Tatra Mt. Math. Publ. **33** (2006), 169–177

Tatra Mountains
Mathematical Publications

FAULT CRYPTANALYSIS FOR BREAKING A5/1

Marcin Gomułkiewicz — Mirosław Kutyłowski — Paweł Wlaź

ABSTRACT. We present a fault attack on the A5/1 stream cipher. It is performed by imposing error bits on shift registers. The attack works very efficiently: it needs only a small amount of generated bits and does not require the correct sequence for our comparison (which differentiates our attack from the other known fault attacks.)

In this way we demonstrate a weakness of ciphers based on LFSR's and clocking scheme, such as used in A5/1, implemented in hardware. This is a serious warning showing that a hardware implementation of such a simple scheme as A5/1 does not necessarily improve security—even if the algorithm looks very well suited for a hardware implementation.

1. A5/1 cipher

The A5/1 algorithm is used for encryption of GSM traffic between cellular phones and base stations. The hardware consists of three linear shift registers R1, R2 and R3 with 19, 22 and 23 cells respectively. The session key has 64 bits. The algorithm execution consists of *frames*. Each frame is divided into three phases. During the first phase the registers clock regularly 64 times and are fed one by one with all bits of the session key. Then there are another 22 steps executed similarly, but with 22 bits of so called *frame number* instead of the key bits. The frame number is publicly known, but each frame has a different number (the output of the registers have not been used yet.) During the second phase the registers do not clock regularly, but according to a majority rule. For this purpose the bits on positions, 8, 10, 10, of R1, R2, R2 (respectively) are examined (marked black on Figure 1). Let the value b be in majority on these positions (that is, it occurs at least twice.) Then only those shift registers clock, where the position examined contains value b. The second phase lasts for 100 steps, and the output is discarded. During the third phase the registers clock as in the second phase, but their output bits are XOR-ed and form a pseudo-

2000 Mathematics Subject Classification: 94A60.
Keywords: A5/1 cipher, fault cryptanalysis.

random sequence used for stream encryption of the communication. This phase lasts for 228 steps, and then the frame is over.

So we can think of each frame as of the period divided into 414 discrete steps numbered from 0 to 413; we denote them by $t_0, t_1, \dots t_{413}$. Observe that the pseudo-random sequence used for encryption is obtained during the steps $t_{186}, t_{187}, \dots t_{413}$, and that all three registers' state at steps $t_0, t_1, \dots t_{63}$ are independent of the frame (provided the same session key is used).

The A5/1 algorithm is appealing because of its simplicity, speed and ease of implementation. It escapes quite well from the weaknesses of other schemes based on LFSR's due to its majority clocking rule, and therefore it seems to be a good candidate for encrypting communication by simple embedded systems.

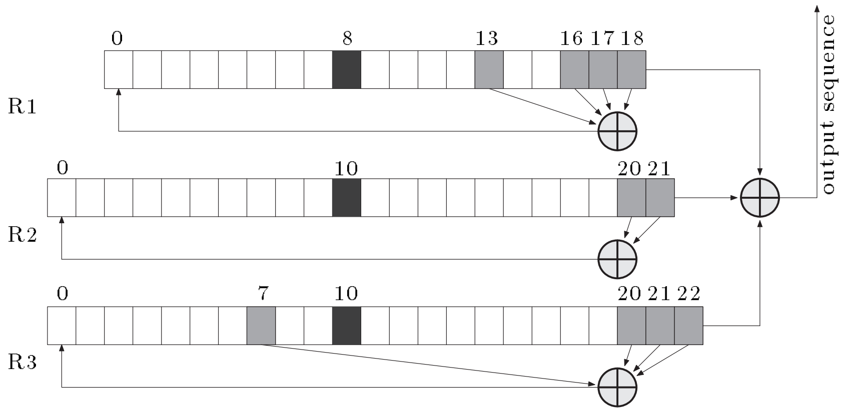

FIGURE 1. A5/1 registers, black bits are the bits examined by the majority rule.

Let us remark that the Bluetooth protocol uses a much weaker pseudo-random bit generator.

Our goal is to show that the hardware implementation of A5/1, if not protected against injecting faults, is insecure: it cannot provide high security level at a low price, despite initial hopes.

2. The first fault model

In our fault model we assume that an attacker can change the state of a chosen register to the all-one state. This assumption is quite likely to be fulfilled for the reasons of a physical nature which will not be discussed here. We only mention that similar assumptions were made in some other work in fault cryptanalysis field [5, 6]. In Section 3 we show how to perform cryptanalysis under much weaker assumptions. We will also assume that this change can be made in the moment when the system is about to clock before calculating the first output bit, that is just before t_{186}.

Although in general the goal of the attacker on A5/1 is the 64 bit session key, we will be satisfied with the state of R1, R2 and R3 before the step t_{64}. Having this we can obtain the pseudo-random sequence of each frame, or by solving the system of 64 independent linear equations-we can find the session key (the independence of this system may be checked by direct calculations).

2.1 Guessing the value of R1 just before t_{186}

Suppose we have changed the state of R2 and R3 into all-one states in the moment t_{186}. Now, observing the output sequence of length n (where $n \leqslant 228$) we can make some conclusions about the bits contained in R1. The longer the sequence is observed, the bigger the chances to guess the exact values of the state of R1 are. Since the size of the register R1 is small, we can precompute a table mapping the sequence observed into states of R1 in t_{186}, provided that n is not too large. Such a mapping is of course multi-valued.

Let $h(s)$ denote the set of possible states of R1, so that when R2 and R3 are set to all-one state, then the system produces the sequence with prefix s. Let h_n denote the function h with a domain restricted to reachable prefixes of the length n. Exact calculations show that only for 1664 prefixes $\operatorname{card}(h_{16}(s)) = 1$, the maximum value of $\operatorname{card}(h_{16}(s))$ is as big as $58\,711$, and is reached for two prefixes, the average value of $\operatorname{card}(h_{16}(s))$ (weighted with probabilities of s occurring) is about $30\,354.3$. This is not very useful for guessing the right value of the state of R1. However, the situation improves as n grows; chosen details are gathered in Table 1. Recall that there are only 2^{19} possible states of R1.

Table 1. Properties of the mappings h_n.

n	16	24	48	63–228
$\max\{\operatorname{card}(h_n(s))\}$	58 711	3656	10	10
$\operatorname{card}\{s\colon \operatorname{card}(h_n(s)) = 1\}$	1664	8893	334 479	338 602
$\operatorname{avr}\{\operatorname{card}(h_n(s)\}$	30 354.3	1857.99	1.62	1.59

We may proceed as follows: Choose for example $n = 48$ as a reasonable value, precompute a table of function h_{48}. There are exactly $412\,110$ reachable prefixes of length 48 (when R2 and R3 are set to all-one state), so the table can be easily precomputed and stored in the RAM memory. The table is sorted according to the prefixes, so finding all possible states of R1 can be done very quickly (e.g., by a binary search.) So now we have the state of R1, or some possibilities (at most 10 and about 1.62 on average) which can be verified as in 2.4. The state is of course valid only for this frame, so we have to go back to t_{64}.

2.2 Guessing the state of R1 before t_{64}

What we are really interested in is the state of R1 just before t_{64}. But we know nothing about the real values of R2 and R3, because we have destroyed them. So the standard method of backtracking the whole system ([3]) cannot be applied here. We will use a simpler technique here. If we think of the bits produced by a single register as of random bits, which are independent and unbiased, then it is quite easy to estimate how many times R1 have clocked during the second phase. Since the probability that one register clocks during one step is exactly $3/4$, the probability that during 100 steps R1 clocks k times is close to $\binom{100}{k}\left(\frac{3}{4}\right)^k\left(\frac{1}{4}\right)^{100-k}$. The expected value is 75 and we then calculate that

$$\Pr(65 \leqslant k \leqslant 84) \approx 0.98\,.$$

So we have only twenty possible values of k with the probability 0.98. Therefore we just go back with the state of R1 (well, one of 1.62 states on average) for 65 to 84 steps, and we then have on average about $20 \cdot 1.62 = 32.4$ possible values of R1 just before t_{64}.

The calculations are done under the assumption that these bits are indeed random, which is not the case, but in this context it is a good approximation. In many other works calculations are performed after making similar simplifications (see [1, 2, 3]). We have run a simulator a number of times for 10^4, 10^6 and 10^9 frames with random session keys and random frame numbers. The fraction of 0.98 for $k \in \{65, 66, \ldots, 84\}$ was confirmed every time for each of the three registers.

2.3 Guessing the state of R2 and R3

Having information about the state of R1 gives us new possibilities for finding the contents of R2 and R3, even without the fault model. After all, an attack on A5/1 without any knowledge about registers and without imposing faults is possible. But since we assume the possibility of setting two registers to the all-one states, we will go the same way (for different frames). So we can guess the state of R2 before t_{64} setting R1 and R3 to the all-one states just before the moment t_{186} and then observing the sequence generated. Let us consider a function that maps the prefixes generated to the states of R2 just before t_{186}. For prefixes of length $n \geqslant 66$ it reaches its optimal value of about 1.44 of possible states for a prefix (average weighted with probability of occurring a prefix). For $n = 48$ this average reaches 1.47 (and maximum 10). Since in this case we have exactly $3\,469\,103$ reachable prefixes of length 48, the table can be precomputed and stored in the RAM memory again. After guessing the state of R2 just before step t_{186} (1.47 possibilities on average), we can clock R2 back for $k \in \{65, 66, \ldots, 84\}$ times, obtaining about $20 \cdot 1.47 = 29.4$ guesses on average. Analogous steps are performed for R3. This time the mapping for $n = 48$ has the average number of states of R3 equal to 2.15 (and optimal value of the average is about 1.44 for $n \geqslant 71$). Now we consider prefixes of length 64, since they consume not much more memory and the average number states of R3 for a given prefix (weighted the probability of occurring the prefix) is very close to the optimal value and is about 1.44 (the maximum is 10). The number of prefixes of length 64, which are reachable in this way, equals $6\,990\,445$, so we precompute the table and store it in RAM again and quickly find all guesses for R3. We clock this state back for $k \in \{65, 66, \ldots, 84\}$ steps again, and obtain about $1.44 \cdot 20 = 28.8$ guesses for the state of R3 just before t_{64}.

2.4 Verification of the guesses

Now, in average case we have about $32.4 \cdot 29.4 \cdot 28.8 \approx 27\,434$ guesses for the whole system just before t_{64}. For any chosen frame we now run the A5/1 system and simultaneously run simulator fed with the frame number and the guesses obtained in the previous stage. In this way the wrong guesses are eliminated.

Note that we need only four frames—three for guessing the states of registers (with injecting the fault) and one for verification. Since the probability that the number k taken in 2.2 is wrong is bounded by 0.02, with probability at least $(0.98)^3 \approx 0.94$ our assumptions were correct and we get the right state.

3. Other fault models

One can make the model less restrictive, thus more realistic. For example the moment of imposing an error do not have to be that exact. One can let it slightly differ from t_{186}. The above considerations are then valid, only complexity of the attack grows, as we additionally have to guess the moment of the fault.

Changing the value of only one bit instead of the whole register could be a good idea, however it seems that the complexity of such an attack is much higher.

Quite another idea is to stop one of the registers from clocking for a moment and wait for a frame with assumed "behaviour". This model was investigated in a recent work [4].

Here we show some details of yet another idea, similar to the idea in Section 2.

Suppose a slightly more realistic situation where we can set most of the register's bits—but not necessarily all of them. In the model we will assume that the bits

$$11 - p, 11 - p + 1, \ldots, 11 + p - 1, 11 + p$$

of register R3 can be set to 1 at some moment (say, before t_{186}). The integer p is a parameter of the model, and cryptanalysis can be done without the exact knowledge of p. If p equals 11, then all bits are set to one and we are back in the model of Section 2. So we will assume that p is less than 11, that is, we have a kind of symmetric stain of ones in our register, as depicted by Fig. 2. Another weakening of the model is that we impose errors only on one register, and we will consider only two frames.

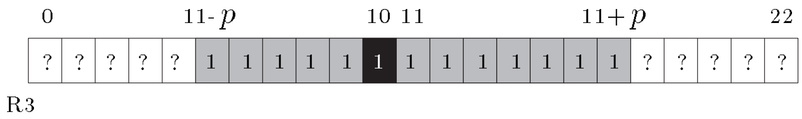

FIGURE 2. Stain of ones in R3.

In the cryptanalysis a reasonable value of p should be assumed. Too small a value will only affect complexity, as we shall see below. Too big a value makes cryptanalysis fail.

The main idea is to create a set of systems of linear equations and check them against the known output [1].

As usual, we assume that we know the output bits of the system. The estimation of the average number of systems of linear equations may be done as follows:

We have $64 - 2p - 1$ unknowns representing bits stored in the shift registers, so we require $64 - 2p - 1$ independent equations to find them. We will gradually guess bits needed to run the system ("decision" bits), move registers, and obtain additional equation from output bits of registers and output bits of the system.

At first we guess as many bits as it is needed for each register to move 10 times. Let us call this phase the *initial guess*. That means 10 bits for R1, 10 bits for R2 and $10 - p$ for R3 (see Fig. 3).

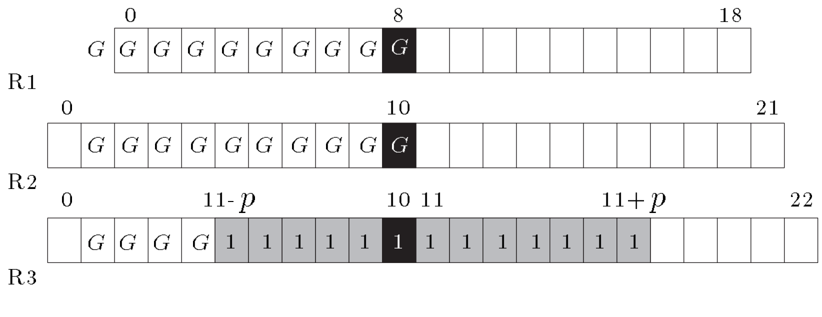

FIGURE 3. The bits from the initial guess are labeled by G.

One clock of the system moves each register with probability $\frac{3}{4}$, so on average we can make about $\frac{40}{3}$ clocks of the system, and obtain additional equations knowing the output. The first output bit is calculated at the beginning of step t_{186} without clocking any registers, so we have one more equation. So far we have guessed $30-p$ bits and we need about $64-2p-1-(30-p)-\frac{40}{3}-1=18\frac{2}{3}-p$ equations more.

After this phase, the guessed bits from R1 reach the output positions, so we cannot expect additional independent equations from the output bits (the guessed bits from R2 also approach the output positions). To simplify the analysis further we assume that all blocks of G-bits pass decision positions at the same step. Now the situation may be illustrated on Fig. 4. Note that the output position of R2 is in fact determined, since the output positions of other registers are known.

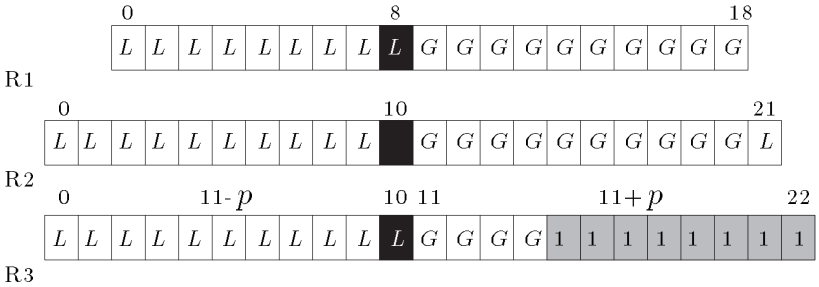

FIGURE 4. The situation before exploring the tree labeled by triples. G means a previously guessed value, L-involved in some linear equation.

We still lack about $18\frac{2}{3}-p$ equations, however, we do not have to explore all $2^{18\frac{2}{3}-p}$ possibilities, because if output bits are known, then some of the guessing of "decision" bits can be rejected immediately.

Consider a tree, constructed at this moment for each initial guess, where a node corresponds to the current state of our system, and is labeled by the "decision" triple of bits. Each child of the node corresponds to a possible next state of the system. If we did not care for fitting the known output, then for example the node labeled $(1,1,0)$ would have exactly four children $(0,0,0)$, $(0,1,0)$, $(1,0,0)$, $(1,1,0)$, because R3 does not move in this situation. Nodes labeled $(1,1,1)$ would have 8 children etc. In $\frac{3}{4}$ of the cases a node would have 4 successors and in $\frac{1}{4}$ of the cases there would be 8 successors, so on average a node would have 5 successors. But in the tree we are constructing about half of the options are rejected due to the known output, so our tree has an average outdegree equal to 2.5.

If the tree considered has a depth h, then it corresponds to h clocks of the whole system, therefore it forces us to guess about $\frac{3}{4}h$ bits on each register. We need $18\frac{2}{3} - p$ bits, so the average depth of a sufficient tree equals

$$\frac{4}{3} \cdot \frac{18\frac{2}{3} - p}{3}.$$

Since the average outdegree of an average internal node is 2.5, we have obtained

$$2.5^{\frac{4}{3} \cdot \frac{18\frac{2}{3} - p}{3}} = 2^{\frac{4}{9} \cdot (18\frac{2}{3} - p) \cdot \log_2 2.5}$$

possibilities.

We did 2^{30-p} initial guesses, and for each guessed equation we find about $2^{\frac{4}{9} \cdot (18.67 - p) \cdot 1.32}$ options, so finally we have about $2^{40.97 - 1.59p}$ systems of equations to solve. Since we will stop after finding the solution that fits our output, the average number of considered equations is about

$$2^{39.97 - 1.59p}.$$

As p grows (to at most 10, of course) this complexity becomes more reasonable. The value of $p \geqslant 7$ leads to less than 2^{30} systems of linear equations. Recall that most equations in such a system have only one variable (since about thirty of $64 - 2p - 1$ equations are just values found during the initial guess, and we express all unknowns in terms of bits of registers in t_{186}.) All other equations may be represented very efficiently using 64-bit-words, and solving such a system requires only few 64-bit-XORs.

After finding the system which fits our output, we have internal states of R1 and R2 in the moment t_{186}. Now, similarly as in Section 2.2 we find states of R1 and R2 before t_{64} (about 400 possibilities, 20 for both registers, with less than 4 % chances for mistake). Having states of R1 and R2 in t_{64} of this frame, we now go to any other frame of the same session, states of R1 and R2 in t_{64} are the same for the whole session, and find the state of R3 by just guessing its 23 bits in t_{64} and running a simulator for the frame.

The complexity of this attack is the complexity of finding solutions of about $2^{39.97 - 1.59p}$ systems of linear equations and about $400 \cdot 2^{23}$ frame runs on a simulator.

REFERENCES

[1] BIHAM, E.—DUNKELMAN, O.: *Cryptanalysis of the A5/1 GSM stream cipher*, in: INDOCRYPT '00, Lecture Notes in Comput. Sci., Vol. 1977, Springer-Verlag, Berlin, 2000, pp. 43–51.

[2] BIRYUKOV, A.—SHAMIR, A.—WAGNER, D.: *Real time cryptanalysis of A5/1 on a PC*, in: FSE '00, Lecture Notes in Comput. Sci., Vol. 1978, Springer-Verlag, Berlin, 2001, pp. 1–18.

[3] GOLIČ, Dj.: *Cryptanalysis of alleged A5 stream cipher*, in: Proceedings of EUROCRYPT '97, Lecture Notes in Comput. Sci., Vol. 1233, Springer-Verlag, Berlin, 1997, pp. 239–255.

[4] GOMUŁKIEWICZ, M.—KUTYŁOWSKI, M.—VIERHAUS, H. T.—WLAŹ, P.: *Synchronization fault cryptanalysis for breaking A5/1*, in: Lecture Notes in Comput. Sci., Vol. 3503, Springer-Verlag, Berlin, 2005, 415–427, http://dx.doi.org/10.1007/11427186_36.

[5] PAILLIER, P.: *Evaluating differential fault analysis of unknown cryptosystems*, Public Key Cryptography (1999), 235–244.

[6] ZHENG, Y.—MATSUMOTO, T.: *Breaking real-world implementations of cryptosystems by manipulating their random number generation*, in: In Pre-proceedings of the Symposium on Cryptography and Information Security, Fukuoka, 29th January – 1st February 1997.

Received October 1, 2004

Marcin Gomułkiewicz
Mirosław Kutyłowski
Institute of Mathematics
Wroclaw University of Technology
ul. Wybrzeże Wyspiańskiego 27
PL–50-370 Wrocław
POLAND
E-mail: gomulkie@im.pwr.wroc.pl
Miroslaw.Kutylowski@pwr.wroc.pl

Paweł Wlaź
Department of Mathematics
Lublin University of Technology
ul. Nadbystrzycka 38d
PL–20-950 Lublin
POLAND
E-mail: p.wlaz@pollub.pl