A New Approach to Keystream Based Cryptosystems

Imran Erguler, Orhun Kara

Abstract— **One-time pad is a simple secure symmetric cryptosystem in which the message is combined with randomly selected secret key as long as the plaintext to output ciphertext. If the key is truly random, used only once and kept secret, then the one-time pad is perfectly secure against a ciphertext-only attack. Due to nature of the key which is a nondeterministic sequence, system can be thought as use of a pure noise source to mask the message. However it is obviously known that one-time pad is not practical. Hence, deterministic finite state machines, mostly stream ciphers, are used: A keystream sequence is produced by a seed and it is mixed with plaintext for encryption at the transmitter side. Note that since generated keystream sequence is deterministic, anyone having the seed can reproduce it and recover the plaintext easily. In this study, we propose a new keystream based encryption model which provides security enhancement against some well known attacks. For this; firstly we apply forward error correcting codes (FEC) to plaintext. Next, we add a nondeterministic noisy sequence to keystream and obtain a nondeterministic bit sequence which is combined with plaintext to generate ciphertext. Even though the receiver does not know the content of the nondeterministic sequence, he can still obtain the original message. On the other hand, for an attacker knowing plaintext does not mean knowing the deterministic keystream. This forces a new definition for cryptanalysis of keystream sequences in case of known plaintext. We also give a new definition of ``good codes'' in terms of improving security in this mode.l**

Index Terms— **Forward error correcting, one-time pad, stream ciphers,**

I. Introduction

STREAM ciphers are commonly used to provide a reliable and efficient method for communicating securely. The main goal in stream ciphers is to expand a short key K into a long keystream Z which is difficult to distinguish from a truly random bit stream. In other words, they play role in generating long pseudorandom bit sequences from a short and random key. Usually, these sequences are mixed with plaintext M by using XOR operation to produce ciphertext C. Actually, this idea is not new and has a long history. In 1917, G. Vernam introduced the remarkably simple one-time pad in which the message is combined with randomly chosen secret key of same length to produce ciphertext [1]. For binary data, obviously this operation is XOR. If the key bits are completely random and key is never reused, then one-time pad is perfectly secure against a ciphertext-only attack in which the attacker has no knowledge about plaintext, eventhough an infinite computational power can be available to the attacker. Since the key is truly random, i.e. a nondeterministic sequence, we can think the one-time pad as adding a pure noise sequence to the message to hide it. Nondeterministic behaviour of the system actually is the major difference between todays stream ciphers and one-time pad. For the former, keystream sequence is deterministic and anyone can easily reproduce the same sequence with knowledge of the cipher initial state. Thus, finite state machine (FSM) based stream ciphers can not provide perfect secrecy. Nevertheless inconvenience of using one-time pads in practical life is evident, because it has handicaps such as: required key length is equal to message length, key must be truly random, never reused and difficulty in key distribution and management.

Binary additive stream cipher is a good candidate to provide solutions to the above problems. It is a synchronous stream cipher in which the keystream, plaintext, and ciphertext denoted are sequences of binary digits, and keystream is combined with the plaintext using XOR operation to output ciphertext [2]. For each secret key, the stream cipher generates a different deterministic keystream sequence. Since it is assumed that K is a shared secret between the transmitter and receiver, receiver produces the same keystream sequence and obtains plaintext by XOR'ing ciphertext with keystream. In the remaining parts of the study, we suppose the stream cipher is a binary additive stream cipher unless otherwise is stated.

In this study, we propose a new keystream encryption model and analyze its security by studying over some well known ciphers. According to this model, the sender produces a nondeterministic keystream sequence, call it noisy-keystream. There are two sequences whose combination builds this noisy-keystream sequence. One of them is produced by a FSM and the other is constructed truly random. Although the receiver can generate only the deterministic part of the keystream produced by the FSM, he can perfectly decrypt the messages. To do this, at the transmitter side plaintext sequence M is encoded into codeword sequence V by using error-correcting codes. Then a truly random binary noise sequence is mixed with FSM keystream Z. We assume that the noise sequence itself can be corrected by the error correcting code. Next, this result is added to V to generate ciphertext. At the receiver side, ciphertext is XOR'ed with FSM keystream and noisy code string $\hat{V}$ is obtained. Since error correcting coding is used,

O. Kara is with Ntional Research Institute of Electronics and Cryptology TUBITAK-UEKAE, Gebze, TURKEY (e-mail: orhun@ uekae.tubitak.gov.tr).
I. Erguler is with Electrical-Electronics Engineering Department, Bogazici University, Bebek, TURKEY (e-mail: ierguler@ uekae.tubitak.gov.tr).

original message M is recovered from the noisy code sequence successfully. On the other hand, for a known plaintext attack (an amount of keystream sequence is given to the attacker and by using this the attacker tries to recover information about the cipher), the attacker gets corresponding noisy-keystream by XOR'ing ciphertext with V but not the original FSM keystream itself. Thus, known plaintext attack complexities will now change and we show how it effects security of the stream ciphers in favor of the cryptographers. The beauty of the model can be figured as: For one-time pad and traditional stream ciphers, the receiver must know/produce same keystream sequence to recover the message. On the other hand for the proposed model, it is not necessary for receiver to know the noisy-keystream in decryption process. In other words, noisy-keystream is not directly distributed to the trusted parties before communication. So we achieved to create an asymmetry in symmetric cryptosystem besides security enhancement: The transmitter produces two secret sequences whereas it is enough that corresponding receiver generates only the deterministic part of the noisy-keystream for decryption.

The outline of the paper is the following. In Section 2, we give detailed description of the encryption model, definitions and its some characteristics. In Section 3, we consider some popular known plaintext attacks and investigate security enhancement of the proposed model on some known stream ciphers. In Section 4, we explain some advantages and disadvantages of the model. Section 5 gives a new definition for ``good codes" in terms of improving security according to this model. Finally, we conclude in Section 6.

II. A General Description Of The Proposed Model

We present a new approach for encryption systems in which plaintext is XOR'ed with a keystream sequence. For this model at the transmitter site, plaintext sequence M is encoded into codeword sequence V by using error-correcting codes, specifically block codes. That is plaintext sequence is segmented into blocks of fixed length; each block, denoted by $\mathbf{m}$, consists of k plaintext bits as $\mathbf{m}= (m_1,m_2,...,m_k)$. Then each block $\mathbf{m^i}$ is encoded to codeword $\mathbf{v^i}$ with length n, where $n>k$. Then n bit length keystream block $\mathbf{z}$ is XOR'ed with a noise sequence, think it as an error vector $\mathbf{e}= (e_1,e_2,...,e_n)$ whose Hamming weight is represented as $w(\mathbf{e})$. The result at this step is called noisy-keystream block and expressed as $\hat{\mathbf{z}}^{\mathbf{i}} = \mathbf{e^i} \oplus \mathbf{z^i}$. At the last step, each $\mathbf{v^i}$ is summed with $\hat{\mathbf{z}}^{\mathbf{i}}$ to produce ciphertext block which is equivalent to say $\mathbf{c^i} = \mathbf{v^i} \oplus \hat{\mathbf{z}}^{\mathbf{i}}$. For a perfect noiseless channel, the receiver side then sums $\mathbf{c^i}$ with corresponding $\mathbf{z^i}$ for decryption process and gets $\mathbf{v^i} \oplus \mathbf{e^i}$. Finally, by means of code decoding $\mathbf{m^i}$ is extracted from $\mathbf{v^i} \oplus \mathbf{e^i}$.

Content of $\mathbf{e^i}$ can be changed for different applications: In first case, $\mathbf{e^i}$ can be generated by a True Random Number Generator such that $w(\mathbf{e^i}) \le t$, then plaintext block $\mathbf{m^i}$ is recovered exactly correct at the receiver, where t is the error correcting capability of the code. Also, $w(\mathbf{e^i})$ can be determined by a statistical distribution such as Gaussian, i.e. $w(\mathbf{e^i})$ behaves like noise element of a communication channel. In this case, it is not guarantied that original message is received without an error, because $w(\mathbf{e^i})$ can be greater than t for some blocks. Although errors can be observed in the decoded message, some applications can tolerate this fault, like most of the voice communications, if error amount is not severe. In fact, this mode provides higher security than the first case, however some applications are very sensitive and just occurrence of a single bit error fails the protocol. Hence, in remaining part of the study we only consider application of the first case.

The proposed model does not have any negative effect on data integrity of message for the receiver. However there is some confusion from a cryptanalyst point of view. Suppose the cryptanalyst intends to apply a known plaintext attack which requires D bits of data. Let $j = \lfloor D/k \rfloor$, so he knows $\mathbf{m^1}$, $\mathbf{m^2}$,..., $\mathbf{m^j}$ plaintext block sequence. Also it is assumed error control coding algorithm used in the system is publicly known. Thus, the attacker can get corresponding codewords $\mathbf{v^i}$'s $\forall i \in \{1,..., j\}$ easily. In classical known plaintext attacks some piece of keystream sequence Z is obtained by XOR'ing known plaintext data and corresponding ciphertext as $Z = M \oplus C$. However for this scenario, situation is a little bit different. When the attacker XOR's known codeword $\mathbf{v^i}$ with $\mathbf{c^i}$ as $\mathbf{c^i} \oplus \mathbf{v^i} = \mathbf{v^i} \oplus \mathbf{e^i} \oplus \mathbf{z^i} \oplus \mathbf{v^i} = \mathbf{e^i} \oplus \mathbf{z^i}$, he gets a noisy version of $\mathbf{z^i}$ rather than the exact keystream block. As a result, now the attacker must also consider in which positions errors exist in the key stream. Furthermore, the errors in the sequence is created randomly so he can not find a relation between the error locations and state of stream cipher or key K. Therefore he should do a separate analysis to get correct keystream sequence. In other words, the additional problem that the attacker faces is to eliminate the noise from the keystream. This makes the cryptanalyst's problem much more difficult. The worst case for the attacker to solve this problem is trying all possible error locations and then mounting his classical attack method. However note that it is a combinatorial problem and the attack becomes infeasible if required keystream amount of the attack is high. We analyze increase in complexities for some attacks in the next section.

III. Attacks

In this section, we investigate security contribution of the proposed model against some well known attacks and to confirm the idea some ciphers against these attacks with adaptation of the model are reanalyzed

A. Time-Memory Trade-off Attacks

Cryptanalytic Time-Memory Trade-off (TMTO) is an efficient known plaintext attack type for stream ciphers, since most of the time keystream sequences are independent from plaintext and a common pre-computed table could be used in analysis of different ciphertext blocks. TMTO attack against

stream ciphers was firstly presented through independent works by Babbage [3] and Golic [4], hence known as BG model. Let N represent total solution space of the generator internal state. According to these models; by using D different known keystreams of length log$N$ and a pre-computed table, which stores M randomly chosen possible internal states and their corresponding log$N$ length outputs, possible internal state of the stream generator is determined. If $M=N/D$ is chosen, and $DM = N$ is satisfied, from the birthday paradox there exists a good chance of finding a match between the D keystreams and outputs of the table. After recovering a possible internal state, the generator is run forwards if it is desired to obtain rest of the plaintext or run backwards to get the secret encryption key K. The TMTO curve of the attack can be written as follows:

$$TM=N$$

where T denotes the time complexity of the attack with satisfying $1 \le T \le D$. A new TMTO idea was presented by Biryukov and Shamir in [5]. According to this study r/D different tables are constructed. Each table has m elements each of which is iterated r times. For these parameters $M=mr/D$ and $T=r^2$ are obtained, where $mr^2=N$. The trade-off curve of the study is given below:

$$TM^2D^2=N^2$$

with satisfying $1 \le D^2 \le T$. By means of this model, the need for known data becomes smaller compared to BG model.

Known data does not necessarily come from a single successive bit sequence, i.e. it can be obtained by accumulation of some shorter successive bit sequences. This scenario is very realistic for most of the stream cipher based communication systems. In these systems data is transmitted in a constant length, call it Γ, frame format and for each frame the cipher uses different IVs with the same encryption key. In other words, after production of Γ kesytream bits, the stream cipher is reinitialized with a different IV. Let F denote the number of frames available to the attacker. Regarding the above attack, number of log$N$ bit length windows that the attacker obtains from each frame is $\Gamma - \log N + 1$. Hence D can be expressed as $D = F \cdot (\Gamma - \log N + 1)$.

Now lets examine our encryption model against this attack. Suppose an error correcting coding with parameters $[n,k,t]$ is used and $w(\mathbf{e}^i)=t$. The attacker analyzes frames individually, tries to find a match between log$N$ bit keystream windows and outputs of the pre-computed table. Also if there exists a match, it is assumed that attacker can check correctness of possible internal state. Actually another amount of known data is usually used to verify correctness of the candidate internal state which requires an extra effort according to our model, but now we ignore this cost. The effective way from the point of the attacker is finding all possible keystream sequences corresponding to each frame, and then searching log$N$-bit length windows at the table. Notice that for each n bits of noisy-keystream sequence, the number of possible candidates for the correct keystream block is $\binom{n}{t}$. Let τ represent number of such blocks in Γ, then we can write $\tau = \lfloor \Gamma / n \rfloor$. Note that if $\Gamma - n/\tau > 0$, possibilities for the remained $l = \Gamma - n/\tau$ bits exist. Hence total number of possible keystream sequences in a single frame will be

$$\binom{n}{t}^{\tau} \sum_{i=\max(0,t+l-n)}^{\min(t,l)} \binom{l}{i}. \qquad (1)$$

We know that verification of the attack depends on correctness of the known data, so the attacker needs to try all of these possible candidates to ensure obtained cipher state is the correct one. Recall that the attacker uses F different frames and each search in the table costs r^2/D so the total required time complexity of the attack, denoted as T, now becomes

$$\mathrm{T} = F \cdot (\Gamma - \log N + 1) \cdot \frac{r^2}{D} \binom{n}{t}^{\tau} \sum_{i=\max(0,t+l-n)}^{\min(t,l)} \binom{l}{i},$$

which is equal to

$$\mathrm{T} = T \cdot \binom{n}{t}^{\tau} \sum_{i=\max(0,t+l-n)}^{\min(t,l)} \binom{l}{i}. \qquad (2)$$

It is obvious that for BG model, (2) is still valid if $D=T$.

Example 1. Golic proposed a time-memory trade-off attack against GSM A5/1 stream cipher in [4]. A5/1 stream cipher is a binary linear feedback shift register (LFSR) based key stream generator. It combines three LFSRs of lengths 19, 22 and 23 bits which are denoted by R1, R2 and R3 respectively. All of these registers have primitive feedback polynomials and each register is updated according to its own feedback polynomial. Each LFSR has a single clocking tap in bit 8 for R1, bit 10 for R2 and bit 10 for R3; denoted as C1, C2 and C3 respectively. Clocking mechanism of each LFSR is determined according to the majority rule: Each clock cycle majority of C1, C2, and C3 is calculated and two or three registers whose clocking tap value is the same as majority bit are clocked. The output of A5/1 is produced by XOR'ing the most significant bit of each register. The initial state of A5/1 is carried out as follows: All of the registers are first zeroed and then 64 bit secret session key K_c and 22 bit frame number F_n XOR'ed (ignoring majority rule) in parallel into the least significant bits (lsb) of the three registers. In the next step all LFSRs are clocked for 100 clock cycles according to majority rule, however no output is produced. Finally, three LFSRs are clocked according to majority rule to generate 228 bits of key stream sequence. 114 of 228 bits are used as the keystream bits for one direction of communication and remained 114 bits are used as the keystream for the other direction of communication. Thus, we can evaluate the system as each

frame constitutes one 228 successive bit block to the attacker. However in [4], due to description of A5/1 alleged version, it is mentioned that after production of first 114 bits keystream, A5/1 is clocked 100 times without producing output and then outputs remained 114 bits.

According to this model, $N=2^{63.32}$, $F=2^{21}$, $\Gamma=114$ which gives $51\log N$ length windows. Since each frame gives two 114 bit blocks, 102 windows are provided from a single frame. For these values time complexity of the attack is computed as $T=102\cdot F\approx 2^{27.67}$. We use the same A5/1 architecture to make the results comparable. Now lets see how the security of the system can be improved by using our proposed encryption model. Suppose an error correcting coding with parameters $[n,k,t]$: [15,7,2] is used and $w(\mathbf{e}^i)=2$. For $\Gamma=114$, we obtain $\tau=7$ and $l=9$. By using (2), we get

$$\mathrm{T}=T\cdot\binom{15}{2}^7\sum_{i=0}^{2}\binom{9}{i}=2^{27.67}\binom{15}{2}^7\sum_{i=0}^{2}\binom{9}{i}\approx 2^{80.198}.$$

One can see that required time complexity of the attack becomes larger than that of exhaustive search, so the proposed model makes the attack ineffective.

B. Guess and Determine Attacks

In this type of attack, an attacker begins by guessing some internal variables of the stream cipher and then he tries to determine the other variables based on the observed keystream sequence. If guess is correct, it is confirmed by running the cipher forwards for some time and checked to see the match between the output from the guess and the observed sequence. In case of a wrong guess, simply a new guess is made and process is repeated. Mostly, guess-and-determine attacks require some amount of known keystream sequence to recover unguessed parts of the cipher and then to verify correctness of the guess. Hence, even a single bit change in the used keystream block can cause failure of the attack. As a consequence, the proposed model can provide serious security enhancement against such type of attacks.

Now lets try to model guess-and determine type attacks. Assume, X denotes guessed part of the cipher state, A and F represents known successive keystream bit block used in each guess and number of guesses for a given A respectively. Suppose W denotes set of different keystream blocks A derived from known keystream to be used in guesses. Moreover let the function $f(X,A)=Y$ output Y which stands for predicted whole state of the cipher with given parameters X and A. Also the function $g(Y,Z)$ returns TRUE if Y agrees with keystream Z, and FALSE otherwise. So the attack can be summarized as indicated in Fig. 1.

Now suppose only noisy version of keystream $\hat{Z}$ is available to the attacker. After the attacker builds W from $\hat{Z}$, he must consider all possible candidates for each A block; due to existence of noise. Let the function $h(A)$ output possible A candidates whose set is represented by Q for a given A. Then attack procedure is proceeded as above. The attack process with considering our proposed model is shown in Fig.2.

```
i=0; j=0;
   while j<|W|
      A=W(j);
      j=j+1;
       while i<F
            Y=f(X(i),A);
            if(g(Y,Z))
return;
            i=i+1;
```

Fig. 1: A generic guess-and-determine attack

```
i=0; j=0; k=0;
   while j<|W|;
      A=W(j);
      Q=h(A);
      j=j+1;
       while k<|Q|
          A'=Q(k);
          k=k+1;
          while i<F
             Y=f(X(i),A');
             if(g(Y,Z))
return;
             i=i+1;
```

Fig. 2: Guess-and-determine attack with adaptation of the proposed model

Let D and T denote required known keystream amount and time complexity of the original attack respectively. If T stands for the total time complexity of the attack within adaptation of the proposed model, then we can still use (2) to express it. Notice that adaptation of the model results in an extra loop in the attack procedure. Actually, this is the complexity factor that the model provides to increase security of the system. Now $\tau=\lfloor A/n\rfloor$, $l=A-n\tau$ and used code has parameters $[n,k,t]$. As it can be seen, the security contribution of the proposed model against guess-and-determine attacks is related to how A large is and of course it depends on used stream cipher algorithm. Note that as A increases computation volume in second loop also exponentially increases. Below, we analyze security improvement of the proposed model for two different algorithms ORYX and A5/1: For the first algorithm A is low so model provides little contribution, however for A5/1 it is shown that the attack becomes impractical.

Example 2. ORYX stream cipher is a simple binary LFSR based key stream generator used in North American digital cellular systems to protect cellular data transmission. It performs encryption by XOR'ing plaintext bytes with the generated key stream bytes [6]. ORYX combines three 32-bit

LFSRs denoted by $LFSR_A$, $LFSR_B$ and $LFSR_K$ respectively. It also has an S-box which holds a permutation of the numbers 0-255, i.e. a permutation for one byte referred to as L which is not changed throughout the call. It is produced by a known algorithm that is initialized with a value that is transmitted in the clear during the call setup. Each byte of keystream is generated as follows: $LFSR_K$ and $LFSR_A$ is stepped once but $LFSR_A$ is shifted with one of two different feedback polynomials. The decision of which function is to be used depends on one of high eight bits of $LFSR_K$. $LFSR_B$ is stepped either once or twice depending on another one of the high eight bits of $LFSR_K$. Then the high bytes of the current states of $LFSR_A$ and $LFSR_B$ permuted with L and added to high byte of the current state of $LFSR_K$ to form a keystream byte i.e.:

Keystream= $\{High8_K + L[High8_A] + L[High8_B]\} \mod 256\}$.

ORYX was cryptanalized by D. Wagner et. al. in [6] with a guess-and-determine type of attack, requiring about 25-27 bytes of known plaintext and 2^{16} as time complexity. The attack is mounted as follows: The ORYX can be thought to have a 96-bit keyspace, since the total length of the three LFSRs is 96 bits. Let $Z(i)$ denote the known byte of keystream produced at time i. Also the high eight bits of $LFSR_A$, $LFSR_B$ and $LFSR_K$ at time i are represented $High8_A(i)$, $High8_B(i)$ and $High8_K(i)$ respectively. Thus initially these bits are $High8_A(0)$, $High8_B(0)$ and $High8_K(0)$. At time $i = 1$, $LFSR_A$ and $LFSR_K$ are clocked once producing $High8_A(1)$ and $High8_K(1)$, $LFSR_B$ is shifted either once or twice producing $High8_B(1)$. The first byte of keystream is generated by the combining function applied to the high eight bits of each LFSR. At the first step of the attack the contents of $High8_A(1)$ and $High8_B(1)$, totaly 16 bits, are guessed and the corresponding $High8_K(1)$ is deduced by using the combination function with the known byte of keystream $Z(1)$. After this guess part, the attack proceeds iteratively. In each iteration a set of predictions for $High8_A(i)$, $High8_B(i)$ and $High8_K(i)$ for $2 \le i \le 25$ is formed and guess is evaluated by comparing the corresponding keystream byte with the predicted values. After each iteration of i, there are one unknown bit for each of $High8_A(i+1)$, $High8_K(i+1)$ and one or two unknown bits for $High8_B(i+1)$. Therefore a total of 12 possible states are tried for each iteration. If none of the tried states generates the known keystream byte, then it means guesses for $High8_A(1)$ and $High8_B(1)$ were wrong and it is started over with new guesses. Otherwise attack continues in a similar fashion with exploring each branch of possible states that is consistent with the known keystream. After 24 successful iterations the internal states of all three shift registers are obtained with a high probability. Since each iteration requires one byte of known keystream, and one byte is required for the initial guess of the $High_8$ bits, attacks needs 25 bytes of known keystream and at most 2^{16} guesses to recover the 96-bit entire ORYX state. Within these results, it is seen that ORYX stream cipher offers a very low level of security.

Now lets assume the proposed model is adapted to the communication system and examine the resistance of the cipher against the attack. In this example, we intend to emphasize the significance of A for the complexity of the attack. According to the mentioned guess-and-determine attack for ORYX; only first byte is crucial regarding the proposed model i.e. A=8 bits, although required keystream amount D is 25 bytes. It is not necessarily to try all possible candidates for the remained 24 bytes, because in each iteration 12 possible states are tried and to conclude guess is possibly correct: It is checked whether the generated output has hamming distance with noisy-keystream block is sensible or not. Suppose an error correcting coding with parameters $[n,k,t]$:[15,7,2] is used and $w(\mathbf{e}^i)$=2. According to experimental results in [6], the attack yields 99% success about 25 bytes of known plaintext. For the first 8 bits of noisy-keystream sequence, the number of possible candidates for the correct keystream block is $\sum_{i=0}^{2}\binom{8}{i} \approx 2^{5.21}$. As a result, the total time complexity of the attack now becomes $2^{5.21}2^{16}=2^{21.21}$. Actually this algorithm is not a good example to show the security power of the proposed model against guess-and-determine attacks. However one can see that it still increases resistance of the system 37 times, though A is very small (less than a codeword length) which is not the usual case for most of the stream ciphers.

Example 3. In [7], Biham and Dunkelman presented a guess-and-determine type attack which breaks A5/1 cipher in $2^{39.91}$ A5/1 clocking with requiring $2^{20.8}$ keystream bits. The basic idea behind the attack is to wait until an event which leaks a large amount of information about the key occurs and then to exploit it. It is assuming for 10 consecutive rounds the register R3 is not clocked. Then firstly guessing twelve bits from R1, R2 and R3 (9 bits from R1, R3[10], R3[22] and R2[0]) and using known keystream sequence, contents of R1 and R2 are recovered. Next, some possible values for unknown bits of R3 are tried by using a pre-computed table and the whole state of the cipher is revealed. Finally, correctness of the guess is verified by checking produced output with known keystream. Since the location of occurrence of such a special event is unknown, about 2^{20} possible starting locations are examined. Thus this process requires $228/(228\text{-}97).2^{20}=2^{20.8}$ bits of keystream. Suppose we apply the proposed model to A5/1 cipher with code parameters $[n,k,t]$:[15,7,2] and $w(\mathbf{e}^i)$=2. For each guess 97 bits of keystream block is used, so A=97. If we compute the values in (2), we get $\tau = 6$, l=7 and the total time complexity of the attack in this case becomes

$$2^{39.91} \cdot \binom{15}{2}^6 \sum_{i=0}^{2}\binom{7}{i} = 2^{85.06}.$$

Notice that by means of the proposed model the attack complexity becomes worse than that of exhaustive search.

C. Correlation Attacks

One very important class of attacks on LFSR-based stream ciphers is correlation attacks. The original idea was described by Siegenthaler considering nonlinear combination generators

[7]. Let **z** denote a keystream sequence of length *N*, as $\mathbf{z}=z_1,z_2,\ldots,z_N$. Assume that the keystream sequence is generated from a generator using *n* different LFSRs. The main idea is that if a correlation between the known output sequence and the output of one individual LFSR exists, it is possible to mount a "divide-and-conquer" attack on the individual LFSR. If one can detect a correlation between the output of one of the shift registers, called the target LFSR, and the keystream, such that $P(u_i = z_i) \neq 0.5$ for $i \geq 1$, where u_i is the output of the target LFSR, then one can attempt to find the initial state of the target LFSR. The attack can be mounted as follows: Let $P(u_i = z_i) = \frac{1}{2} + \varepsilon$ and the length of the registers be $l_1,l_2,\ldots,l_n$. For each possible state of the target LFSR$_i$ i.e. totally 2^{l_i-1}, the following correlation metric is computed

$$C(u,z) = \sum_{j=1}^{N} (-1)^{u_j \oplus z_j} \quad . \tag{3}$$

The expected value of this metric is equal to $2N\varepsilon$ when **u** is the correct value of the target initial state. It can be seen that the complexity of finding whole state of the generator is reduced from $\prod_{j=1}^{N} 2^{l_j-1}$ to $\sum_{j=1}^{N} 2^{l_j-1}$ by using this attack. If ε is close to zero, the required keystream length is $N = O(\varepsilon^{-2})$.

Suppose now we adapt our proposed model to the output of such a combination generator. Let $\hat{\mathbf{z}}$ be the noisy-keystream sequence and expressed as $\hat{\mathbf{z}} = \mathbf{e} \oplus \mathbf{z}$. Moreover $P(u_i = z_i) = \frac{1}{2} + \varepsilon_1$ and $P(z_i = \hat{z}_i) = \frac{1}{2} + \varepsilon_2$. Then one can see that

$$P(u_i = \hat{z}_i) = [P(u_i = z_i) \cdot P(z_i = \hat{z}_i)] + [P(u_i \neq z_i) \cdot P(z_i \neq \hat{z}_i)]$$

$$P(u_i = \hat{z}_i) = \left(\frac{1}{2} + \varepsilon_1\right)\left(\frac{1}{2} + \varepsilon_2\right) + \left(\frac{1}{2} - \varepsilon_1\right)\left(\frac{1}{2} - \varepsilon_2\right) = \frac{1}{2} + 2\varepsilon_1\varepsilon_2 \quad .$$

We can explain the total bias as $\hat{\varepsilon} = 2\varepsilon_1\varepsilon_2$. Hence, required keystream amount becomes $N = O(\hat{\varepsilon}^{-2}) = O(1/(4{\varepsilon_1}^2{\varepsilon_2}^2))$. Notice that to decrease the bias $\hat{\varepsilon}$ in the sequence, i.e. increase *N*, ε_2 must be very close to 0. To realize this we can use the following encryption system:

$$\mathbf{c}^{\mathbf{i+1}} = \mathbf{v}^{\mathbf{i+1}} \oplus \mathbf{z}^{\mathbf{i+1}} \oplus \mathbf{e}^{\mathbf{i+1}} \oplus S(\mathbf{z}^{\mathbf{i}})$$

where $i \geq 1$ and *S* is a bijective mapping function taking *n*-bit length input and producing evenly distributed *n*-bit output. Apart from the classical technique, in this system we add the term $S(\mathbf{z}^{\mathbf{i}})$. In other words, in generation of each ciphertext block $\mathbf{c}^{\mathbf{i+1}}$, we use *S*() of the previous keystream block. From the receiver side this process does not result in an extra cost regarding time, because $S(\mathbf{z}^{\mathbf{i}})$ can be computed separately from the received data. On the other hand, for an attacker recovering the $\mathbf{z}^{\mathbf{i+1}}$ now requires knowledge of $\mathbf{z}^{\mathbf{i}}$, so guessing $\mathbf{e}^{\mathbf{i+1}}$ individually is not enough to capture $\mathbf{z}^{\mathbf{i+1}}$. Moreover for this system the noisy-keystream becomes $\hat{\mathbf{z}}^{\mathbf{i+1}} = \mathbf{z}^{\mathbf{i+1}} \oplus \mathbf{e}^{\mathbf{i+1}} \oplus S(\mathbf{z}^{\mathbf{i}})$. Thus, we get rid of the bias $\hat{\varepsilon}$ in the sequence under the assumption that $S(\mathbf{z}^{\mathbf{i}})$ is evenly distributed.

Actually, one can criticize use of *S*() function in the system and claim the above contribution is stemmed from the nature of the *S*(), not from the proposed model. However note that for classical scenarios *S*() individually contributes nothing in terms of security, since it is bijective and z^i is known. Hence, an attacker can easily take $S^{-1}(z^i)$. For the proposed model; it improves security of the system, because computing complexity of $S^{-1}(z^i)$ is equal to getting z^i complexity.

Example 4. Geffe Generator is one of the simplest ways of combining LFSRs to output keystream and suffers from Siegenthaler correlation attack [9]. It uses three maximum length LFSRs R1, R2, R3, whose lengths respectively L_1, L_2 and L_3 are pairwise relatively prime. Let a_i represent the individual output of *i*'th LFSR. Then output of the generator at the moment *t* is $z(t) = a_1(t)a_2(t) \oplus a_2(t)a_3(t) \oplus a_3(t)$. The weakness of the Geffe generator comes from the fact that

$$P(z(t) = a_1(t)) = P(a_1(t) = 1) + P(a_2(t) = 0)P(a_1(t) = a_3(t)) = \frac{1}{2} + \frac{1}{2}\frac{1}{2} = \frac{3}{4}.$$

In a similar way $P(z(t) = a_3(t)) = 3/4$. Thus, we can write $P(z(t) = a_1(t) = a_3(t)) = 1/2 + 1/4 = 3/4$. As it can be seen that there is a serious bias as $\hat{\varepsilon} = 1/4$. With those correlations, if the primitive polynomials only have three terms each, and the largest LFSR is of length *L*, the current states of all three LFSRs can be recovered on a captured segment of the output sequence 37*L*-bits long [10]. Also computation expense of the attack is 896*L* bit operations.

Now suppose we apply the presented model for the Geffe generator. We use (2) to derive time complexity of the system. Assume error-code has parameters [*n,k,t*]. For this case $\tau = \lfloor 37L/n \rfloor$ and $l = 37L - n\tau$. Lets give a simple example to see the security effect of the proposed model on Geffe generator. Suppose L_1=23, L_2=29 and L_3=31 with satisfying above polynomial requirements and code parameters [*n,k,t*]:[15,7,2] are used. Then required keystream amount becomes 37*L*=1147 bits. From there, $\tau = \lfloor 37L/n \rfloor$ and *l*=7, so time complexity of the attacks becomes $T \cdot \binom{15}{2}^{76} \sum_{i=0}^{2} \binom{7}{i} = T \cdot 2^{515.19}$, where *T* is complexity of the attack without adaptation of the model. Therefore for this example we can say that the proposed model increases security of the generator $2^{515.19}$ times more and the attack becomes impractical.

IV. Cons And Pros

In the previous section, we examine the security contribution of our model by showing how some attack scenarios change. It is presented that the model improves security of the ciphers although some exceptional cases should be found such that security enhancement is not considerable. Exceptions may exist because attacks on some ciphers require very few keystream bits or some weakness in the stream cipher gives extra clue to the attacker to verify his success; such as the ORYX stream cipher is cryptanalyzed easily in [6] due to this reason. Of course besides security, a stream generator should be applicable and satisfy some requirements of practical communication. From this perspective lets see some disadvantages of our model. For an [*n,k*] linear block code decrease in output rate of the stream generator with a factor of *k/n* is obvious, because *k* bits of plainextex block is expanded to *n* bits of codeword block. It is one of major disadvantages of the model. Also, we know that a TRNG must be used in the system which may increase space complexity for a hardware based design and may be a problem regarding speed. Another issue is integration of code encoding/decoding functions may reduce speed of the cipher.

Nevertheless, possible solutions and trade-offs for these problems can be provided. First of all, *k/n* rate is a trade-off between throughput loss and security enhancement, since as rate decreases the error correcting capacity of the codes increases naturally, i.e. security contribution increases and throughput of the generator decreases. So the cryptographer should consider this trade-off with obeying the design requirements. Furthermore it is not necessary that TRNG has to work parallel with FSM part of the model. Because noise sequence generated by TRNG is independent from the secret key and also from receiver. So noise sequences should be generated previously and stored in a pool. In addition, we know that many communication systems use FEC to avoid noisy environment of channel. Hence, one can consider modulation, coding and cryptography based on this model at the same time and integrate them to increase optimization.

V. New Fec Assessment Criteria

Error correcting codes are evaluated in terms of their information rates and error correcting capabilities. The former is given by the ratio *k/n* where *k* is the dimension and *n* is the length of a code whereas the latter assessed by the relative minimum distance given as *d/n*. A code of minimum distance *d* can correct up to $t = \lfloor (d-1)/2 \rfloor$ errors in a block. On the other hand, the correcting capacity of the codes used for encryption via noisy-keystream generators can be evaluated in a different manner since the errors are introduced on purpose. Therefore, any vector can be decodable if there exists a unique code word which is closest the vector. For an $[n,k,d]_{F_q}$ code *C*, define the number of correctable vectors as

$$f = \#\left\{x \in F_q^n : \exists y_0 \in C \therefore d(x,y_0) < d(x,y) \forall y \neq y_0, y \in C\right\}.$$

Remark that *f* is the number of all vectors that can be corrected. An obvious lower bound for *f* is given as $\sum_{i=0}^{t} \binom{n}{i} \leq f$. Hence, it is expected that a large minimum distance results in a large *f*. However, it is not always true that a *d*-good code is also a *f*-good code. A code having larger minimum distance may have a smaller *f*.

The number of correctable codes with the block length of an error correcting code used in a noisy-keystream encryption plays a crucial role in enhancing the security level of the deterministic key stream generator. Because, the number of correctable codes and the block length both determine the entropy of the nondeterministic keystream generator. The entropy will be log *f* for each block. Hence any nondeterministic keystream of length *N* has an entropy of $\left\lfloor \frac{N}{n} \right\rfloor \log f$. This quantity determines the uncertainty of the keystream and hence the amount of the additional difficulty in analyzing the deterministic keystream generator. We arrive at a new definition of a ``good code" in terms of *t*. A good code is one having the relative number of correctable codes, *f/n* as large as possible for a given fixed information rate. It is a question how to construct such good codes.

VI. Conclusion

In this study, we presented a new approach for keystream generation of stream ciphers. The crucial idea behind the model is construction of a nondeterministic keystream sequence which is unknown to other parties initially, however the receiver can still obtain plaintext by using FEC. By means of the proposed model, we showed that security of stream ciphers can be improved and some known plaintext attacks can become ineffective. We consider two main directions for research in this area. First, specs of "good codes" for this model can be defined and new codes can be introduced with considering trade-off between code efficiency and security contribution. Second, it is known that FEC is an important tool to avoid noise in communication. Therefore in design process of a device modulation, coding and cryptography can be handled together and an appropriate model for communication can be determined. As a result security, speed, performance etc. should be addressed at the same time.

References

[1] G. Vernam, "Cipher Printing Telegraph Systems for Secret Wire and Radio Telegraphic Communications", *Journal American Institute of Electrical Engineers*, vol. 55, 1926, pp. 109-115.

[2] A. Menezes, P.C. van Oorschot, and S. A.Vanstone, *Handbook of Applied Cryptography*. CRC Press Inc., 1997.

[3] S. Babbage, "A Space/Time Trade-Off in Exhaustive Search Attacks on Stream Ciphers", *In: European Convention on Security and Detection IEE Conference Publication*, no. 408, 1995.

[4] J. Golic, "Cryptanalysis of Alleged A5 Stream Cipher", *In EUROCRYPT 1997, Lecture Notes in Computer Science*, vol. 1233, 1997, pp. 239-255.

[5] A. Biryukov and A. Shamir, "Cryptanalytic Time/Memory/Data Trade-Offs for Stream Ciphers", *In ASIACRYPT 2000, Lecture Notes in Computer Science*, vol. 1976, 2000, pp. 1-13.

[6] D. Wagner, L. Simpson, E. Dawson, J. Kelsey, W. Millan and B. Schneier, "Cryptanalysis of ORYX", *In SAC 1998, Lecture Notes in Computer Science*, vol. 1556, 1998, pp. 296-305

[7] E. Biham and O. Dunkelman, "Cryptanalysis of the A5/1 GSM stream Cipher", *In INDOCRYPT 2000, Lecture Notes in Computer Science,* vol. 1977, 2000, pp. 43-51.

[8] T. Siegenthaler, "Decrypting a Class of Stream Ciphers Using Ciphertext only", *IEEE Trans. on Comp.*, vol. C–34, 1985, pp. 81-85.

[9] P. R. Geffe, "How to Protect Data with Ciphers that Are Really Hard to Break", *Electronics*, vol. 46, no. 1, 1973, pp. 99-101.

[10] K. C. Zeng, C.-H Yang, C.–H. Wei and T. R. N. Rao, "Pseudorandom Bit Generators in Stream–Cipher Cryptography", *IEEE Computer*, vol. 24, no.2, 1991, pp. 8-17.