Synchronization Fault Cryptanalysis for Breaking A5/1 [★]

Marcin Gomułkiewicz¹, Mirosław Kutylowski¹,
Heinrich Theodor Vierhaus², and Paweł Wlaź³

¹ Wrocław University of Technology
{gomulkie, mirekk}@im.pwr.wroc.pl

² Brandenburg University of Technology, Cottbus
htv@informatik.tu-cottbus.de

³ Lublin University of Technology
p.wlaz@pollub.pl

Abstract. A5/1 pseudo-random bit generator, known from GSM networks, potentially might be used for different purposes, such as secret hiding during cryptographic hardware testing, stream encryption in piconets and others. The main advantages of A5/1 are low cost and a fixed output ratio.

We show that a hardware implementation of A5/1 and similar constructions must be quite careful. It faces a danger of a new kind of attack, which significantly reduces possible key space, allowing full recovery of A5/1 internal registers' content. We use "fault analysis" strategy: we disturb the A5/1 encrypting device (namely, clocking of the LFSR registers) so it produces an incorrect keystream, and through error analysis we deduce the state of the internal registers. If a secret material is used to initialize the generator, like in GSM, this may enable recovering the secret. The attack is based on unique properties of the clocking scheme used by A5/1, which is the basic security component of this construction.

The computations that have to be performed in our attack are about 100 times faster than in the cases of the previous fault-less cryptanalysis methods.

Keywords: fault cryptanalysis, A5/1, GSM, LFSR

1 Introduction

In this paper we consider the security of A5/1 algorithm. The algorithm is a pseudo-random bit generator used e.g. by GSM networks for keystream generation. It is extremely simple in design: it consists of three LFSR registers, which output is XOR-ed. The most important feature of this algorithm is its' LFSRs' clocking mechanism.

The cryptographic strength of A5/1 algorithm comes from a non-linear clocking rule. All LFSR-based designs, which do not have a non-linear component, can be broken easily with simple linear algebra. A lot of attention was devoted to the question how to combine LFSRs – which are easy to build and fast – in order to get a cryptographically strong generator. A5/1 is an extremely simple solution to this question, another

[★] Partially supported by DAAD project PROKRYPTO and KBN project no. 4 T11D 005 24. Accepted for 4th International Workshop on Efficient and Experimental Algorithms (WEA'05), to be published in LNCS, copyright Springer Verlag

one is a shrinking generator [8]. A5/1 has an advantage over the shrinking generator, it produces an output bit at each step, while the shrinking generator requires a buffer for collecting the bits before outputting them.

Potential Applications A5/1 is at the moment probably the most widely used stream cipher. It has been designed so that its' cheap and efficient hardware implementation is possible, so the idea to use A5/1 for purposes other than those it was originally created for might be tempting. This is additionally motivated by the fact that using existing and tested ciphers instead of designing new ones is advised from a security's viewpoint. Such tasks could include for example purposes related to secure testing of integrated circuits that store sensitive data. Every chip that is currently produced contains an appropriate test circuitry, used generally right after production and before final packaging. However, sometimes integrated circuits have to be checked thoroughly and frequently – this occurs for many crucial security components implemented in hardware, a good example are secure signature creation devices for which a hardware fault must be detected as soon as possible. If such a chip contains secret data, possibility to perform checks by an unauthorized personnel poses a security threat, especially in presence of some modern side-channel cryptanalysis techniques, such as timing attacks, DPA or fault analysis. To avoid problems, the test output should be scrambled. Due to technical reasons, algorithms such as A5/1 are very well suited for becoming scramblers – the data from the testing circuit comes through a kind of serial interface and can be combined with the random bits of the scrambler before sending them out.

A5/1 Security Several attacks on encryption algorithms A5/1 and A5/2 (a weaker algorithm that may be used instead of A5/1 by GSM phones) have been proposed, see [4, 5, 10]; the most recent work [2] is based on some fundamental cryptographic flaws found in GSM protocol, and describes a way to recover A5/2 secret session key using only a couple of milliseconds of encrypted communication within a second on a home computer. Authors suggest that although similar method can be used against stronger A5/1, that attack would be rather cumbersome in practice: one of possible setups is that if one had 8 seconds of encrypted communication and 5000 PCs had completed preprocessing in 1 year's time filling up 176 200GB disks, another 1000 PCs could perform an attack in real-time. However, while such an effort is rather unrealistic for individuals, it is well within the range of a middle sized company or university, not to mention wealthy and highly motivated attackers.

New Result The main goal of this paper is to inspect threats of A5/1 device security that come from the design of the shift registers' clocking. We show that disturbing a clocking sequence yields an alternative, quite efficient attack on A5/1. For instance, it allows to reconstruct the contents of A5/1 internal registers with a moderate computational effort in a certain fault model. This attack may be seen as a special case of attacks on A5/1 predicted in [4], however as far as we know it is the only attack that use fault-analysis strategy, and at the same time it is probably the most efficient one. The attack is based on stopping a LFSR register from clocking at a given moment. Our paper shows that the countermeasures against hardware faults of this kind are necessary.

Paper Organization We start with an overview of A5/1 stream cipher and briefly describe the idea of fault analysis. Then we introduce the idea of resynchronization, and describe the way it can be used to guess some part of A5/1 internal state, and thus mount an effective attack; we discuss some difficulties and their possible solutions. At the end we discuss technical feasibility of attacks on physical devices and necessary precautions.

2 Preliminaries

2.1 The A5/1 Stream Cipher

The A5/1 algorithm is known for several years: although never officially published, its construction has been found via reverse-engineering [7]. The algorithm, as used by GSM, has three linear shift registers R1, R2 and R3 with 19, 22 and 23 bit cells respectively. The session key has 64 bits. The data transferred is divided into so-called *frames*, each 228-bit long: 114 bits of incoming data and 114 bits of outgoing data. The frame number is 22 bit long. For each frame the keystream generation is divided into three phases. In the first phase, registers are zeroed and then clocked regularly 64 times, while their leftmost (see Fig. 1) bits are XOR-ed with consecutive bits of the session key. Similarly in another 22 similar steps 22 bits of the *frame number* are inserted (actually, the frame number is a fixed bit permutation of *TDMA frame number*, cycled in 2^{22} long periods.) In the second phase registers do not clock regularly – which of the registers clock depend on bits on positions 8, 10, 10 in (respectively) R1, R2, R3, called *clocking windows*. The registers clock according to a majority rule: in the next step the only registers (2 or 3) that clock are the ones which “decision” bits from clocking windows (black cells on Fig. 1) are equal to the majority of these bits (majority of the three bits, a, b, c can be expressed as $ab + ac + bc$.) The second phase lasts for 100 steps and its’ output is discarded. Finally, during the third phase the registers work as in the second phase, but their output bits are XOR-ed; the results are the key bits that are XOR-ed with the plaintext. This phase lasts for 228 steps and then the frame is over. For the next frame, the whole procedure restarts with a new frame number.

2.2 Fault Cryptanalysis

Fault analysis is a fairly recent cryptanalysis tool; it was introduced by Boneh *et. al* [6]. The idea is to cause distortions during operation of a cryptographic device so that it produces a faulty output; comparing faulty and correct output can yield significant information about the secrets contained within the device. Among others, a spectacular attacks of this kind have been proposed against improperly implemented RSA scheme [13], and against symmetric algorithms [3, 9].

3 Resynchronization Attack

We consider an attack in which the adversary holds the encryption device with a secret key inside. The adversary’s goal is to find the state of the linear shift registers at

Fig. 1. A5/1 registers. Black bits are the majority bits.

some moment of the computation. Having this state one can apply known computational methods [5] to trace back the states of the shift registers to the moment when the secret key is involved in the computation and derive the secret key.

3.1 Clocking Fault

We assume that an adversary, say Bob, holds an A5/1 encryption device and can use it freely. Since the input and output data are known for Bob, he gets the pseudorandom sequence generated by the device. Bob can insert faults into device's operation. For the sake of simplicity in this paper we discuss only one scenario: we assume that in an arbitrary moment Bob can block the shift that would be performed by one of the shift registers (say R1.) Hence, if R1 does not clock at this moment, this has no effect on the state of shift registers and therefore has no effect on the output. In the opposite case the computation gets disturbed: in the next step majority function is computed from different bits and changes in clocking may propagate. Typically, the shift registers clock in a different way and, consequently, different pseudorandom bits are generated.

Obviously, if after blocking R1 for one step no change in the output sequence occurs, then with high probability R1 does not clock at this moment in a correct computation. So by checking whether the output changes Bob gets some information on the bits that determine clocking. By performing many such trials one may hope to gather enough information to reconstruct the states of the shift registers. The problem is that putting these information together seems to be hard as the number of possible cases grows rapidly. Our goal is to find a method that allows making conclusions about the contents of the shift registers possibly less frequently, but when succeeded, only a small number of candidates for relatively large blocks of bits exist (strategy similar to the one presented in [4].)

3.2 Resynchronization

Assume that at step t' clocking of a register was prohibited. We say that shift registers are *synchronized* at step t of a computation, if during steps t' through t each of the shift registers clocks exactly the same number of times as in a fault free computation with the same initial state. Of course, clocking patterns need not to be the same, only the total number of moves for each shift register counts.

A key observation is that after a clocking fault **resynchronization** is possible and occurs in quite specific situations. For instance, consider the following configuration:

$$\begin{array}{ll}
 \text{R1:} & \boxed{1}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\dots\boxed{0}\boxed{0}\boxed{0} \quad \leftarrow n-1 \text{ zeroes} \\
 \text{R2:} & \boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\dots\boxed{0}\boxed{0}\boxed{0} \quad \leftarrow n \text{ zeroes} \\
 \text{R3:} & \boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\dots\boxed{0}\boxed{0}\boxed{0} \quad \leftarrow n \text{ zeroes}
 \end{array}$$

Assume that a fault occurs at the first step in R1. It is easy to see that after n steps the registers get resynchronized. Of course, the above example (due to the construction of R1) is valid only for $n < 10$.

Now let us introduce some notation. The block of bits starting in the clocking windows that provides resynchronization immediately after k steps is called **resynchronization pattern** or **k -resynchronization pattern**, **RSP k** for short; each resynchronization pattern consist of three blocks of bits, one per shift register (R1, R2 and R3.) For instance, the example above considered for $n = 2$ yields an RSP2 of the form (10,00,00).

Any pattern can be checked offline for resynchronization property, so all resynchronization patterns of small length can be found via an exhaustive search. For instance, there are only two RSP3: (011,111,111) and (100,000,000), which in fact we have already seen. The things become more interesting for RSP4. There are the following resynchronization patterns of length 4:

$$\begin{array}{ll}
 (011, 1101, 110), & (011, 110, 1101), \quad (100, 0010, 001), \quad (100, 001, 0010) \\
 (011, 1101, 1101), & (100, 0010, 0010), \\
 (0111, 1111, 1111), & (1000, 0000, 0000).
 \end{array}$$

Essentially, we have here only three different cases, we can obtain each other pattern by replacing the contents of R2 and R3, or by flipping each bit.

The things are really exciting for larger lengths. Through an exhaustive search we have identified all resynchronization patters of lengths 5 through 9. The number of these patters equals: 30 for RSP5, 112 for RSP6, 480 for RSP7, 2068 for RSP8, and 8992 for RSP9. The number of the resynchronization patterns fits very well cryptanalysis needs: it is not too small (which would make finding resynchronization difficult) and not too big (large number of possible patterns would make hard to guess the pattern that has actually occurred.)

3.3 Outline of Cryptanalysis

The attack is based on the following observations:

- if we observe resynchronization after n steps, then it is plausible to assume that one of patterns RSP k occurred, for $k = n$ or k slightly bigger than n ,

- the clocking fault does not change the contents of the shift registers, only their timing is affected; as a consequence, before a resynchronization occurs, the output is built from the same bits, but the moments when particular bits are used by the output XOR-gate differ. The important thing is that these changes can be derived from resynchronization pattern.

The attack consists of the following phases:

Phase 1: run the device without faults and with faults injected at different moments; look for a situation where resynchronization occurs after some (about 5 to 9) steps and collect the corresponding output streams. For a given resynchronization length, there is a number of possible RSPs, typically couple thousands; we execute the remaining phases for each of these patterns.

Phase 2: compute clocking pattern that follows from the resynchronization pattern assumed – in this way derive clocking in both faulty and non-faulty execution for the period between the moment of injecting the fault and the moment of resynchronization. Given the clocking pattern, construct a system of linear equations describing the output in this period. For this purpose take a number of variables describing some number of rightmost bits in each of the shift registers. Then express the output bits in terms of these variables (each expression is a (mod 2) sum of three variables.) This is possible, since the clocking pattern is known. Note that we have two sets of equations – one corresponding to a faulty execution and one for the error-free one; moreover, the faulty computation gives us **different** equations with **the same** variables. For example, for RSP7 there are typically 13 equations of 18 variables. Solve this systems of linear equations. If the system has no solution then stop considering this resynchronization pattern. Otherwise, we can express a number of bits on the right side of shift registers at the moment when a fault occurs by but a few of these bits.

Phase 3: In this phase we have many bits of registers known (about 44 bits in the case of RSP9; about 21 in the case of RSP5.) To find the rest of them we will gradually guess the values of unknown bits needed for the clocking mechanism, make one move of the system and construct a linear equation with current rightmost bits of registers and the output bit. All equations are expressed in terms of bits of registers at the moment when clocking fault is caused.

Some of the guesses will contradict the system of linear equations, other will lead to full rank linear system with 64 unknowns – the solution should be then verified by comparing to the original keystream.

3.4 Some Details of the Attack

Resynchronization Probability After injecting a clocking fault resynchronization occurs after 5 to 9 steps with probability of about 1.5% (0.0148462...). If we may afford to look for resynchronization occurring less frequently (meaning more experiments with the device) we should rather concentrate on longer patterns, say 8 or 9 steps long.

Linear Equations for Phase 2 First let us consider a toy example of an RSP3. Consider one of RSP3s mentioned before – (011, 111, 111). Consider the state of registers before injecting a fault and denote the bits stored by R1, R2 and R3 by, respectively, $a_0, \dots, a_{18}, b_0, \dots, b_{21}$ and $c_0, \dots, c_{22}$ (see Fig. 1.) Let the output observed in the fault free computation be $x_1, x_2, x_3, \dots$ and in the faulty computation – $y_1, y_2, y_3, \dots$. Since we assume resynchronization after 3 steps, $x_3 = y_3$. By inspecting the clocking sequence in both computations one can derive the following equalities for the output values:

$$\begin{cases} a_{17} + b_{20} + c_{21} = x_1 \\ a_{16} + b_{19} + c_{20} = x_2 \\ a_{16} + b_{18} + c_{19} = x_3 \end{cases} \quad \begin{cases} a_{18} + b_{20} + c_{21} = y_1 \\ a_{17} + b_{19} + c_{20} = y_2 \\ a_{16} + b_{18} + c_{19} = y_3 \end{cases}$$

Since $x_3 = y_3$, this yields a system of 5 equations, which are in this case independent. So we may express 5 variables occurring in this system through expressions with 4 remaining variables. For instance:

$$\begin{cases} a_{16} = b_{20} + c_{20} + x_1 + x_2 + y_2 \\ a_{17} = b_{20} + c_{21} + x_1 \\ a_{18} = b_{20} + c_{21} + y_1 \\ b_{18} = b_{20} + c_{19} + c_{20} + x_1 + x_2 + x_3 + y_2 \\ b_{19} = b_{20} + x_1 + y_2 \end{cases}$$

In this example no values of x_i, y_i can contradict the system, which could only happen, if the rank was lower than the number of equations. However, for instance for RSP5 (100, 00110, 0011) the system considered consists of 9 equations and 13 variables, rank of its left side is 7 and it has a solution if and only if

$$\begin{cases} x_1 + x_2 + y_1 + y_2 = 0 \\ x_1 + x_3 + y_1 + y_3 = 0 \end{cases}$$

So this RSP5 will be excluded for 75% cases of the values of x_i, y_i .

In general, systems of equations defined in Phase 3 are underdefined, but the rank of the system matrix is quite high; more thorough statistics can be found in the Appendix A.1 [12]. If we consider only resynchronization after 5, 6, 7, 8 and 9 steps, then there are 11682 RSPs (with different lengths.) We may also consider *fixed length* RSPs by filling in additional bits so that all RSPs have the same length. Then we get 1992622 RSPs of length 27. The average case is that resynchronization occurs after 6.70807 steps, (the variable length) RSP contains 16.421 bits, the system of linear equations contains 17.1326 variables and its rank is 10.5344. So once we choose an RSP, on average we express 33.5536 bits of the shift registers by guessing some 6.5982 bits. Since the difference between the number of equations and the rank is 1.88174, for an average system, in about 73% cases RSP guess is not consistent with the bits x_i, y_i , so it can be quickly filtered out.

Uncertainty About Resynchronization Pattern Length Unfortunately, we cannot be sure about the exact number of steps before resynchronization occurs: even if the output bits from a given step onwards are the same, it may happen that resynchronization

occurs a few steps later, and the same keystream bits before resynchronization came out by accident. If resynchronization occurs after step t , then with probability approximately $\frac{1}{2}$ the outputs are the same already after step $t - 1$ (we would have probability exactly $\frac{1}{2}$ after replacing each shift register by an independent random generator.) However, finding exact probability that output of an RSP n will synchronize after step $n - 1$, $n - 2, \dots$ is nontrivial due to some subtleties of resynchronization. For instance consider an RSP4 (011, 1101, 1101). Denote the contents of the registers exactly as in the case of an RSP3. Then the output sequence starting from the moment of injecting a fault equals

$$a_{18} + b_{20} + c_{21}, \quad a_{18} + b_{19} + c_{20}, \quad a_{17} + b_{18} + c_{19}, \quad a_{16} + b_{17} + c_{18},$$

while the correct system outputs

$$a_{17} + b_{20} + c_{21}, \quad a_{17} + b_{19} + c_{20}, \quad a_{16} + b_{18} + c_{19}, \quad a_{16} + b_{17} + c_{18}.$$

So the output for this RSP4 will synchronize after step 4 if and only if

$$a_{16} + b_{18} + c_{19} \neq a_{17} + b_{18} + c_{19}$$

or just $a_{16} \neq a_{17}$ for which probability is about $\frac{1}{2}$. We observe the same output already after step 3 if

$$\begin{cases} a_{16} + b_{18} + c_{19} = a_{17} + b_{18} + c_{19} \\ a_{17} + b_{19} + c_{20} \neq a_{18} + b_{19} + c_{20}, \end{cases}$$

that is, if $a_{16} = a_{17}, a_{17} \neq a_{18}$. This occurs with probability roughly $\frac{1}{4}$. For observing output resynchronization after step 2 conditions are

$$\begin{cases} a_{16} + b_{18} + c_{19} = a_{17} + b_{18} + c_{19} \\ a_{17} + b_{19} + c_{20} = a_{18} + b_{19} + c_{20} \\ a_{17} + b_{20} + c_{21} \neq a_{18} + b_{20} + c_{21}, \end{cases}$$

that is, if $a_{16} = a_{17}, a_{17} = a_{18}, a_{17} \neq a_{18}$, which can never happen. So the remaining probability is allocated to the event that the outputs get the same already after step 1 after the moment of fault injection.

Exactly the same effects can be observed for the following RSP4: (100, 0010, 0010), (011, 1101, 110), (011, 110, 1101), (100, 0010, 001), (100, 001, 0010). For two other RSP4, namely (0111, 1111, 1111) and (1000, 0000, 0000), the probabilities of resynchronization of the output after step 4, 3, 2 and 1 are, respectively, $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$ and $\frac{1}{8}$. Thus, if we consider fixed length RSP4s (all consisting of 12 bits), then we get two RSPs for which the outputs can re-synchronize after step 2, and 20 RSPs for which it is impossible. So probability that output of random registers with RSP4 resynchronizes after 2nd step is $\frac{1}{8} \cdot \frac{1}{11} = 0.011(36)$. Experimental results are very close to these values.

4 Cryptanalysis Implementation

4.1 Some Notes on Implementation

Phase 1 We have to guess which RSP really occurred: we know we should not rely on outputs' similarities only. For that reason, we should rather exclude situations where

Table 1. Which RSP is responsible for resynchronization of outputs

observed		actual RSP length								
RSP	prob.	n	$n+1$	$n+2$	$n+3$	$n+4$	$n+5$	$n+6$	$> n+6$	
$n=1$	1.93%	0%	80.84%	5.05%	6.63%	3.42%	1.79%	... 2.27%	...	
$n=2$	1.72%	91.29%	5.70%	0.36%	1.18%		1.47%			
$n=3$	0.39%	49.41%	34.02%	4.05%	5.16%	3.13%	1.81%	... 2.42%	...	
$n=4$	0.43%	62.07%	23.64%	3.16%	4.72%	2.72%	1.58%	... 2.11%	...	
$n=5$	0.34%	60.61%	23.33%	3.76%	5.11%	3.03%	1.78%	... 2.38%	...	
$n=6$	0.28%	56.47%	25.53%	4.32%	5.63%	3.38%	2.00%	1.16%	1.51%	
$n=7$	0.25%	56.84%	24.91%	4.43%	5.68%	3.39%	2.01%	1.18%	1.56%	
$n=8$	0.22%	56.06%	25.12%	4.62%	5.81%	3.50%	2.07%	1.20%	1.62%	
$n=9$	0.20%	55.60%	25.21%	4.77%	5.85%	3.55%	2.10%	1.22%	1.70%	

outputs seem to synchronize after 9 steps – investigating Table 1 shows that the chances that some of RSP_k , $k \in \{5, 6, 7, 8, 9\}$ occurred are rather slim (about 55%.) Therefore we shall concentrate on outputs that seem to synchronize after 5 to 8 steps. Of course if we have observed resynchronization after, say, 7 steps, we need not to consider RSP5s and RSP6s. Each RSP guess gives us possible values of some bits within the registers – typically 12 to 27. We may think of these as of trivial equations (*variable = value.*) Obviously, these equations are independent.

Phase 2 This phase basically sieves the possible guesses from Phase 1 out, and adds some equations given by the guess. Phase 2 gives significant boost to our calculations: not only we are able to quickly falsify about 73% of the wrong guesses from the previous phase, but also, since we are considering two different outputs at the same time, given by two different clockings, we get twice as much equations. Some of them are unfortunately dependent; luckily, the rank of an appropriate matrix is still quite high. As it has been pointed before, because of the moderate number of RSPs it is possible to precompute the equations given by every single RSP – this significantly speeds up the search and also makes the whole search easier to implement on parallel computers.

Phase 3 In Phase 3 we try to fill the bits on the left side of the pattern guessed in Phase 1 (i.e. we add trivial equations describing some bits close to the left end of the registers) and afterwards we construct more equations, just as in Phase 2, concerning the bits that are at the rightmost positions when the newly guessed bits enter the clocking window. Of course now only one equation for each step is given. Once non-contradictory set of 64 independent equations is found, we solve it and check if the solution is consistent with output given by an original device. A rough estimation of the number of cases shows that about $2^{34.23}$ systems of linear equations with 64 unknowns need to be solved in an average case. More details regarding complexity of our approach can be found in Appendix A.2, see [12].

It is also possible to perform Phase 3 in a different, simplified way: guess gradually bits in interesting positions and check it against the known output. This approach can

be used when for some reason one does not want to perform lots of matrix operations necessary to solve linear equations' system.

Let us remark that in this attack, compared to the previous ones [4, 10], the number of cases to be considered has been reduced about 100 times. This is possible since we take for cryptanalysis only very particular sequences. On the other hand, their frequency is close to 1.5%. This is advantageous, because generally cost of running / simulating A5/1 is negligible comparing to cryptanalysis' cost.

4.2 Test Implementation

We have tested our cryptanalysis procedure on a home computer, namely AMD Athlon XP 1800+ based PC running Debian GNU/Linux.

In the precomputational phase we have found and stored all RSPs and corresponding systems of equations together with their solutions and conditions under which the system has solutions. The space required for the data is less than 1MB and the computations were performed on the same home PC.

Our proof-of-concept implementation finds the whole 64-bit-long contents of the three registers given two outputs that resynchronize after 5 to 8 steps. This first implementation is not (yet) optimized for code efficiency. We believe that a lot of fine tuning in algorithm design is possible to speed it up considerably. In fact, many directions seem to be promising; the problem is to choose the most efficient tricks.

When assumed RSP is of length 5, after Phases 1 and 2 we are left with about one thousand (partially filled) candidates, checking each of them takes few minutes on average. For (assumed) RSP of length 9 Phases 1 and 2 yield about one million candidates, but hundreds of them may be checked in a second.

It is also worth noting that the computation very easily scales up to a larger number of CPUs: different processors can simply check different candidates in parallel.

5 Technical Feasibility

One may try to implement fault attack using test capabilities of integrated circuits (IC). It seems to be a plausible idea, since testing should enable to run single components, that is, disable the other components. However, while a "raw" chip or die typically has extra test pins accessible via needles, they are not included in the packaging of the IC and cannot be accessed from the circuit board. The package of an IC must be opened for such purpose, which requires a highly sophisticated semiconductor test lab's equipment.

Manipulation of functional chips based on intrusive technologies is a different subject. Internal structures of an IC can be influenced by needles mechanically and electrically, by electron and ion beams electrically, and optically by lasers. In today's deep sub-micron structures, using needles is no longer feasible except for specific pad areas. However, a focused charged electron or ion beam can very well influence specific wires of an IC or even a single memory cell. Re-scattered electrons from a charged lines can be observed on a scanning electron microscope and can exhibit whether such a line is at "low" or at "high." Again the advances in semiconductor technology limit such mechanisms. With up to seven layers of metal interconnects stacked upon each

other in state-of-the-art digital CMOS technologies [1], the higher levels will shield the lower level effectively. Removal of higher layers and successive active operation of lower levels for observation is close-to impossible. Furthermore, shielding of signal lines from observation is quite well possible by physical IC design. Because of that, if the chip have been designed without some special precautions, our attack is of little or no concern.

Trapdoors With currently available IC technology possibilities of external manipulations of clocking seem to be limited. However, the manufacturers may introduce some trapdoors through a certain IC design – one can put a certain line on a higher level in order to make it vulnerable to intrusive technologies. In such a scenario it would be necessary to possess an advanced test lab to perform the synchronization attack. This situation might be comfortable for security agencies – the number of manufacturers with the access to sophisticated semiconductor technology is small and they can be relatively easily monitored. Controlling all / majority of parties with appropriate technological tools would successfully limit the potential attackers, and at the same time allow the secret services to use the synchronization attack.

6 Conclusions

We have shown that if one can stop clocking in a chosen register for a single step, and run A5/1 for the same initial contents, then with a reasonable number of experiments one can find a case in which the contents of the registers can be fully reconstructed.

So the main point is that during security evaluation of hardware devices implementing A5/1 one has to prove infeasibility of resynchronization attack against the A5/1 component with physical equipment available today.

Further Attacks One can consider different scenarios in which essentially the same idea can be applied. Consider the case in which we can change a single bit at a random position in one of the registers. Attacks of this kind should be considered very carefully, since such faults might be likely in appropriate physical conditions. The details of retrieving the contents of the shift registers are different in this case (perhaps more confusing) but the general attack idea based on resynchronization remains the same.

If one could assume the possibility of changing the state of consecutive bits of chosen register to 1, then another attack can be applied. In this case no knowledge of correct sequence is needed, see [11].

Source of Weaknesses The problem with the A5/1 algorithm is that the keystream is generated from quite weak components, even if their outputs are combined in a clever way. So a distortion of a single component has limited consequences and the effect may cancel out in certain circumstances – which is exactly the opposite to an avalanche effect.

It turns out that taking the bits for clocking in the middle (which is reasonable from the point of view of the previous attacks) becomes advantageous for synchronization attack. Namely, the following design aspects help to mount the attack:

- for a relatively long resynchronization period no bit of the resynchronization pattern reaches the output position, while
- all the bits of resynchronization pattern are already in the registers when the fault occurs.

However, the main feature is that one can consider separately the bits on each side of the resynchronization pattern. This reduces complexity of the attack against brute force by an order of magnitude.

Countermeasures The change of the registers' length does not reduce the gain obtained in our attack, it only influences the number of the remaining bits that have to be found.

Another way to defend against synchronization attacks would be redesigning the way in which the shift registers cooperate. Certainly, considering more than one clocking window in each register would make the attack much harder – the resynchronization pattern would consist of many blocks of bits. Consequently, we would have to choose shorter patterns and in this way gain less information on the remaining bits. However, we cannot be sure that such a modification does not bring new dangers. Additionally, one can use feedback bits from all three registers for each of the shift registers. In such a case resynchronization would be unlikely.

References

1. A. Allan, D. Edenfield, W. H. Joyner, A. B. Kahng, M. Roger, Y. Zorian, *2001 Technology Roadmap for Semiconductors*, IEEE Computers, Jan. 2002, pp. 42–53
2. Elad Barkan, Eli Biham, Nathan Keller, *Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication*, Crypto'2003, LNCS 2729, Springer, 2003, pp. 600–616
3. Eli Biham, Adi Shamir, *Differential Fault Analysis of Secret Key Cryptosystems*, Crypto'97, LNCS 1294, Springer, 1997, pp. 513–525
4. Eli Biham, Orr Dunkelman, *Cryptanalysis of the A5/I GSM stream cipher*, INDOCRYPT'2000, LNCS 1977, Springer, 2000, pp. 43–51
5. Alex Biryukov, Adi Shamir, David Wagner, *Real time cryptanalysis of A5/I on a PC*, FSE'2000, LNCS 1978, Springer, 2001, pp. 1–18
6. Dan Boneh, Richard A. DeMillo, Richard J. Lipton, *On the importance of checking cryptographic protocols for faults*, Eurocrypt'97, LNCS 1233, Springer 1997, pp. 37–51
7. Marc Briceno, Ian Goldberg, David Wagner, *A pedagogical implementation of A5/I and A5/2 "voice privacy" encryption algorithms*, <http://cryptome.org/gsm-a512.htm>, 1999
8. Don Coppersmith, Hugo Krawczyk, Yishay Mansour, *The Shrinking Generator*, Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology, LNCS 773, Springer, 1993, pp. 22–39
9. Pierre Dusart, Gilles Letourneux, Olivier Vivolo, *Differential Fault Analysis on A.E.S.*, ACNS'2003, LNCS 2846 Springer 2003, pp. 293–306
10. Jovan Dj. Golić, *Cryptanalysis of Alleged A5 Stream Cipher*, Eurocrypt'97, LNCS 1233, Springer, 1997, pp. 239–255
11. Marcin Gomułkiewicz, Mirosław Kutylowski, Paweł Wlaź, *Fault Cryptanalysis for Breaking A5/I*, to appear in Tatra Mountains Mathematical Publications

12. Marcin Gomułkiewicz, Mirosław Kutylowski, Heinrich Theodor Vierhaus, Paweł Właż, *Synchronization Fault Cryptanalysis for Breaking A5/I. Appendix*, <http://kutyłowski.im.pwr.wroc.pl/a5/> or <http://mat.pol.lublin.pl/a5/>, 2005
13. Marc Joyce, Arjen K. Lenstra, Jean-Jacques Quisquater, *Chinese Remaindering Based Cryptosystems in the Presence of Faults*, J. of Cryptology 12(4), 1999, pp. 241–245