Fault Cryptanalysis and the Shrinking Generator

Marcin Gomułkiewicz[1] Mirosław Kutyłowski[1]
Paweł Wlaź[2]

1. Wrocław University of Technology

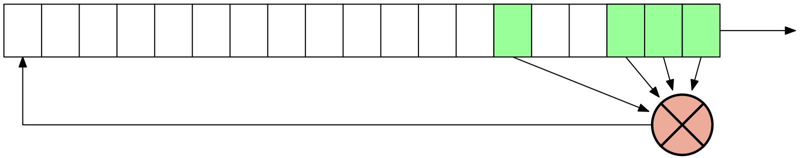
2. Lublin University of Technology

5th International
Workshop on Efficient and Experimental Algorithms

Secure Pseudorandom Generators

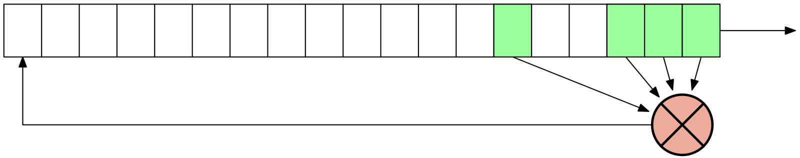
- required for a number of purposes:
  - stream encryption
  - generating random material for security protocols
- hardware solutions:
  - mobile devices with communication capabilities (better security than Bluetooth,...)
  - simple devices (sensors, ...)

LFSR



- in a step:
  - the rightmost bit = the current output bit,
  - all bits move one position to the right,
  - the leftmost bit obtained as a linear combination of bits from certain positions

LFSR



- in a step:
  - the rightmost bit = the current output bit,
  - all bits move one position to the right,
  - the leftmost bit obtained as a linear combination of bits from certain positions
- LFSR, although “good enough” in statistical terms, is cryptographically very weak: easily broken by building a system of linear equations

Shrinking Generator – Features

- pseudorandom number generator
- extremely simple design, yet cryptographically strong
- a clever combination of the output of two simple generators (e.g. LFSRs)

Shrinking Generator – Design

- components:
  - input generator A with output $a_1, a_2, a_3, \ldots$
  - control generator C with $c_1, c_2, c_3, \ldots$
- output of the shrinking generator is composed of those and only those of a_i for which $c_i = 1$.

a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	input generator
0	1	0	1	1	0	1	0	1	control generator
	a_2		a_4	a_5		a_7		a_9	output

Shrinking Generator – Strength

- Many similar architectures proposed (Geffe's generator, stop-and-go, step1-step2)
  all of them turned out to be weak in some sense
- attacks on the shrinking generator:
  - known attacks have complexity exponential in the length of LFSR's used
    - Golič, O'Connor, 1994,
    - Meier, Staffelbach, 1994, Mihaljevic, 1996,
    - Davson, Golič, Simpson, 1998,
  - for known feedback of LFSR's
    - Ekdahl, Johansson, Meier, 2003

Fault Cryptanalysis

classical cryptanalysis :

- only output (and input) considered
- mainly computational methods with nonsolid mathematical background
- *but sometimes works*

Fault Cryptanalysis

classical cryptanalysis :

- only output (and input) considered
- mainly computational methods with nonsolid mathematical background
- *but sometimes works*

fault cryptanalysis: a tamper proof device holding secrets inside

goal: reconstruct the secret key, internal state ...

method: generate faults and analyze the outputs

requirements: no proof required that the result is correct – one can simply check through experiments, *but it should work at least in some cases*

Attack 1: Stopping the Control Generator

- stop the control generator for a few cycles
- observe the changes in the output
- guess the control sequence

Basic Idea – Some Notation

let's denote:

- $A = a_1 a_2 a_3 \ldots$ the input bitstream
- $C = c_1 c_2 c_3 \ldots$ the control bitstream
- $Z^0 = z_1^0 z_2^0 z_3^0 \ldots$ the output from a correct computation
- $Z^1 = z_1^1 z_2^1 z_3^1 \ldots$ the output with the control generator held for 1 step
- $Z^2 = z_1^2 z_2^2 z_3^2 \ldots$ the output with the control generator held for 2 steps
- $\ldots$

Basic Idea – The Easy View

A	a_i	a_{i+1}	a_{i+2}	a_{i+3}	a_{i+4}	a_{i+5}	a_{i+6}
C	1	1	0	1	0	0	1
Z^0	a_i	a_{i+1}	a_{i+2}	a_{i+3}	a_{i+4}	a_{i+5}	a_{i+6}
Z^1	a_{i+1}	a_{i+2}	a_{i+3}	a_{i+4}	a_{i+5}	a_{i+6}	a_{i+7}
Z^2	a_{i+2}	a_{i+3}	a_{i+4}	a_{i+5}	a_{i+6}	a_{i+7}	a_{i+8}
Z^3	a_{i+3}	a_{i+4}	a_{i+5}	a_{i+6}	a_{i+7}	a_{i+8}	a_{i+9}
$\dots$				$\dots$			

Basic Idea – Simple Algorithm

- guess that the number of zeroes between two ones is $0, 1, 2, \ldots$, and check if appropriate bits are equal
- if they are, the guess might be right; if they are not, the guess is incorrect for sure
- algorithm linear in size of its' input data

problem: more equations $\Rightarrow$ less false alarms,
but more zeroes $\Rightarrow$ less equations!

More Realistic Fault Model

- one can stop the control generator for a while (upper bounded), but the exact duration remains unknown and random
- considered tables cannot be constructed directly (placement of rows is unknown)

Reconstructing Ordering of Rows

Data $Z^{\pi(l)}$ $(l = 1, 2, \ldots)$ that was obtained by stopping the control generator for a number of steps

Aim if we could guess π, we could sort the rows out, and perform the Basic Attack

Question how to retreive π?

Finding π

assume that the control sequence starts with $0\dots011$; then if the Z^k and $Z^{k'}$ sequences are consecutive, that is represent consecutive rows for the Basic Attack, then

$$z_2^k = z_1^{k'} \ .$$

if they are not, this equation holds with probability of $\frac{1}{2}$; so, sometimes we can say that some row *cannot* be “the next” after the other

Guessing, guessing...

Idea let's think of the rows as of vertices in directed graph: an edge from vertex k to k' exists $\iff$ row k' **can be next one** after k

π unknown permutation is one of the Hamilton's path in the graph defined

Problem graph is dense, besides finding such paths is NP-complete

Solution ...?

The Solution

Graph is (too) dense? Then make it sparse!

- if we assume that C starts with $00\ldots0\underbrace{11\ldots1}_{N}$ we have $z_2^k = z_1^{k'}$ *and* $z_3^k = z_2^{k'}$ *and* $z_4^k = z_3^{k'}$ *and* $\ldots$
- so the probability of a false alarm exponentially decreases with N

Implementation

- simulations carried out on a regular PC
- calculations last from a split of a second to a couple of hours
- they tend to give at most a few possible candidates, always including the right one
  checking the candidates is straightforward
- for the interested – source code availble (unfortunately with comments in Polish)

Attack 2: Destroying the Control Generator

- the control generator is jammed – its' output bits are completely unrelated to the correct ones, we consider them random, independent, etc.
- observe the output
- guess the input sequence

Basic Idea – Some Notation (again)

let's denote:

- $A = a_1 a_2 a_3 \ldots$ the input bitstream
- $C^i = c^i_1 c^i_2 c^i_3 \ldots$ the ith control bitstream
- $Z^i = z^i_1 z^i_2 z^i_3 \ldots$ output when C^i is the control generator

Probability, probability...

(for a while let $C = c_1 c_2 c_3 \ldots$ and $Z = z_1 z_2 z_3 \ldots$)

- if $c_1 = 1$, then $z_1 = a_1$
  $\longrightarrow$ probability equals $\frac{1}{2}$
- if $c_1 = 0$ and $c_2 = 1$, then $z_1 = a_2$
  $\longrightarrow$ probability equals $\frac{1}{4}$
- $\ldots$
- if exactly $i-1$ of $c_1, c_2, \ldots, c_{j-1}$ are 1 and $c_j = 1$, then $z_i = a_j$
  $\longrightarrow$ probability equals $\binom{j-1}{i-1}\left(\frac{1}{2}\right)^j$ (obviously $i > j$)

Some properties

let X_i be the random variable distributed so that

$$\Pr(X_i = j) = \begin{cases} 0, & \text{for } i > j \\ \binom{j-1}{i-1}\left(\frac{1}{2}\right)^j, & \text{for } i \le j \end{cases}$$

then it can be very easily easily shown that

- $\mathrm{E}[X_i] = 2i$
- $\mathrm{VAR}[X_i] = 2i$

and not quite that easily that

- $\Pr\left(X_k \le 2k \cdot e^{\sqrt{\frac{-2\log p}{k}}}\right) \ge 1 - p$

...so as we can see our probability distribution behaves nicely

Frequency, frequency...

assume we have made N independent experiments, gaining N outputs: $Z^1, Z^2, \ldots Z^N$

- in about $\frac{1}{2}N$ cases z_1 *was* a_1,
- in about $\frac{1}{4}N$ cases z_1 *was* a_2,
- $\ldots$
- in about $\binom{j-1}{i-1}\left(\frac{1}{2}\right)^j N$ cases z_i *was* a_j

Equations, equations...

let $p_{i,j} = \binom{j-1}{i-1}\left(\frac{1}{2}\right)^j$; then

$$\begin{cases}
Np_{1,1}a_1 & + & Np_{1,2}a_2 & + & Np_{1,3}a_3 & + & Np_{1,4}a_4 & + & \ldots & \approx & \sum_{k=1}^{N} z_1^k \\
 & & Np_{2,2}a_2 & + & Np_{2,3}a_3 & + & Np_{2,4}a_4 & + & \ldots & \approx & \sum_{k=1}^{N} z_2^k \\
 & & & & Np_{3,3}a_3 & + & Np_{3,4}a_4 & + & \ldots & \approx & \sum_{k=1}^{N} z_3^k \\
 & & & & & & Np_{4,4}a_4 & + & \ldots & \approx & \sum_{k=1}^{N} z_4^k \\
 & & & & & & & & \ldots & \approx & \ldots
\end{cases}$$

Problem

- we can write as much equations as we wish
- *but* there is always infinitely many variables!

...so we have to cut somewhere

Solution

- we can assume that some $p_{i,j}$ are so small that can be neglected
- consider only partial equations' systems
- do best effort to solve it (hopefully faster than via exhaustive search)

The Algorithm

- we choose a number of equations w
- for each $k = 1, 2, \ldots, w$ we choose v_k: the number of variables considered as important:

$$\Pr(X_k \leq v_k) \geq 1 - p$$

  for some arbitrarily chosen parameter p
- we consider a set of equations:

$$N \sum_{i=1}^{v_k} p_{k,i} a_i = \sum_{l=1}^{N} z_1^l$$

- in the interesting setups it is true that $2k \leq v_k \leq 3k$
  $\longrightarrow$ still more variables than equations

The Algorithm, cntd.

- for the first equation ($k = 1$) we consider all possible values of variables
- for the next equations we consider only variables not considered before
- we keep some pool of the “best” solutions
- “goodness” of a solution s is measured by some reasonable metric:

$$M(s,i) = \sum_{m=1}^{i} \left(\frac{1}{n} \sum_{j=1}^{n} z_m^j - \sum_{j=m}^{v_m} p_{m,j} x_j \right)^2$$

The Results

- simulations carried out on a regular PC
- calculations last from a couple of seconds to a couple of hours
- results are generally not 100% accurate, but they significantly correlate (58% – 95%) to the original values
- for the interested – source code available (unfortunately with comments in Polish)

Conclusions

- security of simple cryptodevices is still a problem
- design methods inherited from “software”-cryptography may do harm for hardware cryptography
  example: avelanche property makes the second attack possible

Thanks for your attention!