PAPER

Reconstructing AES Key Schedule Images with SAT and MaxSAT*

Xiaojuan LIAO[†a)], **Hui ZHANG**[†b)], ***Nonmembers***, ***and*** **Miyuki KOSHIMURA**[††c)], ***Member***

SUMMARY Cold boot attack is a side channel attack that recovers data from memory, which persists for a short period after power is lost. In the course of this attack, the memory gradually degrades over time and only a corrupted version of the data may be available to the attacker. Recently, great efforts have been made to reconstruct the original data from a corrupted version of AES key schedules, based on the assumption that all bits in the charged states tend to decay to the ground states while no bit in the ground state ever inverts. However, in practice, there is a small number of bits flipping in the opposite direction, called *reverse flipping errors*. In this paper, motivated by the latest work that formulates the relations of AES key bits as a Boolean Satisfiability problem, we move one step further by taking the reverse flipping errors into consideration and employing off-the-shelf SAT and MaxSAT solvers to accomplish the recovery of AES-128 key schedules from decayed memory images. Experimental results show that, in the presence of reverse flipping errors, the MaxSAT approach enables reliable recovery of key schedules with significantly less time, compared with the SAT approach that relies on brute force search to find out the target errors. Moreover, in order to further enhance the efficiency of key recovery, we simplify the original problem by removing variables and formulas that have relatively weak relations to the whole key schedule. Experimental results demonstrate that the improved MaxSAT approach reduces the scale of the problem and recover AES key schedules more efficiently when the decay factor is relatively large.

key words: *cold boot attack, maximum satisfiability, advanced encryption standard, key recovery*

1. Introduction

1.1 Background

A dynamic random access memory (DRAM) cell is essentially a capacitor that can be either charged or discharged, indicating that a bit is in *the charged state* or *the ground state*, respectively. DRAM remanence, presented by Halderman in 2008 [1], refers to that after power is lost, the DRAM holds its state for several seconds, and for minutes or even hours if the chips are kept at low temperature. Cold boot attack [1] is a sophisticated side channel attack that exploits DRAM remanence effects to recover sensitive data from a running computer. It poses a particular threat to systems that typically store sensitive data in memory. For example, several disk encryption systems have been defeated by the cold boot attack, such as BitLocker and dm-crypt [2]. These on-the-fly disk encryption softwares typically store the encryption key in DRAM while the disk is mounted, which opens a door for an attacker to access the contents of DRAM to learn the key and decrypt the disk.

Given the nature of the cold boot attack, memory bits gradually decay over time once power is removed, and finally, only a corrupted image of memory contents may be available to the attacker. The recovery of a cryptographic key from a corrupted image of memory contents is usually achieved by exploiting the redundancy of key material inherent in cryptographic algorithms. In practice, many encryption programs store data pre-computed from the encryption keys to speed up computation. For block ciphers, a key schedule, made up of multiple roundkeys, is usually precomputed from the secret key. These data contain much more information than the key itself, by which one can efficiently reconstruct the original key even in the presence of errors [2]. The focus of this paper is to recover AES encryption keys from a cold boot attack.

Advanced Encryption Standard (AES) is a specification for the encryption of electronic data established by the U.S. National Institute of Standards and Technology (NIST) in 2001, and is currently used as a worldwide prevalent symmetric cryptographic algorithm. As a kind of block ciphers, AES is vulnerable to the cold boot attack, via which the attacker could extract the encryption key from memory. An AES encryption key refers to a key schedule consisting of multiple roundkeys that are expanded from an *initial key*, via the key expansion algorithm [3]. The length of an AES initial key is 128, 192, and 256 bits, referred to as AES-128, AES-192, and AES-256, respectively. The AES key schedule is the primary source of key redundancy, which enables an attacker to reconstruct the initial key by exploiting the known bits present in the memory, even if the content he extracts has a moderate amount of errors.

1.2 Previous Works

In this section, we first introduce the models related to decay patterns and then survey existing works for recovering the AES key schedule from a cold boot attack.

When memory is out of power, the refresh cycle of

Manuscript received June 9, 2015.
Manuscript revised August 21, 2015.
Manuscript publicized October 6, 2015.

[†]The authors are with School of Computer Science and Technology, Southwest University of Science and Technology, Sichuan, China.
[††]The author is with Graduate School of Information Science and Electrical Engineering, Kyushu University, Fukuoka-shi, 819–0395 Japan.

[*]Part of this paper was presented at IEEE 25th International Conference on Tools with Artificial Intelligence (ICTAI), 2013.

a) E-mail: liao_xiaojuan@swust.edu.cn
b) E-mail: h-zhang@swust.edu.cn
c) E-mail: koshi@inf.kyushu-u.ac.jp

DOI: 10.1587/transinf.2015EDP7223

DRAM is interrupted. Halderman et al. [1] observed that most memory bits tend to decay to the ground states as time goes on, with a constant and small fraction of bits flipping to the charged states. The ground state could be encoded as either 0 or 1, depending on how the cell is wired, thus the decay of memory bits is overwhelmingly either $0 \rightarrow 1$ or $1 \rightarrow 0$. The decay direction in a given region could be inferred by comparing the numbers of 1s and 0s since in an uncorrupted key schedule, we expect these to be approximately equal [1]. For simplicity of elaboration, in the rest of the paper, we assume 1 as the charged state, and 0 as the ground state. Then the decay direction is overwhelmingly $1 \rightarrow 0$, with a small fraction from 0 to 1.

We denote the probability of 1 degrading to 0 by decay factor δ_0, and the the probability of 0 flipping to 1 by some fixed δ_1. Generally, δ_0 reflects the extent of decay, which approaches to 1 as time goes on after power is lost. By contrast, δ_1 is relatively constant and tiny, from 0.05% to 0.1% [1]. According to the different settings of δ_1, existing works model the decay patterns by either of the following cases.

- Perfect assumption: since δ_1 is quite small in true cold boot attacks, the decay direction is assumed only from 1 to 0 with no bit flipping in the opposite direction, i.e., δ_1 is set 0. Therefore, all 1s present in the decayed key schedule are correct with absolute certainty.
- Realistic assumption: δ_1 is from 0.05% to 0.1%. In this setting, both $1 \rightarrow 0$ and $0 \rightarrow 1$ coexist in an attack, thus none of the key bits in the decayed key schedule are known with absolute certainty.

The technique of identifying AES keys in memory has been developed in [2]. In the following, we investigate previous works on how to reconstruct AES key schedules after they have been extracted from memory. The AES key schedule contains a large amount of linearity, which allows one to search for a small set of keys exhaustively and then combine these small pieces into the overall key. The method proposed in [1] takes advantage of the high amount of linearity. Instead of trying to reconstruct the entire key at a time, their algorithm cuts up the 128-bit roundkey into four subsets of 32 bits, and uses 24 bits of the subsequent roundkey as redundancy. These small sets are decoded in order of likelihood, and then combined into a candidate key, which could be checked against the full schedule. Their work reconstructed AES-128 keys with $\delta_0 = 15\%$ and $\delta_1 = 0.1\%$ in a fraction of a second, and up to half of keys with $\delta_0 = 30\%$ and $\delta_1 = 0.1\%$ within 30 seconds.

As far as we concern, [1] is the only work that works for the realistic assumption, while many existing works adopt the perfect assumption. The algorithm presented in [4] makes better use of the AES key schedule structure, by modeling the search of keys in a depth-first tree under tree-pruning constraints. Their method allows one to recover the AES-128 key schedules with averagely 300 seconds for $\delta_0 = 70\%$ and $\delta_1 = 0$. Although it was noted in [4] that the proposed algorithm did still work for realistic assumption, the methodology was not mentioned in their work, and the performance in this case was not demonstrated in [4]. Later on, motivated by the dramatic speed-up of Boolean Satisfiability (SAT) solvers, Abdel et al. [5] took the initial step to model the AES key recovery problem as a SAT problem by making full use of bits equal to 1. Based on the perfect assumption, all 1s in the decayed key schedule are correct, thus a set of correct constraints could be constructed from 1s. On the other hand, the large amount of redundant information available in the AES key schedule could also be formulated as constraints that have to be satisfied in a SAT problem. As a result, by employing CryptoMiniSat [6], an XOR supported SAT solver, their approach considerably improved the performance of AES key recovery, in terms of both the recovery speed and the maximum recoverable decay factor. Specifically, the authors reported that recovering AES-128 key schedules could be fulfilled with $\delta_0 = 70\%$ and $\delta_1 = 0$ in around 1.2 seconds on average, and recovering keys with $\delta_0 = 80\%$ and $\delta_1 = 0$ became possible.

1.3 Our Contribution

The performance obtained in [5] confirmed the superiority of SAT solvers in AES key recovery. However, Abdel et al. [5] thoroughly excluded the possibility of reverse flipping errors, by assuming that all 1s are correct with absolute certainty. In fact, based on $\delta_1 = 0.1\%$, the event of reverse flipping is expected to arise around 1 or 2 times in a real key recovery attack for AES-128 where the total number of key bits is 1408, and more times for AES-192 and AES-256, which contain larger number of bits apt to decay. Although bits flipping from 0 to 1 are very rare in number, these fatal events are sufficient to derail SAT solvers, leading SAT solvers to mistake the wrong 1s as the correct constraints to infer the rest unknown bits. As a result, conflicts may occur during the reasoning and SAT solvers would fail to recover the correct key schedule, with unsatisfiable outputted.

In this paper, we show how to employ SAT and MaxSAT to recover AES key schedules in the presence of reverse flipping errors. More specifically, we make the following contribution:

- We extend the SAT approach in [5], which is originally designed for AES key recovery under the perfect assumption, to adapt to the realistic assumption.
- Motivated by the feature of MaxSAT which satisfies as many constraints as possible by eliminating the minority unsatisfied ones, we recast the problem of AES key recovery under the realistic assumption as a partial MaxSAT problem. Experimental results show that the MaxSAT approach could recover decayed AES key schedules with $\delta_0 = 76\%$ and $\delta_1 = 0.1\%$ almost three times faster than the SAT approach.
- We further improve the presented MaxSAT approach by simplifying the key recovery problem. Experimental results demonstrate that, when the decay factor is relatively large, i.e., δ_0 is no less than 60%, the improved approach enables more efficient key recovery.

2. Preliminaries

Boolean Satisfiability (SAT) is the first problem shown to be NP-complete [7], which is used to determine whether there exists an assignment of Boolean values that makes a propositional Boolean formula evaluate true. If such an assignment exists, the formula is satisfiable, otherwise, the formula is unsatisfiable. A *propositional Boolean formula* is a Boolean formula that contains only logic operations *and*, *or* and *not*. Typically, a propositional Boolean formula is expressed in conjunction normal form (CNF), a widely-used expression consisting of a conjunction (logic *and*) of one or more clauses. A *clause* is a disjunction (logic *or*) of one or more literals, and a *literal* is an occurrence of a Boolean variable (e.g., x) or its negation (e.g., $\neg x$).

The versatility and effectiveness of SAT solving techniques show the potential use of SAT solvers as a tool for cryptanalysis. Several cryptanalytic attacks using SAT solvers emerged, ranging from cryptanalysis of block ciphers [8], [9] and stream ciphers [10], [11] to asymmetric-key algorithms [12], [13] and hash functions [14], [15]. Another line of research focuses on the attempts to make SAT solvers more cryptanalytic-friendly. Soos et al. [6] implemented several steps towards a specialized SAT solver for cryptography, including native support for the XOR operation, Gaussian elimination, and logical circuit generation. Facilitated by XOR supported mechanisms in CryptoMiniSat, attacks against stream ciphers such as Crypto-1 [10] could be accelerated considerably, compared with other standard SAT solvers that only support CNF representations.

Given a propositional Boolean formula, if it is unsatisfiable, SAT solvers only report that no solution exists, without any information on unsatisfiable instances. In many realistic situations, not all specified constraints can be satisfied at the same time, and we seek for an optimized assignment of variables that could satisfy as many constraints as possible. At this point, SAT fails by only outputting unsatisfiable and MaxSAT comes to rescue. Maximum Satisfiability (MaxSAT) [16] is an optimized version of SAT, which could measure the degree of unsatisfiability. Given a propositional Boolean formula, MaxSAT tries to find an assignment that maximizes the number of satisfied clauses. Partial MaxSAT is a variation of MaxSAT. In a partial MaxSAT instance, some clauses are declared *soft* and the rest are *hard*. The problem amounts to finding an optimal assignment that satisfies all hard clauses and the maximum number of soft clauses. Also, in practice, a partial MaxSAT instance is usually given as a CNF. In this paper, we identify a set of clauses as a conjunction of clauses.

3. Modeling Bits in the AES-128 Key Schedule

AES is a widely spread symmetric key algorithm that uses the same cryptographic keys for both encryption and decryption. In this paper, we focus on AES-128 and our method could be extended to AES-192 and AES-256 in a straightforward way since the key schedule for 128-bit, 192-bit, and 256-bit encryption are very similar, with only some constants changed.

An AES-128 key schedule consists of 11 roundkeys, each made up of 128 bits. The 0^{th} roundkey is equal to the initial key itself, which is bijectively mapped to the subsequent 10 roundkeys, via the public AES key expansion algorithm. Each bit in the key schedule, either 0 or 1, is naturally expressed as a Boolean variable. We denote the i^{th} bit of the r^{th} roundkey by b_i^r, where $0 \leqslant r \leqslant 10$ and $0 \leqslant i \leqslant 127$. The key schedule components are addressed with the following two notations: the round-dependent word array and the substitution box. The round-dependent word array, denoted by $R(r)$, contains the values $\{02\}_H^{r-1}$ for the least significant byte and 0 for the rest bytes, with $\{02\}_H^{r-1}$ being powers of $\{02\}_H$ in the Galois field $GF\left(2^8\right)$, where $\{02\}_H$ is the hexadecimal representation of 2. The substitution box, abbreviated with S-box, is a basic component regarding security in AES key schedule, which operates independently on a byte, by taking the multiplicative inverse in the finite field $GF\left(2^8\right)$ using the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$ and then applying an affine transformation over $GF(2)$. For more details, we refer readers to the literature [3]. According to the hardware implementation of the AES key expansion algorithm, an S-box operation could be split into eight functions in algebraic normal form (ANF)[†], with 1-byte input and 1-bit output [17]. We denote each of the eight functions by $S_x\left(B_i^r\right)$, where $x\ (= 0, \ldots, 7)$ is the index of the function, and B_i^r is an input byte starting with b_i^r, following the least-significant-bit-first convention, i.e., $B_i^r = \{b_i^r, b_{i+1}^r, \ldots, b_{i+7}^r\}$. The output of an S-box is then obtained by combining the outputs of these eight functions into a byte, with the decreasing significance of x. Specifically, each bit in 1-10 roundkeys is described by the following formulas.

$$\begin{aligned}
b_i^r &= b_i^{r-1} \oplus S_{i\,mod\,8}\left(B_{104+8\cdot\lfloor i/8\rfloor}^{r-1}\right) \oplus R_i(r), 0 \leqslant i \leqslant 23,\\
b_i^r &= b_i^{r-1} \oplus S_{i\,mod\,8}\left(B_{96}^{r-1}\right) \oplus R_i(r), 24 \leqslant i \leqslant 31, \qquad (1)\\
b_i^r &= b_i^{r-1} \oplus b_{i-32}^r, 32 \leqslant i \leqslant 127.
\end{aligned}$$

where $1 \leqslant r \leqslant 10$, $R_i(r)$ is the i^{th} bit of the round constant word array $R(r)$, and $\lfloor x \rfloor$ is the floor function that returns the largest integer not greater than x. The relations among bits exhibited in Eq. (1) show that each bit in the subsequent 10 rounds is associated with its former bits. In particular, b_i^r $(0 \leqslant i \leqslant 31, 1 \leqslant r \leqslant 10)$ is determined by 9 bits, i.e., b_i^{r-1} and $B_{104+8\cdot\lfloor i/8\rfloor}^{r-1}$, and b_i^r $(32 \leqslant i \leqslant 127, 1 \leqslant r \leqslant 10)$ is determined by 2 bits, i.e., b_i^{r-1} and b_{i-32}^r. Thus, an error occurring on a bit could be rectified by examining the values of its related 9 or 2 bits. Similarly, if several bits are decayed, they could be recovered correctly as long as a sufficient number of bits are known. For convenience, we call the relations characterized in Eq. (1) as *bit-relations*.

[†]A logical formula is considered to be in ANF if it is an XOR of a constant and one or more conjunctions of Boolean variables.

4. Recovering the AES-128 Key Schedule with SAT/MaxSAT

This section introduces three approaches that work for the true cold boot case. Specifically, the SAT and MaxSAT approaches described in Sect. 4.1 and 4.2 take the complete bit-relations as the constraints to infer corrupted key bits, thus the aim is to recover the complete AES key schedule. In comparison, the improved MaxSAT approach presented in Sect. 4.3 seeks to simplify the key recovery problem by taking only part of bit-relations into consideration, thus the goal is to recover only a portion of round keys. Based on the observations made in [1], δ_1 is very small (though non-zero) while δ_0 may be relatively large.

4.1 SAT Approach

For a SAT solver, the attempt of finding an assignment to variables is made on the premise that all the specified constraints are satisfied without any exception. This feature is suitable for the perfect assumption where all 1s present in the decayed key schedule are correct with absolute certainty. In addition, the bit-relations in the AES key schedule can be easily formulated as a SAT problem which lead itself naturally to SAT solvers [5]. Therefore, given enough number of 1s in the corrupted key schedule, a SAT solver could infer the values of other unknown bits by formulating all the 1s and bit-relations as hard constraints.

However, in case of some 0s flipping to 1s, a SAT solver could not be aware of such reverse flipping errors, instead, it still takes all 1s as absolutely correct. Misled by the incorrect information, the solver would fail to recover the key schedule by outputting unsatisfiable as a result of the conflicts arising during the reasoning. Obviously, the heart of addressing this problem is to find and rectify the reverse flipping errors. A straightforward way would be to conduct exhaustive search over all 1s in a bit-by-bit fashion, by exploiting the fact that a SAT solver would always output unsatisfiable as long as reverse flipping errors exist. Only if all the wrong 1s are cleared, the solver is probable to find the correct assignment of variables. Particularly, when there is one reverse flipping error, to distinguish this wrong 1 from other correct 1s, we modify the decayed key schedule by turning one of these 1s to 0, then take the modified key schedule as the input of a SAT solver. If the solver outputs satisfiable, it means that the bit that we just modified is the reverse flipping error. Otherwise, we revert that bit; convert the next 1 to 0, and run the solver again. This process would repeat until the solver outputs satisfiable, i.e., it locates and corrects the reverse flipping error. In the worst case, the solver needs to run $(n+1)$ times, where n is the number of 1s in the decayed key schedule. The complexity of the exhaustive search is closely related to the actual number and the location of the reverse flipping errors, neither of which are unknown to a SAT solver. If the number of reverse flipping errors is k, in the best case, the solver needs to run $\left(\sum_{0\leqslant i\leqslant k-1} C_n^i + 1\right)$ times, while in the worst case, it has to try $\sum_{0\leqslant i\leqslant k} C_n^i$ times, where C_n^i is the number of i-combinations of n. It is clear that a SAT solver has to run multiple times to recover a decayed key schedule with reverse flipping errors. Obviously, this method is tedious and cumbersome.

4.2 MaxSAT Approach

We can solve the problem significantly better by taking advantage of the partial MaxSAT. That such a connection exists should be no surprise: we are in a situation where the majority of 1s are correct, with only a small fraction of 1s flipping from 0, and we wish to find out such reverse flipping errors and recover the true bits. A partial MaxSAT solver treats the bit-relations as hard constraints, while considers the bits equal to 1 as soft constraints, i.e., it takes the possibility that the present 1s may be incorrect into consideration. Solving the partial MaxSAT problem amounts to finding an assignment of variables that satisfy all hard constraints and the maximum number of soft constraints. As long as the reverse flipping errors account for a small percentage among all 1s, they can be surely cleared by a partial MaxSAT solver, which always satisfies the majority of soft constraints by excluding the unsatisfied minority. By using MaxSAT solvers, the exhaustive search over all 1s is eliminated.

Modeling 1s as soft clauses is fulfilled without further elaboration. In particular, if the i^{th} bit of the r^{th} round-key presents 1 in the decayed key schedule, we only need to declare $b_i^r = 1$ as a soft clause to a MaxSAT solver. By contrast, modeling bit-relations as hard clauses is not such straightforward. The main problem is to handle the XOR operation, which is fundamental in characterizing bit-relations. In the following, we elaborate the way of describing the following two kinds of formulas in CNF representations.

(1) XOR formula: a formula connected by XOR operator, with all the terms being individual Boolean variables.
(2) ANF formula: a formula connected by XOR operator, with terms consisting of a constant (0 or 1) and conjunctions of Boolean variables.

A naive way of describing an XOR formula in CNF representation introduces 2^{n-1} clauses, where n is the size of the XOR formula. This straightforward way, called *direct conversion*, applies to the case that the size of an XOR formula is relatively small. For example, in an AES-128 key schedule, to describe the bit-relation $b_i^r = b_i^{r-1} \oplus b_{i-32}^{r-1}$ $(32 \leqslant i \leqslant 127, 1 \leqslant r \leqslant 10)$ in CNF, only the following four clauses are sufficient:

$$\neg b_i^r \vee b_i^{r-1} \vee b_{i-32}^{r-1}, \qquad b_i^r \vee \neg b_i^{r-1} \vee b_{i-32}^{r-1},$$
$$b_i^r \vee b_i^{r-1} \vee \neg b_{i-32}^{r-1}, \qquad \neg b_i^r \vee \neg b_i^{r-1} \vee \neg b_{i-32}^{r-1}.$$

Obviously, this interpretation is unmanageable for large n as the number of clauses goes up exponentially with the size of an XOR formula. To overcome this exponential explosion, a long XOR formula is usually cut up into manageable groups, each represented by an auxiliary variable. Here we set the size of manageable groups as k, i.e., we introduce an auxiliary variable for each k variables, thus the number of clauses to describe one group is no more than 2^k. The long XOR formula is then represented as a new XOR of $\lceil n/k \rceil$ auxiliary variables, as well as a set of clauses for describing the manageable groups. If the size of the new XOR formula is still overlong, then more auxiliary variables are introduced to further cut up the formula. This process repeats until the size of the new XOR formula is manageable. We call this method as *cut-up conversion*.

Up to this point, we have discussed the way of describing both short and long XOR formulas in CNF representations. To encode ANF into CNF, we need to first introduce additional Boolean variables to substitute the conjunctions, so that turn the ANF formula to an XOR, which could be handled by either direct or cut-up conversion. Formally, a conjunction of Boolean variables, denoted by $x_0 \wedge x_1 \wedge \cdots \wedge x_n$, could be represented by a new Boolean variable a with $n+1$ additional clauses:

$$x_1 \vee \neg a,\ x_2 \vee \neg a, \ldots, x_n \vee \neg a, \neg x_1 \vee \neg x_2 \cdots \vee \neg x_n \vee a.$$

Thus the ANF formula is converted to a set of clauses and an XOR of a constant and several additional Boolean variables. If the constant is 0, we remove it safely without further conversion. Otherwise, we eliminate the constant 1 by turning one of the Boolean variables to its negation. Specifically, $1 \oplus x_1 \oplus x_2 \ \cdots \oplus x_n$ is tautologically equivalent to $\neg x_1 \oplus x_2 \ \cdots \oplus x_n$.

4.3 Improved MaxSAT Approach

The AES-128 key schedule consists of 11 rounds, in which the initial round (i.e., the 0th round) is the 128-bit initial key and the expanded 10-round key bits could be all derived from the initial key. Then the ultimate goal could be refined as to recover the 128-bit initial key, rather than the whole 1408-bit key schedule. With this in mind, in this section, we try to simplify the key recovery problem by removing some variables and formulas in the expanded rounds.

Definition. *A variable b_i^r ($r > 0$) in the expanded rounds is called a* leaf variable *if there is only one formula containing b_i^r. The only formula associated with the leaf variable b_i^r is called a* leaf formula*, denoted by f_i^r. The tuple $\left(b_i^r, f_i^r\right)$ is called a* leaf set.

In a complete AES-128 key schedule, there are 32 leaf variables, i.e., the last 32 bits in the last round, denoted by b_i^{10}, $i \in \{96, \cdots, 127\}$. The corresponding leaf formula f_i^{10} is $b_i^{10} = b_i^9 \oplus b_{i-32}^{10}$. By contrast, any other variable in the expanded rounds is contained in more than one formula, one of which describes how this variable is calculated from former variables, and the rest indicate the way of generating latter variables. Obviously, compared with non-leaf variables, leaf variables have weaker relations to other variables, as there is only one constraint on the leaf variable b_i^{10}, specified by the leaf formula f_i^{10}. Without f_i^{10}, the value of b_i^{10} becomes unconstrained and could be omitted in key expansion.

On the other hand, in the MaxSAT approach, variables equal to 1 have high probability (around 99.9%) to be correct and are treated as soft clauses by the MaxSAT solver. In comparison, a variable (suppose x) equal to 0 contributes nothing to the problem solving, because this 0 may be the original value of x, or the decayed one, while the decay factor δ_0 is totally unknown to the solver. Therefore, it could be said that variables equal to 0 provide less information than those equal to 1.

Our improved approach is to search and remove the leaf sets, i.e., the combinations of leaf variable and the corresponding leaf formula, where all variables in the leaf formula are equal to 0. Since leaf variables have relatively weak relations to the whole key schedule and bits equal to 0 provide little useful information to the solver, it could be inferred that removing such leaf sets would not result in vital information loss. Instead, elimination of them would reduce the scale of the key recovery problem, which plays a positive role to increase the efficiency of problem solving.

The following algorithm exemplifies the way of eliminating qualified leaf sets in the last round. The input is a complete AES-128 key recovery problem, denoted by $\{B, F, S\}$, where B and F is a set of 1408 bits and formulas representing bit-relations, respectively, and S is a set of bits presenting 1 in the decayed key schedule. The output is a simplified key recovery problem represented as $\{B', F', S\}$, where B' and F' is a subset of B and F, respectively.

Algorithm Simplify the AES key recovery problem by removing leaf sets in the last round.

```
Input:
    A complete AES-128 key recovery problem ${B, F, S}$;
    $B = \{b_i^r\}$ is a set of key bits, $i \in \{0, \cdots, 127\}$, $r \in \{0 \cdots, 10\}$;
    $F = \{f_i^r\}$ is a set of formulas characterizing bit-relations, $i \in \{0, \cdots, 127\}$, $r \in \{0 \cdots, 10\}$;
    $S$ is a set of variables that present 1 in the corrupted key schedule.
Output:
    A simplified AES-128 key recovery problem $\{B', F', S\}$.
 1: $LB \leftarrow \varnothing$; $LF \leftarrow \varnothing$; $RB \leftarrow \varnothing$; $RF \leftarrow \varnothing$. {$LB$ and $LF$ is a set of leaf variables and formulas, respectively; $RB$ and $RF$ is a set of removable variables and formulas, respectively.}
 2: for $i = 96, 97, \cdots, 127$ do
 3:     $LB.push\left(b_i^{10}\right)$; $LF.push\left(f_i^{10}\right)$;
 4: end for
 5: while $LB \neq \varnothing$ do
 6:     $b_i^r \leftarrow LB.pop$; $f_i^r \leftarrow LF.pop$;
 7:     if $S$ contains none of variables in $f_i^r$ then
 8:         $RB.push\left(b_i^r\right)$; $RF.push\left(f_i^r\right)$; {$\left(b_i^r, f_i^r\right)$ could be removed.}
 9:         if $i > 31$ then
10:             $LB$.push($b_{i-32}^r$); $LF$.push($f_{i-32}^r$); {Since $b_i^r$ is removable, $b_{i-32}^r$ becomes the new leaf bit, the successor of $b_i^r$.}
11:         end if
12:     end if
13: end while
14: $B' \leftarrow B \setminus RB$; $F' \leftarrow F \setminus RF$;
15: return $\{B', F', S\}$;
```

Steps 1-4 initialize four sets LB, LF, RB, and RF. LB and LF is a set of leaf variables and formulas, initialized with $\{b_i^{10}\}$ and $\{f_i^{10}\}$, respectively, where $96 \leq i \leq 127$. RB and RF respectively represents a set of removable variables and formulas. In the beginning of the algorithm, both RB and RF are empty.

Steps 5-13 describe the way of finding out removable variables and formulas from leaf sets. To be specific, first, the condition whether LB is empty is examined. That LB is empty means that all the leaf variables have been checked and the loop terminates. If LB is not empty, the last element in set LB (set LF) is removed and assigned to a new variable b_i^r (formula f_i^r), where the indexes i and r are both consistent with those of the removed element (Step 6). For each leaf formula f_i^r, check whether S contains any variables in f_i^r. If S contains none of variables in f_i^r, which signifies that all the variables in f_i^r are 0, then the set $\left(b_i^r, f_i^r\right)$ is removable, and we push b_i^r and f_i^r to RB and RF, respectively (Steps 7-8). Now that the leaf set $\left(b_i^r, f_i^r\right)$ has been removed, a new leaf set arises as the successor of the removed one, denoted by $\left(b_{i-32}^r, f_{i-32}^r\right)$ if $i > 31$. $i > 31$ guarantees that the new leaf set still belongs to the current round (Steps 9-11). After b_{i-32}^r and f_{i-32}^r is pushed to LB and LF, respectively, whether the new leaf variable and formula could be removed is determined by the values of variables appearing in f_{i-32}^r.

The while loop terminates when there is no more leaf variables to be dealt with. Finally, we get the simplified AES-128 key recovery problem $\{B', F', S\}$, where B' is the difference between B and RB, and F' is the difference between F and RF.

It is worth noting that the elimination of leaf sets not only happens in the last round, but also may occur in the $(10-m)^{th}$ round, where $1 \leq m \leq 9$. However, in practice, the number of removable leaf sets in these rounds dramatically decreases with the increase of m. This is because each variable in the earlier rounds is contained in a relatively large number of formulas. To remove such variable, it is necessary to guarantee that all the variables appearing in these related formulas are 0. For example, each variable in the last block of the 9^{th} round is involved in eight S-box functions and two short XOR functions. One of the requirements of removing such variable is that all the variables in these ten functions are 0, which is such a rigorous restrictive condition that is difficult to satisfy.

5. Experiments and Comparisons

In this section, we evaluate the performance of AES-128 key recovery with SAT and MaxSAT. We employed CryptoMiniSat as the solver to evaluate the SAT approach as it supports XOR operations natively. In comparison, the solver chosen for the MaxSAT approach is Pwbo2.0. The way of selecting the appropriate MaxSAT solver from a variety of candidates is elaborated in our previous work [18].

5.1 Generating Problem Instances

According to [1], in a true cold boot attack, δ_1 is around 0.1% and δ_0 increases as time goes on. To be consistent with the realistic assumption, we generate the problem instances as follows. The test generator first derives a key schedule from a randomly selected initial key where the number of 0s and 1s are approximately equal, then it randomly converts 1s to 0s with the probability of δ_0, and converts 0s to 1s with the probability of δ_1. We set $\delta_1 = 0.1\%$ and vary δ_0 from 30% to 76%. Since the time for recovering a key schedule observed in the experiments rises dramatically when δ_0 grows over 70%, we set δ_0 from 30% to 70% with 10% increments, and 70% to 76% with 2% increments, respectively. At each fixed decay factor δ_0, 100 problem instances are generated. In this setting, the number of reverse flipping errors in the generated problem instances ranges from 0 to 2.

To investigate the performances of our approaches under specific number reverse flipping errors, we generate additional problem instances for the following cases:

(1) Each instance contains only 1 reverse flipping error.
(2) Each instance contains 2 reverse flipping errors.

Methods of generating instances for these two cases are almost the same as that for the realistic assumption, except that we remove the setting of δ_1 and limit the number of reverse flipping errors to 1 and 2, respectively. For case (2), due to the extended time for key recovery, we range δ_0 from 30% to 74% and generate 40 instances for each fixed δ_0.

5.2 Selecting the Appropriate Cutting Number

As mentioned in Sect. 4.2, to alleviate the exponential explosion problem when converting long XOR formulas to a set of clauses, each long XOR formula is cut up into manageable groups. The size of such groups is called *cutting number*, denoted by k. Generally speaking, the optimal value of k may change with different situations and is usually determined via empirical observations. For instance, to convert systems of low-degree sparse multivariate equations into CNF, the optimal cutting number is 6 [19], while to convert a linear ANF in a stream cipher *Bivium-n*, the optimal cutting number is 5 [20]. In order to select the appropriate cutting number in our work, we evaluate the performance of MaxSAT approach with $k = 4, 5, 6$, under a time limit of 900 seconds for each instance. δ_0 ranges from 30% to 74%, and the number of reverse flipping errors is set 1 and 2, respectively exhibited in Table 1 and 2. Number in round bracket means the number of instances that were successfully solved within the time limit and is omitted in the table if all the 100 instances were solved. Tests were carried out on a 2.0GHz quad-core Intel i7-4510U processor with 8GB RAM.

As can be seen from Table 1 and 2, when the decay factor δ_0 is relatively small, i.e., δ_0 is no more than 60%, cutting by 4 achieves higher efficiency than the other two settings. By contrast, when δ_0 goes over 60%, cutting by 5 becomes

Table 1 Average runtime (seconds) with $k = 4, 5, 6$ ($\#RFE = 1$)

δ_0 (%)	30	40	50	60	70	72	74
$k = 4$	0.63	0.68	0.84	1.47	15.23	20.79	73.78(96)
$k = 5$	0.77	0.83	1.00	1.64	10.10	17.91	69.50(96)
$k = 6$	1.19	1.23	1.47	2.10	19.28	28.65	103.25(94)

Table 2 Average runtime (seconds) with $k = 4, 5, 6$ ($\#RFE = 2$)

δ_0 (%)	30	40	50	60	70	72	74
$k = 4$	0.82	0.98	1.47	3.90	42.50	136.58(97)	250.34(62)
$k = 5$	0.97	1.19	1.57	4.01	36.99	118.60(97)	253.23(69)
$k = 6$	1.48	1.70	2.12	5.42	59.75	158.81(94)	234.44(55)

more desirable. Since solving problems with higher decay factor is more deserving, we choose $k = 5$ as the cutting number for the MaxSAT approach.

5.3 Experimental Results of SAT and MaxSAT Approach

In this subsection, we evaluated the performance of AES-128 key recovery with CryptoMiniSat and Pwbo2.0, respectively. Tests were carried out on a 2.6GHz quad-core Intel i5-2540 processor with 8GB RAM. The benchmark results for true cold boot attacks are summarized in Table 3, where $\delta_1 = 0.1\%$ and δ_0 ranges from 30% to 76%.

In general, the MaxSAT approach enables more efficient key recovery than the SAT approach, at all the varied decay factors. Specifically, the solver time needed by the MaxSAT approach rises monotonically with the increase of δ_0, indicating that it is more difficult to fulfill the key recovery at increasing decay factor. In comparison, for the SAT approach, the solver time presents a down and up trend as δ_0 increases. We explain the reasons as follows. First, at the low decay factor, particularly when δ_0 is 30%, the number of 1s in a decayed schedule is relatively large, implying that CryptoMiniSat needs to run a considerable number of times to find out the reverse flipping errors. At this point, the solver time is maintained in high level. Later, with the increase of the decay factor, the number of 1s drops, leading to fewer number of times to run CryptoMiniSat. In addition, the increasing decay factor is not large enough to affect the high efficiency of CryptoMiniSat to solve an input file. In this situation, the solver time declines. Finally, as the decay factor further rises, although the number of times that the solver needs to run decreases, the difficulty of solving a single input file increases strikingly. As a result, the computation time rises sharply when δ_0 is over 70%.

Table 3 demonstrates that the MaxSAT approach outperforms the SAT one in recovering AES-128 key schedules for a true cold boot attack under the realistic assumption, where the number of reverse flipping errors ranges from 0 to 2. In the following, we carried out two additional tests, as described in case (1) and (2), to estimate the performance of the two approaches under a specific number of reverse flipping errors, shown in Table 4 and Table 5, with the reverse flipping number 1 and 2, respectively.

When there is only one reverse flipping error, the

Table 3 Average runtime of SAT/MaxSAT approaches

δ_0 (%)	CryptoMiniSat (s)	Pwbo2.0 (s)
30	45.81	0.94
40	28.47	0.96
50	19.67	1.17
60	26.52	1.56
70	225.38	12.53
72	678.45	26.78
74	1004.16	231.61
76	1116.35	296.42

MaxSAT approach runs slightly faster than the SAT approach, as indicated in Table 4. In particular, when the decay factor reaches 76%, the solver time of CryptoMiniSat for the worst case is more than 2.9 hours, with the average time at 8 minutes and median time at 3 minutes, respectively. In the MaxSAT approach, the worst case for the recovery time is obtained in 1.8 hours, with the average and median time at 6.4 and 2.4 minutes, respectively.

Table 5 shows the time statistics of the SAT and MaxSAT approaches for the decay factor from 30% to 74%, with exactly 2 reverse flipping errors in each instance. Evidently, the MaxSAT approach is far superior to the other one at all decay factors. In particular, when the decay factor is 74%, solver time of CryptoMiniSat grows averagely to 4 hours with the median time at 2.2 hours. Moreover, nearly 2 days are consumed by recovering the key schedule for the worst case. By contrast, the average and median time for Pwbo2.0 to solve the same set of instances is 36 and 7.9 minutes, respectively, with the worst-case recovery time at 5.7 hours. The low efficiency of the SAT approach is owing to the large number of times for searching the reverse flipping errors, while the MaxSAT solver only runs one time, by excluding the minority errors with optimized algorithms. Moreover, from Table 4 and 5, we can see that St.Dev. of MaxSAT is smaller than that of SAT for most of decay factors. This indicates that MaxSAT approach is not only more time efficient, but also more stable than the SAT approach.

It is worth noting that when recovering the same decayed AES key schedule, the MaxSAT approach generates overwhelmingly larger number of clauses than the other one. To be specific, the number of hard clauses for capturing AES-128 bit-relations is 372,240 for the MaxSAT approach, around 6 times more than that for the SAT approach, which generates only 51,440 hard clauses. The conspicuous gap between the two numbers attributes to the distinct strategies for handling the XOR operation. As an XOR supported SAT solver, CryptoMiniSat prunes redundant clauses dynamically along with the process of assigning Boolean values to variables natively, which considerably decreases the number of clauses introduced for handling XOR formulas. By contrast, to date, there have not been any MaxSAT solvers that could support XOR natively, thus an XOR formula has to be converted to CNF by the direct conversion and cut-up conversion introduced in Sect. 4.2. Unfortunately, both conversions are executed in a static way, in spite that some in-

Table 4 Runtime statistics of SAT/MaxSAT approaches using 100 instances with 1 reverse flipping error

Decay Factor δ_0 (%)		30	40	50	60	70	72	74	76
CryptoMiniSat (s)	Avg.	2.05	1.31	1.97	5.03	43.53	47.60	280.83	480.10
	Med.	1.76	1.17	1.44	2.29	29.02	43.63	67.48	186.01
	Max	5.16	3.94	8.91	78.32	642.39	233.65	3491.27	10498.97
	Min	0.09	0.07	0.11	0.18	1.05	0.58	0.75	2.02
	St.Dev.	1.42	0.97	1.63	8.68	69.66	43.68	646.06	1196.45
Pwbo2.0 (s)	Avg.	1.04	1.09	1.35	2.25	14.95	28.35	122.52	384.35
	Med.	0.79	0.91	1.15	1.94	8.95	13.31	25.22	142.75
	Max	2.37	2.39	3.24	6.35	262.72	599.01	3122.61	6492.83
	Min	0.71	0.70	0.72	0.76	1.02	1.51	2.28	4.34
	St.Dev.	0.45	0.40	0.54	1.31	29.02	61.90	344.66	869.45

Table 5 Runtime statistics of SAT/MaxSAT approaches using 40 instances with 2 reverse flipping errors

Decay Factor δ_0 (%)		30	40	50	60	70	72	74
CryptoMiniSat (s)	Avg.	198.64	162.25	224.69	329.62	3047.82	4909.57	14715.61
	Med.	171.62	186.39	127.72	153.84	2077.35	3517.53	7936.68
	Max	387.54	371.53	1358.53	2112.36	9747.16	17027.60	161389.01
	Min	13.74	8.61	7.30	27.48	38.42	10.44	62.77
	St.Dev.	87.96	89.68	276.60	493.63	2907.31	5077.26	26695.94
Pwbo2.0 (s)	Avg.	1.46	1.56	2.18	4.68	47.73	245.18	2160.49
	Med.	1.02	1.17	1.88	3.17	37.34	97.37	473.76
	Max	7.12	8.22	5.83	19.19	220.87	1848.42	20687.92
	Min	0.23	0.90	0.91	1.00	5.05	6.71	20.05
	St.Dev.	1.11	1.22	1.20	3.94	37.33	358.49	3982.25

Table 6 Average runtime of OMA and IMA

Decay Factor	#RFE = 1		#RFE = 2	
δ_0 (%)	OMA (s)	IMA (s)	OMA (s)	IMA (s)
30	0.78	0.77	1.02	1.00
40	0.82	0.85	1.16	1.17
50	0.99	1.03	1.61	1.62
60	1.59	1.64	4.28	3.92
70	10.84	10.46	38.15	36.04
72	17.44	16.35	151.46	133.19
74	246.09	192.93	1802.97	1638.23
76	358.79	301.60	-	-

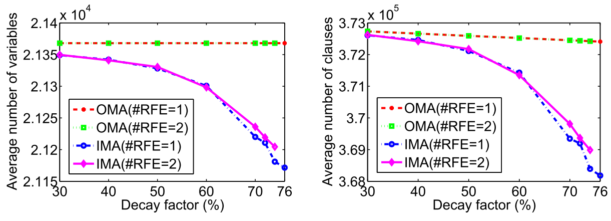


Fig. 1 Average number of variables and clauses

troduced clauses are redundant for the current assignment of variables. Though the XOR-CNF conversions in MaxSAT are not as refined as the strategies adopted by CryptoMiniSat, experimental results still show that the MaxSAT approach excels the SAT one, and we say with assurance that the development of a MaxSAT solver that supports XOR natively would further stretch the advantage of the MaxSAT approach.

5.4 Experimental Results of Improved MaxSAT Approach

This subsection evaluates the improved MaxSAT approach (IMA) developed in Sect. 4.3 and compares the performance of IMA with the original MaxSAT approach (OMA) presented in Sect. 4.2. δ_0 ranges from 30% to 76% for 1 reverse flipping error and from 30% to 74% for 2 reverse flipping errors. Both approaches were tested on a 2.0GHz quad-core Intel i7-4510U processor with 8GB RAM.

As summarized in Table 6, when the decay factor δ_0 is relatively small, i.e., δ_0 is no more than 60%, the performance of two approaches is quite similar. In comparison, when δ_0 goes over 60%, IMA becomes more time efficient than OMA. In particular, when δ_0 reaches 76%, the average runtime with 1 reverse flipping error by OMA is around 358 seconds, while the IMA approach takes merely 300 seconds to recover the same set of corrupted key schedules.

In order to explain the reason why IMA is superior to OMA, we investigate the average numbers of variables and clauses in OMA and IMA, depicted in Fig. 1. Obviously, no matter whether the number of reverse flipping errors is one or two, for solving the same set of instances, both variables and clauses in IMA are fewer than those in OMA. Generally, given unchanged problem difficulty, MaxSAT computation time reduces with the decreased number of variables and clauses. That is why IMA is more time efficient than OMA at fixed δ_0. It is worth noting that though the numbers of variables and clauses decline with the increase of δ_0, the average runtime rises sharply. This is because the growth of decay factor makes the key recovery problem more difficult, heavily influencing the efficiency of problem solving.

6. Conclusion

In this paper, we made the first step in a study of applying SAT and MaxSAT solvers to fulfill the reliable AES key recovery from a true cold boot attack, in which reverse flipping errors account for a small fraction of all 1s in decayed key schedules. Specifically, limited by the bit-relations that are encoded as hard constraints, the SAT approach treats all 1s in the decayed key schedule as hard constraints and search the reverse flipping errors in a bit-by-bit manner. By contrast, the MaxSAT approach encodes all 1s as soft constraints and thus recovers key schedules by satisfying as many soft constraints as possible. In addition, we moved one step further to simplify the key recovery problem by removing leaf variables and formulas that have weak relations to the whole problem, so as to further speed up the recovery process. Experimental results demonstrated that, compared to the SAT approach that searches for the reverse flipping errors exhaustively bit by bit, our MaxSAT approach succeeds in recovering the same set of key schedules within significantly less time, and the improved MaxSAT approach further enhances the efficiency, especially when the decay factor is relatively large.

Acknowledgements

We wish to thank Prof. R. Hasegawa and Prof. H. Fujita for their constructive suggestions and support of our research. This work is partially supported by SWUST Doctorial Research Fund No.15zx7114, No.14zx7147, and JSPS KAKENHI Grant No.25330085. The last author made use of AIST Super Green Cloud in his experiment.

References

[1] J.A. Halderman, S.D. Schoen, N.A. Heninger, W. Clarkson, W. Paul, J.A. Calandrino, A.J. Feldman, J. Appelbaum, and E.W. Felten, "Lest we remember: Cold boot attacks on encryption keys," Proc. USENIX USENIX Security Symposium (SEC'08), California, USA, pp.45–60, July 2008.

[2] N.A. Heninger, "Error correction and the cryptographic key," Master's thesis, USA, 2011.

[3] Federal Information Processing Standards Publication (FIPS 197), "Announcing the advanced encryption standard (aes)," 2001.

[4] A. Tsow, "An improved recovery algorithm for decayed aes key schedule images," Proc. of Selected Areas in Cryptography (SAC'09), pp.215–230, 2009.

[5] A.A. Kamal and A.M. Youssef, "Applications of SAT solvers to aes key recovery from decayed key schedule images," Proc. of International Conference on Emerging Security Information Systems and Technologies (SECURWARE'10), pp.216–220, 2010.

[6] M. Soos, K. Nohl, and C. Castelluccia, "Extending SAT solvers to cryptographic problems," Proc. of International Symposium on the Theory and Applications of Satisfiability and Testing (SAT'09), pp.244–257, 2009.

[7] S.A. Cook, "The complexity of theorem-proving procedures," Proc. 3rd Annual ACM Symposium on Theory of Computing, pp.151–158, 1971.

[8] N. Courtois and G.V. Bard, "Algebraic cryptanalysis of the data encryption standard," Proc. of IMA International Conference on Cryptography and Coding, pp.152–169, 2007.

[9] N. Courtois, G.V. Bard, and D. Wagner, "Algebraic and slide attacks on keeloq," Proc. of International Workshop of Fast Software Encryption, pp.97–115, 2008.

[10] F.D. Garcia, G.K. Gans, R. Muijrers, P. Rossum, R. Verdult, R.W. Schreur, and B. Jacobs, "Dismantling mifare classic," Proc. of European Symposium on Research in Computer Security (ESORICS '08), pp.97–114, 2008.

[11] T. Eibach, E.Pliz, and G. Völkel, "Attacking bivium using SAT solvers," Proc. of International Symposium on the Theory and Applications of Satisfiability and Testing (SAT'08), pp.63–76, 2008.

[12] R.T. Faizullin, I.G. Khnykin, and V.I. Dylkeyt, "The SAT solving method as applied to cryptographic analysis of asymmetric ciphers," The Computing Research Repository (CoRR), vol.abs/0907.1755, 2009.

[13] C. Patsakis, "RSA private key reconstruction from random bits using SAT solvers," IACR Cryptology ePrint Archive (IACR), vol.26, 2013.

[14] I. Mironov and L. Zhang, "Applications of SAT solvers to cryptanalysis of hash functions," Proc. of International Symposium on the Theory and Applications of Satisfiability and Testing (SAT'06), pp.102–115, 2006.

[15] E. Homsirikamol, P. Morawiecki, M. Rogawski, and M. Srebrny, "Security margin evaluation of sha-3 contest analists through SAT-based attacks," Computer Information Systems and Industrial Management, vol.7564, pp.56–67, 2012.

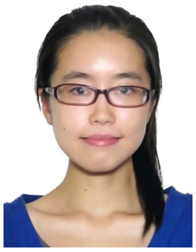
[16] A. Biere, M. Heulu, H. Maaren, and T. Walsh, Handbook of Satisfiability, IOS Press, 2009.

[17] X. Zhang and K.K. Parhi, "High-speed vlsi architectures for the aes algorithm," IEEE Trans. Very Large Scale Integration (VLSI) Systems, vol.12, no.9, pp.957–967, 2004.

[18] X. Liao, H. Zhang, M. Koshimura, H. Fujita, and R. Hasegawa, "Using maxsat to correct errors in aes key schedule images," Proc. of International Conference on Tools with Artificial Intelligence (ICTAI'13), pp.284–291, Nov. 2013.

[19] G. Bard, N. Courtois, and C. Jefferson, "Efficient methods for conversion and solution of sparse systems of low-degree multivariate polynomials over gf(2) via sat-solvers," IACR Cryptology ePrint Archive (IACR), vol.24, 2007.

[20] B. Chen, "Strategies on algebraic attacks using sat solvers," Proc. of International Conference for Young Computer Scientists (ICYCS'08), pp.2204–2209, Nov. 2008.



Xiaojuan Liao received B.E. degree from University of Electronic Science and Technology of China in 2009 and received D.E. degree from Kyushu University in 2014. She has been a Lecturer in Southwest University of Science and Technology since 2014. Her research interests include maximum satisfiability and its applications.

Hui Zhang received B.E. degree from University of Electronic Science and Technology of China in 2006 and received D.E. degree from Kyushu University in 2014. He has been a Lecturer in Southwest University of Science and Technology since 2014. His research interests include cryptographic algorithms and cryptanalysis.

Miyuki Koshimura received B.S. and M.S. degrees from University of Tsukuba in 1984 and 1986, respectively. He received D.E. degree from Kyushu University in 2002. He has been an Assistant Professor in Kyushu University since 2009. He was a research associate of Kyushu University from 1995 to 2009. His research interests include automated reasoning and its applications.