SAT Solvers in the Context of Stream Ciphers
Presentation for Journées C2

M. Soos, K. Nohl, C. Castelluccia

PLANETE team INRIA

8th of October 2009
Motivations

Wide usage of cryptography

- Authentication (e.g. NaviGO)
- Privacy protection (Tor)

Previous work on SAT solver-based analysis

- Solving Crypto-1 in 200 s
- Solving Bivium B in $2^{43}$ s

Black-box usage

- Representation not well-optimised for SAT solvers
- Solving not well-examined through statistics
- Solver not optimised for the problem
Goals

Understand the bottlenecks

- Through statistics
- Through visualisations

Remove the bottlenecks

- Adapting the solver to the problem
- Adapting the problem representation to the solver
What is a SAT solver

Solves a problem in CNF

CNF is an “and of or-s”

\[-x_1 \lor \neg x_3 \lor \neg x_2 \lor x_3 \lor x_1 \lor x_2\]

Uses DPLL(\(\varphi\)) algorithm

1. If formula \(\varphi\) is trivial, return SAT/UNSAT
2. Picks a variable \(v\) to branch on
3. \(v \leftarrow \text{value}\)
4. Simplifies formula to \(\varphi'\) and calls DPLL(\(\varphi'\))
5. if SAT, output SAT
6. if UNSAT, \(v \leftarrow \text{opposite value}\)
7. Simplifies formula to \(\varphi''\) and calls DPLL(\(\varphi''\))
8. if SAT, output SAT
9. if UNSAT, output UNSAT
Stream ciphers

Shift register-based stream ciphers

- Use a set of *shift registers*
- Shift registers’ *feedback function* is either linear or non-linear
- Uses a *filter function* to generate 1 secret bit from the state
- Working: clock-filter-clock-filter... = feedback-filter-feedback-filter...

Example cipher
Problem with XOR-s

The truth

\[ a \oplus b \oplus c \]

must be put into the solver as

\[ a \lor \neg b \lor \neg c \] \hspace{1cm} (1) \hspace{1cm} \neg a \lor \neg b \lor c \] \hspace{1cm} (3)
\[ a \lor b \lor c \] \hspace{1cm} (2) \hspace{1cm} \neg a \lor b \lor \neg c \] \hspace{1cm} (4)

So, straightforward conversion takes \(2^{n-1}\) clauses to model an \(n\)-long XOR
Solution until now

Example

\[ x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5 \oplus x_6 \oplus x_7 \oplus x_8 \oplus x_9 \]

Modelled in CNF:

\[ \neg i_1 \oplus x_1 \oplus x_2 \oplus x_3 \oplus x_4 \]
\[ \neg i_2 \oplus x_5 \oplus x_6 \oplus x_7 \oplus x_8 \oplus x_9 \]
\[ i_1 \oplus i_2 \]

Problems

- Still very long to model
- Needs extra vars
Solution to XOR: xor-clause

Example

\[ a \oplus b \oplus c \]

Represents regular clauses

\[ a \lor \neg b \lor \neg c \quad (1) \]
\[ a \lor b \lor c \quad (2) \]
\[ \neg a \lor \neg b \lor c \quad (3) \]
\[ \neg a \lor b \lor \neg c \quad (4) \]

changes appearance to match the situation

Example set-up

\[ a = \text{true} \quad b = \text{true} \quad c = \text{false} \]

\[ \Rightarrow \neg a \lor \neg b \lor c \]
Solution to XOR: xor-clause

Example

\[ a \oplus b \oplus c \]

Represents regular clauses

\[ a \lor \neg b \lor \neg c \quad (1) \]  \quad \neg a \lor \neg b \lor c \quad (3)
\[ a \lor b \lor c \quad (2) \]  \quad \neg a \lor b \lor \neg c \quad (4)

changes appearance to match the situation

Results

- 2.2x speed
- Order of magnitude savings in memory
Solution to XOR: xor-clause

Example

\[ a \oplus b \oplus c \]

Represents regular clauses

\[ a \lor \neg b \lor \neg c \quad (1) \]
\[ \neg a \lor \neg b \lor c \quad (3) \]
\[ a \lor b \lor c \quad (2) \]
\[ \neg a \lor b \lor \neg c \quad (4) \]

changes appearance to match the situation

Challenges overcome

- MiniSat is complex, we needed to completely understand it
- Design choices were difficult: e.g. we use special memory alloc. to maximise cache-hit
Dynamic behaviour analysis

Example search tree

Visualised
- Guesses
- Propagations
- Generated learnt clauses
- Clause group causing the propagation

Calculated stats
- Average depth
- Most conflicted clauses
- No. of guess/branch
- Most guessed vars
- Most propagated vars

M. Soos, K. Nohl, C. Castelluccia (PLANETE team INRIA)

SAT solvers in the context of stream ciphers
8th of October 2009
Gaussian elimination

Reasoning

- Gaussian elimination is efficient for solving systems of linear equations
- xor-clause is a linear equation → use Gauss elim. to solve them

Implementation

<table>
<thead>
<tr>
<th>A-matrix</th>
<th>N-matrix</th>
</tr>
</thead>
</table>
| \[
\begin{bmatrix}
v_{10} & v_8 & v_9 & v_{12} & \text{aug} \\
1 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 \\
\end{bmatrix}
\] |
| \[
\begin{bmatrix}
v_{10} & v_8 & v_9 & v_{12} & \text{aug} \\
1 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 \\
\end{bmatrix}
\] |
Gaussian elimination

Reasoning

- Gaussian elimination is efficient for solving systems of linear equations
- xor-clause is a linear equation → use Gauss elim. to solve them

Implementation

<table>
<thead>
<tr>
<th>A-matrix with (v_8) assigned to true</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \begin{bmatrix} 1 &amp; -1 &amp; 1 &amp; 1 &amp; 1 \ 0 &amp; -1 &amp; 1 &amp; 1 &amp; 1 \ 0 &amp; -1 &amp; 0 &amp; 1 &amp; 0 \ 0 &amp; -0 &amp; 0 &amp; 0 &amp; 0 \end{bmatrix} ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N-matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \begin{bmatrix} 1 &amp; 1 &amp; 1 &amp; 1 &amp; 1 &amp; 0 \ 0 &amp; 0 &amp; 1 &amp; 1 &amp; 1 \ 0 &amp; 1 &amp; 0 &amp; 1 &amp; 1 \ 0 &amp; 1 &amp; 0 &amp; 0 &amp; 1 \end{bmatrix} ]</td>
</tr>
</tbody>
</table>
Gaussian elimination

Reasoning
- Gaussian elimination is efficient for solving systems of linear equations
- xor-clause is a linear equation → use Gauss elim. to solve them

Implementation

<table>
<thead>
<tr>
<th>A-matrix with $v_8$ assigned to true</th>
<th>N-matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_{10}$ $v_8$ $v_9$ $v_{12}$ aug</td>
<td>$v_{10}$ $v_8$ $v_9$ $v_{12}$ aug</td>
</tr>
<tr>
<td>1  -  1  1  1</td>
<td>1  1  1  1  0</td>
</tr>
<tr>
<td>0  -  1  1  1</td>
<td>0  0  1  1  1</td>
</tr>
<tr>
<td>0  -  0  1  0</td>
<td>0  1  0  1  1</td>
</tr>
<tr>
<td>0  -  0  0  0</td>
<td>0  1  0  0  1</td>
</tr>
</tbody>
</table>

Resulting xor-clause:

$v_8 \oplus v_{12}$
Gaussian elimination

Reasoning

- Gaussian elimination is efficient for solving systems of linear equations
- xor-clause is a linear equation → use Gauss elim. to solve them

Implementation

A-matrix

\[
\begin{bmatrix}
1 & - & 1 & 1 & 1 \\
0 & - & 1 & 1 & 1 \\
0 & - & 0 & 1 & 0 \\
0 & - & 0 & 0 & 0
\end{bmatrix}
\]

N-matrix

\[
\begin{bmatrix}
1 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 1
\end{bmatrix}
\]

Resulting xor-clause:

\[v_{12} = \text{false} \quad \leftarrow \quad v_{8} \oplus v_{12}\]
### Gaussian elimination results

<table>
<thead>
<tr>
<th>No. help bits</th>
<th>Gaussian elimination active until level</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Inactive</td>
</tr>
<tr>
<td>Crypto-1</td>
<td>12</td>
</tr>
<tr>
<td>HiTag2</td>
<td>18</td>
</tr>
<tr>
<td>Bivium B</td>
<td>60</td>
</tr>
</tbody>
</table>

### Highlights

- Search space reduced by up to 87%
- Speedup between 0-15%
- A mix of linear and non-linear methods
- Adds possibility to add other algebraic tools → potentially major speedup
Outline
Logical circuit representation

Example

Feedback Function

States

Filter Function

Key stream

Legend
- Variables → boxes
- Functions → hexagons

Complexity measures
- Depth of keystream bit
- Dependency no.: state ↔ keystream
- Difficulty of functions: representation

M. Soos, K. Nohl, C. Castelluccia (PLANETE team INRIA) SAT solvers in the context of stream ciphers 8th of October 2009
Logical circuit representation

Example

Legend
- Variables → boxes
- Functions → hexagons

Complexity measures
- Depth of keystream bit
- Dependency no.: state ↔ keystream
- Difficulty of functions: representation
Dependency graph generator

Example HiTag2 logical circuit

Usage
- Calculate mentioned statistics
- Visual clue
Dependency graph generator

Example HiTag2 logical circuit

Usage
- Calculate mentioned statistics
- Visual clue
Optimising representation of non-linear functions

Example $\mathbb{GF}(2)$ polynomial

$$x_1 + x_1x_2 + x_2x_3 + x_1x_3$$

Usual representation

$$x_1 + i_1 + i_2 + i_3$$

- No. of clauses: $3 \times 3$ regular + 1 xor-clause
- $\sum$ clause length: 31
- 2 extra variables

Karnaugh-table representation

$$\neg x_1 \lor \neg x_3 \lor \neg x_2 \lor x_3 \lor x_1 \lor x_2$$

- No. of clauses: 3 regular
- $\sum$ clause length: 6
- No extra variables
Background
- Simplified version of Trivium eSTREAM candidate
- Best SAT solver-based attack against it takes $2^{43}$ s
- Non-SAT solver-based attack: $2^{64.5}$ s

Our techniques

Find its secret state in approx. $2^{36.5}$ s
Conclusions

SAT solvers have large potential for cryptanalysis

For best results we need to adapt the problem and solver to each other

Such a system is able to break certain ciphers

Possible future work

Further enhance SAT solvers for stream ciphers

Better understand the solving process to arrive at better problem representation

Use generated statistics for understanding the cipher
Thank you for your time
Research results until now

“Attacking Bivium with MiniSat” by (McDONALD et al.)

“Attacking Bivium Using SAT Solvers” by (EIBACH et al.)
We introduce more randomness

- Reference state bits to assign are picked randomly
- The picked bits are assigned randomly true or false
- Clauses are randomly permuted inside MiniSat
- MiniSat’s internal seed (used to randomly explore the search space) is set randomly
- MiniSat’s random number generator has been replaced