Fault Analysis of Stream Ciphers

Jonathan J. Hoch and Adi Shamir

Department of Computer Science and Applied Mathematics,
The Weizmann Institute of Science, Israel

Abstract. A fault attack is a powerful cryptanalytic tool which can be applied to many types of cryptosystems which are not vulnerable to direct attacks. The research literature contains many examples of fault attacks on public key cryptosystems and block ciphers, but surprisingly we could not find any systematic study of the applicability of fault attacks to stream ciphers. Our goal in this paper is to develop general techniques which can be used to attack the standard constructions of stream ciphers based on LFSR's, as well as more specialized techniques which can be used against specific stream ciphers such as RC4, LILI-128 and SOBER-t32. While most of the schemes can be successfully attacked, we point out several interesting open problems such as an attack on FSM filtered constructions and the analysis of high Hamming weight faults in LFSR's.

Keywords: Stream cipher, LFSR, fault attack, Lili-128, SOBER-t32, RC4.

1 Introduction

1.1 Background

Attacks against cryptosystems can be divided into two classes, direct attacks and indirect attacks. Direct attacks include attacks against the algorithmic nature of the cryptosystem regardless of its implementation. Indirect attacks make use of the physical implementation of the cryptosystem and include a large variety of techniques which either give the attacker some 'inside information' on the cryptosystem (such as power or timing analysis) or some kind of influence on the cryptosystem's internal state such as ionizing radiation flipping random bits in the device's internal memory. Fault analysis is based on a careful study of the effect of such faults (which can affect either the code or the data) on the ciphertext, in order to derive (partial) information about either the key or the internal state of the cryptosystem. Recently Anderson in [1] discovered an extremely low-tech, low-cost technique which allows an attacker with physical access to the cryptoprocessor (especially when implemented on a smartcard) to cause faults at very specific locations. This discovery transfers the ability to perform fault attacks to one's backyard making this kind of attack a major threat to smartcard issuers and users. Fault analysis was first used in 1996 by Boneh, Demillo, and Lipton in [2] to attack number theoretic public key cryptosystems such as RSA (by using a faulty CRT computation to factor the modulus n), and later by Biham and Shamir in [3] to attack product block ciphers such as DES (by using a high-probability differential fault attack on the last few rounds). While these techniques were generalized and applied to other public key and block ciphers in many subsequent papers, there are almost no published results on the applicability of fault attacks to stream ciphers, which requires different types of attacks and analytic tools. A notable exception is the re-synchronization attack of [9] which deals with a similar situation although in their model the changes (which correspond to our faults) are known. Our goal in this paper is to fill this void by embarking on a systematic study of all the standard techniques used to construct stream ciphers, and by analyzing their vulnerability to various types of fault attacks.

1.2 Our Results

We have succeeded in attacking a wide variety of stream ciphers. We first concentrated on attacking constructions based on LFSRs. With the exception of FSM filtered constructions we were able to attack almost any synthetic construction which appeared in the literature. The linearity of the LFSR is at the heart of all of these attacks. Although we have found a couple of attacks against very specific FSM filtered constructions, it would be interesting to find attacks against more general constructions. These results are covered in Section 2, where we present a comprehensive attack strategy against non-linearly filtered LFSRs as well as attacks against other LFSR based constructions. In section 3 we present fault attacks against LILI-128 and Sober (two LFSR based NESSIE candidates) and against RC4. The attack against RC4 applies random faults to the S-table after the initialization to recover the internal state by analyzing the first output byte of RC4 after initialization. All the attacks were analyzed theoretically and verified experimentally, in order to gain better understanding of their actual complexity and success rate. Due to space limitations we omit from this paper results which are harder to describe or require a longer introduction, such as a new DFA-like fault attack on the stream cipher Scream[5]. These results will be included in the full version of this paper.

1.3 The Attack Model

The basic attack model used in this paper assumes that the attacker can apply some bit flipping faults to either the RAM or the internal registers of the cryptographic device, but that he has only partial control over (and knowledge of) their number, location and timing. In addition, he can reset the cryptographic device to its original state and then apply another randomly chosen fault to the same device. In general, there is a tradeoff between the amount of control he has and the number of faults needed to recover the key. This model tries to reflect a situation in which the attacker is in possession of the physical device, and the faults are transient rather than permanent.

2 LFSR Based Stream Ciphers

2.1 Introduction

A very common component in stream ciphers is the Linear Feedback Shift Register (LFSR). LFSR's have provably long cycles and good statistical properties, but due to their inherent linearity LFSRs do not generate good output streams by themselves. Hence, LFSRs are typically used in conjunction with some non-linear component. There are three general constructions for implementing a stream cipher based on LFSRs:

– Filter the output of the LFSR(s) through a non-linear function.
– Have the clocking of one LFSR controlled by the output sequence of another LFSR.
– Filter the output of the LFSR(s) through a finite state machine.

In this section we develop several types of fault attacks against such generic constructions. We denote the length of the LFSR by n, the XOR of the original and faulted value of the LFSR at the time the fault was introduced by Δ, and the number of flipped bits (i.e., the Hamming weight of Δ) by k.

2.2 Attacks on Non-linearly Filtered LFSR Based Stream Ciphers

Let $(x_1, x_2, ..., x_n)$ be the internal state of the LFSR where $x_i \in \{0, 1\}$. A non-linear filter applied to a LFSR is a boolean function $f(x_{i_1}, x_{i_2}, .., x_{i_t})$ whose inputs are a subset of the LFSR's internal state bits (typically, $n \leq 128$ and $t \leq 12$). More generally the inputs to the function may come from several LFSRs. Each output bit is produced either by evaluating f on the current state, or by using a lookup table of pre-computed values of f. The LFSR is then clocked and f is evaluated again on the resulting state to generate the next output bit. Existing attacks against this construction include the algebraic attack [12] which is generally infeasible when t is not extremely small and the re-synchronization attack [9] which shares a similar setting with our attack.

Now assume that the attacker has the power to cause low Hamming weight faults in the LFSR's internal state bits. The attack will proceed as follows:

1. Cause a fault and produce the resulting output stream
2. Guess the fault
3. Check the guess, if incorrect guess again
4. Repeat 1-3 $O(t)$ times
5. Solve a system of linear equations in the original state bits

We need to show how to check the correctness of a guess and then how to construct the system of linear equations. Notice that due to the linearity of the LFSR clocking operation L, if we know the initial difference Δ due to the fault then at any time i the difference will be $L^i(\Delta)$. To verify a guess for Δ we predict the future differences in the t input bits to f. Whenever this difference is 0 we expect to see an output difference of 0. If our guess was incorrect, then for half of these occasions we will see a non-zero output difference. So on average after 2^{t+1} output bits we expect to reject incorrect guesses.

This identification procedure leaves room for optimization by a preprocessing stage in which the for each possible non-zero difference location, all inconsistent faults are pre-computed. This Enables us to simultaneously reject all faults inconsistent with the observed differences. This can greatly reduce the time complexity of the attack; the details of this approach are not presented here due to space limitations.

Now let us concentrate on a single output bit. For each faulted stream the attacker observes the difference in the output bit and can compute the input difference to f. We collect pairs of input/output differences corresponding to the same output bit location. Given about t pairs we can narrow down by exhaustive search the possible input bits to one possibility. By determining these bits we get linear equations in terms of the initial state bits. Using the same faulted output streams we can also compute the input differences for other output bits collecting more linear equations. Once we collect enough $(\theta(n))$ equations we can solve the set of equations and determine the initial LFSR state.

We can sometimes improve the amount of data needed for the attack by analyzing the structure of f. Define $A = \{\Delta \mid Pr[f(x) \oplus f(x \oplus \Delta) = 0] > \frac{1}{2} + \epsilon\}$. After guessing Δ, the initial difference, we compute as before the differences $\Delta_n = L^n(\Delta)$ at any future time. When $\Delta_n \in A$ we know that with probability at least $\frac{1}{2} + \epsilon$ the difference in the output of f will be 0. I.e, the average of the difference over the output bits for which $\Delta_n \in A$ should be $\frac{1}{2} + \epsilon$. If our guess of Δ was incorrect then we expect to see an average of $\frac{1}{2}$. Thus after seeing about $O(\epsilon^2 \frac{|A|}{2^n})$ we should be able to tell with high probability whether our guess of Δ was correct. Analysis of f will show us the optimal ϵ and whether we achieve an advantage over the previous strategy.

If the Hamming weight of the faults is very low then we can apply another strategy to reduce the amount of data required by guessing and verifying m faults simultaneously. This will increase the time complexity by a factor of $\binom{n}{k}^{m-1}$, but we can now check our guess by comparing the relative difference in the input of f for each pair of the $m+1$ streams. This gives us a probability of approximately $2^{-t\binom{m+1}{2}}$ of having a zero relative difference, thus reducing the amount of data required by a factor of $\binom{m+1}{2}$.

So far we assumed that the function f is known, but we can apply a fault attack even if f is unknown. First notice that in order to verify a guessed fault in the simple variation we did not need to know f. So we can carry out steps 1-4 even when the non-linear function f is unknown or key-dependent.

Definition 1. *Let $D(i)$ be the set of input-output difference pairs resulting from the faults at position i in the output stream.*

Definition 2. *A 0-order linear structure of f is n-bit vector γ s.t. for all X $f(X) = f(X \oplus \gamma)$*

Proposition 1. *The 0-order linear structures of f form a vector space.*

Now if for two positions i and j $D(i) = D(j)$ and $|D(i)| = 2^t$ then either the un-faulted inputs X, Y to f at positions i and j were the same or $X \oplus Y$ is a 0-order linear structure of f. Analysis of $D(i)$ can give us the linear structures of f in time $O(t2^t)$ using the Walsh-Hadamard transform of f [9], [11]. In either case, we get linear equations in the original state variables. After recovering the LFSR state we can easily recover f. Even if $D(i) \bigcap D(j) < 2^t$ we can still conclude with high probability that $X = Y$ (or $X \oplus Y$ is a 0-order linear structure) if the intersection is large enough. Experimental results show that for a random 10-bit boolean function f, about 300 faults were sufficient to successfully carry out the attack.

Another improvement can be made to the total amount of data required by comparing the new faults against the already identified ones. For example, after the first fault has been identified we compare the next fault against the original data and the first faulted stream. When 2^t faults are required this will reduce the total amount of data to $O(t2^t)$ instead of $O(2^{2t})$.

The only property of the LFSR which we used for these attacks is that we can compute future differences based on the initial fault. Thus the attacks generalize directly to a construction composed of several LFSRs connected to the same non-linear filter, providing that the total Hamming weight of the faults in all the registers is low. However, we were unable to find any fault attacks utilizing faults with high (and thus un-guessable) Hamming weight.

2.3 Attacks on Clock Controlled LFSR Based Stream Ciphers

The basic clock controlled LFSR construction is composed of two components: the clock LFSR and the data LFSR. The output stream is a subsequence of the output of the data LFSR which is determined by the clock LFSR. For example, when the clock LFSR output bit is 0 clock the data LFSR once and output its bit, and when the clock LFSR bit is 1 clock the data LFSR twice and output its bit. Unless specified otherwise, all attacks in this section will refer to this construction.

Other variations include considering more than one bit of the clock LFSR to control the clocking of the data LFSR (E.g., in LILI-128 two bits of the clock LFSR are used to decide whether to clock the data LFSR one to four times). The last variation considered here is the shrinking generator [6] in which the output bits of the clock LFSR decide whether or not the current data LFSR output bit will be sent to the output stream, and thus there is no fixed upper bound on the time difference between consecutive output bits. Existing attacks against clock controlled constructions include correlation attacks [10], algebraic attacks [12] and re-synchronization attacks [10].

Throughout this section we will use the term *data stream* to indicate the sequence produced by the data LFSR $\{d_i\}_{i=1}^{\infty}$ as opposed to the *output stream* denoted $S = \{S_i\}_{i=1}^{\infty}$ which is the sequence of output bits produced by the device. The control sequence produced by the clock LFSR will be denoted $\{c_i\}_{i=1}^{\infty}$, and we define $pos_S(i)$ to be the position of the i^{th} bit of the output stream S in the data stream.

A phase shift in the data register. A phase shift is a fault in which one of the components is clocked while the other is not. Once the phase shift takes place the device continues operating as usual. In a clock controlled construction a phase shift in the data LFSR can give us information about the clock register. Denote by S the non-faulted output stream and by $\hat{S}$ the faulted output stream. Notice that for every bit i after the fault $pos_{\hat{S}}(i) = pos_S(i) + 1$ since the data register was clocked one extra time. So the attacker looks for i s.t. $\hat{S}_i \neq S_{i+1}$, this implies that at the i^{th} location the data register was clocked twice. Thus we can recover a bit of the clock LFSR state (which corresponds to a linear equation in the original state) each time we have such an occurrence.

110101001001010 - clock register
0010101101010101001010101 - data register
110100100101001 - output stream

0101011010101010010101010 - data register after phase shift
000101101100011 - output stream

110100100101001 - original output stream
001011011000110 - faulted output stream
Each bit in the original sequence is compared with the bit to its left in the faulted sequence. When a difference is observed the clock register must have been 1.

```
*1***1**1**1*1* - Partial data recovered by comparing the two sequences.
110101001001010 - The actual clock register.
```

Fig. 1. An example of a Phase Shift Attack

We need about twice the length of the clock register to recover the whole state since the probability of such an occurrence is $\frac{1}{2}$. After recovering the clock LFSR's state we can easily recover the data LFSR's since we now know the position of each output bit in the data stream.

It is left as an easy exercise to show that this attack can be adapted to deal with phase shift faults in the shrinking generator and the stop & go generator.

Faults in the clock register. For simplicity of description we assume that the attacker can apply random single bit faults to the clocking LFSR at a chosen point in the execution. The full attack, which is too technical to describe here, can be carried out even if the timing of the fault is not exactly known and it affects a small number of bits. The first stage of the attack will be to produce the n possible separate faulted output streams by applying a single bit fault at the same timing (at different locations) to the clocking register. We will designate the stream resulting from a fault in the i^{th} location by S^i, S^i_j being the j^{th} bit of S^i (counting from the timing of the fault). Let us observe S^i_j for a fixed j s.t. $j < n$. This condition assures that the feedback of the clock register has not

affected the output stream yet as a result of the fault. I.e., the only changes are a result of the single bit change at the i^{th} location. If $i \geq j$ then the fault will not have enough time to affect S_j^i and $S_j^i = S_j$. However, if $i < j$ then similar to the phase shift example, $|pos_{S^i}(j) - pos_S(j)| = 1$. If $c_i = 1$ then we will get $pos_{S^i}(j) - pos_S(j) = -1$ (we have clocked the data LFSR one time less) and $pos_{S^i}(j) - pos_S(j) = 1$ if $c_i = 0$. Now assume that for all i S_j^i is the same. This implies that both neighbors of the original bit in the data stream are identical to the bit itself.
...$0\hat{0}0$... - the original data stream where the $\hat{*}$ was chosen for the output
...$\hat{0}00$... - the original data with faulted clocking
...$00\hat{0}$... - the original data with faulted clocking
The only other case in which this could happen is if the first j bits of the clock register were identical, since then we only see one of the neighbors. By choosing j large enough we can neglect this possibility. If we see $j-1$ streams which are identical in the j^{th} bit but different from the original j^{th} bit then the data stream must have looked as follows:
...$1\hat{0}1$... - the original data stream where the $\hat{*}$ was chosen for the output
In this case we know that both neighbors of the bit in the data stream were equal. If the next output bit in the actual stream was different from the neighbors, then the data register must have been clocked twice.
...$0\hat{0}0\hat{1}$... - the $\hat{*}$ bits were chosen for the output
...$1\hat{0}1\hat{0}$... - the $\hat{*}$ bits were chosen for the output
In this case we have recovered a bit of the clock LFSR or more generally a linear equation in the original LFSR state. By analyzing similar structures we show that there is a probability of at least $\frac{6}{32}$ of this situation occurring. Hence we can get about $\frac{3n}{16}$ linear equations. We now repeat the attack and collect another batch of faulted streams with the timing of the faults changed. After repeating this procedure ~ 10 times we will have collected an over-determined set of equations which we can solve for the clocking LFSR's original state. After recovering the clock LFSR we can easily solve for the data LFSR. The attack requires about $10n$ faults and for each fault a little more than n bits (for unique identification of the streams). This attack is also applicable to the decimating and stop & go generators since the effect of a single bit fault in the control LFSR is also locally identical to a phase shift in the data LFSR.

Faults in the data register. The next attack will focus on the data LFSR, but before we give a description of the attack we will show a general algorithm for recovering the clock register given the data register.

For a clock controlled construction $pos(i) = \Sigma_{j=1}^{i} c_j$ is the position of the i^{th} bit of the output stream in the data stream. The input to the algorithm will be the sequence $\{d_i\}$ and we will identify $pos(i)$ for various i. Notice that each value of $pos(i)$ gives us a linear equation in the original state of the LFSR, since each of the c_i's can be represented as a linear combination of the original state bits and $pos(i)$ is a linear combination of the c_i's. Once we have collected enough values we can solve the set of equations for the initial state of the clock LFSR. The algorithm works by keeping a list of all possible values of $pos(i)$ for each output bit of the device. This is done by simple elimination: check for each existing position in the list whether it is possible to receive the actual output with one of the possible values of c_i. Now if we find an i such that the list of candidates for $pos(i)$ is a single value we know the corresponding $pos(i)$. Experimental results show that given a random initial state for LFSRs of size 128 bits, the algorithm finds the original state after seeing a few hundred bits, finding a linear equation every 5 or 6 bits. If the output sequence was not produced from $\{d_i\}$ then the algorithm finds an inconsistency in the output stream (the size of the list shrinks to zero) after at most a few tens of bits. This behavior can also be studied analytically. Let x_i and y_i be the minimal and maximal candidate values for $pos(i)$ respectively. Assuming y_i is not the real value for $pos(i)$ let us calculate the expectation of $y_{i+1} - y_i$. This expectation is bounded from above by $\frac{5}{4}$, since there is a probability of $\frac{1}{2}$ that the maximum grows by 2 and a probability of $\frac{1}{4}$ that the maximum grows by 1. On the other hand the expectation of $x_{i+1} - x_i$ is bounded from below by $\frac{1}{2} + \frac{2}{4} + \frac{3}{8} = \frac{11}{8}$ so the expectation of the change to the size of the list of possibilities for $pos(i)$ is negative. I.e., the size of the list is expected to shrink unless one of the endpoints is the true position. This implies that the average size of the list is constant and thus the running time is linear. Now our attack will proceed as follows:

1. Generate a non-faulted output stream of length $10n$
2. Re-initialize the device, and cause a low Hamming weight fault in the data register
3. Generate a new (faulted) stream of length $10n$
4. Guess the fault and verify by running the above algorithm with the calculated difference in the data stream and the output stream difference
5. Repeat until the guess is consistent with the output stream
6. Recover the data register state from the actual output and the known clocking register

Since the clocking register was not affected, the difference in the output stream is equivalent to a device with the same clocking and with the data register initialized to the fault difference. Since given a guess of the initial state of the data register, the attacker can calculate the difference at any future point, we can apply the algorithm for recovery of the clock register. For incorrect guesses of the fault, the algorithm will find the inconsistency and for the correct guess the algorithm will find the initial state of the clock register.

2.4 Attacks on Finite State Machine Filtered LFSR Based Stream Ciphers

In this section we will show some attacks on a basic FSM filtered LFSR construction. The FSM contains some memory whose initial content is determined by the key. Each time the LFSR is clocked, the LFSR output bit is inserted into a specific address determined by a subset of the LFSR's state, and the bit previously occupying that memory location is sent to the output. The number of memory bits will be denoted by M and thus there are $\log M$ address bits. The leading attacks against general FSM filtered LFSR constructions are algebraic attacks [12], but these attacks are only feasible against very specific constructions.

Randomizing the LFSR. Assume that the attacker has perfect control over the timing of the fault, and that he can cause a fault which uniformly randomizes the LFSR bits used to address the FSM. The first output bit after the fault has been applied will be uniformly distributed over the bits currently stored in the FSM. By repeating the fault at the same point in time we can recover the number of ones currently stored in the FSM. If we do the same at a different point in time we can, by examining the actual output stream, recover the total number of ones entering the FSM. This gives us a linear equation in the initial LFSR state. By collecting enough equations we can solve for the initial state.

Faults in the FSM. If a random fault is applied to the current contents of the FSM the output stream will have differences at the timings when the LFSR points to the faulted bits' addresses. We start by giving some intuition about the attack. Assume that the LFSR points to the same address at two consecutive clockings. If the fault in the FSM happened at this location before these points in time, only the first occurrence of this location in the output stream will be faulted. When examining the second occurrence no matter what fault occurred in the FSM the bit will not be faulted as long as the timing of the fault was before the first occurrence. When we notice a case like this we know that the address is the same in the two consecutive timings, this gives us linear relations on the bits of the LFSR. By collecting enough relations we can derive the LFSR state. More generally, let p be the probability of a single bit in the FSM being affected by the fault and let us assume that the timing of the fault is uniformly distributed over an interval $[t_1, t_2]$ of length T. The probability of a difference in bit t between the faulted and non-faulted streams is $\frac{t-t_1}{t_2-t_1}p$ provided that this is the first occurrence of the address. If the most recent occurrence of the same address before time t is at time t_0 then the probability is $\frac{t-t_0}{t_2-t_1}$. So by estimating this probability within $\frac{1}{2(t_2-t_1)}$ we can tell when the address bits were the same at two different timings t_0 and t. This gives us $\log M$ linear equations in the original LFSR bits. We repeat this $\frac{n}{\log M}$ times and recover the initial state of the LFSR from the resulting set of linear equations

3 Fault Attacks on Actual Stream Ciphers

3.1 A Fault Attack on LILI-128

In this section we will bring some of the techniques presented into action in a fault attack against LILI-128 [4], one of the NESSIE candidates. For existing attacks on this stream cipher see [12] and [13].

LILI-128 is composed of two LFSRs: $LFSR_c$, which is 39 bits long, and $LFSR_d$, which is 89 bits long (with a total of 128 bits of internal state). Both have primitive feedback polynomials. For each keystream bit:

- The keystream bit is produced by applying a nonlinear function f_d to 10 of the bits in $LFSR_d$.
- $LFSR_c$ is clocked once. Two bits from $LFSR_c$ determine an integer c in the range $\{1, 2, 3, 4\}$.
- $LFSR_d$ is clocked c times.

The keystream generator is initialized simply by loading the 128 bits of key into the registers. Keys that cause either register to be initialized with all zeroes are considered invalid. The exact function f_d used, which bits are taken as inputs and the feedback polynomials of the LFSRs are irrelevant to the attack.

The first stage of the attack is to apply a random one bit fault to the data register. Repeat this until 89 (the length of $LFSR_d$) distinct streams are observed. Now repeat the same with the construction clocked once before applying the faults. Notice that some of the streams produced will be the same as in the first batch. This is due to the fact that applying the fault and then shifting the LFSR is equivalent to shifting the LFSR and then performing the fault, provided the fault did not affect the feedback. By counting how many streams are repeated one can deduce how many times $LFSR_d$ was clocked, which provides two bits of $LFSR_c$. Thus after repeating the experiment about 20 times we can recover the full $LFSR_c$ state. Once this state is known we can use the algorithm presented in section 1.2 to recover the state of $LFSR_d$. Notice that no further faults are necessary and the data collected in the previous stage can be reused. A tradeoff between the number of faults used and the length of the attack can be achieved by stopping after part of the state has been recovered and guessing the rest.

3.2 A Fault Attack on SOBER-t32

SOBER-t32 [7] is another NESSIE candidate with a LFSR based design. SOBER is composed of a LFSR, a non linear filter (NLF) and a form of irregular decimation called *stuttering*. The LFSR works over the field $GF(2^{32})$, and produces a stream of 32-bit words $L_1, L_2, ...$ called the L-stream. The internal state of the LFSR will be denoted $\sigma_i = (s_i, s_{i+1}, ..., s_{i+16})$, and σ_0 will denote the initial state. The L-stream is fed through the NLF to produce 32-bit words $N_1, N_2, ...$ called the N-stream, $N_i = NLF(\sigma_i)$. The stuttering decimates the N-stream as follows: the first N-word N_1 is the first stutter control word SCW. The SCW is partitioned into 16 pairs of bits, each pair of bits is read in order and accordingly one of four actions is performed:

1. Clock the LFSR once but do not output anything.
2. Clock the LFSR once, output the current N-word xored with $0x6996C53A$ and clock the LFSR again (without producing more output).
3. Clock the LFSR twice and then output the current N-word.

4. Clock the LFSR once and then output the current N-word xored with $0x96693AC5$.

When all the bits of the SCW have been read, the LFSR is clocked and the output of the NLF becomes the next SCW. The NLF is defined as $NLF(\sigma_i) = ((f(s_i + s_{i+16}) + s_{i+1} + s_{i+6} \oplus Konst) + s_{i+13}$ where f is some non linear function whose exact definition is not relevant to the attack. The key determines σ_0 and $Konst$. Existing attacks against SOBER-t32 can be found in [14]and [15].

The attack will proceed in two stages. The aim of the first stage is to strip away the stuttering and recover the full N-stream, i.e., the output after the NLF. The aim if the second stage is to recover the original state of the LFSR based on the faults seen in the N-stream.

Stripping the Stuttering. To achieve this goal we assume that we can apply random single bit faults to the output of the N-stream. If we damage a word which is not a stutter control word, then depending on whether the word appeared in the original stuttered output we will see either a single bit difference in the faulted output stream or no change at all. If we fault a stutter control word, then we will see a significant difference in the output stream. However, we know that both streams originated from the same N-stream hence we can use them to reconstruct the original N-stream. To check whether two output words originated from the same N-word we simply check if their xor is in the set $\{0, 0x6996C53A, 0x96693AC5\}$, and the probability of a wrong identification is negligible since we are matching 32-bit words. We know that in each stream the order of the words is the same so with enough faults we can fully reconstruct the N-stream. Since the probability of a N-word being sent to the output is slightly below $\frac{2}{3}$ (remember that $\frac{1}{17}$ of the N-words are used as SCWs) it is enough to cause ~ 10 faults in the SCW to ensure that we reconstruct a significant part of the N-stream. Since the probability of causing a fault in a SCW is $\frac{1}{17}$, we can carry out this stage of the attack with less than 200 faults.

Recovering the LFSR State. Now we will use faults to the LFSR to retrieve its original state. Assume for now that the fault occurred in σ_{13} where σ is the current state of the LFSR. Let us denote the timing of the fault by i, i.e., we faulted σ_{i+13}. Notice that we have not assumed control over the timing of the fault, only over the location of the fault within the LFSR. We observe the first nonzero difference in the output stream which results from our fault. If N_t was sent to the output then the observed difference with respect to subtraction mod 2^{32} will be $\sigma_{i+13} - \hat{\sigma}_{i+13} = \pm 2^j$ where $\hat{\sigma}_{i+13}$ represents the faulted version and j is the bit faulted. If N_i was not sent to the output then the first observed difference is very unlikely to be of the above form. The sign of the difference will give us the original bit in the j^{th} position (we are exploiting here the nonlinearity of $+$ with respect to $\oplus$). Notice that until now we have not used the fact that we know the N-stream. Since we know the position of the current output word in the N-stream we know the exact place of the bit recovered in the L-stream and hence have a linear equation in the original bits of the equivalent $GF(2)$ LFSR. By repeatedly applying faults we can recover enough linear equations and reconstruct the initial state. Notice that what actually remains to be shown is how to identify faults whose first effect on the N-stream is when the fault is in σ_{13}. But as we have shown before, such faults have a unique signature (an output difference of $\pm 2^j$) which allows us to identify them. Some care must be taken as to not confuse them with faults in σ_1 or σ_6, this can be done by rejecting candidates for which the output difference (in the N-stream) is a single bit. After reconstructing the LFSR state we can find $Konst$ from the equation for the NLF, the observed N-stream and the calculated L-stream.

In the full description of SOBER-t32, there is also a key-loading procedure which mixes the secret key and session key to initialize $Konst$ and σ_0. A similar fault attack can be applied to recover the secret key from the session key and the initial state.

3.3 An Attack on RC4

RC4 is a stream cipher designed by Ron Rivest in 1987. Its source code was kept as a trade secret until an alleged version was posted to the Cyberpunks mailing list [8]. RC4 consists of a key scheduling which initializes the permutation S, initialization and a generation loop. The key schedule will not be of interest for our attack. The most successful attacks against RC4 are guess and determine [8] but even these are prohibitively time consuming (more than 2^{700} time).

```
Initialization:
    $i = 0$
    $j = 0$
Generation Loop:
    $i = i + 1$
    $j = j + S[i]$
    Swap S[i] and S[j]
    Output S[S[i]+S[j]]
```

Fig. 2. Pseudo-code for RC4

Our attack will proceed in three stages:

1. Apply a fault to the S table and generate a long stream (repeat many times)
2. Analyze the resulting streams and generate equations in the original entries of S
3. Solve these equations to reconstruct S.

We assume that the attacker can fault a single entry of the S table immediately after the key-scheduling. Our first observation is that the attacker can recognize which value was faulted. I.e., if $S[x] = a$ and the fault changed its value to b then we will identify both a and b (but not x). This can be done by observing the frequency of each symbol in the output stream. If a was changed to b then a will never appear in the output stream, while b will appear with double frequency. Thus we need a stream of length about 10,000 bytes to reliably identify a and b. Our next mission is to identify faults in $S[1]$. This is done by looking at the first output byte. If this byte changed as a result of the fault then one of three cases must hold:

1. $S[1]$ was faulted
2. $S[S[1]]$ was faulted
3. $S[S[1] + S[S[1]]]$ was faulted

We know what the original value of $S[S[1] + S[S[1]]]$ was so we can check if the fault affected this cell (by identifying a and b). If we fault $S[1]$ and can identify the fault, i.e. $S[1]$ changed from a to b, then we know two things. First the original value of $S[1]$ was a and second, $S[b + S[b]] = c$ where c is the actual observed output in the faulted stream. So our first issue is how to recognize faults. If case 2 holds then with high probability the second output byte $S[S[2]+S[S[1]+S[2]]]$ will not be faulted. If the first case holds then the second output byte will always be faulted.

Now that we have identified a fault that affected $S[1]$ and changed its value from a to b we know two things: $S[1] = a$ and $S[b + S[b]] = c$ where c is the first output byte of the faulted stream. For each fault in $S[1]$ we get an equation, and after collecting many such equations we start utilizing our knowledge of $S[1]$ to deduce other values is S. For example, if $S[1] = 17$ then the equation $S[1 + S[1]] = 7$ will give us the value of $S[18] = 7$. We deduce as many values as possible from the given equations. If at the end we have not recovered $S[S[1]]$ then we guess its value. From our knowledge (guess) of $S[S[1]]$ we can carry out an analysis of the second output byte and recover more equations, this time of the form $S[b + S[b + S[1]]] = d$ (where d is the second output byte). Empirical results show that at this stage we recover on average 240 entries of S, and this is more than enough to deduce the rest from the observed non-faulted stream. We can easily reject incorrect guesses of $S[S[1]]$ by either noticing an inconsistency in the equations we collect or by recovering S and comparing the output stream to the observed one.

4 Summary of Results

The complexity of the attacks described in the previous sections are summarized in the table below. For the synthetic constructions an asymptotic analysis was done while for LILI-128, RC4 and SOBER-t32, the analysis was done for the recommended parameters of the ciphers. The parameters n,t,T,k and M are as defined in the relevant subsection. For the sake of simplicity the results for the clocking constructions assume that the length of the clocking LFSR is the same as the length of the data LFSR. Note that there are many possible tradeoffs between the various parameters, and the table describes only one of the potential combinations in each case.

5 Summary

We have shown that fault attacks are an extremely powerful technique for attacking stream ciphers. We demonstrated their applicability to a wide variety of

Attack	#Faults	Data	Time	Space
Filtered LFSRs (known filter)	t	$t2^t$	$\binom{n}{k}2^t + n^3$	$t2^t + n^2$
Filtered LFSRs (unknown filter)	2^t	$t2^t$	$\binom{n}{k}t2^t + n^3$	$t2^t + n^2$
Clock controlled (faults in clock register)	n	n^2	n^3	n^2
Clock controlled (faults in data register)	1	n	$\binom{n}{k}n + n^3$	n^2
FSM filtered LFSR (totally randomized)	nM^2	nM^2	$nM^2 + n^3$	n^2
FSM filtered LFSR (faults in FSM)	$T^2\frac{n}{\log M}$	$T^3\frac{n}{\log M}$	n^3	$T^3\frac{n}{\log M} + n^2$
LILI-128	10K	1M	2^{25}	1M
SOBER-t32	1K	100K	2^{30}	100K
RC4	2^{16}	2^{26}	2^{26}	2^{16}

Fig. 3. Summary of out results

synthetic and actual schemes, and identified several interesting open problems. Further work on this subject could include practical attacks on smart card implementations of stream ciphers, and finding attacks on more general classes of stream ciphers which are not based on LFSR's or arrays of updated values.

References

1. Ross Anderson *Optical Fault Induction*, June 2002
2. Boneh, Demillo, and Lipton *On the Importance of Checking Cryptographic Prtocols for Faults*, September 1996
3. Biham, Shamir *A New Cryptanalytic Attack on DES: Differential Fault Analysis*, October 1996
4. E. Dawson A. Clark J. Golic W. Millan L. Penna L. Simpson *The LILI-128 Keystream Generator*, November 2000.
5. Shai Halevi, Don Coppersmith & Charanjit Jutla *Scream an efficient stream cipher*, June 2002
6. Coppersmith, Krawczyk & Y. Mansour *The Shrinking Generator*, Proceedings of Crypto'93, pp.22–39, Springer-Verlag, 1993
7. Philip Hawks & Gregory G. Rose *Primitive Specification and Supporting Documentation for SOBER-t32 Submission to NESSIE*, June 2003.
8. Itsik Mantin & Adi Shamir *A Practical Attack on Broadcast RC4*, FSE 2001
9. Jovan Dj. Golic & Guglielmo Morgari *On the Resynchronization Attack*, FSE 2003
10. Jovan Dj. Golic & Guglielmo Morgari *Correlation Analysis of the Alternating Step Generator*, Designs, Codes and Cryptography, 31, 51–74, 2004
11. S. Dubuc, *Characterization of linear structures*, Designs, Codes and Cryptography, vol. 22, pp. 33-45, 2001
12. Nicolas Courtois and Willi Meier, *Algebraic Attacks on Stream Ciphers with Linear Feedback*, Eurocrypt 2003
13. Steve Babbage, *Cryptanalysis of LILI-128*, Proceedings of the 2nd NESSIE Workshop, 2001
14. Steve Babbage, Christophe De Cannière, Joseph Lano, Bart Preneel, Joos Vandewalle, *Cryptanalysis of SOBER-t32*, FSE 2003
15. Joo Yeon Cho and Josef Pieprzyk, *Algebraic Attacks on SOBER-t32 and SOBER-128*, FSE 2004