Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers

Alex Biryukov and Adi Shamir

Computer Science Department
The Weizmann Institute
Rehovot 76100, Israel.

Abstract. In 1980 Hellman introduced a general technique for breaking arbitrary block ciphers with N possible keys in time T and memory M related by the tradeoff curve $TM^2 = N^2$ for $1 \leq T \leq N$. Recently, Babbage and Golic pointed out that a different $TM = N$ tradeoff attack for $1 \leq T \leq D$ is applicable to stream ciphers, where D is the amount of output data available to the attacker. In this paper we show that a combination of the two approaches has an improved time/memory/data tradeoff for stream ciphers of the form $TM^2D^2 = N^2$ for any $D^2 \leq T \leq N$. In addition, we show that stream ciphers with low sampling resistance have tradeoff attacks with fewer table lookups and a wider choice of parameters.
Keywords: Cryptanalysis, stream ciphers, time/memory tradeoff attacks.

1 Introduction

There are two major types of symmetric cryptosystems: Block ciphers (which encrypt a plaintext block into a ciphertext block by mixing it in an invertible way with a fixed key), and stream ciphers (which use a finite state machine initialized with the key to produce a long pseudo random bit string, which is XOR'ed with the plaintext to obtain the ciphertext).

Block and stream ciphers have different design principles, different attacks, and different measures of security. The open cryptanalytic literature contains many papers on the resistance of block ciphers to differential and linear attacks, on their avalanche properties, on the properties of Feistel or S-P structures, on the design of S-boxes and key schedules, etc. The relatively few papers on stream ciphers tend to concentrate on particular ciphers and on particular attacks against them. Among the few unifying ideas in this area are the use of linear feedback shift registers as bit generators, and the study of the linear complexity and correlation immunity of the ciphers.

In this paper we concentrate on a general type of cryptanalytic attack known as a time/memory tradeoff attack. Such an attack has two phases: During the preprocessing phase (which can take a very long time) the attacker explores the general structure of the cryptosystem, and summarizes his findings in large tables (which are not tied to particular keys). During the realtime phase, the attacker

T. Okamoto (Ed.): ASIACRYPT 2000, LNCS 1976, pp. 1–13, 2000.

is given actual data produced from a particular unknown key, and his goal is to use the precomputed tables in order to find the key as quickly as possible.

In any time-memory tradeoff attack there are five key parameters:

- N represents the size of the search space.
- P represents the time required by the preprocessing phase of the attack.
- M represents the amount of random access memory (in the form of hard disks or DVD's) available to the attacker.
- T represents the time required by the realtime phase of the attack.
- D represents the amount of realtime data available to the attacker.

2 Tradeoff Attacks on Block and Stream Ciphers

In the case of block ciphers, the size N of the search space is the number of possible keys. We assume that the number of possible plaintexts and ciphertexts is also N, and that the given data is a single ciphertext block produced from a fixed chosen plaintext block. The best known time/memory tradeoff attack is due to Hellman [5]. It uses any combination of parameters which satisfy the following relationships: $TM^2 = N^2$, $P = N$, $D = 1$ (see Section 3 for further details). The optimal choice of T and M depends on the relative cost of these computational resources. By choosing $T = M$, Hellman gets the particular tradeoff point $T = N^{2/3}$ and $M = N^{2/3}$.

Hellman's attack is applicable to any block cipher whose key to ciphertext mapping (for a fixed plaintext) behaves as a random function f over a space of N points. If this function happens to be an invertible permutation, the tradeoff relation becomes $TM = N$, which is even better. An interesting property of Hellman's attack is that even if the attacker is given a large number D of chosen plaintext/ciphertext pairs, it is not clear how to use them in order to improve the attack.

Stream ciphers have a very different behavior with respect to time/memory tradeoff attacks. The size N of the search space is determined by the number of internal states of the bit generator, which can be different from the number of keys. The realtime data typically consists of the first D pseudorandom bits produced by the generator, which are computed by XOR'ing a known plaintext header and the corresponding ciphertext bits (there is no difference between a known and a chosen plaintext attack in this case). The goal of the attacker is to find at least one of the actual states of the generator during the generation of this output, after which he can run the generator forwards an unlimited number of steps, produce all the later pseudorandom bits, and derive the rest of the plaintext. Note that in this case there is no need to run the generator backwards or to find the original key, even though this is doable in many practical cases.

The simplest time/memory tradeoff attack on stream ciphers was independently described by Babbage [2] and Golic [4], and will be referred to as the **BG attack**. It associates with each one of the N possible states of the generator the string consisting of the first $log(N)$ bits produced by the generator from that state. This mapping $f(x) = y$ from states x to output prefixes y can be viewed as

a random function over a common space of N points, which is easy to evaluate but hard to invert. The goal of the attacker is to invert it on some substring of the given output, in order to recover the corresponding internal state. The preprocessing phase of the attack picks M random x_i states, computes their corresponding y_i output prefixes, and stores all the (x_i, y_i) pairs in a random access memory, sorted into increasing order of y_i. The realtime phase of the attack is given a prefix of $D + log(N) - 1$ generated bits, and derives from it all the D possible windows $y_1, y_2, ..., y_D$ of $log(N)$ consecutive bits (with overlaps). It lookups each y_j from the data in logarithmic time in the sorted table. If at least one y_j is found in the table, its corresponding x_j makes it possible to derive the rest of the plaintext by running the generator forwards from this known state[1]. The threshold of success for this attack can be derived from the birthday paradox, which states that two random subsets of a space with N points are likely to intersect when the product of their sizes exceeds N. If we ignore logarithmic factors, this condition becomes $DM = N$ where the preprocessing time is $P = M$ and the attack time is $T = D$. This represents one particular point on the time/memory tradeoff curve $TM = N$. By ignoring some of the available data during the actual attack, we can reduce T from D towards 1, and thus generalize the tradeoff to $TM = N$ and $P = M$ for any $1 \leq T \leq D$.

This $TM = N$ tradeoff is similar to Hellman's $TM = N$ tradeoff for random permutations and better than Hellman's $TM^2 = N^2$ tradeoff for random functions (when $T = M$ we get $T = M = N^{1/2}$ instead of $T = M = N^{2/3}$). However, this formal comparison is misleading since the two tradeoffs are completely different: they are applicable to different types of cryptosystems (stream vs. block ciphers), are valid in different parameter ranges ($1 \leq T \leq D$ vs. $1 \leq T \leq N$), and require different amounts of data (about D bits vs. a single chosen plaintext/ciphertext pair).

To understand the fundamental difference between tradeoff attacks on block ciphers and on stream ciphers, consider the problem of using a large value of D to speed up the attack. The mapping defined by a block cipher has two inputs (key and plaintext block) and one output (ciphertext block). Since each precomputed table in Hellman's attack on block ciphers is associated with a particular plaintext block, we cannot use a common table to simultaneously analyse different ciphertext blocks (which are necessarily derived from different plaintext blocks during the lifetime of a single key). The mapping defined by a stream cipher, on the other hand, has one input (state) and one output (an ouput prefix), and thus has a single "flavour": When we try to invert it on multiple output prefixes, we can use the same precomputed tables in all the attempts. As a result, tradeoff attacks on stream ciphers can be much more efficient than tradeoff attacks on block ciphers when D is large, but this possibility had not been explored so far in the research literature.

[1] Note that y_j may have multiple predecessors, and thus x_j may be different from the state we look for. However, it can be shown that these "false alarms" increase the complexity of the attack by only a small constant factor.

3 Combining the Two Tradeoff Attacks

In this section we show that it is possible to combine the two types of tradeoff attacks to obtain a new attack on stream ciphers whose parameters satisfy the relation $P = N/D$ and $TM^2D^2 = N^2$ for any $D^2 \leq T \leq N$. A typical point on this tradeoff relation is $P = N^{2/3}$ preprocessing time, $T = N^{2/3}$ attack time, $M = N^{1/3}$ disk space, and $D = N^{1/3}$ available data. For $N = 2^{100}$ the parameters $P = T = 2^{66}$ and $M = D = 2^{33}$ are all (barely) feasible, whereas the Hellman attack with $T = M = N^{2/3} = 2^{66}$ requires an unrealistic amount of disk space M, and the BG attack with $T = D = N^{2/3} = 2^{66}$ and $M = N^{1/3} = 2^{33}$ requires an unrealistic amount of data D.

3.1 Hellman's Time/Memory Tradeoff Attack on Block Ciphers

The starting point of the new attack on stream ciphers is Hellman's original tradeoff attack on block ciphers, which considers the random function f that maps the key x to the ciphertext block y for some fixed chosen plaintext. This f is easy to evaluate but hard to invert, since the problem of computing $x = f^{-1}(y)$ is exactly the cryptanalytic problem of deriving the key x from the given ciphertext block y.

To perform this difficult inversion of f with an algorithm which is faster than exhaustive search, Hellman uses a preprocessing stage which tries to cover the N points of the space with a rectangular $m \times t$ matrix whose rows are long paths obtained by iterating the function f t times on m randomly chosen starting points. The startpoints are described by the leftmost column of the matrix, and the corresponding endpoints are described by the rightmost column of the matrix (see Fig. 1). The output of the preprocessing stage is the collection of (startpoint, endpoint) pairs of all the chosen paths, sorted into increasing endpoint values. During the actual attack, we are given a value y and are asked to find its predecessor x under f. If this x is covered by one of the precomputed paths, the algorithm repeatedly applies f to y until it reaches the stored endpoint, jumps to its associated startpoint, and repeatedly applies f to the startpoint until it reaches y again. The previous point it visits is the desired x.

A single matrix cannot efficiently cover all the N points, (in particular, the only way we can cover the approximately N/e leaves of a random directed graph is to choose them as starting points). As we add more rows to the matrix, we reach a situation in which we start to re-cover points which are already covered, which makes the coverage increasingly wasteful. To find this critical value of m, assume that the first m paths are all disjoint, but the next path has a common point with one of the previous paths. The first m paths contain exactly mt distinct points (since they are assumed to have no repetitions), and the additional path is likely to contain exactly t distinct points (assuming that t is less than $\sqrt{N}$). By the birthday paradox, the two sets are likely to be disjoint as long as $t \cdot mt \leq N$, and thus we choose m and t which satisfy the relation $mt^2 = N$, which we call **the matrix stopping rule**.

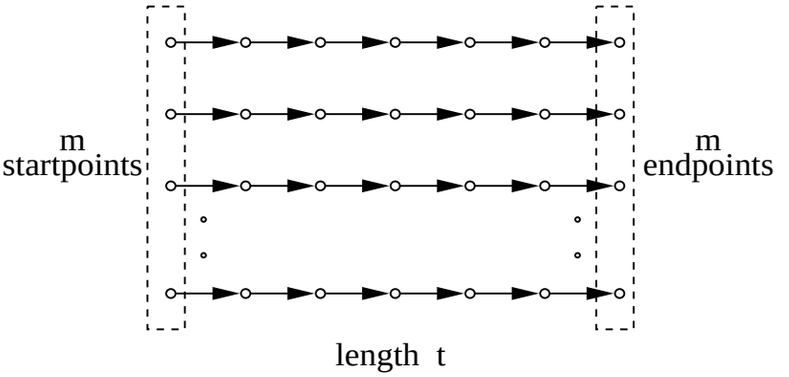

Fig. 1. Hellman's Matrix

A single $m \times t$ matrix with $mt^2 = N$ covers only a fraction of $mt/N = 1/t$ of the space, and thus we need t "unrelated" matrices to cover the whole space. Hellman's great insight was the observation that we can use variants f_i of the original f defined by $f_i(x) = h_i(f(x))$ where h_i is some simple output modification (e.g., reordering the bits of $f(x)$). These modified variants of f have the following properties:

1. The points in the matrices of f_i and f_j for $i \neq j$ are essentially independent, since the existence of a common point in two different matrices does not imply that subsequent points on the two paths must also be equal. Consequently, the union of t matrices (each covering mt points) is likely to contain a fixed fraction of the space.
2. The problem of computing x from the given $y = f(x)$ can be solved by inverting any one of the modified functions f_i over the modified point $y_i = f_i(x) = h_i(f(x)$.
3. The value of $y_i = f_i(x)$ can be computed even when we do not know x by applying h_i to the given $y = f(x)$.

The total precomputation requires $P \approx N$ time, since we have to cover a fixed fraction of the space in all the precomputed paths. Each matrix covers mt points, but can be stored in m memory locations since we only keep the startpoint and endpoint of each path. The total memory required to store the t matrices is thus $M = mt$. The given y is likely to be covered by only one of the precomputed matrices, but since we do not know where it is located we have to perform t inversion attempts, each requiring t evaluations of some f_i. The total time complexity of the actual attack is thus $T = t^2$. To find the tradeoff curve between T and M, we use the matrix stopping rule $mt^2 = N$ to conclude that $TM^2 = t^2 \cdot m^2t^2 = N^2$. Note that in this tradeoff formula the time T can be anywhere in the range $1 \leq T \leq N$, but the space M should be restricted

to $N^{1/2} \leq M \leq N$, since otherwise $T > N$ and thus the attack is slower than exhaustive search.

3.2 An Improved Attack on Stream Ciphers

As explained earlier in this paper, the main difference between tradeoff attacks on block ciphers and on stream ciphers is that in a block cipher each given ciphertext requires the inversion of a different function, whereas in a stream cipher all the given output prefixes can be inverted with respect to the same function by using the same precomputed tables.

To adapt Hellman's attack from block ciphers to stream ciphers, we use the same basic approach of covering the N points by matrices defined by multiple variants f_i of the function f which represents the state to prefix mapping. Note that partially overlapping prefixes do not necessarily represent neighboring points in the graph defined by the iterations of f, and thus they can be viewed as unrelated random points in the graph. The attack is successful if any one of the D given output values is found in any one of the matrices, since we can then find some actual state of the generator which can be run forward beyond the known prefix of output bits. We can thus reduce the total number of points covered by all the matrices from about N to N/D points, and still get (with high probability) a collision between the stored and actual states.

There are two possible ways to reduce the number of states covered by the matrices: By making each matrix smaller, or by choosing fewer matrices. Since each evaluation step of f_i adds m states to the coverage, it is wasteful to choose m or t which are smaller than the maximum values allowed by the matrix stopping rule $mt^2 = N$. Our new tradeoff thus keeps each matrix as large as possible, and reduces the number of matrices from t to t/D in order to decrease the total coverage of all the matrices by a factor of D. However, this is possible only when $t \geq D$, since if we try to reduce the number of tables to less than 1, we are forced to use suboptimal values of m and t, and thus enter a less efficient region of the tradeoff curve.

Each matrix in the new attack requires the same storage size m as before, but the total memory required to store all the matrices is reduced from $M = mt$ to $M = mt/D$. The total preprocessing time is similarly reduced from $P = N$ to $P = N/D$, since we have to evaluate only $1/D$ of the previous number of paths. The attack time T is the product of the number of matrices, the length of each path, and the number of available data points, since we have to iterate each one of the t/D functions f_i on each one of the D given output prefixes up to t times. This product is $T = t^2$, which is the same as in Hellman's original attack.

To find the time/memory/data tradeoff in this attack, we again use the matrix stopping rule $mt^2 = N$ in order to eliminate the parameters m and t from the various expressions. The preprocessing time is $P = N/D$, which is already free from these parameters. The time $T = t^2$, memory $M = mt/D$, and data D clearly satisfy the invariant relationship:

$$TM^2D^2 = t^2 \cdot (m^2t^2/D^2) \cdot D^2 = m^2t^4 = N^2$$

This relationship is valid for any $t \geq D$, and thus for any $D^2 \leq T \leq N$. In particular, we can use the parameters $P = T = N^{2/3}$, $M = D = N^{1/3}$, which seems to be practical for N up to about 100.

4 Time/Memory/Data Tradeoff Attacks with Sampling

One practical problem with tradeoff attacks is that random access to a hard disk requires about 8 milliseconds, whereas a computational step on a fast PC requires less than 2 nanoseconds. This speed ratio of four million makes it crucial to minimize the number of disk operations we perform, in addition to reducing the number of evaluations of f_i. An old idea due to Ron Rivest was to reduce the number of table lookups in Hellman's attack by defining a subset of **special points** whose names start with a fixed pattern such as k zero bits.

Special points are easy to generate and to recognize. During the preprocessing stage of Hellman's attack, we start each path from a randomly chosen point, and stop it only when we encounter another special point (or enter a loop, which is unlikely when $t \leq \sqrt{N}$). Consequently, we know that the disk contains only special endpoints. If we choose $k = log(t)$, the expected length of each path remains t (with some variability), and the set of mt endpoints we store in all the t tables contains a large fraction of the N/t possible special points.

The main advantage of this approach is that during the actual attack, we have to perform only one expensive disk operation per path (when we encounter the first special point on it). The number of evaluations of f_i remains $T = t^2$, but the number of disk operations is reduced from t^2 to t, which makes a huge practical difference.

Can we use a similar sampling of special points in tradeoff attacks on stream ciphers? Consider first the case of the BG tradeoff with $TM = N$, $P = M$, and $1 \leq T \leq D$. We say that an output prefix is special if it starts with a certain number of zero bits, and that a state of the stream cipher is special if it generates a special output prefix. We would like to store in the disk during preprocessing only special pairs of (state, output prefix). Unlike the case of Hellman's attack (where special states appeared on sufficiently long paths with reasonable probability, and acted as natural path terminators), in the BG attack we deal with degenerate paths of length 1 (from a state to its immediate output prefix), and thus we have to use trial and error in order to find special states.

Assume that the ratio between the number of special states and all the states is R, where $0 < R < 1$. Then to find the M special states we would like to store during preprocessing, we have to try a much larger number M/R of random states, which increases the preprocessing time from $P = M$ to $P = M/R$. The attack time reduces from $T = D$ to $T = DR$, since only the special points in the given data (which are very easy to spot) have to be looked up in the disk. To make it likely to have a collision between the M special states stored in the disk and the DR special states in the data, we have to apply the birthday paradox to the smaller set of NR special states to obtain $MDR = NR$. The invariant satisfied for all the possible values of R is thus

$$TP = MD = N \quad \texttt{for} \quad 1 \leq T \leq D$$

An interesting consequence of this tradeoff formula is that the sampling technique had turned the original BG time/memory tradeoff ($TM = N$) into two independent time/preprocessing ($TP = N$) and memory/data ($MD = N$) tradeoffs, which are controlled by the three parameters m, t, and R. For $N = 2^{100}$ the first condition is easy to satisfy, since both the preprocessing time P and the actual time T can be chosen as 2^{50}. However, the second condition is completely unrealistic, since neither the memory M nor the data D can exceed 2^{40}.

We now describe the effect of this sampling technique on the new tradeoff $TM^2D^2 = N^2$ described in the previous subsection. The main difference between Hellman's original attack on block ciphers and the modified attack on stream ciphers is that we use a smaller number t/D of tables, and force T to satisfy $T \geq D^2$. Unlike the case of the BG attack, the preprocessing complexity remains unchanged as N/D, since we do not need any trial and error to pick the random startpoints, and simply wait for the special endpoints to occur randomly during our path evaluation. The total memory required to store the special points remains unchanged at $M = mt/D$. The total time T consists of t^2 evaluations of the f_i functions but only t disk operations. We can thus conclude that the resultant time/memory/data tradeoff remains unchanged as $TM^2D^2 = N^2$ for $T \geq D^2$, but we gain by reducing the number of expensive disk operations by a factor of t. Rivest's sampling idea thus has no asymptotic effect on Hellman-like tradeoff curves for block and stream ciphers, but drastically changes the BG tradeoff curve for stream ciphers.

5 Tradeoff Attacks on Stream Ciphers with Low Sampling Resistance

The $TM^2D^2 = N^2$ tradeoff attack has feasible time, memory and data requirements even for $N = 2^{100}$. However, values of $D \geq 2^{25}$ make each inversion attack very time consuming, since small values of T are not allowed by the $T \geq D^2$ condition, while large values of T do not benefit in practice from the Rivest sampling idea (since the $T =$ evaluations of f_i functions dominate the $\sqrt{T}$ disk operations).

At FSE 2000, Biryukov, Shamir and Wagner [3] introduced a different notion of sampling, which will be called **BSW sampling**. It was used in [3] to attack the specific stream cipher A5/1, but that paper did not analyse its general impact on the various tradeoff formulas. In this paper we show that by using BSW sampling, we can make the new $TM^2D^2 = N^2$ tradeoff applicable with a larger choice of possible T values and a smaller number of disk operations.

The basic idea behind BSW sampling is that in many stream ciphers, the state undergoes only a limited number of simple transformations before emitting its next output bit, and thus it is possible to enumerate all the special states which generate k zero bits for a small value of k without expensive trial and error (especially when each output bit is determined by few state bits). This is

almost always possible for $k = 1$, but gets increasingly more difficult when we try to force a larger number of output bits to have specific values. The sampling resistance of a stream cipher is defined as $R = 2^{-k}$ where k is the maximum value for which this direct enumeration is possible. Stream ciphers were never designed to resist this new kind of sampling, and their sampling resistance can serve as a new quantifiable design-sensitive security measure. In the case of A5/1, Biryukov Shamir and Wagner show that it is easy to directly enumerate the 2^{48} out of the 2^{64} states whose outputs start with 16 zeroes, and thus the sampling resistance of A5/1 is at most 2^{-16}. Note that BSW sampling is not applicable at all to block ciphers, since their thorough mixing of keys and plaintexts makes it very difficult to enumerate without trial and error all the keys which lead to ciphertexts with a particular pattern of k bits during the encryption of some fixed plaintext.

An obvious advantage of BSW sampling over Rivest sampling is that in the BG attack we can reduce the attack time T by a factor of R without increasing the preprocessing time P. We now describe how to apply the BSW sampling idea to the improved tradeoff attack $TM^2D^2 = N^2$.

Consider a stream cipher with $N = 2^n$ states. Each state has a **full name** of n bits, and an **output name** which consists of the first n bits in its output sequence. If the cipher has sampling resistance $R = 2^{-k}$, we can associate with each special state a **short name** of $n - k$ bits (which is used by the efficient enumeration procedure to define this special state), and a **short output** of $n - k$ bits (which is the output name of the special state without the k leading zeroes). We can thus define a new random mapping over a reduced space of $NR = 2^{n-k}$ points, where each point can be viewed as either a short name or a short output. The mapping from short names to short outputs is easy to evaluate (by expanding the short names of special states to full names, running the generator, and discarding the k leading zeroes), and its inversion is equivalent to the original cryptanalytic problem restricted to special states.

We assume that $DR \geq 1$, and thus the available data contains at least one output which corresponds to some special state (if this is not the case we simply relax the definition of special states). We try to find the short name of any one of these DR special states by applying our $TM^2D^2 = N^2$ inversion attack to the reduced space with the modified parameters of DR and NR instead of D and N. The factor R^2 is canceled out from the expression $TM^2(DR)^2 = (NR)^2$, and thus the tradeoff relation remains unchanged. However, we gain in two other ways:

1. The original range of allowed values of T was lower bounded by D^2, which could be problematic for large values of D. This lower bound is now reduced to $(DR)^2$, which can be as small as 1. This makes it possible to use a wider range of T parameters, and speed up actual attacks.
2. The number of expensive disk operations is reduced from t to tR, since only the DR special points in the data have to be searched in the t/D matrices at a cost of one disk operation per matrix. This can greatly speed up attacks

with moderate values of t in which the t disk operations dominate the t^2 function evaluations.

Table 1 summarizes the behavior of the three types of tradeoff attacks under the two types of sampling techniques discussed in this paper. It explains why BSW sampling can greatly reduce the time T, even though it has no effect on the asymptotic tradeoff relation itself. Only this type of sampling enabled [3] to attack A5/1 and find its 64 bit key in a few minutes of computation on a single PC using only 4,000 disk operations, given the data contained in the first two seconds of an encrypted GSM conversation.

Sampling type	BG attack on stream ciphers	Hellman's attack on block ciphers	Our attack on stream ciphers
Rivest	new tradeoffs: $TP = MD = N$ for $1 \le T \le D$ increased P	unmodified tradeoff: $TM^2 = N^2$ for $1 \le T \le N$ fewer disk operations	unmodified tradeoff: $TM^2D^2 = N^2$ for $D^2 \le T \le N$ fewer disk operations
BSW	unmodified tradeoff: $TM = N, 1 \le T \le D$	inapplicable to block ciphers	unmodified tradeoff: $TM^2D^2 = N^2$, wider range, $(RD)^2 \le T \le N$ even fewer disk operations

Table 1. The effect of sampling on tradeoff attacks.

References

1. D. Coppersmith, H. Krawczyk, Y. Mansour, *The Shrinking Generator*, Proceedings of Crypto'93, pp.22–39, Springer-Verlag, 1993.
2. S. Babbage, *A Space/Time Tradeoff in Exhaustive Search Attacks on Stream Ciphers*, European Convention on Security and Detection, IEE Conference Publication No. 408, May 1995.
3. A. Biryukov, A. Shamir, and D. Wagner, *Real Time Cryptanalysis of A5/1 on a PC*, Proceedings of Fast Software Encryption 2000.
4. J. Golic, *Cryptanalysis of Alleged A5 Stream Cipher*, Proceedings of Eurocrypt'97, LNCS 1233, pp. 239–255, Springer-Verlag 1997.
5. M. E. Hellman, *A Cryptanalytic Time-Memory Trade-Off*, IEEE Transactions on Information Theory, Vol. IT-26, N 4, pp.401–406, July 1980.
6. W. Meier, O. Staffelbach, *The Self-Shrinking Generator*, Proceedings of Eurocrypt'94, pp.205–214, Springer-Verlag, 1994.

A The Sampling Resistance of Various Stream Cipher Constructions

As we have seen in the main part of the paper low sampling resistance of a stream cipher allows for more flexible tradeoff attacks. In this appendix we briefly review several popular constructions and discuss their sampling resistance.

A.1 Non-linear Filter Generators

In many proposed constructions a single linear feedback shift register (LFSR) is tapped in several locations, and a non-linear function f of these taps produces the output stream. Such stream ciphers are called *non-linear filter generators*, and the non-linear function is called a *filter*. The sampling resistance of such constructions depends on the location of the taps and on the properties of the function f. A crucial factor in determining the sampling resistance of such constructions is how many bits of the function's input must be fixed so that the function of the remaining bits is linear.

Multiplexor is a boolean function, which takes $s = \log t + t$ bits of the output, and treats the first $\log t$ bits as an address of the bit in the next t bits. This bit becomes the output of the function. In order to linearize the output of the multiplexor one needs to fix only $\log t$ bits. Multiplexor is thus a weak function in terms of linearization. The actual sampling resistance of the multiplexor is influenced by the minimal distance between the address taps and the minimal distance from the address taps to the output tap.

As a second example, consider the filter function

$$f(x_1, \ldots, x_s) = g(x_1, \ldots, x_{s-1}) \oplus x_s.$$

If there is a gap of length l between tap x_s and the other taps $x_1, \ldots, x_{s-1}$, then the sampling resistance is at most 2^{-l}, since by proper choice of the $s-1$ bits we can linearize the output of the function f. Suppose that our aim is to efficiently enumerate all the 2^{n-l} states that produce a prefix of l zeroes. We can do this by setting the $n-l$ non-gap bits to an arbitrary value, and then at each clock we choose the x_s bit in a way that zeroes the function f (assuming that feedback taps are not present in the gap of l bits).

Sum of Products A sum of products is the following boolean function: Pick a set of disjoint pairs of variables from the stream cipher's state: (x_{i_1}, x_{i_2}), $\ldots (x_{i_{s-1}}, x_{i_s})$. Define the filter function as:

$$f(x_1, \ldots, x_s) = \bigoplus_{j=1}^{s-1} x_{i_j} \cdot x_{i_{j+1}}.$$

A sum of products becomes a linear function if $s/2$ of its variables (one for each pair) are fixed. If these variables are all set equal to zero then f becomes the

constant function $f = 0$. We can thus expect this function to have a moderate resistance to sampling. The non-linear order of this function is only 2 and thus by controlling any pair $x_{i_j} x_{i_{j+1}}$ we can create any desired value of the filter function. For example if the target pair is (x_{i_1}, x_{i_2}) then the function f can be decomposed into:

$$f(x_1, \ldots, x_s) = x_{i_1} x_{i_2} \oplus g(x_{i_3}, \ldots, x_{i_s}).$$

At each step if the value of g is zero, the values of the target pair can be chosen arbitrarily out of $(0,0), (0,1), (1,0)$. If however $g = 1$, then the value of the target pair must be $(1,1)$. Thus if the control pair is in a tap-less region of size $2l$ with a gap l between the controlling taps, the sampling resistance of this cipher is at most 2^{-l}.

As another example, suppose that a consecutive pair of bits is used as a target pair. It seems problematic to use a consecutive pair for product linearization, since sometimes we have to set both bits to 1. This is however not the case if we relax our requirements, and use output prefixes with non-consecutive bits forced to have particular values. For example, prefixes in which every second bit is set to zero (and with arbitrary bits in between) can be easily generated in this sum of adjacent products.

Suppose now that in each pair the first element is from the first half of the register and the second element comes from the second half. Suppose also that the feedback function taps the most significant bit and some taps from the lower half of the register. In this case the sampling resistance is only $2^{-n/2}$. We set to arbitrary values the $n/2$ bits of the lower half of the register and guess the most significant tap bit. This way we know the input to the feedback function and linearize the output function. Forcing the output of the filter function at each step yields a linear equation (whose coefficients come from the lower half of the register and whose variables come from the upper half). After $n/2$ steps we have $n/2$ linear equations in $n/2$ variables which can be easily solved. This way we perform enumeration of all the states that produce the desired output.

Moreover, if all pairs in the product are consecutive, then even a more interesting property holds. We can linearize the function just by fixing a subset of $n/2$ even (or odd) bits of the register, and thus linearization is preserved even after shifting the register (with possible interference of the feedback function).

A.2 Shrinking and Self-Shrinking Generators

The shrinking generator is a simple construction suggested by [1] which is not based on the filter idea. This generator uses two regularly clocked LFSRs and the output of the first one decides whether the output of the second will appear in the output stream or will be discarded. This generator has good statistical properties like long periods and high linear complexity. A year later a self-shrinking generator (which used one LFSR clocked twice) was proposed by [6]. The output of the LFSR is determined by a pair of most significant bits a_{n-1}, a_n of the LFSR state: If $a_{n-1} = 1$ the output is a_n, and if $a_{n-1} = 0$ there is no output

in this clock cycle. This construction has the following sampling algorithm: pick arbitrary value for $n/2$ decision bits, and for each pair with a decision bit equal to 1 set the corresponding output bit to 0. If the decision bit is 0 then we have freedom of choice and we enumerate both possibilities. The sampling resistance of this construction is thus $2^{-n/4}$.