Software-Hardware Trade-offs; application to A5/1 Cryptanalysis

Thomas Pornin[1] and Jacques Stern[2]

[1] Département d'Informatique, École Normale Supérieure, 45 rue d'Ulm, 75005 Paris, France, thomas.pornin@ens.fr
[2] idem, jacques.stern@ens.fr

Abstract This paper shows how a well-balanced trade-off between a generic workstation and dumb but fast reconfigurable hardware can lead to a more efficient implementation of a cryptanalysis than a full hardware or a full software implementation. A realistic cryptanalysis of the A5/1 GSM stream cipher is presented as an illustration of such trade-off. We mention that our cryptanalysis requires only a minimal amount of cipher output and cannot be compared to the attack recently announced by Alex Biryukov, Adi Shamir and David Wagner[2].

Keywords: A5/1, GSM, stream cipher, FPGA, cryptanalysis, trade-off

1 Introduction

There are two main species of computer devices that are used by cryptanalysts: generic all-purposes workstations, and specialized hardware devices. Among the latters, Field Programmable Gate Arrays are more and more used, since they give a good performance/cost ratio and a fast and cheap development cycle.

Operations that are easy to implement on a FPGA include all bit permutations, shifts, bitwise logical operations, and small lookup tables. This makes them especially well-suited for implementing block ciphers such as DES, and all stream ciphers and random generators using Linear Feedback Shift Registers (LFSR). However, the low-level structure of such devices makes it almost impossible to implement of a high-level algorithm whose behaviour is dependant on the input data. A branching process or a recursive search in a tree are definitely out of reach.

Workstations, on the contrary, are good at running complex algorithms, since conditional execution, function calls and stack memory structures are natural on these platforms. They are also especially optimized at performing complex mathematic operations such as integer multiplications or floating point calculations. Yet, they are ineffective at more simple operations, in proportion to their cost: an expensive 21264 Alpha processor will perform only four bitwise logical operations per cycle on 64-bit registers, despite its over 15 millions transistors.

We present here a study on a trade-off between these two technologies. The chosen algorithm is A5/1; this stream cipher is used in GSM mobile phones to ensure confidentiality of "over the air" communication. A5/1 was published in [1] unofficially, but Alex Biryukov, Adi Shamir and David Wagner claim in [2] that they received confirmation from the GSM organization that this design is the true A5/1 as used in GSM phones. Therefore we will assume that the A5/1 described here is indeed the correct algorithm; anyway, this algorithm is merely used as an illustration of our technique.

Recently, Alex Biryukov, Adi Shamir and David Wagner presented in [2] an impressive attack against the A5/1 cipher; this attack is a time-memory trade-off that requires a non-negligeable amount of known plaintext (about 25000 bits). Since the internal state of A5/1 is only 64-bit, it should be recoverable with only 64-bit of known plaintext; we therefore consider this framework, where only 64 consecutive bits or so of plaintext (and the corresponding ciphertext) have been intercepted. Moreover, the time-memory trade-off of [2] is made very effective due to many features of A5/1 that allow some smart optimizations. We do not use such features, and our work should be applicable to other similar ciphers.

The hardware used is a Compaq XP-1000 workstation (21264 Alpha processor at 500 MHz) and Compaq (formerly Digital) Pamette cards; a Pamette is a PCI card that includes five Xilinx 4010E FPGA. One of these FPGA is used to handle the PCI bus; there is room for some SDRAM connected to two of the FPGA. At the time of writing this paper, an XP-1000 is a 3000$ workstation, and a Pamette costs about 1000$.

2 Description of A5/1

A5/1 is a neat design that uses a very small amount of silicium when implemented in hardware. It includes three LFSR, with a clocking sequence depending on the internal state of the three registers. It outputs a stream of bits that is combined (by mean of an exclusive or) with the data to encipher.

The three LFSR are of length 19, 22 and 23 bits. At each clock cycle, a majority bit is calculated, from the three middle bits of the registers; those registers which middle bit agrees with the majority are shifted. Then the output bit is the exclusive or of the three final bits of the registers. Figure 1 illustrates this mechanism. A full description of the algorithm may be found in [1].

The majority clocking implies that, at each clock cycle, there are four possible moves:

- register 1 and 2 are shifted
- register 1 and 3 are shifted
- register 2 and 3 are shifted
- all registers are shifted

The internal state is loaded with a 64-bit session key and a 22-bit known counter; the cipher is then ran for 100 cycles and the corresponding output bits discarded, and then 228 bits are produced for enciphering the data. Then the cipher is reset, with the same key and the next counter value. The key can easily be recovered from the internal state at any moment with a critical

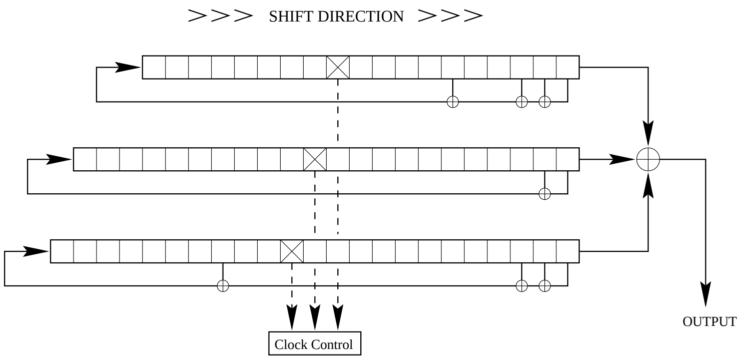

Figure 1: The A5/1 Stream Cipher

branching process exposed in [3]; therefore, once one internal state of A5/1 has been revealed, the cryptanalysis is to be considered complete, since the same session key is used throughout the entire phone conversation.

In GSM phones, the session keys are produced with another algorithm, which might depend on the operator. Marc Briceno, Ian Goldberg and David Wagner, who published in May 1999 in [1] the first complete description of A5/1, claim that all the implementations they checked used 54-bit session keys (that is, 64-bit keys with 10 fixed bits set to 0). Although other operators could decide otherwise, it is probable that this convention will be maintained by operators in the name of backward compatibility. Still, our work does not make use of this feature.

3 Software Cryptanalysis of A5/1

A first cryptanalysis of A5/1 was first informally presented by Ross Anderson, who published in 1994 [4] an alleged description of A5/1 (which turned out to be mostly correct, except for the position of bits for clocking and linear feedback). The idea is to guess the two first registers, and half of the third register, which is basically enough to know the clocking sequence and deduce the second half of the third register by solving a system of linear equations. This attack is applicable to the real A5/1, with a workload of about 2^{52} guesses (each implying the resolution of a system of a dozen linear equations).

Then Jovan Golić presented, in 1997 [3], a complex cryptanalysis with an average complexity of slightly above 2^{40} operations; however, each operation is a resolution of a 64×64 linear system, and some of the assumptions used to get the claimed complexity are somehow irrealistic since they lead to an overly complex and slow implementation.

The Golić attack is, basically, guessing the clock sequence for a given number of clock cycles, adjusting it if necessary by adding some more guesses; knowing the clock sequence, each output bit is a linear equation of known internal state bits. The guess also gives other linear equations, which describe the majority function. When enough equations are obtained, the system is reversed, and the potential initial state is recovered and tested against the remaining known output bits.

We implemented a simplified version of the Golić attack. This is done by backtrack in a tree representing at depth n the different internal states after n clock cycles. So each node has four subtrees, since there are four possible moves at each cycle. Each guess in the backtrack process is taking one of the four branches; this gives us three equations:

- two equations represent the clock control calculation
- one equation is the calculation of the output bit

These equations are linear in $\mathbf{Z}_2$, with 64 unknown values. These values are the 64-bit initial state, and, at each step of the algorithm, each bit of one LFSR is a linear combination of several bits of the initial sate of the same LFSR; this combination depends only on the number of times the LFSR has been clocked since the initial state. So, if we call c_1, c_2 and c_3 the clocking bits of the respective three LFSR at one step, and guess that registers 1 and 2 move, and register 3 does not, we get the following two equations:

$$c_1 + c_2 = 0$$
$$c_1 + c_3 = 1$$

The third equation is similar: if, after clocking, the end bits of the three LFSR are respectively e_1, e_2 and e_3, and the output bit is v, then we have the following:

$$e_1 + e_2 + e_3 = v$$

We maintain, during the backtrack, a system of such equations describing the previous steps of the algorithm starting from the initial state; this system is triangular, which means the following: for each equation n, there exists one of the unknowns such that its coefficient in equation n is 1, and such that in all following equations (equations $n+1$, $n+2$, $\dots$) its coefficient is 0. When the system is complete (64 equations), equation 64 is:

$$x = k_1$$

where x is one of the unknowns, and k_1 is a constant value (0 or 1). Equation 63 is:

$$y + k_2 x = k_3$$

where y is another unknown value, and k_2 and k_3 are constants. So, once x value is known, y is known too. We can go on with this process up to equation 1, and therefore simply recover the whole 64 unknown values. This is the standard, well-known method of linear system solving, due to Gauss.

So, when we add one equation to the yet incomplete system, we need to perform the Gaussian elimination of this equation relatively to the preceedings. If we call u_i the unknown value whose coefficient is 1 in equation i and 0 in all equations j for $j > i$, we apply the following algorithm when we add equation n to the system:

1. call X the equation to add
2. for i from 1 to $n-1$
3. if u_i has a coefficient 1 in X, add equation i to X
4. next i
5. append X to the system
6. find the first non-zero coefficient in X, call u_n the corresponding unknown

The last action of this algorithm may fail, if all coefficients of X are set to 0 by the elimination process. Then X is either $0 = 0$ or $0 = 1$. If we get $0 = 0$, this means that the new equation can de deduced linearly from the preceedings, so we just throw it away and keep on with the backtrack. However, if we get $0 = 1$, we are lucky: we know that the path in the tree of possible clocking sequences, up to the point that has been reached, is wrong. If this happens at clocking step 19, we go back to clocking step 18, and assume that the last guess was wrong. So we forget the equations added by that last move, an go on with another guess for that move. This is where we optimize the Golić attack: we can keep all equations corresponding to step 1 to 18, and we do not have to perform the Gaussian elimination on them again.

This calculation can be implemented effectively on modern workstations: since each coefficient is 0 or 1, it can be stored as one bit. Each equation is a 65-bit word (64 bits for the 64 coefficients, one bit for the constant on the right hand side of the equation). An addition of two equations is a bitwise exclusive or, a native operation on modern processors. Finding the first bit set to one in a 64-bit word may be performed by a dichotomic process, which gives the result in 6 masking/compare/shift group of operations.

Once we have 64 linearly independant equations, in a triangular representation, we might solve the system, recover the initial state, and run A5/1 with this initial state to see if it matches the known output; however, it is more efficient to keep on with the elimination. At each step, since the system is complete, all added equations will be reduced to either $0 = 0$ or $0 = 1$. Only one of the four possible clocking steps will produce two $0 = 0$ equations (since the system is complete, it contains the whole information on the execution of A5/1, and the clocking behaviour is deterministic given this information), and the third equation, depending on the ouput bit, will yield $0 = 0$ with probability 0.5, and $0 = 1$ otherwise. So, on average, we must go two steps further in the backtrack process to check the correctness of the guessed clocking sequence (this means six more equations reduced).

Experiments show that total eliminations (equation X has a left hand side equal to 0) are very rare before step 21; this is coherent with the intuitive idea that we cannot find anything on the internal state of A5/1 before the registers have wrapped around. The complexity of the backtrack is therefore the expected value $4^{64/3} \times 6$, which is about $2^{45.3}$. Each operation is the Gaussian elimination of one equation according to an average of about 64 preceedings linear equations (most of the computation time is spent in the leaves of the tree, where the linear system is complete, or almost). This is the complexity for the whole search; on the average, we find the correct clocking sequence after exploring half of the tree, so the complexity is $2^{44.3}$. We claim that this is the same complexity as the Golić one, expressed in a more realistic unit.

Our implementation takes 400 days on a Compaq-XP1000 (21264 Alpha processor at 500 MHz) to explore the full tree; this yields an average software-only cryptanalysis time of 200 days on one workstation.

4 Hardware Cryptanalysis of A5/1

4.1 Description of the FPGA Pamette Card

The Pamette card includes five Xilinx 4010E FPGA chips; one of them is dedicated to the handling of the PCI bus. Each 4010E is a matrix of reconfigurable units called Configurable Logic Blocks (CLB).

Each CLB includes:

- two $4 \rightarrow 1$ reconfigurable lookup tables
- one $3 \rightarrow 1$ reconfigurable lookup table, two entries of which are the outputs of the two preceeding lookup tables
- two one-bit registers

Figure 2 gives an insight of a CLB. The two $4 \rightarrow 1$ and the $3 \rightarrow 1$ functions are fully configurable (they are implemented as lookup tables). There are four outputs, two of them corresponding to two one-bit registers; each register is controlled by an "enable" input, that can be set either to 1 (the register always updates) or connected to one output of a CLB (possibly the same). The initial value of each register is either 0 and 1, and this is configurable.

There are $24 \times 24 = 576$ CLB in a 4010E chip; the interconnecting matrix is also highly configurable; up to eight parallel signals can be carried between two rows of CLB.

The Xilinx chips are connected with each other through 16-bit and 8-bit busses; two of the four available chips are connected to optional static RAM, and to the fifth chip (the PCI-handler) with a 32-bit wide bus. The whole card may be clocked up to 66 MHz, depending on the design (to run at 66 MHz, there must be only one CLB and no long routing between two registers).

A more complete description of a Xilinx chip is available from Xilinx (see [5] for details). The Pamette itself is from Compaq (formerly Digital) and is described in [6].

4.2 Implementation of A5/1 on a Pamette

It is possible to implement A5/1 on a Xilinx 4010E with the following characteristics:

- At each cycle, one step of A5/1 is performed.

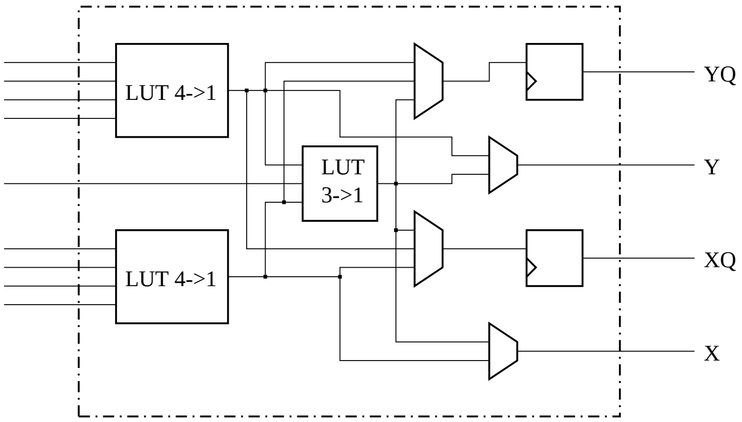

Figure 2: A Configurable Logic Bloc

- It is possible to reload the LFSRs with new values in one cycle.
- The resulting design runs at 50 MHz.
- 12 parallel instances of A5/1 may be put into each Xilinx chip.

The details of the implementation are fairly straightforward. The clocking bit is calculated from the three clocking bits of the registers, with one extra bit indicating that a new initial state must be loaded into the registers. This clocking bit is used as the "enable" input for the registers. The clocking bit calculation requires only half a CLB; we also use half a CLB for each bit in each LFSR (one bit register to store the value, one lookup table to feed the register with either the preceeding value, or a new value, when another initial state must be loaded). With the feedback computation (one half-CLB) and the comparison with the reference value (1.5 to 3.5 CLB), a whole instance of A5/1 requires only 36 or 38 CLBs (depending whether we want to compute 32, 64 or more output bits). So we may store twelve instances of A5/1 on each Xilinx chip and still have room for the synchronization clocks and the shared counter, which gives the successive initial states to try (the twelve instances try the same initial states except for some bits, so they share the counter).

Since the design is really compact (all critical data exchanges are local to one small area of the chip) and the computational depth (maximal number of CLB between two registers) is small (only 2 CLB at most must be gone through at each cycle), we can run the whole design at 50 MHz.

Therefore, we have 48 parallel implentations of A5/1 in one Pamette, that may try 64 A5/1 steps in 64 cycles. One more cycle will be needed for the reload of the state, so we can try up to 37 millions initial states per second per Pamette. This is faster than the best known software implementation of A5/1, described in [2], which can treat up to 8 steps at a time, but runs at the speed of a workstation's RAM. Pamette cards give a high degree of intrinsic parallelism.

So, if we want to perform an exhaustive search on the 64-bit internal state, we need about 15800 Pamette-years, that is 15800 years with one Pamette, or

1 year with 15800 Pamettes. We might want to take advantage of the alleged fact that session keys are only 54-bit, not 64-bit. Then we must, for each guess, perform the 100 discarding states, which drops the Pamette efficiency to about 14.5 millions key tests per second; however, the exhaustive search workload is divided by a factor of $2^{10} = 1024$, which leads to a total effort of about 39 Pamette-years. This is reachable by many agencies and businesses around the world, although not quite efficient for daily cryptanalysis.

For completeness, we must add that the infrastructure needed is small: actually, there is no real problem of data bandwidth. Each instance of A5/1 runs isolated from the others, and the only data that has to be exchanged is the initial setting of the FPGA (but loading a Xilinx 4010E with a given design is a matter of milliseconds), and one bit from one instance of A5/1 to indicate that a matching initial state has been found. The controlling PC has a very simple job: it waits for the bit to be set, and measures the time taken from the beginning of the search. From this measure, the matching state can be narrowed to a set of only a few millions candidates, which can be precisely tried in a few seconds on the cheapest of nowadays PCs.

5 The Software-Hardware Trade-off

An intuitive, information theory oriented point of view is that the minimal workload to cryptanalyse A5/1 is something like $4^{64/3}$. In this approach, the clocking sequence is considered as intractable; it cannot be controlled except with an exhaustive search. Since the initial state is 64-bit, we need 64 binary data in order to cryptanalyse A5/1. From each step, we get one bit from the ouput, and two bits from the clocking sequence, since four clocking steps are possible. We will not have our 64 equations until we have considered at least 64/3 steps, and then the exhaustive search will have cost us $4^{64/3}$ operations.

The software cryptanalysis presented in section 3 sticks as close as possible to this workload. On the contrary, the hardware exhaustive search is way above it, but may be ran on a really dumb but fast device. Indeed, any conditionnal code (and there is many in a backtrack) is a pain to implement on a FPGA; it usually ends up in reimplementing a complete cpu, which is a misuse of the hardware, since a real cpu of comparable cost will be much more optimized for this task.

The main idea of the trade-off is to make part of the job with a software implementation, but to jump over the “complexity barrier” with an hardware implementation. This is a trade-off between an increased workload and the possibility to perform part of this workload on an efficient hardware device.

The software part is the beginning of the software cryptanalysis. We perform an exhaustive search on the clocking sequence on the first n steps of A5/1 (for instance, with $n = 17$). Each guess will give us $3n$ linear equations in the initial state; the workstation will then solve each system, that is exhibit the $(64 - 3n + 1)$ 64-bit vectors that represent a basis for the affine subspace in $\mathbf{Z}_2^{64}$ which holds all the solutions to this $3n$ equations system ($(64 - 3n)$ vectors for the basis of the corresponding linear subspace, and one more vector as origin of the affine subspace). Then these vectors are sent to the Pamette card, which then exhaustively tries all elements of this subspace as initial states; this is a matter of 2^{64-3n} executions of A5/1.

For instance, for $n = 17$, the software part will have to generate 2^{34} systems; solving each system is about twice the cost of performing the Gauss elimination on each of them. For each system, the affine subspace of solutions contains 2^{13} elements, so the workload for the Pamette Card is $2^{34} \times 2^{13} = 2^{47}$ executions of A5/1, at Pamette speed. With a correct balancing between the software and the hardware part (that is, an appropriate choice of n), we can achieve a much lower cryptanalysis time than a full-hardware or full-software solution.

There are two possible optimizations in this method:

- Most of the time, the $3n$ equations are linearly independant. It is therefore not necessary to handle the rare case where the reversing of the system gives either an impossibility or a wider subspace of solutions: we just discard these occurrences. Therefore, 5% or so of cryptanalysis are not successful; we believe this is acceptable. This does not actually improve performances but greatly simplifies the implementation.
- It is not needed to test in the Pamette each initial state against the whole 64-bit output. All we need to do is test against enough bits so that the average case is that no initial state matches, and so that it is very unlikely that two or more initial states match. For instance, if $n = 17$, me might try only 32 bits of output; one subspace every 32768 (on average) will contain a match, and this can be handled easily in software. The hardware testing will be twice faster than if all 64 bits had to be matched for in hardware.

The optimal choice of n heavily depends on the number of workstations and the number of Pamettes available (in fact, on the ratio between these two numbers). We give here numbers for one XP-1000 Alpha station, and two 4010E Pamettes; these are durations for execution of the workload:

n	**soft. load**	**soft. time**	**hard. load**	**hard. time**
16	2^{32}	0.09	2^{48}	22
17	2^{34}	0.39	2^{47}	11
18	2^{36}	1.7	2^{46}	5
19	2^{38}	7.1	2^{45}	2.5
20	2^{40}	30	2^{44}	1.3

The time figures are expressed in days; this corresponds to the full cryptanalysis, that is the worst case. The average cryptanalysis will take up half of the time given. We consider that the Pamette will try to match against 32 bits of output.

We see that, when there are two Pamettes for each workstation, the optimal n is 18, which allows cryptanalysis in 2.5 days on average. By comparison, with the same investment, we could have two workstations, that would perform the full software cryptanalysis in 200 days (average time: 100 days). So we have a factor of forty in performance, and still have some computing power available on the workstation (some of which is used to check the about 2^{14} subspaces that contain an initial state that gives 32 correct output bits; this means 2^{24} software checks, that is only a few seconds on the workstation). A full hardware exhaustive search is definitely out of the question, even if we take benefit of the reduced session key size.

6 Conclusion

We showed how a right balancing between a tricky software and a dumb hardware implementations can dramatically speed up a cryptanalysis of A5/1. With a small investment (less than 20000$), it is quite possible to uncover an intercepted GSM communication in a realistic interception scenario: although we do not have a complete specification of the GSM protocol, we believe that it is easy to guess 64 bits of communication. This is, in our point of view, a much more applicable attack in the real world, than the (although impressive) attack from Biryukov, Shamir and Wagner, since this latter requires an average of two seconds of exact plaintext.

Morevover, we did not use any specific characteristic of A5/1 such as the position of clocking bits or the feedback function, so this study has a much wider impact than GSM privacy. All LFSR-based pseudo-random generators, with a data-controlled clock sequence, might be affected by this technique. We strongly suggest that such generators be given an internal state of 128 bits at least.

References

1. Marc Briceno, Ian Goldberg, David Wagner, *A pedagogical implementation of A5/1*, web publication, `http://www.scard.org/gsm/body.html`, 1999.
2. Alex Biryukov, Adi Shamir, David Wagner, *Real Time Cryptanalysis of A5/1 on a PC*, presented at FSE2000.
3. Jovan Dj. Golić, *Cryptanalysis of Alleged A5 Stream Cipher* Lecture Notes in Computer Science, Advances in Cryptology, proceedings of EUROCRYPT'97, pp. 239–255, 1997.
4. Ross Anderson, *A5 (Was: HACKING DIGITAL PHONES)* Usenet communication on sci.crypt, alt.security and uk.telecom, June 17th 1994.
5. The Xilinx web site, `http://www.xilinx.com/`
6. The Pamette main web site, `http://www.research.digital.com/SRC/pamette/`