Real Time Cryptanalysis of the Alleged A5/1 on a PC (preliminary draft)

Alex Biryukov Adi Shamir*

December 9, 1999

Abstract

A5/1 is the strong version of the encryption algorithm used by about 100 million GSM customers in Europe to protect the over-the-air privacy of their cellular voice and data communication. The best published attacks against it require between 2^{40} and 2^{45} steps. This level of security makes it vulnerable to hardware-based attacks by large organizations, but not to software-based attacks on multiple targets by hackers.

In this paper we describe a new attack on A5/1, which is based on subtle flaws in the tap structure of the registers, their noninvertible clocking mechanism, and their frequent resets. The attack can find the key in less than a second on a single PC with 128 MB RAM and two 73 GB hard disks, by analysing the output of the A5/1 algorithm in the first two minutes of the conversation. The attack requires a one time parallelizable data preparation stage whose complexity can be traded-off between 2^{38} and 2^{48} steps. The attack was verified with an actual implementation, except for the preprocessing stage which was extensively sampled rather than completely executed.

DISCLAIMER: This paper considers the narrow issue of the cryptographic strength of A5/1, and not the broader issue of the practical security of fielded GSM systems, about which we make no claims. We base our attack on the unofficial version of the algorithm which was derived by reverse engineering actual GSM SIM cards and published at `http://www.scard.org`, and use the term A5/1 to refer to this particular version. Any discrepancies between this version and the real algorithm used in GSM telephones may affect the validity or performance of our attack.

*Computer Science department The Weizmann Institute Rehovot 76100, Israel.

1 Introduction

The over-the-air privacy of GSM telephone conversations is protected by the A5 stream cipher. This algorithm has two main variants: The stronger A5/1 version is used by about 100 million customers in Europe, while the weaker A5/2 version is used by another 100 million customers in other markets.

The approximate design of A5/1 was leaked in 1994, and the exact design of both A5/1 and A5/2 was reverse engineered by Briceno from actual GSM smart cards in 1999 and confirmed against official test vectors (see [2]). In this paper we assume that these unofficial descriptions of the algorithm and protocol are accurate.

In this paper we describe a cryptanalytic attack on A5/1, in which a single PC can extract the conversation key in less than a second from the output produced by the algorithm in the first two minutes. Many of the ideas are applicable to other stream ciphers as well, and define new measures of strength.

The paper is organized in the following way: Section 2 contains a full description of the A5/1 algorithm. Previous attacks on A5/1 are surveyed in Section 3, and an informal description of the new attack is contained in Section 4. Finally, Section 5 contains various implementation details and an analysis of the expected success rate of the attack, based on large scale sampling with an actual implementation.

2 Description of the A5/1 stream cipher

A GSM conversation is sent as a sequence of frames. Each frame contains 114 bits representing the digitized A to B communication, and 114 bits representing the digitized B to A communication. A new frame is sent every 4.6 milliseconds, and each frame has a publicly known 22 bit frame counter F_n which cycles every 2^{22} frames (i.e., a few hours).

Each GSM phone conversation can be encrypted by a new session key K, which is derived in a noninvertible way from the user's master key and a random value by another algorithm known as A8. For each frame, K is mixed with the frame counter F_n, and the result serves as the initial state of a generator which produces 228 pseudo random bits. These bits are XOR'ed with the 228 bits of the plaintext to produce the 228 bits of the ciphertext.

A5/1 is built from three short linear feedback shift registers (LFSR) of lengths 19, 22, and 23 bits, which are denoted by $R1, R2$ and $R3$ respectively. The rightmost bit in each register is labelled as bit zero. The taps of $R1$ are at bit positions 13,16,17,18; the taps of $R2$ are at bit positions 20,21; and the taps of $R3$ are at bit positions 7, 20,21,22 (see Figure 1). When a register is clocked, its taps are XORed together, and the result is stored in the rightmost bit of the left-shifted register. The three registers are clocked in a stop/go fashion using the following majority rule: Each register has a single "clocking" tap (bit 8 for

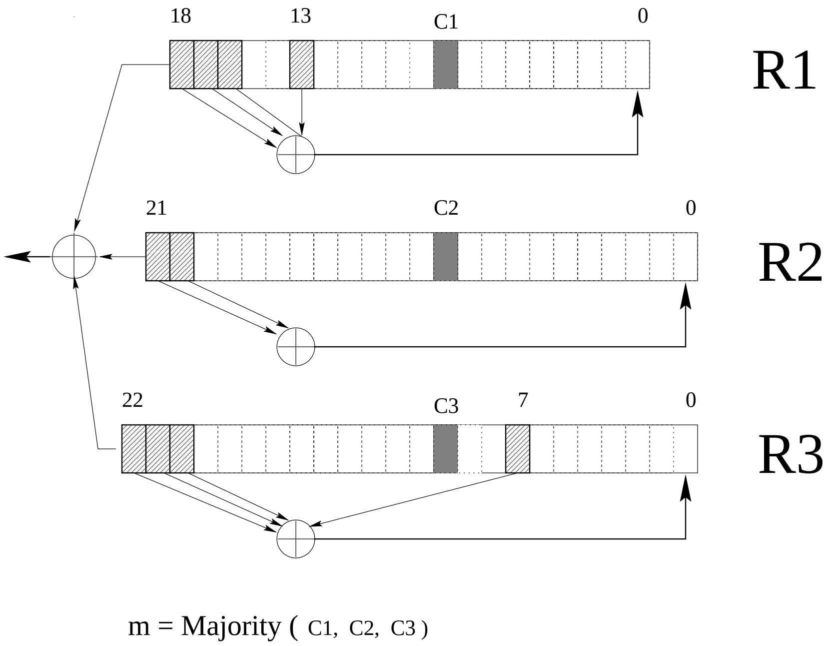

Figure 1: The A5/1 stream cipher.

$R1$, bit 10 for $R2$, and bit 10 for for $R3$); each clock cycle, the majority function of the clocking taps is calculated and only those registers whose clocking taps agree with the majority bit are actually clocked. Note that at each step either two or three registers are clocked, and that each register moves with probability 3/4 and stops with probability 1/4.

The process of generating pseudo random bits from the session key K and the frame counter F_n is carried out in four steps:

- The three registers are zeroed, and then clocked for 64 cycles (ignoring the stop/go clock control). During this period each bit of K (from lsb to msb) is XOR'ed in parallel into the lsb's of the three registers.

- The three registers are clocked for 22 additional cycles (ignoring the stop/go clock control). During this period the successive bits of F_n (from lsb to msb) are again XOR'ed in parallel into the lsb's of the three registers. The contents of the three registers at the end of this step is called the initial state of the frame.

- The three registers are clocked for 100 additional clock cycles with the stop/go clock control but without producing any outputs.

- The three registers are clocked for 228 additional clock cycles with the stop/go clock control in order to produce the 228 output bits. At each clock cycle, one output bit is produced as the XOR of the msb's of the three registers.

3 Previous attacks

The attacker is assumed to know some pseudo random bits generated by A5/1 in some of the frames. This is the standard cryptanalytic assumption in stream ciphers, and we do not consider in this paper the crucial issue of how one can obtain these bits in fielded GSM systems. For the sake of simplicity, we assume that the attacker has complete knowledge of the outputs of the A5/1 algorithm for the first two minutes of the conversation, and his goal is to find the key in order to decrypt the remaining part of the conversation, or to use such keys in order to attack the user's master key. Since GSM telephones send a new frame every 4.6 milliseconds, the first two minutes of the conversation contain $120 * 1000/4.6 \approx 2^{15}$ frames.

At the rump session of Crypto 99, David Wagner announced an attack on A5/2 which requires very few pseudo random bits and just $O(2^{16})$ steps. This demonstrated that the "export version" A5/2 is totally insecure.

The security of the A5/1 encryption algorithm was analyzed in several papers. Some of them are based on the early imprecise description of this algorithm, and thus their details have to be slightly modified. The known attacks can be summarized in the following way:

- Briceno[2] found out that in all the deployed versions of the A5/1 algorithm, the top 10 of the 64 key bits were always set to zero. The complexity of exhaustive search is thus reduced to $O(2^{54})$. [1]

- Anderson and Roe[1] proposed an attack based on guessing the 41 bits in the shorter R_1 and R_2 registers, and deriving the 23 bits of the longer R_3 register from the output. However, they occasionally have to guess additional bits to determine the majority-based clocking sequence, and thus the total complexity of the attack is about $O(2^{45})$. Assuming that a standard PC can test ten million guesses per second, this attack needs more than a month to find one key.

- Golic[3] described an improved attack which requires $O(2^{40})$ steps. However, each operation in this attack is much more complicated, since it is based on the solution of a system of linear equations. In practice, this algorithm is not likely to be faster than Anderson's attack on a PC.

[1] Our new attack is not based on this assumption, and is thus applicable to A5/1 implementations with full 64 bit keys. It is an interesting open problem whether we can speed it up by assuming that 10 key bits are zero.

- Golic[3] describes a general time-memory tradeoff attack, and concludes that it is possible to find the A5/1 key in 2^{22} probes into random locations in a precomputed table with 2^{42} 128 bit entries. Since such a table requires a 64 terabyte hard disk, the space requirement is unrealistic. Alternatively, it is possible to reduce the space requirement to 862 gigabytes, but then the number of probes increases to $O(2^{28})$. Since random access to the fastest commercially available PC disks requires about 6 milliseconds, the total probing time is almost three weeks. In addition, this tradeoff point can only be used to attack GSM phone conversations which last more than 3 hours, which again makes it unrealistic.

4 Informal Description of the New Attack

We start with an executive summary of the key ideas of the new attack. More technical descriptions of the various steps will be provided in the next section.

Key idea 1: Use a time-memory tradeoff attack.

The starting point for the new attack is the time-memory tradeoff described in Golic[3], which is itself an adaptation of the time-memory tradeoff developed by Hellman for block ciphers. Such attacks can be applied to almost any cryptosystem, but they are practical only when the number of internal states is relatively small. A5/1 has this weakness, since its state at any given time is defined by the 19+22+23 bits in its three shift registers, and thus it has exactly $n = 2^{64}$ possible states. The basic idea of the time-memory tradeoff is to keep a large set A of precomputed states on a hard disk, and to consider the large set B of states through which the algorithm progresses during the actual generation of output bits. Any intersection between A and B will enable us to identify an actual state of the algorithm from stored information.

Key idea 2: Identify states by prefixes of their output sequences.

Each state defines an infinite sequence of output bits produced when we start clocking the algorithm from that state. We can look for equality between unknown states by comparing their output sequences. In fact, it usually suffices to compare the $log(n)$ initial bits in the output sequences in order to make a correct decision. The mapping from states to prefixes is easy to compute and deterministic, whereas the mapping from prefixes to states is difficult to compute and may yield 0, 1, or several possible results. To choose the set A during precomputation, pick a subset of states, compute their output prefixes, and store the (prefix, state) pairs sorted into increasing prefix values. Given actual outputs of the A5/1 algorithm, extract all their (partially overlapping) prefixes, and define B as the set of their corresponding (unknown) states. Searching for common states in A and B can be efficiently done by probing the sorted data A on the hard disk with prefix queries from B.

Key idea 3: A5/1 can be efficiently inverted.

As observed by Golic, the state transition function of A5/1 is not uniquely invertible: The majority clock control rule implies that up to 4 states can converge to a common state in one clock cycle, and some states have no predecessors. However, we can run A5/1 backwards by exploring the tree of possible predecessor states, and backtracking from dead ends. This process is modeled as a branching process, in which each vertex independently chooses the number of its sons according to some common probability distribution. The branching process is critical in the sense that the average number of sons of each vertex is exactly 1. Critical branching processes are known to produce trees in which the expected number of vertices in the first k levels grows only linearly in k. As a result, if we find a common state in the disk and data, we can efficiently run the algorithm backwards (at most 328 steps) to obtain a small number of candidates for the initial state of the frame. The weakness we exploit here is not the unavoidable fact that the branching process is critical, but the fact that due to the frequent reinitializations there is a very short distance from intermediate states to initial states.

Key idea 4: The key can be extracted from the initial state of any frame.

Here we exploit the weakness of the A5/1 key setup routine. Assume that we know the state of A5/1 immediately after the key and frame counter were used, and before the 100 mixing steps. Since the clock control is disabled before this stage, we can run the linear registers backwards in a unique way, and reverse the effect of the known frame counter. The result is a vector of linear forms in the key bits, from which we can extract the actual key via matrix multiplication by a precomputed 64×64 binary matrix. Since the previous step may suggest several starting states, we can choose the correct key by mixing it with the next frame counter, running A5/1 forward for more than 100 steps, and comparing the results with the actual data in the next frame.

Key idea 5: The Golic birthday attack on A5/1 is marginally impractical.

By the well known birthday paradox, A and B are likely to have a common state when their sizes a and b satisfy $a * b \approx n$. We would like a to be bounded by the size of commercially available PC hard disks, and b to be bounded by the number of overlapping prefixes in a typical GSM telephone conversation. Reasonable bounds on these values (justified later in this paper) are $a \approx 2^{35}$ and $b \approx 2^{22}$. Their product is 2^{57}, which is about 100 times smaller than $n = 2^{64}$. To make the intersection likely, we either have to increase the storage requirement from a hundred gigabytes to ten terabytes, or to increase the length of the conversation from two minutes to three hours. Neither approach seems to be practical, but the gap is not huge and a relatively modest improvement by two orders of magnitude is all we need to make it practical.

Key idea 6: Use biased birthday attacks.

One of the main ideas in the improved attack is to consider sets A and B which are not chosen with uniform probability distribution among all the possible states. Assume that each state s is chosen for A with probability $P_A(s)$, and is chosen for B with probability $P_B(s)$. If the means of these probability distributions are a/n and b/n, respectively, then the expected size of A is a, and the expected size of B is b.

The birthday threshold happens when $\sum_s P_A(s)P_B(s) \approx 1$. For independent uniform distributions, this evaluates to the standard condition $a * b \approx n$. However, when the probability distributions are correlated the results can be dramatically different: When A consists of the $n/2$ states that start with 0 and B consists of the $n/2$ states that start with one, the two sets are disjoint in spite of their size, while if all the probabilities are concentrated on one common string, the two sets intersect even though their sizes are 1.

In the new attack, we choose states for the disk and states in the data with two non-uniform probability distributions which have very strong positive correlation. This makes our time memory tradeoff much more efficient than the one used by Golic. This is made possible by the fact that in A5/1, the initial state of each new frame is rerandomized very frequently with different frame counters.

Key idea 7: Use special states.

A major problem in implementing time-memory tradeoff attacks is that access to disk is about a million time slower than a computational step, and thus it is crucial to minimize the number of times we look for data in the hard disk. An idea due to Ron Rivest (in the context of time memory tradeoffs of block ciphers) is to keep on the disk only special states which are guaranteed to produce output bits starting with a particular pattern α of length k. When we scan the data, we access the disk only when we encounter such a prefix. Clearly, we do not miss any intersections by skipping the other prefixes in the actual data, while reducing the number b of disk probes by a factor of about 2^k. The number of points a we have to memorize remains unchanged, since in the formula $a * b \approx n$ both b and n are reduced by the same factor 2^k. The downside is that we have to work 2^k times harder during the preprocessing stage, since only 2^{-k} of the random states we try produce outputs with such a k bit prefix. If we try to reduce the number of disk access steps in the time memory attack on A5/1 from 2^{22} to 100, we have to increase the preprocessing time by a factor of about 40,000, which makes it impractically long.

Key idea 8: Special states can be efficiently sampled in A5/1.

One of the specific weaknesses of A5/1 which we exploit in our attack is that we can easily generate all the states which produce output sequences that start with a particular k-bit pattern α with $k = 16$ without trying and discarding other states. This is due to a poor choice of the clocking taps, which makes the register bits that affect the clock control and the register bits that affect the output unrelated for about 16 clock cycles, so we can choose them independently. This easy access to special states does not happen in good block ciphers, but can happen in stream ciphers due to their simpler transition functions. In fact, the maximal value of k for which special states can be sampled without trial and error can serve as a new security measure for stream ciphers, which we call its **sampling resistance**. As demonstrated in this paper, high values of k can have a major impact on the efficiency of time-memory tradeoff attacks on such cryptosystems.

Key idea 9: A5/1 is very efficient on a PC.

The A5/1 algorithm was designed to be efficient in hardware, and its straightforward software implementation is quite slow. To execute the preprocessing stage, we have to run it on a distributed network of PC's up to 2^{48} times, and thus we need an extremely efficient way to compute the effect of one clock cycle on the three registers.

We exploit the following weakness in the design of A5/1: Each one of the three shift registers is so small that we can precompute all its possible states, and keep them in three cyclic arrays where successive locations in each array represent successive states of the corresponding shift register. In fact, we don't have to keep the full states in the arrays, since the only information we have to know about a state is its clocking tap and its output tap. A state can thus be viewed as a triplet of indices (i, j, k) into three large arrays whose entries are pairs of bits (see Figure 2). Since there is no mixing of the values of the three registers, their only interaction is in determining which of the three indices should be incremented by 1. This can be determined by a precomputed table with three input bits (the clocking taps) and three output bits (the increments of the three registers). When we clock A5/1 in our software implementation, we don't shift registers or compute feedbacks - we just add a 0/1 vector to the current triplet of indices. A typical two dimensional variant of such movement vectors in triplet space is described in Figure 3. Note the local tree structure determined by the deterministic forward evaluation and the nondeterministic backward exploration in this triplet representation.

Since the increment table is so small, we can expand it to a large precomputed table with 2^{24} entries, whose inputs are the three bytes to the right of the clocking taps in the three registers, and outputs are the three increments to the indices which allow us to jump directly to the state which is 8 clock cycles away. The total amount of RAM needed for the state arrays and precomputed movement tables is less than 128 MB, and the total cost of advancing the three registers for 8 clock cycles is one table lookup and three integer additions! A similar table lookup technique can be used to compute in a single step output bytes instead of output bits, and to speed up the process of running A5/1 backwards.

5 Detailed Description of the Attack

In this section we fill in the missing details, and analyse the success rate of the new attack.

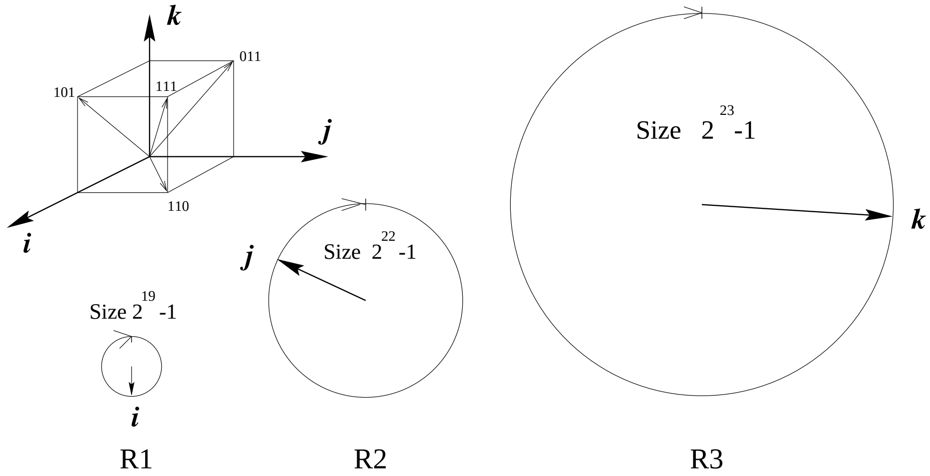

Figure 2: Triplet representation of a state.

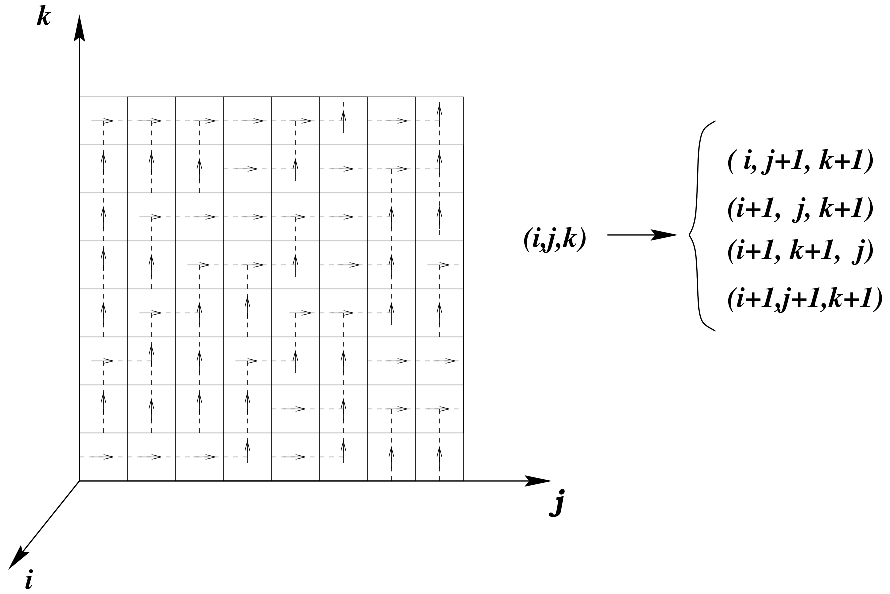

Figure 3: The state-transition graph in the triplet representation of A5/1.

5.1 Efficient Sampling of Special States

Let α be any 16 bit pattern of bits. To simplify the analysis, we prefer to use an α which does not coincide with shifted versions of itself (such as $\alpha = 1000...0$) since this makes it very unlikely that a single 228-bit frame contains more than one occurrence of α.

The total number of states which generate an output prefix of α is about $2^{64} * 2^{-16} = 2^{48}$. We would like to generate all of them in a (barely doable) 2^{48} preprocessing stage, without trying all the 2^{64} possible states and discarding the vast majority which fail the test. The low sampling resistance of A5/1 is made possible by several flaws in its design, which are exploited in the following algorithm:

- Pick an arbitrary 19-bit value for the shortest register $R1$. Pick arbitrary values for the rightmost 11 bits in $R2$ and $R3$ which will enter the clock control taps in the next few cycles. We can thus define $2^{19+11+11} = 2^{41}$ partial states.

- For each partial state we can uniquely determine the clock control of the three registers for the next few cycles, and thus determine the identity of the bits that enter their msb's and affect the output.

- Due to the majority clock control, at least one of $R2$ and $R3$ shifts a new (still unspecified) bit into its msb at each clock cycle, and thus we can make sure that the computed output bit has the desired value. Note that about half the time only one new bit is shifted (and then its choice is forced), and about half the time two new bits are shifted (and then we can choose them in two possible ways). We can keep this process alive without time consuming trial and error as long as the clock control taps contain only known bits whereas the output taps contain at least one unknown bit. A5/1 makes this very easy, by using a single clocking tap and placing it in the middle of each register: We can place in $R2$ and $R3$ 11 specified bits to the right of the clock control tap, and 11-12 unspecified bits to the right of the output tap. Since each register moves only 3/4 of the time, we can keep this process alive for about 16 clock cycles, as desired.

- This process generates only special states, and cannot miss any special state (if we start the process with its partial specification, we cannot get into an early contradiction). We can similarly generate any number $c < 2^{48}$ of randomly chosen special states in time proportional to c. As explained later in the paper, this can make the preprocessing faster, at the expense of other parameters in our attack.

5.2 Efficient Disk Probing

To leave room for a sufficiently long identifying prefix of 35 bits after the 16-bit α, we allow it to start only at bit positions 1 to 177 in each one of the given frames (i.e., at a distance of 101 to 277 from the initial state). The expected number of occurrences of α in the data produced by A5/1 during a two minute conversation is thus $2^{-16} * 177 * 120 * 1000/4.6 \approx 71$. This is the expected number of times b we access the hard disk. Since each random access takes about 6 milliseconds, the total disk access time becomes negligible (about 0.4 seconds).

5.3 Efficient Disk Storage

The data items we store on the disk are (prefix, state) pairs. The state of A5/1 contains 64 bits, but we keep only special states and thus we can easily encode them in 48 bits, by specifying all the decisions made during the efficient sampling procedure. We can further reduce the state to less than 40 bits (5 bytes) by leaving some of these bits unspecified. This saves a considerable fraction of the disk space prepared during preprocessing, and the only penalty is that we have to try a small number of candidate states instead of one candidate state for each one of the 71 relevant frames. Since this part is so fast, even in its slowed down version it takes less than a second.

The prefix produced from each special state is nominally of length 16+35=51 bits. However, the first 16 bits are always the constant α, and the next 35 bits are stored in sorted order on the disk. We can thus store the full value of these 35 bits only once per sector, and encode on the disk only their small increments (with a default value of 1). Other possible implementations are to use the top parts of the prefixes as direct sector addresses or as file names. With these optimizations, we can store each one of the sorted (prefix, state) pairs in just 5 bytes. The largest commercially available PC hard disks (such as IBM Ultrastar 72 ZX or Seagate Cheetah 73) have 73 gigabytes. By using two such disks, we can store $146 * 2^{30}/5 \approx 2^{35}$ pairs during the preprocessing stage, and characterize each one of them by its (usually unique) 35-bit prefix.

5.4 Efficient Tree Exploration

The forward state-transition function of A5/1 is deterministic, but in the reverse direction we have to consider four possible predecessors. About 3/8 of the states have no predecessors, 13/32 of the states have one predecessor, 3/32 of the states have two predecessors, 3/32 of the states have three predecessors, and 1/32 of the states have four predecessors.

Since the average number of predecessors is 1, Golic assumed that the critical branching process will provide a good model for the generated trees of predecessors. We were surprised to discover that in the case of A5/1, there was a very

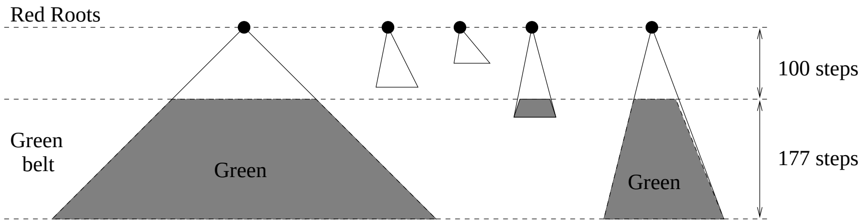

Figure 4: Trees of different sizes.

significant difference between the predictions of the model and our experimental data. For example, the theory predicted that only 2% of the states would have some predecessor at depth 100, whereas in a large sample of 100,000,000 trees we generated from random A5/1 states the percentage was close to 15%. Another major difference was found in the tail distributions of the number of sons at depth 100: Theory predicted that in our sample we should see some cases with close to 1000 sons, whereas in our sample we never saw trees with more than 120 sons at depth 100.

5.5 The Biased Birthday Attack.

To analyse the performance of our biased birthday attack, we introduce the following notation:

Definition 1 *A state s is coloured* **red**, *if the sequence of output bits produced from state s starts with α (i.e., it is a special state). The subspace of all the red states is denoted by R.*

Definition 2 *A state is coloured* **green**, *if the sequence of output bits produced from state s contains an occurrence of α which starts somewhere between bit positions 101 and 277. The subspace of all the green states is denoted by G.*

The red states are the states that we keep in the disk, look for in the data, and try to collide by comparing their prefixes. The green states are all the states that could serve as initial states in frames that contain α. Non-green initial states are of no interest to us, since we discard the frames they generate from the actual data.

The size of R is approximately 2^{48}, since there are 2^{64} possible states, and the probability that α occurs right at the beginning of the output sequence is 2^{-16}. Since the redness of a state is not directly related to its separate coordinates i, j, k in the triplet space, the red states can be viewed as randomly and sparsely located in this representation. The size of G is approximately $177 * 2^{-48}$ (which is still a small fraction of the state space) since α has 177 opportunities to occur along the output sequence.

Since a short path of length 277 in the output sequence is very unlikely to contain two occurrences of α, the relationship between green and red states is essentially many to one: The set of all the relevant states we consider can be viewed as a collection of disjoint trees of various sizes, where each tree has a red state as its root and a "belt" of green states at levels 101 to 277 below it (see Figure 4). The weight $W(s)$ of a tree whose root is the red state s is defined as the number of green states in its belt, and s is called k-heavy if $W(s) \geq k$.

The crucial observation which makes our biased birthday attack efficient is that in A5/1 there is a huge variance in the weights of the various red states. We ran the tree exploration algorithm on 100,000,000 random states and computed their weights. We found out that the weight of about 85% of the states was zero, because their trees died out before reaching depth 100. Other weights ranged all the way from 1 to more than 26,000.

Figure 5 describes for each x which is a multiple of 100 the value y which is the total weight of all the trees whose weights were between x and $x+100$. The total area under the graph to the right of $x = k$ represents the total number of green states in all the k-heavy trees in our sample.

The initial mixing of the key and frame number, which ignores the usual clock control and flips the least significant bits of the registers about half the time before shifting them, can be viewed as random jumps with uniform probability distribution into new initial states: even consecutive frame counters can lead to far away initial states in the triplet space. When we restrict our attention to the frames that contain α, we get a uniform probability distribution over the green states, since only green states can serve as initial states in such frames.

The red states, on the other hand, are not encountered with uniform probability distribution in the actual data. For example, a red state whose tree has no green belt will never be seen in the data. On the other hand, a red state with a huge green belt has a huge number of chances to be reached when the green initial state is chosen with uniform probability distribution. In fact the probability of encountering a particular red state s in a particular frame which is known to contain α is the ratio of its weight $W(s)$ and the total number of green states $177 * 2^{48}$, and the probability of encountering it in one of the 71 relevant frames is $P_B(s) = 71 * W(s)/(177 * 2^{48})$.

Since $P_B(s)$ has a huge variance, we can maximize the expected number of collisions $\sum_s P_A(s) * P_B(s)$ by choosing red points for the hard disk not with uniform probability distribution, but with a biased probability $P_A(s)$ which maximizes the correlation between these distributions, while minimizing the expected size of A. The best way to do this is to keep on the disk only the heaviest trees. In other words, we choose a threshold number k, and define $P_A(s) = 0$ if $W(s) < k$, and $P_A(s) = 1$ if $W(s) \geq k$. We can now easily compute the expected number of collisions by the formula:

$$\sum_s P_A(s) * P_B(s) = \sum_{s|W(s) \geq k} 71 * W(s)/(177 * 2^{48})$$

which is just the number of red states we keep on the disk, times the average weight of their trees, times $71/(177 * 2^{48})$.

In our actual attack, we keep 2^{35} red states on the disk. This is a 2^{-13} fraction of the 2^{48} red states. With such a tiny fraction, we can choose particularly heavy trees with an average weight of 12,500. The expected number of colliding red states in the disk and the actual data is $2^{35} * 12,500 * 71/(177 * 2^{48}) \approx 0.61$. This expected value makes it quite likely that a collision will actually exist. [2]

The intuition behind the biased time memory tradeoff attack is very simple. We store red states, but what we really want to collide are the green states in their belts (which are accessible from the red roots by an easy computation). The 71 green states in the actual data are uniformly distributed, and thus we want to cover about 1% of the green area under the curve in Figure 5. Standard time memory tradeoff attacks store random red states, but each stored state increases the coverage by just 177 green states on average. With our optimized choice in the preprocessing stage, each stored state increases the coverage by 12,500 green states on average, which improves the efficiency of the attack by almost two orders of magnitude.

5.6 Efficient Determination of Initial States

One possible disadvantage of storing heavy trees is that once we find a collision, we have to try a large number of candidate states in the green belt of the colliding red state. Since each green state is only partially specified in our compact 5-byte representation, the total number of candidate green states can be hundreds of thousands, and the real time part of the attack can be relatively slow.

However, this simple estimate is misleading. The parasitic red states obtained from the partial specification can be quickly discarded by evaluating their outputs beyond the guaranteed occurrence of α and comparing it to the bits in the given frame. In addition, we know the exact location of α in this frame, and thus we know the exact depth of the initial state we are interested in within the green belt. As a result, we have to try only about 70 states in a cut through the green belt, and not the 12,500 states in the full belt.

5.7 Reducing the Time Complexity of Preprocessing

The 2^{48} complexity of the preprocessing stage can make it too time consuming for a small network of PC's. In this section we show how to reduce this

[2] Note that in time memory tradeoff attacks, it becomes increasingly expensive to push this probability towards 1, since the only way to guarantee success is to memorize the whole state space.

complexity by any factor of up to 1000, by slightly increasing either the space complexity or the length of the attacked conversation.

The efficient sampling procedure makes it possible to generate any number $c < 2^{48}$ of random red states in time proportional to c. To store the same number of states in the disk, we have to choose a larger fraction of the tested trees, which have a lower average weight, and thus a less efficient coverage of the green states. Table 1 describes the average weight of the heaviest trees for various fractions of the red states, which was experimentally derived from our sample of 100,000,000 A5/1 trees. The implied tradeoff is very favorable: If we increase the fraction from 2^{-13} to 2^{-7}, we can reduce the preprocessing time by a factor of 64 (from 2^{48} to 2^{42}), and compensate by either doubling the length of the attacked conversation from 2 minutes to 4 minutes, or doubling the number of hard disks from 2 to 4. The extreme point in this tradeoff is to store in the disk all the sampled red states with nonzero weights (the other sampled red states are just a waste of space, since they will never be seen in the actual data). In A5/1 about 15% of the red states have nonzero weights, and thus we have to sample about 2^{38} red states in the preprocessing stage in order to find the 15% among them (about 2^{35} states) which we want to store, with an average tree weight of 1180. To keep the same probability of success, we have to attack conversations which last about half an hour.

A further reduction in the complexity of the preprocessing stage can be obtained by the early abort strategy: Explore each red state to a shallow depth, and continue to explore only the most promising candidates which have a large number of sons at that depth. This heuristic does not guarantee the existence of a large belt, but there is a clear correlation between these events.

To check whether the efficiency of our biased birthday attack depends on the details of the stream cipher, we ran several experiments with modified variants of A5/1. In particular, we concentrated on the effect of the clock control rule, which determines the noninvertibility of the model. For example, we hashed the full state of the three registers and used the result to choose among the four possible majority-like movements (+1,+1,+1), (+1,+1,0), (+1,0,+1), (0,+1,+1) in the triplet space. The results were very different from the real majority rule. We then replaced the majority rule by a minority rule (if all the clocking taps agree, all the registers move, otherwise only the minority register moves). The results were very similar to the majority-like hashing case, and very different from the real majority case (see Figure 6). It turns out that in this sense A5/1 is actually stronger than its modified versions, but we do not currently understand the reason for this strikingly different behavior. We believe that the type of data in Table 1, which we call the **tail coverage** of the cryptosystem, can serve as a new security measure for stream ciphers with noninvertible state transition functions.

Acknowledgements: We would like to thank Ross Anderson, Mike Roe,

Average Weights							
2^{-4}	2432	2^{-5}	3624	2^{-6}	4719	2^{-7}	5813
2^{-8}	6910	2^{-9}	7991	2^{-10}	9181	2^{-11}	10277
2^{-12}	11369	2^{-13}	12456	2^{-14}	13471	2^{-15}	14581
2^{-16}	15686	2^{-17}	16839	2^{-18}	17925	2^{-19}	19012
2^{-20}	20152	2^{-21}	21227	2^{-22}	22209	2^{-23}	23515
2^{-24}	24597	2^{-25}	25690	2^{-26}	26234		

Table 1: The average weight of the heaviest trees for various fractions of R.

Jovan Golic, Marc Briceno, Ian Goldberg, and David Wagner for their pioneering contributions to the analysis of A5/1, which made this paper possible.

References

[1] R. Anderson, M. Roe, *A5*, `http://jya.com/crack-a5.htm`, 1994.

[2] M. Briceno, I. Goldberg, D. Wagner, A pedagogical implementation of A5/1, `http://www.scard.org`, May 1999.

[3] J. Golic, *Cryptanalysis of Alleged A5 Stream Cipher*, proceedings of EUROCRYPT'97, LNCS 1233,pp.239–255, Springer-Verlag 1997.

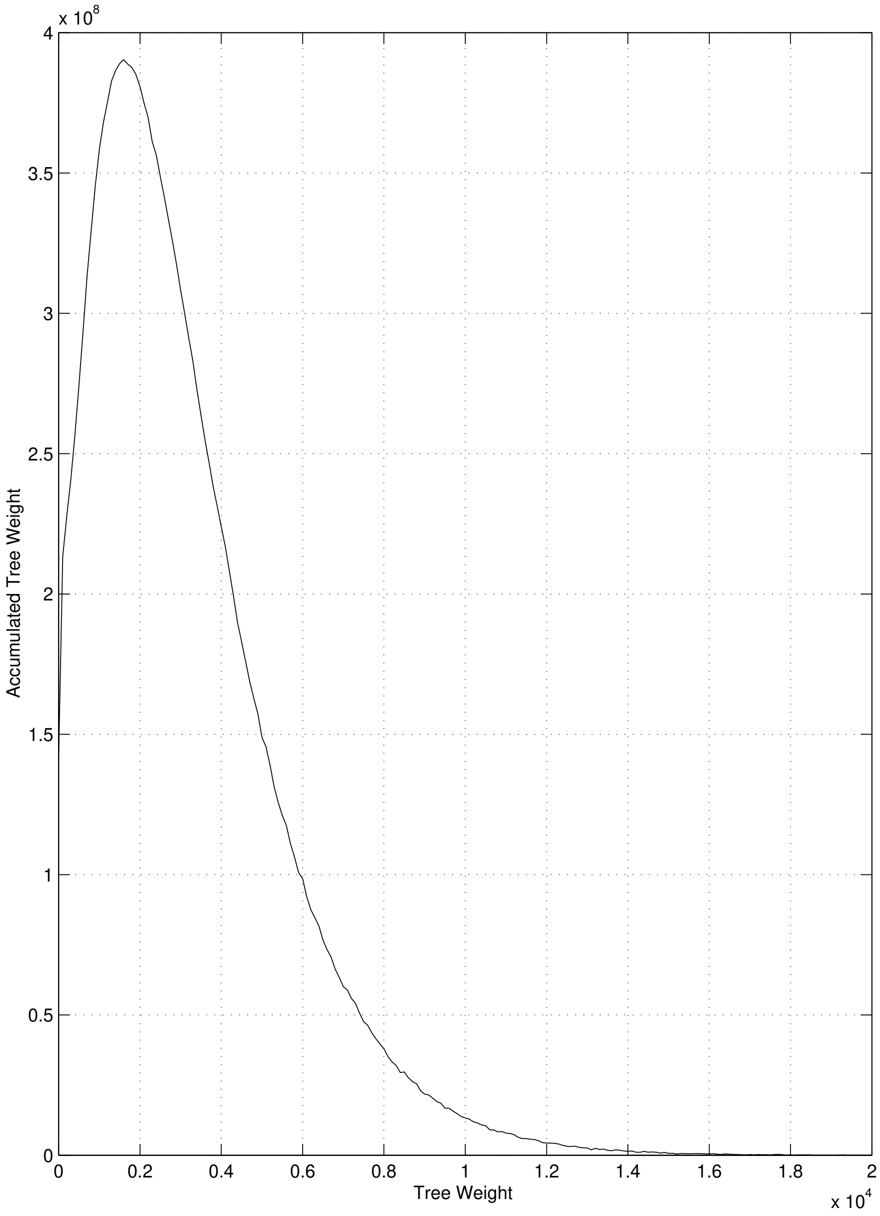

Figure 5: Weight distribution for the majority function.

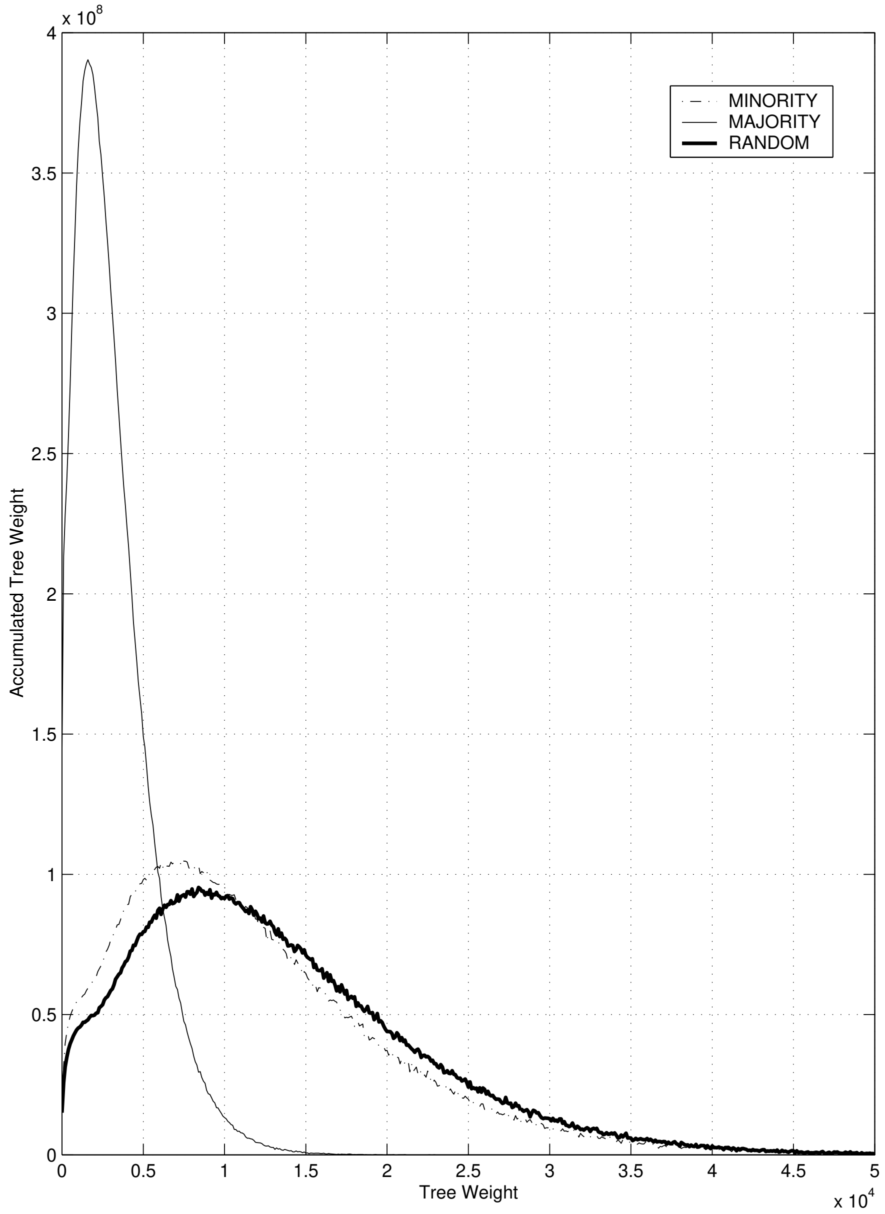

Figure 6: Comparison of the weight distributions for different clock-control functions.