On the Role of the Inner State Size in Stream Ciphers

Erik Zenner

Theoretische Informatik
University of Mannheim (Germany)
e-mail: zenner@th.informatik.uni-mannheim.de

Abstract. Many modern stream ciphers consist of a keystream generator and a key schedule algorithm. In fielded systems, security of the keystream generator is often based on a large inner state rather than an inherently secure design. Note, however, that little theory on the initialisation of large inner states exists, and many practical designs are based on an ad-hoc approach. As a consequence, an increasing number of attacks on stream ciphers exploit the (re-)initialisation of large inner states by a weak key schedule algorithm.
In this paper, we propose a strict separation of keystream generator and key schedule algorithm in stream cipher design. A formal definition of inner state size is given, and lower bounds on the necessary inner state size are proposed. After giving a construction for a secure stream cipher from an insecure keystream generator, the limitations of such an approach are discussed. We introduce the notion of inner state size efficiency and compare it for a number of fielded stream ciphers, indicating that a secure cipher can be based on reasonable inner state sizes. Concluding, we ask a number of open questions that may give rise to a new field of research that is concerned with the security of key schedule algorithms.

Keywords: Stream cipher, keystream generator, initialisation, inner state.

1 Introduction

Background: Let $m = (m_1, m_2, \ldots)$ be a message consisting of blocks $m_t \in \{0,1\}^w$. A stream cipher is a pair of efficient algorithms, where encryption transforms a message block m_t into a ciphertext block $c_t \in \{0,1\}^w$ and decryption implements the inverse transformation. Both encryption and decryption run under the control of a key K and a counter t. Note that the use of a counter is the critical difference between a stream cipher and a block cipher.

A frequent building block for stream ciphers is a keystream generator, i.e. a finite state machine that transforms K into a pseudorandom bitstream $z = (z_1, z_2, \ldots)$ with $z_t \in \{0,1\}^w$. In most cases, z_t is added bitwise to m_t for encryption and to c_t for decryption.

While a large body of literature exists on the design of keystream generators (cf. [33, 27]), the remaining aspects of stream cipher design are less well researched. Only few guidelines exist for the choice of important parameters like key length, inner state size, or the number of bits produced before re-keying. The same uncertainty exists with respect to the key setup algorithm that transforms the key into a starting state for the generator.

The consequences in practical stream cipher design are twofold. On one hand, an increasing number of stream ciphers is broken not by attacking the keystream generator, but by attacking the key setup algorithm (e.g. RC/4 as used in the WEP protocol [35], or A5/1 from the GSM standard [17]). There exist a few general attack techniques against weak setup functions for stream ciphers exist (e.g. resynchronisation attacks [13, 21]), but no design criteria for good initialisation functions. Considering recent research progress on related key attacks for pseudorandom functions (see [6] and subsequent work), more problems for stream ciphers designed in an ad-hoc manner are to be expected in the future.

On the other hand, when a cipher is successfully attacked, a common solution is to change the parameters while keeping the general design intact. Examples include increasing the inner state size (e.g. for LILI-128 [9]) or decreasing the security level (e.g. for Sober-128 [25]). For some ciphers, huge security margins for the parameters are used in the first place (e.g. more than 33,000 bit of inner state for SEAL [32]).

Paper outline: This paper is intended as a starting point for future research on the design of stream ciphers. We consider the construction of such ciphers from two primitives: a keystream generator and a matching initialisation function. Observe that the inner state of the cipher forms the interface between those two primitives. While a large inner state is advantageous for the security of the keystream generator, it makes the task of the initialisation algorithm more difficult. Thus, the inner state should be chosen as large as necessary, but as small as possible.

Our goal is to improve the understanding of the necessity and the limitations of the inner state. To this purpose, in section 2, we introduce a formal model of the stream ciphers considered. In section 3, we derive a formal definition of the inner state size, along with an illustration why such a definition is not as trivial as it may seem. Section 4 discusses the cryptographic relevance of inner states, giving lower bounds on the minimum size as well as a construction for a secure stream cipher when inner state size and initialisation time are not critical. In section 5, we observe that the inner state size has to be limited in most practical stream ciphers. This leads us to the definition of inner state efficiency, yielding a measure of how much the inner state size actually contributes to the security of the keystream generator. We also give concrete values of inner state efficiency for a number of practical stream ciphers. Concluding, in section 6, our results are summarised, and a list of open questions for future research on stream cipher design is presented.

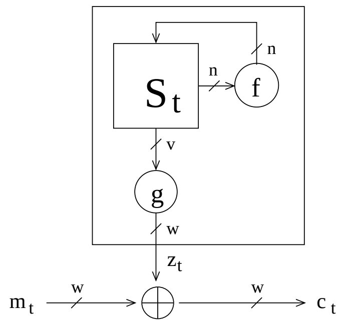

Fig. 1. General model of a keystream generator

2 Terminology

2.1 Keystream generators

Basic model: In [33], Rueppel defined a *keystream generator* as consisting of the following components (see the box in figure 1):

(a) An inner state $S_t \in \mathcal{S}$ with $\mathcal{S} \subseteq \{0,1\}^n$,
(b) an update function $f : \mathcal{S} \to \mathcal{S}$ that modifies the inner state with each clock, and
(c) an output function $g : \{0,1\}^v \to \{0,1\}^w$, $w \leq v \leq n$, that uses the inner state to compute w keystream bits with each clock.

Deployment in stream ciphers: Keystream generators are often used in cryptography to implement efficient stream ciphers. A wide-spread design requires the following additional components:

(A) A secret *key* $K \in \{0,1\}^l$ that is not necessarily identical to the inner state,
(B) an *initialisation function* $h : \{0,1\}^l \times \{0,1\}^m \to \mathcal{S}$ that derives the initial inner state S_0 from the key K and m bit of additional information (like an initial value or a nonce), and
(C) the *xor-function* $\oplus : \{0,1\}^w \times \{0,1\}^w \to \{0,1\}^w$ which adds the keystream bitwise modulo 2 to the plaintext, generating the ciphertext[1].

2.2 Attacker model

Prior knowledge: The attacker is assumed to know everything about the stream cipher with the exception of the key K and the current inner state S_t. In particular, he is aware of the set $\mathcal{S}$ of possible inner states and of the functions f, g and h. He also knows (or even controls) the m bit of public information that are used, along with the key, to set up the inner state S_0.

[1] In fact, other functions like addition modulo 2^w are also possible, but rarely used.

Class of attack: We consider a known-plaintext attack. Since knowledge of plaintext and ciphertext implies knowledge of the keystream, it can be assumed that the attacker has $L \ll 2^l$ known keystream bits at his disposal.

Computational resources: The attacker can do any computation that requires less steps than a complete search over the key space. This bound holds both for the precomputation and the actual attack phase, implying a bound of strictly less than 2^l bits of memory that can be used.

Notion of success: An attacker is considered successful if he can correctly predict previously unknown keystream bits, or if he can distinguish the output of the keystream generator from truly random bits. We say that the keystream generator is broken if there exists an attacker whose success probability differs significantly from pure guessing. Note that in particular, reconstruction of a correct tupel (t, S_t) or of the key K implies a successful attack in the above sense.

3 Defining the size of inner states

3.1 Problem illustration

Surprisingly, defining the size of the inner state is not as trivial as it may seem. Consider the following examples as an illustration.

SOBER-128: This nonlinear filter generator proposed by Hawkes and Rose [25] is the ideal case; the inner state consists of a 17 stage LFSR over GF(2^{32}) with primitive feedback polynomial. Thus, all $(2^{32})^{17} = 2^{544}$ inner states with the exception of the all zero state can be attained over time if the generator is run long enough without re-keying. Thus, there is little doubt that the inner state has size 544 bit.

RC4: The algorithm that is generally assumed to be Rivest's RC4 generator [29] takes some more thought. At first glance, the 8-bit version uses an inner state table that consists of 256 bytes, along with two 8-bit variables, yielding an inner state of $2048 + 16 = 2064$ bit.

However, this table is used to store permutations over $\mathbb{Z}_{256}$, which reduces the number of possible table states to $256! \approx 2^{1684}$. Thus, it could be argued that the inner state is about $1684 + 16 = 1700$ bit long.

If, however, the size is defined by the number of states that can actually be attained, things get even more complicated. The initial values of the 8-bit variables are key-independent, and it was demonstrated by Finney [18] that an easily characterised class of inner states can never occur. Thus, the inner state size lies somewhere between 1684 (derived from the number of valid starting states) and 1700 (derived from the number of representable states), where none of the bounds is tight.

SEAL 3.0: The generator proposed by Coppersmith and Rogaway [32] uses two different kinds of inner states. On one hand, there are 8 32-bit variables and 12 counter bits that change constantly over time. Since they can in principle attain all possible values, they contribute 268 bit to the inner state size.

On the other hand, however, the algorithm uses huge lookup tables R, S and T with a total size of 32,768 bits. These tables are generated from the 160-bit key and a 32-bit nonce using a hash function in counter mode. Since they are never modified during keystream generation, only 2^{192} different assignments to the tables are possible. One might be tempted to state that the tables contribute only 192 bit to the size of the inner state, but then again, no efficient algorithm is known that distinguishes a valid table setting from an invalid one. This means that in practice, a possible attacker faces the full state space of 33,036 bits.

Adding to the conceptional confusion, one table contains values that are used only once during the encryption process. Thus, given enough processing time, the corresponding entries could be calculated as need arises, making it possible to replace a 8,192 bit table by a simple 6-bit counter in an algorithmic implementation. This raises the question of whether or not the inner state size is reduced, too.

3.2 Autonomous finite state machines

A naïve candidate for the inner state size is the parameter n, as described in section 2.1. This parameter denotes the length of the inner state representation. There is, however, the obvious problem that the same generator may be represented in different ways, yielding different values of n depending on the concrete implementation.

Instead, in order to derive a unique definition of the inner state size, we consider an *autonomous finite state machine* (AFSM) implementing the generator. Such an AFSM consists of a set $\mathcal{S}$ of inner states, and for each inner state $S \in \mathcal{S}$, there exists

- a *transition rule* that defines the next state $f(S)$ for S, and
- a *label* defining the output $g(S)$ generated from S.

In addition, each finite state machine needs a set $\mathcal{S}_0$ of valid *starting states*.

Note that there exists an infinite number of AFSM describing the generator. In particular, the size of the AFSM (i.e. the number of inner states) can vary arbitrarily. Thus, in order to find a unique value for the number of inner states, we need to revert to the notion of the *minimal AFSM* describing the generator.

An AFSM is said to *generate* an (infinite) output sequence $z = (z_0, z_1, \ldots)$ if there exists a starting state $S_0 \in \mathcal{S}_0$ such that $z_i = g(f^i(S_0))$. Two AFSM A and B are said to be *equivalent* if all (infinite) output sequences produced by A are also produced by B, and vice versa. As a consequence, all AFSM that describe a given keystream generator are equivalent. An AFSM is said to be *minimal* if no equivalent AFSM of smaller size exists. Thus, if a minimal AFSM for a given generator can be found, its size yields the minimal number of inner states required to implement the generator.

3.3 Valid starting states

The size of a minimal AFSM for a given generator depends on the set $\mathcal{S}_0$ of valid starting states. Consider, as a toy example, a 2-bit version of the RC4 generator, where the inner state consists of two 2-bit variables and a table representing a permutation over $\{0, 1, 2, 3\}$.

1. If we allow all assignments to the two 2-bit variables and all assignments to a 4×2-bit table as initial states, we end up with a minimal AFSM with $2^{4+8} = 4096$ inner states, all of which are starting states.
2. If we allow all assignments to the variables, but restrict the table entries to correct permutations, the minimal AFSM will have $2^4 \cdot 4! = 384$ inner states, all of which are starting states.
3. If we initialise the variables to zero (as we should for a correct RC4 implementation), we have only $4! = 24$ starting states left, and exactly 24 states are no longer reachable. Thus, the size of the minimal AFSM drops to 360.
4. If we take the initialisation function h and the key length l into account, the number of starting states may even be smaller, which in turn may or may not affect the size of the minimal AFSM.

Note that the inner state size should depend only on the keystream generator. Thus, the initialisation function must not be considered when defining $\mathcal{S}_0$, discarding case 4. However, what should be known is the interface between the keystream generator and the initialisation function: A set of conditions that the output of *any* initialisation function must meet in order to guarantee the correct working of the generator. In the case of 2-bit RC4, those conditions would be the ones described in case 3: Both variables must be set to zero, and the table must contain an arbitary permutation over $\{0, 1, 2, 3\}$.

3.4 Final definition

Basic model, extended: A keystream generator consists of an inner state space $\mathcal{S}$, an update function, and an output function, as described in conditions (a) to (c) in subsection 2.1. However, it must also have an additional component, namely

(d) a Boolean predicates $C : \mathcal{S} \to \{0, 1\}$, such that an inner state S is a valid starting state iff $C(S) = 1$.

Analogously, condition (B) in subsection 2.1 must be corrected such that a stream cipher contains

(B) an *initialisation function* $h : \{0, 1\}^l \times \{0, 1\}^m \to \mathcal{S}$ that derives a starting state S_0 from the key K and m bit of additional information (like an initial value or a nonce), such that $C(S_0) = 1$.

Inner state size: Given the above definitions of a keystream generator, a (minimal) autonomous finite state machine and its size, we can define the unique inner state size of the generator as follows:

Definition 1. *Let G be a keystream generator as defined above, and let A be a minimal AFSM implementing G. Then the inner state size of the generator G is defined as $\hat{n} := \lceil \log_2(|A|) \rceil$, where $|A|$ is the number of inner states of A.*

4 Cryptographic strength from large inner states

4.1 The necessity of large inner states

In the following, let l denote the key length, $\hat{n}$ the inner state size and n the length of the inner state representation ($n \geq \hat{n}$). For most practical stream ciphers, it can be observed that $n > l$ holds. We will briefly discuss that this is in fact a necessary condition for secure stream ciphers.

First lower bound: The main design goals of practical stream ciphers are security and efficiency. In order to achieve the efficiency goal, the functions f, g and h are chosen to be as simple as possible. In particular, $g : \{0,1\}^v \to \{0,1\}^w$ is constructed such that $w \leq v < \min\{l, n\}$.

Lemma 1. *Let the output function g depend on $v < l$ inner state bits and let the output be balanced. Then the keystream generator can not be secure if $n < l + w$.*

Proof. For such a generator, a divide-and-conquer attack can be mounted: The attacker guesses all v bits of the inner state representation that are input to g (since $v < l$, this is feasible in our attack model). He then verifies whether the output of g matches the observed value z_0. Since g is balanced, only 2^{-w} of all assignments meet this criterion, strongly reducing the search space. The attacker can now mount a complete search over the remaining assignments, yielding an attack in 2^{n-w} steps. If $n < l+w$, this attack would be more efficient than brute force search over the key space of the stream cipher. □

Since the value n depends on the implementation and is thus not under the control of the cipher designer, the inner state size must be chosen such that the above attack becomes infeasible for all implementations.

Corollary 1. *If $v < l$, a necessary condition for a secure keystream generator is $\hat{n} \geq l + w$.*

Note that for many ciphers, this attack can be extended using a backtracking approach [22, 37, 36], yielding an even greater lower bound on the minimum size of the inner state.

Second lower bound: The requirement for a large inner state gets even stronger if the attacker has a large amount of keystream bits at his disposal. In this case, time-memory-data tradeoff attacks have to be taken into account, as follows.

Lemma 2. *Let L be the number of keystream bits available to the attacker. Then the keystream generator can not be secure if $n < l + \log(L)$.*

Proof. A general time-memory-data tradeoff [3, 22, 7] for $w = 1$ can be conducted as follows:

- *Precomputation phase:* The attacker draws a large sample (say, $2^{l-\epsilon}$) of inner states at random from $\mathcal{S}$. For each sample state S_i, the generator is run to produce an l-bit output z_i. The tuple (z_i, S_i) is stored in a table, indexed by z_i.
- *Attack phase:* The attacker segments the known output stream into roughly L overlapping frames $\tilde{z}_j$ of l bits[2]. For each frame, he checks whether $\tilde{z}_j$ is contained in the table, and if yes, he extracts the inner state S.

By the birthday paradoxon, there is high probability for a collision between the set of samples z_i in the table and the set of observations $\tilde{z}_j$ in the keystream if $2^{l-\epsilon} \cdot L \approx 2^n$. Since this attack requires $2^{l-\epsilon}$ precomputations and L table-lookups, it is feasible for the attacker if $n \approx l + \log(L) - \epsilon$, where ϵ is small.
Note that this proof can be generalised for arbitrary values of w by using frame lengths that are multiples of w, yielding the same result. □

Again, the cipher designer can not control n, but only the inner state size $\hat{n}$. Remembering that an attacker who is restricted to 2^l operations can read at most $L = 2^l$ output bits, we obtain the following lower bound:

Corollary 2. *If the generator produces arbitrarily large output streams, a necessary condition for a secure keystream generator is $\hat{n} \geq 2l$.*

4.2 A generic construction

We have seen that for efficient and secure stream ciphers, the inner state size $\hat{n}$ must be strictly larger than the key size l. An obvious question is: What happens if we increase $\hat{n}$ further? An interesting observation is that a large inner state can be used to make up for the deficiencies of a relatively weak keystream generator.

Constructing the stream cipher: Let $H = \{H_n \mid n \in \mathbb{N}\}$ be a family of cryptographically secure hash functions $H_n : \{0,1\}^* \to \{0,1\}^n$. Let $G = \{G_n \mid n \in \mathbb{N}\}$ be a family of keystream generators with $n = \hat{n}$.[3] Furthermore, let the generator be such that the mapping from state space to the first n keystream bits is bijective. Finally, we assume that there exists a known parameter c, $0 < c < 1$, such

[2] To be exact, there are $L - l + 1$ such frames.

[3] Many keystream generators are of that kind, e.g. many LFSR-based combination and filter generators or clock-controlled generators.

that for any generator $G_n \in G$ and given n bits of output stream, predicting additional output bits will require at least 2^{cn} computational steps for all but $O(1)$ cases.

Given these building blocks, we can construct a stream cipher with security level l as follows. First, choose n such that $c \cdot n > l$, and use G_n as keystream generator. The n bits of inner state for generator G_n are initialised using the matching hash function $H_n : \{0,1\}^l \times \{0,1\}^m \to \{0,1\}^n$.

Security against inversion: It can be shown that such a stream cipher is secure against inversion attacks, as long as no assumption about G_n and H_n is violated.

Lemma 3. *If the stream cipher (G_n, H_n) can be inverted in less than 2^l steps, then the hash function H_n can be inverted in less than $2^l + n$ steps.*

Proof. Assume that there exists an attacker A who, given the description of (G_n, H_n) and at least n bit of cipher output z, can invert the stream cipher in less than 2^l steps. Then we can construct an inverter A' who, given a valid output y of the hash function H_n, finds a corresponding input x such that $H_n(x) = y$.

- A' runs the keystream generator on inner state representation y, producing n bit of cipher output $z = G_n(y)$.
- A' invokes attacker A with input z and obtains a key k with $G_n(H_n(k)) = z$.
- A' outputs k.

Note that k meets the condition $G_n(H_n(k)) = z$. Since G_n is injective, there exists only one intermediate value y with $G_n(y) = z$, implying that $H_n(k) = y$. Thus, A' has inverted the hash function, using $2^l + n$ computational steps. □

Security against prediction: Analogously, it can be shown that the stream cipher is secure against prediction attacks, as long as the output of keystream generator G_n can not be predicted in less than 2^{cn} computational steps in all but a small number of cases.

Lemma 4. *If the stream cipher (G_n, H_n) can be predicted in less than 2^l steps, then the keystream generator G_n can be predicted in less than 2^l steps on a significant subset of its inputs.*

Proof. Assume that there exists an attacker A who, given the description of (G_n, H_n) and output bits $(z_0, \ldots, z_{n-1})$, can predict output bits $(z_n, \ldots, z_{n+d-1})$ correctly in less than 2^l steps. Then we can construct a trivial predictor A' who, given a valid output $(z_0, \ldots, z_{n-1})$ of G_n, can predict the subsequent output bits $(z_n, \ldots, z_{n+d-1})$ in at least 2^l different cases.

- A' runs A on input $(z_0, \ldots, z_{n-1})$ and obtains bits $(z_n, \ldots, z_{n+d-1})$.
- A' outputs $(z_n, \ldots, z_{n+d-1})$.

Note that due to the injectivity of G_n, $(z_0, \ldots, z_{n-1})$ was generated from a unique starting state S_0. For the analysis, we have to distinguish two cases:

(a) If S_0 is a possible output of H_n, the sequence $(z_0, \ldots, z_{n-1})$ is a correct output of the stream cipher (G_n, H_n). Thus, if A predicts correctly for the stream cipher, A' predicts correctly for the generator.
(b) If, however, no key k exists such that $H_n(k) = S_0$, the behaviour of A (and thus of A') is undefined - the prediction may or may not be correct.

In any case, the running time of A' is identical to the running time of A, yielding an effort of less than 2^l steps. Note that the algorithm is always right if case (a) occurs, yielding a correct prediction in at least 2^l (out of 2^n) cases. $\square$

5 Inner state efficiency

5.1 Disadvantages of large inner states

In fact, practical stream ciphers often use a relatively weak keystream generator and rely on the inner state size and the key schedule algorithm for security. Since constructing a cipher in the above way is tempting, why not use it as a general design rule?

With all their advantages as demonstrated in sections 4.1 and 4.2, there are also three arguments against large inner states. The first (and most obvious) one is that memory is not for free. While on a modern PC, sufficient memory should be available, other platforms like encryption hardware or smartcards may require a more economical use of resources. A second problem is that cryptographic memory must be protected from observation (both on general purpose and specialised hardware), and that an increase in memory size increases the options of an attacker, e.g. for side-channel attacks.

But there is third, more subtle reason why large inner states do not only provide advantages: most practical stream ciphers have to be re-initialised on a regular basis; i.e. after producing a fixed number of output bits, a new inner state is computed from the same key K, but with different additional information. This can be for synchronisation purposes, but also due to cryptographical reasons.[4] However, for a stream cipher with a large inner state, either performance or security of the initialisation procedure is impaired.[5] If the cipher re-initialises rather often, the function h must be computable in as few computational steps as possible. On the other hand, a good mixing of key and nonce into the starting state can not be obtained in a small number of computational steps. The lack of widely accepted design criteria for such initialisation functions further complicates the problem.

[4] Note that once the number of keystream bits available to the attacker gets large, most keystream generators become vulnerable to a wide range of cryptanalytic techniques, like time-memory-data tradeoffs, correlation attacks, differential attacks, or algebraic attacks.
[5] This line of research was first proposed to us by W. Meier [31].

Cipher		$l_{\max}$	$\hat{n}$	σ	γ
A5/1	[8]	64	64	32.0	0.5000
Lili-128	[14]	128	128	48.0	0.3750
E_0	[1]	128	132	49.0	0.3712
Sober-t32	[26]	256	544	158.0	0.2904
SNOW 1.0	[15]	256	576	100.0	0.1736
Scream	[24]	128	4,116	100.0	0.0243
RC4 (8bit)	[29]	256	1,700	30.6	0.0180
Leviathan	[30]	256	6,784	39.0	0.0057
Seal 3.0	[32]	160	33,036	43.0	0.0013

Table 1. Keystream generators of fielded stream ciphers (details: appendix A)

5.2 Efficient inner state size

For all of the above reasons, the inner state size must not be too large, even though a certain minimum size for the inner state is necessary, as shown in subsection 4.1. Note that the lower bound on the required inner state size depends on the quality of the keystream generator. In order to make comparisons between different generators possible, we introduce a measure of inner state efficiency, much in analogy to the well-known efficient key size.

Definition 2. *Let G be a keystream generator, and let A be the best known attack against G. The efficient inner state size of G is a number $\sigma \in \mathbb{R}$ such that executing A takes as many computational steps as a brute force search over 2^σ starting states of G. The quotient $\gamma = \sigma/\hat{n}$ is denoted as the inner state efficiency and is a measure for the quality of the keystream generator G.*

5.3 Comparison of fielded stream ciphers

For an arbitrary generator, the inner state efficiency lies in the range of $0 \leq \gamma \leq 1$. However, using the time-memory-data tradeoff technique presented in subsection 4.1, it can be shown that for a generator that produces at least $2^{\hat{n}/2}$ output bits, the inner state efficiency is restricted to $0 \leq \gamma \leq 0.5$. But what values of γ are encountered in practice?

In table 1, the efficiency δ of inner states is compared for a number of contemporary stream ciphers. We denote by $l_{\max}$ the maximum key length of the overall stream cipher. Note that σ represents the most efficient attack that (a) has been published at the moment and that (b) targets the keystream generator only. Also note that runtime estimates for cryptographic attacks are always somewhat tricky, but the order of magnitude should be correct. In appendix A, a short description is given on how values $\hat{n}$ and σ have been derived.

It can be observed that the stream ciphers with particularly large internal states have very low inner state efficiencies. It could be argued that the attacks that determined σ for these ciphers are distinguishing attacks, and that in all cases, it is not known how to transform them into prediction or key reconstruction algorithms.[6] But then again, distinguishing attacks could be mounted against the ciphers with smaller inner states, too, with less drastic results. As a consequence, the ciphers with large inner states do not seem to enjoy a real advantage over ciphers with small values for $\hat{n}$. In other words: It should be possible to construct a secure stream cipher from a generator with a reasonable inner state size.

6 Conclusions and research directions

Directions for future research: The following is an incomplete list of open questions that might be of interest for a more thorough understanding of stream cipher design.

- How much inner state is required to make a stream cipher secure?
- What is the right cryptographic primitive for a key schedule algorithm? Is a full-scale pseudorandom function generator required, or can we get away with a less strict security requirement?
- What constructions for provably secure stream ciphers can be given, in particular in the concrete security setting [5]?
- Can a set of practical design guidelines for key schedule algorithms be developed, in analogy to the design guidelines for block ciphers or keystream generators?
- What are the conditions for a direct attack on the key? Or, put another way: What can be said about the relationship between the keystream generator and the key schedule? Is it possible to develop a good stream cipher by using generic keystream generators and key setup algorithms independently? Or should both algorithms be chosen in an orthogonal way, being dependent on one another?
- What knowledge about key schedule algorithms from block ciphers can be re-used for stream cipher initialisation?

Summary: We have considered a wide-spread type of stream cipher and have given a definition for the inner state size of such ciphers. It was shown that an increase in inner state size can increase the security of the keystream generator used, but at the same time can slow down or even weaken the initialisation function. As a consequence, we propose to evaluate the strength of the keystream generator used relative to its inner state size. To this end, the notion of *inner state efficiency* was introduced. In an ad-hoc survey of practical stream ciphers, ciphers with large inner states displayed no cryptographical advantage over those with small inner states. This is an indication that efficient use of inner states is not just a theoretical claim, but is feasible in practice. To this end, we gave a number of questions that may be addressed by future research.

[6] In addition, these ciphers are not meant to be re-synchronised frequently. Thus, they can indeed afford larger internal states under running time considerations, since re-initialisation is required only rarely.

Acknowledgements

The author wishes to thank Frederik Armknecht, Stefan Lucks and Willi Meier for encouragement and inspiring discussions.

References

1. *Bluetooth Specification v1.1*, 1999. www.bluetooth.com.
2. F. Armknecht and M. Krause. Algebraic attacks on combiners with memory. In D. Boneh, editor, *Advances in Cryptology - Crypto 2003*, volume 2729 of *LNCS*, pages 162–175. Springer, 2003.
3. S. Babbage. A space/time tradeoff in exhaustive search attacks on stream ciphers. In *European Convention on Security and Detection*, volume 408 of *IEE Conference Publication*, May 1995.
4. S. Babbage, C. De Cannière, J. Lano, B. Preneel, and J. Vandewalle. Cryptanalysis of Sober-t32. In T. Johansson, editor, *Proc. FSE 2003*, volume 2887 of *LNCS*, pages 111–128. Springer, 2003.
5. M. Bellare. Practice-oriented provable security. In I. Damgård, editor, *Lectures on Data Security*, volume 1561 of *LNCS*, pages 1–15. Springer, 1999.
6. E. Biham. New types of cryptanalytic attacks using related keys. In T. Helleseth, editor, *Advances in Cryptology - Eurocrypt '93*, volume 765 of *LNCS*, pages 398–409. Springer, 1993.
7. A. Biryukov and A. Shamir. Cryptanalytic time/memory/date tradeoffs for stream ciphers. In T. Okamoto, editor, *Proc. Asiacrypt 2000*, volume 1976 of *LNCS*, pages 1–13. Springer, 2000.
8. M. Briceno, I. Goldberg, and D. Wagner. A pedagogical implementation of A5/1. `http://www.scard.org/gsm/a51.html`.
9. A. Clark, E. Dawson, J. Fuller, H.-J. Lee J. Dj. Golić, W. Millan, S.-J. Moon, and L. Simpson. The LILI-II keystream generator. In L. Batten and J. Seberry, editors, *Proc. ACISP 2002*, volume 2384 of *LNCS*, pages 25–39. Springer, 2002.
10. D. Coppersmith, S. Halevi, and C. Jutla. Cryptanalysis of stream ciphers with linear masking. In M. Yung, editor, *Advances in Cryptology - Crypto 2002*, volume 2442 of *LNCS*, pages 515–532. Springer, 2002.
11. N. Courtois. Fast algebraic attacks on stream ciphers with linear feedback. In D. Boneh, editor, *Advances in Cryptology - Crypto 2003*, volume 2729 of *LNCS*, pages 176–194. Springer, 2003.
12. P. Crowley and S. Lucks. Bias in the LEVIATHAN stream cipher. In M. Matsui, editor, *Proc. FSE 2001*, volume 2355 of *LNCS*, pages 211–218. Springer, 2002.
13. J. Daemen, R. Govaerts, and J. Vandewalle. Resynchronisation weakness in synchronous stream ciphers. In T. Helleseth, editor, *Advances in Cryptology - Eurocrypt '93*, volume 765 of *LNCS*, pages 159–167. Springer, 1994.
14. E. Dawson, A. Clark, J. Golić, W. Millan, L. Penna, and L. Simpson. The LILI-128 keystream generator. `http://www.isrc.qut.edu.au/resource/lili/lili_nessie_workshop.pdf`.
15. P. Ekdahl and T. Johansson. SNOW - a new stream cipher. `http://www.it.lth.se/cryptology/snow/`. NESSIE project submission.
16. P. Ekdahl and T. Johansson. A new version of the stream cipher SNOW. In H. Heys and K. Nyberg, editors, *Proc. SAC 2002*, volume 2595 of *LNCS*, pages 47–61. Springer, 2002.

17. P. Ekdahl and T. Johansson. Another attack on A5/1. *IEEE Trans. Information Theory*, 49(1):284–289, 2003.
18. H. Finney. An RC4 cycle that can't happen. Newsgroup post to sci.crypt, September 1994.
19. S. Fluhrer. Cryptanalysis of the SEAL 3.0 pseudorandom function family. In M. Matsui, editor, *Proc. FSE 2001*, volume 2355 of *LNCS*, pages 135–143. Springer, 2002.
20. S. Fluhrer and D. McGrew. Statistical analysis of the alleged RC4 keystream generator. In B. Schneier, editor, *Proc. FSE 2000*, volume 1978 of *LNCS*, pages 19–30. Springer, 2001.
21. J. Dj. Golić and G. Morgari. On the resynchronization attack. In T. Johansson, editor, *Proc. FSE 2003*, volume 2887 of *LNCS*, pages 100–110. Springer, 2003.
22. J.Dj. Golić. Cryptanalysis of alleged A5 stream cipher. In W. Fumy, editor, *Advances in Cryptology - Eurocrypt '97*, volume 1233 of *LNCS*, pages 239–255. Springer, 1997.
23. J.Dj. Golić. Linear statistical weakness of alleged RC4 keystream generator. In W. Fumy, editor, *Advances in Cryptology - Eurocrypt '97*, volume 1233 of *LNCS*, pages 226–238. Springer, 1997.
24. S. Halevi, D. Coppersmith, and C. Jutla. Scream: A software-efficient stream cipher. In J. Daemen and V. Rijmen, editors, *Proc. FSE 2002*, volume 2365 of *LNCS*, pages 195–209. Springer, 2002.
25. P. Hawkes and G. Rose. Primitive specification for Sober-128. `http://www.qualcomm.com.au/Sober128.html`.
26. P. Hawkes and G. Rose. Primitive specification and supporting documentation for Sober-t32. NESSIE project submission, October 2000.
27. S. Jiang and G. Gong. Cryptanalysis of stream cipher - a survey. Technical Report CORR-2002-29, University of Waterloo, 2002.
28. T. Johansson and A. Maximov. A linear distinguishing algorithm on Scream. Presented at ISIT 2003, available at `http://www.it.lth.se/movax/Publications/2002/Scream/disting.pdf`.
29. Itsik Mantin. Analysis of the stream cipher RC4. Master's thesis, Weizmann Institute of Science, Rehovot, Israel, November 2001.
30. D. McGrew and S. Fluhrer. The stream cipher Leviathan. NESSIE project submission, October 2000.
31. W. Meier. personal communication, August 2003.
32. P. Rogaway and D. Coppersmith. A software-optimized encryption algorithm. *Journal of Cryptology*, 11(4):273–287, Fall 1998.
33. R. Rueppel. Stream ciphers. In G. Simmons, editor, *Contemporary Cryptology - The Science of Information Integrity*, pages 65–134. IEEE Press, 1992.
34. M.-J. Saarinen. A time-memory tradeoff attack against LILI-128. In J. Daemen and V. Rijmen, editors, *Proc. FSE 2002*, volume 2365 of *LNCS*, pages 231–236. Springer, 2002.
35. A. Stubblefield, J. Ioannidis, and A. Rubin. Using the Fluhrer, Mantin and Shamir attack to break WEP. Technical Report TD-4ZCPZZ, AT&T labs, August 2001.
36. E. Zenner. On the efficiency of clock control guessing. In *Proc. ICISC '02*, LNCS. Springer, 2003.
37. E. Zenner, M. Krause, and S. Lucks. Improved cryptanalysis of the self-shrinking generator. In V. Varadharajan and Y. Mu, editors, *Proc. ACISP '01*, volume 2119 of *LNCS*, pages 21–35. Springer, 2001.

A Deriving $\hat{n}$ and σ

In the following, the inner states and best known attacks for the keystream generators in section 5.3 are discussed. In most cases, inner states can be subdivided into a linear part (i.e. inner states whose update function is linear), a nonlinear part, and key-dependent S-boxes which may or may not be bijective.

Note that for each stream cipher, only those attacks that target the keystream generator directly are considered.

- *A5/1:* The inner state is purely linear, consisting of 64 bit.
  Numerous attacks have been proposed against the full A5/1 stream cipher, all taking into account that in practice, only a small number of output bits is available to the attacker. If, however, we assume an arbitrary amount of output bits, the generic time-memory-tradeoff attack is most efficient, yielding $\sigma = 32$.
- *E_0:* The inner state consists of a 128-bit linear part and a 4-bit nonlinear part.
  A number of attacks against the E_0 generator have been proposed in literature. For the time being, the algebraic attack technique proposed by Courtois [11] using equations developed by Armknecht and Krause [2] seems to be the most efficient, requiring roughly 2^{49} computational steps. The attack does, however, require more consecutive output bits than the cipher produces between two re-initialisations.
- *Leviathan:* The inner state consists of a 48-bit counter and 4 permutation tables over $\{0,1\}^8$. Thus, the overall inner state size is $16 + 4 \cdot 1,684 = 6,784$ bit.
  The best known attack against Leviathan is a distinguisher by Crowley and Lucks [12], requiring 2^{39} bits of output and a similar work effort.
- *LILI-128:* The inner state consists of two independent linear states of sizes 39 and 89 bit, respectively, yielding a total inner state size of 128 bit.
  Given the construction of the cipher, a security level of 128 bit was not achievable in the first place due to lemma 1. As a consequence, a number of attacks on LILI-128 have been published, the most efficient one being a specialised time-memory attack by Saarinen [34] that requires roughly 2^{48} computational steps. Note that the attack proposed by Courtois in [11] formally requires less computational steps, but needs 2^{60} output bits.
  In the meantime, a successor LILI-II has been published [9]. The inner state size has been almost doubled to 255 bits, with the allowed key size still at 128 bit. No cryptanalysis of LILI-II has been published so far.
- *RC4 (8-bit version):* As already mentioned in section 3.1, the inner state size of RC4 is difficult to analyse. It consists of two 8-bit state variables and a table that implements a permutation $\{0,1\}^8$ that changes over time. Normally, this would yield an inner state size of $16+1,684 = 1,700$ bit. However, the starting values for the state variables are key-independent, and it was shown by Finney [18] that a fraction of 1/256 states can never be reached. Experiments on smaller versions of RC4 seem to indicate that the fraction

of non-reachable states is even larger but still small enough that $1,700$ is a good approximation of the inner state size.
Numerous attacks against RC4 have been described. A particularly strong attack against its keystream generator was proposed by Golić [23] and improved by Fluhrer and McGrew [20]. The attack is a distinguisher that requires $2^{30.6}$ output bits and a similar amount of work.

- *Scream:* The generator uses an evolving state of 128 bit, a round key of 256 bit, a mask table (16 times 16 bytes) of 2,048 bit, and an S-Box that implements a permutation over $\{0,1\}^8$ (1,684 bit). Thus, the inner state size is 4,116 bit.
The best attack against Scream that we are aware of is a linear distinguisher by Johansson and Maximov [28]. It requires about 2^{100} output bits and a similar work effort. Note, however, that Scream has been published only quite recently, i.e. it has not yet received the full cryptanalytic consideration.
- *Seal 3.0:* The generator uses a 12 bit counter, 8 32-bit state words, and a set of lookup tables consisting of 1024 32-bit words, contributing up to $32,768$ bit to the inner state. Thus, the inner state size of the generator is $33,036$ bit.
While the state words are re-initialised every $2^6 \cdot 2^7 = 2^{13}$ output bits, the tables are re-initialised once every 2^{19} output bits. Thus, SEAL has two initialisation functions h_1 and h_2, and can be seen considered as a stream cipher $(H, G) = ((h_1, h_2), g)$. Note that the best known attack - a distinguishing attack by Fluhrer [19] that requires rougly 2^{43} computational steps - is only applicable if (h_2, g) is considered as the keystream generator.[7]
- *Snow 1.0:* The linear part contributes 16 32-bit words to the inner state, while the nonlinear part adds another 2 32-bit words, yielding a total inner state size of 576 bit.
Amongst the attacks proposed against Snow 1.0, the most efficient is a distinguisher by Coppersmith et al. [10], requiring about 2^{100} computational steps.
An updated version Snow 2.0 with equal inner state size has been proposed [16], but no cryptanalytic results are available yet.
- *Sober-t32:* Here, the inner state is purely linear, consisting of 17 32-bit words. Thus, the inner state size is 544 bit.
The most efficient attack against full Sober-t32 is a distinguisher presented by Babbage et al. [4], requiring $2^{153+5} = 2^{158}$ output bits and a similar work effort.
Recently, a new version Sober-128 with equal inner state size but reduced key length was published [25]. However, no cryptanalytic results are available for the time being.

[7] The inner generator g is in fact a one-time pad: Without additional knowledge about the initialisation function, it is secure in an information-theoretical sense.