SAT Solvers in the Context of Cryptography

Presentation at Rennes

Mate Soos

UPMC LIP6, PLANETE team INRIA, SALSA team INRIA

7th of May 2010
Motivations and goals

Motivations
- Cryptographic primitives could possibly be broken using automated SAT solving tools
- If not, then analysis of cryptographic primitives might be possible using SAT solvers

Goals
- Show why SAT solvers work so well to analyse and/or break cryptographic primitives
- Draw attention to the drawbacks and bottlenecks and how they could be overcome
What is a SAT solver

Solves a problem in CNF

CNF is an “and of or-s”

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

Uses DPLL(ϕ) algorithm

1. If formula ϕ is trivial, return SAT/UNSAT
2. Picks a variable v to branch on
3. v ← value
4. Simplifies formula to ϕ’ and calls DPLL(ϕ’)
5. if SAT, output SAT
6. if UNSAT, v ← opposite value
7. Simplifies formula to ϕ” and calls DPLL(ϕ”)
8. if SAT, output SAT
9. if UNSAT, output UNSAT
Search tree

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
SAT solver internals

Conflict clauses
- Generated when current assignment doesn’t satisfy a clause
- Collection of information leading to UNSAT
- Used to avoid similar wrong parts of the tree next time

Most important parts
- Lazy data structures
- Learning (and forgetting)
- How to pick a variable
Cryptographic problems

Stream ciphers
- Generates pseudorandom keystream given public IV and secret key
- Step-by-step iteration is easy to describe in ANF
- ANF is relatively easy to convert to CNF

Block ciphers
- Encodes a plaintext to a ciphertext given a secret key
- Can have relatively difficult internal parts e.g. S-box
- May be difficult to model in CNF

Hash functions
- Generates one-way, (second)preimage-resistant fingerprint of text
- Usually has relatively difficult internal parts e.g. circular left-shift
- Difficult to model in CNF
Advantages of SAT solvers in the context of cryptography

Lazy data structures
- Fast back-tracking
- Keep partially computed values in memory

Learnt clauses
- Trim the search tree
- Act as memory

Variable activity heuristics
- Search and find good points of entry
- E.g. key bits, shift register states, etc.
Lazy data structures

Watchlists

- Only act upon a clause when we have to
- When all literals are assigned false except for one \( \rightarrow \) assign free literal to true:

\[
v1 \lor v2 \lor v3 \\
v1 = \text{false, } v2 = \text{false} \quad \rightarrow \quad v3 = \text{true}
\]

Internal variables

- It is possible to describe complex functions without internal variables using Karnaugh maps
- But it *slows down* the solving
- Many think they are a necessary evil. They in fact *help*
- They let the solver go back to a point in the search tree without the need to re-compute values
Lazy data structures

Example stream cipher

Explanation
- Hexagons: Filter and feedback functions
- Boxes: Variables (state and internal)
- Green: Final filter functions
- Yellow: Initial state
- Red: Feedback functions
Lazy data structures

Example stream cipher

Explanation

- Hexagons: Filter and feedback functions
- Boxes: Variables (state and internal)
- Red: Internal variables
Lazy data structures

Example stream cipher

- Hexagons: Filter and feedback functions
- Boxes: Variables (state and internal)
- Blue: Dependency of 4th output bit
Learnt clauses

Memory model
- Learnt clauses record each conflict (i.e. no solution is in that part of the search tree)
- They act as memory — but too much memory makes it hard to search effectively
- Trade-off: some learnt clauses are erased periodically

Learnt clause erasure strategy
- If a learnt clause is active part of a new conflict, its activity is increased
- All other clauses’ activity is decreased
- So, learnt clauses that actively help to trim the search tree are preserved

In the context of cryptography — demonstration
./cryptominisat --stats --grouping hitag2.cnf | less
Variable activity heuristics

Searching for a good branch

- Variables are initially randomly branched on
- Variables that appear in during conflict have their activity increased
- All other variables’ activity is decreased
- Most active variables are branched first

Effect of heuristic

- Most difficult variables branched on first
- Difficult = its value affects a lot of other variables
- It divides the problem equally into two parts: \( v = \text{true and false} \)
- If less important variable is picked, the problem is not equally divided: unbalanced tree, search depth becomes huge
Variable activity heuristics in the context of cryptography

Variable activity in crypto-problems

- Stream cipher with initialisation: branches on key bits
- Stream cipher without initialisation: branches on shift register state
- Algebraic side-channel attack: branches on internal variables of the side-channel information round

Demonstration

- Grain — with initialisation
- HiTag2 — without initialisation, shifted 31:
  - Shift register
  - 0 31 78 103
- ASCA for PRESENT — not available
Outline
Disadvantages of SAT solvers in the context of cryptography

**Problem structure lost**
- CNFs is information-short
- Functions: Filter function? Bit-shift?
- Data: Side-channel information? Observed ciphertext?

**Probabilities difficult to handle**
- All clauses must be true
- How could we model $P($information is correct$) = 0.4$?
Problem structure is lost

<table>
<thead>
<tr>
<th>ANF vs. CNF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cannot find out that, e.g.</td>
</tr>
<tr>
<td>$v_1 \oplus v_2 \oplus v_3 = \text{true}$</td>
</tr>
<tr>
<td>$v_1 \oplus v_2 \oplus v_4 = \text{true}$</td>
</tr>
<tr>
<td>$\therefore v_3 = v_4$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Internal variables — what do they represent?</th>
</tr>
</thead>
<tbody>
<tr>
<td>May try to branch on variables introduced to cut up long XOR-s</td>
</tr>
<tr>
<td>Too many monomials may lead to too many variables — could disorientate variable activity heuristics</td>
</tr>
<tr>
<td>Result: In extreme cases, trivial problems take hours to solve</td>
</tr>
</tbody>
</table>
Probabilities difficult to handle

Adding statistical information

- Clauses cannot have probabilities associated with them
- Example: \( P(v_{10} \lor v_{11} \lor v_{12} = \text{true}) = 0.4 \). How can this be modelled?
- Solution: add multiple informations of low probability in one clause:

\[ v_1 \lor v_2 \lor v_3 \]

where \( v_1 \leftrightarrow v_{10} \lor v_{11} \lor v_{12} \)

and \( v_2 \leftrightarrow v_{20} \lor v_{21} \lor v_{22} \)

... 

Probabilistic information may lead to problems

- With (small) probability \( 0.6^3 \): \( v_1 \lor v_2 \lor v_3 = \text{false} \)
- Leads to UNSAT — need to re-start search
- So, statistical information is difficult to model
Conclusions

Concluding remarks

- SAT solvers are an effective way to analyse cryptographic problems.
- Can be used to break simple cryptographic routines automatically.
- But for complex ciphers, careful translation is needed.

Future work

- Recover information from CNF
  - e.g. discover and effectively use XOR functions
- Add information to the CNF:
  - e.g. clause categories: key, ciphertext, side-channel info
- Handle probabilistic information
Thank you for your time

Any questions?