Resynchronization Weaknesses in Synchronous Stream Ciphers

Joan Daemen, René Govaerts and Joos Vandewalle

Katholieke Universiteit Leuven, Laboratorium ESAT
Kardinaal Mercierlaan 94, B-3001 Heverlee, Belgium
email: joan.daemen@esat.kuleuven.ac.be

Abstract. In some applications for synchronous stream ciphers, the risk of loss of synchronization cannot be eliminated completely. In these cases frequent resynchronization or resynchronization upon request may be necessary. In the paper it is shown that this can lead to significant deterioration of the cryptographic security. A powerful general attack on nonlinearly filtered linear (over $\mathbb{Z}_2$) systems is presented. This attack is further refined to efficiently cryptanalyze a linear system with a multiplexer as output function.

1 Introduction

Synchronous stream ciphers have the advantage that digit-value errors in the ciphertext only affect the corresponding digits in the plaintext, i.e. there is no error propagation due to the decryption process. The price paid for this property is that perfect synchronization is required between sender and receiver. If synchronization is lost, the output of the decryptor is unintelligible for the receiver and synchronization has to be regained.

If the plaintext has enough redundancy to identify correctness of decryption, the resynchronization (resync) can be performed by the receiver without the intervention of the sender. This process involves trying all possible offsets between the sender's and the receiver's clock. In applications where high-speed online decryption is needed, automatic synchronization recovery can be technologically infeasible. The risk of synchronization loss (SL) can be reduced somewhat at the cost of inserting synchronization patterns in the ciphertext. On the other hand the consequences of SL have to be limited. This can be achieved by performing resync at fixed timesteps (*fixed resync*). For instance, if resync occurs every M digits, SL will on the average result in the loss of M/2 digits. The appropriate frequency of resync depends on the probability of SL and the nature of the application. If there is a channel from the receiver to the sender, the receiver can request resync (*requested resync*) to the sender upon detection of SL.

The aim of resync is to ensure that encryptor and decryptor have the same internal state at a certain time. An internal state different from all previous resync states has to be chosen, to prevent the re-use of keysequences. For stream ciphers whose security is (partly) based on the secrecy of the internal state, the resync state may not be communicated in the clear at the time of resync. Since

T. Helleseth (Ed.): Advances in Cryptology - EUROCRYPT '93, LNCS 765, pp. 159-167, 1994.

the reason for resync is the possible unavailability of the stream encryption, this new internal state has to be encrypted with an additional cipher. This introduces extra complexity and cost into the system.

A more elegant (and cheap) solution is to specify the new internal state in terms of information that is only known to the sender and receiver. In this case sender and receiver calculate the new internal state from the original internal state (or key) and a public parameter (e.g. the time at the moment of resync) using a publicly known algorithm. At the time of resync no confidential information has to be transmitted, hence no additional cipher system is required. The subject of this paper is the impact of this form of resync on the cryptographic security of practical stream ciphers.

In the following section we will define the different components of a stream cipher system with resynchronization. The term *key secure* is introduced and defined with respect to several systems that differ in their resync abilities. Section 3 treats the cryptographic security of nonlinearly filtered linear systems. It contains a general attack and an attack that is specific for a multiplexer output function.

2 The Cryptographic Setting

Stream encryption is performed by encrypting the plaintext digit by digit. The plaintext digit at time t denoted by x^t is transformed into a ciphertext digit y^t by a transformation $f_e()$. This transformation depends on a keystream digit z^t and must be invertible with respect to x^t. In all practical systems the ciphertext, plaintext and keystream digits belong to the same alphabet, denoted by $\mathcal{A}$. We have

$$\begin{aligned} y^t = f_e(x^t, z^t) \quad &\text{and} \quad x^t = f_d(y^t, z^t) \\ \text{with} \quad & x = f_d(f_e(x, z), z), \ \forall x, z \in \mathcal{A} \end{aligned} \tag{1}$$

For the majority of designs the digits are bits. In this case (1) can be simplified to (with $\oplus$ denoting XOR)

$$y^t = x^t \oplus z^t$$

In most other designs the plaintext, keysequence and ciphertext bits are lumped into m-bit digits and the encryption can be described by

$$y_i^t = x_i^t \oplus z_i^t \tag{2}$$

where $x^t = (x_1^t, x_2^t \ldots x_m^t)$, $y^t = (y_1^t, y_2^t \ldots y_m^t)$, $z^t = (z_1^t, z_2^t \ldots z_m^t)$ and $\oplus$ denotes bitwise XOR.

2.1 The Synchronous Stream Encryptor/Decryptor

In *synchronous* stream ciphers the keystream digits z_i are generated independently of the message stream by a pseudorandom sequence generator(PSG). This is a finite state machine whose operation is governed by the two rules

$$s^{t+1} = \mathrm{F_s}(s^t, p) \tag{3}$$

$$z^t = \mathrm{f_o}(s^t, p) \tag{4}$$

with s^t the internal state at time t and p an optional parameter that is fixed during pseudorandom sequence generation. The finite state machine is initialized by loading the *initialization vector* v into the stateregister and by loading the parameter p. In hardware this loading mechanism requires additional circuitry. Therefore the initialization is sometimes integrated into the state-transition function:

$$s^{t+1} = \mathrm{F_s}(s^t, p, e) \tag{5}$$

where the e is an external input that is used to 'load' the initial state η digits at a time. Initialization is performed by resetting the internal state to a fixed value and subsequently iterating the finite state machine a specified number of times. With this mechanism the initialization vector v is defined as the array of η-digit e-inputs during the initialization process. During pseudorandom sequence generation this e-input must be constant and the state-transition can be modeled by (4). The time origin is defined by the initialization, hence the keysequence z looks like $z^0 z^1 z^2 \ldots$ and the initial state is s^0.

The cryptographic security of the stream cipher can be based entirely on the secrecy of (part of) the initial state, entirely on the secrecy of the parameter p or on the secrecy of both. This secret information is generally referred to as the *key* K. The dependence of the initialization vector v and the parameter p on the key K can be formally expressed by a function $\mathrm{F_k}()$:

$$(v, p) = \mathrm{F_k}(K) \tag{6}$$

In this paper the specification of a PSG consists by definition of the three functions $\mathrm{F_s}(), \mathrm{f_o}()$ and $\mathrm{F_k}()$.

In general the cryptographic security that is required from a cipher system depends on the application. Therefore, *cryptographic security of a cipher system* can best be defined as security in the worst possible circumstances. Clearly, a cipher that is claimed to cryptographically secure by this definition, is claimed to be secure in all applications. We introduce the term *key secure* to denote cryptographic systems whose security is *indicated* by the entropy of the key K.

Definition 1. A pseudorandom sequence generator is *key secure* if for any a priori distribution Ω of K, the cheapest way of using any known part of the keysequence to obtain knowledge about the other part of the keysequence includes complete determination of the specific value of K by exhaustive key search (with respect to Ω).

This definition implies that the cryptanalist has only access to the output of the PSG. The term *cheap* must be seen in the context of the real world where hardware, software, manpower etc. are for sale.

2.2 The Resync Mechanism

The initialization vector v and the parameter p for a given resync depend not only on the key but also on a publicly known randomization variable ρ by a publicly known *randomization function* $\mathrm{F_r}()$:

$$(v_i, p_i) = \mathrm{F_r}(K, \rho) \tag{7}$$

In most designs the parameter p is independent of ρ. If resync is performed on fixed timesteps, ρ stands for the serial number of the resync. For requested resync ρ can be the time given by a commonly available clock or a string that is sent by the receiver together with the request. The set of all possible ρ inputs is denoted by $\mathcal{R}$. If resync is performed it can be advantageous to do a number (say μ) of 'blank' iterations of the PSG after the loading of the initial state. During these iterations no pseudorandom digits are given at the output. This can be modeled by removing the first μ digits of the keystream: $z = z^{\mu} z^{\mu+1} z^{\mu+2}$. The resync mechanism (RM) consists of the randomization function $\mathrm{F_r}()$ and the constant μ.

2.3 Fixed Resync Setting

In a practical situation the security of an encryption system has to be evaluated in its totality. In a fixed resync application the stream encryption system consists basically of 3 components: the PSG, the randomization function $\mathrm{F_r}()$ and the total number of resyncs, denoted by ℓ.

In this setting the keysequence is no longer the result of iterating the PSG starting from a single initial state. After every resync a 'new' keysequence is started. Since these keysequences are the result of initialization vectors v_i (and parameters p) that are computed from the same key, they are not independent. It will be advantageous to describe the keysequences (that are in fact successive in time) as emerging simultaneously from a number of finite state machines. The initialization vector and parameter of each of these machines is given by the RM. For the i-th machine this is expressed as

$$(v_i, p_i) = \mathrm{F_r}(K, \rho_i) \Longrightarrow z_i : z_i^{\mu} z_i^{\mu+1} z_i^{\mu+2} \ldots \tag{8}$$

The definition of key security of a PSG can be adapted in a natural way to this system:

Definition 2. A fixed resync stream encryption system is *key secure* if for any a priori distribution Ω of K, the cheapest way of using any known part of the keysequences z_i to obtain knowledge about the other part of the keysequences z_i includes complete determination of the specific value of K by exhaustive key search (with respect to Ω).

The parallel representation emphasizes that not one but ℓ digits per clockcycle are available to the cryptanalist. This can possibly be exploited in a key-reconstruction algorithm. On the other hand, possible correlations between the different keysequences can be exploited to predict some sequences from other ones.

2.4 Requested Resync Setting

To investigate the cryptographic security of an encryption system in a requested resync application only the PSG and the RM are considered. The number ℓ is no longer given in advance. If the cryptanalist can impersonate as the receiver, he (or she) can request resync with a *chosen* ρ. Say the set of all possible keysequences for a given key is denoted by Λ_k. We have

$$\Lambda_k = \{z : \mathrm{F}_r(K, \rho) \Longrightarrow z \text{ with } \rho \in \mathcal{R}\} \tag{9}$$

We define the key security of a requested resync system as follows:

Definition 3. A requested resync stream encryption system is *key secure* if for any a priori distribution Ω of K, the cheapest way of using any known part of all $z_i \in \Lambda_k$ to obtain knowledge about the other part of all $z_i \in \Lambda_k$ includes complete determination of the specific value of K by exhaustive key search (with respect to Ω).

It is clear that a system that is key secure in a requested resync application, can be used to make a key secure fixed resync application.

2.5 Resync Security of a PSG

Stream ciphers proposals in the literature usually contain a specification of the PSG and no specification of a RM. If one is confronted with the problem of choosing a PSG as a component of a fixed or requested resync stream cipher system, the function $\mathrm{F}_k()$ has to be replaced by an appropriate RM.

A RM consists of a randomization function $F_r(K, \rho)$ and a positive integer μ. The function $F_r(K, \rho)$ has to fulfill the following two criteria to be regarded as a valid randomization function.

- $\rho_1 \neq \rho_2 \Rightarrow F_r(K, \rho_1) \neq F_r(K, \rho_1)$: different public arguments must not give rise to equal initialization vectors v *and* parameters p.
- $k_1 \neq k_2 \Rightarrow F_r(K_1, \rho) \neq F_r(K_2, \rho)$: different keys must not give rise to equal initialization vectors v *and* parameters p.

At the best, the specific choice of the RM has no influence on the security of the system. At the worst, no secure system can be built around the PSG without choosing a randomization function that has certain cryptographic properties on its own. For the complexity of the overall system it is advantageous to have a RM as simple as the PSG allows.

For a given PSG classes in the space of all possible RM can be identified that give rise to loss of security. The result of this process can be a belief or claim that a PSG is *resync key secure* with respect to a subclass Ξ of all possible RM, as defined in

Definition 4. A PSG is *resync key secure* with respect to Ξ if the requested resync system composed of the PSG and all possible RM in Ξ are key secure.

The statement that a PSG is key secure according to Def.1 is equivalent to the statement that it is resync key secure with respect to the resync mechanism where $F_r()$ is defined by $F_k()$ and $\mathcal{R} = \emptyset$. Because the randomization function has no public argument, no resync can be applied in this case.

2.6 Packet encryption with Resync Key Secure PSGs

The ease and security of resync in a resync key secure PSG allows the use of synchronous stream encryption even for small data packets. Every time a new packet is encrypted the PSG is resynchronized with a unique parameter such as the packet identifier (that is not encrypted) as public parameter. If the PSG is in fact resync key secure, the security of the system is guaranteed by the size of the keyspace.

3 Attacks on Nonlinearly Filtered Linear Systems

As an illustration the effect of resync on a popular class of PSGs is examined. In this class the state-transition function is linear over $\mathbb{Z}_2$ and the nonlinear output function is fixed. The cryptographic security is based on the inability to reconstruct the internal state from the output of the nonlinear output function. We suppose that a linear randomization function is used. In the parallel representation introduced in Sect. 2.5 the operation of the i-th system can be described by matrix equations:

$$s_i^0 = AK \oplus R_i \tag{10}$$

$$s_i^{t+1} = Fs_i^t \tag{11}$$

Where A and F are binary matrices. Say the number of statebits is n. It is supposed that F is nonsingular (as it is in all practical proposals). R_i is a vector that is fixed by ρ_i in (8). The attack focuses on the reconstruction of the entire internal state for a certain PSG at a certain moment in time: s_p^τ. From this internal state, s_p^0 can be computed and subsequently all other initial states s_i^0. This knowledge is sufficient to calculate all keysequences z_i.

$$s_p^0 = F^{-\tau} s_p^\tau \tag{12}$$

$$s_i^0 = s_p^0 \oplus R_i \oplus R_p \tag{13}$$

The difference modulo 2 of any two simultaneous states s_i^t and s_j^t can be calculated:

$$s_i^t \oplus s_j^t = F^t(R_i \oplus R_j) \ . \tag{14}$$

The output function computes the keysequence digit from the internal state. Since in most practical designs the keysequence digits are bits, we will consider only binary output functions. If m-bit keysequence digits are produced, the output function can be decomposed in its binary components.

In most practical proposals, the keysequence bit only depends on a subset of statebits. Suppose the values of these statebits are collected in the binary vector u with φ components. The vector u is a projection of the internal state s. This can be expressed in a matrix equation

$$u_i^t = \mathrm{G}s_i^t \ . \tag{15}$$

The linear relations between the internal states can be exploited to strengthen existing attacks or to construct new attacks.

3.1 A Simple General Resync Attack

Suppose the vector u_p^τ can be completely reconstructed for a certain τ. This gives the cryptanalist φ bits of s_p^τ. Since s_p^τ depends on s_p^0 by $\mathrm{F}^\tau s_p^0 = s_p^\tau$ this results in φ linear equations for the bits of s_p^0. If a number m of vectors u_p^t can be reconstructed, this results in $m\varphi$ (not necessarily linearly independent) linear relations for the bits of s_p^0. If n independent linear relations are found s_p^0 can be fixed. Therefore reconstructing $\lceil \frac{n}{\varphi} \rceil$ u_p^t vectors is sufficient if all relations are independent. In the rare case that there are too much linear dependencies, more u-vectors can be reconstructed until n independent linear relations are found. The resulting set of equations can be efficiently solved by methods of linear algebra.

Now we are left with the problem of reconstructing u_p^t for some t. (In the following of this section we will omit the superscript t.) Suppose z_p is known to the cryptanalist. The correct u_p has to fulfill $\mathrm{f_o}(u_p) = z_p$. For every z_i that is known to the cryptanalist there is such a relation. We have $\mathrm{f_o}(u_i) = z_i$ and $u_i = u_p \oplus \mathrm{G}(\mathrm{R}_i \oplus \mathrm{R}_p)$. Substitution yields

$$\mathrm{f_o}(u_p \oplus \mathrm{G}\mathrm{F}^t(\mathrm{R}_i \oplus \mathrm{R}_p)) = z_i \tag{16}$$

for any z_i that is known to the cryptanalist. This is a set of nonlinear boolean equations with φ variables. If the number of equations is larger than φ, the number of solutions is expected to converge to 1. If φ is small enough, the solution can be found by performing exhaustive search over the space of all possible u-vectors. This involves on the average 2^φ evaluations of $\mathrm{f_o}()$. If a certain bit of u only appears in the function $\mathrm{f_o}()$ in linear combination with another bit, the absolute values of these bits can not be deduced from the above equations. However, their XOR can be deduced, yielding a linear relation. This can be converted to a linear relation for the initial state without a problem. Since the output function only depends on the XOR of the two bits and not on the bitvalues itself, only $2^{\varphi-1}$ inputs have to be tested.

For this attack to be possible, the number of resyncs has to be larger than φ. The cryptanalist must know at least φ keysequence bits for a number of $\lceil \frac{n}{\varphi} \rceil$ timesteps. The expected workload is

$$\left\lceil \frac{n}{\varphi} \right\rceil 2^\varphi \tag{17}$$

evaluations of $f_o()$ and some additional linear algebra computations. The attack can easily be parallelized by distributing the search over a number of different processors. It can be observed that the complexity of this attack only grows linearly with the size of the internal state. For this general attack to be infeasible, it is essential that the output function depends on a large number of statebits.

Say we have a PSG with a linear state-transition and a nonlinear output function. The evaluation of the resync security of such a PSG for linear randomization functions boils down to the analysis of the output function. The cryptanalytic problem is constructing an algorithm that can extract a vector u in feasible time and space if $f_o(u \oplus e)$ can be asked for all possible binary vectors e. From the definition exhaustive key search must be the most efficient attack for key secure stream ciphers, hence the complexity of this algorithm imposes an upper limit on the size of the keyspace for a key secure system.

The general attack works independent of the specific properties of the output function. In the following section we will show that a significant speedup can be attained in cases where the structure of the $f_o()$ can be exploited.

3.2 A Simple Specific Resync Attack

Consider the multiplexer generator [1]. This is a linear finite state machine with a multiplexer as output function. A multiplexer with β address inputs has 2^β data inputs. If the address inputs are denoted by $a^0a^1 \ldots a^{\beta-1}$, the data inputs by $d^0d^1 \ldots d^{2^\beta-1}$ and the output by z, the operation of the multiplexer can be described by

$$z = d^a \text{ with } a = \sum a_i 2^i \ . \tag{18}$$

Here a is the interpretation of $a^{\beta-1} \ldots a^1a^0$ as an integer. The vector u_p can be split up in a_p and d_p. Since $u_i \oplus u_j$ is completely known for all i, j , so are $a_i \oplus a_j$ and $d_i \oplus d_j$. Suppose the keysequence bits z_p and z_i are known and that $a_p \oplus a_i = 0$. If $z_p = z_i$ the integer a_p (and equivalently a_i) must select a bit in d such that $d_p^{a_p} \oplus d_i^{a_p} = 0$. If $z_p \neq z_i$ a similar argument applies. In both cases about half of the possible values for a_p are eliminated. This can be repeated for every two keystream bits z_e, z_f where $a_e \oplus a_f = 0$. The correct value for a_p will be found after investigating β pairs (or a few more). If the correct value of a_p is found in this way, the remaining $2^\beta - \beta$ values of d_p can systematically be scanned by considering all keysequence bits z_{g_i} where $a_{g_i} = a_p \oplus i$ for all $0 \leq i < 2^\beta$.

The complete process of fixing the u-vector takes about $\beta + 2^\beta$ evaluations of the multiplexer function. This has to be repeated $\lceil \frac{n}{\varphi} \rceil$ times where $\varphi = \beta + 2^\beta$. The complexity of the complete attack can be approximated by

$$\left\lceil \frac{n}{\varphi} \right\rceil \varphi \approx n \tag{19}$$

output function evaluations and some additional linear algebra computations. In fact, the linear algebra will be responsible for the largest part of the workload in realistic cases.

The applicability of this attack is somewhat limited by the demands for the difference patterns in a. However, if the total number of resyncs is not significantly smaller than 2^β this is no problem. A multiplexer system that can be attacked by this scheme is currently recommended in [2] for video encryption. This system does 256 resyncs and has $\beta = 5$. In [3] we present a ciphertext-only attack that can be implemented in hardware for on-line decryption of the video component without the key.

4 Conclusions

It is pointed out that the application of resynchronization can jeopardize the security of a synchronous stream cipher. This is illustrated by a general attack on the class of nonlinearly filtered linear (over $\mathbb{Z}_2$) systems that comprises a large number of practical proposals. A specific attack is presented for linear systems where the output function is a multiplexer. Both attacks are conceptually very simple and show that applying resynchronization can result in total loss of security for a large number of published stream cipher proposals.

References

[1] R. A. Rueppel, 'Stream Ciphers', in *Contemporary Cryptology*, Gustavus J. Simmons Ed. IEEE Press, New York.

[2] 'Specification of the systems of the MAC/packet family'. EBU Technical Document 3258–E, Oct 1986.

[3] J. Daemen, R. Govaerts, J. Vandewalle, Cryptanalysis of MUX-LFSR Based Scramblers, in *Proceedings of SPRC '93*, 15–16 February, Roma. (to appear)