A structural analysis of the A5/1 state transition graph*

Andreas Beckmann
Goethe-Universität Frankfurt
Institut für Informatik
Frankfurt am Main, Germany
beckmann@cs.uni-frankfurt.de

Jaroslaw Fedorowicz
Goethe-Universität Frankfurt
Institut für Informatik
Frankfurt am Main, Germany
fedorow@cs.uni-frankfurt.de

Jörg Keller
FernUniversität in Hagen
Fakultät für Mathematik und Informatik
Hagen, Germany
joerg.keller@fernuni-hagen.de

Ulrich Meyer
Goethe-Universität Frankfurt
Institut für Informatik
Frankfurt am Main, Germany
umeyer@cs.uni-frankfurt.de

We describe efficient algorithms to analyze the cycle structure of the graph induced by the state transition function of the A5/1 stream cipher used in GSM mobile phones and report on the results of the implementation. The analysis is performed in five steps utilizing HPC clusters, GPGPU and external memory computation. A great reduction of this huge state transition graph of 2^{64} nodes is achieved by focusing on special nodes in the first step and removing leaf nodes that can be detected with limited effort in the second step. This step does not break the overall structure of the graph and keeps at least one node on every cycle. In the third step the nodes of the reduced graph are connected by weighted edges. Since the number of nodes is still huge an efficient bitslice approach is presented that is implemented with NVIDIA's CUDA framework and executed on several GPUs concurrently. An external memory algorithm based on the STXXL library and its parallel pipelining feature further reduces the graph in the fourth step. The result is a graph containing only cycles that can be further analyzed in internal memory to count the number and size of the cycles. This full analysis which previously would take months can now be completed within a few days and allows to present structural results for the full graph for the first time. The structure of the A5/1 graph deviates notably from the theoretical results for random mappings.

1 Introduction

GSM (Global System for Mobile Communications) is the set of standards for cellular networks of the second generation and dates back to 1990. Even though the fourth generation of mobile communication standards was introduced in 2010, more than 75% of all mobile connections that were established worldwide at the end of 2010 were using GSM [9]. The security of GSM-based communication and the underlying A5 stream cipher family has been studied by several groups, see e.g. [5, 8, 13].

We present an efficient method to analyze the cycle structure of the directed graph $G = (V,E)$ induced by the A5/1 stream cipher, which serves to encrypt speech data over the air between GSM mobile phones and the base station. Each state of this stream cipher can be described using 64 bits and the set of states corresponds to the nodes V. A state transition function $f : V \mapsto V$ can be derived and the edges can be computed as $E = \{(x, f(x)) : x \in V\}$. Each node of graph G has an outdegree of one, so that $|V| = |E| = 2^{64}$ and the graph consists of one or more weakly connected components (WCC). Each WCC contains one cycle with root directed trees attached to the cycle and the root node being part of the

*Partially supported by the DFG grant ME 3250/1-3, and by MADALGO – Center for Massive Data Algorithmics, a center of the Danish National Research Foundation.

A.J. Wijs, D. Bošnački, S. Edelkamp (Eds.): GRAPH
Inspection and Traversal Engineering (GRAPHITE 2012)
EPTCS 99, 2012, pp. 5–19, doi:10.4204/EPTCS.99.4

cycle. The cycle structure (number and sizes of cycles and WCCs) can be compared to expected values known for random mappings, e.g. $\Theta(\sqrt{n})$ nodes are expected to be on cycles in a random mapping of size n. Notable deviations from these expectations, in particular cycles shorter than the already small expectation, may indicate a weakness of the cipher because these non-random structures might enable various types of attacks.

The size of the graph prevents an explicit construction in internal or external memory, thus excluding algorithms that modify the original graph structure during a reduction, and calls for specific algorithmic solutions to reduce the data size to a feasible level. In contrast to prior work, our approach takes days instead of months by tailoring the hardware and algorithm for each solution step, and derives much more information about the graph than only the cycle lengths.

Our new findings complete and confirm previous partial results and estimated properties of the A5/1 graph. The techniques used for the A5/1 graph may be adjusted to different random mappings, e.g. hash chains [12].

The remainder of this work is organized as follows: in Section 2 we give a short description of the A5/1 stream cipher in general and derive some important properties. Related work on this topic is summarized in Section 3. In Section 4 the five steps to retrieve the structure information of the A5/1 state transition graph are described. Details on the implementation of the algorithms are presented in Section 5 and the results are discussed in Section 6. We conclude in Section 7.

2 The A5/1 stream cipher

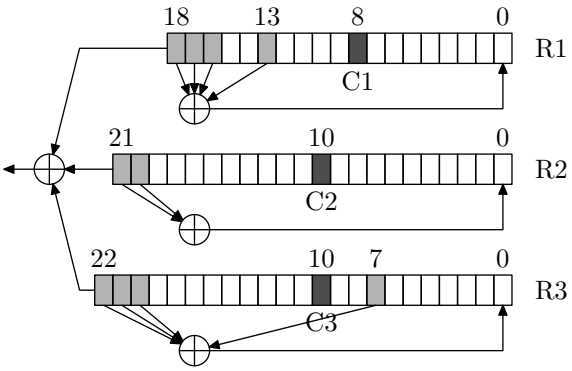

Figure 1: The linear feedback shift registers of the A5/1 stream cipher

In the following we provide a description of the internal state of the A5/1 stream cipher. Its application for en-/decrypting GSM communication is described in [4].

The A5/1 stream cipher consists of three different irregularly clocked linear feedback shift registers (LFSRs) that are combined via a clock control as shown in Figure 1. Whenever a register is clocked, the feedback bits (e.g. 13, 16, 17, and 18 for $R1$) are XORed and inserted into bit 0 after a left shift. The feedback taps of the three LFSRs in the A5/1 stream cipher were chosen in a way that the registers have maximum length periods, i.e. all other possible states of a register will be generated before a state will be generated for the second time. The all-zero value of a register is not allowed as it would not lead to a different state, therefore register $R1$ has $2^{19}-1$, $R2$ has $2^{22}-1$ and $R3$ has $2^{23}-1$ valid and reachable states.

To determine which register is clocked in each iteration of the A5/1 stream cipher, each register has a bit position marked as the clock tap (C1, C2 and C3) and a majority clock function takes these three bits as arguments. A register is clocked if its clock bit equals the majority value of the three clock bits. That means that either two or all three registers are clocked at the same time in each iteration.

The values of the three clock bits form eight different combinations. For each clock bit there are two combinations where this bit differs from the other two causing it not to be clocked. Therefore a single register is clocked in three out of four cases.

The 64 bit value obtained by concatenating the bits of the three LFSRs describes a single state x of the A5/1 stream cipher. Clocking the registers according to the majority clocking rule leads to the follow-up state $f(x)$. Note that the state transition function can be inverted easily so that backward clocking of the stream cipher is possible and produces up to four different predecessors states.

Since the A5/1 graph is based on LFSRs, the following observations have been made. A cycle in the A5/1 generator implies that all three LFSRs have been cycled at least once. It is not possible to generate a cycle while keeping one register unclocked. Therefore it is possible to restrict observation to a fixed value in one register. Each cycle in the graph will contain at least one state with this fixed register value. Since following the state transition function from an arbitrary state finally leads to a cycle, it also leads to a state with the fixed value.

The minimum size L of a cycle in the state transition graph can be calculated as follows: Since the LFSRs in A5/1 stream cipher have maximum length periods, each cycle contains nodes that are built from all valid states of every register, especially the $2^{23}-1$ valid states of the largest register $R3$. A single register is expected to be clocked in three out of four cases therefore the whole stream cipher must be clocked $L = \frac{4}{3}(2^{23}-1) = 11184809.\overline{3}$ times on average to see all values of $R3$. The average R3 cycle length L thus bounds the minimum cycle length, and other cycle lengths are multiples of L. On a path from an arbitrary state to the cycle the fixed register value will appear regularly (every L steps on average).

3 Related work

This work is based on previous work by Beckmann and Keller [2]. They described the algorithm and method that is able to fully analyze the structure of the A5/1 Graph.

One evaluation of the state transition function of the A5/1 stream cipher maps each state to another state in the same state space. In the literature these mappings are often called *random mappings*, because it is assumed that the mapping function maps a state to another random state. Most random mapping graphs are derived from hash functions, pseudo random number generators (PRNGs) or stream ciphers like the A5/1. Flajolet and Odlyzko [7] studied several average case properties of random mapping graphs.

Other expected properties of the graph produced by the A5/1 stream cipher were described by Golic [8]. He analyzed the branching process of the root directed trees that are attached to the cycles by the root of each tree. The branching process of the trees can be modeled by a Galton-Watson branching process with the expected number of offsprings $\mu = 1$ and a related variance of $\sigma^2 = \frac{9}{8}$. A Galton-Watson process with an expected number of one child in each generation is called critical because the probability that the generated tree is limited in size equals one.

The properties of the model above only hold under the assumption that the branching probability of each node in the sub-tree is independent of the branching probability of the nodes in the same or in the preceding levels. This is not guaranteed in the special case of the A5/1 graph because there is a weak

dependency between these probabilities that might influence the expected properties. We will compare these theoretical values with our observed values in Section 6.

Keller [10] provides first details on the number of components and the length of the largest cycle which leads to the result that the graph produced by the A5/1 stream cipher looks non-random. Later the number of components was further refined in [2].

Exact results have not been published yet, so that our work is the first to describe the exact number of components together with other details.

4 The Algorithms

To completely analyze all cycles and their properties in the A5/1 state transition graph a five step approach is described in this section. The first step reduces the state transition graph of size 2^{64} to a *candidate graph* of about 2^{40} nodes. Candidates must satisfy these three constraints:

$$x \text{ is candidate} \iff x.R3 = fixedR3 \quad (1)$$

$$\wedge \quad f(x).R3 \neq fixedR3 \quad (2)$$

$$\wedge \quad f^{-1}(x) \neq \varnothing \quad (3)$$

The $R3$ register of a candidate must equal an arbitrary predefined value $fixedR3$ (1). The $R3$ register of the following state must differ from the $fixedR3$ value (2). Leaves in the state transition graph are not considered as candidates (3). An edge (x,x') in the candidate graph corresponds to the unique path from candidate x to candidate x' in the state transition graph, and is labeled with the length of that path. Thus, in the candidate graph, nodes have an outdegree of 1 as in the state transition graph. The distribution of indegrees however will be different in both graphs. The candidate graph has the property being much smaller than the state transition graph G, but that each cycle of G is still present in the candidate graph, because each cycle of G contains at least one candidate. In general, when this property cannot be guaranteed, a random placement of candidates can be used to achieve the property with high probability, at least for larger cycles.

Removing the outgoing edges of all candidate nodes in the state transition graph G partitions G into *segments*. Each segment is a tree with the root being a candidate node. A *shallow segment* is defined as a segment that does not exceed a specific depth, the depth D being defined as the maximum distance of any node in the segment to the segment's root. The selection of that depth is a tuning parameter for the second step resulting in a time/space tradeoff. The candidate node of a shallow segment must be a leaf in the candidate graph, as long as the specified depth D is smaller than the minimum distance L between candidate nodes in the original graph, which is the case for the A5/1 graph. In general, the candidate of a shallow segment is a leaf in the candidate graph only with high probability.

The second step performs for each candidate node a Depth First Search (DFS) in the state transition graph with reverse directed edges by following f^{-1} to identify and remove shallow segments. The remaining candidate nodes represent the *skeleton graph* of the state transition graph.

The third step connects the nodes of the skeleton graph with weighted edges by following f and thus constructs the graph.

The fourth step cuts the leaves of the skeleton graph iteratively until only cycles remain. A fifth step calculates the number and size of the cycles in internal memory.

Further investigation is performed on the intermediate results to recover the information discarded in the first reduction step and to analyze some other properties of the A5/1 graph.

4.1 First step: selection of candidate nodes

The definition of a candidate reduces the state transition graph with 2^{64} nodes to a candidate graph with about 2^{40} nodes.

Restricting observation to nodes with an arbitrary $R3$ value $fixedR3$ as defined in Equation (1) results in $(2^{19}-1)\cdot(2^{22}-1)\approx 2^{41}$ nodes. It is also possible to fix an arbitrary $R1$ or $R2$ value, but fixing a value of the biggest register $R3$ instead leads to a smaller set of nodes and thus to a better reduction.

The direct predecessor node of a candidate could also have a $R3$ value of $fixedR3$ but different values in both registers $R1$ and $R2$. That implies that there exist chains of nodes with the same $R3$ value $fixedR3$. Equation (2) defines a candidate to be the last element in the chain and thus eliminates 2^{39} candidates, calculated as follows:

Due to the restriction of the all-zero value for each register, the number of chains depends on the value of the clocking bit c_3 of the predefined $R3$ value $fixedR3$. By definition in Equation (2) the $R3$ register of a candidate will be clocked in the next clock of the stream cipher. Multiplying the conditional probability $\Pr(R3 \text{ is clocked}|c_3)$ with the fixed number of nodes $SR3$ having the same $R3$ value $fixedR3$ gives us the expected number of chains. Let $c_i \in \{0,1\}$ be the value of the clocking bit of register i.

$$\begin{aligned}
&\mathrm{E}(\#\text{Chains}|c_3) = \Pr(R3 \text{ is clocked}|c_3)\cdot SR3\\
&= [1-\Pr(R3 \text{ is not clocked}|c_3)]\cdot SR3\\
&= [1-\Pr(c_1 = c_2 \neq c_3|c_3)]\cdot SR3\\
&= [1-\Pr(c_1 \neq c_3|c_3)\cdot\Pr(c_2 \neq c_3|c_3)]\cdot SR3\\
&= [1-\frac{2^{18}-c_3}{2^{19}-1}\cdot\frac{2^{21}-c_3}{2^{22}-1}]\cdot(2^{19}-1)(2^{22}-1)\\
&= (2^{19}-1)(2^{22}-1)-(2^{18}-c_3)(2^{21}-c_3) \qquad (4)
\end{aligned}$$

It follows that choosing a $fixedR3$ value with $c_3 = 0$ results in $\mathrm{E}(\#\text{Chains}|c_3=1)-\mathrm{E}(\#\text{Chains}|c_3=0) = 2359295$ (0.00014%) fewer chains. Therefore Equation (2) leads to a reduction of the number of candidates by 2^{39} ($\approx 25\%$) for $c_3 = 0$ but only by $2^{39}-2359295$ for $c_3 = 1$.

For completeness, the expected length of a chain conditioned on c_3 is quite close to ≈ 1.333 nodes for both values of c_3 and can be calculated as follows:

$$\begin{aligned}
\mathrm{E}(\text{chain length}|c_3) &= \frac{SR3}{\mathrm{E}(\#\text{Chains}|c_3)}\\
&= \frac{(2^{19}-1)(2^{22}-1)}{(2^{19}-1)(2^{22}-1)-(2^{18}-c_3)(2^{21}-c_3)}
\end{aligned}$$

Another reduction of the candidate nodes is accomplished by Equation (3). According to Golic [8] the probability of an arbitrary node having no predecessors is $\frac{3}{8}$. Keeping the two relevant bits of the $R3$ register fixed reduces the probability to $\frac{1}{4}$. Thus another reduction of $(2^{19}-1)\cdot(2^{22}-1)\cdot\frac{1}{4}\approx 2^{39}$ candidate nodes is achieved and results in a total number of approximately 2^{40} candidate nodes.

4.2 Second step: backward clocking

The second step removes the candidate nodes rooting shallow segments from the candidate graph by performing a depth restricted reverse DFS starting in each of the 2^{40} candidate nodes, i.e. it removes leaves from the candidate graph that can be identified with restricted effort, to further reduce the graph

size. See Section 6 for experimental results of the time/space tradeoff on the maximum depth of the reverse traversal described earlier.

The state transition function f of the A5/1 stream cipher can be easily reverted to f^{-1}, so that the predecessors nodes of a node can be computed during the traversal. As a result of the majority clock function of the A5/1 stream cipher every node can have zero or up to four direct predecessors. Recall that the number of predecessors of a candidate node is between one and four, due to the restriction in Equation (3).

The number of predecessors depends on the clock bit and on the next higher bit of each register. The 64 possible combinations of these six bits are covered by six rules introduced by Golic [8] and allow to precompute look-up tables.

The 2^{40} DFS traversals from the candidate nodes can be executed in parallel on multiple processors since the traversals are performed on different segments and thus on disjoint sets of nodes. Identifying the shallow segments and removing the associated candidate nodes results in a subset of the candidate nodes that represents the nodes of the skeleton graph.

Let N be the number of candidates, i.e. the number of nodes with properties (1) to (3). The complexity of step 2 is $O(N \cdot D)$, as the average number of predecessors for nodes in the state transition graph is 1, and thus the number of nodes expected in each level is a constant. The step can be perfectly parallelized.

4.3 Third step: forward clocking

In this forward clocking step the nodes of the skeleton graph get connected by weighted edges. The weight of an edge is the number of clocks of the A5/1 stream cipher needed to reach the destination node from the source node. The algorithm in Figure 2 clocks each node repeatedly until the next candidate node is reached.

The average number of clocks of the A5/1 stream cipher $L = 11184809.\overline{3}$ needed to reach a state with the same $R3$ was calculated in Section 2. Depending on the search depth of the second step, there might be still a lot of nodes left that needs to be clocked on average L times each.

The iterations of the loop can be parallelized as they are independent. The huge number of evaluations of the state transition function requires an efficient implementation to finish this task in a reasonable time. Therefore a GPGPU[1] bitslice approach that was used in [13] was adopted and executed on several GPUs in parallel.

Let N' be the number of nodes in the skeleton graph resulting from step 2. As skeleton nodes have distance L from each other on average, the complexity of step 3 is $O(N' \cdot L)$. It can be perfectly parallelized. If step 2 would have been skipped, step 3 would take $O(N \cdot L)$. Thus, in step 2 we must choose D such that $O(N \cdot D + N' \cdot L) < O(N \cdot L)$. This can be done by sampling a small number of segments.

4.4 Fourth step: cutting leaves

The resulting graph of the previous algorithm that is given as a list of edges is now going to be further reduced by an iterative leaf cutting algorithm, cf. Figure 3. Repeating this algorithm until no further reduction is gained will result in a graph containing only cycles. The sub-tree size and depth information is preserved and stored in the root of the removed sub-tree. A list B with entries *(source node, distance, destination node, sub-tree size, sub-tree depth)* is expected as input, where *sub-tree size* and *sub-tree depth* are initially set to zero. Note that the *source* entries in the input list B are unique, since each

[1] General Purpose computing on Graphics Processing Units

```
input list A of skeleton nodes

B = ∅
fixedR3 ← A[0].R3
for all a ∈ A do
    b.source ← a
    count ← 0
    repeat
        a ← clock(a)
        count ← count + 1
    until a.R3 = fixedR3
    while a is not candidate do
        a ← clock(a)
        count ← count + 1
    end while
    b.distance ← count
    b.destination ← a
    B = B ∪ {b}
end for

output list B of weighted edges
```

Figure 2: Forward clocking

```
input list B of edges with entries (source, distance,
destination, size, depth)

for all b ∈ B do
    if ∄x ∈ B: x.destination = b.source then
        B = B \ {b}
        find(y ∈ B : y.source = b.destination)
        y.size ← y.size + b.size + 1
        y.depth ← max(y.depth, b.depth)
    end if
end for

output reduced list B
```

Figure 3: Cutting leaves

node has only one outgoing edge. The input list B of edges is expected to be huge and won't fit into the internal memory of a single PC. An efficient external memory algorithm is presented that performs only sequential operations on the list. The parallel streaming pipeline of the STXXL[2] was used to implement this task. See further details on the implementation in Section 5.3.

4.5 Fifth step: count cycles

To count the cycles and to determine their sizes an internal algorithm, described in Figure 4, is applied on the remaining list of edges from the previous step. The expected number of nodes lying on a cycle should be small enough to fit in the internal memory of a single PC. Otherwise consider applying an efficient external cycle structure algorithm presented in [11].

```
input edge list B containing only cycles

visited ← ∅
for all b ∈ B do
    cycleSize ← 0
    leader ← ∞
    FINDCYCLE(b)
    if cycleSize ≠ 0 then
        output leader, cycleSize
    end if
end for

procedure FINDCYCLE(b)
    while b.source ∉ visited do
        visited ← visited ∪ {b.source}
        leader ← min(leader, b.source)
        cycleSize ← cycleSize + 1
        find(c ∈ B : c.source = b.destination)
        b ← c
    end while
end procedure
```

Figure 4: Counting cycle lengths

[2]Standard Template Library for Extra Large Data Sets, `http://stxxl.sourceforge.net`

5 Details on the implementation

5.1 Second step: backward clocking

The back clocking step is a crucial part of the whole task of finding the cycles in the A5/1 graph because the problem size of the following steps depends on the reduction factor in this step. On each of the approximately 2^{40} candidate nodes an efficient limited reverse traversal must be performed. Hybrid parallelism is applied for this task by utilizing several compute nodes of a HPC using MPI[3] (for communication between compute nodes) and OpenMP[4] (for fully utilizing a compute node from a single process).

The main problem in this step is to generate the predecessors of a node in an efficient way. Therefore a small (256 bytes) look-up table is precomputed and used to quickly identify which registers need to be clocked back to receive all valid predecessor nodes of the current node. The look-up table is indexed by a six bit number that is composed of the concatenation of the six relevant bits (clock taps and neighbor bits) of the three registers. For $fixedR3$ we used 0x2AAA00 (010101010101**0**1000000000 in binary representation).

Another optimization is applied to the calculation of the feedback bit of the registers with more than two feedback taps ($R1$ and $R3$). XORing several bits of a single 32-bit value can be realized by a single multiplication with a well chosen constant value.

A first node-limited (100000 nodes) BFS implementation was executed on a small cluster with 128 CPU cores. It took 92 hours to check all candidate nodes and with the chosen tuning parameter 98.77% of the candidate nodes were identified representing a shallow segment and thus removed. The same configuration was executed on the LOEWE-CSC[5] supercomputer utilizing 2048 cores. It took 12 hours to finish this task which is almost twice the expected time. This loss of efficiency arises from the static way of distributing the candidate nodes to the compute nodes. A perfect dynamic distribution would result in an expected total running time of about 6.5 hours, and will be implemented in a next version.

A depth-limited (3000 levels) DFS implementation was executed on the 128 CPU core cluster. It takes the same amount of time as the BFS implementation but leads to a better reduction of 98.96% shallow segments found.

5.2 Third step: forward clocking

Forward clocking of each node of the skeleton graph is a huge task that requires around 2^{57} evaluations of the clocking function of the A5/1 stream cipher. Because of the simplicity of this algorithm and its SIMD (Single Instruction Multiple Data) nature it is preferred to implement and to run that job on massive parallel special purpose hardware like GPUs.

Simplified, a CUDA-enabled GPU consists of multiple Streaming Multiprocessors (SM), each possessing thousands of registers, a small private cache and a SIMD processing unit that can process 32 threads simultaneously. All SMs have access to a large global memory. A CUDA application runs thousands of threads, partitioned into blocks that are mapped onto the SMs. Each block is then executed in groups of 32 threads, called warps. Due to the SIMD nature of the processing unit, each thread of a warp has to execute the same instruction (otherwise they are sequentialized). However, this restriction does not apply to different warps of the same block. Intra-block communication and synchronization

[3] Message Passing Interface

[4] `http://openmp.org`

[5] `http://csc.uni-frankfurt.de/index.php?id=51`

are fast and easily implemented using the SM's private cache, but inter-block communication has to be realized via the global memory and is very expensive. For more details on architecture and programming of these GPUs we refer to the CUDA C Programming Guide [14].

A first straightforward GPU implementation executed on a single NVIDIA Tesla C1060 GPU with 1.3 GHz clock frequency and was 85 times faster compared to a CPU implementation running on a single core with a clock frequency of 2.6 GHz. A different concept of clocking the A5/1-registers on the GPU is introduced to speed up the computation by another factor of 4.5. This speedup is achieved by implementing a bitslice approach [3] which has previously been adopted by the A5/1 Security Project [13].

Applying the bitslice approach requires an implementation of the state transition function using only bit operations like AND, OR, XOR, and NOT. The input data needs to be transposed, i.e. an array of 32 variables each containing a 64-bit state needs to be transformed into an array of 64 32-bit variables where variable v_i consists of the 'slice' of the i-th bit from all input values. Thereafter 32 evaluations of the function can be executed in parallel. The overhead of transposing the data in internal memory before and after the GPU calculation phase must be added to the total number of operations of the bitslice method.

A straightforward implementation of the bitslice approach by having each thread of a block clocking 32 nodes in parallel results in a bad performance due to the high amount of local data per thread. Therefore another approach is applied where each of the three warps of a single block operates on another register of the nodes. The 64 variables that store the 32 nodes are spread among the three warps (96 threads) of a block as follows:

- Thread 0-31 operates only on the R1 register consuming 19 variables of the data
- Thread 32-63 operates only on the R2 register consuming 22 variables of the data
- Thread 64-95 operates only on the R3 register consuming 23 variables of the data

The whole block of 96 threads operates on $32 \cdot 32 = 1024$ nodes in parallel. Intra-block sync operations guarantee that the critical part of exchanging the clocking bits between the threads through the private cache is performed in the correct order.

This algorithm is executed on NVIDIA Tesla C1060 and GTX 580 GPUs using the CUDA programming framework. A Tesla C1060 GPU (compute capability 1.3) is able to clock 3933 nodes/s each 11170000 times while the newer architecture GTX 580 (compute capability 2.0) is able to clock 9800 nodes/s. The speedup of NVIDIA's 2.0 generation mainly results from the higher amount of registers and the decrease of the warp allocation granularity, increasing the occupancy of the GPU from 38% (compute capability 1.3) to 50% on the newer architecture.

Due to performance reasons the GPU clocks each node a constant number of times (without checking whether a new candidate is reached) and the CPU finishes the clocking of each node while checking after each clock if the next candidate is reached. A number of 11170000 clocks is found to be safe for the particular predefined $fixedR3$ value used but might be to high for different $fixedR3$ values.

It takes nine days and three hours on five Tesla C1060 GPUs to finish the whole computation. Five GTX 580 GPUs would take 4 days and 8 hours to finish the whole computation. The reason why this implementation does not scale perfectly is because the CPU phase and GPU phase are not executed concurrently until now, but it will be implemented in a next version.

The resulting skeleton graph contains about $2^{33.6}$ nodes and consumes 252 GB of disk space.

5.3 Fourth step: removing leaves

The edge list generated in the previous step is too big to fit in the internal memory of a single PC. Therefore an external memory approach was implemented using the STXXL minimizing the number of

I/Os. An introduction to the STXXL library is presented in [6]. Details of external-memory algorithms are described in [15].

STXXL provides a streaming and pipelining feature so that the intermediate results do not need to be stored in temporary files, instead a pipeline can be implemented where each element of data is pushed through. To speedup the internal sorting and to overlap I/O and computation the parallel pipeline feature of the STXXL is used that is described in [1].

The edge list is sorted ascending according to the source node. The source nodes S are unique and the destination nodes D represent a subset of the source nodes, $D \subseteq S$. In each iteration all source nodes $S \setminus D$ are identified as leaves and removed.

The implemented iterative parallel pipeline is shown in Figure 5. The destination nodes of the edge list (1) are extracted in (2) and sorted ascending by (3) and (4). Stage (5) generates a unique destination node stream. At this point (*) in each iteration we can calculate the number of leaves that will be removed in the current iteration because the number of unique destination nodes equals the number of inner nodes. If the number of leaves is small due to the structure of the graph, we can decide to apply another external memory heuristic to analyze the structure of the cycles.

Stage (6) performs a parallel scan on both input streams and identifies each source node as leaf if this node does not appear in the unique destination node stream. Inner edges are pushed into a temporary file (7) and the destination node with size and depth information of a leaf are piped to (8).

The rest of the pipeline takes care that the successor node of each leaf that was removed in this iteration receives the information about the number of its removed predecessors. Therefore the destination nodes, size and depth of the removed edges in (6) are sorted in (8) ascending by the destination. Stage (9) removes double destination nodes while summing up the size and keeping the maximum depth of each element in a run of equal destination nodes.

In (10) the size and depth information of the removed leaves is attached to their according successor node that appears in the left input stream generated from (7). To prepare the next iteration all edges are pushed into (11) and sorted runs of the destination nodes are created in (12). In the next iteration of the loop stage (4) reads the output from (12) and (6) gets the edges streamed from (11). The loop terminates if the number of destination nodes in the stream in (*) equals the number of edges stored in (11).

If only cycles are left the graph is small enough to perform the last step of identifying and counting the size of each cycle in internal memory as described in Figure 4. If the resulting graph would be still large and not fit in the internal memory of a single PC consider an efficient external cycle structure algorithm presented in [11].

The computation time of the implementation is I/O bound, that means that its execution speed mainly depends on the performance of the underlying I/O system. Experiments with the data stored on a 4 hard disk RAID-0 system showed a total running time of 48 minutes while I/O was ongoing for 93 % of this time with an average throughput of 350 MB/s. The total running time of the implementation given the data stored on a 4 solid-state drive RAID-0 system was 40 minutes and I/O was active 70 % of the time with an average rate of 565 MB/s.

The implementation performs 88 iterations on the data which is also the maximum depth of a tree attached to a cycle in the skeleton graph. During the whole computation a factor of 2.8 times the size of the input data was read from disk and a factor of 0.8 was written to disk.

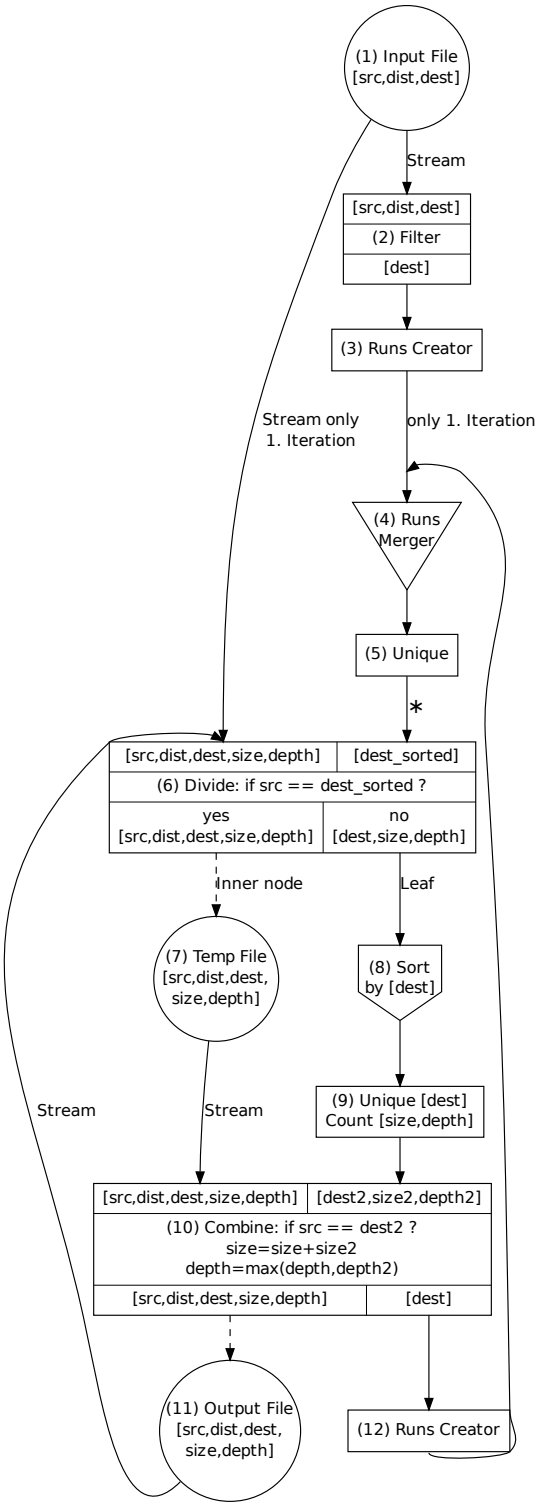

Figure 5: STXXL pipeline

cycle size	number of cycles
1	78745
2	17957
3	7214
4	3647
5	2138
6	1300
7	867
8	566
9	441
10	261
11	181
12	155
13	105
14	77
15	66
16	54
17	50
18	36
19	19
20	17
21	12
22	9
23	6
24	9
25	5
26	1
27	2
28	2
29	4
31	2
32	1
33	1
34	5
38	1
42	1
total	113957

Figure 6: Number of cycles of the A5/1 graph with same size (cycle sizes in multiples of L).

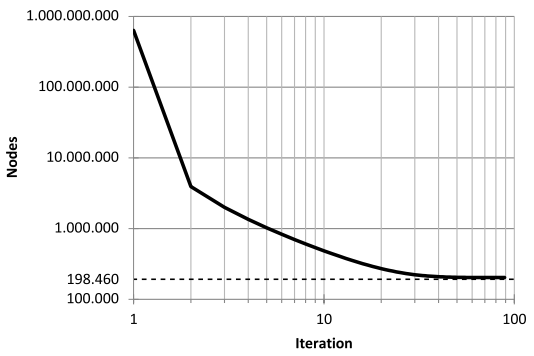

Figure 7: Reduction of nodes during the 88 iterations of the leaf removing algorithm

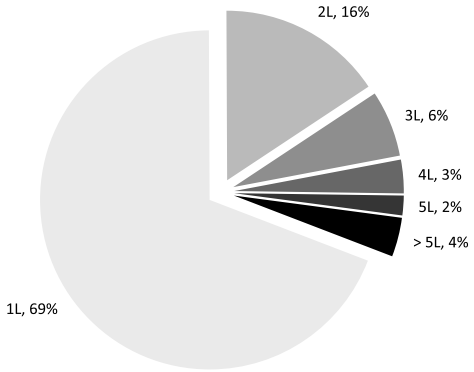

Figure 8: Distribution of the cycle sizes (indicated by xL) in the A5/1 graph

6 Results

6.1 Reduction of the skeleton graph

The reduction of the nodes of the skeleton graph during the leaf cutting algorithm of the fourth step takes 88 iterations of the loop until only cycles are left. The reduction in each iteration is presented in Figure 7. More than 99.38% of the nodes were identified as leaves in the first iteration.

6.2 Number and size of cycles and components

The total number of cycles is 113957. The size of a cycle is a multiple of the average expected minimum length L introduced in Chapter 2. Figure 6 lists all cycles with their specific cycle size as multiples of L. Figure 8 shows an overview of the cycle size distribution in total. It turns out that more than two thirds of the cycles have a length of just one L. Traversing along the cycles and counting the nodes revealed that $2,219,735,820,460 \approx 2^{41}$ nodes of the state transition graph's 2^{64} nodes are lying on a cycle.

6.3 Distribution of the minimum cycle length L

Since the minimum cycle length L deviates around the mean value of $11184809\frac{1}{3}$ it is interesting to see how the real minimum cycle lengths are distributed around the mean value. Therefore a random set of around 2^{33} nodes was generated and analyzed. From the sample set it turns out that the experimental expected segment size $\mu = 11184857$ is quite close to the expected L. The standard deviation of the sample set is around $\sigma = 1818$ leading to normal distribution plotted in Figure 9.

6.4 Number of nodes on depth level

The following data is generated by traversing the segments of a random sample set of 2^{26} nodes keeping the $R3$ register constant. For each node of the random sample set the number of nodes in each depth level was counted, the depth of a node being its distance from segment's root, i.e. candidate, in the state transition graph G. The average number of nodes on a specific depth level is shown in Figure 10. Note that the graph in Figure 10 does not take into account the probability of a traversal reaching that depth, instead it gives the average number of nodes expected to be found on a specific level if a traversal reaches

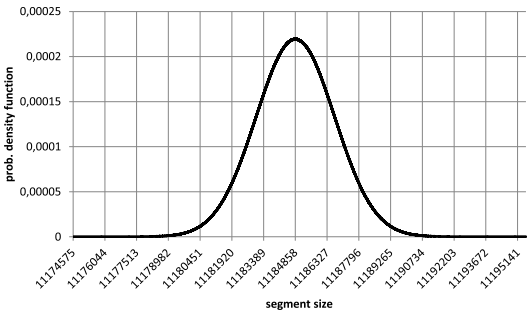

Figure 9: The estimated probability distribution of the minimum cycle length L

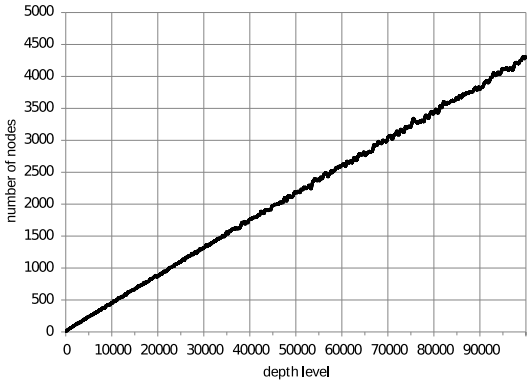

Figure 10: Expected number of nodes on a specific depth level

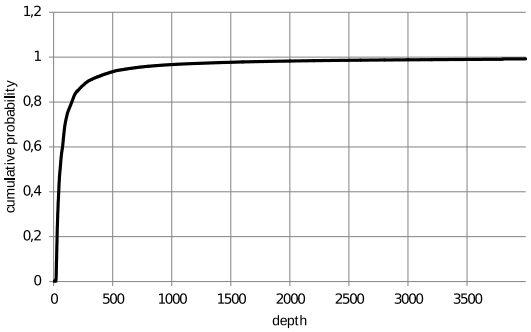

Figure 11: The cumulative distribution function of the segment depth

fraction of segments	maximum depth
50%	56
90%	319
99%	3539
99.9%	39065

Figure 12: Maximum depth of segments

that depth. Taking the probability of reaching that depth into account the number of expected nodes on a specific depth level is a constant of ≈ 1.7.

6.5 Segment depths

Another interesting fact is the expected maximum depth of a segment. There is a time/space tradeoff on the maximum search depth to be chosen in the backward clocking step. Therefore the maximum segment depth of a random sample set of around 2^{26} nodes is calculated and the cumulative distribution function of the result is plotted in Figure 11. Note that the $R3$ register of each node in that random sample set is kept constant as in the backward clocking algorithm. It turns out that 50% of the segments have a depth less than 56, other prominent values are shown in Figure 12.

6.6 Results compared to expected properties of random graph

In order to verify if the graph generated by the A5/1 stream cipher equals a graph generated by a random mapping, the results are compared to the the expected properties of random mappings in Table 1. The total number of nodes in the A5/1 graph is $n = (2^{19}-1)\cdot(2^{22}-1)\cdot(2^{23}-1)$. Exact values are given if observed.

The experimental results lead to the conclusion that the structure of the A5/1 graph is non-random.

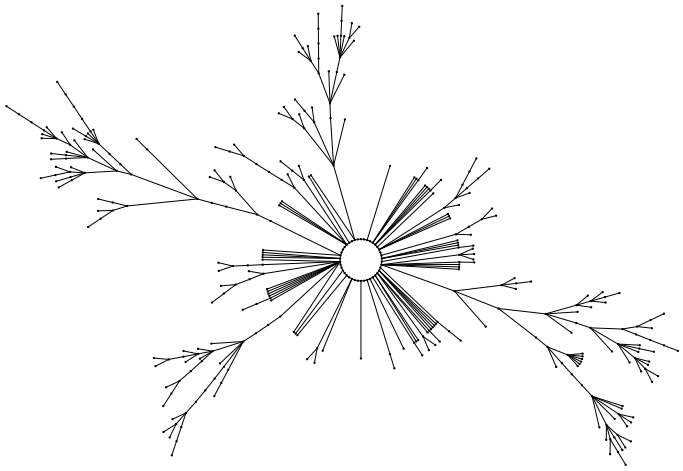

Figure 13: The skeleton graph of the WCC with the largest cycle of 42 segments after one iteration of leaf cutting

6.7 Size and structure of WCCs

The size of the skeleton graph of a WCC varies between four and 4194575 segments before applying any leaf cutting iteration. After one iteration of leaf cutting the sizes varies between 1 and 1210 segments. The skeleton graph of the WCC with the largest cycle of $42 \cdot L$ and a total number of 370 segments after one iteration of leaf cutting is presented in Figure 13.

7 Conclusion and further work

The structure of the graph induced by the A5/1 stream cipher is completely analyzed using state-of-the-art techniques and methods. Most of the results presented in the previous chapter differ from the expected values, thus the structure of this graph notably deviates from a random mapping.

A five step approach is applied to retrieve these results. Each step is optimized by hand for best performance considering the special aspects and properties of the graph produced by the A5/1 stream cipher. With these optimizations the properties of the graph can be calculated in a reasonable time.

Massive parallel computations on the GPU is an interesting field with high potentials in applications on huge data sets. Traditional algorithms must be ported under consideration of the different architecture

property	formula [7]	expected value	experimental value	deviation
# of components:	$\frac{1}{2}\log n$	22	113957	$10^{3.71}$
largest cycle:	$\approx 0.78248\sqrt{n}$	$\approx 2^{31.6}$	$\approx 2^{28.81}(469758320)$	$10^{-0.84}$
largest component:	$\approx 0.75782n$	$\approx 2^{63.6}$	$\approx 2^{52.34}$	$10^{-3.39}$
max segment depth:	$0.6\sqrt{n}$	$\approx 2^{31.3}$	$\approx 2^{29.87}$	$10^{-0.43}$
fraction of leaves:	$1/e$	$\approx 36.79\%$	$\approx 37.5\%(6917518582283894784)$	$10^{0.01}$
# of cycle nodes:	$\sqrt{\pi\frac{n}{2}}$	$\approx 2^{32.3}$	$\approx 2^{41}(2219735820460)$	$10^{2.62}$
# of nodes on level:	1	1	≈ 1.7	$10^{0.23}$

Table 1: Comparison of the expected values to the results

of this special hardware in order to make use of the high computing performance of modern GPUs. New hardware versions and architectures of GPUs are provided by the major manufacturers to satisfy the computing needs of modern applications.

Most of the algorithms presented in this work are optimized for the A5/1 graph but some ideas can be ported to other graphs produced by other state transition functions. The scaling behavior of the implementations of the algorithms can be further improved with the ideas mentioned in chapter 5.

References

[1] Andreas Beckmann, Roman Dementiev & Johannes Singler (2009): *Building a parallel pipelined external memory algorithm library*. In: *Parallel Distributed Processing – IPDPS 2009*, IEEE, pp. 1–10, doi:10.1109/IPDPS.2009.5161001.

[2] Andreas Beckmann & Jörg Keller (2007): *Parallel-External Computation of the Cycle Structure of Invertible Cryptographic Functions*. In: *Proc. 15th EUROMICRO Int. Conf. Parallel, Distributed and Network-Based Processing PDP '07*, pp. 526–533, doi:10.1109/PDP.2007.61.

[3] Eli Biham (1997): *A fast new DES implementation in software*. In Eli Biham, editor: *Fast Software Encryption*, *Lecture Notes in Computer Science* 1267, Springer Berlin / Heidelberg, pp. 260–272, doi:10.1007/BFb0052352.

[4] Eli Biham & Orr Dunkelman (2000): *Cryptanalysis of the A5/1 GSM Stream Cipher*. In Bimal Roy & Eiji Okamoto, editors: *Progress in Cryptology – INDOCRYPT 2000*, *Lecture Notes in Computer Science* 1977, Springer Berlin / Heidelberg, pp. 43–51, doi:10.1007/3-540-44495-5_5.

[5] Cryptome.org: *GSM A5 Files Published on Cryptome and Elsewhere*. Available at `http://cryptome.org/0001/gsm-a5-files.htm`. (accessed January 10, 2012).

[6] Roman Dementiev, Lutz Kettner & Peter Sanders (2008): *STXXL: standard template library for XXL data sets*. *Software: Practice and Experience* 38(6), pp. 589–637, doi:10.1002/spe.844.

[7] Philippe Flajolet & Andrew Odlyzko (1990): *Random Mapping Statistics*. In Jean-Jacques Quisquater & Joos Vandewalle, editors: *Advances in Cryptology – EUROCRYPT '89*, *Lecture Notes in Computer Science* 434, Springer Berlin / Heidelberg, pp. 329–354, doi:10.1007/3-540-46885-4_34.

[8] Jovan Dj. Golić (1997): *Cryptanalysis of Alleged A5 Stream Cipher*. In Walter Fumy, editor: *Advances in Cryptology – EUROCRYPT '97*, *Lecture Notes in Computer Science* 1233, Springer Berlin / Heidelberg, pp. 239–255, doi:10.1007/3-540-69053-0_17.

[9] Wireless Intelligence of GSMA Media LLC (February 1, 2011): *Number of connections by bearer technology Q4 2010*. Received via email.

[10] Jörg Keller (2007): *Efficient Sampling of the Structure of Crypto Generators' State Transition Graphs*. In Andrew Blyth & Iain Sutherland, editors: *EC2ND 2006*, Springer London, pp. 3–12, doi:10.1007/978-1-84628-750-3_1.

[11] Jörg Keller & Jop Sibeyn (2001): *Beyond External Computing: Analysis of the Cycle Structure of Permutations*. In Rizos Sakellariou et al., editors: *Euro-Par 2001 Parallel Processing*, *Lecture Notes in Computer Science* 2150, Springer Berlin / Heidelberg, pp. 333–342, doi:10.1007/3-540-44681-8_48.

[12] Leslie Lamport (1981): *Password authentication with insecure communication*. *Commun. ACM* 24, pp. 770–772, doi:10.1145/358790.358797.

[13] Karsten Nohl & Sascha Krissler: *A5/1 Security Project*. Available at `http://reflextor.com/trac/a51`. (accessed January 10, 2012).

[14] NVIDIA: *CUDA C Programming Guide*. Available at `http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf`.

[15] Jeffrey S. Vitter (2008): *Algorithms and Data Structures for External Memory*. now Publishers, doi:10.1561/0400000014.