Limits of SAT Solvers in Cryptography

Mate Soos

Presentation at CASED

21st of July 2011
Story Line

1. Introduction to SAT Solvers and Cryptography
2. Advantages of SAT in Crypto
3. Limitations of SAT in Crypto
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(\(\varphi\)) algorithm

1. If \(\varphi\) trivial, return SAT/UNSAT
2. call DPLL(\(v \leftarrow\) value)
3. if SAT, output solution
4. if UNSAT, call DPLL(\(v \leftarrow\) opposite value)
5. if SAT, output solution
6. return UNSAT
Example Search Tree

Guess until conflict

Backtrack

First conflict

Solution Found
SAT Solver Internals

- Lazy data structures: watchlists, fast backtracking
- Learning (and forgetting): what to learnt and what to forget?
- Picking variables: which ones to branch on, and in what order?
- Restarts: when to restart, how far to restart
## Cryptographic Problems

<table>
<thead>
<tr>
<th>Stream ciphers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generates pseudorandom keystream given public IV and secret key</td>
</tr>
<tr>
<td>Step-by-step iteration easy to describe in ANF</td>
</tr>
<tr>
<td>Easy to model in CNF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Block ciphers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encodes a plaintext to a ciphertext given a secret key</td>
</tr>
<tr>
<td>Relatively difficult internal parts e.g. S-box</td>
</tr>
<tr>
<td>May be difficult to model in CNF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hash functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generates one-way, (second)preimage-resistant fingerprint</td>
</tr>
<tr>
<td>Usually has difficult internal parts e.g. adder</td>
</tr>
<tr>
<td>Difficult to model in CNF</td>
</tr>
</tbody>
</table>
Advantages of SAT in Crypto

- Find good points of entry: picking variables
- Partially evaluate the function: lazy data structures
- Effectively store explored search space: learnt clauses
Limitations of SAT in Crypto

**Structure lost**
- CNFs is “plain” – adders, multipliers not evident
- Higher-level reasoning is very difficult
- Cannot find out that, e.g.
  \[ v_1 \oplus v_2 \oplus v_3 = \text{true}, \quad v_1 \oplus v_2 \oplus v_4 = \text{true} \]
  \[ \therefore v_3 = v_4 \]
- Gauss elim. needs exponential resolution operations

**Probabilities difficult to handle**
- All clauses must be true
- Example: \[ P(v_{10} \lor v_{11} \lor v_{12} = \text{true}) = 0.4. \] How to model?
- Introduce indicator variable, make it depend on multiple low-probability events
One-to-one Translation has Limited Potential

Past
- One-to-one translation has been tried on many crypto-primitives
- With varying sophistication levels
- Disappointing results on strong primitives (e.g. SHA1, AES, MD5)

Future
- Don’t model the algorithm one-to-one
- Model a particular aspect of it, e.g. differential path
- Challenges: what to model, how to model, how to interpret results
**Conclusions**

**Concluding remarks**
- SAT solvers can be effective on some crypto problems
- Can break simple cryptographic routines automatically
- But it’s far from plug-and-play for complex crypto-primitives

**Future work**
- Make the plug-and-play experience better for simple problems
- Find crypto-primitive properties to model that could lead to attacks
- Refine properties modelled, refine modelling techniques
Thank you for your time

Any questions?