Differential Cryptanalysis in Stream Ciphers

Eli Biham[1] Orr Dunkelman*[2]

[1]Computer Science Department, Technion.
Haifa 32000, Israel
biham@cs.technion.ac.il
[2]Katholieke Universiteit Leuven,
Dept. of Electrical Engineering ESAT/SCD-COSIC.
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
orr.dunkelman@esat.kuleuven.be

Abstract. In this paper we present a general framework for the application of the ideas of differential cryptanalysis to stream ciphers. We demonstrate that some differences in the key (or the initial state or the plaintext) are likely to cause predicted differences in the key stream or in the internal state. These stream differences can then be used to analyze the internal state of the cipher and retrieve it efficiently. We apply our proposed ideas to stream ciphers of various designs, e.g., regularly clocked LFSRs, irregularly clocked LFSRs such as A5/1, and permutation-based stream ciphers such as RC4.

Keywords: Differential cryptanalysis, Stream ciphers, RC4, A5/1, LILI-128, LILI-II, Trivium, Toyocrypt.

1 Introduction

Differential cryptanalysis [6] was suggested as a general method for analyzing various cryptographic primitives. The technique was widely applied to block ciphers and hash functions, and many new constructions of these primitives are specifically designed to withstand this attack.

The ideas of differential cryptanalysis were also used in the analysis of several public key cryptosystems. Patarin's 2R cryptosystem [27] was analyzed using differential cryptanalysis due to its block-cipher like description [4]. Recently, attacks on another multivariate public key cryptosystem using differential techniques was presented in [17, 13].

Differential ideas were also used in differential fault analysis [7]. In this kind of attack, the attacker can introduce errors during the computation, leading to an error in the output. By examining the difference between an unfaulty computation and a faulty one, the attacker can deduce information on the computation.

These ideas were used in various cryptanalytic attacks on stream ciphers. For example, in [30] it is shown that two IVs with some given difference may produce

* The research presented in this paper was partially supported by the Clore scholarship programme.

the same key stream. Other works on stream ciphers also explored differences, but till now, no comprehensive approach for these ideas in the framework of stream ciphers has been presented.

In this paper we show that differential ideas can be used in the world of stream ciphers. We show that a key difference (or even an initial value difference) can be used to predict the stream differences (with some probability). This phenomena is easily observed and detected where the key loading procedure and the key updating procedure are linear (as in an LFSR), as well as in nonlinear procedures that initialize the internal state of the stream cipher (like in RC4 [28]).

We define the terms *differential characteristics* and *differential* for stream ciphers. We note that as there are several types of stream ciphers, the terms is not a simple adaptation of the block cipher terms. For example, in synchronous stream ciphers, the differential characteristics can be defined either from the (key, IV) pair into the internal state, from the internal state to itself or from the internal state of the stream cipher into the key stream. As in cryptanalysis of block ciphers, differential characteristics are concerned with the exact evolution of differences, while differentials are only concerned with the input and output differences.

We then show that it is possible to use these characteristics to analyze stream ciphers. We show that the existence of good characteristics in a stream cipher is an evidence for a problem in the cipher's design. Moreover, differential characteristics can be used to explain many of the previous attacks on stream ciphers.

We discuss several attack scenarios: We start by discussing distinguishing attacks in the presence of several streams produced by different (key, IV) pairs. We continue with the use of differentials for key recovery attacks. Finally, we show a speed up in exhaustive search in the presence of differentials. We also shortly discuss the effects of our works on differential fault analysis of a stream cipher.

After defining the concept of differential characteristics in stream ciphers, we analyze a generic LFSR-based stream cipher (with a nonlinear combining function). We present a simple differential attack on Toyocrypt that requires only two streams of 128 bits each. We then show that the ideas are applicable also for nonlinearly clocked LFSRs, e.g., the GSM stream cipher A5/1 [9], which uses three LFSRs that are irregularly clocked. We also show that even ciphers based on permutations, such as RC4 [28], are susceptible to differential attacks.

By combining the results, we present a framework that stream cipher designers must be aware of. A stream cipher which has no high probability differentials (or even impossible differentials) is expected to be immune to resynchronization attacks, related-key attacks, and re-keying attacks. The last statement is even stronger for self synchronizing stream ciphers or ciphers that propose authenticated encryption, where the attacker can introduce differences through the plaintext (or ciphertext) as well.

Section 2 describes previous works and related ideas. In Section 3 we define differential characteristics and differentials for stream ciphers. Section 4 presents the differential characteristics for LFSRs and for A5/1 and ways for using them

in cryptanalysis. Section 5 suggests differential characteristics for RC4 along with ways of using them for the reconstruction of the internal state. In Section 6 we show how to describe previously published attacks on stream ciphers using the concept of differential characteristics, and interpret these new description. In Appendix A we give a short description of some of the ciphers we attack. We apply our results to Trivium, LILI-128 and LILI-II in Appendix B. Appendix C contains a short description of how to deal with differentials to increase the speed of exhaustive key search. Finally, Section 7 summarizes this paper.

2 Related Concepts

There are several approaches for attacking stream ciphers that are relatively close to the ones presented in this paper. These approaches are mostly ad-hoc techniques used to break a specific stream cipher.

In his attack on LILI-128 [11], Babbage considers several streams produced by the same key with different IVs [2]. This model is a realistic one, as most stream ciphers are designed to support a fixed key with a multiple possible IVs.

The keying mechanism of LILI-128 loads the XOR of the key and IVs into the state machine without any processing. When the same key is used with different IVs, guessing a small part of the key (as done in Babbage's attack) leads to the knowledge of a small part of the internal state in many key streams. This was used to attack LILI-128 with about 64 output streams (produced under the same key with different public IVs) with time complexity of about 2^{39} operations.

Another related concept is in the attacks presented in [15, 21, 23], where RC4 is analyzed. In these attacks, by using different known IVs, it is possible to gather statistical information on the internal state of RC4. While in Babbage's attack any set of IVs would be suitable to the attack, in these attacks on RC4, the attacker waits for IVs that generate some specific internal state that has some easy to identify property. Once the property is identified, it discloses information about the secret key (as for different keys different IVs achieve this property).

In [30] a different approach is presented in an attack on Py [30]. The attack uses different IVs with a fixed key. While the RC4 attacks usually wait for some internal state to be generated, this attack on Py [5] shows that for any given key, there is a set of 2^{16} IVs of which two are expected to initialize the same internal state. Hence, for these IVs, the same output stream is expected. This attack has the strictest conditions of the other attacks, as it assumes a chosen IV scenario (which is still a very probable model).

The concept of weak key classes is also related to our work. Some of the applications of our proposed concepts reduce the time complexity of various attacks, like exhaustive key search, for special groups of keys. Just like in block ciphers, there are keys for which the encryption process is not as strong as for most keys. In the case of stream ciphers, this approach is usually applicable to ciphers where some components depend on the key alone, and are not updated. For example, in the attack on ABCv2 [1] presented in [29] it was shown that if the key initialization leads to 32 internal constants whose least significant bits

are all set to 0, then there is a bias in one of the bits of the stream words. This bias is then used for a standard distinguishing and key recovery attacks. One can view also the attacks on RC4 as a weak keys attacks (as the statistical property holds for some of the (key,IV) pairs).

We note that our framework can treat even the cases of a repeated nonce (IVs) attacks which recently been proposed against several authenticated encryption schemes [31]. In the scenario of repeated nonce, the attacker can ask for the encryption/decryption of different messages under the same (key,IV) pair. When the encryption process is independent of the texts, this has no effect on the security, but when the encryption process depends on the provided plaintexts, i.e., in a synchronous stream cipher or in an authenticated-encryption mode of operation, this might result in interesting attacks as presented in [26, 31].

3 Differential Characteristics of Stream Ciphers

There are several kinds of stream ciphers: synchronous, self synchronizing, and those which provide authentication. Each of these options defines the interface that the stream cipher has and thus defines the possible differentials for the cipher.

3.1 Synchronous Stream Ciphers

Most of the stream ciphers are synchronous stream ciphers which provide no authentication. In this case, the stream cipher's only input is the keying material (that might include the IV). These ciphers can be defined using a set of three algorithms — an internal state initialization procedure $S = INIT(key, IV)$, where S denotes the internal state, an internal state update function $S = UPDATE(S)$, and and an output function $KS = OUTPUT(S)$ that produces the key stream.

For such ciphers there are three types of differential characteristics:

- $(\Delta key, \Delta IV) \xrightarrow{INIT} \Delta S$, where a difference in the key or the IV generates a difference in the internal state,
- $\Delta S \xrightarrow{UPDATE} \Delta S$ through the internal state update function,
- $\Delta S \xrightarrow{OUTPUT} \Delta KS$, where a difference in the internal state generates a key stream difference.

It is also possible to define the concatenation of these characteristics, i.e., $(\Delta key, \Delta IV) \xrightarrow{INIT+OUTPUT} \Delta KS$. Moreover, if $(\Delta key, \Delta IV) \xrightarrow{INIT} \Delta S$ has probability p_1 and $\Delta S \xrightarrow{UPDATE} \Delta KS$ has probability p_2, then $(\Delta key, \Delta IV) \xrightarrow{INIT+UPDATE} \Delta KS$ has probability $p_1 \cdot p_2$ (assuming the characteristics are independent of each other).

Up till now we discussed differential characteristics, i.e., a prediction on the exact evolution of differences through the various functions of the cipher. Just like in block ciphers, we are usually interested only in the input and the output

differences. Such a prediction is named in block ciphers *a differential* and we use the same name for stream ciphers as well. A differential of a stream cipher is a prediction that a given input difference (be it the key, the IV, or the internal state) produce some output difference (be it the key stream or the internal state). As in block ciphers, we are not interested in what exactly happens in the cipher when the differential is satisfied, but only with the probability of the differential. We note that just like in block ciphers, it is hard to compute the exact probability of a given differential, but we can of course use the probability of a given characteristic with the same input and output differences as a lower bound.

The reader might wonder whether the above separation is necessary, as for such stream ciphers the attacker can control only the difference $(\Delta key, \Delta IV)$ and observe only the differences of the key stream. We shall demonstrate later, that even differentials from the internal state to the key stream may be very useful for cryptanalysis.

3.2 Self Synchronizing Stream Ciphers

For self synchronizing stream ciphers, or ciphers that offer an authenticated encryption, the UPDATE and the OUTPUT functions have an additional input, the plaintext. In case of authenticated encryption, there is an additional algorithm TAG, which transforms the internal state into a MAC tag. While in case of self synchronizing stream ciphers, there is a decryption process. The handling of the latter case is similar to the $OUTPUT$ function though being its inverse (with a fixed state).

We now summarize the functions for such stream ciphers:

- $S = INIT(K, IV)$, where K is the key and IV is the IV.
- $S = UPDATE(S, P)$, where P is the plaintext word.
- $C = OUTPUT(S, P)$, where C is the resulting ciphertext word.
- $(S, P) = DECRYPT(S, C)$, where C is the ciphertext word and where P is the resulting plaintext word.
- $tag = TAG(S)$, which is the tag produced by the internal state (if authenticated encryption is offered).

The possible characteristics in this case are:

- $(\Delta key, \Delta IV) \xrightarrow{INIT} \Delta S$, where a difference in the key or the IV generates a difference in the internal state,
- $\Delta S \xrightarrow{UPDATE} \Delta S$ through the internal state update function,
- $(\Delta S, \Delta P) \xrightarrow{UPDATE} \Delta S$ is the differential that predicts the difference in the internal state given the current difference of the internal state and the difference in the plaintext word.
- $(\Delta S, \Delta P) \xrightarrow{OUTPUT} \Delta C$ is the differential that predicts the ciphertext difference given the internal state difference and the plaintext difference.

– $(\Delta S, \Delta C) \xrightarrow{DECRYPT} (\Delta S, \Delta P)$ is the differential that predicts the plaintext difference given the internal state difference and the ciphertext difference.
– $\Delta S \xrightarrow{TAG} \Delta tag$ is the differential that predicts the tag difference.

4 Differentials of LFSR Based Stream Ciphers

Let us consider the most basic stream cipher design — an LFSR (with a primitive feedback polynomial) with a combining logic which outputs one bit of key stream each clock. We shall assume that this LFSR is initialized with a (key, IV) pair of the length of the LFSR. Then the LFSR starts to produce key stream.

It is easy to see that given two (key, IV) pairs, denoted by (k_1, IV_1) and (k_2, IV_2), such that $\Delta k = k_1 \oplus k_2$ and $\Delta IV = IV_1 \oplus IV_2$ are known, then the difference in the internal state is known for any given number of clocks. This allows to estimate the output difference of the combining function, i.e., we can estimate the key stream differences. For example, when all the bits that the combining function uses have no difference, then there is no difference in the key stream. There are cases for which the difference would have some probability, e.g., if there is a difference in only one bit which is combined through a monomial of degree 3, there is portability of 1/4 that there is a difference in the key streams in the corresponding bit.

This fact can also be used to distinguish the output of the stream cipher under two related keys (or IVs) from two random strings. Moreover, in some cases, it can be used to identify information about the internal state. For example, in the previous example, where the only difference to the combining function is in one bit that enters a monomial of degree 3, in case there is a difference in the output stream, we can detect the values of two bits of the LFSRs.

4.1 Differential Cryptanalysis of Toyocrypt-HS1

We demonstrate the above concept on Toyocrypt-HS1 [32], which is a 128-bit key stream cipher. The cipher is composed of a 128-bit LFSR which is initialized with a 128-bit key and outputs each round one bit using the combining function

$$f(s_0, \ldots, s_{127}) = s_{127} \bigoplus_{i=0}^{62} s_i s_{\alpha_i} \bigoplus s_{10} s_{23} s_{32} s_{42} \bigoplus \\ s_1 s_2 s_9 s_{12} s_{18} s_{20} s_{23} s_{25} s_{26} s_{28} s_{33} s_{38} s_{41} s_{42} s_{51} s_{53} s_{59} \bigoplus \Pi_{i=0}^{62} s_i$$

where $\alpha_i \in \{63, \ldots, 125\}$ is a constant list of indices with the property that each of the values appears once.[1]

4.1.1 Distinguishing Attack on Toyocrypt It is easy to see that given two keys whose difference is known, the differences in the LFSRs can be predicted for

[1] We adopt the notations of [32], i.e., the LFSR is clocked and s_{127}, the most significant bit, is fed back through the one-to-many LFSR configuration.

the consecutive stream. For example, if the key difference is in bit s_{126}, then the key stream difference is zero in the first bit and one in the second bit necessarily.

Thus, given two streams generated by related keys, we check whether the first output bit is the same and the second one differs. If this is the case, we output that the stream cipher in use is Toyocrypt. Otherwise, the cipher cannot be Toyocrypt. The attack has a success rate of 75% (it might deduce that a different stream cipher is Toyocrypt in 25% of the cases).

Given several pairs of output streams, we can increase the success rate of the attack. If any of the pairs of streams lead to the conclusion that the pair was not produced using Toyocrypt, then the cipher is necessarily not Toyocrypt. Otherwise, given s pairs of streams, the probability of error is 0.25^s.

4.1.2 Improving Exhaustive Key Search of Toyocrypt We can also improve the time complexity of an exhaustive key search using the above differential. Assume that the attacker tries a key K, for which the first bit of the key stream does not agree with the actual key stream. Then, the key $K \oplus s_{126}$ can also be automatically discarded without any consideration, reducing the expected time complexity of an exhaustive key search by a factor of 2^{126} trials. This fact might seem unnatural, as we are given only one key stream, but as the differential has probability 1 it can be used to predict (one bit of) the key stream, thus resulting in the ability to try two keys at the expense of one.

Thus, even though the exhaustive key search attack does not require the related-key model, it can still benefit from the existence of differentials. This has a close relation to the ideas of complementation properties. For example, DES's complementation property can be viewed as a related-key differential with probability 1. Using it, there is a speed up in exhaustive key search by a factor of 2.

4.1.3 Differential Key Recovery Attack on Toyocrypt Toyocrypt-HS1 is updated in a one-to-many manner (i.e., bit s_{127} is XORed into several bits during the clocking of the LFSR). Thus, differences that are introduced in a relatively least significant position, do not propagate through the LFSR for a long period of time (thus maintaining the low hamming weight of the difference). For example, if the difference is $\Delta S = s_0$, then it would take 128 clockings in order for this difference to affect other bits.

Given two key streams produced by two keys whose initial state difference is $\Delta S = s_0$ it is possible to reconstruct most of the internal state. This follows from the fact that the differential $\Delta S = s_0 \xrightarrow{OUTPUT} s_{\alpha_0} = \Delta KS$ holds with probability 1. If $s_{\alpha_0} = 0$ then there is no difference in the first bit of the key streams, while otherwise there is a difference.[2] We note that the next bit may

[2] We can assume that $s_1, \ldots, s_{62}$ are not all 1. Otherwise, if the described attack fails in retrieving the internal state, we may deduce that this assumption is wrong, thus disclosing 62 bits of the internal state, and allowing for repetition of the attack with the required modifications.

produce various differences (as s_1 is part of several nonlinear terms). However, with probability $1 - 2^{-17}$ the difference in the second bit discloses the value of s_{α_1} in a similar manner. Thus, by disregarding the output difference when there is a difference in s_{10}, s_{23}, s_{32}, or s_{42}, almost the entire register can easily be recovered with a success rate of $(1 - 2^{-17})^{17} = 0.99987$. The remaining seven bits (corresponding to $s_0, s_{\alpha_{10}-10}, s_{\alpha_{23}-23}, s_{\alpha_{32}-32}, s_{\alpha_{42}-42}, s_{126}$, and s_{127}) can easily be recovered by exhaustive search.

4.2 Differential Cryptanalysis of A5/1

Our technique can be applied to irregularly clocked LFSRs, where the number of clocks of the LFSR is determined by some pseudo-random controller. A5/1 is the stream cipher used to protect the privacy of GSM calls. The key length is 64 bits, and the IV (called a frame number) is 22-bit long. The key and the IV are used to initialize 3 LFSRs which are irregularly clocked. The description of A5/1 can be found in Appendix A.1

The combined 86 bits are used to linearly initialize an internal state of 64 bits. Thus, A5/1 has many equivalent differentials of the form $(\Delta K, \Delta IV) \xrightarrow{INIT} \Delta S$. Each state difference ΔS (after loading the 86 bits) has exactly 2^{22} possible $(\Delta K, \Delta IV)$ tuples that generate ΔS.

The best differential[3] (for which $\Delta S \neq 0$ after loading the key and frame number) has input difference $(\Delta k, \Delta IV) = (0000015F102D07E2_x, 08EDB3_x)$. With probability 1 it leads to two internal states $S^1 = (R_1^1, R_2^1, R_3^1)$ and $S^2 = (R_1^2, R_2^2, R_3^2)$ that have a difference $\Delta S = S^1 \oplus S^2 = (R_1^1 \oplus R_1^2, R_2^1 \oplus R_2^2, R_3^1 \oplus R_3^2) = (0, 000800_x, 0)$, i.e., $R_1^1 \oplus R_1^2 = 0$, $R_2^1 \oplus R_2^2 = 000800_x$, and $R_3^1 \oplus R_3^2 = 0$.

Each of the two states is clocked for 100 clocks using the irregular clocking mechanism. At the beginning there is no difference in the bits that enter the clocking mechanism, and thus, the number of times each register in S^1 is clocked is equal to the number of times its counterpart is clocked in S^2. Therefore, the difference in R_2 propagates till it arrives to bit $R_2[20]$. The next time R_2 is clocked after this event, the bit difference is fed back to register R_2, and the difference between the states becomes $(0, 200001_x, 0)$. After another clocking of R_2 the difference between R_2^1 and R_2^2 is 000003_x, i.e., in bits $R_2[0, 1]$.

The difference between the R_2 values continues to propagate, i.e., clocked whenever R_2^1 and R_2^2 are clocked (which happens together, as both registers have no difference in the clock controlling bits), until the difference reaches the clock controlling bit. Our findings show that with probability of $0.197 = 2^{-2.346}$ the difference between the states becomes $(0, 001800_x, 0)$, i.e., all registers were clocked the same number of times, and the difference does not affect the clock controlling bits anymore.

From this point, again, there is no difference in the clock controlling bits, and the difference of R_2 propagates with probability 1, until the difference in

[3] There are 2^{22} (key, frame number) tuples that lead to exactly the same internal state. The differential we describe can start in $2^{22} - 1$ other values for $(\Delta key, \Delta IV)$ tuples that lead to a difference $(0, 000800_x, 0)$ after the loading of the keys.

Event	Number of Times R_2 was Clocked	Internal State Difference	Probability
After key initialization	0	$(0, 000800_x, 0)$	1
When the difference arrives $R_2[21]$	10	$(0, 200001_x, 0)$	1
When the difference leaves $R_2[21]$	11	$(0, 000003_x, 0)$	1
When the difference arrives clocking	20	$(0, 000600_x, 0)$	1
When the difference passes clocking	22	$(0, 001800_x, 0)$	$2^{-2.346}$
When the difference leaves $R_2[21]$	32	$(0, 000005_x, 0)$	1
When the difference arrives clocking	40	$(0, 000500_x, 0)$	1
When the difference passes clocking	43	$(0, 002800_x, 0)$	$2^{-2.890}$
When the difference leaves $R_2[21]$	53	$(0, 00000F_x, 0)$	1
When the difference arrives clocking	60	$(0, 000780_x, 0)$	1
When the difference passes clocking	64	$(0, 007800_x, 0)$	$2^{-3.439}$
When the difference leaves $R_2[21]$	74	$(0, 000011_x, 0)$	1

Table 1. The evolution of A5/1 internal state differences from a $(\Delta key, \Delta IV)$ difference of $\Delta key = 0000015F102D07E2_x$ and $\Delta IV = 08EDB3_x$.

R_2 is 300000_x. As before, this difference affects the least significant bits of R_2 every time R_2 and R_2^* are clocked. This difference evolves into $(0, 000500_x, 0)$ and passes through the clocking mechanism with probability $0.135 = 2^{-2.890}$. Then, the difference wraps round the register to become $(0, 00000F_x, 0)$. This difference propagates to $(0, 000011_x, 0)$ with probability $0.092 = 2^{-3.439}$. We outline the evolution of the difference of the internal states in Table 1.

After 100 clockings in which the output is discarded, A5/1 starts to output key stream bits. When the output stream starts to pour out, the difference in the output streams can be identified (if the two states have the predicted difference).

The problem is that the exact difference is unknown, as the exact number of times register R_2 was clocked during the initialization is unknown. However, the most probable number of clockings of register R_2 is 76 [24]. This number of clockings is encountered with probability 0.092, i.e., with probability $0.197 \cdot 0.135 \cdot 0.092 \cdot 0.092 = 0.000225 \approx 1/4442$ the difference between the two states is indeed $(0, 000044_x, 0)$. We note that there are several similar differentials in which the number of times R_2 was clocked is slightly different (and whose probabilities are slightly lower than 1/4442).

If the above differential holds, the first output bit has a zero difference, i.e., $(0, 000044_x, 0) \xrightarrow{OUTPUT} 0$ with probability 1. Actually, the first three bits of the output stream are necessarily without a difference, and with probability of at least 0.58 (the output stream may not necessarily differ even if there is a difference in the number of times each register is clocked) the first four bits have a zero difference.

We have experimentally verified the differentials with several tests of million samples each. Table 2 lists the most probable R_2 differences that were encoun-

Difference	Probability			Standard
	Average	Minimal	Maximal	Diviation
200008_x	0.001017	0.000962	0.001062	0.00002828
000011_x	0.000988	0.000957	0.001025	0.00001934
300004_x	0.000984	0.000937	0.001043	0.00002826
000022_x	0.000915	0.000874	0.000962	0.00002275
380002_x	0.000880	0.000836	0.000934	0.00002815
$3C0001_x$	0.000779	0.000727	0.000809	0.00002385
000044_x	0.000778	0.000741	0.000822	0.00002650
$1E0000_x$	0.000612	0.000581	0.000659	0.00002277
000088_x	0.000611	0.000572	0.000671	0.00002982
$0F0000_x$	0.000477	0.000448	0.000496	0.00001472
000110_x	0.000458	0.000432	0.000487	0.00002367

Average corresponds to the average observed probability of all samples.
Maximal corresponds to the sample with the maximal observed probability.
Minimal corresponds to the sample with the minimal observed probability.
The standard deviation is of the probabilities observed in various samples.

Table 2. R_2 differences after initialization from an initial difference of $(0, 800_x, 0)$ when R_1 and R_3 have no difference. Each test had 1,000,000 test samples.

tered when R_1 and R_3 have no difference. As can be seen from the table, the actual probability that internal state difference $\Delta S = (0, 000800_x, 0)$ develops into $\Delta S = (0, 000044_x, 0)$ after 100 clocks is actually $0.000778 \approx 1/1285$, which is about 3.5 times larger than the predicted value. This is explained by two factors. First, our probability calculation takes into consideration at most 10 bits of each register (i.e., if the difference in the number of times each register is clocked becomes the same only after 11 clockings of any of the registers then we did not count it as part of the probability of a "right" transition). The second factor is the fact that it appears that internal states which pass one of the iterations, are more likely to pass another iteration and

We note that the most probable difference in R_2 after the initialization is 200008_x, which corresponds to 73 clockings of R_2 during the initialization. This is explained by the fact that for pairs of values for which the differential holds, R_2 is slightly less likely to be clocked (as the register values satisfy some conditions).

The difference in R_2 is with probability 0.0085 one of the top 11 possible values, and in total with probability of about 0.01, the differential is followed (up to the number of times R_2 is clocked).

4.3 Differential Attacks on A5/1

The differentials of A5/1 can be used in several ways: The simplest way is when two related encryptions are performed. In the related-key model [3], the attacker can observe several output streams produced by different (but related) keys. As in the case of Toyocrypt-HS1, when two output streams have a difference

predicted by the differential, we can use the correlations between the internal state and the stream differences to learn information about the internal state.

For example, by identifying that the difference has just passed bit $R_2[21]$, we know that the registers R_2^1 and R_2^2 were clocked. The disclosed information reduces the number of possible internal states that the attacker has to check.

In the case of A5/1 this information can also be used as an indication on how many times R_2^1 and R_2^2 were clocked during the initialization phase. Thus, we can use the streams produced by the respective keys in attacks that require this knowledge, e.g., the attack of [24].

4.3.1 Speeding Exhaustive Key Search Another instance when the differentials can be used is in an exhaustive key search. Assume that $\Delta S \xrightarrow{OUTPUT} \Delta KS$, given a key stream KS if a tested internal state S produces $KS \oplus \Delta KS$, then it is very likely that the actual internal state is $S \oplus \Delta S$. We explore this approach in Appendix C.

4.3.2 Differential Fault Analysis of A5/1 The final scenario where these differentials can be used is in differential fault analysis [7]. In the attack, a fault is introduced to some of the computation (memory, computation, etc.), and the change (or lack of it) in the output is used for the attack. Essentially, the differentials can be found in case of a fault is introduced in $R_2[11]$, and then used to retrieve the internal state.

5 Differential Cryptanalysis of RC4

RC4 [28] is a widely spread stream cipher. It is used in many security protocols like SSL, IPsec, and WEP. As such, it received a lot of cryptanalytic attention [15, 16, 19–21, 25]. Despite all the weaknesses found both in RC4 and in the way it is used, until now no shortcut attacks on RC4 are known. We give the description of RC4 in Appendix A.2. We note that all additions and subtractions with respect to RC4 are done modulo 256.

5.1 Differentials of RC4

In [19] it was shown that there are good differentials for keys of length 256 bytes. The difference between the two keys is in the last byte, so that after the initialization, the two internal states differ in three bytes. Another differential suggested in [19] is to have in the last two bytes a difference of the form $(+\delta, -\delta)$ for some non-zero δ. In both cases the output streams are expected to be the same in the first few bytes.

As keys of 256 bytes are very rarely used, we concentrate on keys of 256 bits which are more common (e.g., used in some WEP implementations). Thus, a

difference in the key affects the internal state several times during the initialization phase. This leads to a more complex differentials in the internal state, and of course, in the output stream.

Let K and K^* be two 256-bit keys. Let $K[j]$ by the j'th byte of the key, and let the key K^* be:

$$K^*[i] = \begin{cases} K[i] & i = 0, \ldots, 29 \\ K[i] + 1 & i = 30 \\ K[i] - 1 & i = 31 \end{cases} \quad (1)$$

During the initialization phase, the first thirty steps of the initialization are the same (as the key bytes are the same). If the value of j is updated to be 30 when $i = 30$ (for a random key this event happens with probability 2^{-8}), then $j^* = 31$. Due to this difference, the two states, denoted by S and S^*, respectively, differ in bytes 30 and 31, as in S no swap is performed, while in S^* the two values $S^*[30]$ and $S^*[31]$ are swapped. In the next step $i = 31$, if j is updated to 31 (which again happens with probability 2^{-8}), then j^* is updated to 30. After the second swap in S^*, the two states become the same. The overall probability that the two states are the same after 32 steps of initialization is[4] 2^{-16}.

The same event can happen for $i = 62, 63$ in the initialization (again with probability 2^{-16}), and again when $i = 32m - 2, 32m - 1$, for $m = 3, \ldots, 7$ during the the initialization. As this happens eight times before the initialization of the states is complete, the probability that $S = S^*$ after initialization is[5] 2^{-128}. Once such two keys are found, they produce the same output stream, as there is no difference in the state after the state initialization.

Note that we can have the two consecutive bytes with the difference in any location in the key, i.e., have a difference of the form $K[20] = K^*[20] - 1$, $K[21] = K^*[20] + 1$. This does not affect the probability that the two keys generate the same internal permutation. Thus, each key has 62 counterparts (31 possible pairs of consecutive bytes, and two options for the difference $(-1, +1)$ or $(+1, -1)$). Each of these 62 differences has probability 2^{-128} to generate the same output stream.

5.1.1 Exhaustive Key Search on RC4 This observation means that the time complexity of exhaustive key search on 256-bit RC4 is $2^{256} - 62 \cdot 2^{128}$, as in case the differential holds (which we can check given only one of the keys), we do not need to check two keys, but only one of them. To identify whether a given key generates the same internal state as another keys, we check the value of j during the initialization. For example, if the value of j when $i = 30 + 32m$ is $j = 30 + 32m$ and when $i = 31 + 32m$ the value of j is $j = 31 + 32m$, for all $m = 0, \ldots, 7$, then the key K that caused these values, has a counterpart K^*,

[4] Another key difference is $(2, -2, -2, 2)$ (i.e., difference of 2 in bytes 28 and 31, and difference of -2 in bytes 29 and 30). For this key difference, the probability of the transition is slightly lower, i.e., $\frac{254}{256} \cdot 2^{-16}$. The difference $(3, -3, 0, -3, 3)$ also passes this phase with probability $(\frac{254}{256})^2 \cdot 2^{-16}$.

[5] For keys with the difference $(2, -2, -2, 2)$ the probability is $2^{-128.1}$.

for which the internal state is the same after initialization. This K^* does not need to be checked, as it causes the same internal state.[6]

5.2 Differential Attacks on RC4

Consider two keys with difference as shown in Equation (1). With probability 2^{-112} they generate the same state after 254 steps of the initialization. Then, the last two steps of initialization affect four state bytes. Thus, there is a difference between the states in eight positions. This leads to two output streams that are very close to each other at the beginning, just like in the case of [19].

As long as j evolves in the same way for the two keys, the number of bytes in which the internal state defers remains eight. Thus, we expect that about 32 RC4 output bytes are generated before j defers for the two keys. Each of these 32 output bytes has probability $(248/256)^2 = 0.938$ to be equal in the two key streams, so we expect that about 30 of these 32 bytes are equal in the two key streams. One can incorporate this fact to the exhaustive key search on RC4 to further reduce its time complexity (Appendix C).

It is also possible to extract some probabilistic key information about the key once these two cases are found. When the above process happens, then $30 = j = 29 + S[30] + K[30]$ and $31 = j = 30 + S[31] + K[31]$. There is a high probability (of about 0.79) that $S[30]$ was not altered by the key initialization process so far. If this is the case, then $S[30] = 30$, and thus $K[30] = 227$. The same is true for $S[31]$ and $K[31]$, leading to the fact that when such a pair is found, it is very likely that $K[31] = 226$. Thus, 256-bit keys which do not affect $S[30]$ and $S[31]$ in the first 30 steps of initializations and satisfy $K[30] = 227, K[31] = 226$, are more likely (along with their counterpart) to satisfy the above differential characteristic.

We note that these observations can be adapted to any key length of RC4. However, for keys shorter than 22 bytes (176 bits), the probability of two keys to generate the same internal state in the manner described earlier is so small, such that the expected number of pairs of keys which satisfy the characteristic is smaller than 1. Thus, for short keys there might be no such instances. On the other hand, as the key grows longer, the number of times the initialization process has some probability is reduced. For example, keys of 256 bytes have probability 2^{-16} to generate the same internal state. When taking into consideration all the $2 \cdot 255$ characteristics a given 256-bit key can be paired to generate the same internal state, the time complexity of exhaustive key search becomes about $2^{2048} - 2^{2041}$. We note that the number of internal states is about 2^{1700} (of which 2^{1684} are consistent with RC4's initialization and update procedures), hence it is obvious that many keys lead to the same internal state.

Another use for differentials of RC4 is in combination with the attack by Mantin based on the glimpse property [23]. In that attack, the attacker uses the glimpse property that states that if z is the output word, then $\Pr[S[j] = i - z] \approx$

[6] Using all the other differentials, the total number of possible states for a 256-bit key is about $2^{256} - 866 \cdot 2^{128}$.

2/256 (rather than the expected 1/256). The extra 1/256 probability is due to the case $j = S[i] + S[j]$. The attack is based on the standard use of a bias of some event. Thus, as the bias of this property is about 1/256, the attacker needs about $c \cdot (1/256)^2$ samples for the application of the attack.

Assume that we have a differential[7] that predicts $\Delta S[i] = \delta_1$, $\Delta S[j_{after}] = -\delta_1$ and $\Delta j_{before} = -\delta_1$. If this is the case, Δj_{after} is expected to be 0, as $\Delta i = 0$ as well, then we expect $\Delta z = i - \delta_1$ as well whenever the **glimpse holds** (which happens with probability 1/256). Thus, if the probability of the differential is p_1, then with probability $p_1/256$ we expect that the specific output byte have a difference $i - \delta_1$.

This value is below the value expected for a random stream. However, if there exists a second differential of this form (with δ_2 instead of δ_1 and with probability p_2 instead of p_1), then we can use three streams (reusing one of the streams), and the value of the specific output byte is known in all three of them with probability $p_1 \cdot p_2/256$. If p_1 and p_2 are sufficiently high (i.e., $p_1 \cdot p_2 > 1/256$), then this event can be easily detected with probability higher than the random case, thus clearly indicating when the glimpse property is satisfied. This can be used to reduce the required key stream used in [23], and even reduce the time complexity, as the cases for which the glimpse property is satisfied are easily identifiable.

Another implication our result have is on attacks based on time-memory tradeoff on the internal state. For long keys the possible internal states does not compromise a set in the size of the key (i.e., have the same number of internal states as number of keys). This can slightly improve such attacks on RC4.

6 Describing Previous Attacks Using the Differential Notations

As we noted earlier, there are several attacks on stream ciphers titled as differential attacks. These attacks do have a differential behavior, as they are concerned with the difference that are caused due to a difference in the plaintext or IVs.

6.1 A Differential Attack on Py and PyPy

In [30] a differential in the key/IV-setup of Py (and PyPy) is given. The differential shows that with probability $2^{-23.2}$ two IVs which differ in two consecutive bytes are likely to produce the same internal state. This property is then used to retrieve the key, as the specific IVs for which the differential holds suggest information about the key.

The IV setup of Py and PyPy accepts an IV of size *ivsize* and initialize a byte permutation P along with a byte array of size *ivsize* called EIV. For this process, the authors of [30] have observed that if there is a difference $\Delta iv[i] = 1$

[7] Note that we need a differential that predicts several things, and we are not concerned in the exact differential path that led to this situation.

and $\Delta iv[i+1] \neq 0$ for any byte $i \geq 1$, then the first loop of the IV initialization is the same.

Then, s is updated to have a difference of 1 (up to carry) when the difference is introduced. This difference affects $s0$, and in turn $EIV[i]$. In the next round, there is a probability of about 2^{-8} that the difference in s is canceled, which leads to a difference after the second IV initialization process in $EIV[i]$.

In the second loop this process is repeated, while this time, the attacker needs that the difference in $EIV[i]$ is canceled as well as the difference in s (when processing $iv[i]$ and $iv[i+1]$). Each of these two events happens with probability 2^{-16}, and the resulting differential has a total probability of 2^{-24}. In reality, due to the structure of the basic *internal_permutation*() and carry issues, the actual probability is slightly higher, i.e., $2^{-23.2}$.

6.2 A Differential Attack on Helix

In [26] the stream cipher Helix [14] is attacked. Helix offers an authenticated encryption with keys of 256 bits and IVs of 128 bits. The user key is used to generate a "working key" which along with the IV (called nonce) interact with the plaintext and the internal state to produce the output word. After the encryption of the message, the internal state is manipulated to extract an authentication tag.

The attack of [26] is against Helix when the same key is used with the same IV several times. While in this case there is no reason to believe the encryption to be secure, the attacker is able to extract information about the internal state of the cipher. This is due to a differential from the input to the output word. If the same internal state is achieved (due to the same (key, IV) pair), and there is an input difference Δ in the plaintext word, the attacker observes a Δ' output difference in the output word (the following key stream word)[8], and thus it is possible to identify the two respective internal state words x, y, as they satisfy:

$$\Delta' = (x + y) \oplus (x + (y \oplus \Delta)).$$

This equation corresponds to the fact that the difference Δ is introduced to the word y (by means of XOR), which is then added with x and the result is the output word. Thus, it is claimed that given 93 pairs with specially chosen Δ's it is possible to find x and y efficiently.

We note that the differential $(\Delta S = 0, \Delta P = \Delta) \xrightarrow{UPDATE+OUTPUT} \Delta KS = \Delta'$ is conditional on the values of x and y. While for $x = 0$ it is always holds that $\Delta = \Delta'$, for other values of x it is possible to associate probabilities of the differential (y is equal to an XOR of a fixed unknown word and the plaintext word). For example, setting $\Delta = \Delta' = 1_x$, we obtain that the differential holds with probability $1/2$, but when it holds then the least significant bit of x is necessarily zero. In [26] an algorithm for finding the possible (x, y) pairs is given using 93 values for Δ (and observing which differential was chosen).

[8] We note that the actual difference that is observed is $\Delta' \lll 29$.

7 Summary

We have presented a new technique for the analysis of stream ciphers. The existence of differential characteristics and differentials in the stream cipher has several implications with respect to the security of stream ciphers. Thus, stream cipher designers should take these issues into consideration when designing new stream ciphers.

Stream ciphers with high probability differentials are susceptible to many attacks: distinguishing and key recovery in the related-key/IV model, faster exhaustive key searches, and fault analysis. Stream ciphers that also offer authenticated encryption are also susceptible to repeated nonce attacks, as well as forging of the tags in case there exist good differentials.

We also found out that even characteristics that deal with the evolution of internal state differences (without considering their affect on the output) can be used for analysis. These characteristics are especially useful for exhaustive key search or time-memory tradeoff attack which aim the internal state of the cipher rather its key space.

Thus, we conclude that differential cryptanalysis is a versatile and important tool in the cryptanalyzer toolbox, even when she tries to break stream ciphers.

8 Acknowledgments

The authors would like to thank Ed Dawson for providing us with [32].

References

1. Vladimir Anashin, Andrey Bogdanov, Ilya Kizhvatov, Sandeep Kumar, *ABC Is Safe And Sound*, eSTREAM submission, 2006. Available online at *http://www.ecrypt.eu.org/stream/papersdir/079.pdf*.
2. Steve Babbage, *Cryptanalysis of LILI-128*, preproceedings of NESSIE 2nd workshop, Egham, 2001.
3. Eli Biham, *New Types of Cryptanalytic Attacks Using Related Keys*, Journal of Cryptology, vol. 7, number 4, pp. 229–246, Springer-Verlag, 1994.
4. Eli Biham, *Cryptanalysis of Patarin's 2-Round Public Key System with S Boxes (2R)*, Advances in Cryptology, proceedings of EUROCRYPT 2000, Lecture Notes in Computer Science 1807, pp. 408–416, Springer, 2000.
5. Eli Biham, Jennifer Seberry, *Py: A fast and secure stream cipher using rolling arrays*, eSTREAM submission, 2005. Available online at *http://www.ecrypt.eu.org/stream/p2ciphers/py/py_p2.ps*.
6. Eli Biham, Adi Shamir, *Differential Cryptanalysis of the Data Encryption Standard*, Springer-Verlag, 1993.
7. Eli Biham, Adi Shamir, *Differential Fault Analysis of Secret Key Cryptosystems*, Advances in Cryptology, proceedings of CRYPTO 97, Lecture Notes in Computer Science 1294, pp. 513–525, Springer, 1997.
8. Dan Boneh, Richard A. DeMillo, Richard J. Lipton, *On the Importance of Checking Cryptographic Protocols for Faults (Extended Abstract)*, Advances in Cryptology, proceedings of EUROCRYPT 97, Lecture Notes in Computer Science 1233, pp. 37–51, Springer, 1997.

9. Marc Briceno, Ian Goldverg, David Wagner, *A Pedagogical Implementation of the GSM A5/1 and A5/2 "voice privacy" encryption algorithms*. Available online at *http://www.scard.org/gsm/a51.html*.
10. Andrew Clark, Ed Dawson, Joanne Fuller, Jovan Dj. Golic, Hoon Jae Lee, William Millan, Sang-Jae Moon, Leone Simpson, *The LILI-II Keystream Generator*, proceedings of ACISP 2002, Lecture Notes in Computer Science 2384, pp. 25–39, Springer, 2002.
11. Ed Dawson, Andrew Clark, Jovan Dj. Golic, William Millan, Lyta Penna, Leonie R. Simpson, *The LILI-128 Keystream Generator*, preproceedings of NESSIE 1st workshop, Leuven, 2000.
12. Christophe De Cannière, Bart Preneel, *Trivium Specifications*, eSTREAM proposal, available on-line at *http://www.ecrypt.eu.org/stream/p3ciphers/trivium/trivium_p3.pdf*, 2005.
13. Vivien Dubois, Pierre-Alain Fouque, Jacques Stern, Cryptanalysis of SFLASH with Slightly Modified Parameters, Advances in Cryptology, proceedings of EUROCRYPT 2007, Lecture Notes in Computer Science 4515, pp. 327–341, Springer, 2007.
14. Niels Ferguson, Doug Whiting, Bruce Schneier, John Kelsey, Stefan Lucks, Tadayoshi Kohno, *Helix: Fast Encryption and Authentication in a Single Cryptographic Primitive*, proceedings of Fast Software Encryption 10, Lecture Notes in Computer Science 2887, pp. 330–346, Springer, 2003.
15. Scott R. Fluhrer, Itsik Mantin, Adi Shamir, *Weakness in the key scheduling algorithm of RC4*, proceedings of SAC'01, Lecture Notes in Computer Science 2259, pp. 1–24, Springer, 2001.
16. Scott R. Fluhrer, D. A. McGrew, *Statistical Analysis of the Alleged RC4 Stream Cipher*, proceedings of Fast Software Encryption 6, Lecture Notes in Computer Science 1978, pp. 19–30, Springer, 2001.
17. Pierre-Alain Fouque, Louis Granboulan, Jacques Stern, *Differential Cryptanalysis for Multivariate Schemes*, Advances in Cryptology, proceedings of EUROCRYPT 2005, Lecture Notes in Computer Science 3494, pp. 341–353, Springer, 2005.
18. Martin Hell, Thomas Johansson, Willie Meier, *Grain — A Stream Cipher for Constrained Environments*, preproceedings of ECRYPT's Symmetric Key Encryption Workshop, Aarhus, 2005.
19. Alexander L. Grosul, Dan S. Wallach, *A Related-Key Analysis of RC4*, Rice University technical report TR00-358, 2000.
20. Lars R. Knudsen, Willie Meier, Bart Preneel, Vincent Rijmen, S. Verdoolagee, *Analysis Methods for (Alleged) RC4*, Advances in Cryptology, proceedings of ASIACRYPT 1998, Lecture Notes in Computer Science 1514, pp. 327–341, Springer, 1998.
21. Itsik Mantin, Adi Shamir, *A Practical Attack on Broadcast RC4*, proceedings of Fast Software Encryption 7, Lecture Notes in Computer Science 2355, pp. 152–164, Springer, 2001.
22. Itsik Mantin, *Predicting and Distinguishing Attacks on RC4 Keystream Generator*, Advances in Cryptology, proceedings of EUROCRYPT 2005, Lecture Notes in Computer Science 3494, pp. 491–506, Springer, 2005.
23. Itsik Mantin, *A Practical Attack on the Fixed RC4 in the WEP Mode*, Advances in Cryptology, proceedings of ASIACRYPT 2005, Lecture Notes in Computer Science 3788, pp. 395–411, Springer, 2005.
24. Alexander Maximov, Thomas Johansson, Steve Babbage, *An Improved Correlation Attack on A5/1*, proceedings of SAC'04, Lecture Notes in Computer Science 3357, pp. 1–18, Springer, 2004.

25. Ilya Mironov, *(Not So) Random Shuffles of RC4*, Advances in Cryptology, proceedings of CRYPTO 2002, Lecture Notes in Computer Science 2442, pp. 304–319, Springer, 2002.
26. Frédéric Muller, *Differential Attacks against the Helix Stream Cipher*, proceedings of Fast Software Encryption 11, Lecture Notes in Computer Science 3017, pp. 94–108, Springer, 2004.
27. Jacques Patarin, Louis Granboulan, *Asymmetric Cryptography with S Boxes*, proceedings of ICICS'97, Lecture Notes in Computer Science 1334, pp. 369–380, Springer, 1997.
28. Ronald L. Rivest, *RSA security Response to weaknesses in key scheduling algorithm of RC4*, Technical note, RSA Data Security, Inc., 2001. [The structure of RC4 was never published officially, it was leaked in 1994 to the Internet. This note confirms that the leaked code is indeed RC4].
29. Hongjun Wu, Bart Preneel, *Cryptanalysis of ABC v2*, 2006. Available online at *http://www.ecrypt.eu.org/stream/papersdir/2006/029.pdf*.
30. Hongjun Wu, Bart Preneel, *Attacking the IV Setup of Py and Pypy*, eSTREAM website, 2006. Available online at *http://www.ecrypt.eu.org/stream/papersdir/2006/050.pdf*.
31. Hongjun Wu, Bart Preneel, *Differential-Linear Attacks against the Stream Cipher Phelix*, eSTREAM website, 2006. Available online at *http://www.ecrypt.eu.org/stream/papersdir/2006/056.pdf*.
32. Author unknown, *Cryptographic Techniques Specifications TOYOCRYPT-HS1*, 2000.

A Short Descriptions of Several Stream Ciphers

A.1 A Short Description of A5/1

The stream cipher A5/1 [9] is used to protect GSM cellular phone conversations in most European countries and in the USA. It consists of three LFSRs whose outputs are XORed to produce the output of the cipher. Each step, the LFSRs are irregularly clocked. One bit is extracted from any of the LFSRs, and the majority of these bits is computed. A register whose bit agrees with the majority is clocked, while if the bit disagrees with the majority then the register is not clocked. Thus, each step two or all three registers are clocked.

The three registers are denoted by R_1, R_2, and R_3. The lengths of R_1, R_2, and R_3 are 19, 22, and 23 bits, respectively. The registers are updated according to their primitive polynomials, which are summarized in Table 3, along with the location of the bit that enters the clocking mechanism. We use little endian notations, i.e., the output bit of the register is the most significant bit of the register while the least significant bit is the new bit fed back to the register. We denote the j'th bit of the i'th register by $R_i[j]$.

The initialization step takes the 64 bit key K along with a 22-bit frame number (which is publicly known value) and loads them into the registers in a linear manner. All three registers are initialized to zero. All registers are clocked and the output is fed back and XORed with the next key bit (or frame number bit). After loading the 86 bits (64 bits of key plus 22 bits of frame number)

Register Number	Length in bits	Primitive Polynomial	Clock-controling bit (LSB is 0)	Feedback Taps
1	19	$x^{19}+x^5+x^2+x+1$	8	18,17,16,13
2	22	$x^{22}+x+1$	10	20,21
3	23	$x^{23}+x^{15}+x^2+x+1$	10	22,21,20,7

Table 3. The A5/1 Registers Parameters

into the 64 bits of the registers, the state is irregularly clocked 100 times, and the output is discarded. In each round afterwards, the registers are irregularly clocked and the XOR of the most significant bit of each of the registers is used as the output.

A.2 Description of RC4

RC4 accepts keys of variable length. In most applications the key length is in the range of 40 to 256 bits. RC4 treats the key as a byte array $K[\cdot]$ of $l \leq 256$ bytes. The internal state is a byte array of 256 entries $S[\cdot]$ initialized to $S[i] = i$, and then, the key is mixed into the array using the following procedure:

- $j := 0$
- for $i := 0$ to 255 do
  - $j := j + S[i] + K[i \bmod l] \pmod{256}$
  - swap($S[i]$,$S[j]$)
- $i := j := 0$

We note that all addition operations with respect to RC4 are done modulo 256.

After the initialization, RC4 outputs one stream byte each round. Just before the byte stream is generated, the internal state is updated according to the following procedure:

- $i++; \quad i := i \pmod{256}$
- $j := j + S[i] \pmod{256}$
- swap($S[i]$,$S[j]$)
- output $S[S[i] + S[j]]$

B Examples of Other Stream Ciphers Susceptible to Differential Cryptanalysis

B.1 Differentials of Trivium

Trivium [12] is a hardware oriented stream cipher which was selected to the third phase of the eSTREAM project. The cipher is based on three LFSRs with a combined length of 288 bits, where each round the registers affect each other.

We denote the internal state of the cipher by $s_1, \ldots s_{288}$, where the first register is composed of $(s_1, \ldots, s_{93})$, the second is composed of $(s_{94}, \ldots, s_{177})$, and the last one is composed of $(s_{178}, \ldots, s_{288})$. The combined output and update function is as follows:

– Let $t_1 = s_{66} \oplus s_{93}$, let $t_2 = s_{162} \oplus s_{177}$, and let $t_3 = s_{243} \oplus s_{288}$.
– Output $z = t_1 \oplus t_2 \oplus t_3$.
– Update the t_i's according to

$$t_1 = t_1 \oplus s_{91} \cdot s_{92} \oplus s_{171}; \quad t_2 = t_2 \oplus s_{175} \cdot s_{176} \oplus s_{264}; \quad t_3 = t_3 \oplus s_{286} \cdot s_{287} \oplus s_{69};$$

– Update the registers as follows:

$$(s_1, \ldots, s_{93}) = (t_3, s_1, \ldots, s_{92}); \quad (s_{94}, \ldots, s_{177}) = (t_1, s_{94}, \ldots, s_{176});$$

$$(s_{178}, \ldots, s_{288}) = (t_2, s_{178}, \ldots, s_{287});$$

The initialization loads the 80-bit key to $s_1, \ldots, s_{80}$, the 80-bit IV to $s_{94}, \ldots, s_{173}$, and sets all the remaining bits (besides s_{286}, s_{287}, and s_{288} which are set to 1) to 0. Then the cipher is clocked for $4 \cdot 288 = 1152$ cycles without producing output.

Considering a pair of (key, IV) values such that there is a difference only in bit IV_1 give rise to the following differential. The difference in IV_1 becomes a difference in bit s_{94} which does not affect other bits until it arrives to position s_{162} (after 68 rounds). Then it causes a difference in t_2, leading to a difference in bit 178. When the difference arrives to s_{171} it affects s_{94}, starting the propagation of a difference through the second register. The difference continues to propagate until it arrives to position s_{175} then, if s_{176} is zero, then the difference does not propagate into t_2. This happens in the next clocking as well, and thus, the difference after the initial difference arrived (and passed) s_{177} is only in positions s_{102}, s_{178}, and s_{194}. Thus, after 84 clockings, the one bit difference evolves into a three bit difference with probability 2^{-2}.

This difference evolves after 77 more rounds with probability 2^{-2} into the difference $s_{12}, s_{28}, s_{99}, s_{178}, s_{184}, s_{255}$, and s_{271}. After another 71 rounds, the difference becomes $s_1, s_7, s_{30}, s_{36}, s_{52}, s_{81}, s_{99}, s_{100}, s_{108}, s_{126}, s_{178}, s_{194}, s_{244}, s_{250}$ with probability 2^{-6}. After a total of 377 initialization rounds the internal state has a difference of s_{10},s_{16},s_{19},s_{22},s_{25},s_{28},s_{34},s_{43}, s_{46},s_{52},s_{55},s_{61},s_{64},s_{70}, s_{79},s_{82}, s_{91},s_{106},s_{112},s_{115},s_{118},s_{124},s_{130}, s_{133},s_{139},s_{169},s_{175},s_{184},s_{187},s_{190}, s_{214},s_{217},s_{220}, s_{226}, s_{235}, s_{238},s_{241},s_{244},s_{253},s_{256}, s_{259},s_{286} with probability 2^{-78}, in the following round, the probability drops below 2^{-79}, which means that for a given key, there are not enough IV pairs for which at least one follows the differential. Moreover, the differential best differential for the initialization we could find has probability of 2^{-840}, which means that this highly unlikely that any of the keys has an IV pair for which the differential holds.

B.2 LILI-128

A positive example for the strength of differential cryptanalysis is the LILI-128 stream cipher [11] for which the relation between keys was exploited in [2]. LILI-128 has two LFSRs. The first LFSR, $LFSR_c$ of 39 bits, outputs (using a nonlinear function) how many clocks (i.e., 1, 2, 3, or 4 clocks) the second LFSR, $LFSR_d$ of 89 bits, should be clocked. The output of LILI-128 is application of

a nonlinear function on selected bits from the second LFSR. The initialization just loads the key XORed with the IV into the two registers.

We first note that LILI-128 has several sets of differentials that predict with probability 1 that there is no difference in the first few bits of the output stream. For example, there are 2^{79} differentials of the form of a difference in one bit of $LFSR_d$ which predict no difference in the first bit of the input. We note that there are also very good differential when there is a difference only in $LFSR_c$. For example, flipping bit 11 of $LFSR_c$ has no affect on the output for 22 rounds (3 rounds until bit 8 is fed back, and 19 more rounds till the difference in the feedback affects the number of times $LFSR_d$ is clocked).

We now present a simple key recovery attack on LILI-128 based on differential cryptanalysis. We define four internal state differences $\Delta IV^4 = e_{50}$ (i.e., bit 50 of IV which is loaded to bit 11 of $LFSR_d$), $\Delta IV^3 = e_{49}$, ΔIV^2 and ΔIV^1. We start with several pairs of (key, IV), such that $\Delta key = 0$ and $\Delta IV = \Delta IV^4$. If after one clocking of the cipher we obtain no difference in all the pairs, we are assured that the first clocking of $LFSR_d$ was not 4 bits. In case there is a difference we get that the clockings of $LFSR_d$ was of 4, and thus, bits 12 and 20 of $LFSR_d$ are 1.

The same can be repeated with (key, IV) pairs with differences $\Delta key = 0$ and $\Delta IV = \Delta IV^3$. If we found out that $LFSR_d$ was not clocked by 4 bits, then this test check whether $LFSR_d$ was clocked by 3 bits. We continue and identify two bits of $LFSR_c$ and the amount of bits clocked in the first round. We can then repeat the attack (taking into consideration the number of times the register $LFSR_d$ was clocked). We can repeat the attack 20 more times, and retrieve the entire $LFSR_c$, which is then can be easily used to retrieve $LFSR_d$.

The probability that a difference in bit 7 leads to a difference in the output is 1/2, and for 10 tests, the probability that there is no output difference in all of them is 2^{-10}. Thus, about 10 (key, IV) pairs are sufficient to verify whether the number of clockings is 1 (or 2, or 3). Thus, the expected number of (key, IV) pairs for retrieving two bits of $LFSR_c$ is 25.[9] Thus, the entire $LFSR_c$ can be easily retrieved (with probability over 98%) using 500 (key, IV) pairs.

B.3 LILI-II

The stream cipher LILI-128 has been improved in LILI-II [10]. The improvement is based on increasing the length of $LFSR_c$ to 128 bits and the length of $LFSR_d$ to 127 bits. Also, the combining function for $LFSR_d$ has been improved (to depend on 12 bits of $LFSR_d$ rather than on 10). Finally, the key and IV initialization procedure have been changed. $LFSR_c$ is first initialized to be the XOR of the key and IV and $LFSR_d$ is initialized to the XOR of the the key without its first bit and the IV without its last bit. Then, the cipher is used to produce 255 bits of output, and the first 128 bits are loaded into $LFSR_c$, and the remaining 127 bits are loaded into $LFSR_d$. Then, the cipher is again used to

[9] In the rare event that all possible shifts "fail", i.e., non has output difference, we can ask for more pairs until the right shift is determined.

generate 255 bits of output, where as before the first 128 are loaded into $LFSR_c$, and the rest are loaded into $LFSR_d$.

If a pair of (key, IV) pairs, (k_1, IV_1) and (k_2, IV_2), satisfy that $k_1 \oplus k_2 = IV_1 \oplus IV_2 = 1^{128}$ (where 1^{128} is a string of 128 bits of 1), then the two (key, IV) produce the same initial state, and therefore, produce the same 255 bits of stream, which in turn would produce the same 255 bits of stream, which results in the same pseudo random string.

C Differentially Weak Keys and Their Effect on Exhaustive Key Search

The existence of differentials from the key schedule to the output or from the internal state to the output can be used to speed up exhaustive key search. The main idea is to reduce the need to check all keys by checking only one out of two keys/internal states, and discarding the counterpart in case the differential does not hold.

Consider for example A5/1. Two internal states which differ only in bit $R_2[11]$ after the key initialization generate an output difference of 80 bits with probability $0.197 \cdot 0.135 = 0.0266 \approx 1/37.6$. These are the first 80 rounds of the differential suggested in Table 1. As during the trial test of a given internal state we can check whether it has a counterpart that satisfies the differential requirement for the internal state, we can easily check whether the generated stream differs from the given stream in the same places predicted by the differential. When this happens, the attacker can discard another internal state without affecting the success of the attack. Thus, in this specific case, the attacker performs $(1 - 1/37.6)$ trials on average for each possible internal state, the exhaustive search time is reduced by a factor of about 2.6%.

We note that due to the existence of other much shorter differentials, an easy early abort strategy can be used in conjunction with this approach. The resulting combination prevents discarding all guesses that are found out to be wrong after 10 clockings, but as the majority of the wrong internal states are discarded by the early abort by that time, there is no real affect on the time complexity of the attack.

Finally, we note that these ideas can be used also in the case of guess-and-determine attacks, where the attacker guesses parts of key/internal state in order to retrieve the remaining unknown information. Returning to the example of A5/1, the common approach is to guess the two shorter registers R_1 and R_2, and construct the possible values for R_3. Again, using differentials, it is possible to easily identify wrong guesses.

We conclude that if an attacker has a differential for a stream cipher with probability p, she can speed up the time of exhaustive key search by a similar factor. Multiple differentials can also be used, but one must be careful to take into consideration the dependence between them.