Attacking AES via SAT

Matthew Gwynne
Computer Science Department
Swansea University

BCTCS
Warwick, April 7, 2009
Introduction

In the following talk, a general translation framework, based around SAT, is considered, with the aim of providing a platform for research into the properties of a variety of cryptographic ciphers and other problems.

In this case the Advanced Encryption Standard (AES) is considered, which has had a lot of interest from other research areas, centring around either it’s algebraic structure or purely around it’s cryptographic structure. This appears to be the first treatment of this cipher using SAT as a basis.
The notion of “breaking” used in this presentation, is that given a single plaintext and single block of ciphertext, can one deduce a key. To find unique keys, given multiple plaintext, ciphertext pairs, this could then be generalised.
Overview

1. AES and SAT
2. Translation of AES into SAT
3. Evaluation of initial translation
4. Future work
AES and SAT

Translation of AES into SAT

Evaluation of initial translation

Future work
AES has been approached using other methods such as:

1. Algebraic methods using both Gröbner bases.
2. Algebraic methods using sparse equation systems.

SAT is especially attractive here as it has had a lot of success in the past 10 years, for example:

1. Finding the van der Waerden number $\text{vdw}_2(6, 6) = 1132$.
2. Solving a variety of scheduling and modelling problems including the verification of train signalling systems.

Such successes highlight SAT as a strong and interesting method and provide additional motivation for it’s use in studying AES. See the recent Handbook of Satisfiability.
Boolean Satisfiability is the problem, given a *propositional formula* $F$, of finding an *assignment* $\varphi : \text{var}(F) \rightarrow \{0, 1\}$ such that $\varphi(F) = 1$.

For example

$$(v_1 \lor v_2) \land (v_1 \lor \neg v_2) \land (\neg v_1 \lor v_2)$$

has the satisfying assignment

$$< v_1 \rightarrow 1, v_2 \rightarrow 1 >.$$
In general, the SAT problem is NP-complete, and thus is an appropriate modelling language for hard problems such as the breaking of AES.

Given successes with modelling, and other hard problems, as well as interest and work in the form of

1. SAT solvers (see minisat, picosat etc)
2. SAT competitions
3. SAT conferences (see SAT 2009, held at Swansea)

SAT provides a good platform for attacking and analysing hard problems.
AES at a high level is made up of the following basic operations:

\[
\begin{align*}
R & : \{0, 1\}^{128} \rightarrow \{0, 1\}^{128} \\
KS & : \{0, 1\}^{128} \times \{1, \ldots, 10\} \rightarrow \{0, 1\}^{128} \\
AES & : \{0, 1\}^{128} \times \{0, 1\}^{128} \rightarrow \{0, 1\}^{128}
\end{align*}
\]

where $R$ is the round operation, made up of smaller operations including the cryptographically important $\textbf{Sbox}$, and $KS$ is the key schedule, which generates a round key for each round.
AES then has the following basic structure:

```plaintext
AES(P, K) {
    P' = KS(K, 0) + P;
    for (r = 1; r <= 10; ++r) {
        P' = R(P') + KS(K, r);
    }
    return P'
}
```

The round function $R$ is made up of several components to permute the bits, usually at the 8-bit word level (considered as elements of the finite field with 256 elements), and in particular the Sbox, applied byte-wise, providing the strong cryptographic properties of the AES.
Translation of AES into SAT
The OKlibrary (http://ok-sat-library.org) is an open-source research environment for those interested in solving hard problems via SAT and other research areas surrounding SAT and other NP-complete problems.

As such, the following translations are available as part of the OKlibrary, and the aim is to have:

1. Generality in the translation, to make it easy to replace components such as the Sbox with other random permutations or the identity function, for example.
2. Good software engineering (Documentation, tests, plans available).
Certain aspects of AES may benefit from special knowledge which isn’t explicit in a CNF representation. Therefore, we would like to introduce a notion of *generalised SAT*:

1. Centred around the idea of “constraints” and “active” clauses.
2. Using the SAT algorithms and notions (such as CNF-like representations).
3. Allowing more powerful inference for and recombination of “sub-constraints” modelling the ability of SAT to recombine different aspects of the problem.
4. Allowing more compact representations.

In a certain sense “merging” SAT and CSP ideas to create a more powerful framework.
Rewrite System

The current translation mechanism works by setting up a set of “constraints”, and rewriting each constraint into a simpler constraints.

For example:

\[
aes\_round(i_1, \ldots, i_{128}, o_1, \ldots, o_{128})
\]

is translated to:

\[
\{ \ \text{aes\_sbox}(i_1, \ldots, i_8, t_1, \ldots, t_8), \ \text{aes\_sbox}(i_9, \ldots, i_{16}, t_9, \ldots, t_{16}), \\
\text{aes\_mul03}(t_1, \ldots, t_8, t'_1, t'_8), \ldots \ \}\n\]

To obtain a (boolean CNF), certain “final constraints” are treated as boolean functions and translated into CNF.
Operations such as the Sbox, multiplication over finite fields, xor etc were studied using their full CNF representations (considering them as simple boolean functions), and then small (although not necessarily the smallest) equivalent representations were generated, i.e., no unnecessary auxiliary variables were introduced.

The study of such representations is a continuing area of research, and relevant research goals here are

1. Study the space of AES translations
2. Learn more about the structure of AES
3. Determine how SAT algorithms react to the structure of AES
Translation

<table>
<thead>
<tr>
<th>Rounds</th>
<th>2</th>
<th>6</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Variables</td>
<td>2272</td>
<td>4672</td>
<td>11617</td>
</tr>
<tr>
<td>Number of Clauses</td>
<td>28616</td>
<td>59312</td>
<td>151144</td>
</tr>
<tr>
<td>Number of Literal Occurrences</td>
<td>172752</td>
<td>352264</td>
<td>894288</td>
</tr>
</tbody>
</table>

Figure: Statistics for our initial AES CNF translations.

The maximum clause size for each is 9 (from Sbox translation).

Note

This translation is only the first natural translation, not a canonical translation. The aim is to study the space of translations and to understand AES.
Evaluation of initial translation
For breaking the AES, i.e., for trying to deduce key bits, given the plaintext and ciphertext, the translation, along with common SAT solvers such as minisat, picosat, OKsolver etc., performs very poorly.

For example, with 118 key bits set, and only 10 unknown, minisat took nearly 20 hours to deduce the remaining key bits and find the satisfying assignment. *Far* slower than simple brute force methods.
AES Encoding

It would seem at first that at least encoding and decoding should be a simple operation for a SAT solver. Is this the case with this translation?
AES Encoding

It would seem at first that at least encoding and decoding should be a simple operation for a SAT solver. Is this the case with this translation?

This isn’t the case, for instance, the well known minisat solver takes 86.9s for a full 10 round encryption, with over 250000 conflicts.

The problem seems is that with 8-bits input, the chosen Sbox-representation doesn’t immediately allow inference, and some work must be done. For example, some assignments of 8 input bits lead to the Sbox clause-set having no clauses of size less than 2, meaning decisions must be made.
One idea for improving is to use a different CNF representation for the Sbox, one that given any set of input and output bits, if the Sbox is then unsatisfiable, it is immediately known, either:

1. Without any reductions (i.e. the empty clause occurs)
2. Using unit clause propagation (available in many solvers including DPLL).

In the case of point 1, such a clause-set is the set of all prime implicates, which yields a fairly large representation (136253 clauses), which would likely yield a much larger clause-set than is reasonable in the current system, and so might hurt performance.
Future work
Further Work

1. Study the Sbox and it’s translations in full detail.
2. Study the AES with alternative Sbox replacements such as random permutations or the identity function.
3. Consider small scale variations of the AES, such as those using different finite fields.
4. Extend translation to generate other non-SAT formulations for comparison (such as CSP representations for use with Gecode).
5. Combine work done using Gröbner bases into a generalised SAT framework (See “Algebraic Aspects of the AES”).
References I


References II
