherited ideas of the block cipher AES [10]. Unlike Rijndael, where the substitution was generated based on Nyberg's design, new ciphers have one or more randomly generated S-boxes. Their main advantage is a description by a system of equations of degree 3 [77].

Substitutions for modern symmetric primitives are usually implemented in the form of lookup tables. Considering that lots of symmetric algorithms (e.g., Rijndael, PRESENT, ARIA, Keccak, etc.) use XOR as the key mixing routine, S-boxes are the only elements defining nonlinearity of encryption transformation and the level of resistance against cryptanalytic attacks [5]. Moreover, the number of encryption cycles is calculated based on cryptographic parameters of a nonlinear mapping, given in advance.

Aspects of vectorial Boolean functions used in symmetric cryptography as substitutions and their relevant cryptographic properties are presented in this section.

2.1. DEFINITIONS AND NOTATIONS

Let n and m be two positive integers. Define by $\mathbb{F}_2^n$ a vector space of all binary vectors of length n, where $\mathbb{F}_2$ is the Galois field with elements $\{0,1\}$. Then an (n,m)-function is a vectorial Boolean function $F : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$. Boolean functions $f_1, f_2, \ldots, f_m$, such that $F(x_1, \ldots, x_n) = (f_1(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n))$, and their linear combinations are called coordinate and component functions of F, respectively. If $m = 1$ then a vectorial Boolean function has a single output bit and is called a Boolean function. To find algebraic properties of (n,m)-functions, a vector space is often induced by a structure of the finite field $\mathbb{F}_{2^n}$.

For any positive integers n and m, a function F from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ is called differentially δ-uniform if for every $a \in \mathbb{F}_2^n \setminus \{0\}$ and every $b \in \mathbb{F}_2^m$ the equation $F(x) + F(x+a) = b$ admits at most δ solutions [65, 78]. Vectorial Boolean functions used as S-boxes in cryptographic primitives must have low differential uniformity to provide high resistance to differential cryptanalysis (see Subsection 1.3.1). For the special case $n = m$ differentially 2-uniform functions are called almost perfect nonlinear (APN). Since $\delta \geq 2$, they are optimal regarding this criterion. The notion of APN function is closely related

to the notion of almost bent (AB) function [79]. The last one can be described in terms of the Walsh transform for a function $F : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$

$$\lambda(u, v) = \sum_{x \in \mathbb{F}_2^n} (-1)^{v \cdot F(x) + u \cdot x},$$

where $u \in \mathbb{F}_2^n$, $v \in \mathbb{F}_2^m$ and "$\cdot$" denotes scalar products in $\mathbb{F}_2^n$ and $\mathbb{F}_2^m$, respectively.

The set $\{\lambda(u, v) \mid (u, v) \in \mathbb{F}_2^n \times \mathbb{F}_2^m, v \neq 0\}$ is called the Walsh spectrum of F. If $n = m$ and the Walsh spectrum of F consists of $\{0, \pm 2^{\frac{n+1}{2}}\}$ then the function F is called AB [79]. AB functions exist for n odd only and oppose an optimal resistance to linear cryptanalysis (see Subsection 1.3.2). Every AB function is APN but the converse is not true in general (see [51, 80] for a comprehensive survey on APN and AB functions).

The natural way of representing $F : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ is algebraic normal form (ANF)

$$F(x_1, x_2, \ldots, x_n) = \sum_{I \in \mathbb{P}(\{1,\ldots,n\})} a_I \left(\prod_{i \in I} x_i \right), \qquad a_I \in \mathbb{F}_2^m,$$

where $\mathbb{P}(z)$ denotes the power set of z. The algebraic degree $deg(F)$ of F is the degree of its ANF. F is called affine if $deg(F)$ is at most 1. An affine vectorial Boolean function with $F(0, \ldots 0) = 0$ is linear.

2.2. Cryptographic properties of vectorial Boolean functions

While Boolean functions are adopted mainly as filtering functions in stream ciphers, vectorial Boolean function are used in block and authenticated ciphers, and hash functions as substitutions. For theoretical analysis the univariate representation is one of the best ways to consider cryptographic properties of the binary mappings. However, field operations are not so well optimized as operations with Boolean functions in modern computers, especially for large n. Therefore, it makes sense to represent cryptographic properties of (n, m)-functions using the set of component functions. All definitions and indicators are well-known and one can see [51, 80] for more details.

First of all, let's consider the properties of Boolean functions. A Boolean function of n variables is called balanced if $\sum_{x=0}^{2^n-1} f(x) = 2^{n-1}$, where $x =$

$(x_1, x_2, \ldots, x_n)$. The correlation between an arbitrary Boolean function $f(x)$ and the set of all linear functions is determined by Walsh transformation

$$W(w) = \sum_{x=0}^{2^n-1} (-1)^{f(x) \oplus l_w(x)},$$

where $l_w(x) = w \cdot x = w_1 x_1 \oplus w_2 x_2 \oplus \ldots w_n x_n$. The nonlinearity is related to the Walsh values as

$$NL(f) = \frac{1}{2} \left(2^n - \max_{w \in \mathbb{F}_2^n \setminus \{0\}} |W(w)| \right).$$

Autocorrelation of f noted as $r_f(\alpha)$ shows how the function differs from itself shifted on several positions, i.e.

$$r_f(\alpha) = \sum_{x=0}^{2^n-1} (-1)^{f(x) \oplus f(x \oplus \alpha)},$$

where $\alpha \in \mathbb{F}_2^n$. For cryptography the maximal value of the function $r_f(\alpha)$ is of interest, and can be found as

$$AC_{max}(f) = \max_{\alpha \in \mathbb{F}_2^n \setminus \{0\}} \left| r_f(\alpha) \right|.$$

Let σ be the sum-of-squares indicator, then

$$\sigma = \sum_{\alpha=0}^{2^n-1} r_f^2(\alpha).$$

Let $hw(\alpha)$ be a binary Hamming weight of $\alpha \in \mathbb{F}_2^n$ [51]. Then it is said that $f(x)$ satisfies propagation criterion of order k ($PC(k)$) if and only if for all nonzero vectors $\alpha \in \mathbb{F}_2^n$ such that $1 \leq hw(\alpha) \leq k$ the following is true

$$\sum_{x=0}^{2^n-1} f(x) \oplus f(x \oplus \alpha) = 2^{n-1}.$$

The strict avalanche criterion (SAC) corresponds to $PC(1)$.

A Boolean function is correlation immune of order m ($CI(m)$) if the equation $W(w) = 0$ holds for all $w \in \mathbb{F}_2^n$, where $1 \leq hw(w) \leq m$. If the function is

balanced and satisfies $CI(m)$ simultaneously, then such a function is called m-resilient.

The minimum algebraic degree of $g(x) \neq 0$ of the set $\{g \mid f(x) \cdot g(x) = 0\}$ $\cup \{g \mid (f(x) \oplus 1) \cdot g(x) = 0\}$ is called algebraic immunity (AI) of a Boolean function f.

Using the above definitions let's describe cryptographic properties of substitutions. Suppose S is the table representation of a vectorial Boolean function $F = (f_1, \ldots, f_m)$ from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$. Define $\{h_j = j \cdot F \mid 0 < j < 2^m\}$ as the set of the component functions of F. Then

- nonlinearity of S is
$$NL(S) = \min_{0<j<2^m} \left(NL(h_j)\right);$$
- minimum degree of S is
$$deg(S) = \min_{0<j<2^m} \left(deg(h_j)\right);$$
- the maximum value of autocorrelation spectrum of S is
$$AC_{max}(S) = \max_{0<j<2^m} \left(AC_{max}(h_j)\right);$$
- S satisfies strict avalanche criterion if every h_j satisfies SAC;
- S satisfies propagation criterion of order k if every h_j satisfies $PC(k)$;
- S is correlation immune of order k if every h_j is $CI(k)$;
- S is balanced (permutation) if every h_j is balanced;
- S is k-resilient if every h_j is k-resilient.

Similar properties for vectorial Boolean functions are given in [51].

While the maximum value of the approximation table (λ) can be calculated directly from the nonlinearity of the S-box as $\lambda = 2^{n-1} - NL(S)$, the maximum value of the differential table cannot be directly evaluated from the component functions. For the given S-box the indicator δ-uniformity defined in 1.3.1 and 2.1 is equivalent to the maximum value of MDT.

The ways to represent a substitution as a system of equations over $\mathbb{F}_2$ are given in [60, 63]. Define density as the fraction of nonzero elements in a system of equations. Then, a substitution provides better protection against algebraic attacks (see 1.3.3) if the system

- has higher degree;
- has fewer equations;
- is more dense.

Unambiguous theoretical relations between these parameters is an unsolved problem [81]. Suppose the degree of a system of equations is the maximal algebraic degree of all polynomials this system consists of. Then algebraic immunity of the S-box ($AI(S)$) means the smallest degree of the system describing this substitution.

2.3. Equivalence of vectorial Boolean functions

Two functions $F, G : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ are called extended affine equivalent if there exist such affine permutations $A_1 = L_1(x) + c_1, A_2 = L_2(x) + c_2$ and an arbitrary linear function $L_3(x)$ such that

$$F(x) = A_1 \circ G \circ A_2(x) + L_3(x).$$

If $L_3(x) = const$, or $L_3(x) = 0, c_1 = 0$, and $c_2 = 0$ then F and G are affine, or linear equivalent, respectively. Moreover, for at least one missing element of $L_1(x), L_2(x), L_3(x), c_1, c_2$ the functions are called restricted EA (REA) equivalent [82].

Any affine function $A : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ can be represented in matrix form

$$A(x) = K \cdot x \oplus C,$$

where K is an $m \times n$ matrix and $C \in \mathbb{F}_2^m$. All operations are performed in $\mathbb{F}_2$, thus the above equation can be rewritten as

$$\begin{pmatrix} a_0 \\ a_1 \\ \cdots \\ a_{m-1} \end{pmatrix}_x = \begin{pmatrix} k_{0,0} & \cdots & k_{0,n-1} \\ k_{1,0} & \cdots & k_{1,n-1} \\ \vdots & \ddots & \vdots \\ k_{m-1,0} & \cdots & k_{m-1,n-1} \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ \cdots \\ x_{n-1} \end{pmatrix} \oplus \begin{pmatrix} c_0 \\ c_1 \\ \cdots \\ c_{m-1} \end{pmatrix}$$

where $a_i, c_i, x_s, k_{j,s} \in \mathbb{F}_2$. This representation allows to describe EA-equivalence in matrix form

$$F(x) = M_1 \cdot G(M_2 \cdot x \oplus V_2) \oplus M_3 \cdot x \oplus V_1$$

where elements of $\{M_1, M_2, M_3, V_1, V_2\}$ have dimensions $\{m \times m, n \times n, m \times n, m, n\}$.

In [83] F and G are considered as $G_F(x, y) = \{\{x, y\} \mid y = F(x)\}$. They are Carlet-Charpin-Zinoviev (CCZ) equivalent, if for $F_2(x) = L_3(x) + L_4 \circ G(x)$ and permutation $F_1(x) = L_1(x) + L_2 \circ G(x)$ the following equation holds

$$F(x) = F_2 \circ F_1^{-1}(x),$$

where $L_1(x)$, $L_2(x)$, $L_3(x)$, $L_4(x)$ are arbitrary affine functions.

CCZ-equivalence is the most general known equivalence of functions for which differential uniformity and extended Walsh spectrum are invariants. In particular every function CCZ-equivalent to an APN (respectively, AB) function is also APN (respectively, AB). EA-equivalence is a special case of CCZ-equivalence [51]. The algebraic degree of a vectorial Boolean function is invariant under EA-equivalence but, in general, it is not preserved by CCZ-equivalence.

3. Summary of Papers

This thesis is based on seven papers. A synopsis of each paper is given in the following subsections.

3.1. Paper I

Several approaches which use binary decision diagrams for algebraic attacks are well-known in open literature. The efficiency of BDD-based attacks is demonstrated both for general models and for particular cases such as A5/1, E0 and Trivium [84, 85]. In this paper we extend the previous results on block ciphers and present new specific strategies and approaches for solving compressed right hand side (CRHS) systems [86].

Most ciphers use only one nonlinear element, which is usually represented as a lookup table. Hence, we are interested in finding a BDD that represents a given S-box mapping $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$. Let the input and output bits of the S-box be $x_0, \ldots, x_{n-1}$ and $y_0, \ldots, y_{m-1}$, respectively. Denote the levels of a binary tree as $\{x_0, \ldots, x_{n-1}, y_0, \ldots, y_{m-1}\}$. For each value of substitution create a path from the source node on top to the sink node (true node) at the bottom, and all edges direct downwards. If the edges are divided into 0-edges and 1-edges, then we can uniquely represent an arbitrary S-box using a BDD upto the order of variables.

Since each level is represented in general as a linear combination of input and output bits, then a linear layer of a cryptographic primitive does not add extra variables and as a consequence does not affect the complexity of the operations performed on the tree [86]. This representation allows to join several BDDs using adjacent variables on different levels. Thereby, the entire encryption algorithm can be described as one big BDD or as the set of smaller BDDs.

Several operations such as swapping and adding levels, and absorbing linear dependencies are also defined on this special version of BDD. While joining together many BDDs and absorbing all linear dependencies, the solving complexity depends heavily on the order the BDDs are joined. Finding the ordering of BDDs that gives the minimum complexity is probably a hard problem. During our experiments we have not found a strategy for ordering that is universally best. However, we described automatic ordering, divide-and-conquer and order by cryptanalysis strategies for how to join and absorb, with the aim to keep the complexity down.

We apply the proposed attack on DES with a reduced number of rounds, MiniAES and the EA-equivalence problem. Our experiments have shown that 6-round DES can be broken in approximately one minute on an ordinary computer. This is a factor 2^{20} improvement over the best earlier algebraic attack on DES using MiniSAT [62].

There have been several earlier attempts to break MiniAES [60, 61, 87]. Approaches that exploit the short key in MiniAES (only 16 bits) succeed very quickly, but the general methods of F4 and XL/XSL failed to solve systems representing more than one round of MiniAES. The approach we use in the paper does not exploit the short key, while still solving systems representing 10 rounds of MiniAES using approximately 45 minutes and 8GB of memory. In addition, the BDD method has shown the advantages compared to a Gröbner basis and CryptoMiniSat for solving the EA-equivalence problem.

Despite the excellent practical results, a number of unresolved issues still remain. The main one concerns the theoretical estimates of the complexity of the BDD attack.

3.2. PAPER II

For most new algorithms evaluation of the resistance to known attacks, such as differential, linear or algebraic, is provided by the designers. However, an independent verification of the results is always needed [88]. To conduct such

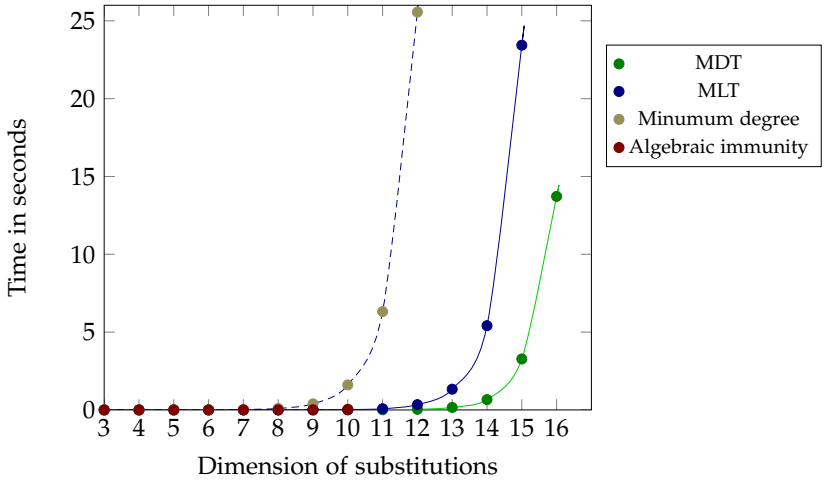


Fig. 4: *The relationship between the dimension of random substitutions and time of calculation*

research, tools for analysis of both basic components and entire encryption algorithms are required. On the other hand, universal approaches would also be a useful supplement for the designers of prospective algorithms. Choosing linear layers is a relatively simple task when only few indicators are considered [89]. The situation is completely opposite for nonlinear layers which usually consist of parallel application of substitutions.

As was mentioned in Section 2 vectorial Boolean functions have lots of cryptographic properties. While for a given S-box some properties are calculated directly from the formula, others require special knowledge (i.e. for algebraic immunity). Today there are a number of tools that can be considered a partial solution to the problem [57, 90–93]. However, the cryptographic community needs a universal approach to calculate indicators for arbitrary binary mappings. In this paper a tool for generating and analyzing arbitrary vectorial Boolean functions $F : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ was given.

The proposed library (package) S-box includes methods for calculation of all indicators described in Section 2. In particular one can find δ-uniformity, nonlinearity or maximum of the linear approximation table, minimum degree, algebraic immunity, maximum value of autocorrelation spectrum, correlation immunity and other cryptographic properties for arbitrary vectorial Boolean functions. In addition to this, there are implemented several methods for

Table 1: *Comparison of 8-bit S-boxes*

Properties	AES	GOST R 34.11-2012	STB 34.101.31-2011	Kalyna's S0 [21]	Proposed in [77]
MDT	4	8	8	8	8
NL	112	100	102	96	104
Absolute indicator	32	96	80	88	80
SSI	133120	258688	232960	244480	194944
Minimum degree	7	7	6	7	7
Algebraic immunity	2	3	3	3	3

generating substitutions with predefined properties based on Gold, Kasami, Welch, inverse and other well-known functions. The library also contains a number of auxiliary functions such as finding the univariate polynomial or the system of equations describing the substitution; checking the APN property, or CCZ- equivalence; generating look up tables based on the user-defined univariate polynomials and many others.

The performance and arbitrary dimension of binary nonlinear mappings were the main criteria for the S-box library. Calculation of some indicators are based on known results [94, 95], while others (i.e. cyclic properties or algebraic immunity) were optimized during the research and experiments. Fig. 4 shows the time complexity of several frequently used methods for $n = m$.

From a practical point of view, Sbox can be used to analyze nonlinear components of the existing or prospective cryptographic primitives. An example of the substitution comparison is given in Table 1.

In conclusion, the library includes lots of functions for computing the properties of permutations and methods of generation. Despite this, there are many directions for improvement and development. The library is designed to facilitate extension of its functionality quite easily, for instance by combining to optimize methods for calculation of indicators such as minimum degree or autocorrelation, or by realizing a native integration with Sage and creation of a universal test environment.

3.3. PAPER III

Since substitutions are one of the main components that determine the security of modern cryptographic algorithms, many cryptographic criteria must be considered for a new cryptographic primitive. Taking into account the large number of existing indicators, their controversy and partial interdependence, it is most likely impossible to generate a substitution that satisfies all known requirements. This became a reason to use a substitution satisfying only mandatory criteria essential for a particular symmetric algorithm. Such substitutions are called optimal [10, 51, 80]. Optimality criteria may vary depending on which cipher is considered.

After investigation of existing and prospective attacks the following criteria were highlighted as significant

- maximum value of minimum degree;
- maximum algebraic immunity with the minimum number of equations;
- absence of fixed points (cycles of length 1);
- substitution must be bijective (permutation);
- minimum value of δ-uniformity and maximum value of nonlinearity limited by parameters listed above.

In particular, for $n = 8$ an optimal permutation has algebraic degree 7, algebraic immunity 3 and 441 equations, δ-uniformity under 8, nonlinearity over 100 and without fixed points.

The majority of theoretical methods for generation of vectorial Boolean functions have extreme characteristics of δ-uniformity and nonlinearity, but at the same time do not possess other properties (i.e., high value of algebraic immunity) which are necessary for next-generation symmetric cryptographic primitives.

The first and most obvious solution is to generate random permutations and check them on optimality. After 12 hours of cluster operation (4096 cores) there were found 27 optimal permutations with $NL = 100$. Four of them were CCZ-inequivalent. After 48 hours (22 years on 1 core) the program run on the same cluster didn't find any substitution with $NL \geq 102$.

A counterexample was found in STB 34.101.31-2011 [20]. The optimal substitution has $NL = 102$. Thus we found another way to generate substitutions with $NL \geq 100$. Instead of trying to find a random permutation or apply the

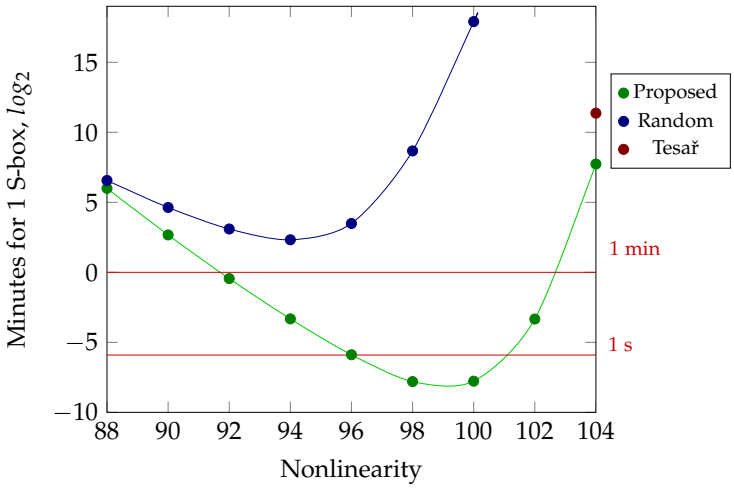


Fig. 5: *Performance comparison of the substitution generation methods*

hill climbing technique, it was decided to solve the problem from the other side. We started with the best known permutation and modified it (swapped values) until the expected result was achieved. As was proven in [96] one swap does not much influence nonlinearity and δ-uniformity.

Before presenting our result at the conference we found another algorithm which produces the same result [97]. The performance comparison (Fig. 5) shows that our proposed method is 10 times faster then Tesař's [97].

After 107 hours of cluster operations, that are equivalent to 50 years on a single-processor computer, there were not found better substitutions. The practical results of both methods show that there are no optimal substitutions with nonlinearity greater than 104. However, there are permutations with nonlinearity 106 and algebraic immunity 2, in which the number of equations is small (e.g. 1). Hereby, the question about existence of optimal substitutions with nonlinearity more than 104 remains open.

Four substitutions used in the new Ukrainian block cipher and hash function were generated by the proposed method. An additional criterion which the substitutions must satisfy is belonging to different CCZ-equivalent classes. Details stated in Paper V.

3.4. PAPER IV

In 2010 at the RusCrypto'10 conference a prototype of the prospective hash function also known as Stribog (Steebog) [27, 98] was presented. Two years latter this hash function was accepted as the governmental standard GOST R 34.11-2012 [29]. The description of the hash function available in public literature was only algorithmic. To prove some cryptographic properties it is necessary to have a common mathematical representation as has been made in this paper.

The core of the hashing algorithm is the $L \circ P \circ S$ transformation. Transformation of the state into an 8×8 byte matrix gave a general idea of each transformation. Further investigations showed that S and P transformations have analogues in AES. While S is identical to the SubBytes routine, P is similar to ShiftRows. Unlike ShiftRows, P transposes the state instead of shifting it by a constant number positions. The most difficult task was to identify the L transformation, which is a multiplication by a 64×64 binary matrix.

In summary, the main issue was to find the irreducible polynomial which gives the representations of transformations over $\mathbb{F}_{2^8}$ that produce the same outputs. Based on the assumption that the matrix used in L possesses the MDS property, the polynomial $f(x) = x^8 + x^6 + x^5 + x^4 + 1$ was found. Using this polynomial all basic components of the hash function were described in AES-like form.

This representation allows to use the wide trail strategy to prove the resistance of the hash function to differential and linear cryptanalysis. At the same time, the existing attacks can be easily adapted to GOST R 34.11-2012 [99]. Additionally, this gives access to well-known optimization techniques for increasing performance on a variety of platforms [10]. Using a table approach a fast cross-platform implementation of Stribog was proposed [100, 101].

3.5. PAPER V

As stated before, the choosing of essential properties for new substitutions is not a trivial task. In this paper an analysis of the absence of fixed points criterion is given. If one considers the round function instead of a single substitution, then even for the AES S-box fixed points can be achieved.

The investigation is based on the fact that a cipher has lots of isomorphic (equivalent) representations. For AES the ShiftRows, MixColumns and

AddRoundKey routines are linear transformations with respect to XOR. Manipulations with these transformations give different representations [55]. It is shown that at least one fixed point can be found for the AES substitution in case of using XOR operation in AddRoundKey.

Applying the same model the advantages of additional modulo 2^n were shown. A mixing key routine based on the modulo operation adds more nonlinearity and as a result reduces the number of possibilities for adversaries. The analysis shows the necessity of additional requirements for multiple substitutions used in one cryptographic primitive.

Proposition 1. *Substitutions S_1, S_2, ..., S_l used in a nonlinear layer must belong to different classes of equivalence.*

Since CCZ-equivalence is the most general case of known equivalences, it makes sense to check whether substitutions belong to different CCZ-equivalence classes.

The more practical result was achieved independently for Zorro [102]. The core of that attack has the same principles that were described in this paper.

3.6. PAPER VI

The behavior of nonlinear feedback shift registers is poorly understood, which, in turn, results in a lack of criteria for selecting parameters that directly affect security. To achieve this, designers of stream ciphers often combine linear and nonlinear registers. MICKEY is an example of such ciphers.

In several papers the theoretical weaknesses of MICKEY were presented [103–106]. It was shown in particular that choosing constants in the wrong way may lead to security problems. The shared idea of all these attacks is the construction of a backward states tree. After collecting the results from all previous papers it became possible to evaluate theoretically the probabilities of all possible branches in the tree. We proved both theoretically and practically that in key/IV load mode the expectation value of degree approximately equals 2. The analogous value for preclock and key-generation mode is approximately equal to 1. Thus, knowing the internal state of registers it is always possible to perform reverse steps to acquire the state after key initialization function. However, the inverse key/IV load modes produce a complete binary tree.

The other parts of the paper describe some practical observations. First, it is noted that each reverse step increases the probability of subtree cutting off

Table 2: *Complexities for Solving REA-equivalence Problem*

#	Restricted EA-equivalence	Complexity	$G(x)$
1	$F(x) = M_1 \cdot G(M_2 \cdot x)$	$O\left(n^2 \cdot 2^n\right)$	P
2	$F(x) = M_1 \cdot G(M_2 \cdot x \oplus V_2) \oplus V_1$	$O\left(n \cdot 2^{2n}\right)$	P
3	$F(x) = M_1 \cdot G(x \oplus V_2) \oplus V_1$	$O\left(2^{2n+1}\right)$	†
4	$F(x) = M_1 \cdot G(x \oplus V_2) \oplus V_1$	$O\left(n \cdot 2^{3n}\right)$	A
5	$F(x) = G(M_2 \cdot x \oplus V_2) \oplus V_1$	$O\left(n \cdot 2^n\right)$	P
6	$F(x) = G(x \oplus V_2) \oplus M_3 \cdot x \oplus V_1$	$O\left(n \cdot 2^n\right)$	A
7	$F(x) = M_1 \cdot G(x \oplus V_2) \oplus M_3 \cdot x \oplus V_1$	$O\left(2^{2n+1}\right)$	‡
8	$F(x) = M_1 \cdot G(x \oplus V_2) \oplus M_3 \cdot x \oplus V_1$	$O\left(n \cdot 2^{3n}\right)$	A

P - permutation; A - arbitrary;
† - G is under condition $\{2^i \mid 0 \leq i \leq m-1\} \subset \mathrm{img}(G')$ where $G'(x) = G(x) + G(0)$;
‡ - G is under condition $\{2^i \mid 0 \leq i \leq m-1\} \subset \mathrm{img}(G')$ where $G'(x) = G(x) \oplus L_G(x) \oplus G(0)$.

with all previous states. This property exists since there is a high probability of orphan states. Therefore, in some cases key bits could be found uniquely. Second, since the functions used for different modes are the same, it allows to generate key-streams shifted by a fixed number of bits for different pairs of key and IV. However, the conditions imposed on the use of keys and IVs stated in the MICKEY's specification do not give the opportunity to apply the attack in the real world. In the end, the meet-in-the-middle attack based on the backward states tree is proposed.

Taking into consideration everything mentioned above, the proposed method for analysis of MICKEY-like ciphers allows to justify the choice of the encryption algorithm parameters based on the estimation of branch points degree probabilities.

3.7. Paper VII

In [91] Alex Biryukov et al. have shown that in the case when given functions are permutations of $\mathbb{F}_2^n$, the complexity of determining their linear and affine equivalence equals $O\left(n^2 \cdot 2^n\right)$ and $O\left(n \cdot 2^{2n}\right)$, respectively. In Paper VII we

Table 3: *Practical Comparison of Solving REA-equivalence Problem*

#	n=6		n=8		n=10		n=12		n=14	
	ES	KM	ES	KM	ES	KM	ES	KM	ES	KM
1	69	12	125	14	197	17	285	20	389	22
2	81	15	141	19	217	24	309	28	417	32
3	47	13	79	17	199	21	167	25	223	29
4		21		27		34		40		46
5	47	12	79	14	199	17	167	20	223	22
6	48	9	80	11	120	14	168	16	224	18
7	83	13	143	17	219	21	311	25	419	29
8		21		27		34		40		46

consider the conditions under which the complexities of checking vectorial Boolean functions $F, G : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ on REA-equivalence can be reduced.

Matrix form is used for EA-equivalence representation in both presented and [91] methods. This approach allows to prove a number of propositions. Most of them are summarized in Table 2. The first two rows present the complexities from [91].

Proposition 2. *Any linear function $L : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ can be converted to a matrix with complexity $O(n)$.*

Since the considered functions have different REA-equivalent representations, the complexities can not be directly compared to each other. Therefore, Table 3 presents the comparison of known methods (KM) with exhaustive search (ES) based on the calculated complexities (in binary logarithm form) for most interesting values of n.

It is easy to see that for some of the above cases the complexity takes polynomial time. Obtained results give a practical method for checking arbitrary vectorial Boolean functions on REA-equivalence.

4. Conclusions

The research conducted solved a number of current important scientific tasks related to improving methods of cryptanalysis and developing of new requirements for advanced symmetric cryptoalgorithms. In particular, backwards

states cryptanalysis of the stream cipher MICKEY, and BDD-based algebraic attacks on DES and MiniAES show that even well-studied ciphers may have weaknesses. Consideration of these attacks at the design stage of new primitives enables to create better and more secure cryptographic algorithms.

In the post-AES era many cryptoprimitives providing high-level security have random substitutions. The main filtering criteria are balancedness, absence of fixed points, δ-uniformity, minimum degree, algebraic immunity and nonlinearity. At the same time, promising algebraic cryptanalysis is not yet fully understood, and the boundaries of its application are not clear.

A new heuristic method for generating S-boxes has been proposed based on the gradient descent method for generation of Boolean functions. It allows to generate substitutions with the best properties known to date at low cost resources. In particular, for $n = 8$ case the application of the method gives permutations with absence of fixed points, and indicators δ-uniformity 8, nonlinearity 104, minimum degree 7 and algebraic immunity 3. These substitutions surpass analogues used in standards STB 34.101.31-2011, GOST R 34.11-2012 and in the draft standard of the new Russian block cipher.

Advanced design approaches of symmetric cryptographic algorithms introduce additional requirements for S-boxes. One such requirement is that all permutations used in a nonlinear layer belong to different equivalence classes. Satisfying this reduces the number of weak isomorphic representations of an encryption algorithm. As a consequence, it becomes necessary to find equivalent transformations that can be used to construct isomorphic representations.

Several new methods for checking the equivalence of two binary nonlinear mappings have been proposed. These methods are based on the conversion of a linear function defined over a field $\mathbb{F}_{2^n}$ to the matrix form. Under certain conditions the complexity can be reduced to polynomial. The approaches used in proving of the proposed methods can be additionally applied to find original high-level representations of cryptographic primitives such as GOST R 34.11-2012.

The main practical result is the designed software for effective generation and calculation of indicators of arbitrary nonlinear binary mappings. This allows one to create and analyze arbitrary nonlinear components used in symmetric cryptographic primitives. Besides this, a patch for OpenSSL based on a cross-platform implementation of GOST R 34.11-2012 noted in Paper IV was created by Dmitry Olshansky [107]. Most of these results have also been

used in one of the Ukraine's leading companies that provides services in the field of information security.

REFERENCES

[1] SHIMEALL, T., SPRING, J.: *Introduction to information security: A strategic-based approach*. Syngress Publishing, 2013.

[2] SCHNEIER, B.: *Applied cryptography: Protocols, algorithms, and source code in C*. Wiley, 1996.

[3] GORBENKO, I., GORBENKO, Y.: *Applied cryptology: Theory. Practice. Application*. LLC Publishing "Fort", 2013. (In Ukrainian).

[4] VAN TILBORG, H. C., JAJODIA, S.: *Encyclopedia of cryptography and security*. Springer, 2011.

[5] KNUDSEN, L. R., ROBSHAW, M.: *The block cipher companion*. Information Security and Cryptography. Springer Berlin Heidelberg, 2011.

[6] MENEZES, A. J., VAN OORSCHOT, P. C., VANSTONE, S. A.: *Handbook of applied cryptography*. CRC Press, 2010.

[7] PRENEEL, B., BIRYUKOV, A., CANNIÈRE, C. D., ET AL.: NESSIE: Final report of European project number IST-1999-12324, named New European Schemes for Signatures, Integrity, and Encryption. *Electronic source,* 2004. `https://www.cosic.esat.kuleuven.be/nessie/Bookv015.pdf`.

[8] FIPS 46–3: Data Encryption Standard (DES). *National Institute of Standards and Technology,* 1993.

[9] LOUKIDES, M., GILMORE, J.: *Cracking DES: Secrets of encryption research, wiretap politics and chip design*. O'Reilly Media, 1998.

[10] DAEMEN, J., RIJMEN, V.: *The design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.

[11] FIPS PUB 197: Advanced Encryption Standard (AES). *National Institute of Standards and Technology,* 2001.

[12] ROBSHAW, M. J. B., BILLET, O. (eds.): *New stream cipher designs - The eSTREAM finalists*, vol. 4986 of *Lecture Notes in Computer Science*. Springer, 2008.

[13] CID, C., ROBSHAW, M., ET AL.: The eSTREAM portfolio in 2012. *Electronic source,* 2012. `http://www.cspforum.eu/D.SYM.10-v1.pdf`.

[14] CRYPTREC: Report of the cryptographic technique monitoring subcommittee. *Electronic source,* 2003. `http://www.cryptrec.go.jp/report/c03_wat_final.pdf`.

[15] CRYPTREC: Specifications of e-Government recommended ciphers. *Electronic source,* 2013. `http://www.cryptrec.go.jp/english/method.html`.

[16] Charnes, C., O'Connor, L., Pieprzyk, J., Safavi-Naini, R., Zheng, Y.: Comments on Soviet encryption algorithm. In De Santis, A. (ed.), *Advances in Cryptology — EUROCRYPT'94*, vol. 950 of *Lecture Notes in Computer Science*, pp. 433–438. Springer Berlin Heidelberg, 1995.

[17] Isobe, T.: A single-key attack on the full GOST block cipher. In Joux, A. (ed.), *Fast Software Encryption*, vol. 6733 of *Lecture Notes in Computer Science*, pp. 290–305. Springer Berlin Heidelberg, 2011.

[18] Dinur, I., Dunkelman, O., Shamir, A.: Improved attacks on full GOST. In Canteaut, A. (ed.), *Fast Software Encryption*, vol. 7549 of *Lecture Notes in Computer Science*, pp. 9–28. Springer Berlin Heidelberg, 2012.

[19] Aleksev, E., Smyshlyaev, S.: GOST 28147-89: "Do not rush to bury him." Part 1. Security of the algorithm. *Electronic source,* 2013. http://www.cryptopro.ru/blog/2013/08/27/gost-28147-89-ne-speshi-ego-khoronit-chast-1-stoikost-algoritma. (In Russian).

[20] STB 34.101.31-2011: Information technology and security. Information security. Cryptographic encryption algorithms and control of integrity. p. 35, 2011.

[21] Oliynykov, R., Gorbenko, I., Dolgov, V., Ruzhentsev, V.: Results of Ukrainian national public cryptographic competition. In *Tatra Mountains Mathematical Publications*, vol. 47, pp. 99–113. Mathematical Institute of Slovak Academy of Sciences, 2010.

[22] Oliynykov, R.: Next generation of block ciphers providing high-level security. *Winter School in Information Security, Finse,* 2014. `https://www.frisc.no/wp-content/uploads/2014/05/finse2014-oliynykov.pdf`.

[23] Kayser, R. F.: Announcing request for candidate algorithm nominations for a new cryptographic hash algorithm (SHA-3) family. In *Federal*

Register, vol. 72, pp. 62 212–62 220. National Institute of Standards and Technology, 2007.

[24] Chang, S.-j., Perlner, R., Burr, W. E., et al.: Third-round report of the SHA-3 cryptographic hash algorithm competition. *National Institute of Standards and Technology,* 2012.

[25] FIPS PUB 202 (draft): SHA-3 standard: Permutation-based hash and extendable-output functions. *National Institute of Standards and Technology,* May 2014.

[26] Grebnev, S., Dmukh, A., Dygin, D., Matyukhin, D., Rudskoy, V., Shishkin, V.: Asymmetric reply to SHA-3: Russian hash function draft standard. *In Pre-proceedings of Workshop on Current Trends in Cryptology (CTCrypt 2012),* 2012.

[27] GOST R 34.11-20__ (draft) revision 1: Information technology. Cryptographic data security. Hash function. *Electronic source,* 2010. `http://infotecs.ru/laws/gost/proj/gost3411.pdf`. (In Russian).

[28] Kazymyrov, O., Kazymyrova, V.: Algebraic aspects of the Russian hash standard GOST R 34.11-2012. *In Pre-proceedings of 2nd Workshop on Current Trends in Cryptology (CTCrypt 2013),* pp. 160–176, 2013.

[29] GOST R 34.11-2012: Information technology. Cryptographic data security. Hash-function. *Federal Agency on Technical Regulation and Metrology,* p. 34, 2013.

[30] Shishkin, V., Dygin, D., Lavrikov, I., Marshalko, G., Rudskoy, V., Trifonov, D.: Low-weight and hi-end: draft russian encryption standard. *In Pre-proceedings of 3rd Workshop on Current Trends in Cryptology (CTCrypt 2014),* 2014.

[31] Gauravaram, P., Knudsen, L. R., Matusiewicz, K., Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S. S.: Grøstl – a SHA-3 candidate. *Submission to NIST (Round 3),* 2011. `http://www.groestl.info/Groestl.pdf`.

[32] CAESAR: Call for submissions, final (2014.01.27). *Electronic source,* 2014. `http://competitions.cr.yp.to/caesar-call.html`.

[33] Poschmann, A. Y.: *Lightweight cryptography: Cryptographic engineering for a pervasive world*. Ph.D. thesis, Ruhr University Bochum, Germany,

2009.

[34] Bellare, M., Namprempre, C.: Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In Okamoto, T. (ed.), *Advances in Cryptology — ASIACRYPT 2000*, vol. 1976 of *Lecture Notes in Computer Science*, pp. 531–545. Springer Berlin Heidelberg, 2000.

[35] Black, J., Halevi, S., Krawczyk, H., Krovetz, T., Rogaway, P.: UMAC: Fast and secure message authentication. In Wiener, M. (ed.), *Advances in Cryptology — CRYPTO' 99*, vol. 1666 of *Lecture Notes in Computer Science*, pp. 216–233. Springer Berlin Heidelberg, 1999.

[36] ISO/IEC 9797-1: Information technology – Security techniques – Message authentication codes (MACs) – Part 1: Mechanisms using a block cipher. *ISO/IEC JTC 1/SC 27*, p. 40, 1999.

[37] Rogaway, P.: Nonce-based symmetric encryption. In Roy, B., Meier, W. (eds.), *Fast Software Encryption*, vol. 3017 of *Lecture Notes in Computer Science*, pp. 348–358. Springer Berlin Heidelberg, 2004.

[38] Rogaway, P.: Authenticated-encryption with associated-data. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, pp. 98–107. ACM, New York, NY, USA, 2002.

[39] Namprempre, C., Rogaway, P., Shrimpton, T.: AE5 security notions: Definitions implicit in the CAESAR call. *Cryptology ePrint Archive, Report 2013/242*, 2013. `http://eprint.iacr.org/`.

[40] Selmer, E. S.: *Linear recurrence relations over finite fields*. University of Bergen, 1966.

[41] Rueppel, R. A.: *Analysis and Design of Stream Ciphers*. Communications and Control Engineering Series. Springer Berlin Heidelberg, 1986.

[42] Shannon, C. E.: A mathematical theory of communication. In *Bell system technical journal*, vol. 27, pp. 623–656. University of Illinois Press, 1948.

[43] Barker, E., Kelsey, J.: Recommendation for random number generation using deterministic random bit generators. *NIST Special Publication 800-90A*, 2012.

[44] Merkle, R.: One way hash functions and DES. In Brassard, G. (ed.), *Advances in Cryptology — CRYPTO' 89 Proceedings*, vol. 435 of *Lecture Notes in Computer Science*, pp. 428–446. Springer New York, 1990.

[45] Damgård, I.: A design principle for hash functions. In Brassard, G. (ed.), *Advances in Cryptology — CRYPTO' 89 Proceedings*, vol. 435 of *Lecture Notes in Computer Science*, pp. 416–427. Springer New York, 1990.

[46] Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: On the indifferentiability of the sponge construction. In Smart, N. (ed.), *Advances in Cryptology – EUROCRYPT 2008*, vol. 4965 of *Lecture Notes in Computer Science*, pp. 181–197. Springer Berlin Heidelberg, 2008.

[47] Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Cryptographic sponge functions. *Submission to NIST (Round 2),* 2011. `http://sponge.noekeon.org/CSF-0.1.pdf`.

[48] Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. In Menezes, A., Vanstone, S. (eds.), *Advances in Cryptology-CRYPT0'90*, vol. 537 of *Lecture Notes in Computer Science*, pp. 2–21. Springer Berlin Heidelberg, 1991.

[49] Alekseychuk, A., Schevtsov, A.: Upper estimates of imbalance of bilinear approximations for round functions of block ciphers. In *Cybernetics and Systems Analysis*, vol. 46, pp. 376–385. Springer US, 2010.

[50] Kovalchuk, L., Bezditnyi, V.: Upper bounds for the average probabilities of difference characteristics of block ciphers with alternation of Markov transformations and generalized Markov transformations. In *Cybernetics and Systems Analysis*, vol. 50, pp. 386–393. Springer US, 2014.

[51] Carlet, C.: *Vectorial Boolean functions for cryptography*. Boolean Models and Methods in Mathematics, Computer Science, and Engineering. Cambridge University Press, 2010.

[52] Matsui, M., Yamagishi, A.: A new method for known plaintext attack of FEAL cipher. In Rueppel, R. A. (ed.), *Advances in Cryptology — EUROCRYPT' 92*, vol. 658 of *Lecture Notes in Computer Science*, pp. 81–91. Springer Berlin Heidelberg, 1993.

[53] Nyberg, K.: Linear approximation of block ciphers. In De Santis, A. (ed.), *Advances in Cryptology — EUROCRYPT'94*, vol. 950 of *Lecture*

Notes in Computer Science, pp. 439–444. Springer Berlin Heidelberg, 1995.

[54] Matsui, M.: Linear cryptanalysis method for DES cipher. In Helleseth, T. (ed.), *Advances in Cryptology — EUROCRYPT '93*, vol. 765 of *Lecture Notes in Computer Science*, pp. 386–397. Springer Berlin Heidelberg, 1994.

[55] Bard, G. V.: *Algebraic cryptanalysis*. Springer, 2009.

[56] Canteaut, A.: Open problems related to algebraic attacks on stream ciphers. In Ytrehus, Ø. (ed.), *Coding and Cryptography*, vol. 3969 of *Lecture Notes in Computer Science*, pp. 120–134. Springer Berlin Heidelberg, 2006.

[57] Albrecht, M.: *Algorithmic algebraic techniques and their application to block cipher cryptanalysis*. Ph.D. thesis, Royal Holloway, University of London, the United Kingdom, 2010.

[58] Courtois, N., Pieprzyk, J.: Cryptanalysis of block ciphers with overdefined systems of equations. *Cryptology ePrint Archive, Report 2002/044*, 2002. `http://eprint.iacr.org/`.

[59] Weinmann, R.-P.: *Evaluating algebraic attacks on the AES*. Ph.D. thesis, Darmstadt University of Technology, Germany, 2003.

[60] Kleiman, E.: *High performance computing techniques for attacking reduced version of AES using XL and XSL methods*. Ph.D. thesis, Iowa State University, USA, 2010.

[61] Bulygin, S., Brickenstein, M.: Obtaining and solving systems of equations in key variables only for the small variants of AES. In *Mathematics in Computer Science*, vol. 3, pp. 185–200. Birkhäuser-Verlag, 2010.

[62] Courtois, N., Bard, G.: Algebraic cryptanalysis of the Data Encryption Standard. In Galbraith, S. (ed.), *Cryptography and Coding*, vol. 4887 of *Lecture Notes in Computer Science*, pp. 152–169. Springer Berlin Heidelberg, 2007.

[63] Kazymyrov, O., Oliynykov, R.: Construction of an overdefined system of equations describing the cipher "Labyrinth". In *Applied Radio Electronics*, vol. 8, pp. 247–251. Kharkiv National University of Radioelectronics, 2009. (In Russian).

[64] Kazymyrov, O., Oliynykov, R.: Algebraic properties of Kalyna's key schedule. In *Radioelectronic and computer systems*, vol. 5, pp. 61–66. National Aerospace University, Ukraine, 2010. (In Russian).

[65] Nyberg, K.: Differentially uniform mappings for cryptography. In Helleseth, T. (ed.), *Advances in Cryptology - EUROCRYPT'93*, vol. 765 of *Lecture Notes in Computer Science*, pp. 55–64. Springer Berlin Heidelberg, 1994.

[66] Biham, E.: New types of cryptanalytic attacks using related keys. In Helleseth, T. (ed.), *Advances in Cryptology — EUROCRYPT '93*, vol. 765 of *Lecture Notes in Computer Science*, pp. 398–409. Springer Berlin Heidelberg, 1994.

[67] Biryukov, A., Khovratovich, D.: Related-key cryptanalysis of the full AES-192 and AES-256. In Matsui, M. (ed.), *Advances in Cryptology – ASIACRYPT 2009*, vol. 5912 of *Lecture Notes in Computer Science*, pp. 1–18. Springer Berlin Heidelberg, 2009.

[68] Bogdanov, A., Khovratovich, D., Rechberger, C.: Biclique cryptanalysis of the full AES. In Lee, D., Wang, X. (eds.), *Advances in Cryptology – ASIACRYPT 2011*, vol. 7073 of *Lecture Notes in Computer Science*, pp. 344–371. Springer Berlin Heidelberg, 2011.

[69] Phan, R.-W.: Advanced slide attacks revisited: Realigning slide on DES. In Dawson, E., Vaudenay, S. (eds.), *Progress in Cryptology – Mycrypt 2005*, vol. 3715 of *Lecture Notes in Computer Science*, pp. 263–276. Springer Berlin Heidelberg, 2005.

[70] Biryukov, A., Wagner, D.: Slide attacks. In Knudsen, L. (ed.), *Fast Software Encryption*, vol. 1636 of *Lecture Notes in Computer Science*, pp. 245–259. Springer Berlin Heidelberg, 1999.

[71] Saarinen, M.-J.: A chosen key attack against the secret S-boxes of GOST. *Unpublished manuscript*, 1998.

[72] Kazymyrov, O.: Practical recovery of long-term keys of the GOST 28147 cipher based on slide attack. In *Science and social problems: Computerization and information technology*, pp. 272–273. Kharkiv National University of Radioelectronics, 2011. (In Russian).

[73] Isobe, T., Shibutani, K.: All subkeys recovery attack on block ciphers: Extending meet-in-the-middle approach. In Knudsen, L., Wu, H. (eds.), *Selected Areas in Cryptography*, vol. 7707 of *Lecture Notes in Computer Science*, pp. 202–221. Springer Berlin Heidelberg, 2013.

[74] Oliynykov, R.: *Methods for analysis and synthesis of perspective symmetric cryptographic transformations*. A thesis for a doctor of technical sciences degree in the specialty 05.13.05 – computer systems and components, Kharkiv National University of Radio Electronics, Ukraine, 2014. (In Russian).

[75] Zhou, Y., Feng, D.: Side-channel attacks: Ten years after its publication and the impacts on cryptographic module security testing. *Cryptology ePrint Archive, Report 2005/388*, 2005. `http://eprint.iacr.org/`.

[76] Nikova, S., Rechberger, C., Rijmen, V.: Threshold implementations against side-channel attacks and glitches. In Ning, P., Qing, S., Li, N. (eds.), *Information and Communications Security*, vol. 4307 of *Lecture Notes in Computer Science*, pp. 529–545. Springer Berlin Heidelberg, 2006.

[77] Kazymyrov, O., Kazymyrova, V., Oliynykov, R.: A method for generation of high-nonlinear S-boxes based on gradient descent. In *Mathematical Aspects of Cryptography*, vol. 5, pp. 71–78. Steklov Mathematical Institute, 2014.

[78] Nyberg, K.: Perfect nonlinear S-boxes. In Davies, D. (ed.), *Advances in Cryptology - EUROCRYPT'91*, vol. 547 of *Lecture Notes in Computer Science*, pp. 378–386. Springer Berlin Heidelberg, 1991.

[79] Chabaud, F., Vaudenay, S.: Links between differential and linear cryptanalysis. In *Advances in Cryptology—EUROCRYPT'94*, pp. 356–365. Springer, 1995.

[80] Budaghyan, L.: *Construction and analysis of cryptographic functions*. Habilitation Thesis, University of Paris 8, France, 2013.

[81] Kazymyrov, O., Oliynykov, R.: Choosing substitutions for symmetric cryptographic algorithms based on the analysis of teir algebraic properties. In *Mathematical modeling. Information Technology. Automated control systems.*, vol. 925, pp. 79–86. V. N. Karazin Kharkov National University, Ukraine, 2010. (In Russian).

[82] Budaghyan, L., Kazymyrov, O.: Verification of restricted EA-equivalence for vectorial Boolean functions. In Özbudak, F., Rodríguez-Henríquez, F. (eds.), *Arithmetic of Finite Fields*, vol. 7369 of *Lecture Notes in Computer Science*, pp. 108–118. Springer Berlin Heidelberg, 2012.

[83] Carlet, C., Charpin, P., Zinoviev, V.: Codes, bent functions and permutations suitable for DES-like cryptosystems. In *Designs, Codes and Cryptography*, vol. 15, pp. 125–156. Kluwer Academic Publishers, 1998.

[84] Krause, M.: BDD-based cryptanalysis of keystream generators. In Knudsen, L. (ed.), *Advances in Cryptology — EUROCRYPT 2002*, vol. 2332 of *Lecture Notes in Computer Science*, pp. 222–237. Springer Berlin Heidelberg, 2002.

[85] Stegemann, D.: Extended BDD-based cryptanalysis of keystream generators. In Adams, C., Miri, A., Wiener, M. (eds.), *Selected Areas in Cryptography*, vol. 4876 of *Lecture Notes in Computer Science*, pp. 17–35. Springer Berlin Heidelberg, 2007.

[86] Schilling, T., Raddum, H.: Solving compressed right hand side equation systems with linear absorption. In Helleseth, T., Jedwab, J. (eds.), *Sequences and Their Applications – SETA 2012*, vol. 7280 of *Lecture Notes in Computer Science*, pp. 291–302. Springer Berlin Heidelberg, 2012.

[87] Cid, C., Murphy, S., Robshaw, M.: Small scale variants of the AES. In Gilbert, H., Handschuh, H. (eds.), *Fast Software Encryption*, vol. 3557 of *Lecture Notes in Computer Science*, pp. 145–162. Springer Berlin Heidelberg, 2005.

[88] Shirai, T., Shibutani, K.: On the diffusion matrix employed in the Whirlpool hashing function. *NESSIE public reports,* 2003. `https://www.cosic.esat.kuleuven.be/nessie/nessie/reports/phase2/whirlpool-20030311.pdf`.

[89] Augot, D., Finiasz, M.: Direct construction of recursive MDS diffusion layers using shortened BCH codes. *In Pre-proceedings of Fast Software Encryption (FSE 2014),* 2014.

[90] Stein, W., et al.: Sage mathematics software (version 6.2). *The Sage Development Team,* 2014. `http://www.sagemath.org`.

[91] Biryukov, A., De Cannière, C., Braeken, A., Preneel, B.: A toolbox for cryptanalysis: Linear and affine equivalence algorithms. In Biham, E. (ed.), *Advances in Cryptology — EUROCRYPT 2003*, vol. 2656 of *Lecture Notes in Computer Science*, pp. 33–50. Springer Berlin Heidelberg, 2003.

[92] Burnett, L.: *Heuristic optimization of Boolean functions and substitution boxes for cryptography*. Ph.D. thesis, Queensland University of Technology, Australia, 2005.

[93] Lafitte, F., Heule, D. V., Hamme, J. V.: Cryptographic Boolean functions with R. In *The R Journal*, vol. 3, pp. 44–47. June, 2011.

[94] Mishra, P. R.: Calculating cryptographic degree of an S-box. *Cryptology ePrint Archive, Report 2014/145*, 2014. http://eprint.iacr.org/.

[95] Chmiel, K.: Fast computation of approximation tables. In Saeed, K., Pejaś, J. (eds.), *Information Processing and Security Systems*, pp. 125–134. Springer US, 2005.

[96] Yu, Y., Wang, M., Li, Y.: Constructing differential 4-uniform permutations from know ones. *Cryptology ePrint Archive, Report 2011/047*, 2011. http://eprint.iacr.org/.

[97] Tesař, P.: A new method for generating high non-linearity S-boxes. In *Radioengineering*, vol. 19, pp. 23–26. Brno University of Technology, 2010. http://www.radioeng.cz/fulltexts/2010/10_01_023_026.pdf.

[98] Matyukhin, D., Rudskoy, V., Shishkin, V.: A perspective hashing algorithm. *Materials of XII scientific conference RusCrypto'2010*, 2010. http://www.ruscrypto.ru/resource/summary/rc2010/ruscrypto_2010_054.zip. (In Russian).

[99] AlTawy, R., Youssef, A. M.: Integral distinguishers for reduced-round Stribog. *Cryptology ePrint Archive, Report 2013/648*, 2013. http://eprint.iacr.org/.

[100] Kazymyrov, O., et al.: Source code of the cross-platform implementation of Stribog. *GitHub repository*, 2013. https://github.com/okazymyrov/stribog.

[101] Kazymyrov, O., Leontiev, S., Popov, V., Smyshlyaev, S.: On creating effective software implementations of national cryptographic

standards. *Materials of XV scientific conference RusCrypto,* 2013. http://www.ruscrypto.ru/accotiation/archive/rc2013. (In Russian).

[102] Guo, J., Nikolic, I., Peyrin, T., Wang, L.: Cryptanalysis of Zorro. *Cryptology ePrint Archive, Report 2013/713*, 2013. http://eprint.iacr.org/.

[103] Hong, J., Kim, W.-H.: TMD-tradeoff and state entropy loss considerations of streamcipher MICKEY. In Maitra, S., Veni Madhavan, C., Venkatesan, R. (eds.), *Progress in Cryptology - INDOCRYPT 2005*, vol. 3797 of *Lecture Notes in Computer Science*, pp. 169–182. Springer Berlin Heidelberg, 2005.

[104] Jansen, C., Helleseth, T., Kholosha, A.: Cascade jump controlled sequence generator and Pomaranch stream cipher. In Robshaw, M., Billet, O. (eds.), *New Stream Cipher Designs*, vol. 4986 of *Lecture Notes in Computer Science*, pp. 224–243. Springer Berlin Heidelberg, 2008.

[105] Jansen, C.: The state space structure of the MICKEY stream cipher. *Proceedings of the 32rd WIC Symposium on Information Theory in the Benelux and The 1st Joint WIC/IEEE Symposium on Information Theory and Signal Processing in the Benelux,* 2011.

[106] Jansen, C.: Analysis of the nonlinear function of the Mickey S-register. *Proceedings of the 33rd WIC Symposium on Information Theory in the Benelux and The 2nd Joint WIC/IEEE Symposium on Information Theory and Signal Processing in the Benelux,* pp. 60–67, 2012.

[107] Olshansky, D.: Introduce GOST R 34.11-2012 hash function. *Electronic source,* 2014. http://rt.openssl.org/Ticket/Display.html?id=3311.

Scientific Results

I

PAPER I

ALGEBRAIC ATTACKS USING BINARY DECISION DIAGRAMS *

Oleksandr Kazymyrov Håvard Raddum

*Kazymyrov, O., Raddum, H.: Algebraic attacks using binary decision diagrams. *In Pre-proceedings of BalkanCryptSec 2014*, pp. 31–44, 2014.

Algebraic Attacks Using Binary Decision Diagrams

Oleksandr Kazymyrov[†] Håvard Raddum[‡]

[†] University of Bergen, Norway
Oleksandr.Kazymyrov@ii.uib.no
[‡] Simula Research Laboratories, Norway
haavardr@simula.no

Abstract

Algebraic attacks have been developed against symmetric primitives during the last decade. In this paper we represent equation systems using binary decision diagrams, and explain techniques for solving them. Next, we do experiments with systems describing reduced versions of DES and AES, as well as systems for the problem of determining EA-equivalence. We compare our results against Gröbner basis and CryptoMiniSat.

Keywords: binary decision diagram, block cipher, algebraic attack, symmetric primitives.

1. Introduction

The main idea of modern algebraic attacks is to describe an encryption scheme via a system of equations and solve it. Equivalence of finding a key to a cryptosystem and solving a system of equations was first mentioned by Claude Shannon [1]. However, algebraic attacks against cryptographic primitives began to develop actively only in the early 2000s. Several methods to attack hash functions, stream and block ciphers have been described in [2–10]..

In the middle of the 20th century it was proposed to use binary decision diagrams (BDDs) for representing Boolean functions [11, 12]. This representation has several advantages. Many logical operations on BDDs can be implemented by polynomial-time graph manipulation algorithms, and the memory consumption can be extremely low, even for very complex Boolean functions [12]. Most modern cryptographic primitives are based on binary

logic because of the large spread of binary computers. Therefore, the description of cryptographic transformations using Boolean or vectorial Boolean functions is an easy task.

Several attacks based on BDDs exist for stream ciphers. Their efficiency was demonstrated both for general methods and for particular cases on A5/1, E0 and Trivium [13, 14]. In this paper we extend previous results on block ciphers and present new specific strategies and approaches for solving systems of equations based on BDDs.

We apply the proposed methods on DES with reduced number of rounds, on MiniAES (a small variant of Rijndael) and on the problem of determining EA-equivalence. Our experiments on DES allow us to break six rounds in approximately one minute on a MacBook Air 2013. This is a factor 2^{20} improvement over the best earlier algebraic attack on DES using MiniSAT [5]. There have been several earlier attempts to break MiniAES [7, 15, 16]. Approaches that exploit the short key in MiniAES (only 16 bits) succeed very quickly, but the general methods of F4 and XL/XSL failed to solve systems representing more than one round of MiniAES. The approach we use in this paper does not exploit the short key, while still solving systems representing 10 rounds of MiniAES using approximately 45 minutes and 8GB of memory.

The rest of the paper is organized as follows. Section 2 explains BDDs and the fundamental operations we do on them. Section 3 describes our approach for solving BDD systems, and introduces some solving strategies. Section 4 gives the details and results of our algebraic attack against DES and MiniAES as well as the EA-equivalence problem, comparing complexities against SAT-solver and Gröbner base techniques. Finally, Section 5 concludes the paper and give some directions for further research.

2. Binary Decision Diagram Fundamentals

The literature discusses several variants of BDDs. For clarity we will always mean a *zero-suppressed, reduced*, and *ordered* BDD in this paper. A comprehensive treatment of BDDs can be found in [12]. In this section we only give a brief description with emphasis on visualization and our use of a BDD.

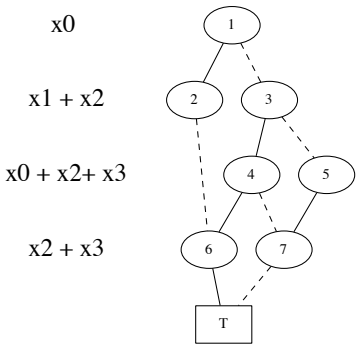


Fig. 1: *Example of a BDD with four levels.*

2.1. Binary Decision Diagrams

A BDD is a directed acyclic graph. Exactly one node in the graph, called the *source node*, has no incoming edges, and exactly one node in the graph, called the *sink node*, has no outgoing edges. All nodes except for the sink node are called *internal* nodes, and have one or two outgoing edges, called the 0-*edge* and/or the 1-*edge*. In most other descriptions of BDDs, each internal node is associated with a variable. In this paper each internal node will be associated with a **linear combination** of variables. There are no edges between nodes associated with the same linear combination.

When visualizing a BDD, we draw the graph from top to bottom, with the source node on top, the sink node at the bottom, and all edges directed downwards. All internal nodes are organized in horizontal *levels* between the sink and source nodes. One level consists of all nodes associated to one particular linear combination, and we write the linear combination to the left of the level. Dotted edges indicate 0-edges while solid lines indicate 1-edges. An example of a BDD with four levels associated with linear combinations in four variables is shown in Fig. 1.

In the literature there are various ways to understand a BDD. Some interpret a BDD to represent a family of sets while others see a BDD as an efficient encoding of a Boolean function. In this paper we put emphasis on the fact that a path from the source to the sink node assigns values to the linear combinations of the levels. If we choose the b-edge ($b \in \{0,1\}$) out from a node, we assign the value b to the linear combination associated with the level of the node. Any path from the source to the sink node gives values to

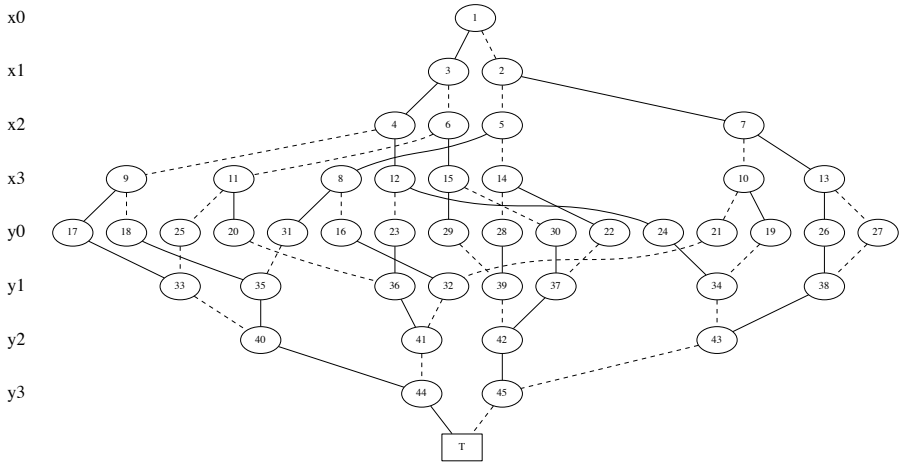


Fig. 2: *BDD representation of the S-box {5,C,8,F,9,7,2,B,6,A,0,D,E,4,3,1}.*

the linear combinations and can be regarded as a right-hand side in a system of linear equations.

2.2. Representing an S-box as a BDD

We are interested in finding a BDD that represents a given S-box with n input bits and m output bits. Let the input bits and output bits of the S-box be $x_0, \ldots, x_{n-1}$ and $y_0, \ldots, y_{m-1}$, respectively. Let the first n levels be associated with $x_0, \ldots, x_{n-1}$ (x_0 for the source node and x_{n-1} for level n), and build a complete binary tree from x_0 to x_{n-1}. Next, assign $y_0, \ldots, y_{m-1}$ to the m lowest levels (with y_0 at the highest of these), and build a complete binary tree upwards from the sink node to the y_0-level, branching in 0-edges and 1-edges. Then there will be only one path from a given node at the y_0-level to the sink node. There will be 2^m nodes at the y_0-level, each representing a unique path to the sink node, assigning values to $y_0, \ldots, y_{m-1}$.

Any path from the source node down to level x_{n-1} will assign values to the input bits $x_0, \ldots, x_{n-2}$. Selecting a 0-edge or a 1-edge out of a node at the x_{n-1}-level will complete the assignment of input bits. This edge is connected to the node at the y_0-level whose unique path to the sink node will give the correct output of the S-box. Joining all nodes at the x_{n-1}-level to all nodes at the y_0-level in this way will complete the construction of the BDD. Fig. 2 shows an example of a BDD representing a 4×4 S-box.

2.3. Basic Operations on a BDD

We must be able to run the *reduction* algorithm [17] on a BDD, bringing the BDD into a reduced state. The reduction algorithm basically merges nodes representing equivalent Boolean functions, thus minimizing the number of nodes in the BDD. For a fixed order of the linear combinations, a reduced BDD is unique. There are two other operations that forms the core of *linear absorption* (explained later). Both were described in [18], but we repeat them briefly here for completeness.

2.3.1. Swapping levels.

This operation swaps the linear combinations at two adjacent levels, and was first described in [19], using single variables. When changing the order of the levels, nodes and edges must be re-arranged in the BDD to preserve the underlying function. Fortunately, swapping levels is a local operation, meaning that only nodes and edges at the two involved levels need to be touched while the rest of the BDD remains intact. The algorithm for swapping two levels has low complexity, but the number of nodes on the lowest level may double in the worst case.

After swapping two levels, the BDD may not be in the reduced state, and it may be necessary to run the reduction algorithm. Hence, the number of nodes in the BDD after swapping two levels may increase or decrease. By repeatedly swapping levels one may put the set of linear combinations for the levels into any desired order. Finding the order of levels that give the fewest nodes is an NP-complete problem [20].

2.3.2. Adding levels.

Traditionally, the levels in a BDD have been associated with single variables and not linear combinations. It has therefore not been natural to think of "adding" one level onto another. This changes when we have linear combinations associated with the levels. If l_1 and l_2 are two linear combinations associated to two adjacent levels (l_1 above l_2), we are interested in replacing l_2 with $l_1 + l_2$. The algorithm for adding levels was first described in [18], and follows the same logic as with swapping levels. Nodes and edges at the levels for l_1 and l_2 must be rearranged to preserve the underlying function, but the rest of the BDD remains the same. The complexity is similar to swapping, and the number of nodes at the new level associated to $l_1 + l_2$ may double in

the worst case. The reduction algorithm should be run after adding levels to make sure the BDD remains in a reduced state.

3. Solving Systems of Equations with Linear Absorption

By repeatedly swapping and adding linear combinations, we can essentially do all linear operations on the linear combinations of the BDD. We can, for instance, perform Gaussian elimination on the set of linear combinations. As will become clear in the following, a barrier to find a solution to a system of equations arises when there are dependencies among the linear combinations of the levels in a BDD. We overcome this problem by using linear absorption. The technique was first described in [18], but we include an example of the procedure here as it is central in our approach to solve non-linear equation systems.

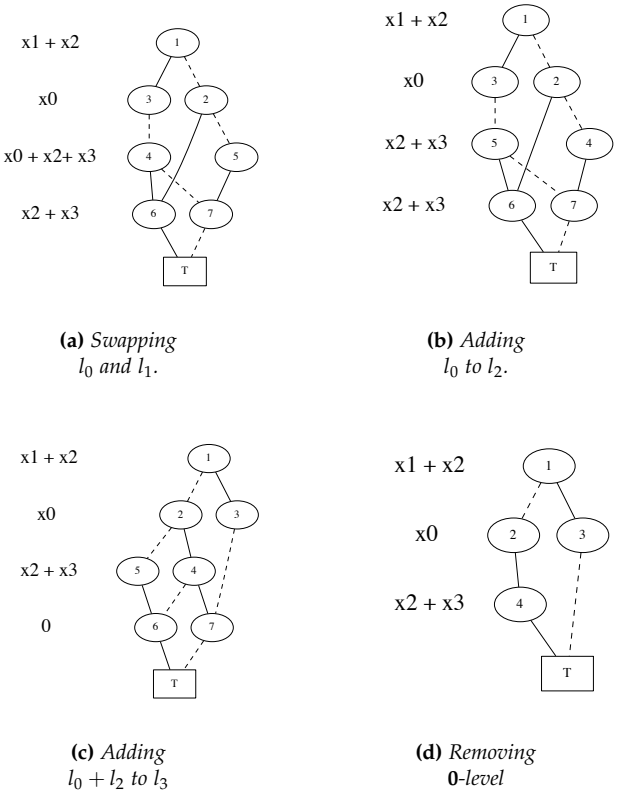
3.1. Absorbing One Linear Dependency

The attentive reader will have noticed that the linear combinations in the example BDD in Section 2.1 are not independent. If we label them l_0, l_1, l_2, l_3 from top to bottom we have $l_0 + l_2 + l_3 = 0$. Thus, when we select a path in the BDD and create the corresponding linear system of equations, we may or may not get a consistent system. If the values assigned to l_0, l_2 and l_3 sum to 0 we get a solution, if not, the system is inconsistent. We use linear absorption to remove all paths that yield inconsistent systems as follows.

First, swap l_0 and l_1 to obtain the BDD in Fig. 3a. Next, use addition of linear combinations to add l_0 onto l_2, and obtain the BDD in Fig. 3b. Finally, we use addition again to add $l_0 + l_2$ to l_3. This creates the **0**-vector as linear combination for the lowest level, resulting in the BDD shown in Fig. 3c.

When selecting a path in the BDD now, it does not make sense to choose a 1-edge out of a node on the level associated with **0**. Such a path would yield a "$0 = 1$"-assignment. Hence we can delete all outgoing 1-edges from the nodes at the **0**-level. Now we are certain that any remaining path will yield a system of linear equations that is consistent with the linear dependency $l_0 + l_2 + l_3 = 0$.

Moreover, the whole level associated with **0** can be removed. It is easy to show that the Boolean function represented by a node on this level is



(a) *Swapping* l_0 *and* l_1.

(b) *Adding* l_0 *to* l_2.

(c) *Adding* $l_0 + l_2$ *to* l_3

(d) *Removing* **0**-*level*

Fig. 3: *Absorbing one linear dependency.*

equal to the function for the node pointed to by the 0-edge. Hence, all nodes on the **0**-level can be merged with their children along the 0-edge, and the level disappears. We say that the linear dependency $l_0 + l_2 + l_3 = 0$ has been *absorbed*. The resulting BDD for our example is shown in Fig. 3d.

In general, the removal of 1-edges from a level associated with the **0**-vector may create internal nodes with no incoming edges. We call these *orphan* nodes as they have no parents. After absorbing one linear dependency, all orphan nodes, and subgraphs only reachable through an orphan node, should be removed as part of the reduction procedure.

If there are several dependencies among the linear combinations in a BDD, we can easily find all and absorb them one after another. When all dependencies have been absorbed, we know that *any* remaining path in the BDD will give a consistent linear system of equations, which in turn is solved to find the values of the actual variables.

3.2. Building and Solving Equation Systems as BDDs

It is rather straight-forward to build a system of equations representing a cryptographic primitive as a set of BDDs. For an encryption algorithm, the user-selected key bits become variables. We also look at the cipher blocks between the rounds of the primitive. We assign variable names to the bits between rounds, such that the input and output bits of all non-linear components can be written as linear combinations of variables.

For each non-linear component we then construct the corresponding BDD, like explained for S-boxes in Section 2.2. We replace the x_i and y_j with the linear combinations actually occuring in the cipher. After this we are left with a set of BDDs with linear combinations from the same pool of variables.

To proceed with finding a solution to the system we must join the BDDs together. There exist algorithms for joining two BDDs [17][12, p. 16], but they are somewhat complex, and assume single variables associated with the levels. We do it in a much simpler way:

- To join two BDDs, just replace the sink node of one with the source node of the other.

With this simple operation, we can easily string together some or all BDDs in a system and get fewer, or only one, BDD(s) in the set. If we join all BDDs together, finding a solution is equivalent to finding a path in the joined BDD that gives a right hand side yielding a consistent linear system. As can be expected, for interesting systems there will be many dependencies among the linear combinations in a fully joined BDD, so finding a path respecting all these dependencies is not trivial. We can, however, try to handle this problem with linear absorption, and if we can absorb all linear dependencies in the BDD we know that any remaining path will give a consistent linear system. The algorithm for solving a system of BDDs can then be summed up as:

1. Join BDDs.
2. Absorb all linear dependencies.

3. Select path and solve resulting system of linear equations.

3.3. Complexity

The first and third steps in the general solving algorithm are easy (assuming a modest number of BDDs and variables, which is the case even for full scale ciphers). Hence the second step must be hard if our cryptographic primitives are to remain secure. What we get when joining all BDDs together is a very long and very slim BDD, basically just a string of many small BDDs. The number of nodes at one level may double when adding or swapping two linear combinations, and after absorbing a whole linear dependency the total number of nodes may, in the worst case, double. This seems to lead to exponential growth in the number of dependencies absorbed, but in practice the number of nodes after absorbing a linear dependency is very far from doubling. Remember, the number of nodes may also *decrease* when applying a swap or an add operation.

If we expect a unique or only a few solutions, the BDD after absorbing all dependencies will have only one or a few paths. A BDD with only one path has only one node at each level, tied together with a string of 0- and 1-edges. Since all the systems we are interested in have very few solutions, we know that the number of nodes must decrease sharply before the last dependencies are absorbed. This means we will always reach some tipping point when absorbing dependencies, where the number of nodes in the BDD starts to decrease.

We take as our measure of complexity the largest number of nodes that a BDD contained during linear absorption. This is a measure of memory complexity, and is not equivalent to the time it takes to solve a system. On the other hand, there is no guessing involved in our solving method, and no operations that must be repeated an exponential number of times. Memory rather than time is the resource that constrains us. With 8GB of RAM it is hard to find a system where our solver runs for over an hour without either finishing or running out of memory. Therefore, we believe the largest number of nodes we had during the solving process is the most meaningful measure of complexity.

3.4. Solving strategies

When joining together many BDDs and absorbing all linear dependencies, the solving complexity depends heavily on the order the BDDs are joined. Finding the ordering of BDDs that gives the minimum complexity is probably a hard problem. During our experiments we have not found a strategy for ordering that is universally best. However, we describe here some strategies for how to join and absorb with the aim to keep the complexity down.

3.4.1. Automatic Ordering.

This is a default strategy, that can be applied to any system and does not require any deeper understanding of how the BDDs have been made. The procedure is to look for the subset of BDDs with the smallest total number of nodes, that still contains some dependencies. When this subset is found, we join these BDDs and absorb all dependencies. The number of BDDs in the set will then decrease by at least one, and we repeat the procedure until there is only one BDD left and all dependencies have been absorbed.

3.4.2. Divide-and-Conquer.

This strategy takes the approach that it is always easy to join a *few* BDDs together and absorb all dependencies. To solve the system though, one sooner or later has to join all BDDs, and absorb all remaining dependencies. The assumption is that the true complexity of solving the system will only appear when all BDDs are joined. Thus we would like to have only a minimum of dependencies left when we are forced to join all BDDs together.

The divide-and-conquer strategy is to split the system in two (roughly) equally large halves, such that there are many dependencies within each half but only a few that use linear combinations from BDDs in both halves. We can then attack each half independently, trying to absorb all dependencies. If we succeed, we are left with one BDD with only independent linear combinations in each half. These can now be joined, and we absorb the relatively few remaining dependencies in the full BDD. When attacking one half, we use the Divide-and-Conquer technique recursively, treating the half as a complete system. The recursion stops when a "system" only contains one or two of the original BDDs.

Finding the optimal way to split a system in two equally sized parts seems to be a hard problem in general. However, knowledge of how the system has been constructed can help in this regard, as we will see with DES.

3.4.3. Finding Good Joining Order by Cryptanalysis.

When we are trying to solve a system representing a cryptographic primitive, analysis of the primitive may help in deciding a good order of how to join the BDDs. The strategy is simply to decide on an order for the original BDDs, join all of them into one long BDD, and absorb all linear dependencies. The order of the BDDs should be such that each linear dependency only involves linear combinations on levels that are relatively close to each other. Absorbing each linear combination then becomes a somewhat local operation that only affects a small part of the BDD, keeping the complexity down.

4. Application of the Algebraic Attack Based on BDDs

This section describes practical aspects and results of the proposed attack for DES and MiniAES as well as the time comparison of solving extended affine equivalence (EA) problem using Gröbner basis, CrytoMiniSat and BDD approaches.

4.1. A Practical Attack on Reduced DES

Previous results on solving DES systems can be found in [5] where the authors solve a 6-round version, and in [21] where DES with 6, 7 and 8 rounds are attacked. In all papers it was necessary to fix 20+ of the key bits to their correct values for the attacks to work, reducing the effective key size to at most 36 bits. The actual solving of equation systems was done by MiniSAT [22].

Our best result is that we can solve a 6-round system using 8 chosen plaintext/ciphertext pairs without fixing or guessing any variables. The average complexity is $2^{20.571}$ nodes. Solving the system for 6-round DES with 8 chosen plaintexts takes approximately one minute on a MacBook Air 2013 with 8GB of memory.

4.1.1. Constructing DES System of Equations.

We assume the reader is familiar with the operations of DES [23]. The only non-linear part of DES is the application of the eight 6×4 S-boxes in each round. We assign variables to the output of the S-boxes in each round, except for the last two whose output can be expressed as linear conbinations of other variables and ciphertext bits. The key schedule of DES is linear, so we only need to assign variables to the 56 user-selected key bits. After this the input and output bits of each S-box can be expressed as linear combinations of variables, and we construct a BDD for each S-box as described in Section 2.2. Each BDD contains approximately 185 nodes after reduction, the eight different S-boxes vary slightly in size.

4.1.2. Solving Strategy.

The solving strategy we found to work best for DES is Divide-and-Conquer. We then need to divide our system in two equally big halves, and the key schedule of DES gives a clear hint on how to do this: One half of the 56 key bits *only* appears in the inputs to S-boxes $1-4$, while the other half only appears in the S-boxes $5-8$. This applies to all rounds. For each round, we put the BDDs representing S-boxes $1-4$ into the set A_0, and the BDDs for S-boxes $5-8$ into the set B_0. Then we try to solve A_0 and B_0 independently, using Divide-and-Conquer again. For dividing A_0, we have found (by exhaustive search) that the best division is to group together the same two S-boxes from odd-numbered rounds, i.e. S-boxes 1 and 2, and the other S-boxes (3 and 4) from even-numbered rounds, into set A_1. The other BDDs from A_0 go into the set B_1. See Fig. 4 for a sketch of the division used. B_0 is re-divided similarly, and further divisions are done by exhaustive search on the fly.

4.1.3. Several Plaintexts.

When using several plaintext/ciphertext pairs, we build one DES system for each. These systems will have the same 56 key-bit variables, but variables representing internal state will, in general, be different for each plaintext. However, if we carefully choose the difference between the plaintexts we can reuse a lot of internal variables across different systems. For producing up to eight different plaintexts, we vary only three bits in the left half. Then the input to the first round will be equal for each text, and the difference in the input to the second round will only be in three bits. These bits are chosen so

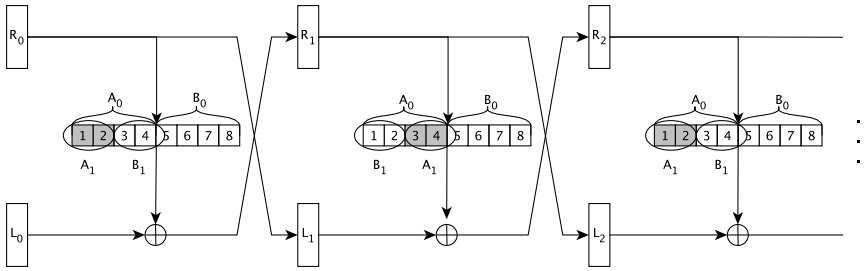


Fig. 4: *Division used for Divide-and-Conquer strategy for DES.*

they only affect one S-box each. Tracing differences further we find that we may reuse variables across systems as far as into the fourth round.

We merge the different systems by joining all BDDs arising from the same S-box in the same round. As these share the same key variables, and often many other variables as well, there are many linear dependencies among the levels in the joined BDD. These dependencies are absorbed, and after this pre-processing we are left with a system of $8r$ BDDs representing an r-round version of DES, regardless of the number of chosen plainexts used.

4.1.4. Extracting Linear Equations.

When using more than four plainetxts in the experiments, we observed that the heaviest step while solving did not occur when joining the sets A_0 and B_0, but rather when joining A_1 and B_1. After all dependencies in A_0 had been absorbed the resulting BDD was very slim, with many levels only containing one node.

If a level only has outgoing b-edges (which is often the case with one-node levels) we know that the associated linear combination is equal to b, ($b \in \{0,1\}$). We can use this linear equation to eliminate a variable from the system. Extracting as many linear equations as possible and eliminating variables after all dependencies in A_0 had been absorbed, it became trivial to absorb all dependencies in B_0, and the full system.

4.1.5. Results for DES Experiments.

We have solved systems representing DES for $4, 5$ and 6 rounds, using $1-8$ plaintext/ciphertext pairs. For each choice of rounds and number of plaintexts

Table 1: *Complexities for solving reduced-round DES-systems. Each cell shows the minimum,* **average** *and maximum complexity observed over 100 instances.*

rounds \ # texts	1	2	3	4	5	6	7	8
4	$2^{22.651}$ $\mathbf{2^{22.715}}$ $2^{22.770}$	$2^{10.800}$ $\mathbf{2^{14.506}}$ $2^{17.473}$	$2^{9.281}$ $\mathbf{2^{10.606}}$ $2^{13.006}$	$2^{9.585}$ $\mathbf{2^{10.257}}$ $2^{12.029}$	$2^{9.748}$ $\mathbf{2^{9.805}}$ $2^{9.892}$	$2^{9.976}$ $\mathbf{2^{10.070}}$ $2^{10.412}$	$2^{10.103}$ $\mathbf{2^{10.203}}$ $2^{10.978}$	$2^{10.283}$ $\mathbf{2^{10.381}}$ $2^{10.446}$
5		$2^{19.472}$ $\mathbf{2^{22.110}}$ $2^{23.805}$	$2^{13.831}$ $\mathbf{2^{16.455}}$ $2^{19.329}$	$2^{11.440}$ $\mathbf{2^{13.526}}$ $2^{15.618}$	$2^{12.126}$ $\mathbf{2^{13.995}}$ $2^{16.633}$	$2^{12.289}$ $\mathbf{2^{14.212}}$ $2^{16.758}$	$2^{12.583}$ $\mathbf{2^{14.410}}$ $2^{16.882}$	$2^{12.749}$ $\mathbf{2^{14.704}}$ $2^{17.414}$
6						$2^{24.506}$ $\mathbf{2^{24.929}}$ $2^{25.352}$	$2^{22.206}$ $\mathbf{2^{22.779}}$ $2^{24.324}$	$2^{19.932}$ $\mathbf{2^{20.571}}$ $2^{21.915}$

we randomly generated and solved 100 systems, recording their complexities. The minimum, maximum and average (in bold) complexities observed are summarized in Table 1.

There are rather large variations in complexities inside most cells in Table 1. In each cell, the key and one plaintext were chosen at random for each of the 100 instances, and some choices give much lower complexity than others. We can not explain the differences, and have not been able to identify which choices lead to low solving complexity.

4.2. A Practical Attack on Scaled-Down Version of AES

There are many scaled-down versions of the AES cipher (MiniAES). The first one that follows Rinjdael's description was proposed in [24]. A few years later Cid et al. analyzed many small AES variants in [7], and Elizabeth Kleiman tried to attack MiniAES in her master and doctoral theses [15, 25]. Also, an equation system representing MiniAES was solved in [16].

4.2.1. Description of MiniAES.

The constants of our studied encryption model are the block length (16 bits) and the key size (16 bits). This scaled-down cipher corresponds to AES-128 [26]. In contrast to original AES, the mini version is a nibble (4 bits) oriented

cipher with the state represented as a 2×2 matrix. The round function consists of four routines: AddroundKey (AK_k), SubBytes (SB), ShiftRows (SR) and MixColumns (MC). The encryption algorithm can be described as

$$E_K(M) = \prod_{i=1}^{r} (AK_{k_i} \circ MC \circ SR \circ SB) \circ AK_{k_0}(M),$$

where r is the number of rounds. The substitution and MDS matrix were taken from [24].

4.2.2. Constructing System of BDDs for MiniAES.

Unlike DES, MiniAES has non-linear components in the key schedule. We assign variables to the output of the S-boxes in each round, except for the last one. Additionally, we have to add 8 extra variables for each round key, except the first one with 16 bits of the user-selected key. Then the number of BDDs and variables is equal to $2r + 4r = 6r$ and $(16 + 8r) + 16(r - 1) = 24r$, respectively.

4.2.3. Solving Strategy.

The best strategy we found for solving the MiniAES systems was to determine a good order of BDDs by cryptanalysis, join all BDDs and do a full absorption of all dependencies.

The order was found by carefully studying which variables that appear in each BDD, both from the key schedule and the encryption function. Variables from the cipher state only appear in two consecutive rounds. It is therefore clear that the four BDDs from the same round should be joined close to each other, and also close to the two BDDs from the key schedule producing the round key used for the round. The BDDs were put together in groups of six this way, following the rounds of the cipher.

The order of the BDDs in each group was determined by looking at which variables that appear in each individual BDD. Two BDDs that share many variables should be adjacent in the final order. After doing this for the four, five and six round versions it became clear that a pattern emerged for the joining order. This pattern was followed for the higher number of rounds.

Table 2: *Complexities of solving MiniAES systems.*

Rounds	4	5	6	7	8	9	10
Complexity	$2^{22.404}$	$2^{23.051}$	$2^{23.440}$	$2^{24.154}$	$2^{24.217}$	$2^{24.862}$	$2^{24.961}$

4.2.4. Results for MiniAES Experiments.

The complexities for solving MiniAES systems for various rounds are summed up in Table 2. Of course, MiniAES only has a 16-bit key and can be broken very fast using for instance CryptoMiniSAT, which essentially does a very intelligent brute force on the key [27, p. 250-251]. In [16], the authors create the polynomials for the ciphertext bits using only the user-selected key bits as variables. This results in 16 polynomials in 16 variables containing approximately 2^{15} terms each, that can be solved using PolyBoRi. Our algorithm does not take advantage of the short key, and should be compared to the earlier attacks described in [7, 15].

For each number of rounds we solved 10 different instances, using 1 known plaintext/ciphertext pair. We were not able to reduce the complexities by using more pairs. The observed complexities for one particular number of rounds did not vary, so the minimum, maximum and average complexities are all the same. We have also changed the S-box and the MixColumn matrix to see if other choices affected the complexity, but we found the complexity remains the same for all variants tried.

It has been observed before that for one plaintext/ciphertext pair in scaled-down AES versions there may be more than one key that encrypts the given plaintext into the given ciphertext. This was shown in our experiments as well, often we had two, three or even four solutions to our systems.

4.3. Problem of determining EA-equivalence

To get a good comparison against other solvers we have chosen the problem of EA-equivalence [28]. This problem is interesting in cryptography, and it can be solved via non-linear systems of equations. There are no special variables in these systems, like key bits, so we get a fair comparison between CryptoMiniSAT, Gröbner bases and the BDD method.

Two functions are EA-equivalent if the following equation holds for all $x \in GF(2)^n$

$$F(x) = M_1 \cdot G(M_2 \cdot x \oplus V_2) \oplus M_3 \cdot x \oplus V_1, \tag{1}$$

Table 3: *Time complexity for solving EA-equivalence problem*

#	n	Number of solutions	Seconds used to solve		
			BDD	GB	SAT
1	4	2	$2^{4.05}$	$2^{3.8}$	$2^{13.71}$
2	4	60	$2^{4.86}$	†	$2^{16.77}$
3	4	2	$2^{3.92}$	$2^{3.9}$	$2^{12.08}$
4	5	1	$2^{10.20}$	$2^{11.43}$	-
5	5	155	$2^{10.48}$	†	-

where elements of $\{M_1, M_2, M_3, V_1, V_2\}$ have dimensions $\{m \times m, n \times n, m \times n, m, n\}$ and M_1 and M_2 are non-singular [28]. For simplicity, we set $n = m$. The EA-equivalence problem can then be formulated as follows:

For given functions $F, G : GF(2)^n \mapsto GF(2)^n$ find M_1, M_2, M_3, V_1, V_2 such that (1) holds or show that such vectors and matrices do not exist.

The variables in the system to be solved are the entries in M_1, M_2, M_3, V_1 and V_2, so the number of variables is $3n^2 + 2n$. The maximum number of equations and a system's degree can be calculated theoretically for given F and G [29]. However, for $n \leq 6$ the system can always be made quadratic by introducing the matrix $M_3^{'} = M_1^{-1} \cdot M_3$ and the vector $V_1^{'} = M_1^{-1} \cdot V_1$ and add them as additional equations and variables.

For $n = 4$ and $n = 5$ the problem of EA-equivalence is tractable, and the complexity comparison of Gröbner basis (GB), CryptoMiniSat (SAT) and proposed approach (BDD) is presented in Table 3 for five different instances. Unlike the two other methods, CryptoMiniSat only finds one solution by default. Therefore, we have used the option "n = +infinity" in CryptoMiniSat to force it to find all solutions and get a fair comparison.

In Table 3 "†" means that the Gröbner bases solver implemented in Sage crashes after several minutes with out-of-memory message, and "–" means that CryptoMiniSat spent more than 78 hours (2^{18} seconds) without finding a solution [30].

5. Conclusions

In this paper we have explained an approach to build and solve equation systems using binary decision diagrams and reported on experiments with

this method. The best previous results on algebraic attacks against DES use guessing at least 20 of the key bits. We improved on these results using BDDs, breaking six rounds of DES without guessing any variables.

For MiniAES we have also received results which are better than the previous algebraic attacks described in [7] and [15]. According to our experiments a system representing 10 rounds of MiniAES can be solved in 45 minutes on an ordinary computer using the BDD approach. However, other attacks exploiting the short key have lower complexities. At the same time, the BDD method is shown to be advantageous compared to Gröbner basis and CryptoMiniSat on solving the EA-equivalence problem.

These experiments indicate the BDD approach can compete with other methods in applications which require solving the non-linear equation systems. There are several open questions to address in future research. Does there exist a generic algorithm giving an order of BDDs that yield low complexity when applying linear absorption? Is it possible to analytically estimate the complexity of solving a BDD system of equations, or do we have to actually run the solver to find out? Which ciphers are most vulnerable against this type of algebraic attacks? We hope the potential of BDDs in cryptanalysis will be thoroughly examined in future research.

References

[1] Shannon, C. E.: A mathematical theory of communication. In *Bell system technical journal*, vol. 27, pp. 623–656. University of Illinois Press, 1948.

[2] Raddum, H., Semaev, I.: Solving multiple right hand sides linear equations. In *Designs, Codes and Cryptography*, vol. 49, pp. 147–160. Springer US, 2008.

[3] Courtois, N.: Fast algebraic attacks on stream ciphers with linear feedback. In Boneh, D. (ed.), *Advances in Cryptology - CRYPTO 2003*, vol. 2729 of *Lecture Notes in Computer Science*, pp. 176–194. Springer Berlin Heidelberg, 2003.

[4] Courtois, N., Bard, G., Wagner, D.: Algebraic and slide attacks on KeeLoq. In Nyberg, K. (ed.), *Fast Software Encryption*, vol. 5086 of *Lecture Notes in Computer Science*, pp. 97–115. Springer Berlin Heidelberg, 2008.

[5] Courtois, N., Bard, G.: Algebraic cryptanalysis of the Data Encryption Standard. In Galbraith, S. (ed.), *Cryptography and Coding*, vol. 4887 of *Lecture Notes in Computer Science*, pp. 152–169. Springer Berlin Heidelberg, 2007.

[6] Helleseth, T., Rønjom, S.: Simplifying algebraic attacks with univariate analysis. In *Information Theory and Applications Workshop (ITA)*, pp. 1–7. Institute of Electrical and Electronics Engineers, 2011.

[7] Cid, C., Murphy, S., Robshaw, M.: Small scale variants of the AES. In Gilbert, H., Handschuh, H. (eds.), *Fast Software Encryption*, vol. 3557 of *Lecture Notes in Computer Science*, pp. 145–162. Springer Berlin Heidelberg, 2005.

[8] Bard, G. V.: *Algebraic cryptanalysis*. Springer, 2009.

[9] Albrecht, M.: *Algorithmic algebraic techniques and their application to block cipher cryptanalysis*. Ph.D. thesis, Royal Holloway, University of London, the United Kingdom, 2010.

[10] Daum, M.: *Cryptanalysis of Hash functions of the MD4-family*. Ph.D. thesis, Ruhr University Bochum, Germany, 2005.

[11] Lee, C.-Y.: Representation of switching circuits by binary-decision programs. In *Bell System Technical Journal*, vol. 38, pp. 985–999. 1959.

[12] Knuth, D. E.: *The Art of Computer Programming. Bitwise Tricks & Techniques. Binary Decision Diagrams*, vol. 4. Addison-Wesley, 2009.

[13] Krause, M.: BDD-based cryptanalysis of keystream generators. In Knudsen, L. (ed.), *Advances in Cryptology — EUROCRYPT 2002*, vol. 2332 of *Lecture Notes in Computer Science*, pp. 222–237. Springer Berlin Heidelberg, 2002.

[14] Stegemann, D.: Extended BDD-based cryptanalysis of keystream generators. In Adams, C., Miri, A., Wiener, M. (eds.), *Selected Areas in Cryptography*, vol. 4876 of *Lecture Notes in Computer Science*, pp. 17–35. Springer Berlin Heidelberg, 2007.

[15] Kleiman, E.: *High performance computing techniques for attacking reduced version of AES using XL and XSL methods*. Ph.D. thesis, Iowa State University, USA, 2010.

[16] Bulygin, S., Brickenstein, M.: Obtaining and solving systems of equations in key variables only for the small variants of AES. In *Mathematics in Computer Science*, vol. 3, pp. 185–200. Birkhäuser-Verlag, 2010.

[17] Bryant, R.: Graph-based algorithms for Boolean function manipulation. In *Computers, IEEE Transactions*, vol. C-35, pp. 677–691. Institute of Electrical and Electronics Engineers, 1986.

[18] Schilling, T., Raddum, H.: Solving compressed right hand side equation systems with linear absorption. In Helleseth, T., Jedwab, J. (eds.), *Sequences and Their Applications – SETA 2012*, vol. 7280 of *Lecture Notes in Computer Science*, pp. 291–302. Springer Berlin Heidelberg, 2012.

[19] Rudell, R.: Dynamic variable ordering for ordered binary decision diagrams. In *Computer-Aided Design, 1993. ICCAD-93. Digest of Technical Papers., 1993 IEEE/ACM International Conference on*, pp. 42–47. Institute of Electrical and Electronics Engineers, 1993.

[20] Bollig, B., Wegener, I.: Improving the variable ordering of OBDDs is NP-complete. In *Computers, IEEE Transactions*, vol. 45, pp. 993–1002. Institute of Electrical and Electronics Engineers, 1996.

[21] Perret, J. C. F. L., Spaenlehauer, P. J.: Algebraic differential cryptanalysis of DES. *In Proceedings Western European Workshop on Research in Cryptology - WEWoRC,* pp. 1–5, 2009.

[22] Eén, N., Sörensson, N.: An extensible SAT-solver. In Giunchiglia, E., Tacchella, A. (eds.), *Theory and Applications of Satisfiability Testing*, vol. 2919 of *Lecture Notes in Computer Science*, pp. 502–518. Springer Berlin Heidelberg, 2004.

[23] FIPS 46–3: Data Encryption Standard (DES). *National Institute of Standards and Technology,* 1993.

[24] Phan, R. C.-W.: Mini Advanced Encryption Standard (Mini-AES): A testbed for cryptanalysis students. In *Cryptologia*, vol. 26, pp. 283–306. Taylor & Francis, 2002.

[25] Kleiman, E.: *The XL and XSL attacks on Baby Rijndael*. Master's thesis, Iowa State University, USA, 2005. `https://orion.math.iastate.edu/dept/thesisarchive/MS/EKleimanMSSS05.pdf`.

[26] FIPS PUB 197: Advanced Encryption Standard (AES). *National Institute of Standards and Technology,* 2001.

[27] Soos, M., Nohl, K., Castelluccia, C.: Extending SAT solvers to cryptographic problems. In Kullmann, O. (ed.), *Theory and Applications of Satisfiability Testing - SAT 2009*, vol. 5584 of *Lecture Notes in Computer Science*, pp. 244–257. Springer Berlin Heidelberg, 2009.

[28] Budaghyan, L., Kazymyrov, O.: Verification of restricted EA-equivalence for vectorial Boolean functions. In Özbudak, F., Rodríguez-Henríquez, F. (eds.), *Arithmetic of Finite Fields*, vol. 7369 of *Lecture Notes in Computer Science*, pp. 108–118. Springer Berlin Heidelberg, 2012.

[29] Eilertsen, A. M., Kazymyrov, O., Kazymyrova, V., Storetvedt, M.: A Sage library for analysis of nonlinear binary mappings. *Preproceedings of Central European Conference on Cryptology (CECC14),* pp. 69–78, 2014.

[30] Stein, W., et al.: Sage mathematics software (version 6.2). *The Sage Development Team,* 2014. `http://www.sagemath.org`.

PAPER II

II

A SAGE LIBRARY FOR ANALYSIS OF NONLINEAR BINARY MAPPINGS *

Anna Maria Eilertsen Oleksandr Kazymyrov
Valentyna Kazymyrova Maksim Storetvedt

*Eilertsen, A. M., Kazymyrov, O., Kazymyrova, V., Storetvedt, M.: A Sage library for analysis of nonlinear binary mappings. In *Pre-proceedings of Central European Conference on Cryptology (CECC14)*, pp. 69–78, 2014.

A Sage Library For Analysis Of Nonlinear Binary Mappings

Anna Maria Eilertsen Oleksandr Kazymyrov
Valentyna Kazymyrova Maksim Storetvedt

Department of Informatics
University of Bergen, Norway
Oleksandr.Kazymyrov@ii.uib.no
{Anna.Eilertsen,Valentyna.Kazymyrova,Maksim.Storetvedt}@student.uib.no

Abstract

Many new ciphers are being introduced every year. Each well-educated researcher with a degree in computer science can create a good cryptoprimitive, which will be unbreakable awhile. The main blocks of modern ciphers are nonlinear components also known as substitutions. The toolbox (library) for decreasing time of analyzing such components is given in this paper. It can be used for investigation cryptographic properties of arbitrary nonlinear binary mappings used in symmetric primitives.

Keywords: substitution, Sage, cryptanalysis, cryptographic properties

1. Introduction

Today's information society has the conventional wisdom to enhance the security of information systems. A significant role in this process has the implementation of cryptographic primitives that provide the required level of cryptographic security. Enough time has passed since cryptography has become public science. As a result, in many areas ciphers have started to be used for protection of sensitive information.

Each year several new cryptoprimitives are issued [1, 2]. Many designers justify the resistance of the developed algorithms to known attacks such as differential, linear or algebraic. Meanwhile, an independent analysis requires tools to analyze both basic components and entire encryption algorithms. On the other hand, if you are a developer of a perspective algorithm, there is a need to analyze basic components as well. Choosing linear layers is a relatively

simple task when only few indicators are considered [3]. The situation is completely opposite for nonlinear layers which are usually presented as parallel application of substitutions (S-boxes). Effective and simultaneous calculation of cryptographic properties of S-boxes is a nontrivial task.

Analysis of current state shows today there is a number of tools that can be considered as a partial solution of the problem [4–8]. However, the cryptographic community needs a universal approach to calculate cryptographic indicators for arbitrary binary mappings. In this paper we propose a tool for generating and analyzing arbitrary vectorial Boolean functions $F : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$.

The rest of the paper is organized as follows. Section 2 describes the theoretical background of substitution's criteria. Section 3 introduces general overview of the "Sbox" library and explains the main components and methods. Section 4 includes practical applications of the proposed library. Finally, Section 5 presents our conclusions.

2. PRELIMINARIES

In this section we present theoretical aspects of representation and construction of vectorial Boolean functions. The relevant properties used in symmetric primitives are also considered in this section. All definitions and indicators are well-known and one can see [9] for more details.

2.1. DEFINITIONS AND NOTATIONS

Let n and m be two positive integers. Define $\mathbb{F}_2^n$ as a vector space of all binary vectors of length n, where $\mathbb{F}_2$ is the Galois field with elements $\{0, 1\}$. Then (n, m)-function is a vectorial Boolean function $F : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$. Boolean functions $f_1, f_2, \ldots, f_m$, such that $F(x_1, \ldots, x_n) = (f_1(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n))$, and their linear combination are called coordinate and component functions of F, respectively. If $m = 1$ then a vectorial Boolean function has a single output bit and is equivalent to a Boolean function. To find algebraic properties of (n, m)-functions, a vector space often has a structure of finite field $\mathbb{F}_{2^n}$.

2.2. CRYPTOGRAPHIC PROPERTIES OF BOOLEAN FUNCTIONS

Suppose $f : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ is a Boolean function of n variables. Algebraic normal form of such function is defined as

$$f(x_1, x_2, \ldots, x_n) = \sum_{I \in \mathbb{P}(1,\ldots,n)} \left(\prod_{i \in I} x_i \right),$$

where $\mathbb{P}(z)$ denotes the power set of z. The algebraic degree of f ($deg(f)$) is the maximum degree of the monomial with nonzero coefficient.

A Boolean function of n variables is called balanced if $hw(f) = 2^{n-1}$, where $hw(f) = \sum_{x=0}^{2^n-1} f(x)$. The correlation between arbitrary Boolean function $f(x)$, where $x = (x_1, x_2, \ldots, x_n)$, and the set of all linear functions is determined by Walsh transformation

$$W(w) = \sum_{x=0}^{2^n-1} (-1)^{f(x) \oplus l_w(x)},$$

where $l_w(x) = w \cdot x = w_1 x_1 \oplus w_2 x_2 \oplus \ldots w_n x_n$. The nonlinearity is related to the Walsh values as

$$NL(f) = \frac{1}{2} \left(2^n - \max_{\forall w, w \neq 0} |W(w)| \right).$$

Autocorrelation of f ($r_f(\alpha)$) shows how the function differs from itself shifted on several positions, i.e.

$$r_f(\alpha) = \sum_{x=0}^{2^n-1} (-1)^{f(x) \oplus f(x \oplus \alpha)},$$

where $\alpha \in \mathbb{F}_2^n$. For cryptography the maximal value of the function $r_f(\alpha)$ is of interest, which can be found as

$$AC_{max}(f) = \max_{\forall \alpha, \alpha \neq 0} \left| r_f(\alpha) \right|.$$

Let σ be the sum-of-square indicator, then

$$\sigma = \sum_{\alpha=0}^{2^n-1} r_f^2(\alpha).$$

Let $hw(\alpha)$ be a binary Hamming weight of $\alpha \in \mathbb{F}_2^n$ [10]. Then we say that $f(x)$ satisfies propagation criterion of order k ($PC(k)$) if and only if for all nonzero vectors $\alpha \in \mathbb{F}_2^n$ the following system is true

$$\begin{cases} 1 \le hw(\alpha) \le k; \\ \sum\limits_{x=0}^{2^n-1} f(x) \oplus f(x \oplus \alpha) = 2^{n-1}. \end{cases}$$

The strict avalanche criterion (SAC) corresponds to $PC(1)$.

A Boolean function is correlation immune of order m ($CI(m)$) if the system of equations

$$\begin{cases} 1 \le hw(w) \le m; \\ W(w) = 0. \end{cases}$$

is valid for all $w \in \mathbb{F}_2^n$. If the function is balanced and satisfy $CI(m)$ simultaneously, then such function is called m-resilient.

To prevent some misunderstanding we present the indicator called algebraic immunity of a Boolean function. The minimum algebraic degree of $g(x) \neq 0$ of the set $\{g \mid f(x) \cdot g(x) = 0\} \cup \{g \mid (f(x) \oplus 1) \cdot g(x) = 0\}$ is called algebraic immunity (AI) of f.

2.3. Cryptographic properties of substitutions

While Boolean functions are adopted mainly as filtering functions in stream ciphers, vectorial Boolean function are used in block ciphers and hash functions as substitutions. For theoretical analysis the univariate representation is one of the best ways to consider cryptographic properties of the binary mappings. However, field operations are not good optimized in modern computers as operations with Boolean functions, especially for large n. Therefore, the representation of (n, m)-functions as the set of component functions is a better way for practical implementations.

Suppose substitution S is a table representation of a vectorial Boolean function $F = (f_1, \ldots, f_m)$ from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$. Define $\{h_j = j \cdot F \mid 0 < j < 2^m\}$ as the set of component functions of F. Then

- nonlinearity of S is

$$NL(S) = \min_{0<j<2^m} \left(NL(h_j)\right);$$

- minimum degree of S is

$$deg(S) = \min_{0<j<2^m} \left(deg(h_j)\right);$$

- the maximum value of autocorrelation spectrum of S is

$$AC_{max}(S) = \max_{0<j<2^m} \left(AC_{max}(h_j)\right);$$

- S satisfies strict avalanche criterion if every h_j satisfies SAC;
- S satisfies propagation criterion of order k if every h_j satisfies $PC(k)$;
- S is correlation immune of order k if every h_j is $CI(k)$;
- S is balanced (permutation) if every h_j is balanced;
- S is k-resilient if every h_j is k-resilient.

The similar properties for vectorial Boolean functions are given in [9].

While the maximum value of the approximation table (λ) can be calculated directly from the nonlinearity of the S-box as $\lambda = 2^{n-1} - NL(S)$, the maximum value of differential table (MDT) cannot be easily evaluated from the component functions. Let δ be the maximum number of times when the input difference maps to the output difference of the given S-box. Then

$$\delta = \max_{\alpha \in \mathbb{F}_2^n, \alpha \neq 0, \beta \in \mathbb{F}_2^m} \#\{x \mid S(x) \oplus S(x \oplus \alpha) = \beta\}.$$

This indicator is also known as δ-uniformity [9].

The ways to represent a substitution as a system of equations over $\mathbb{F}_2$ are given in [11, 12]. Define density as the fraction of nonzero elements in a system of equations. Then, a substitution provides better protection against algebraic attacks if the system

- has higher degree;
- has fewer equations;
- is more dense.

Unambiguous theoretical relation between these parameters is an unsolved problem [12]. When we are talking about the algebraic immunity of an S-box ($AI(S)$), then we mean the smallest degree of the system.

2.4. Equivalence of vectorial Boolean functions

Two functions $F, G : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ are called extended affine (EA) equivalent if there exist such affine permutations $A_1 = L_1(x) + c_1, A_2 = L_2(x) + c_2$ and arbitrary linear function $L_3(x)$ that

$$F(x) = A_1 \circ G \circ A_2(x) + L_3(x).$$

If $L_3(x) = const$, or $L_3(x) = 0, c_1 = 0$, and $c_2 = 0$ then F and G are affine, or linear equivalent, respectively. Moreover, for at least one missing element of $L_1(x), L_2(x), L_3(x), c_1, c_2$ the functions are called restricted EA (REA) equivalent [13].

In [14] F and G are considered as $G_F(x, y) = \{\{x, y\} \mid y = F(x)\}$. They are Carlet-Charpin-Zinoviev (CCZ) equivalent, if for $F_2(x) = L_3(x) + L_4 \circ G(x)$ and permutation $F_1(x) = L_1(x) + L_2 \circ G(x)$ the following equation is hold

$$F(x) = F_2 \circ F_1^{-1}(x),$$

where $L_1(x)$, $L_2(x)$, $L_3(x)$, $L_4(x)$ are arbitrary linear functions.

Both CCZ- and EA-equivalence preserve extended Walsh spectrum and δ-uniformity. However, the minimum degree remains the same only for the EA-equivalent functions. [9].

3. Design specification and main components

In this section we describe basic components of the "Sbox" library, and some methods for both generation and cryptanalysis. The source code is available at github and distributed under GPL v2 [15].

3.1. General overview of the library

There are many ready-made solutions [4, 5, 8]. However, all of them have certain limitations. For example, the class SBox from the package mq in Sage is optimized only for small values [4]. By increasing n, functions do not return the expected results, i.e. the absence of the system of equations of degree 2 for the AES substitution [16]. Most of the other programs or libraries are designed to work with a limited number of properties and/or just with Boolean functions. As a consequence, software to analyze arbitrary vectorial Boolean functions was developed taking into account publicly available optimized algorithms.

The proposed implementation is written as an extension to Sage and presented as a package with the main class "Sbox" [15]. Fig. 1 depicts the general overview of the library. It consists of three main parts: Sbox, CSbox and GSbox. While most cryptanalytic functions are presented in CSbox and connected to C/C++ code via Cython, GSbox includes methods for generation nonlinear binary mappings. Using Python as the main language is beneficial considering that the language is somewhat easy to learn and use. It does in general a slower run time, but this disadvantage may be offset by Cython and C/C++, making extensive mathematical calculations faster.

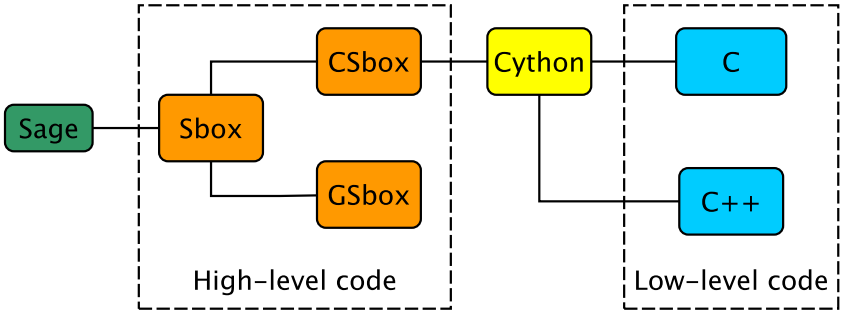


Fig. 1: *The general overview of "Sbox"*

It should also be noted that some methods use integrated in Sage packages (i.e. finite_rings) and cannot be transfered into other software. However, the C/C++ code is an independent implementation and one can easily port it to own software.

3.1.1. Performance comparison with other tools.

The performance comparison of software, even written in one language, is a difficult task. It depends on many parameters including operation system, compilations, memory and processors. We have not created best of the best tool, but we took into account a number of articles with optimized algorithms for calculation of cryptographic indicators [17, 18]. Hence, we will present time comparisons as the most critical indicator of the proposed implementation in Section 4.

Table 1: *Correspondence of vectorial Boolean functions to methods defined in "Sbox.sage"*

Vectorial Boolean functions	Correspondent methods
Gold	gold
Kasami	kasami
Welch	welch
Niho	niho
Inverse	inverse
Dobbertin	dobbertin
Dicson	dicson
APN for $n = 6$	APN6
Optimal permutation polynomials for $n = 4$	OP4

3.2. Generation of vectorial Boolean functions

All functions that belong to this group start with "gen_" and defined in "GSbox.sage". Most of them are based on theoretical methods described in [9, 19]. The correspondence between methods implemented in the library and well-known names are presented in Table 1.

The library has also methods to find CCZ-(CCZ) and EA-equivalence (EA). They do not allow to find vectorial Boolean functions with certain characteristics by themselves, however, they play very important roles in cryptography. For completeness and additional experiments the "Sbox" class includes methods for generating random substitutions and permutations that are specified by "random_substitution" and "random_permutation", respectively.

In order to unify different cases a single method "generate_sbox" was created, which has parameters "method" to specify generation method and "T" to define equivalence. The optional parameter "G" is used for setting a user-defined polynomial in the case of "method=polynomial".

3.3. Cryptanalysis of vectorial Boolean functions

This group contains methods starting with "cr_", which are described in "CSbox.sage". Most of the methods are also based on theoretical algorithms [9]. Nevertheless, the calculation of some indicators has been optimized, i.e. calculation of the algebraic immunity or cycles. Table 2 shows the correspon-

Table 2: *Correspondence between cryptographic properties and methods in the library*

Indicators	Methods in "Sbox"
Balanceness	balanced
Nonlinearity	nonlinearity
Absolute indicator	autocorrelation
Propagation criterion	PC
Correlation immunity	CI
Sum-of-square indicator	SSI
Minimum degree	minimum_degree
Resilience	resilient
Strict avalanche criterion	SAC
Bijection	is_bijection
Maximum of differential table	MDT
Maximum of linear approximation table	MLT
Cycles	cycles
Algebraic immunity	algebraic_immunity_sbox

dence between the indicators described in Section 2, and the methods from the "Sbox" library.

To investigate the number of cryptographic properties in some applications such as generation of pseudo-random sequences (i.e. using OFB mode), it is necessary to study the period lengths of sequences obtained at the output of the block cipher. For these purposes, the "cycles" method was implemented.

This group also contains a number of auxiliary functions such as finding the univariate polynomial or the system of equations describing the substitution; checking APN, or CCZ-equivalence properties [20]. Some examples of the use of different methods are given in Section 4.

3.4. Other useful methods

The last group of methods is optional and is used as an extension of functionality. Table 3 contains the most important methods and their short description. As it can be seen from the table, the methods' names define their functional purpose.

Table 3: *Optional methods of "Sbox'*

Methods	Comments
get_field	return the filed $\mathbb{F}_{2^{\max\{n,m\}}}$
get_ring	return the ring $\mathbb{F}_{2^{\max\{n,m\}}}[x]$
get_mg	return a multiplicative generator of the field
get_sbox	return the substitution
get_polynomial	return the univariate polynomial of the substitution
set_sbox	set a substitution for analysis
Tr_pol	return a trace of input polynomial
g2p	transform a given polynomial with multiplicative element to the internal representation
p2g	inverse method to g2p

4. Application of the "Sbox" library

To give an example of the library usage, we present the main steps of generation of the APN permutation for $n = 6$. First let us give a mathematical definition of such function [20].

Theorem 1. *Let α be a multiplicative generator of $\mathbb{F}_{2^6}$ with irreducible polynomial $f(x) = x^6 + x^4 + x^3 + x + 1$. Then the APN function*

$$F(x) = \alpha x^3 + \alpha^5 x^{10} + \alpha^4 x^{24}$$

is CCZ-equivalent to an APN permutation over $\mathbb{F}_{2^6}$.

For example, for the linear function

$$\mathcal{L}(x, y) = (tr_{6/3}(\alpha^4 x) + \alpha tr_{6/3}(y), tr_{6/3}(\alpha x) + \alpha tr_{6/3}(\alpha^4 y)),$$

where $tr_{6/3} = x + x^{2^3}$, $y = F(x)$, the function $G_H = \mathcal{L}(G_F)$ is the APN permutation.

Using the above description one can easily start repeating the following example in Sage. First we need to identify the main variables, including the ring $\mathbb{F}_{2^6}[x]$ (P), a multiplicative generator (g) of $\mathbb{F}_{2^6}$, and the trace (tr).

```
sage: %runfile ./Sbox.sage
sage: S = Sbox(n=6,m=6)
sage: P = S.get_ring()
sage: g = S.get_mg()
a
sage: tr = S.Tr_pol(x=P("x"),n=6,m=3)
sage: tr
x^8 + x
```

Next it is necessary to specify linear functions and convert them from polynomial representations to matrix forms [13].

```
sage: M1 = S.l2m(tr.subs(P("(%s)*x"%(g^4))))
sage: M2 = S.l2m(g*tr)
sage: M3 = S.l2m(tr.subs(P("(%s)*x"%(g))))
sage: M4 = S.l2m(g*tr.subs(P("(%s)*x"%(g^4))))
```

In the end, we apply CCZ-equivalence to the function F and check on APN properties.

```
sage: F = "g*x^3+g^5*x^10+g^4*x^24"
sage: S.generate_sbox(method="polynomial",G=F,T="CCZ",M1=M1,M2=M2,
    ↪ M3=M3,M4=M4)
sage: S.is_bijection()
True
sage: S.is_APN()
True
sage: S.MDT()
2
```

The same result can be achieved by the following commands.

```
sage: S = Sbox(n=6,m=6)
sage: S.generate_sbox(method='APN6')
sage: S.is_bijection()
True
sage: S.is_APN()
True
```

The above example shows, that hundred of strings of other libraries can be replaced by a few lines of the proposed library to achieve the same functionality. On the other hand, the performance needs to be at a high level. Fig. 2 shows the time complexity of several frequently used methods for $n = m$.

All results are presented as the average values of 1000 runs on Macbook Pro Retina Mid 2012 [21]. One can easily notice that the figure doesn't show the results for every n. This is due to the fact that every function has limitations either in memory or in time. For example, the time needed for counting the value of algebraic immunity is negligible, while the memory is growing very

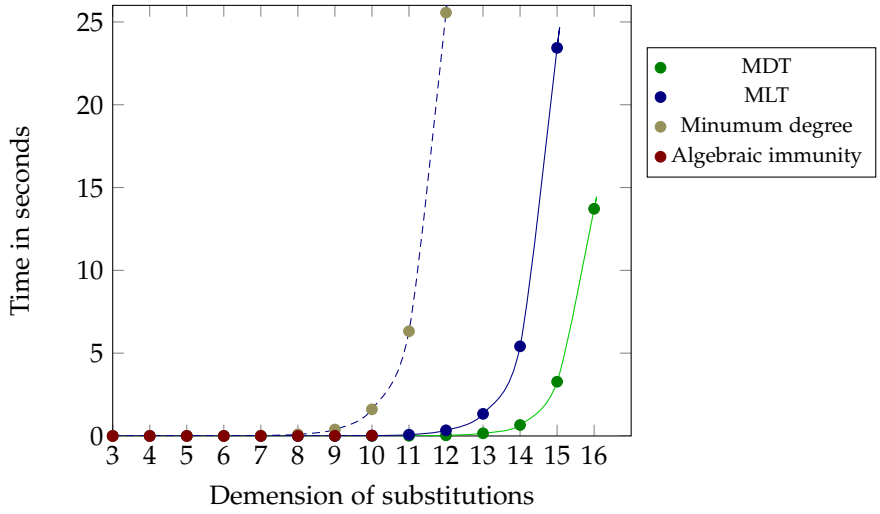


Fig. 2: *The relationship between the dimension of random substitutions and time of calculation*

fast. Consequently, every function has the maximum values of n or m after which the indicators become difficult to calculate on an ordinary computer.

Other examples for different values of n and m is given are Table 4. One can see that the library can work correctly with any dimension, but in the same time has some limitations.

Table 4: *Performance of different methods from the "Sbox" library given in ms*

Properties	(n,m)-functions								
	(6,10)	(8,1)	(8,4)	(10,2)	(10,8)	(3,12)	(12,3)	(12,6)	(12,9)
MDT	0.083	0.17	0.17	2.34	2.34	0.037	37.15	37.15	38.02
MLT	1.23	0.068	0.126	0.114	4.9	0.501	0.708	5.25	41.69
Algebraic immunity	1	8.91	2.24	6.17	22.4	-	39.8	75.9	-
Minimum degree	89.64	0.35	4.77	3.92	329.3	48.67	35.78	320.89	2631
Balancedness	0.102	0.331	0.338	1.26	1.32	0.025	5.13	5.13	5.01
$AC_{max}(S)$	5.89	109.6	110.6	2951.2	2818.4	0.1	112202	117490	114815
Interpolation polynomial	112.2	457.1	457.1	7586	14454	11.74	281838	288403	275422
Cycles	0.263	1.78	2.69	8.13	25.7	0.074	41.69	72.45	158.49
PC	6.03	89.1	89.1	1446	1446	0.098	23442	23442	24547

From a practical point of view the "Sbox" library can be used to analyze nonlinear components of the existing cryptographic primitives. The substitu-

Table 5: *Comparison of 8-bit S-boxes*

Properties	AES	GOST R 34.11-2012	STB 34.101.31-2011	Kalyna's S0	Proposed in [23]
MDT	4	8	8	8	8
MLT	16	28	26	32	24
Absolute indicator	32	96	80	88	80
SSI	133120	258688	232960	244480	194944
Minimum degree	7	7	6	7	7
Algebraic immunity	2	3	3	3	3

tion comparison of AES, GOST R 34.11-2012, STB 34.101.31-2011, "Kalyna" ($S0$) [22], and a substitution proposed in [23] is given in Table 5.

5. Conclusions

It has been indicated quite clear that practical realization of theoretically proved results is time and resource consuming in most cases. Conducted analysis has shown that no sufficiently effective tools of finding characteristics of arbitrary substitutions exist to date.

The "Sbox" library implementation allows to solve many problems related to cryptanalysis. It includes lots of known generation methods and functions for computing permutations' properties. Moreover, the library is designed in the way that gives an opportunity to extent its functionality quite easily.

References

[1] Canetti, R., Garay, J. A. (eds.): *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, vol. 8042 of *Lecture Notes in Computer Science*. Springer, 2013.

[2] International Association for Cryptologic Research: *Fast Software Encryption 2014*, March 2014. `http://fse2014.isg.rhul.ac.uk/`.

[3] Augot, D., Finiasz, M.: Direct construction of recursive MDS diffusion layers using shortened BCH codes. *In Pre-proceedings of Fast Software Encryption (FSE 2014),* 2014.

[4] Stein, W., et al.: Sage mathematics software (version 6.2). *The Sage Development Team,* 2014. `http://www.sagemath.org`.

[5] Biryukov, A., De Cannière, C., Braeken, A., Preneel, B.: A toolbox for cryptanalysis: Linear and affine equivalence algorithms. In Biham, E. (ed.), *Advances in Cryptology — EUROCRYPT 2003*, vol. 2656 of *Lecture Notes in Computer Science*, pp. 33–50. Springer Berlin Heidelberg, 2003.

[6] Burnett, L.: *Heuristic optimization of Boolean functions and substitution boxes for cryptography*. Ph.D. thesis, Queensland University of Technology, Australia, 2005.

[7] Albrecht, M.: *Algorithmic algebraic techniques and their application to block cipher cryptanalysis*. Ph.D. thesis, Royal Holloway, University of London, the United Kingdom, 2010.

[8] Lafitte, F., Heule, D. V., Hamme, J. V.: Cryptographic Boolean functions with R. In *The R Journal*, vol. 3, pp. 44–47. June, 2011.

[9] Carlet, C.: *Vectorial Boolean functions for cryptography*. Boolean Models and Methods in Mathematics, Computer Science, and Engineering. Cambridge University Press, 2010.

[10] Carlet, C.: *Boolean functions for cryptography and error correcting codes*. Boolean Models and Methods in Mathematics, Computer Science, and Engineering. Cambridge University Press, 2010.

[11] Kleiman, E.: *High performance computing techniques for attacking reduced version of AES using XL and XSL methods*. Ph.D. thesis, Iowa State University, USA, 2010.

[12] Kazymyrov, O., Oliynykov, R.: Choosing substitutions for symmetric cryptographic algorithms based on the analysis of teir algebraic properties. In *Mathematical modeling. Information Technology. Automated control systems.*, vol. 925, pp. 79–86. V. N. Karazin Kharkov National University, Ukraine, 2010. (In Russian).

[13] Budaghyan, L., Kazymyrov, O.: Verification of restricted EA-equivalence for vectorial Boolean functions. In Özbudak, F., Rodríguez-Henríquez, F. (eds.), *Arithmetic of Finite Fields*, vol. 7369 of *Lecture Notes in Computer Science*, pp. 108–118. Springer Berlin Heidelberg, 2012.

[14] Carlet, C., Charpin, P., Zinoviev, V.: Codes, bent functions and permutations suitable for DES-like cryptosystems. In *Designs, Codes and Cryptography*, vol. 15, pp. 125–156. Kluwer Academic Publishers, 1998.

[15] Kazymyrov, O., et al.: Source code of the Sbox library. *GitHub repository,* 2014. `https://github.com/okazymyrov/sbox`.

[16] Courtois, N., Pieprzyk, J.: Cryptanalysis of block ciphers with overdefined systems of equations. *Cryptology ePrint Archive, Report 2002/044*, 2002. `http://eprint.iacr.org/`.

[17] Mishra, P. R.: Calculating cryptographic degree of an S-box. *Cryptology ePrint Archive, Report 2014/145*, 2014. `http://eprint.iacr.org/`.

[18] Chmiel, K.: Fast computation of approximation tables. In Saeed, K., Pejaś, J. (eds.), *Information Processing and Security Systems*, pp. 125–134. Springer US, 2005.

[19] Kazymyrov, O., Oliynykov, R.: Application of vectorial Boolean functions for substitutions generation used in symmetric cryptographic transformations. In *Information Processing Systems*, vol. 6 (104), pp. 97–102. V. N. Karazin Kharkov National University, Ukraine, 2012 (In Russian).

[20] Browning, K., Dillon, J., McQuistan, M., Wolfe, A.: An APN permutation in dimension six. In McGuire, G., et al. (eds.), *Finite Fields: Theory and Applications*, vol. 518 of *Contemporary Mathematics*, pp. 33–42. American Mathematical Society, 2010.

[21] Apple: 15-inch MacBook Pro with Retina display. *2.3GHz, 8GB of 1600MHz DDR3L, Mid 2012. Technical Specifications,* February 2013. `http://support.apple.com/kb/sp653`.

[22] Oliynykov, R., Gorbenko, I., Dolgov, V., Ruzhentsev, V.: Results of Ukrainian national public cryptographic competition. In *Tatra Mountains Mathematical Publications*, vol. 47, pp. 99–113. Mathematical Institute of Slovak Academy of Sciences, 2010.

[23] Kazymyrov, O., Kazymyrova, V., Oliynykov, R.: A method for generation of high-nonlinear S-boxes based on gradient descent. In *Mathematical Aspects of Cryptography*, vol. 5, pp. 71–78. Steklov Mathematical Institute, 2014.

PAPER III

A METHOD FOR GENERATION OF HIGH-NONLINEAR S-BOXES BASED ON GRADIENT DESCENT *

Oleksandr Kazymyrov Valentyna Kazymyrova
Roman Oliynykov

III

*Kazymyrov, O., Kazymyrova, V., Oliynykov, R.: A method for generation of high-nonlinear S-boxes based on gradient descent. In *Mathematical Aspects of Cryptography*, vol. 5, pp. 71–78. Steklov Mathematical Institute, 2014.

A Method For Generation Of High-Nonlinear S-Boxes Based On Gradient Descent

Oleksandr Kazymyrov[†] Valentyna Kazymyrova[†]
Roman Oliynykov[‡]

[†] University of Bergen, Norway
Oleksandr.Kazymyrov@ii.uib.no, Valentyna.Kazymyrova@student.uib.no
[‡] Kharkov National University of Radioelectronics, Ukraine
ROliynykov@gmail.com

Abstract

Criteria based on the analysis of the properties of vectorial Boolean functions for selection of substitutions (S-boxes) for symmetric cryptographic primitives are given. We propose an improved gradient descent method for increasing performance of nonlinear vectorial Boolean functions generation with optimal cryptographic properties. Substitutions are generated by proposed method for the most common 8-bits input and output messages have nonlinearity 104, 8-uniformity and algebraic immunity 3.

Keywords: substitution, nonlinearity, symmetric ciphers, vectorial Boolean function.

1. Introduction

S-boxes are one of the main components that determine the robustness of modern symmetric cryptographic primitives. They typically perform the mapping of n-bit input block to the output of m-bits length. Representation of S-boxes varies depending on kind of problem they are used for. In stream ciphers substitutions are usually presented in the form of vectorial Boolean functions [1]. Permutations are a subclass of substitutions and are widely used in block ciphers in a table form. It is quite easy to transform a substitution from one form to another.

To protect cryptographic primitives against various types of attacks a substitution must satisfy a number of criteria [2, 3]. Taking into account the large number of existing characteristics, their controversy and partial interdependence, it is likely impossible to generate a substitution that satisfies all known requirements. This became a reason to use a substitution satisfying only mandatory criteria essential for a particular symmetric algorithm. Such substitutions are called optimal [4]. Optimality criteria may vary depending on which cipher is considered. Generating of permutations with optimal criteria is a time- and resource-consuming task, especially for large n and m.

This problem is partially solved by involving the classes of vectorial Boolean functions, extended affine (EA) and Carlet-Charpin-Zinoviev (CCZ) equivalencies [1, 5]. However, the majority of existent functions have extreme characteristics of δ-uniformity and nonlinearity, but at the same time do not possess other properties (i.e., high algebraic immunity) necessary for symmetric cryptographic primitives. It was shown in 2010 that such substitutions exist [6]. Tesar proposed an algorithm based on combination of genetic algorithm and total tree searching. In this paper we give more simple and efficient way of generation substitutions.

In [7] the authors have enhanced the method for generating secure Boolean functions based on gradient ascension (Hill Climbing) method [8]. In this paper we propose a modified version of the gradient descent method for vectorial case, i.e. for functions from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$.

2. PRELIMINARIES

Arbitrary substitution can be represented in at least three different forms: algebraic normal form (ANF), over the field $\mathbb{F}_{2^n}$ and a lookup table. The majority of block ciphers S-boxes have a lookup table form because of their simple description and understanding. At the same time, an arbitrary permutation can always be associated with a vectorial Boolean function F in $\mathbb{F}_{2^n}[x]$. If the substitution is a permutation, then the function F is unique.

A natural way of representing $F : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ is algebraic normal form

$$\sum_{I \subseteq \{1,\ldots,n\}} a_I \left(\prod_{i \in I} x_i \right), \qquad a_I \in \mathbb{F}_2^m,$$

sum is calculated in $\mathbb{F}_2^m$ [1]. Algebraic degree of F is the degree of its ANF. F is called affine if it has the algebraic degree at most 1. When $F(0) = 0$ affine vectorial Boolean function is linear.

Two functions $F, G : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ are called EA-equivalent if there are such affine permutation functions $A_1(x) = L_1(x) + c_1$, $A_2(x) = L_2(x) + c_2$ and arbitrary linear function $L_3(x)$ that [1, 5]

$$F(x) = A_1 \circ G \circ A_2(x) + L_3(x).$$

If $L_3(x)$ is a constant from the vector space F_2^m, then the functions F and G are called affine-equivalent, and if $L_3(x) = 0, c_1 = 0, c_2 = 0$ they are linear equivalent. Affine equivalence was used to prevent the appearance of fixed points during generation of substitutions for cipher Rijndael [9].

Arbitrary vectorial Boolean function F is δ-uniform if for any $a \in \mathbb{F}_2^n \setminus \{0\}$ and $b \in \mathbb{F}_2^m$ the equation $F(x) + F(x + a) = b$ has no more than δ solutions [1]. Vectorial Boolean functions that are used as substitutions in block ciphers must have a small value of δ-uniformity for a sufficient level of protection against differential attacks [1, 3].

Nonlinearity criterion is closely related to the Walsh transformation, which can be described by the function

$$\lambda(u, v) = \sum_{x \in \mathbb{F}_2^n} (-1)^{v \cdot F(x) + u \cdot x},$$

where the symbol "$\cdot$" denotes the scalar product in vector spaces $\mathbb{F}_2^n$ and $\mathbb{F}_2^m$. Substitutions with small values of Walsh coefficients are optimally protected against linear cryptanalysis [1, 3]. S-boxes with minimal values of $\lambda(u, v)$ exist only for odd n.

These two criteria are most significant when selecting substitutions for new ciphers. However, there are many other criteria such as: propagation criterion, maximum value of autocorrelation spectrum, correlation immunity, algebraic immunity, strict avalanche criterion, etc. [1, 2, 10]. Necessity for most of these criteria has not yet been proven. For example, the substitution used in AES does not satisfy most of them [2].

In this paper the optimal substitution refers to a permutation with

- maximum algebraic degree;
- maximum algebraic immunity with the minimum number of equations;

- maximum values of δ-uniformity and nonlinearity limited by parameters listed above;
- absence of fixed points (cycles of length 1).

For example, for $n = 8$ an optimal permutation has algebraic degree 7, algebraic immunity 3 and 441 equations, δ-uniformity under 8, nonlinearity over 100 and without fixed points.

3. Generation of Substitutions With Chosen Parameters

In [7] the main idea is to decrease the nonlinearity of given Boolean bent sequences. In other words, in given bent sequence (truth table) some bits are changed so that the new sequence is balanced and the nonlinearity is close to nonlinearity of bent function. In this paper is proposed to use the same approach, but with two significant differences

- use vectorial Boolean functions instead of Boolean functions;
- use vectorial Boolean functions (substitutions) with minimum δ-uniformity instead of bent-functions (sequences).

Additionally, in [7] was shown that even a small change of fixed number of bits in the bent-sequence does not guarantee the achievement of nonlinearity closed to the maximum.

However, for the vectorial case the following proposition was proven [11].

Proposition 1. *Let $F : \mathbb{F}_{2^n} \mapsto \mathbb{F}_{2^n}$. Function G is determined so that*

$$\begin{cases} G(p_1) = F(p_2) & p_1 \neq p_2; \\ G(p_2) = F(p_1); & \\ G(x) = F(x) & x \notin \{p_1, p_2\}. \end{cases}$$

Then

$$\delta(F) - 4 \leq \delta(G) \leq \delta(F) + 4,$$
$$Nl(F) - 2 \leq Nl(G) \leq Nl(F) + 2.$$

The nonlinearity function (Nl) of arbitrary vectorial function F is calculated as follows

$$Nl(F) = 2^{n-1} - \frac{1}{2} \cdot max_{u \neq 0, v \in \mathbb{F}_{2^n}} |\lambda(u, v)| .$$

We propose a new method to generate substitutions based on Proposition 1. The algorithm takes as input a bijective vectorial Boolean function $F : \mathbb{F}_2^n \mapsto \mathbb{F}_2^n$ with a minimum value of δ-uniformity, and number of values (NP) in function, which have to be changed during the optimization of cryptographic parameters.

The main steps of the algorithm are presented bellow.

1. Generate a substitution S based on F.
2. Swap NP values of S randomly and generate substitution S_t.
3. Test the S-box S_t for all criteria depending on their computational complexity. If S_t satisfies all of them except the cyclic properties, then go to step 3. Otherwise repeat step 2.
4. Apply equivalence (e.g. affine) to S_t in order to achieve the required properties of cycle structure.
5. Output of the algorithm. Required substitution will be stored in S_t.

Theoretical obtaining of swap iterations' number for arbitrary n-bit vectorial function F becomes an additional topic for the research.

4. Practical Results

Before the algorithm was designed practical opportunity of finding optimal substitutions for $n = 8$ had been tested. The challenge was to find several CCZ-nonequivalent substitutions with nonlinearity equal or greater than 100. For practical realization a cluster with 4096 processors was used [12].

The program generated a random permutation and checked it for optimality. After 12 hours of cluster operation it was found 27 optimal permutations, four of which were CCZ-nonequivalent. An example of the permutation in hexadecimal notation is given in Table 1.

Table 1: *An Example of Substitution with Nonlinearity 100*

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	14	9D	B9	E7	67	4C	50	82	CA	E5	1D	31	0A	C6	B2	51
1	A2	D8	54	90	D0	CE	2D	7D	C7	7E	D7	94	DF	83	8E	6C
2	66	D2	6F	16	1E	76	FE	CC	AA	5A	8F	17	BD	2C	AC	EA
3	7B	65	A9	10	C0	92	EE	BE	6A	6E	48	96	95	E9	32	BC
4	A1	42	D5	A7	81	B4	5F	E6	C2	5D	AD	3A	B7	0C	8D	01
5	98	FD	12	02	75	13	0F	6B	22	E2	AB	F7	7F	BA	97	D1
6	64	D9	C4	59	AF	23	33	37	DE	AE	60	05	63	A8	52	A5
7	4E	E0	DD	71	F2	24	34	57	47	A4	B3	9E	2F	C1	B8	CB
8	2B	D4	0D	36	91	8B	9C	26	25	61	A3	D6	EB	35	53	F4
9	2E	88	80	E4	30	DB	FC	0E	77	8C	93	A6	78	06	E1	EC
A	F9	03	A0	27	DA	EF	5C	00	7A	45	E8	40	1A	4B	5E	73
B	C3	FF	F5	F3	B0	C5	49	21	FA	11	39	84	43	38	85	07
C	F0	79	46	F8	E3	1F	09	B6	CD	55	1C	1B	FB	7C	ED	6D
D	15	56	86	20	68	4A	41	4F	D3	99	08	F6	3F	89	62	04
E	CF	C8	69	9F	19	5B	44	9B	87	B1	3D	BB	DC	2A	BF	58
F	3C	8A	18	3E	72	0B	28	4D	B5	9A	C9	74	29	F1	3B	70

This substitution has the following characteristics

- nonlinearity 100;
- absolute value of the autocorrelation 96;
- minimum algebraic degree 7;
- 8-uniform;
- algebraic immunity: a system of 441 equations of the 3rd degree.

Furthermore, search for substitution with nonlinearity 102 was conducted. However, after 48 hours of cluster operation, which is approximately equal to 22 years of a single-processor computer operation, no substitutions were found. Thus, we can conclude that from a practical point of view, the generation of such permutations is computationally extremely difficult.

However, the algorithm described above allows to find such substitutions. For example, consider the function $F(x) = x^{-1}$. The value of NP equals 26 was experimentally found for $n = 8$ with providing the necessary properties of the substitution. An example of such permutation is presented in Table 2.

Table 2: *An Example of Substitution with Nonlinearity 104*

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	68	8D	CA	4D	73	4B	4E	2A	D4	52	26	B3	54	1E	19	1F
1	22	03	46	3D	2D	4A	53	83	13	8A	B7	D5	25	79	F5	BD
2	58	2F	0D	02	ED	51	9E	11	F2	3E	55	5E	D1	16	3C	66
3	70	5D	F3	45	40	CC	E8	94	56	08	CE	1A	3A	D2	E1	DF
4	B5	38	6E	0E	E5	F4	F9	86	E9	4F	D6	85	23	CF	32	99
5	31	14	AE	EE	C8	48	D3	30	A1	92	41	B1	18	C4	2C	71
6	72	44	15	FD	37	BE	5F	AA	9B	88	D8	AB	89	9C	FA	60
7	EA	BC	62	0C	24	A6	A8	EC	67	20	DB	7C	28	DD	AC	5B
8	34	7E	10	F1	7B	8F	63	A0	05	9A	43	77	21	BF	27	09
9	C3	9F	B6	D7	29	C2	EB	C0	A4	8B	8C	1D	FB	FF	C1	B2
A	97	2E	F8	65	F6	75	07	04	49	33	E4	D9	B9	D0	42	C7
B	6C	90	00	8E	6F	50	01	C5	DA	47	3F	CD	69	A2	E2	7A
C	A7	C6	93	0F	0A	06	E6	2B	96	A3	1C	AF	6A	12	84	39
D	E7	B0	82	F7	FE	9D	87	5C	81	35	DE	B4	A5	FC	80	EF
E	CB	BB	6B	76	BA	5A	7D	78	0B	95	E3	AD	74	98	3B	36
F	64	6D	DC	F0	59	A9	4C	17	7F	91	B8	C9	57	1B	E0	61

It has the following properties

- nonlinearity 104;
- absolute value of the autocorrelation 80;
- minimum algebraic degree 7;
- 8-uniform;
- algebraic immunity: a system of 441 equations of the 3rd degree.

During 1 hour of cluster operations 1152 permutations with nonlinearity 104 were generated, that shows the effectiveness of the proposed method.

Additional tests have shown that for the nonlinearity greater than 104, the substitutions are not optimal in terms of algebraic immunity. However, there are permutations with nonlinearity 106 and algebraic immunity 2, in which the number of equations is small (e.g. 2). Hereby, the question about existence of substitutions with algebraic immunity 3 and nonlinearity more than 104 remains open.

5. Conclusions

The proposed method is based on the already known method of gradient descent, but was adopted for vectorial case. It allows to find substitutions with desired properties, in contrast to the previous one, which only could find separate Boolean functions. Such substitutions can be used in modern symmetric algorithms that demand high level of robustness against various types of attacks.

References

[1] Carlet, C.: *Vectorial Boolean functions for cryptography*. Boolean Models and Methods in Mathematics, Computer Science, and Engineering. Cambridge University Press, 2010.

[2] Kazymyrov, O., Oliynykov, R.: An impact of S-box Boolean function properties to strength of modern symmetric block ciphers. In *Radiotechnics*, vol. 166, pp. 11–16. Kharkiv National University of Radioelectronics, 2011. (In Russian).

[3] Rijmen, V.: *Cryptanalysis and design of iterated block ciphers*. Ph.D. thesis, Katholieke Universiteit Leuven, Belgium, 1997.

[4] Kazymyrov, O., Oliynykov, R.: Application of vectorial Boolean functions for substitutions generation used in symmetric cryptographic transformations. In *Information Processing Systems*, vol. 6 (104), pp. 97–102. V. N. Karazin Kharkov National University, Ukraine, 2012 (In Russian).

[5] Budaghyan, L., Kazymyrov, O.: Verification of restricted EA-equivalence for vectorial Boolean functions. In Özbudak, F., Rodríguez-Henríquez, F. (eds.), *Arithmetic of Finite Fields*, vol. 7369 of *Lecture Notes in Computer Science*, pp. 108–118. Springer Berlin Heidelberg, 2012.

[6] Tesař, P.: A new method for generating high non-linearity S-boxes. In *Radioengineering*, vol. 19, pp. 23–26. Brno University of Technology, 2010. http://www.radioeng.cz/fulltexts/2010/10_01_023_026.pdf.

[7] Izbenko, Y., Kovtun, V., Kuznetsov, A.: The design of Boolean functions by modified hill climbing method. *Cryptology ePrint Archive, Report 2008/111*, 2008. http://eprint.iacr.org/.

[8] Millan, W., Clark, A., Dawson, E.: Boolean function design using hill climbing methods. In Pieprzyk, J., Safavi-Naini, R., Seberry, J. (eds.), *Information Security and Privacy*, vol. 1587 of *Lecture Notes in Computer Science*, pp. 1–11. Springer Berlin Heidelberg, 1999.

[9] Daemen, J., Rijmen, V.: AES proposal: Rijndael. *Electronic source,* 1998. http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf.

[10] Logachev, O., Salnikov, A., Smyshlyaev, S., Yaschenko, V.: *Boolean functions in coding theory and cryptology*. Moscow center for continuous mathematical education, 2012. (In Russian).

[11] Yu, Y., Wang, M., Li, Y.: Constructing differential 4-uniform permutations from know ones. *Cryptology ePrint Archive, Report 2011/047*, 2011. `http://eprint.iacr.org/`.

[12] NOTUR: Technical details of Hexagon. *Electronic source,* 2008. `https://www.notur.no/hardware/hexagon`.

PAPER IV

ALGEBRAIC ASPECTS OF THE RUSSIAN HASH STANDARD GOST R 34.11-2012 *

Oleksandr Kazymyrov Valentyna Kazymyrova

IV

*Kazymyrov, O., Kazymyrova, V.: Algebraic aspects of the Russian hash standard GOST R 34.11-2012. In *Pre-proceedings of 2nd Workshop on Current Trends in Cryptology (CTCrypt 2013)*, pp. 160–176, 2013.

Algebraic aspects of the Russian hash standard GOST R 34.11-2012

Oleksandr Kazymyrov Valentyna Kazymyrova

University of Bergen, Norway
Oleksandr.Kazymyrov@ii.uib.no,
Valentyna.Kazymyrova@student.uib.no

Abstract

New GOST R 34.11-2012 standard has been recently selected by the Russian government to replace the old one. The algorithm is based on the hash function Stribog introduced in 2010. The high-level structure of the new hash function is similar to GOST R 34.11-94 with minor modifications. However, the compression function was changed significantly. Such a choice of the compression algorithm has been motivated by the Rjndael due to simplicity and understandable algebraic structure.

In this paper we consider a number of algebraic aspects of the GOST R 34.11. We show how one can express the cipher in AES-like form over the finite field $\mathbb{F}_{2^8}$, and consider some approaches that can be used for the fast software implementation.

Keywords: hash function, Stribog, GOST R 34.11-2012, finite field.

1. Introduction

Until recently Russia has used a hash function defined by the standard GOST R 34.11-94 [1, 2]. Latest cryptanalytical results show that the standard has weaknesses from the theoretical point of view [3]. Therefore, the government forced to create a new cryptographically strong hash function.

In 2010 at RusCrypto'10 conference a prototype of a perspective hash function also known as "Stribog" [4, 5] was presented. The new algorithm is based on the modified Merkle-Damgård scheme with new compression function and digest sizes of 256 and 512 bits. In 2012 the hash function was accepted as the governmental standard GOST R 34.11-2012 [6–8]. It provides calculation procedure for any binary sequences used in cryptographic methods of information processing including techniques for providing data integrity and

authenticity. This standard can be used for creation, operation and modernization of information systems for different purposes. At the same time the standard GOST R 34.10-2001 was replaced by the new one in 2012 taking into account the new hash algorithm.

The description method of hashing algorithm differs from the AES [9, 10]. It is oriented on engineers and programmers without strong mathematical background and is given in algorithm-like form [7]. Even the Stribog's specification does not give information about algebraic features and properties of basic operations. From the cryptanalytical point of view, it is necessary to have an algebraic structure for being able to find weaknesses and/or prove strengths of the algorithm.

In this paper we give a number of GOST R 34.11-2012 representations and consider an approach that could be applied to find the AES-like form over a finite field $\mathbb{F}_{2^8}$.

2. Description of the GOST R 34.11-2012

Hereinafter we assume that Stribog and GOST R 34.11-2012 are the same algorithms. GOST R 34.11-2012 specifies two iterative hash algorithms called Stribog-256 and Stribog-512 that process output message digest of 256 and 512 bits respectively. These algorithms differ in the initialization vector value and in the truncated message digest to 256 most significant bits (MSBs) in Stibog-256 case. Moreover the standard defines two more transformation that are addition modulo 2^{512} ($\boxplus$) and concatenation of two vectors A and B ($A||B$). The value of IV equals 0^{512} (all zero bits) and $(00000001)^{64}$ (64 bytes of 0x01 each) for Stribog-512 and Stribog-256 respectively.

It should be noted that the byte ordering is not specified in the standard. As in the previous standard bytes of information stored on a hard drive or transmitted to a channel have little-endian notation. That is, the message $M_2 = 0xFBE2E5 \dots E220E5D1$ from Appendix 2.2 [7] is stored on the disk in the from $M_2 = 0xD1E520E2 \dots E5E2FB$. Moreover, decoding the last string using the code page CP1251 (Windows-1251) gives a phrase from "The Tale of Igor's Campaign" [11]. Therefore, the description of the hash function is given in the form provided in the standard. In real applications endianness must be taking into account.

The hash algorithm consists of initialization, iterative and final stages. Figure 1 depicts general iterative structure of the hashing algorithm.

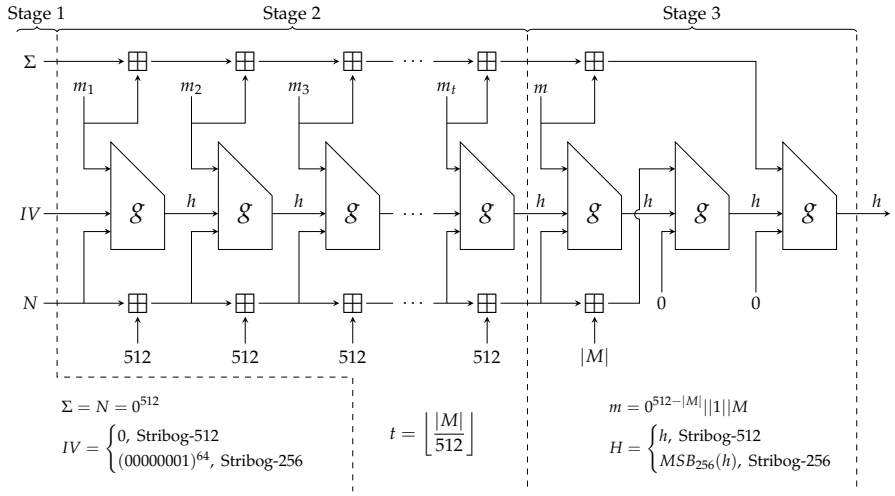


Fig. 1: *Stage Dividing of GOST R 34.11-2012*

At the initialization stage (stage 1) the variables Σ, N and h assign the constant values $0, 0$ and IV respectively. At the next stage, the input message $M = M'||m_i$ divides into messages m_i, $1 \leq i \leq \left\lfloor \frac{|M|}{512} \right\rfloor$, of length 512 bits. Further, for each message m_i the iterative procedure based on a compression function $g_N(h, m)$ is applied. Finally, at the stage 3, consistent application of g_N with different parameters are made for the rest of the message M even if $|M| = 0$.

The standard GOST R 34.11-2012 specifies three main transformations S (SubBytes), P (Transposition) and L (MixColumns). These transformations (see Figure 2) underlie the following compression function $g_N : \mathbb{F}_2^{512} \times \mathbb{F}_2^{512} \mapsto \mathbb{F}_2^{512}$, $N \in \mathbb{F}_2^{512}$

$$g_N(h, m) = E(L \circ P \circ S(h \oplus N), m) \oplus h \oplus m, \quad h, m \in \mathbb{F}_2^{512}.$$

The E function is a block cipher of the form

$$E(K, m) = X[K_{13}] \circ \prod_{i=1}^{12} (L \circ P \circ S \circ X[K_i](m)).$$

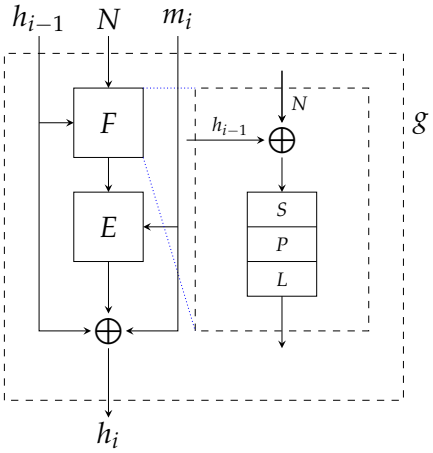


Fig. 2: *Compression Function of GOST R 34.11-2012*

The round keys K_i are calculated using the key schedule procedure with the following algorithm

$$K_i = L \circ P \circ S(K_{i-1} \oplus C_{i-1}),\ K_1 = K,\ i \in \{2, \ldots, 13\}.$$

In [5, 6] values of C_i are defined as the 512-bit constants (see Appendix A). The $X[K_i]$ operation is similar to AddRoundKey(K_i) of AES. The result of $X[K_i](A)$ is the bitwise XOR addition of round key K_i and input vector A.

As in AES the internal state of g_N can be represented as a byte matrix. However, in contrast to AES the Stribog's matrix is 8 by 8 bytes. The correspondence between the input vector B of 64 bytes and the state is presented in Figure 3.

The S transformation is defined as the message partitioning into bytes followed by non-linear bijective mapping of each byte using substitution described in Appendix B. Clearly, the substitution of Stribog differs from the AES one. The maximum absolute value of the bias and the difference probability of the Stribog's S-box equal $\frac{7}{2^6}$ and $\frac{1}{2^5}$ respectively. Other properties are given in Appendix C.

The S transformation is the same as the SubBytes in AES and therefore has the same correspondence between input and output states (Figure 4).

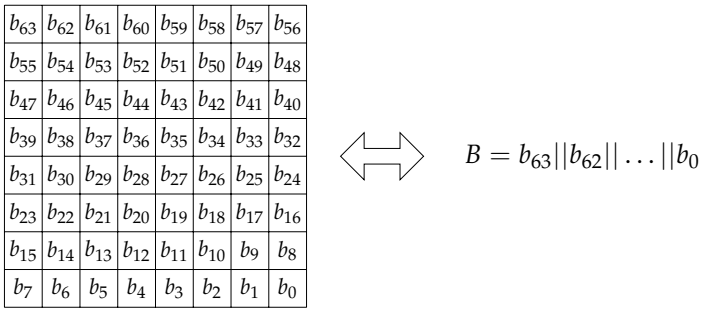


Fig. 3: *State Representation of Stribog*

During the transformation P bits of the input message are grouped into bytes and are permuted in accordance with the permutation τ

$$\begin{aligned}\tau = \{&0, 8, 16, 24, 32, 40, 48, 56, 1, 9, 17, 25, 33, 41, 49, 57,\\ &2, 10, 18, 26, 34, 42, 50, 58, 3, 11, 19, 27, 35, 43, 51, 59,\\ &4, 12, 20, 28, 36, 44, 52, 60, 5, 13, 21, 29, 37, 45, 53, 61,\\ &6, 14, 22, 30, 38, 46, 54, 62, 7, 15, 23, 31, 39, 47, 55, 63\}.\end{aligned}$$

The similar transformation in the AES is ShiftRows. However, P transposes the matrix instead of shifting its rows (Figure 5).

The L transformation is based on a linear transformation l, which is given by the right multiplication by a fixed 64×64 matrix over the field $\mathbb{F}_2$

$$B = A \cdot M,$$

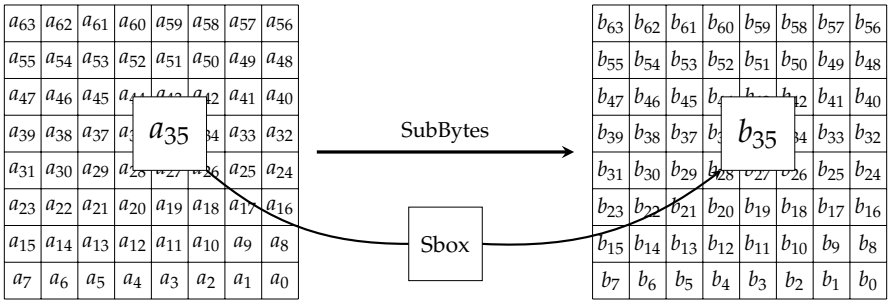


Fig. 4: *The S (SubBytes) Transformation*

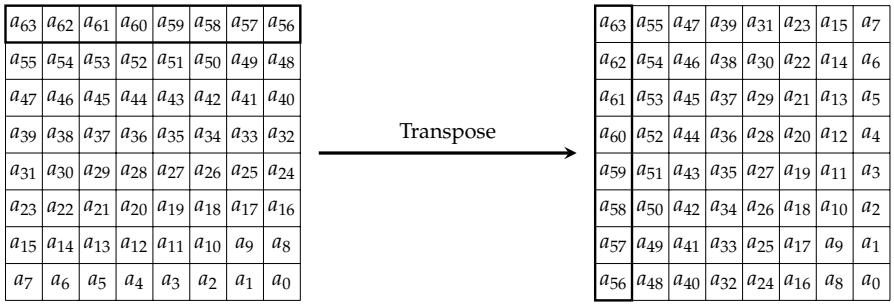


Fig. 5: *The P (Transposition) Transformation*

where A and B are input and output states respectively. Therefore, at the first step of L an input message is converted to the 64-bit vectors. Next, the transformation l applies for each vector (see Appendix D). At the last step, vector values obtained at the previous step are joint into an output message. Figure 6 depicts all these steps which are similar to the MixColumns transformation of AES.

3. AES-like Representation of GOST R 34.11-2012

The description of hash functions, given in the previous section, significantly simplifies understanding of the principles underlying the algorithm compared to one given in the standard GOST R 34.11-2012. However, it does not allow to estimate the security aspects of the hashing algorithm. At the same time,

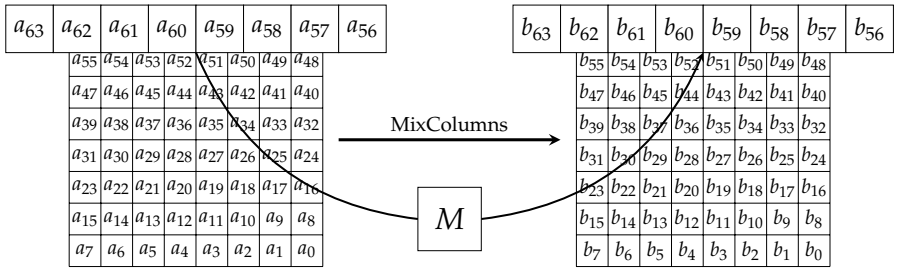


Fig. 6: *The L (MixColumns) Transformation*

the representation of g_N in AES-like form gives the opportunity to use mathematical tools that were created during last 15 years.

Since the state representations in the AES and Stribog are different, a reverse transformation must be applied at the first step to an input message. Suppose R is the transformation which return message with reversed bits. Obviously that $R^{-1} \circ R(x) = R \circ R(x) = x$. Then the compression function of GOST R 34.11-2012 can be performed by following three steps

- reverse input bits;
- AES-like transformations;
- reverse output bits.

The connection between input and output bytes is shown in Figure 7.

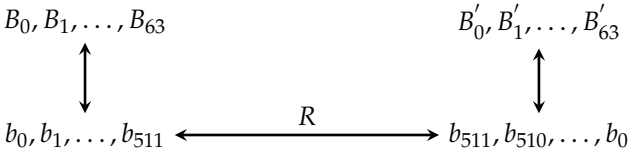


Fig. 7: *Reverse Transformation*

The reverse transformation leads to changing of S, P, L and $X[K]$ transformations of the $g_N(h, m)$ function. Obviously, P and $X[K]$ do not need changes except applying R.

Since the S transformation is based on the constant substitution, applying the function $F'(x) = R \circ F \circ R(x)$, where F is the original S-box, to each byte gives a substitution for AES-like form (Appendix B). It is easy to see that vectorial Boolean functions F' and F are affine equivalent, therefore they have the same properties.

It is well-known that matrix multiplication over $\mathbb{F}_{2^8}$ has at least three forms

- representation over $\mathbb{F}_{2^n}$;
- representation over $\mathbb{F}_2$:
  - using matrix;
  - system of equations.

If a matrix is given over $\mathbb{F}_{2^n}$, then it is easy to find a representation over $\mathbb{F}_2$ for both system of equations and matrix forms. However, the reverse statement in general is not true because of a large amount of possible irreducible polynomials for large n. Nevertheless, for small fields all polynomials are known. There are only 30 irreducible polynomials for $n = 8$ [12].

Let $L : \mathbb{F}_{2^n} \mapsto \mathbb{F}_{2^n}$ be a linear function of the form [13]

$$L(x) = \sum_{i=0}^{n-1} \delta_i x^{2^i}.$$

For $\delta_i = 0$, $1 \leq i < n$, L becomes

$$L(x) = \delta x.$$

This means that any multiplication mapping $\mathbb{F}_{2^n} \mapsto \mathbb{F}_{2^n}$ is a linear transformation of a vector space over $\mathbb{F}_2$ for specified basis. In [13] was shown that multiplication by arbitrary $\delta \in \mathbb{F}_{2^8}$ can be represented as multiplication on a matrix

$$\delta x = \begin{pmatrix} k_{0,0} & \cdots & k_{0,7} \\ k_{1,0} & \cdots & k_{1,7} \\ \vdots & \ddots & \vdots \\ k_{7,0} & \cdots & k_{7,7} \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ \cdots \\ x_7 \end{pmatrix}$$

where $x_i, k_{j,s} \in \mathbb{F}_2$. Using this representation any linear function $L : \mathbb{F}_{2^n} \mapsto \mathbb{F}_{2^m}$ can be converted to a matrix with the computation complexity $O(n)$. Further it was proven that vice versa transformation can be done with the complexity of $O(n^3)$ field operations.

Thus, the algorithm of finding the matrix over F_{2^n} is as follows. For all possible irreducible polynomials convert all $n \times n$ bits submatrices to an element of the field and check MDS property of the resulting matrix.

The matrix over $\mathbb{F}_{2^8}$ with irreducible polynomial $f(x) = x^8 + x^6 + x^5 + x^4 + 1$ received by the algorithm for Stribog is

$$M = \begin{pmatrix} 71 & 05 & 09 & B9 & 61 & A2 & 27 & 0E \\ 04 & 88 & 5B & B2 & E4 & 36 & 5F & 65 \\ 5F & CB & AD & 0F & BA & 2C & 04 & A5 \\ E5 & 01 & 54 & BA & 0F & 11 & 2A & 76 \\ D4 & 81 & 1C & FA & 39 & 5E & 15 & 24 \\ 05 & 71 & 5E & 66 & 17 & 1C & D0 & 02 \\ 2D & F1 & E7 & 28 & 55 & A0 & 4C & 9A \\ 0E & 02 & F6 & 8A & 15 & 9D & 39 & 71 \end{pmatrix}.$$

It should be noted that the binary matrix of Stribog additionally must be transposed [14].

Therefore, the L transformation becomes equivalent to MixColumns of AES and has the form

$$B = M \cdot A.$$

Suppose E^A, L^A, P^A, S^A are AES-like transformations for E, L, P, S respectively. Then it is easy to show (see Appendix E) that the modified $g_N(h, m)$ takes the form depicted in Figure 8.

Since the calculation of block cipher E including key schedule procedure takes most of the time, fast implementation of this part of the hash function is needed for the maximum performance. The description in AES-like form gives access to use tables for increasing performance. Obviously, all optimization techniques described in [10] can be applied to the new standard. Various implementations of the hash function are given in [15].

4. Conclusions

Whole standard has been written in algorithm way and oriented on end developers. Shifting from functional and algorithmic description to logical and mathematical, which is more familiar for cryptographic primitives, allows us to estimate the security properties of Stribog. Our analysis shows that the algorithm of $g_N(h, m)$ is a modified version of AES with block and key lengths equal 512 bits. AES-like representation enables to prove resistant of the hash function to different types of attacks based on differential and linear cryptanalysis. Additionally, such a form shows that Stribog can be implemented by using tables.

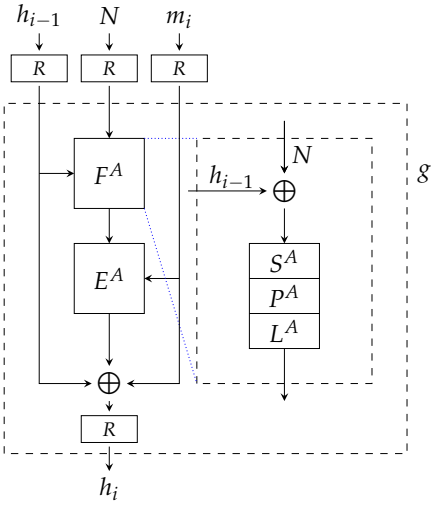


Fig. 8: *The Modified Compression Function for AES-like Representation*

REFERENCES

[1] GOST R 34.11-94: Information technology. Cryptographic data security. Hashing function. *Federal Agency on Technical Regulation and Metrology*, p. 16, 1994.

[2] DOLMATOV, V.: GOST R 34.11-94: Hash Function Algorithm. RFC 5831 (Informational), March 2010. Updated by RFC 6986.

[3] MENDEL, F., PRAMSTALLER, N., RECHBERGER, C., KONTAK, M., SZMIDT, J.: Cryptanalysis of the GOST hash function. In WAGNER, D. (ed.), *Advances in Cryptology – CRYPTO 2008*, vol. 5157 of *Lecture Notes in Computer Science*, pp. 162–178. Springer Berlin Heidelberg, 2008.

[4] MATYUKHIN, D., RUDSKOY, V., SHISHKIN, V.: A perspective hashing algorithm. *Materials of XII scientific conference RusCrypto'2010,* 2010. `http://www.ruscrypto.ru/resource/summary/rc2010/ruscrypto_2010_054.zip`. (In Russian).

[5] GOST R 34.11-20__ (DRAFT) REVISION 1: Information technology. Cryptographic data security. Hash function. *Electronic source,* 2010. `http://infotecs.ru/laws/gost/proj/gost3411.pdf`. (In Russian).

[6] GOST R 34.11-2012: Information technology. Cryptographic data security. Hash-function. *Federal Agency on Technical Regulation and Metrology,* p. 24, 2013. (In Russian).

[7] GOST R 34.11-2012: Information technology. Cryptographic data security. Hash-function. *Federal Agency on Technical Regulation and Metrology,* p. 34, 2013.

[8] Dolmatov, V., Degtyarev, A.: GOST R 34.11-2012: Hash Function. RFC 6986 (Informational), August 2013.

[9] Daemen, J., Rijmen, V.: AES proposal: Rijndael. *Electronic source,* 1998. http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf.

[10] FIPS PUB 197: Advanced Encryption Standard (AES). *National Institute of Standards and Technology,* 2001.

[11] The tale of Igor's campaign. *Electronic source,* 2014. `http://en.wikipedia.org/wiki/The_Tale_of_Igor%27s_Campaign`.

[12] Lidl, R., Niederreiter, H.: *Finite Fields*. Cambridge University Press, Cambridge, 2nd edition, 1997.

[13] Budaghyan, L., Kazymyrov, O.: Verification of restricted EA-equivalence for vectorial Boolean functions. In Özbudak, F., Rodríguez-Henríquez, F. (eds.), *Arithmetic of Finite Fields*, vol. 7369 of *Lecture Notes in Computer Science*, pp. 108–118. Springer Berlin Heidelberg, 2012.

[14] Kazymyrov, O.: Source code of MDS matrices representations over finite fields. *GitHub repository,* 2014. `https://github.com/okazymyrov/MDS`.

[15] Kazymyrov, O., et al.: Source code of the cross-platform implementation of Stribog. *GitHub repository,* 2013. `https://github.com/okazymyrov/stribog`.

A. Constants Values for Key Schedule

The standard GOST R 34.11-2012 specifies the following 12 constants

$$
\begin{aligned}
C_1 =& \texttt{b1085bda1ecadae9ebcb2f81c0657c1f2f6a76432e45d016714eb88d7585c4fc}\\
&\texttt{4b7ce09192676901a2422a08a460d31505767436cc744d23dd806559f2a64507};\\
C_2 =& \texttt{6fa3b58aa99d2f1a4fe39d460f70b5d7f3feea720a232b9861d55e0f16b50131}\\
&\texttt{9ab5176b12d699585cb561c2db0aa7ca55dda21bd7cbcd56e679047021b19bb7};\\
C_3 =& \texttt{f574dcac2bce2fc70a39fc286a3d843506f15e5f529c1f8bf2ea7514b1297b7b}\\
&\texttt{d3e20fe490359eb1c1c93a376062db09c2b6f443867adb31991e96f50aba0ab2};\\
C_4 =& \texttt{ef1fdfb3e81566d2f948e1a05d71e4dd488e857e335c3c7d9d721cad685e353f}\\
&\texttt{a9d72c82ed03d675d8b71333935203be3453eaa193e837f1220cbebc84e3d12e};\\
C_5 =& \texttt{4bea6bacad4747999a3f410c6ca923637f151c1f1686104a359e35d7800fffbd}\\
&\texttt{bfcd1747253af5a3dfff00b723271a167a56a27ea9ea63f5601758fd7c6cfe57};\\
C_6 =& \texttt{ae4faeae1d3ad3d96fa4c33b7a3039c02d66c4f95142a46c187f9ab49af08ec6}\\
&\texttt{cffaa6b71c9ab7b40af21f66c2bec6b6bf71c57236904f35fa68407a46647d6e};\\
C_7 =& \texttt{f4c70e16eeaac5ec51ac86febf240954399ec6c7e6bf87c9d3473e33197a93c9}\\
&\texttt{0992abc52d822c3706476983284a05043517454ca23c4af38886564d3a14d493};\\
C_8 =& \texttt{9b1f5b424d93c9a703e7aa020c6e41414eb7f8719c36de1e89b4443b4ddbc49a}\\
&\texttt{f4892bcb929b069069d18d2bd1a5c42f36acc2355951a8d9a47f0dd4bf02e71e};\\
C_9 =& \texttt{378f5a541631229b944c9ad8ec165fde3a7d3a1b258942243cd955b7e00d0984}\\
&\texttt{800a440bdbb2ceb17b2b8a9aa6079c540e38dc92cb1f2a607261445183235adb};\\
C_{10} =& \texttt{abbedea680056f52382ae548b2e4f3f38941e71cff8a78db1fffe18a1b336103}\\
&\texttt{9fe76702af69334b7a1e6c303b7652f43698fad1153bb6c374b4c7fb98459ced};\\
C_{11} =& \texttt{7bcd9ed0efc889fb3002c6cd635afe94d8fa6bbbebab07612001802114846679}\\
&\texttt{8a1d71efea48b9caefbacd1d7d476e98dea2594ac06fd85d6bcaa4cd81f32d1b};\\
C_{12} =& \texttt{378ee767f11631bad21380b00449b17acda43c32bcdf1d77f82012d430219f9b}\\
&\texttt{5d80ef9d1891cc86e71da4aa88e12852faf417d5d9b21b9948bc924af11bd720}.
\end{aligned}
$$

The modified constants for AES-like representation are given below.

$$C_1^A = e0a2654f9aa601bbc4b22e336c2e6ea0a8cb0625105442458096e64989073ed23 f23a1aeb11d728e680ba274c26e56f4f83ea60381f4d3d7975b53785bda108d;$$

$$C_2^A = edd98d840e209e676ab3d3ebd845bbaa53e550db4386ad3a1a996b48d6e8ad59 8c80ad68f07aab8619d4c4504e577fcfebad0ef062b9c7f258f4b99551adc5f6;$$

$$C_3^A = 4d505d50af6978998cdb5e61c22f6d4390db4606ec5c93838d79ac0927f047cb dede948d28ae574fd1f8394afa7a8f60ac21bc56143f9c50e3f473d4353b2eaf;$$

$$C_4^A = 748bc7213d7d30448fec17c98557ca2c7dc04ac9ccc8ed1bae6bc0b74134eb95 fcac7a16b5384eb9be3c3acc7ea17112bb278eba0587129f4b66a817cdfbf8f7;$$

$$C_5^A = ea7f363ebf1ae806afc657957e456a5e6858e4c4ed00fffbc5af5ca4e2e8b3fd bdfff001ebac79ac52086168f838a8fec6c495363082fc5999e2e2b535d657d2;$$

$$C_6^A = 76be26625e02165facf2096c4ea38efd6d637d4366f84f502ded5938ed655ff3 63710f592d59fe183625428a9f2366b4039c0c5edcc325f69bcb5cb87575f275;$$

$$C_7^A = c92b285cb26a6111cf523c4532a2e8ac20a05214c196e260ec3441b4a3d54990 93c95e98cc7ce2cb93e1fd67e363799c2a9024fd7f61358a37a355776870e32f;$$

$$C_8^A = 78e740fd2bb0fe259b158a9aac43356cf423a58bd4b18b960960d949d3d4912f 5923dbb2dc222d91787b6c398e1fed72828276304055e7c0e593c9b242daf8d9;$$

$$C_9^A = db5ac4c18a22864e0654f8d3493b1c702a39e0655951d4de8d734ddbd0225001 2190b007edaa9b3c244291a4d85cbe5c7bfa68371b593229d9448c682a5af1ec;$$

$$C_{10}^A = b739a219dfe32d2ec36ddca88b5f196c2f4a6edc0c36785ed2cc96f540e6e7f9 c086ccd85187fff8db1e51ff38e78291cfcf274d12a7541c4af6a001657b7dd5;$$

$$C_{11}^A = d8b4cf81b32553d6ba1bf603529a457b1976e2beb8b35df7539d1257f78eb851 9e6621288401800486e0d5d7ddd65f1b297f5ac6b363400cdf9113f70b79b3de;$$

$$C_{12}^A = 04ebd88f52493d1299d84d9babe82f5f4a1487115525b8e761338918b9f701ba d9f9840c2b48041feeb8fb3d4c3c25b35e8d92200d01c84b5d8c688fe6e771ec.$$

B. Stribog's Lookup Tables

The following two tables describe the substitutions for the original GOST R 34.11-2012 and AES-like representations. All values in the table have hexadecimal notation.

Table 1: *Substitution Box of GOST R 34.11-2012*

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	FC	EE	DD	11	CF	6E	31	16	FB	C4	FA	DA	23	C5	04	4D
1	E9	77	F0	DB	93	2E	99	BA	17	36	F1	BB	14	CD	5F	C1
2	F9	18	65	5A	E2	5C	EF	21	81	1C	3C	42	8B	01	8E	4F
3	05	84	02	AE	E3	6A	8F	A0	06	0B	ED	98	7F	D4	D3	1F
4	EB	34	2C	51	EA	C8	48	AB	F2	2A	68	A2	FD	3A	CE	CC
5	B5	70	0E	56	08	0C	76	12	BF	72	13	47	9C	B7	5D	87
6	15	A1	96	29	10	7B	9A	C7	F3	91	78	6F	9D	9E	B2	B1
7	32	75	19	3D	FF	35	8A	7E	6D	54	C6	80	C3	BD	0D	57
8	DF	F5	24	A9	3E	A8	43	C9	D7	79	D6	F6	7C	22	B9	03
9	E0	0F	EC	DE	7A	94	B0	BC	DC	E8	28	50	4E	33	0A	4A
A	A7	97	60	73	1E	00	62	44	1A	B8	38	82	64	9F	26	41
B	AD	45	46	92	27	5E	55	2F	8C	A3	A5	7D	69	D5	95	3B
C	07	58	B3	40	86	AC	1D	F7	30	37	6B	E4	88	D9	E7	89
D	E1	1B	83	49	4C	3F	F8	FE	8D	53	AA	90	CA	D8	85	61
E	20	71	67	A4	2D	2B	09	5B	CB	9B	25	D0	BE	E5	6C	52
F	59	A6	74	D2	E6	F4	B4	C0	D1	66	AF	C2	39	4B	63	B6

Table 2: *Substitution Box of GOST R 34.11-2012 for AES-like form*

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	3F	FB	D7	E0	9F	E5	A8	04	97	07	AD	87	A0	B5	4C	9A
1	DF	EB	4F	0C	81	58	CF	D3	E8	3B	FD	B1	60	31	B6	8B
2	F3	7C	57	61	47	78	08	B4	C9	5E	10	32	C7	E4	FF	67
3	C4	3E	BF	11	D1	26	B9	7D	28	72	39	53	FE	96	C3	9C
4	BB	24	34	CD	A6	06	69	E6	0F	37	70	C1	40	62	98	2E
5	5F	6B	16	D6	3C	1C	1E	A4	8F	14	C8	55	B7	A5	63	F5
6	8C	C2	12	B8	F7	46	59	90	99	0D	6E	1F	F1	AA	51	2D
7	20	9D	73	E7	71	64	4D	36	FA	50	BA	A1	CB	A9	B0	C6
8	77	AF	2C	1A	18	E9	85	8E	EE	F0	0E	D8	21	A2	AE	65
9	23	9E	54	EC	38	1D	89	D9	6C	17	4E	CA	D0	C5	2A	66
A	76	15	13	35	3A	00	DE	D4	74	29	30	FC	56	7A	AC	2F
B	A3	44	5C	9B	80	F9	79	A7	B3	CC	ED	1B	2B	AB	BD	D2
C	88	95	8A	02	5A	CE	94	25	DB	7B	6A	92	75	49	BC	4B
D	5B	6F	45	27	42	41	F6	0B	DD	0A	E2	09	19	BE	01	43
E	68	93	D5	EF	84	22	E3	DA	5D	3D	48	7F	05	F4	7E	03
F	B2	C0	33	91	F2	82	8D	4A	83	52	E1	86	F8	DC	EA	6D

C. S-box Properties of GOST R 34.11-2012

The comparison of Stribog and the AES substitutions is given in the following table. All properties presented in Table 3 were calculated according to the componet functions, which are the linear combinations (with non all-zero coeffcients) of the coordinate functions [13].

Table 3: *Comparison of Stribog and AES Substitutions*

Properties	Stribog	AES
Vectorial Boolean Function		
Balancedness	True	True
Nonlinearity	100	112
Absolute Indicator	96	32
Sum-of-squares Indicator	258688	133120
Propogation Criterion	0	0
Correlation Immunity	0	0
Minimum of Algebraic Degree	7	7
Resiliency	0	0
Strict Avalanche Criterion	False	False
Properties	Stribog	AES
Substitution		
Bijection	True	True
Maximum of Differential Table	8	4
Maximum of Approximation Table	28	16
Cycles Structure	252:243, 46:13	43:27, 242:87, 99:59, 124:81, 143:2
Algebraic Immunity	3(441)	2(39)

D. The Constant Matrix for the l Transformation

The constant matrix is given in Table 4. Each value in the cell has hexadecimal notation and corresponds to a matrix row with index $i \cdot 4 + j$, $i = \{0, \ldots, 15\}, j = \{0, \ldots, 3\}$. For example, the row $21 = 5 \cdot 4 + 1$ is 8a174a9ec8121e5d.

Table 4: *The Constant Matrix of the Standard GOST R 34.11-2012*

i \ j	0	1	2	3
0	8e20faa72ba0b470	47107ddd9b505a38	ad08b0e0c3282d1c	d8045870ef14980e
1	6c022c38f90a4c07	3601161cf205268d	1b8e0b0e798c13c8	83478b07b2468764
2	a011d380818e8f40	5086e740ce47c920	2843fd2067adea10	14aff010bdd87508
3	0ad97808d06cb404	05e23c0468365a02	8c711e02341b2d01	46b60f011a83988e
4	90dab52a387ae76f	486dd4151c3dfdb9	24b86a840e90f0d2	125c354207487869
5	092e94218d243cba	8a174a9ec8121e5d	4585254f64090fa0	accc9ca9328a8950
6	9d4df05d5f661451	c0a878a0a1330aa6	60543c50de970553	302a1e286fc58ca7
7	18150f14b9ec46dd	0c84890ad27623e0	0642ca05693b9f70	0321658cba93c138
8	86275df09ce8aaa8	439da0784e745554	afc0503c273aa42a	d960281e9d1d5215
9	e230140fc0802984	71180a8960409a42	b60c05ca30204d21	5b068c651810a89e
A	456c34887a3805b9	ac361a443d1c8cd2	561b0d22900e4669	2b838811480723ba
B	9bcf4486248d9f5d	c3e9224312c8c1a0	effa11af0964ee50	f97d86d98a327728
C	e4fa2054a80b329c	727d102a548b194e	39b008152acb8227	9258048415eb419d
D	492c024284fbaec0	aa16012142f35760	550b8e9e21f7a530	a48b474f9ef5dc18
E	70a6a56e2440598e	3853dc371220a247	1ca76e95091051ad	0edd37c48a08a6d8
F	07e095624504536c	8d70c431ac02a736	c83862965601dd1b	641c314b2b8ee083

E. The Proof of AES-like Representation of $g_N(h, m)$

Taking into account all statements for L, P, S functions from Section 3, the modified E (E^A) takes the form

$$E^A(K, m) = \Big(R \circ X[K_{13}^A] \circ R\Big) \circ \prod_{i=2}^{12} \Big(\Big(R \circ L^A \circ R\Big) \circ \Big(R \circ P^A \circ R\Big) \circ$$
$$\circ \Big(R \circ S^A \circ R\Big) \circ \Big(R \circ X[K_i^A] \circ R\Big)\Big) \circ \Big(\Big(R \circ L^A \circ R\Big) \circ$$
$$\circ \Big(R \circ P^A \circ R\Big) \circ \Big(R \circ S^A \circ R\Big) \circ \Big(R \circ X[K_1^A] \circ R(m)\Big)\Big) = R \circ X[K_{13}^A] \circ$$
$$\circ \prod_{i=2}^{12} \Big(L^A \circ P^A \circ S^A \circ (X[K_i^A]\Big) \circ \Big(L^A \circ P^A \circ S^A \circ X[K_1^A] \circ R(m)\Big).$$

In fact, the message m is reversed at previous steps before calling the function $g_N(h, m)$. The final R is applied for the result of $g_N(h, m)$. Thus, the

final algorithm of E^A has the form

$$E^A(K,m) = X[K_{13}^A] \circ \prod_{i=1}^{12} \left(L^A \circ P^A \circ S^A \circ X[K_i^A](m) \right).$$

The round keys K_i^A are calculated using received constants C_i^A (see. Appendix A)

$$K_i^A = L^A \circ P^A \circ S^A(K_{i-1}^A \oplus C_{i-1}^A),\ K_1^A = K^A,\ i \in \{2,\ldots,13\}.$$

All of the above lead to the modification of the whole function $g_N(h,m)$ (Figure 8)

$$K^A = L^A \circ P^A \circ S^A(R(h) \oplus R(N))$$
$$g_N(h,m) = R \circ \left(E(K^A, R(m)) \oplus R(h) \oplus R(m) \right).$$

PAPER V

EXTENDED CRITERION FOR ABSENCE OF FIXED POINTS *

Oleksandr Kazymyrov Valentyna Kazymyrova

V

*Kazymyrov, O., Kazymyrova, V.: Extended criterion for absence of fixed points. In *Pre-proceedings of 2nd Workshop on Current Trends in Cryptology (CTCrypt 2013)*, pp. 177–191, 2013.

Extended Criterion for Absence of Fixed Points

Oleksandr Kazymyrov Valentyna Kazymyrova

University of Bergen, Norway
Oleksandr.Kazymyrov@ii.uib.no,
Valentyna.Kazymyrova@student.uib.no

Abstract

One of the criteria for substitutions used in block ciphers is the absence of fixed points. In this paper we show that this criterion must be extended taking into consideration a mixing key function. In practice, we give a description of AES when fixed points are reached. Additionally, it is shown that modulo addition has more advantages then XOR operation.

Keywords: S-box, Block Cipher, Fixed Point, AES.

1. Introduction

Substitution boxes (S-boxes) map an n-bit input message to an m-bit output message. They provide confusion in symmetric algorithms. For different tasks S-boxes are used in various forms. In stream ciphers a substitution is represented usually as a vectorial Boolean function [1]. Permutations are a subclass of substitutions and are commonly used in block ciphers as lookup tables. Regardless of ciphers an S-box can be converted from one form to another one.

Substitutions must satisfy various criteria for providing high level of protection against different types of attacks [2]. A substitution satisfying all criteria is perfect. However, such substitutions do not exist up to date. Therefore, in practice, substitutions satisfying several important criteria are used. They are called optimal S-boxes. Optimality criteria vary from cipher to cipher. Generating permutations with optimal criteria is a quite difficult task, especially for a large n and m. The problem of generating a set of S-boxes with similar properties can be particularly solved by using EA- or CCZ-equivalence [3, 4].

One of criteria is absence of fixed points. It is used in many ciphers for increasing resistance against statistical attacks [5]. Designers of modern cryptographic primitives try to get rid of the fixed points. This is achieved by applying affine equivalence, which is a special case of EA-equivalence. The S-box of advanced encryption standard (AES) was constructed using this technique [5, 6]. However, the application of this method does not totally prevent the appearance of fixed points. In this paper we show an isomorphic (equivalent) form of AES when fixed points are reached.

Two ciphers E_i and E_j are isomorphic to each other if there exist invertible maps $\phi : x^i \mapsto x^j$, $\psi : y^i \mapsto y^j$ and $\chi : k^i \mapsto k^j$ such that $y^i = E_i(x^i, k^i)$ and $y^j = E_j(x^j, k^j)$ are equal for all x^i, k^i, x^j and k^j [7, 8]. Obviously, the cipher can have a lot of isomorphic basic transformations as well as full encryption procedures. The cipher BES is a well-known example of isomorphic AES [9]. Another example of isomorphic AES is the description of encryption procedure using system of equation of degree 2 [10]. We give one more description of AES which includes a substitution with a fixed point while almost all transformations are unmodified.

2. PRELIMINARIES

Arbitrary substitution can be represented at least in three different forms: algebraic normal form (ANF), over field $\mathbb{F}_{2^n}$ and as a lookup table. Most of substitutions used in block ciphers have a table representation because of simplicity of description and understanding [11]. Meanwhile arbitrary S-box S can be always associated with a vectorial Boolean function F in $\mathbb{F}_{2^n}[x]$. If a substitution is a permutation then F is defined uniquely [1].

The natural way of representing F as a function from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ is by its algebraic normal form:

$$\sum_{I \subseteq \{1,\ldots,n\}} a_I \left(\prod_{i \in I} x_i \right), \qquad a_I \in \mathbb{F}_2^m,$$

the sum is being calculated in $\mathbb{F}_2^m$ [1]. The algebraic degree of F is the degree of its ANF. F is called affine if it has algebraic degree at most 1 and it is called linear if it is affine and $F(0) = 0$. A vectorial Boolean function given in table representation can be easily transformed to ANF form and vice versa.

Two functions $F, G : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ are called extended affine equivalent (EA-equivalent) if there exist an affine permutation A_1 of $\mathbb{F}_2^m$, an affine permutation A_2 of $\mathbb{F}_2^n$ and a linear function L_3 from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ such that

$$F(x) = A_1 \circ G \circ A_2(x) + L_3(x). \tag{1}$$

Clearly, A_1 and A_2 can be presented as $A_1(x) = L_1(x) + c_1$ and $A_2(x) = L_2(x) + c_2$ for some linear permutations L_1 and L_2 and some $c_1 \in \mathbb{F}_2^m$, $c_2 \in \mathbb{F}_2^n$. Two functions F and G are linear equivalent if equation (1) is hold for $L_3(x) = 0$, $c_1 = 0$, $c_2 = 0$. If the equation (1) is preserved only for $L_3(x) = 0$, then functions F and G are called affine equivalent [12].

In matrix form EA-equivalence is represented as follows

$$F(x) = M_1 \cdot G(M_2 \cdot x \oplus V_2) \oplus M_3 \cdot x \oplus V_1$$

where elements of $\{M_1, M_2, M_3, V_1, V_2\}$ have dimensions $\{m \times m, n \times n, m \times n, m, n\}$ [3].

An element $a \in \mathbb{F}_2^n$ is a fixed point of $F : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ if $F(a) = a$. The absence of fixed points criterion is defined as follows.

Proposition 1. *A substitution must not have fixed points, i.e.*

$$F(a) \neq a, \quad \forall a \in \mathbb{F}_2^n .$$

For any positive integers n and m, a function F from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ is called differentially δ-uniform if for every $a \in \mathbb{F}_2^n \setminus \{0\}$ and every $b \in \mathbb{F}_2^m$, the equation $F(x) + F(x + a) = b$ admits at most δ solutions [1]. Vectorial Boolean functions used as S-boxes in block ciphers must have low differential uniformity to allow high resistance to differential cryptanalysis [13].

The nonlinearity criterion is closely connected to the notion of Walsh transform which can be described as the function

$$\lambda(u, v) = \sum_{x \in \mathbb{F}_2^n} (-1)^{v \cdot F(x) + u \cdot x},$$

where "$\cdot$" denotes inner products in $\mathbb{F}_2^n$ and $\mathbb{F}_2^m$ respectively [1]. A substitution has an optimal resistance to linear cryptanalysis if the maximum absolute value of Walsh coefficients is small [14]. Substitutions with the smallest value of $\lambda(u, v)$ exist for odd n only.

These two criteria are major while selecting substitutions for new ciphers. However, there are many others criteria like propagation criterion, absolute

indicator, correlation immunity, strict avalanche criterion, etc [1, 2, 15]. It has been still not proven the importance of the criteria for block ciphers. For example, the substitution used in AES does not satisfy most of them [16]. Moreover, no theoretical or practical attacks were proposed on modern block ciphers based on these criteria.

Let $E : \{0,1\}^l \times \{0,1\}^k \mapsto \{0,1\}^l$ be a function taking a key K of length k bits and input message (plaintext) M of length l bits and return output message (ciphertext) $E(M,K)$. For each key K let $E_K : \{0,1\}^l \mapsto \{0,1\}^l$ be a function defined by $E_K(M) = E(M,K)$. Then E is a block cipher if E_K and E_K^{-1} are efficiently computable and E_K is a permutation for every K.

Most of the modern block ciphers are iterative (Fig. 1). Usually a round function is run multiple times with different parameters (round keys). An arbitrary iterative block cipher can be mathematically described as follows

$$E_K(M) = PW_{k_{r+1}} \circ \prod_{i=2}^{r}(R_{k_i}) \circ IW_{k_1}(M),$$

where R is a round procedure, IW is a prewhitening procedure and PW is a postwhitening procedure. In Fig. 1 a key schedule is an algorithm that takes a master key K as input and produces the subkeys $k_1, k_2, \ldots, k_{r+1}$ for all stages of encryption algorithm.

A mixing key procedure of a block cipher is an algorithm which injects a round key into an encryption procedure. In the majority of the modern block ciphers, the mixing key function is implemented as exclusive or (XOR) operation because of low-cost implementations.

3. A Brief Description of AES

AES is a substitution permutation network (SPN) block cipher that supports a fixed block size of 128 bits and a key size of 128, 192 or 256 bits [6]. The number of rounds depends on the key size and is equal to 10, 12 or 14, respectively. The round function consists of four functions: AddRoundKey (σ_k), SubBytes (γ), ShiftRows (π) and MixColumns (θ).

The entire encryption algorithm is described as follows (Fig. 2)

$$E_K(M) = \sigma_{k_{r+1}} \circ \pi \circ \gamma \circ \prod_{i=2}^{r}(\sigma_{k_i} \circ \theta \circ \pi \circ \gamma) \circ \sigma_{k_1}(M).$$

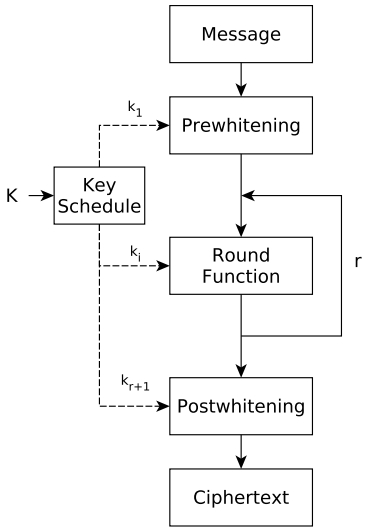


Fig. 1: *General Structure of An Iterative Block Cipher*

The SubBytes transformation processes the state of the cipher using a nonÂlinear byte substitution table that operates on each of the state bytes independently [6]. The *S*-box of AES was generated by finding the inverse element in the field $\mathbb{F}_{2^8}$ followed by applying affine polynomial. In terms of equation (1) the transformation has the form

$$F(x) = A_1(x^{-1}) = L_1(x^{-1}) + c_1.$$

The substitution table generated by vectorial Boolean function $F : \mathbb{F}_{2^8} \mapsto \mathbb{F}_{2^8}$ satisfies the following criteria

- the maximum value of non-trivial XOR difference transformation probability is 2^{-6};
- the maximum absolute value of linear approximation probability bias is 2^{-4};
- the minimum algebraic degree of the component functions is 7 [5, 17].

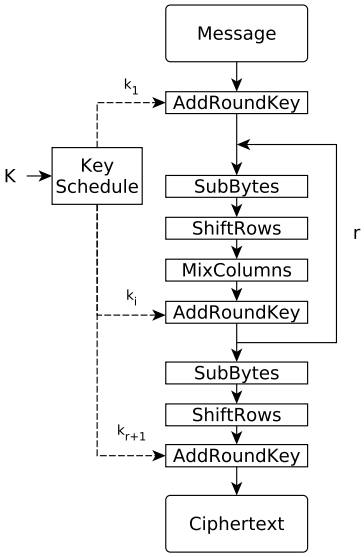


Fig. 2: *Encryption Algorithm of AES*

It should be noticed that the chosen polynomial x^{-1} allows to describe the S-box and the entire cipher by overdefined system of equations of degree 2 [18]. But in the same time it gives resistant to differential, linear and many other cryptanalytical methods. In addition to the general properties, the constant of the AES S-box has been chosen in such way that it has no fixed points [5].

The MixColumns transformation takes all the columns of the state and mixes their data (independently of one another) to produce new columns [6]. This transformation can be represented in different ways. One of them is the matrix multiplication. For an input state x and 4×4 matrix M the output state y of the transformation is described as

$$y = M \cdot x.$$

The matrix with maximum distance separable (MDS) property is used in AES. In terms of Rijndael the MDS property associates with a branch number (β)

$$\beta = \min_{x \neq 0}(W(x) + W(y)),$$

where $W(z)$ is the byte weight of a vector z.

From the definition of MDS matrix, it is known that the maximum differential branch number of m by m matrix is $m+1$ [11, 19]. Hence, MDS matrices have the perfect diffusion property for byte-oriented ciphers.

Multiplication in a field $\mathbb{F}_{2^n}$ is a linear transformation with respect to XOR, so it preserves the linear property [9]

$$\theta(x+y) = \theta(y) + \theta(y).$$

The ShiftRows transformation processes the state by cyclically shifting the last three rows of the state by different offsets [6]. More precisely, row i is moved to the left by i byte positions for $0 \leq i \leq 3$. The ShiftRows is also a linear function that preserves $\pi(x+y) = \pi(y) + \pi(y)$ property.

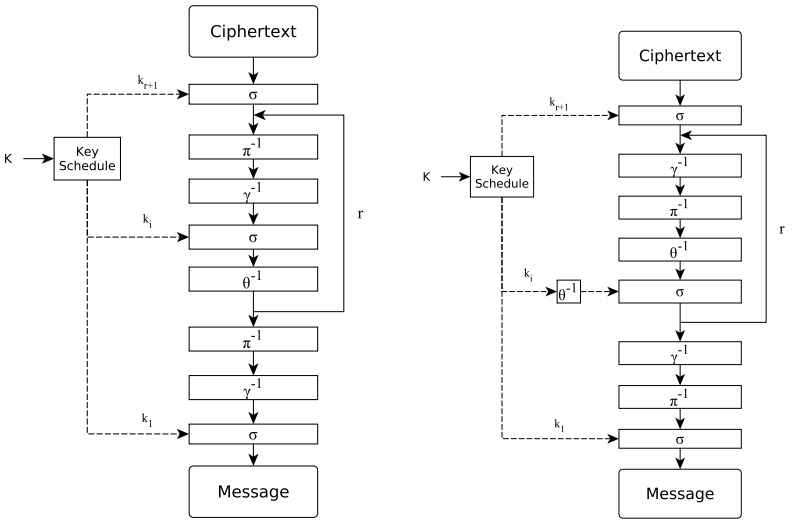
Both MixColumns and ShiftRows transformations help to ensure that the number of active S-boxes is large even after few rounds [5]. These functions are the basis of protection offered by the AES against differential and linear cryptanalysis.

AddRoundKey transformation is the mixing key function in which a round key is added to the state using XOR operation. The length of a round key is equal to the size of the state. XOR operation of two n-bit length vectors a and b can be performed bit by bit n times. Therefore, AddRoundKey operation of AES can be done independently of each byte.

4. A New Cipher Isomorphic to AES

There exist several examples of ciphers isomorphic to AES. For example, the big encryption system (BES) describes AES over $\mathbb{F}_{2^8}$ [9]. On the other hand, the cipher AES can be also represented as the system of multivariate equations of the 2^{nd} degree over $\mathbb{F}_2$ [18]. These two examples are based on the algebraic features of the substitution. However, there is another approach based on linear properties of the basic functions (MixColumns and ShiftRows).

The cipher AES is based on Rijndael that was proposed by Daemen and Rijmen to AES process [20]. Authors have used design simplicity principle, which led to performance improvement and code compactness properties of the cipher on a wide range of platforms. For increasing decryption performance of software implementation they have used precomputed lookup tables and the linear properties of the basic functions.



(a) *Original* **(b)** *Algorithm for the Fast Software Implementation*

Fig. 3: *Decryption Algorithm of AES*

The original decryption algorithm for arbitrary ciphertext C mathematically can be represented as follows (Fig. 3a) [6]

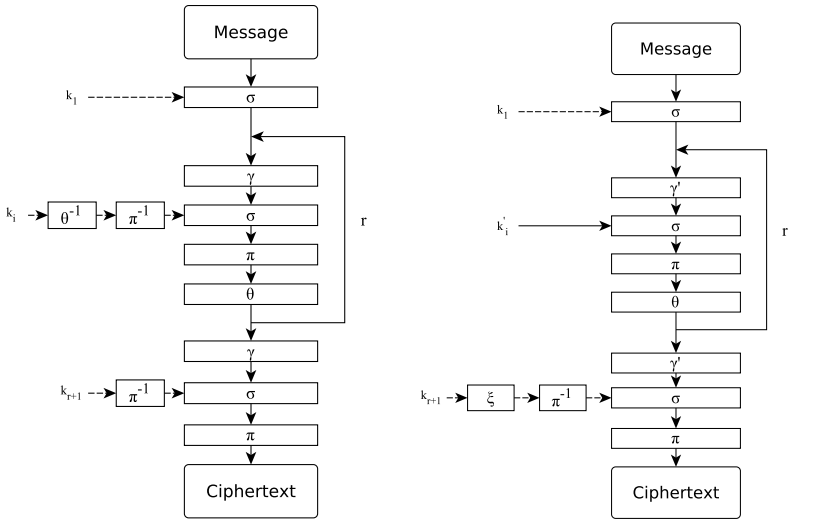
$$D_K(C) = \sigma_{k_1} \circ \gamma^{-1} \circ \pi^{-1} \circ \prod_{i=2}^{r} (\theta^{-1} \circ \sigma_{k_{r-i+2}} \circ \gamma^{-1} \circ \pi^{-1}) \circ \sigma_{k_{r+1}}(C).$$

For using precomputed tables it is necessary to transform the decryption round function to the similar one of encryption algorithm. Since functions γ^{-1} and π^{-1} are computed independently they have the commutative property $\gamma^{-1} \circ \pi^{-1} = \pi^{-1} \circ \gamma^{-1}$ [5, 9]. In Section 3 it was stated that functions θ^{-1} and σ are linear w.r.t. XOR, hence

$$\theta^{-1} \circ \sigma_{k_{r-i+2}} = \sigma_{\theta^{-1}(k_{r-i+2})} \circ \theta^{-1}$$

Thus, the whole decryption algorithm has the form (Fig. 3b)

$$D_K(C) = \sigma_{k_1} \circ \pi^{-1} \circ \gamma^{-1} \circ \prod_{i=2}^{r} (\sigma_{\theta^{-1}(k_{r-i+2})} \circ \theta^{-1} \circ \pi^{-1} \circ \gamma^{-1}) \circ \sigma_{k_{r+1}}(C).$$



(a) *Modified Encryption Algorithm*

(b) *Isomorphic Encryption Algorithm with a Fixed Point*

Fig. 4: *Isomorphism of AES*

Usage of such elementary transformations helps to achieve a significant acceleration of the decryption procedure due to the isomorphic properties of the basic functions [5].

Obviously, the same technique can be applied to the encryption algorithm. However, the task is to find a representation of the cipher in which properties of a new substitution will differ from the original one. For simplicity of description, let us assume that the round keys are independent of each other. Then the encryption procedure takes a form (Fig. 4a)

$$E_K(M) = \pi \circ \sigma_{\pi^{-1}(k_{r+1})} \circ \gamma \circ \prod_{i=2}^{r} (\theta \circ \pi \circ \sigma_{\pi^{-1} \circ \theta^{-1}(k_i)} \circ \gamma) \circ \sigma_{k_1}(M).$$

The last equation shows that the final ShiftRows operation is redundant in terms of resistance to attacks. As it was stated above the availability of this function is necessary for fast implementation of the decryption procedure.

Arbitrary permutation S can be represented as a vectorial Boolean function $F : \mathbb{F}_{2^n} \mapsto \mathbb{F}_{2^n}$ which has the form [3]

$$F(x) = F'(x) + F(0).$$

Since the characteristic of the field is 2, the constant can be moved to the round keys. Let ξ be a function in which the constant $F(0)$ is XORed with all bytes of a state. If the round keys $\pi^{-1} \circ \theta^{-1} \circ \xi(k_i)$ are denoted by k'_i then encryption procedure takes the form (Fig. 4b)

$$E_K(M) = \pi \circ \sigma_{\pi^{-1} \circ \xi(k_{r+1})} \circ \gamma' \circ \prod_{i=2}^{r} (\theta \circ \pi \circ \sigma_{k'_i} \circ \gamma') \circ \sigma_{k_1}(M),$$

where γ' is the SubBytes function which consists of the substitution of the form $F(x) = L(x^{-1})$.

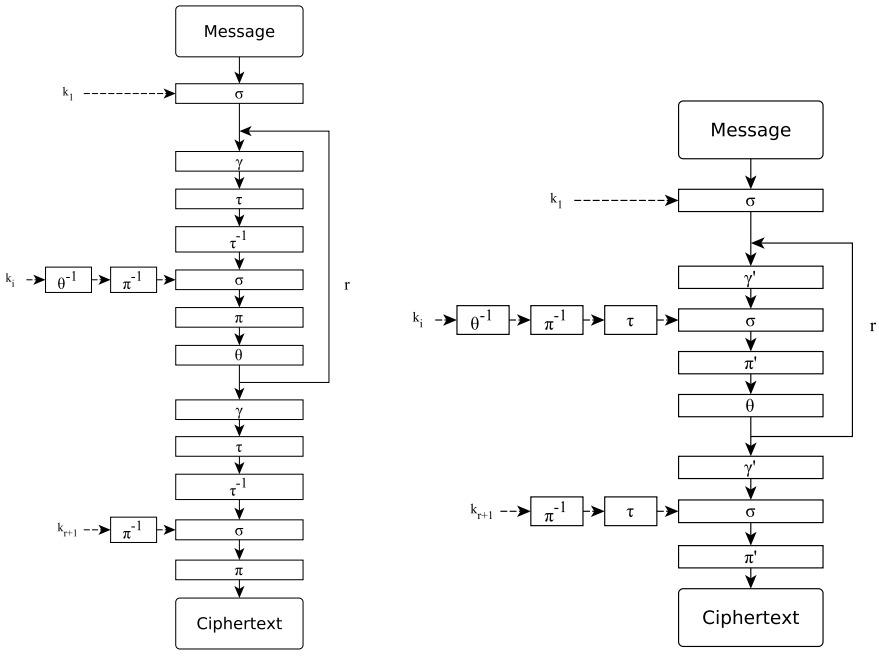
Fig. 4b shows that the structure of the cipher remains unchanged. Clearly, if an adversary finds a round key for modified cipher she also automatically obtains corresponding round key of the original cipher because of the linear dependence of the keys k_i and k'_i. However, the new substitution $F(x) = L(x^{-1})$ has the fixed point in $x = 0$. Consequently, the substitution of AES doesn't satisfy the absence of fixed points criterion.

Described features of the cipher appears from the fact that the operation XOR is linear with respect to MixColumns and ShiftRows. If one replaces AddRoundKey with some nonlinear function (i.e. based on addition modulo 2^n), then it will be impossible to find an isomorphic cipher of such a form. From this point of view a mixed key function based on modulo addition is cryptographically stronger than a function based on XOR operation .

Furthermore, fixed points are directly connected with cyclic properties of substitutions. Inserting an invertible linear function (τ) into the encryption procedure gives a new isomorphic cipher (Fig. 5a). Herewith, the linearized polynomial can be added to the round key and the inverse function can be a part of the new substitution (Fig. 5b). The cyclic properties of the new substitution will depend on the selected function τ.

Thereby, the cyclic and the absence of fixed points properties of a substitution can be controlled by adversary in the case of a linear mixing key function. Thus, a new criterion for substitutions follows from the description above.

Proposition 2. *Substitutions S_1, S_2, ..., S_n used in a confusion layer must belong to different classes of equivalence.*



(a) *Initial Cipher* **(b)** *Isomorphic Cipher*

Fig. 5: *Modified AES with an Invertible Linear Function*

Clearly, if substitutions are in the same class (i.e. EA-equivalent) then the adversary can find an isomorphic cipher which consists of one substitution and modified linear layer. Consequently, there will be no advantages to use multiple substitutions. The criterion has to be considered both in the design of new ciphers and in the analysis of existing ones [21, 22]. Since CCZ-equivalence is the most general case of known equivalence, it makes sense to check whether substitutions belong to different CCZ-equivalence classes.

5. Conclusions

It was shown that the absence of fixed points criterion works only in case if *S*-box is considered as a separate function. There are isomorphic representations of ciphers in which this criterion is not met. The new method of AES

description allows to reconsider some of criteria for substitutions from the practical point of view. This may lead to a weakening of the cipher strength.

Since an invertible linear function can be added to encryption procedure, the adversary can control both the cyclic and absence of fixed points properties of substitutions. It was shown that mixing key function based on modulo addition is more resistant with respect to the absence of fixed points criterion than function based on XOR operation.

Isomorphism of ciphers adds additional restrictions on using multiple substitutions. The proposed criterion can be used to reduce the probability of finding the weakest one.

REFERENCES

[1] CARLET, C.: *Vectorial Boolean functions for cryptography*. Boolean Models and Methods in Mathematics, Computer Science, and Engineering. Cambridge University Press, 2010.

[2] RIJMEN, V.: *Cryptanalysis and design of iterated block ciphers*. Ph.D. thesis, Katholieke Universiteit Leuven, Belgium, 1997.

[3] BUDAGHYAN, L., KAZYMYROV, O.: Verification of restricted EA-equivalence for vectorial Boolean functions. In ÖZBUDAK, F., RODRÍGUEZ-HENRÍQUEZ, F. (eds.), *Arithmetic of Finite Fields*, vol. 7369 of *Lecture Notes in Computer Science*, pp. 108–118. Springer Berlin Heidelberg, 2012.

[4] CARLET, C., CHARPIN, P., ZINOVIEV, V.: Codes, bent functions and permutations suitable for DES-like cryptosystems. In *Designs, Codes and Cryptography*, vol. 15, pp. 125–156. Kluwer Academic Publishers, 1998.

[5] DAEMEN, J., RIJMEN, V.: AES proposal: Rijndael. *Electronic source,* 1998. http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf.

[6] FIPS PUB 197: Advanced Encryption Standard (AES). *National Institute of Standards and Technology,* 2001.

[7] ROSTOVTSEV, A.: Changing probabilities of differentials and linear sums via isomorphisms of ciphers. *Cryptology ePrint Archive, Report 2009/117*, 2009. `http://eprint.iacr.org/`.

[8] Rimoldi, A.: *On algebraic and statistical properties of AES-like ciphers*. Ph.D. thesis, University of Trento, Italy, 2010.

[9] Murphy, S., Robshaw, M.: Essential algebraic structure within the AES. In Yung, M. (ed.), *Advances in Cryptology — CRYPTO 2002*, vol. 2442 of *Lecture Notes in Computer Science*, pp. 1–16. Springer Berlin Heidelberg, 2002.

[10] Bard, G. V.: *Algebraic cryptanalysis*. Springer, 2009.

[11] Knudsen, L. R., Robshaw, M.: *The block cipher companion*. Information Security and Cryptography. Springer Berlin Heidelberg, 2011.

[12] Budaghyan, L., Carlet, C., Pott, A.: New classes of almost bent and almost perfect nonlinear polynomials. In *Information Theory, IEEE Transactions*, vol. 52, pp. 1141–1152. Institute of Electrical and Electronics Engineers, 2006.

[13] Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. In Menezes, A., Vanstone, S. (eds.), *Advances in Cryptology-CRYPT0'90*, vol. 537 of *Lecture Notes in Computer Science*, pp. 2–21. Springer Berlin Heidelberg, 1991.

[14] Matsui, M.: Linear cryptanalysis method for DES cipher. In Helleseth, T. (ed.), *Advances in Cryptology — EUROCRYPT '93*, vol. 765 of *Lecture Notes in Computer Science*, pp. 386–397. Springer Berlin Heidelberg, 1994.

[15] Burnett, L.: *Heuristic optimization of Boolean functions and substitution boxes for cryptography*. Ph.D. thesis, Queensland University of Technology, Australia, 2005.

[16] Kazymyrov, O., Kazymyrova, V.: Algebraic aspects of the Russian hash standard GOST R 34.11-2012. *In Pre-proceedings of 2nd Workshop on Current Trends in Cryptology (CTCrypt 2013)*, pp. 160–176, 2013.

[17] Nyberg, K.: Perfect nonlinear S-boxes. In Davies, D. (ed.), *Advances in Cryptology - EUROCRYPT'91*, vol. 547 of *Lecture Notes in Computer Science*, pp. 378–386. Springer Berlin Heidelberg, 1991.

[18] Courtois, N., Pieprzyk, J.: Cryptanalysis of block ciphers with overdefined systems of equations. *Cryptology ePrint Archive, Report 2002/044*, 2002. `http://eprint.iacr.org/`.

[19] Ailan, W., Yunqiang, L., Xiaoyong, Z.: Analysis of corresponding structure of differential branch of MDS matrixes on finite field. In *Intelligent Networks and Intelligent Systems (ICINIS), 2010 3rd International Conference*, pp. 381–384. Institute of Electrical and Electronics Engineers, 2010.

[20] Nechvatal, J., et al.: Report on the development of the Advanced Encryption Standard (AES). *Electronic source,* 2000. `http://csrc.nist.gov/archive/aes/round2/r2report.pdf`.

[21] Kwon, D., Kim, J., Park, S., Sung, S., et al.: New block cipher: ARIA. In Lim, J.-I., Lee, D.-H. (eds.), *Information Security and Cryptology - ICISC 2003*, vol. 2971 of *Lecture Notes in Computer Science*, pp. 432–445. Springer Berlin Heidelberg, 2004.

[22] Oliynykov, R., Gorbenko, I., Dolgov, V., Ruzhentsev, V.: Results of Ukrainian national public cryptographic competition. In *Tatra Mountains Mathematical Publications*, vol. 47, pp. 99–113. Mathematical Institute of Slovak Academy of Sciences, 2010.

PAPER VI

STATE SPACE CRYPTANALYSIS OF THE MICKEY CIPHER *

Tor Helleseth Cees J.A. Jansen Oleksandr Kazymyrov
Alexander Kholosha

VI

*Helleseth, T., Jansen, C.J.A., Kazymyrov, O., Kholosha, A.: State space cryptanalysis of the MICKEY cipher. In *Information Theory and Applications Workshop (ITA)*, pp. 1–10. Institute of Electrical and Electronics Engineers (IEEE), 2013.

State Space Cryptanalysis of The MICKEY Cipher

Tor Helleseth[†] Cees J.A. Jansen[‡]
Oleksandr Kazymyrov[†] Alexander Kholosha[†]

[†] The Selmer Center, Department of Informatics
University of Bergen, Norway
{Tor.Helleseth, Oleksandr.Kazymyrov, Alexander.Kholosha}@ii.uib.no
[‡] Riebeeckstraat 10, 5684ej Best, The Netherlands
deltacrypto@onsmail.nl

Abstract

In this paper, we consider the key-stream generator MICKEY, whose internal state splits into two parts that are updated both linearly and nonlinearly while clocking the generator. These state update functions also depend on the internal state of the registers, which perform the so-called self-mutual control. We suggest several attack scenarios based on the reverse clocking of the generator and analysis of the acquired backward states tree. Furthermore, we show meet-in-the-middle attack can be applied while simultaneously allowing the generation of shifted key streams for different pairs of keys and initialization vectors. In practice, our theoretical results are verified by extensive computations.

Keywords: MICKEY stream cipher, self-mutual control, nonlinear shift register, backward states tree.

1. Introduction

Nonlinear feedback shift registers (NFSRs) prove to be an extremely promising building block for key-stream generators. Such registers allow efficient hardware implementation and provide nonlinearity so crucial to the security of such a generator. However, the behavior of such nonlinear components is poorly understood, which, in turn, results in a lack of criteria for selecting parameters that directly affect security. The reason for this poor understanding is the difficulty inherent in the analysis of generators involving NFSRs.

Despite widespread use of nonlinear registers in modern stream ciphers, security analysis of such constructions is mostly conjectural and lacks formal estimates and proofs.

Due to the specific application of symmetric ciphers, generators require a large period and linear complexity of key streams [1]. To achieve this, designers of stream ciphers often combine linear and nonlinear registers. It is believed that the linear part should guarantee the required period and the nonlinear part should increase linear complexity. Moreover, the update function of both registers may involve the state of the partner register and implement so-called 'mutual control'. In addition to a secret key, modern key-stream generators, often incorporate the public initialization value (IV), which allows the use of the same key for multiple encryptions. Prior to key-stream generation, the generator is clocked a number of times in a preclock phase and, provided that the key and IV are not set up directly into the register states, can also be clocked during key/IV-loading. Design principles for these modes differ, but in order to achieve the high efficiency in hardware implementation, the design of these modes are often similar.

The Mutual Irregular Clocking KEYstream generator (MICKEY) is an example of the generators described above. In article [2] Hong and Kim noted that the first version of MICKEY (MICKEY v1) is potentially vulnerable to time-memory-data (TMD) tradeoff attack [3, 4]. Based on this research, developers of the current (second) version of MICKEY (MICKEY v2) have modified the algorithm and shown that the new version of the cipher is resistant to this type of attack. A different part [2] deals with a comparative analysis of MICKEY v1's update function and random function as well as an estimation of collisions in a transition states graph. We are continuing the research in this direction and applying a similar technique to MICKEY v2. In addition to employing a method from [2], our approach allows us to theoretically calculate the states of whole backward states tree using update functions of registers. We have thus acquired theoretical results that give us an opportunity to verify data from [2] in different way. A procedure of generating a backward states tree results in an encryption key with a complexity lower than exhaustive search. We also present a new method for generating non-identical pairs of keys and IVs. These pairs allow us to generate the key streams that are shifted by predetermined bits. The theoretical results are also confirmed by practical calculations, and take into account all peculiarities of the most recent version of MICKEY.

The paper is organized as follows. Section 2 introduces the general model of the key-stream generator under analysis and underlines attack scenarios. The remaining part of the paper contains an illustration of our approach to stream cipher MICKEY v2 [5, 6], which is in the portfolio of hardware-oriented eStream ciphers [7]. We begin with the description of MICKEY in subsection 3.1. Further in subsections 3.2-3.4, we theoretically compute probabilities of branch points for all possible degrees in the state transition graph for MICKEY when run in the key-stream generation, preclock and key/IV-lode modes. The recorded values from practical verification of these results are presented in subsection 3.5.

2. Key-Stream Generation Model and Attack Scenarios

There is evidence to show that, in general, cryptanalysis of stream ciphers for recovering a key is a two-step process. The first step is to retrieve the initial state of registers based on encrypted data or key stream. However, the resultant state guarantees recovering messages from one session only, i.e. for one IV. In most cases, the complexity of this step is very high and often close to the exhaustive search. In order to gain complete control over the channel it is necessary to obtain a key. Therefore, an additional attack is performed at the next stage to recover a key based on the register values from the previous step. The complexity of this step is usually significantly less than the first one.

An example of such a stage is an attack on the cipher A5/1 [8], which is implemented in mobile systems of the second generation. The rainbow attack [9] is performed on the first stage and allows us to obtain the state after preclock mode. But in order to recover the key, a different type of attack was developed, a full description of which is provided in [3]. In this paper, we focus on the second stage. Thus we assume that the state of the registers is known.

We consider a particular design of a key-stream generator that consists of two registers (R and S) of length n_R and n_S with the affine ($A^I_{C_R}$) and nonlinear ($N^I_{C_S}$) update function respectively. We also assume that the concrete state update function of registers R and S applied at a certain stage is chosen according to the current state of R and S. This can be seen as self-mutual control of register clocking. To this end, we define vectorial Boolean functions f_R and f_S of $n_S + n_R$ bits each. The outputs of these functions, which in

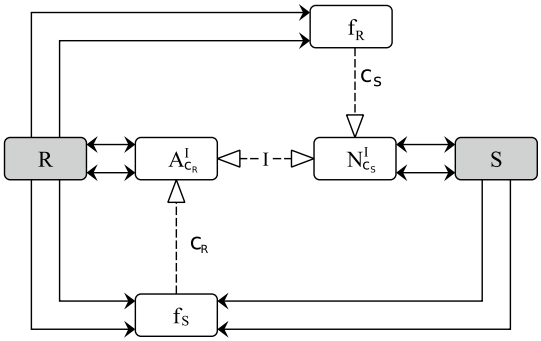


Fig. 1: *Block Model of a Self-Mutual Key Generator*

general consist of a few bits, are denoted by c_S and c_R. Figure 1 shows the design being analyzed in this paper. We also assume that the key and IV are inserted bit-by-bit into the key-stream generator run in the key/IV-load modes (meaning that this bit, denoted I in Figure 1, in some way affects the state update function). The generator is run in the following way: IV-load, key-load, preclock and key-stream generation. The initial state of the generator is always set to a constant.

The considered key-stream generator run in a key-stream generation mode can be seen as an autonomous finite state machine and, thus, the period of the produced output cannot exceed the state space size, i.e. $2^{n_R+n_S}$. However, even if both registers are nonsingular, the way they are updated results in self-mutual control which may cause them to behave in a singular fashion individually, exhibiting orphan states (states with no predecessors) and branch point states. This indicates a reduction in period (since in the maximum period, all states are connected into a full cycle that has no orphan states or branch points). Moreover, analysis of the transitions graph implies many other interesting properties crucial for security and helpful in cryptanalysis. In what follows, by the *degree* of a branch point we understand the number of incoming edges to the branching node in the state transition graph.

In the **first scenario**, assume that an attacker, in some way, knows the internal state of the key-stream generator at some stage during preclock or key-stream generation and knows exactly how many steps the generator was stepped to end up in this state. Then the generator is stepped back appropriately to stop right before preclock (i.e., right after key/IV-load). In

this process, the generator behaves as an autonomous finite state machine with a few options for the preceding state. It is necessary to consider how many candidate states we end up with. This number is equal to the number of leaves in the top level of the tree that represents state transitions (backwards states tree). Amazingly enough, some key-stream generators demonstrate only polynomial growth in the number of leaves contained in each level of the tree (as opposed to the exponential growth demonstrated, for instance, in the full binary tree). This allows us to perform computations stepping several hundred clocks backwards. Obviously, the crucial characteristic of the transition tree is the average branch number, which is defined by probabilities of branch points of different degrees. Clearly, if the expected value is close to 1, then the tree will grow much slower than 2^n.

In the **second scenario**, we assume that the attacker learns the internal state of the key-stream generator at some stage during key-load (preceding preclock) and knows exactly how many steps the generator was stepped to end up in this state. Here, while clocking the generator backwards, we have additional uncertainty in the key-bit that affects the state update function. Branch points in the corresponding tree have a higher degree, and edges are labeled with the appropriate value of the key-bit. Hence, elimination of orphan states can lead to the unique identification of key bits.

The second scenario can be extended as follows. Assume that the attacker learns the internal state of the key-stream generator after only a few steps after the generator has been run in the key-load mode. This number of steps should be small enough to roll back the generator to the beginning of the key-load (end of the IV-load). However, by knowing the initial state of the generator and the IV, it is possible to step the generator forward to the same point at the beginning of the key-load. Therefore, only one path in the tree will correspond to the real state update chain, and this will reveal the portion of the key involved. Furthermore, it is also possible to use a **meet-in-the-middle attack** attack in which a certain portion of initial key bits is checked using brute force and the following bits are recovered using backward clocking in the key-load. At the "meeting point", the generator should acquire the same internal state. This criterion is used for eliminating incorrect keys.

3. Application to MICKEY

The state space of MICKEY includes branch points and orphan states, which is a consequence of the self-mutual control used in the design of the cipher. Assuming a randomly chosen state, it is fairly straightforward to express the probabilities of this state being 1, but it gives no information about the distribution of the states in the whole graph. Therefore, the following four sections show how to find the fraction of the backward states tree based on the probability of branches appearing.

3.1. Brief Description of MICKEY

There are two versions of the cipher MICKEY-80 v2 and MICKEY-128 v2 [5, 6]. Each of them takes two input parameters: initialization vector (IV) and session key (K). The general architecture of the design is shown in Figure 2.

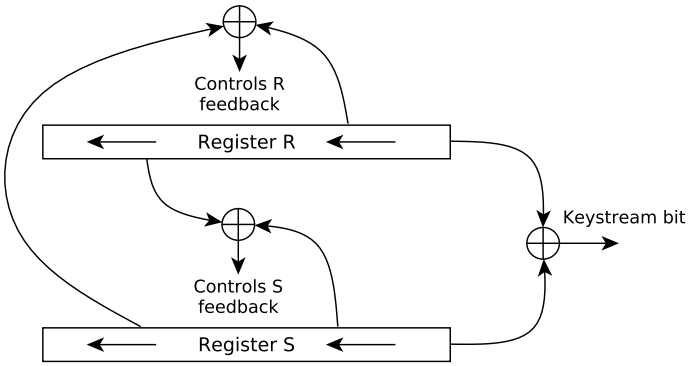


Fig. 2: *Clocking Architecture of the MICKEY Cipher*

Both versions are based on a combination of linear (R) and nonlinear (S) registers with length RL, which is shown in Table 1. Cells of registers have bit values denoted by $r_0, r_1, \ldots, r_{RL-1}$ and $s_0, s_1, \ldots, s_{RL-1}$. Both registers, R and S, are clocked in two different ways depending on the control bit $CB\_R = CB\_SL \oplus CB\_RR$ for R and $CB\_S = CB\_SR \oplus CB\_RL$ for S respectively, where CB_XY means a certain bit of the X register from Table 1 (for example, $CB\_SL = s_{34}$). The update function of registers has an additional

Table 1: *Parameters of the Ciphers MICKEY-80 v2 and MICKEY-128 v2*

Version	RL	Key length	Preclock length	CB_SL	CB_SR	CB_SM	CB_RL	CB_RR
80	100	80	100	34	67	50	33	67
128	160	128	160	54	106	80	53	106

input parameter the so-called input bit (IB_S and IB_R). The difference in parameters of MICKEY-80 v2 and MICKEY-128 v2 is described in Table 1. As can be seen from the table, the ratio of the register length to the other parameters is almost the same for both versions of MICKEY. Nonetheless, the state spaces for MICKEY-80 v2 and MICKEY-128 v2 differ significantly, as will be shown later.

According to the terminology accepted in section 2, vectorial Boolean functions for MICKEY are defined by $f_R = CB\_SR \oplus CB\_RL$, $f_S = CB\_SL \oplus CB\_RR$ and update functions by $A^I_{C_R} = CLOCK\_R$, $N^I_{C_S} = CLOCK\_S$, where $CLOCK\_R$, $CLOCK\_S$ are registers' R and S update functions for the cipher MICKEY respectively. The length of registers is identical and equals $n_s = n_r = RL$.

The cipher runs in the following way:

- initialise the registers R and S with all zeros;
- IV-load (this mode corresponds to $CLOCK\_K\_IV$ function);
- key-load ($CLOCK\_K\_IV$);
- preclock ($CLOCK\_PRECLOCK$);
- key-stream generation ($CLOCK\_KG$).

All modes except key-stream generation work in so-called mix mode. This means that the input bit of the register R depends on a certain bit of the register S, and is denoted by CB_SM. The update clocking function of the key generator ($CLOCK\_KG(R, S, MIXING, INPUT\_BIT)$, where $INPUT\_BIT$ is bit of key or IV) is shown in Algorithm 1.

As already noted in section 2, it is assumed that the adversary knows the state of the registers R and S. Our evaluation of the MICKEY cipher resistance is based on the construction of a backward states tree, as described in detail for A5/1 in [8]. A brief description of the tree construction with respect to the MICKEY cipher is presented below.

The previous states are computed using the functions $CLOCK\_PRECLOCK^{-1}$, $CLOCK\_K\_IV^{-1}$ and $CLOCK\_KG^{-1}$, which are inverse to the clock functions

Algorithm 1 CLOCK_KG

Input: registers R and S, MIXING and INPUT_BIT
Output: updated states of registers R and S
CB_R = CB_SL $\oplus$ CB_RR
CB_S = CB_SR $\oplus$ CB_RL
if MIXING = TRUE **then**
  IB_R = $INPUT\_BIT \oplus$ CB_SM;
else
  IB_R = $INPUT\_BIT$
end if
IB_S = $INPUT\_BIT$
CLOCK_R(R, IB_R, CB_R)
CLOCK_S(S, IB_S, CB_S)

of MICKEY. The algorithm for achieving reverse states results in an exhaustive search for all possible values of input parameters (input, control and feedback bits for both registers) and the elimination of states with impossible conditions. Algorithms of function $CLOCK\_X^{-1}(R, S)$ for MICKEY-80 v2 are given in Appendix A.

It is worth noting that the previous state is not always uniquely determined. The number of branches may vary, depending on the state and mode of the cipher. Three different backward states trees can be acquired for different modes. Any of these trees can be considered as a graph of state transitions of finite-state machines. The general structure of the tree is shown in Figure 3. Hereinafter we will refer to such concepts connected to a tree as: the level is the set of all backward states that may result in an original state after a certain number of clocks; the degree of branch is the number of possible previous states, which give R and S after one clock forward. In K/IV load mode, edges are labeled with the appropriate bit of the sought-for key.

3.2. Key-Stream Generation Mode

Let $\underline{\rho}_0$ and $\underline{\rho}_1$ denote the two predecessors of the R-register by applying the inverses of the two different transition matrices [10]. Similarly, let $\underline{\sigma}_0$ and $\underline{\sigma}_1$ denote the two predecessors of the S-register by applying the inverses of the two different nonlinear register update function. The register self-control is obtained from two taps of both registers. These taps' indices are denoted by c and a, and the corresponding predecessor state bits are denoted by $\rho_{0,c}$,

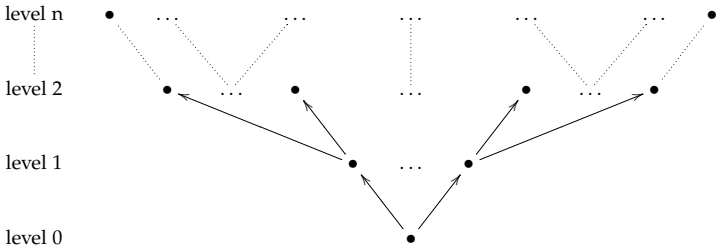


Fig. 3: *A Backward States Tree*

$\rho_{1,c}$, $\rho_{0,a}$, $\rho_{1,a}$, and $\sigma_{0,c}$, $\sigma_{1,c}$, $\sigma_{0,a}$, $\sigma_{1,a}$. The register control signals are denoted by $Rctrl_{ij}$ and $Sctrl_{ij}$ where ij is an index indicating the state pair $(\underline{\rho}_i, \underline{\sigma}_j)$ from which they are derived. From the definition of the MICKEY ciphers the following relations between the register control signals and the predecessor state bits are evident.

$$\begin{aligned}
(Rctrl_{00}, Sctrl_{00}) &= (\rho_{0,c} + \sigma_{0,a}, \rho_{0,a} + \sigma_{0,c}) \\
(Rctrl_{10}, Sctrl_{10}) &= (\rho_{1,c} + \sigma_{0,a}, \rho_{1,a} + \sigma_{0,c}) \\
(Rctrl_{01}, Sctrl_{01}) &= (\rho_{0,c} + \sigma_{1,a}, \rho_{0,a} + \sigma_{1,c}) \\
(Rctrl_{11}, Sctrl_{11}) &= (\rho_{1,c} + \sigma_{1,a}, \rho_{1,a} + \sigma_{1,c})
\end{aligned}$$

From the above equations it is easy to see that the 2^8 values for the 8 predecessor state bits result in 64 distinct values of the 8 *Rctrl* and *Sctrl* control signals. Out of these 64 combinations, 21 give rise to orphan states (BP0), 24 to regular states (BP1), 18 to states with 2 predecessors (BP2), and 1 to states with four predecessors (BP4). None of the 64 values gives rise to states with three predecessors (BP3).

Branch points with four predecessors have the following control signals: $Rctrl_{00} = 0$, $Sctrl_{00} = 0$, $Rctrl_{01} = 0$, $Sctrl_{01} = 1$, $Rctrl_{10} = 1$, $Sctrl_{10} = 0$, $Rctrl_{11} = 1$, and $Sctrl_{11} = 1$. These conditions result in the following six conditions on the values of the corresponding predecessor state bits.

$$\begin{aligned}
(\rho_{0,c} = \sigma_{0,a}) &\quad \wedge \quad (\rho_{0,a} = \sigma_{0,c}) \\
(\rho_{1,c} = 1 + \sigma_{0,a}) &\quad \wedge \quad (\rho_{1,a} = \sigma_{0,c}) \\
(\sigma_{0,a} = \sigma_{1,a}) &\quad \wedge \quad (\sigma_{0,c} = 1 + \sigma_{1,c})
\end{aligned}$$

If we assume a random R-register fill, then, due to the linearity of the R-register, the probabilities of all 16 combinations of the 4 ρ bits are equal to $\frac{1}{16}$. As branch points in the S-register can only occur if the most significant state bit has value 1, we arrive at the following expression for the probability of a branch point state with four predecessors.

$$\begin{aligned} Pr(BP4) &= \frac{1}{2} \cdot \frac{1}{16} Pr((\sigma_{0,a} = \sigma_{1,a}) \wedge \\ &\quad (\sigma_{0,c} = 1 + \sigma_{1,c})) \end{aligned} \tag{1}$$

Assuming a random S-register fill, the values are $Pr(BP4) = 0.00819, 0.00835, 0.00856, 0.00885$, for MICKEY-80 v1, -80 v2, -128 v1, -128 v2, respectively. These values have been calculated using a method described in Section 3 of [10].

In the same way the probability of other state types can be calculated. For orphan states (BP0) we find

$$\begin{aligned} Pr(BP0) = \tfrac{1}{2} \cdot \tfrac{1}{16} \cdot \\ (4Pr((\sigma_{0,a} = \sigma_{1,a}) \wedge (\sigma_{0,c} = \sigma_{1,c})) + \\ 9Pr((\sigma_{0,a} = \sigma_{1,a}) \wedge (\sigma_{0,c} = 1 + \sigma_{1,c})) + \\ 4Pr((\sigma_{0,a} = 1 + \sigma_{1,a}) \wedge (\sigma_{0,c} = \sigma_{1,c})) + \\ 4Pr((\sigma_{0,a} = 1 + \sigma_{1,a}) \wedge (\sigma_{0,c} = 1 + \sigma_{1,c}))). \end{aligned} \tag{2}$$

From expressions (1) and (2) one obtains

$$Pr(BP0) = \frac{1}{8} + 5Pr(BP4). \tag{3}$$

Carrying on in the same way, the probabilities of BP1 and BP2 are obtained

$$Pr(BP1) = \frac{1}{4} - 8Pr(BP4) \tag{4}$$

$$Pr(BP2) = \frac{1}{8} + 2Pr(BP4). \tag{5}$$

The sum of all probabilities is equal to one half, because we have considered only the S-register states with msb equal to 1. S-register states with msb equal to 0, give rise to singularities in the R-register only. Again due to the linearity of the R-register, the probabilities of the three kinds of R-register states (BP0,

BP1, and BP2) are equal to $\frac{1}{8}$, $\frac{1}{4}$, and $\frac{1}{8}$ respectively. The overall probabilities are given below.

$$Pr(BP0) = \frac{1}{4} + 5Pr(BP4) \quad (6)$$

$$Pr(BP1) = \frac{1}{2} - 8Pr(BP4) \quad (7)$$

$$Pr(BP2) = \frac{1}{4} + 2Pr(BP4) \quad (8)$$

From the above expression for the state probabilities, it is seen that the average number of predecessor states equals 1, regardless of $Pr(BP4)$. The variance is given by $Var = \frac{1}{2} + 16Pr(BP4)$, which is less than one (≈ 0.63) for all MICKEY versions.

3.3. Preclock Mode

In preclock mode the MICKEY cipher has an additional modifier signal from the tap with index $N/2$ of the nonlinear S-register to the input of the R-register. In line with Section 3.2, we write $\sigma_{0,N/2}$ and $\sigma_{0,N/2}$ for the bits of $\underline{\sigma}_0$ and $\underline{\sigma}_1$ with index $N/2$. Consequently, this S-register tap modifies the R-register by xor-ing the tap value to the feedback bit of the R-register. Therefore, if there is a 1 value at the appropriate locations c or a of the feedback vector of the R-register, this will modify the control tap values of the R-register, thereby possibly changing the branch point conditions. It turns out that, depending only on the value of the feedback vector bits of the R-register, the cycle structure of preclock mode is either equivalent to that of the key-stream generation mode, or its cycle structure changes to include branch points with three predecessors having different probabilities of occurrence from the key-stream generation mode. The latter case is dealt with in this section. It should be remarked that MICKEY-80 v2 is the only version having preclock mode cycle structures equivalent to the key-stream generation mode, all other versions have exhibiting branch points with three predecessors.

Omitting the details of the calculations, the following results apply

$$Pr(BP4) = \tfrac{1}{2} \cdot \tfrac{1}{16} Pr((\sigma_{0,a} = \sigma_{1,a}) \wedge (\sigma_{0,c} = 1 + \sigma_{1,c}) \wedge (\sigma_{0,N/2} = \sigma_{1,N/2})) \quad (9)$$

$$Pr(BP3) = \tfrac{1}{2} \cdot \tfrac{1}{16} Pr((\sigma_{0,a} = \sigma_{1,a}) \wedge (\sigma_{0,N/2} = 1 + \sigma_{1,N/2})). \quad (10)$$

Again assuming random register fills, the values are $Pr(BP4) \approx 0.0027$ and $Pr(BP3) \approx 0.019$. Note from (9) that $Pr(BP4)$ in preclock mode is less than in the key-stream generation mode. As with the key-stream generation mode, the probabilities of the other state types can be expressed in $Pr(BP4)$ and $Pr(BP3)$, resulting in the expressions below.

$$Pr(BP0) = \tfrac{1}{4} + Pr(BP3) + 5Pr(BP4) \tag{11}$$

$$Pr(BP1) = \tfrac{1}{2} - Pr(BP3) - 8Pr(BP4) \tag{12}$$

$$Pr(BP2) = \tfrac{1}{4} - Pr(BP3) + 2Pr(BP4) \tag{13}$$

From the above expressions for the state probabilities, it is again seen that the average number of predecessor states equals 1, regardless of $Pr(BP4)$ and $Pr(BP3)$. The variance is given by $Var = \frac{1}{2} + 4Pr(BP3) + 16Pr(BP4)$, which is less than one for all MICKEY versions.

3.4. KEY/IV LOAD MODE

Loading key and IV bits into the MICKEY ciphers is realized in mix mode, by shifting in key and IV bits in both registers in parallel. The structure of the state space in this mode can be adapted to include the uncertainty about the values of the key bits, when attempting to step backwards in the state space. This adaptation consists of taking into account branch points arising from one or more states leading to one next state when using two different values for the key bit input. In general, the number of possible predecessors doubles in this model, ranging from 0 (orphan states) through 8 (branch points with four predecessors for both values of the key bit). We will refer to this model as the compound state space. The compound state space can be viewed as the union of the state spaces for all combinations of key bit values. For the MICKEY ciphers there are two different compound state spaces, in a similar way as explained for preclock mode. In this respect, MICKEY-80 v2 is different from the other published versions. For example, MICKEY-80 v2 has no branch points with five and seven predecessors.

Let us denote the probability that a randomly chosen state has i predecessors for a key bit of value k and j predecessors for a key bit of value $k \oplus 1$ by P_{ij}. Then from symmetry arguments, $P_{ij} = P_{ji}$. For MICKEY-80 v2, $P_{04} = 0$, $P_{14} = 0$, and $P_{12} = P_{01}$. the probabilities of all nonzero state types can be

Table 2: *State type probabilities for the compound state space model of the MICKEY-80 v2 in Key/IV Load Mode. State type n-m means a state with n predecessor states for a key bit with value 0, and with m predecessor states for a key bit with value 1.*

State type	80 v2
0	0.32
1	$1.1 \cdot 10^{-4}$
2	0.39
3	$1.1 \cdot 10^{-4}$
4	0.28
6	$1.1 \cdot 10^{-5}$
8	0.014

expressed in four of them, i.e. P_{02}, P_{22}, P_{24}, and P_{44}, obtaining the expressions below for states with nonzero probabilities.

$$P_{00} = P_{22} + 4P_{24} + 3P_{44} \quad (14)$$

$$P_{01} = \frac{1}{4} - P_{02} - P_{22} + P_{24} + 2P_{44} \quad (15)$$

$$P_{11} = 2P_{02} + 2P_{22} - 10P_{24} - 12P_{44} \quad (16)$$

$$(17)$$

The probabilities of the various state types in both models have been determined by calculations and verified by experiments. The results are given in Table 2, showing the differences in the two compound state spaces. The average number of predecessors is two in both cases, one for each key bit value. The variances are significantly different: $Var \approx 2.88$ for MICKEY-80 v2, and $Var \approx 0.89$ for the other MICKEY ciphers., indicating that stepping backwards is harder for MICKEY-80 v2 than for the other MICKEY ciphers.

Of interest is the probability of a state being an orphan for one key bit value and a state with one or more predecessors for the complement of the key bit value. In this case, the uncertainty in the key bit value is resolved, and therefore, the total uncertainty in the key is reduced by one bit. In the case of MICKEY-80 v2, however, this probability is very small ($\approx 3 \cdot 10^{-4}$).

3.5. Computational Results

The tree construction method, described in section 3.1, allows to evaluate following properties of a finite state machine.

3.5.1. Degree Probability.

The backward tree was constructed by using functions $CLOCK\_X^{-1}$ (Appendix A). Probabilities of degrees were calculated with accuracy limited by 18-level tree for random and real values of registers R and S. Tables 3 and 4 show the difference in the degree probability distribution for different initial values and modes.

Table 3: *The Degree Probability for Random Initial States.*

Degree	Key/IV load		Preclock		KG	
	80 v2	128 v2	80 v2	128 v2	80 v2	128 v2
0	0.2982	0.198	0.2802	0.2825	0.3014	0.2718
1	0.00009	0.1031	0.4377	0.459	0.4052	0.4281
2	0.4229	0.4022	0.2735	0.2294	0.2844	0.29
3	0.0001	0.1087	-	0.0256	-	-
4	0.2698	0.1703	0.0085	0.0035	0.0090	0.0101
6	0.00001	0.0177	-	-	-	-
8	0.0089	-	-	-	-	-

Table 4: *The Degree Probability for Real Initial States.*

Degree	Key/IV load		Preclock mode		KG	
	80 v2	128 v2	80 v2	128 v2	80 v2	128 v2
0	0.2773	0.2186	0.3052	0.29	0.3041	0.3038
1	0.00001	0.1047	0.4345	0.4534	0.4323	0.4154
2	0.4331	0.3753	0.2523	0.2256	0.2558	0.2698
3	0.00002	0.1029	-	0.0289	-	-
4	0.28	0.1783	0.008	0.0021	0.0079	0.0111
6	0.00007	0.0203	-	-	-	-
8	0.0095	-	-	-	-	-

The results for real and random points approximately coincide with the theoretical ones from Table 2. Thus, the degree determination method described in subsections 3.2-3.4 could be applied to a stream cipher with the structure given in section 2 at the designing stage.

In key/IV load mode the expectation value of branch points degree for all versions approximately equals 2. Appropriate value for preclock and KG mode is approximately equal to 1. Thus, no matter in what mode the values of the registers were obtained. It is always possible to perform reverse steps and acquire the state after key initialization function.

Table 5 shows the average number of possible states at each level. For reducing the dependency on the initial state 1000 transformations with random values was performed. The results for the other modes are given in Appendix B. Clearly, the number of states increases in accordance with the expectation value.

Table 5: *The Number of Backward States Depending on the Level of a Tree in the Key Load Mode.*

Level	Number of states	
	80 v2	128 v2
0	1	1
1	3	3
2	9	7
3	25	18
4	45	39
5	143	82
6	247	171
7	523	347
8	1183	703
9	2221	1435
10	5075	2904
11	9453	5849
12	18694	11834
13	37702	23801
14	70675	47759
15	136867	95716

3.5.2. Determination of Key Bits Based on a Backward States Tree.

Each reverse step increases the probability of subtree cutting off with all previous states. This property exists since there is a high probability of orphan states (see Table 4). Therefore a key bit could be found uniquely. An example of that situation is shown in Table 6.

For MICKEY-128 v2 the key bit can be uniquely determined at levels 1 and 3 ("1" and "0" bits respectively). However, knowing this information

Table 6: *The Probability Distribution of Key Bits on a 5-level Backward Tree.*

Level	Probability			
	80 v2		128 v2	
	1	0	1	0
1	0.5	0.5	1	0
2	0.5	0.5	0.5	0.5
3	0.5	0.5	0	1
4	0.5	0.5	0.5	0.5
5	0.4857	0.5143	0.5	0.5

does not allow attacker to definitively find the backward state. Therefore, the knowledge of key bit value does not reduce the tree of states and the complexity of determining all bits of the key. Moreover, an opportunity of finding the key bit directly depends on an initial state. For instance, for MICKEY-80 v2 it is impossible to determine the key bits with probability 1.

3.5.3. Identical Key-Streams for Different Key/IV pairs.

Functions used for different modes of the MICKEY cipher allow to generate key-streams shifted by some bits for different pairs of key and IV.

Let z_i^h be i^{th} bit of a key-stream for h^{th} pair of (K_h, IV_h). Suppose also that $K_1 = \{k_0, k_1, \ldots, k_{n-1}\}$, where n the length of the key (Table 1). Then it is possible to find such (K_1, IV_1) and (K_2, IV_2) for which the states of registers will differ by one clock and the key-streams have the property $z_i^2 = z_{i+1}^1$.

Assume that $IV_1 = \{iv_0, iv_1, \ldots, iv_j\}$, $IV_2 = \{iv_0, iv_1, \ldots, iv_j, k_0\}$ and

$$K_2 = K_1 << 1 = \{k_1, k_2, \ldots, k_{n-1}, 0\}. \tag{18}$$

The relative placement of bits for various sets of K/IV is shown in Table 7.

Table 7: *Differences between parameters for various sets of key and IV*

IV_1				K_1						Preclock					Key-Stream Generation				
iv_0	iv_1	…	iv_j	k_0	k_1	k_2	…	k_{n-2}	k_{n-1}	0	0	0	…	0	0	0	0	0	…
iv_0	iv_1	…	iv_j	k_0	k_1	k_2	…	k_{n-2}	k_{n-1}	0	0	0	…	0	0	0	0	0	…
IV_2				K_2						Preclock					Key-Stream Generation				

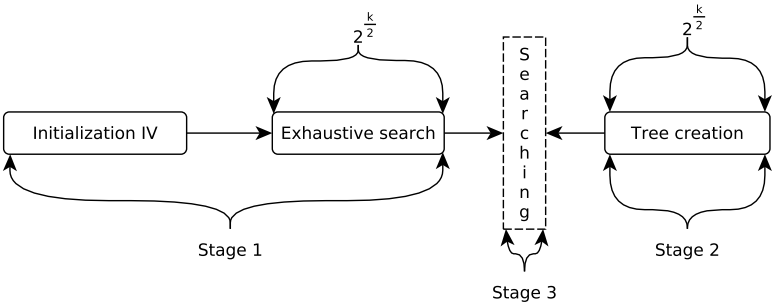


Fig. 4: *Meet-in-the-middle attack on the cipher MICKEY*

Obviously, the state of registers will differ only by one step. Therefore, the key-stream will have the same properties, i.e. $z_i^2 = z_{i+1}^1$. Since preclock mode is equivalent to key/IV mode, when input bit is 0, then the MICKEY cipher has shifting feature. Only one condition is a necessity: the s_{50}^1 bit must equal 0 at the moment of changing between preclock and key-stream generation modes.

The quantity of IV_2 bits can be increased by different value. As a result, K_2 and a key-stream will be shifted to the same amount of bits.

Similar arguments can be applied to 128-bit version of MICKEY. Examples of (K_1, IV_1) and (K_2, IV_2) with the described property for both versions of the cipher are given in Appendix C.

3.5.4. Meet-in-The-Middle Attack on Cipher MICKEY.

The extension of the scenario 2 described in section 2 allows to apply "meet-in-the-middle" attack, which reduces the complexity of finding the key in significant way. Suppose the state of the registers after k-clocks from IV initialization be known. Clearly, k-clocks correspond to k bits of the unknown key. The general scheme with 3-stages of the attack is shown in Fig. 4.

At the first stage calculate input state for initialization key function using the IV value. For obtained state find all possible states for $\frac{k}{2}$ key bits, applying $CLOCK\_K\_IV$ function. At the second stage build the backward states tree up to $\frac{k}{2}$ level using $CLOCK\_K\_IV^{-1}$. Finally, find coincidences in states acquired at the previous stages.

The complexity of exhaustive search attack or building of backward states tree is approximately the same and equals 2^k. Combination of these two methods in meet-in-the-middle attack makes it more practical and leads to the complexity:

$$O_d(k) + O_i(k) + O_f(k) = 2^{\frac{k}{2}} + 2^{\frac{k}{2}} + 2^{\frac{k}{2}} \approx 2^{\frac{k}{2}+2}$$

where $O_d(k)$ is the complexity of exhaustive search for $\frac{k}{2}$ key bits, $O_i(k)$ is the complexity of the backward states tree construction and $O_f(k)$ is the search complexity using hash tables [11].

4. Conclusions

In this article the analysis of self-controlled key-stream generators based on the MICKEY cipher was made. Stepping backwards in the state space of the cipher is possible and feasible in all modes including key/IV load mode. In mix mode the overall structure of the state space depends only on one or two bits in the feedback vector of the linear *R*-register.

From our analysis we conclude that the 128 bit versions are equally strong as the 80 bit version 2 cipher, and that the first 80 bit version is substantially weaker with an effective key length of 50 bits. This difference could have easily been avoided by a minor change in the feedback vector of the *R*-register. It is unknown to the authors if the MICKEY designers were aware of this fact when designing the MICKEY ciphers.

Proposed method allows to estimate degrees' probability at the design stage of MICKEY-like ciphers, and this was proved by practical results. Thus, it is possible to justify the choice of the encryption algorithm parameters.

References

[1] Rueppel, R. A.: *Analysis and Design of Stream Ciphers*. Communications and Control Engineering Series. Springer Berlin Heidelberg, 1986.

[2] Hong, J., Kim, W.-H.: TMD-tradeoff and state entropy loss considerations of streamcipher MICKEY. In Maitra, S., Veni Madhavan, C., Venkatesan, R. (eds.), *Progress in Cryptology - INDOCRYPT 2005*, vol. 3797 of *Lecture Notes in Computer Science*, pp. 169–182. Springer Berlin Heidelberg, 2005.

[3] Biryukov, A., Shamir, A.: Cryptanalytic time/memory/data tradeoffs for stream ciphers. In Okamoto, T. (ed.), *Advances in Cryptology — ASIACRYPT 2000*, vol. 1976 of *Lecture Notes in Computer Science*, pp. 1–13. Springer Berlin Heidelberg, 2000.

[4] Biryukov, A., Shamir, A., Wagner, D.: Real time cryptanalysis of A5/1 on a PC. In Goos, G., Hartmanis, J., van Leeuwen, J., Schneier, B. (eds.), *Fast Software Encryption*, vol. 1978 of *Lecture Notes in Computer Science*, pp. 1–18. Springer Berlin Heidelberg, 2001.

[5] Babbage, S., Dodd, M.: The stream cipher MICKEY 2.0. *A eSTREAM Portfolio Stream Cipher,* 2006. http://www.ecrypt.eu.org/stream/p3ciphers/mickey/mickey_p3.pdf.

[6] Babbage, S., Dodd, M.: The stream cipher MICKEY-128 2.0. *A eSTREAM Portfolio Stream Cipher,* 2006. http://www.ecrypt.eu.org/stream/p2ciphers/mickey128/mickey128_p2.pdf.

[7] Robshaw, M. J. B., Billet, O. (eds.): *New stream cipher designs - The eSTREAM finalists*, vol. 4986 of *Lecture Notes in Computer Science*. Springer, 2008.

[8] Golić, J.: Cryptanalysis of alleged A5 stream cipher. In Fumy, W. (ed.), *Advances in Cryptology — EUROCRYPT '97*, vol. 1233 of *Lecture Notes in Computer Science*, pp. 239–255. Springer Berlin Heidelberg, 1997.

[9] Stevenson, F.: A5/1 decryption. *Electronic source,* 2013. https://opensource.srlabs.de/projects/a51-decrypt/wiki/Wiki/history.

[10] Jansen, C.: The state space structure of the MICKEY stream cipher. *Proceedings of the 32rd WIC Symposium on Information Theory in the Benelux and The 1st Joint WIC/IEEE Symposium on Information Theory and Signal Processing in the Benelux,* 2011.

[11] Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C., et al.: *Introduction to algorithms*, vol. 2. MIT press Cambridge, 2001.

A. Description of Inverse Functions

A.1. Algorithm of CLOCK_S^{-1}

The algorithm for finding previous states of S register of cipher MICKEY-80 v2.

Algorithm 2 CLOCK_S^{-1}

Input: bits IB_S, CB_S, FB_S and register S
Output: TRUE if branch exist, otherwise FALSE
$S' \leftarrow S$
if FB_S = TRUE **then**
  if CB_S = TRUE **then**
    for i=0 to 99 **do**
      $\hat{s_i} \leftarrow s'_i \oplus \text{FB1}_i$;
    end for
  else
    for i=0 to 99 **do**
      $\hat{s_i} \leftarrow s'_i \oplus \text{FB0}_i$;
    end for
  end if
else
  for i=0 to 99 **do**
    $\hat{s_i} \leftarrow s'_i$;
  end for
end if
if $\hat{s_0} \neq 0$ **then**
  return FALSE
end if
$s_{99} \leftarrow \text{FB\_S} \oplus \text{IB\_S}$;
$s_{98} \leftarrow \hat{s}_{99}$;
for i=98 to 1 **do**
  $s_{i-1} \leftarrow \hat{s_i} \oplus ((s_i \oplus COMP0_i) \wedge (s_{i+1} \oplus COMP1_i))$;
end for
return TRUE

A.2. ALGORITHM OF CLOCK_R^{-1}

The algorithm for finding previous states of R register of cipher MICKEY-80 v2.

Algorithm 3 CLOCK_R^{-1}

Input: bits IB_R, CB_R, FB_R and register R
Output: TRUE if brach exist, otherwise FALSE
R$'$ $\leftarrow$ R
if CB_R = TRUE **then**
  $r_0 \leftarrow r_0 \oplus$ (FB_R $\wedge$ RTAPS$_0$);
  for i=1 to 99 **do**
    $r_i \leftarrow r_i \oplus r_{i-1} \oplus$ (FB_R $\wedge$ RTAPS$_i$);
  end for
  if $r_{99} \neq$ (FB_R $\oplus$ IB_R) **then**
    return FALSE
  end if
else
  if ($r_0 \oplus$ (FB_R $\wedge$ RTAPS$_0$)) $\neq 0$ **then**
    return FALSE
  end if
  for i=1 to 99 **do**
    $r_{i-1} \leftarrow r_i \oplus$ (FB_R $\wedge$ $RTAPS_i$);
  end for
  $r_{99} \leftarrow$ FB_R $\oplus$ IB_R;
end if
return TRUE

A.3. ALGORITHM OF CLOCK_K_IV^{-1}

The algorithm for finding previous states in the K/IV mode of cipher MICKEY-80 v2.

Algorithm 4 CLOCK_K_IV^{-1}

Input: registers R and S
Output: all possible branches
branches $\leftarrow$ Ø
for $s = 0$ to 64 **do**
  FB_S $\leftarrow (s \gg 0) \wedge 1$;
  IB_S $\leftarrow (s \gg 1) \wedge 1$;
  CB_S $\leftarrow (s \gg 2) \wedge 1$;
  FB_R $\leftarrow (s \gg 3) \wedge 1$;
  IB_R $\leftarrow (s \gg 4) \wedge 1$;
  CB_R $\leftarrow (s \gg 5) \wedge 1$;
  TS $\leftarrow$ S;
  TR $\leftarrow$ R;
  if CLOCK_S^{-1}(TS,IB_S,CB_S,FB_S) $\neq$ TRUE **then**
    continue;
  end if
  if CLOCK_R^{-1}(TR,IB_R,CB_R,FB_R) $\neq$ TRUE **then**
    continue;
  end if
  INPUT_BIT_S $\leftarrow$ IB_S;
  INPUT_BIT_R $\leftarrow$ IB_S $\oplus s_{50}$;
  CONTROL_BIT_R $\leftarrow s_{34} \oplus r_{67}$;
  CONTROL_BIT_S $\leftarrow s_{67} \oplus s_{33}$;
  if CONTROL_BIT_R = CB_R **and** CONTROL_BIT_S = CB_S **and** INPUT_BIT_R = IB_R **then**
    branches $\leftarrow$ Append(branches,[TS;TR;INPUT_BIT_S]);
  end if
end for
return branches

A.4. ALGORITHM OF CLOCK_PRECLOCK^{-1}

The algorithm for finding previous states in the preclock mode of cipher MICKEY-80 v2.

Algorithm 5 CLOCK_PRECLOCK^{-1}

Input: registers R and S
Output: all possible branches
branches $\leftarrow \emptyset$
for $s = 0$ to 32 **do**
  FB_S $\leftarrow (s \gg 0) \wedge 1$;
  CB_S $\leftarrow (s \gg 1) \wedge 1$;
  FB_R $\leftarrow (s \gg 2) \wedge 1$;
  IB_R $\leftarrow (s \gg 3) \wedge 1$;
  CB_R $\leftarrow (s \gg 4) \wedge 1$;
  TS $\leftarrow$ S;
  TR $\leftarrow$ R;
  if CLOCK_S^{-1}(TS,0,CB_S,FB_S) $\neq$ TRUE **then**
    continue;
  end if
  if CLOCK_R^{-1}(TR,IB_R,CB_R,FB_R) $\neq$ TRUE **then**
    continue;
  end if
  INPUT_BIT_R $\leftarrow s_{50}$;
  CONTROL_BIT_R $\leftarrow s_{34} \oplus r_{67}$;
  CONTROL_BIT_S $\leftarrow s_{67} \oplus s_{33}$;
  if CONTROL_BIT_R = CB_R **and** CONTROL_BIT_S = CB_S **and** INPUT_BIT_R = IB_R **then**
    branches $\leftarrow$ Append(branches,[TS;TR;0]);
  end if
end for
return branches

A.5. ALGORITHM OF CLOCK_KG^{-1}

The algorithm for finding previous states in the key-stream generating mode of cipher MICKEY-80 v2.

Algorithm 6 CLOCK_KG^{-1}

Input: registers R and S
Output: all possible branches
branches $\leftarrow \emptyset$
for $s = 0$ to 16 **do**
  FB_S $\leftarrow (s \gg 0) \wedge 1$;
  CB_S $\leftarrow (s \gg 1) \wedge 1$;
  FB_R $\leftarrow (s \gg 2) \wedge 1$;
  CB_R $\leftarrow (s \gg 3) \wedge 1$;
  TS $\leftarrow$ S;
  TR $\leftarrow$ R;
  if CLOCK_S^{-1}(TS,0,CB_S,FB_S) $\neq$ TRUE **then**
    continue;
  end if
  if CLOCK_R^{-1}(TR,0,CB_R,FB_R) $\neq$ TRUE **then**
    continue;
  end if
  CONTROL_BIT_R $\leftarrow s_{34} \oplus r_{67}$;
  CONTROL_BIT_S $\leftarrow s_{67} \oplus s_{33}$;
  if CONTROL_BIT_R = CB_R **and** CONTROL_BIT_S = CB_S **then**
    branches $\leftarrow$ Append(branches,[TS;TR;0]);
  end if
end for
return branches

B. The Number of States on Each Level of The Backward Tree

The following tables show the dynamics of increasing the number of states depending on the quantity of steps taken backward.

Table 8: *The Number of Backward States Depending on the Level of a Tree in the Preclock mode.*

Level	Number of states	
	80 v2	128 v2
0	1	1
1	1	1
2	1	2
3	2	3
4	3	4
5	4	6
6	5	8
7	7	8
8	5	11
9	5	14
10	10	15
11	12	15
12	13	15
...	...	...
99	25	30
100	32	27
...	...	...
159	-	59
160	-	63

C. Example of Key-Streams With Different Length of IV

In appendix examples of identical key-streams for different IV length are described. All values are presented as bytes in hexadecimal notation except the first and the last values of Z_i, which are bits.

Table 9: *The Number of Backward States Depending on the Level of Tree in the Key-stream Generation Mode.*

Level	Number of states	
	80 v2	128 v2
0	1	1
1	1	2
2	1	3
3	2	4
4	3	6
5	3	6
6	4	4
7	5	3
8	3	4
...	...	...
125	17	64
126	17	81
127	16	91
128	20	97

C.1. MICKEY-80 v2

$$
\begin{aligned}
K_1 &= \{d3, ec, f0, 84, 8a, 1d, b1, b7, 4a, dd\} \\
IV_1 &= \{58, e5, 77, 0a, 9c, a2, 34, c7, cd, 5e\} \text{ (79 bits)} \\
K_2 &= \{a7, d9, e1, 09, 14, 3b, 63, 6e, 95, ba\} \\
IV_2 &= \{58, e5, 77, 0a, 9c, a2, 34, c7, cd, 5f\} \text{ (80 bits)} \\
Z_1 &= \{0, B7, 61, 27, 92, C5, 85, 91, 51, 18, 2A, D6, 7C, 8C, C8, \\
&\quad C7, 04\} \\
Z_2 &= \{B7, 61, 27, 92, C5, 85, 91, 51, 18, 2A, D6, 7C, 8C, C8, \\
&\quad C7, 04, 1\}
\end{aligned}
$$

C.2. MICKEY-128 v2

$$K_1 = \{c9, 55, e7, 7a, 80, 13, 1a, ad, 40, 45, d9, 6c, 71, 04, 97, 9c\}$$

$$
\begin{aligned}
IV_1 &= \{4e, db, 6e, 01, 05, 98, 2b, 30, c3, 56, 5a, ed, 80, 85, \\
&\quad 18, aa\} \text{ (127 bits)} \\
K_2 &= \{92, ab, ce, f5, 00, 26, 35, 5a, 80, 8b, b2, d8, e2, 09, 2f, 38\} \\
IV_2 &= \{4e, db, 6e, 01, 05, 98, 2b, 30, c3, 56, 5a, ed, 80, 85, \\
&\quad 18, ab\} \text{ (128 bits)} \\
Z_1 &= \{0, 79, 23, 91, 05, E1, DD, 2D, 9D, 83, 3E, B4, 78, 52, \\
&\quad E5, A6, 66\} \\
Z_2 &= \{79, 23, 91, 05, E1, DD, 2D, 9D, 83, 3E, B4, 78, 52, \\
&\quad E5, A6, 66, 1\}
\end{aligned}
$$

PAPER VII

VERIFICATION OF RESTRICTED EA-EQUIVALENCE FOR VECTORIAL BOOLEAN FUNCTIONS *

Lilya Budaghyan Oleksandr Kazymyrov

*Budaghyan, L., Kazymyrov, O.: Verification of restricted EA-equivalence for vectorial Boolean functions. In Özbudak, F., Rodríguez-Henríquez, F. (eds.), *Arithmetic of Finite Fields*, vol. 7369 of *Lecture Notes in Computer Science*, pp. 108–118. Springer Berlin Heidelberg, 2012.

Verification of Restricted EA-equivalence for Vectorial Boolean Functions

Lilya Budaghyan Oleksandr Kazymyrov

Department of Informatics,
University of Bergen, Norway
{Lilya.Budaghyan,Oleksandr.Kazymyrov}@uib.no

Abstract

We present algorithms for solving the restricted extended affine equivalence (REA-equivalence) problem for any m-dimensional vectorial Boolean functions in n variables. The best of them has complexity $O(2^{2n+1})$ for REA-equivalence $F(x) = M_1 \cdot G(x \oplus V_2) \oplus M_3 \cdot x \oplus V_1$. The algorithms are compared with previous effective algorithms for solving the linear and the affine equivalence problem for permutations by Biryukov et. al [1].

Keywords: EA-equivalence, matrix representation, S-box, vectorial Boolean function.

1. Introduction

Vectorial Boolean functions play very important role in providing high-level security for modern ciphers. They are used in cryptography as nonlinear combining or filtering functions in the pseudo-random generators (stream ciphers) and as substitution boxes (S-boxes) providing confusion in block ciphers. Up to date an important question of generation of vectorial Boolean functions with optimal characteristics to prevent all known types of attacks remains open. Sometimes equivalence (i.e. EA or CCZ) is used for achieving necessary properties without losing other ones (i.e. δ-uniformity, nonlinearity) [2].

But very often inverse problem occurs: it is needed to check several functions for equivalence. For instance, when finding a new vectorial Boolean function it is necessary to verify whether it is equivalent to already known ones as it happens with some of block ciphers, where several substitutions are used, (i.e. ARIA [3] or Kalyna [4, 5]). The complexity of exhaustive search for checking EA-equivalence for functions from $\mathbb{F}_2^n$ to itself equals $O\left(2^{3n^2+2n}\right)$.

Table 1: *Best Complexities for Solving REA-equivalence Problem*

Restricted EA-equivalence	Complexity	m	$G(x)$	Source
$F(x) = M_1 \cdot G(M_2 \cdot x)$	$O\left(n^2 \cdot 2^n\right)$	$m = n$	Permutation	[1]
$F(x) = M_1 \cdot G(M_2 \cdot x \oplus V_2) \oplus V_1$	$O\left(n \cdot 2^{2n}\right)$	$m = n$	Permutation	[1]
$F(x) = M_1 \cdot G(x \oplus V_2) \oplus V_1$	$O\left(2^{2n+1}\right)$	$m \geq 1$	†	Sec. 3
$F(x) = M_1 \cdot G(x \oplus V_2) \oplus V_1$	$O\left(m \cdot 2^{3n}\right)$	$m \geq 1$	Arbitrary	Sec. 3
$F(x) = G(M_2 \cdot x \oplus V_2) \oplus V_1$	$O\left(n \cdot 2^m\right)$	$m \geq 1$	Permutation	Sec. 3
$F(x) = G(x \oplus V_2) \oplus M_3 \cdot x \oplus V_1$	$O\left(n \cdot 2^n\right)$	$m \geq 1$	Arbitrary	Sec. 3
$F(x) = M_1 \cdot G(x \oplus V_2) \oplus M_3 \cdot x \oplus V_1$	$O\left(2^{2n+1}\right)$	$m \geq 1$	‡	Sec. 3
$F(x) = M_1 \cdot G(x \oplus V_2) \oplus M_3 \cdot x \oplus V_1$	$O\left(m \cdot 2^{3n}\right)$	$m \geq 1$	Arbitrary	Sec. 3

† - G is under condition $\{2^i \mid 0 \leq i \leq m-1\} \subset \mathrm{img}(G')$ where $G'(x) = G(x) + G(0)$.
‡ - G is under condition $\{2^i \mid 0 \leq i \leq m-1\} \subset \mathrm{img}(G')$ where G' is defined as (4).

When $n = 6$ the complexity is already 2^{120} that makes it impossible to perform exhaustive computing.

In the paper [1] Alex Biryukov et al. have shown that in case when given functions are permutations of $\mathbb{F}_2^n$, the complexity of determining REA-equivalence equals $O\left(n^2 \cdot 2^n\right)$ for the case of linear equivalence and $O\left(n \cdot 2^{2n}\right)$ for affine equivalence. In this paper we consider more general cases of REA-equivalence for functions from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ and specify results, when complexity can be reduced to polynomial. The complexities of our algorithms and the best previous known ones are given in Table 1.

2. PRELIMINARIES

For any positive integers n and m, a function F from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ is called *differentially δ-uniform* if for every $a \in \mathbb{F}_2^n \setminus \{0\}$ and every $b \in \mathbb{F}_2^m$, the equation $F(x) + F(x+a) = b$ admits at most δ solutions [6]. Vectorial Boolean functions used as S-boxes in block ciphers must have low differential uniformity to allow high resistance to differential cryptanalysis (see [7]). In the important case when $n = m$, differentially 2-uniform functions, called *almost perfect nonlinear* (APN), are optimal (since for any function $\delta \geq 2$). The notion of APN function is closely connected to the notion of *almost bent* (AB) function

[8] which can be described in terms of the *Walsh transform* of a function $F : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$

$$\lambda(u,v) = \sum_{x \in \mathbb{F}_2^n} (-1)^{v \cdot F(x) + u \cdot x},$$

where "$\cdot$" denotes inner products in $\mathbb{F}_2^n$ and $\mathbb{F}_2^m$, respectively. The set $\{\lambda(u,v) \mid (u,v) \in \mathbb{F}_2^n \times \mathbb{F}_2^m, v \neq 0\}$ is called the *Walsh spectrum* of F and the set $\Lambda_F = \{|\lambda(u,v)| \mid (u,v) \in \mathbb{F}_2^n \times \mathbb{F}_2^m, v \neq 0\}$ the *extended Walsh spectrum* of F. If $n = m$ and the Walsh spectrum of F equals $\{0, \pm 2^{\frac{n+1}{2}}\}$ then the function F is called AB [8]. AB functions exist for n odd only and oppose an optimum resistance to linear cryptanalysis (see [9]). Every AB function is APN but the converse is not true in general (see [10] for comprehensive survey on APN and AB functions).

The natural way of representing F as a function from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ is by its algebraic normal form (ANF):

$$\sum_{I \subseteq \{1,\ldots,n\}} a_I \left(\prod_{i \in I} x_i \right), \qquad a_I \in \mathbb{F}_2^m,$$

(the sum being calculated in $\mathbb{F}_2^m$). The algebraic degree $deg(F)$ of F is the degree of its ANF. F is called affine if it has algebraic degree at most 1 and it is called linear if it is affine and $F(0) = 0$.

Any affine function $A : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ can be represented in matrix form

$$A(x) = M \cdot x \oplus C, \tag{1}$$

where M is an $m \times n$ matrix and $C \in \mathbb{F}_2^m$. All operations are performed in $\mathbb{F}_2$, thus (1) can be rewritten as

$$\begin{pmatrix} a_0 \\ a_1 \\ \cdots \\ a_{m-1} \end{pmatrix}_x = \begin{pmatrix} k_{0,0} & \cdots & k_{0,n-1} \\ k_{1,0} & \cdots & k_{1,n-1} \\ \vdots & \ddots & \vdots \\ k_{m-1,0} & \cdots & k_{m-1,n-1} \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ \cdots \\ x_{n-1} \end{pmatrix} \oplus \begin{pmatrix} c_0 \\ c_1 \\ \cdots \\ c_{m-1} \end{pmatrix}$$

with $a_i, x_i, c_i, k_{j,s} \in \mathbb{F}_2$.

Two functions $F, G : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ are called *extended affine equivalent* (EA-equivalent) if there exist an affine permutation A_1 of $\mathbb{F}_2^m$, an affine permutation A_2 of $\mathbb{F}_2^n$ and a linear function L_3 from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ such that

$$F(x) = A_1 \circ G \circ A_2(x) + L_3(x).$$

Clearly A_1 and A_2 can be presented as $A_1(x) = L_1(x) + c_1$ and $A_2(x) = L_2(x) + c_2$ for some linear permutations L_1 and L_2 and some $c_1 \in \mathbb{F}_2^m$, $c_2 \in \mathbb{F}_2^n$.

Definition 1. *Functions F and G are called restricted EA-equivalent (REA-equivalent) if some elements of the set* $\{L_1(x), L_2(x), L_3(x), c_1, c_2\}$ *are in* $\{0, x\}$.

There are two special cases

- linear equivalence when $\{L_3(x), c_1, c_2\} = \{0, 0, 0\}$;
- affine equivalence when $L_3(x) = 0$.

In matrix form EA-equivalence is represented as follows

$$F(x) = M_1 \cdot G(M_2 \cdot x \oplus V_2) \oplus M_3 \cdot x \oplus V_1$$

where elements of $\{M_1, M_2, M_3, V_1, V_2\}$ have dimensions $\{m \times m, n \times n, m \times n, m, n\}$.

We say that functions F and F' from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ are *CCZ-equivalent* if there exists an affine permutation $\mathcal{L}$ of $\mathbb{F}_2^n \times \mathbb{F}_2^m$ such that $G_F = \mathcal{L}(G_{F'})$, where $G_H = \{(x, H(x)) \mid x \in \mathbb{F}_2^n\}, H \in \{F, F'\}$. CCZ-equivalence is the most general known equivalence of functions for which differential uniformity and extended Walsh spectrum are invariants. In particular every function CCZ-equivalent to an APN (respectively, AB) function is also APN (respectively, AB). EA-equivalence is a particular case of CCZ-equivalence [11]. The algebraic degree of a function is invariant under EA-equivalence but, in general, it is not preserved by CCZ-equivalence.

3. Verification of Restricted EA-equivalence

Special types of REA-equivalence, which are considered in this paper, are shown in Table 2.

Hereinafter assume that obtaining the value $F(x)$ for any x takes one step. Pre-computed values of function $F(x), F^{-1}(x)$ and corresponding substitutions are used as input for the algorithms. Thereafter, complexity of

Table 2: *Special types of REA-equivalence*

REA-equivalence	Type
$F(x) = M_1 \cdot G(x) \oplus V_1$	I
$F(x) = G(M_2 \cdot x \oplus V_2)$	II
$F(x) = G(x) \oplus M_3 \cdot x \oplus V_1$	III
$F(x) = M_1 \cdot G(x) \oplus M_3 \cdot x \oplus V_1$	IV

representing functions in needed form is not taken into account, as well as memory needed for data storage. This assumptions are introduced to be able to compare complexities of algorithms of the present paper with those of [1] where the same assumptions were made.

There are $2^{m \cdot n}$ choices of linear mappings. The complexity of obtaining the $m \times n$ matrix M satisfying the equation

$$F(x) = M \cdot G(x)$$

using exhaustive search method is $O(2^n \cdot 2^{m \cdot n})$, where $O(2^{m \cdot n})$ and $O(2^n)$ are complexities of checking all matrices for all possible $x \in \mathbb{F}_2^n$. Another natural method is based on system of equations. The complexity in this case depends only on the largest of the parameters n and m. Indeed, for square matrices we can benefit from the asymptotically faster Williams method based on system of equations with complexity $O(n^{2.3727})$ [12]. Besides, for $n \leq 64$ we can use 64-bit processor instructions to bring the complexity to $O(n^2)$ because two rows (columns) can be added in 1 step. Since any system of m equations with n variables can be considered as a system of k equations with k variables where $k = \max\{n, m\}$ then the complexity of solving such a system is

$$\mu = O(k^2)\ , \tag{2}$$

which gives the complexity of finding M by this method.

Proposition 1. *Any linear function $L : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ can be converted to a matrix with the complexity $O(n)$.*

Proof. We need to find an $m \times n$ matrix M satisfying

$$L(x) = M \cdot x$$

Suppose $\text{rows}_M(i) = (m_{ij})$, $\forall j \in \{0, 1, \ldots, n-1\}$ and $\text{cols}_M(j) = (m_{ij})$, $\forall i \in \{0, 1, \ldots, m-1\}$ are the i-th row and the j-th column of matrix M, respectively. Each value of $x \in \{2^i \mid 0 \leq i \leq n-1\}$ is equivalent to a vector with 1 at the i-th row

$$2^0 = \begin{pmatrix} 1 \\ 0 \\ \ldots \\ 0 \end{pmatrix} \quad 2^1 = \begin{pmatrix} 0 \\ 1 \\ \ldots \\ 0 \end{pmatrix} \quad 2^{n-1} = \begin{pmatrix} 0 \\ 0 \\ \ldots \\ 1 \end{pmatrix}.$$

Clearly, every column, except the i-th, becomes zero when multiplying the matrix M to $x = 2^i$. Hence, each column of matrix M can be computed from

$$L(2^i) = \text{cols}_M(i),\ i \in \{0, 1, \ldots, n-1\}.$$

For finding all columns of M it is necessary to compute n values of $L(2^i)$, $0 \leq i \leq n-1$. Consequently the complexity of transformation is $O(n)$. □

Proposition 2. *Let $F, G : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ and $G'(x) = G(x) \oplus G(0)$. Then the complexity of checking F and G for REA-equivalence of type I equals*

- $O(2^{n+1})$ *in case when for any $i \in \{0, \ldots, m-1\}$ there exists $x \in \mathbb{F}_2^n$ such that $G'(x) = 2^i$;*
- $O(m \cdot 2^{2n})$ *in case G is arbitrary.*

Proof. Let $F'(x) = F(x) \oplus F(0)$. Then REA-equivalent of type I

$$F'(x) \oplus F(0) = M_1 \cdot G'(x) \oplus M_1 \cdot G(0) \oplus V_1$$

can be rewritten in the following form

$$\begin{cases} F(0) = M_1 \cdot G(0) \oplus V_1; \\ F'(x) = M_1 \cdot G'(x). \end{cases} \tag{3}$$

In case of $G(0) = 0$ we get $V_1 = F(0)$, but in general it's necessary first to find M_1 from equation $F'(x) = M_1 \cdot G'(x)$. If the set $\{2^i \mid 0 \leq i \leq m-1\}$ is the subset of the image set of G', then the problem of finding $m \times m$ matrix M_1 is equivalent to the problem of converting linear function to matrix form with additional testing for all x in $\mathbb{F}_2^n$. It is possible to find M_1

with the complexity $O(m)$ as was shown in Proposition 1. The complexity of finding the pre-images of G' of elements 2^i, $\forall i \in \{0, \dots, m-1\}$ equals $O(2^n)$ as well as the complexity of checking $F'(x) = M_1 \cdot G'(x)$ for given M_1. In cryptography, in most cases $2^n \gg m$, so the complexity $O(m)$ can be neglected. Therefore the total complexity of verification for equivalence of F and G equals $O(2^n + 2^n + m) \approx O(2^{n+1})$.

Let now G be arbitrary and $F'(x)_i$ be the i-th bit of $F'(x)$. Denote $\mathrm{img}(G')$ the image set of G' and $u_{G'} = |\mathrm{img}(G')|$ the number of elements of $\mathrm{img}(G')$. Let also $N_{G'}$ be any subset of $\mathbb{F}_2^n$ such that $|N_{G'}| = u_{G'}$ and $|\{G'(a) | a \in N_{G'}\}| = u_{G'}$. Then to find M_1 it is necessary to solve a system below for all $i \in \{0, \dots, m-1\}$

$$F'(x_j)_i = \mathrm{rows}_{M_1}(i) \cdot G'(x_j), \;\; \forall x_j \in N_{G'},\; 0 \le j \le u_{G'} - 1 \Leftrightarrow$$

$$\Leftrightarrow \begin{cases} F'(x_0)_i = \mathrm{rows}_{M_1}(i) \cdot G'(x_0); \\ F'(x_1)_i = \mathrm{rows}_{M_1}(i) \cdot G'(x_1); \\ \dots \\ F'(x_{u_{G'}-1})_i = \mathrm{rows}_{M_1}(i) \cdot G'(x_{u_{G'}-1}). \end{cases}$$

For every i, $i \in \{0, \dots, m-1\}$, the complexity of solving the system highly depends on $u_{G'}$ and m and equals $O(\max\{u_{G'}, m\}^2)$ according to (2). Then the total complexity of finding M_1 for all m bits is $O(m \cdot \max\{u_{G'}, m\}^2)$. If value $u_{G'} \approx 2^n$, then $O(m \cdot 2^{2n})$. □

Remark 1. If it is known in advance that functions F and G in Proposition 2 are REA-equivalent of type I, then the complexity of verification $F'(x) = M_1 \cdot G'(x)$ can be ignored and the total complexity for the case $\{2^i \mid 0 \le i \le m-1\} \subset \mathrm{img}(G')$ becomes $O(2^n)$.

Proposition 3. *Let $F, G : F_2^n \mapsto F_2^n$ and G be a permutation. Then the complexity of checking F and G for REA-equivalence of type II is $O(n)$.*

Proof. Denote $H(x) = G^{-1}(F(x))$. Then the equality $F(x) = G(M_2 \cdot x \oplus V_2)$ becomes

$$H(x) = M_2 \cdot x \oplus V_2 \,.$$

Taking $x = 0$ we get $V_2 = H(0)$ and the equivalence can be represented as $H'(x) = M_2 \cdot x$, where $H'(x) = H(x) \oplus H(0)$. The method and the complexity of finding n by n matrix M_2 are similar to finding the matrix corresponding to the linear function. Therefore, the complexity equals $O(n)$. □

Proposition 4. *Let $F, G : F_2^n \mapsto F_2^m$. Then the complexity of checking F and G for REA-equivalence of type III equals $O(n)$.*

Proof. Denote $H(x) = F(x) \oplus G(x)$, then REA-equivalence

$$F(x) = G(x) \oplus M_3 \cdot x \oplus V_1$$

takes the form

$$H(x) = M_3 \cdot x \oplus V_1 \ .$$

And we have the same situation as in Proposition 3, but with $m \times n$ matrix. Thus the complexity of finding M_3 and V_1 (or showing its non-existence) equals $O(n)$. □

Every vectorial Boolean function admits the form

$$H(x) = H'(x) \oplus L_H(x) \oplus H(0) \ , \tag{4}$$

where L_H is a linear function and H' has terms of algebraic degree at least 2.

Proposition 5. *Let $F, G : F_2^n \mapsto F_2^m$ and G' be defined by (4) for G. Then the complexity of checking F and G for REA-equivalence of type IV equals*

- $O(2^{n+1})$ *in case* $\{2^i \mid 0 \leq i \leq m-1\} \subset img(G')$,
- $O(m \cdot 2^{2n})$ *in case G is arbitrary.*

Proof. Using (4) REA-equivalence of type IV can be rewritten as

$$F'(x) \oplus L_F(x) \oplus F(0) = M_1 \cdot G'(x) \oplus M_1 \cdot L_G(x) \oplus M_3 \cdot x \oplus M_1 \cdot G(0) \oplus V_1$$

and gives the system of equations

$$\begin{cases} F'(x) = M_1 \cdot G'(x); \\ L_F(x) = M_1 \cdot L_G(x) \oplus M_3 \cdot x; \\ F(0) = M_1 \cdot G(0) \oplus V_1. \end{cases}$$

It's easy to see that for a given M_1 one can easily compute M_3 and V_1 from the second and the third equations of the system. The first equation of the system leads to the two different cases for the function G' considered in Proposition 2. Hence, according to Proposition 2, the total complexity for finding G' equals $O(2^{n+1})$ and $O(m \cdot 2^{2n})$, respectively. It should be noted that the complexity of finding the matrix M_3 is not taken into account since $2^{n+1} \gg n$. □

If we add one of V_1, V_2 values to REA-equivalence, then the complexity will increase in 2^m or 2^n times respectively. REA-equivalance with V_1, V_2 and corresponding complexities are shown in Table 1. It should be mentioned that types I and III of REA-equivalence are particular cases of type IV. But taking into account different restrictions for the function G it is necessary to check all these types of EA-equivalence.

The presented methods of verification of REA-equivalence were checked using the free open source mathematical software system Sage [13]. An example of a program for the most general case (type IV) of REA-equivalence in case $\{2^i \mid 0 \leq i \leq m-1\} \subset \mathrm{img}(G')$ is shown in Appendix A. The corresponding algorithm is presented in Algorithm 7.

Algorithm 7 Checking Functions for REA-equivalence of Type IV

Input: $F'(x), L_F(x), F(0), G'(x), L_G(x), G(0)$
Output: True if F is EA-equivalent to G
for $V_2 = 0$ **to** 2^n **do**
  $H'(x) \leftarrow G'(x \oplus V_2)$;
  $L_H(x) \leftarrow L_G(x \oplus V_2)$;
  $H(0) \leftarrow G(V_2)$;
  for $i = 0$ **to** $m-1$ **do**
    x $\leftarrow 2^i$;
    $find(2^i == G(y))$;
    SetColumn(M_1,i,$H(y)$);
  end for
  $V_1 \leftarrow M_1 \cdot H(0) \oplus F(0)$;
  for $i = 0$ **to** $n-1$ **do**
    x $\leftarrow 2^i$;
    SetColumn(M_3,i,$L_F(x) \oplus M_1 \cdot L_H(x)$);
  end for
  for $i = 0$ **to** $2^n - 1$ **do**
    if $F(x)$!= $M_1 \cdot H(x \oplus V_2) \oplus M_3 \cdot x \oplus V_1$ **then**
      goto next V_2;
    end if
  end for
  return True
end for
return False

4. Conclusions

The present paper studies complexities of checking functions for special cases of EA-equivalence and it is shown that for some of this cases the complexity of checking takes polynomial time. Obtained results give a practical method for checking functions on equivalence. The best result is with the complexity $O(2^{2n+1})$ for checking REA-equivalence of the form $F(x) = M_1 \cdot G(x \oplus V_2) \oplus M_3 \cdot x \oplus V_1$ under some condition on G.

References

[1] Biryukov, A., De Cannière, C., Braeken, A., Preneel, B.: A toolbox for cryptanalysis: Linear and affine equivalence algorithms. In Biham, E. (ed.), *Advances in Cryptology — EUROCRYPT 2003*, vol. 2656 of *Lecture Notes in Computer Science*, pp. 33–50. Springer Berlin Heidelberg, 2003.

[2] Daemen, J., Rijmen, V.: *The design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.

[3] Kwon, D., Kim, J., Park, S., Sung, S., et al.: New block cipher: ARIA. In Lim, J.-I., Lee, D.-H. (eds.), *Information Security and Cryptology - ICISC 2003*, vol. 2971 of *Lecture Notes in Computer Science*, pp. 432–445. Springer Berlin Heidelberg, 2004.

[4] Gorbenko, I., Dolgov, V., Oliynykov, R., Ruzhentsev, V.: Perspective symmetric block cipher "Kalyna" – basic statements and specification. In *Applied Radio Electronics*, vol. 6, pp. 195–208. Kharkiv National University of Radioelectronics, 2007. (In Ukrainian).

[5] Oliynykov, R., Gorbenko, I., Dolgov, V., Ruzhentsev, V.: Results of Ukrainian national public cryptographic competition. In *Tatra Mountains Mathematical Publications*, vol. 47, pp. 99–113. Mathematical Institute of Slovak Academy of Sciences, 2010.

[6] Nyberg, K.: Differentially uniform mappings for cryptography. In Helleseth, T. (ed.), *Advances in Cryptology - EUROCRYPT'93*, vol. 765 of *Lecture Notes in Computer Science*, pp. 55–64. Springer Berlin Heidelberg, 1994.

[7] Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. In Menezes, A., Vanstone, S. (eds.), *Advances in Cryptology-CRYPT0'90*, vol. 537 of *Lecture Notes in Computer Science*, pp. 2–21. Springer Berlin Heidelberg, 1991.

[8] Chabaud, F., Vaudenay, S.: Links between differential and linear cryptanalysis. In *Advances in Cryptology—EUROCRYPT'94*, pp. 356–365. Springer, 1995.

[9] Matsui, M.: Linear cryptanalysis method for DES cipher. In Helleseth, T. (ed.), *Advances in Cryptology — EUROCRYPT '93*, vol. 765 of *Lecture Notes in Computer Science*, pp. 386–397. Springer Berlin Heidelberg, 1994.

[10] Carlet, C.: *Vectorial Boolean functions for cryptography*. Boolean Models and Methods in Mathematics, Computer Science, and Engineering. Cambridge University Press, 2010.

[11] Carlet, C., Charpin, P., Zinoviev, V.: Codes, bent functions and permutations suitable for DES-like cryptosystems. In *Designs, Codes and Cryptography*, vol. 15, pp. 125–156. Kluwer Academic Publishers, 1998.

[12] Williams, V. V.: Breaking the Coppersmith-Winograd barrier. *UC Berkeley and Stanford University*, 2011. `http://www.cs.rit.edu/~rlc/Courses/Algorithms/Papers/matrixMult.pdf`.

[13] Stein, W., et al.: Sage mathematics software (version 4.8.2). *The Sage Development Team*, 2012. `http://www.sagemath.org`.

A. Source Code for Verification of REA-equivalence of Type IV

```
#!/usr/bin/env sage

# Global variables
bits=0
length=0
k=0
P=0

def check_rEA4(F,G):
  r'''
    Return True if
      - F(x) = M1 * G(x) + M3 * x + V
      - G'(x) is permutation, where G(x) = G'(x) + L_G(x) + G(0)
  '''
  M1 = matrix(GF(2),nrows=bits,ncols=bits)
  M3 = matrix(GF(2),nrows=bits,ncols=bits)

  polF = F
  polG = G

  V1 = polF.constant_coefficient()
  V2 = polG.constant_coefficient()

  polF += V1
  polG += V2

  V1 = V1.integer_representation()
  V2 = V2.integer_representation()

  polFc=polF.coeffs()
  polFc += [P("0") for i in xrange(length-len(polFc))]
  polGc=polG.coeffs()
  polGc += [P("0") for i in xrange(length-len(polGc))]

  L1 = zero_vector(length).list()
  L2 = zero_vector(length).list()

  for i in xrange(bits):
    if polFc[1<<i] != 0:
      L1[1<<i] = polFc[1<<i]
      polFc[1<<i] = 0

    if polGc[1<<i] != 0:
```

```
        L2[1<<i] = polGc[1<<i]
        polGc[1<<i] = 0

    L1 = P(L1)
    L2 = P(L2)
    polF = P(polFc)
    polG = P(polGc)

    sboxF = range(length)
    sboxG = range(length)

    sboxL1 = [L1.subs(k(ZZ(i).digits(2))).integer_representation() for i
        ↪ in xrange(length)]
    sboxL2 = [L2.subs(k(ZZ(i).digits(2))).integer_representation() for i
        ↪ in xrange(length)]
    sboxF  = [polF.subs(k(ZZ(i).digits(2))).integer_representation() for
        ↪ i in xrange(length)]
    sboxG  = [polG.subs(k(ZZ(i).digits(2))).integer_representation() for
        ↪ i in xrange(length)]

    sboxFt=sboxF[:]
    sboxGt=sboxG[:]

    if len(set(sboxG).intersection(set([2^g for g in xrange(bits)]))) !=
        ↪ bits:
      #print ">>> sboxG hasn't all values of {0} <<<".format([2^g for g
          ↪ in xrange(bits)])
      return None

    for i in xrange(bits):
      x=sboxGt.index(1<<i)
      M1.set_column(i,ZZ(sboxFt[x]).digits(base=2,padto=bits))

    sboxM = range(length)

    V = ZZ((M1*vector(GF(2),ZZ(V2).digits(base=2,padto=bits))).list(),2)
        ↪ ^^ V1
    for i in xrange(length):
      sboxM[i] = sboxL1[i] ^^ ZZ((M1*vector(GF(2),ZZ(sboxL2[i]).digits(
          ↪ base=2,padto=bits))).list(),2) ^^ V

    sboxT=sboxM[:]

    V = vector(GF(2),ZZ(sboxT[0]).digits(base=2,padto=bits))

    if sboxT[0] != 0:
      sboxT = [ g^^sboxT[0] for g in sboxT ]
```

```
  for i in xrange(bits):
    x=1<<i
    M3.set_column(i,ZZ(sboxT[x]).digits(base=2,padto=bits))

  sbox = range(length)

  sF  = [F.subs(k(ZZ(i).digits(2))).integer_representation() for i in
      ↪ xrange(length)]
  sG  = [G.subs(k(ZZ(i).digits(2))).integer_representation() for i in
      ↪ xrange(length)]

  for i in xrange(length):
    sbox[i]=vector(GF(2),ZZ(sG[i]).digits(base=2,padto=bits))

    sbox[i]=M1*sbox[i]

    tx=M3*vector(GF(2),ZZ(i).digits(base=2,padto=bits))

    sbox[i]=vector(GF(2),[ZZ(sbox[i].get(j)) ^^ ZZ(tx.get(j)) ^^ ZZ(V.
        ↪ get(j)) for j in xrange(len(sbox[i]))])

    sbox[i]=ZZ(sbox[i].list(),2)

  if sbox == sF:
    return [M1,M3,V]
  else:
    return None

def is_EA_equivalent(F,G,functions):

  for v2 in xrange(length):
    polG=G.subs(P("x+{0}".format(k(ZZ(v2).digits(2))))).mod(P("x^{0}+x
        ↪ ".format(length)))

    ret=check_rEA4(F,polG)

    if ret != None:
      M1=ret[0]
      M3=ret[1]
      V1=ret[2]
      V2=vector(GF(2),ZZ(v2).digits(base=2,padto=bits))
      if functions == True:
        return [M1,V1,V2,M3]
      else:
        return True
```

```
126     return False

128 def main(argv=None):
      global bits,length,k,P
130
      bits=6
132   length=1<<bits
      k=GF(2^bits,'a')
134   P=PolynomialRing(k,'x')

136   F=P.random_element(length-1)
      G=P.random_element(length-1)
138
      # Test polynomials for bits=6
140   #G=P("a^63*x^0 + a^61*x^1 + a^23*x^2 + a^39*x^3 + a^15*x^4 + a^21*x
          ↪ ^5 + a^57*x^6 + a^37*x^7 + a^3*x^8 + a^23*x^9 + a^26*x^10 + a
          ↪ ^40*x^11 + a^48*x^12 + a^26*x^13 + a^51*x^14 + a^43*x^15 + a
          ↪ ^32*x^16 + a^13*x^17 + a^33*x^18 + a^48*x^19 + a^36*x^20 + a
          ↪ ^1*x^21 + a^11*x^22 + a^40*x^23 + a^42*x^24 + a^62*x^25 + a
          ↪ ^11*x^26 + a^22*x^27 + a^5*x^28 + a^6*x^29 + a^59*x^30 + a
          ↪ ^10*x^31 + a^51*x^32 + a^4*x^33 + a^13*x^34 + a^63*x^35 + a
          ↪ ^54*x^36 + a^26*x^37 + a^58*x^38 + a^39*x^39 + a^53*x^40 + a
          ↪ ^34*x^41 + a^28*x^42 + a^27*x^43 + a^40*x^44 + a^25*x^45 + a
          ↪ ^10*x^46 + a^58*x^47 + a^30*x^48 + a^34*x^49 + a^35*x^50 + a
          ↪ ^49*x^51 + a^53*x^52 + a^35*x^53 + a^49*x^54 + a^7*x^55 + a
          ↪ ^55*x^56 + a^39*x^57 + a^53*x^58 + a^29*x^59 + a^52*x^60 + a
          ↪ ^45*x^61 + a^9*x^62 + a^26*x^63")
      #F=P("a^44*x^0 + a^34*x^1 + a^7*x^2 + a^5*x^3 + a^51*x^4 + a^40*x^5
          ↪ + a^27*x^6 + a^23*x^7 + a^28*x^8 + a^63*x^9 + a^20*x^10 + a
          ↪ ^38*x^11 + a^12*x^12 + a^16*x^13 + a^18*x^14 + a^39*x^16 + a
          ↪ ^53*x^17 + a^62*x^18 + a^17*x^19 + a^50*x^20 + a^13*x^21 + a
          ↪ ^15*x^22 + a^29*x^23 + a^33*x^24 + a^12*x^25 + a^22*x^26 + a
          ↪ ^49*x^27 + a^7*x^28 + a^43*x^29 + a^28*x^30 + a^53*x^31 + a
          ↪ ^5*x^32 + a^59*x^33 + a^22*x^34 + a^26*x^35 + a^45*x^36 + a
          ↪ ^39*x^37 + a^49*x^38 + a^9*x^39 + a^58*x^40 + a^13*x^41 + a
          ↪ ^14*x^42 + a^43*x^43 + a^61*x^44 + a^38*x^45 + a^10*x^46 + a
          ↪ ^9*x^47 + a^25*x^48 + a^44*x^49 + a^30*x^50 + a^12*x^51 + a
          ↪ ^16*x^52 + a^24*x^53 + a^56*x^54 + a^3*x^55 + a^40*x^56 + a
          ↪ ^23*x^57 + a^49*x^58 + a^39*x^59 + a^58*x^60 + a^11*x^61 + a
          ↪ ^55*x^62 + a^29*x^63")
142
      print "F\t= {0}".format(F)
144   print "G\t= {0}".format(G)

146   ret=is_EA_equivalent(F,G,functions=True)

148   if ret != False:
```

```
    [M1,V1,V2,M3]=ret
150    print "EA\t\t\t\t= {0}".format(True)
    print "V1:\n{0}".format(V1)
152    print "V2:\n{0}".format(V2)
    print "M1:\n{0}".format(M1)
154    print "M3:\n{0}".format(M3)
  else:
156    print "EA\t\t\t\t= {0}".format(False)

158 if __name__ == "__main__":
  sys.exit(main())
```