Cloud-oriented SAT Solver based on obfuscating CNF

Ying Qin, ShengYu Shen, and Yan Jia

National University of Defense Technology, School of Computer Science, ChangSha, China
http://www.nudt.edu.cn

Abstract. Propositional satisfiability (SAT) has been widely used in hardware and software verification. Outsourcing complex SAT problem to Cloud can leverage the huge computation capability and flexibility of Cloud. But some confidential information encoded in CNF formula, such as circuit structure, may be leaked to unauthorized third party.
In this paper, we propose a novel Cloud-oriented SAT solving algorithm to preserve privacy. **First**, an obfuscated CNF formula is generated by embedding a Husk formula into the original formula with proper rules. **Second**, the obfuscated formula is solved by a state-of-the-art SAT solver deployed in Cloud. **Third**, a simple mapping algorithm is used to map the solution of the obfuscated formula back to that of the original CNF formula.
Theoretical analysis and experimental result show that our algorithms can significantly improve security of the SAT solver with linear complexity while keeping its solution space unchanged.

Keywords: SAT solver; CNF formula; Privacy; Obfuscating; Cloud computing

1 Introduction

Propositional satisfiability [1] (SAT) has been widely used in hardware and software verification [2][3]. With the rapid increase of the hardware and software system size, the size of SAT problem generated from verification also increases rapidly.

On the other hand, Cloud computing can provide elastic computing resource, which make outsourcing hard SAT problem to public Cloud [19][20] very attractive. However, unauthorized access to outsourced data prevent it from widely deployed.

In formal verification, circuit, code and properties will first be converted into CNF (Conjunctive Normal Form) formula by Tsentin coding[4] before SAT solving. After that, circuit structure and other confidential information are still existed in CNF formula. To prevent unauthorized user from accessing these confidential information, it is necessary to obfuscate CNF formula before outsourcing.

This paper presents a novel Cloud SAT solver based on Obfuscating CNF. **First**, the original CNF formula S_1 is obfuscated into another formula S by embedding another CNF formula S_2 with unique solution. And the embedding rules guarantee that S's graph structure are significantly different from that of S_1. **Second**, S is sent to a SAT solver in Cloud, which returns a solution R_O. **Third**, the solution R of S_1 is obtained from the solution of R_O by projection. its correctness can be guaranteed by the embedding rules in the first step.

The major contributions of this paper are: **First**, by obfuscating, confidential information in the original CNF formula, such as circuit structure, will be destroyed in the obfuscated CNF formula; **Second**, the obfuscated CNF formula can be solved by state-of-the-art SAT solver. **Third**, the theoretical analysis and experiments shows that obfuscating and solution recovery algorithms linear complexity, reducing the impact on the overall performance of SAT solving.

The remainder of this paper is organized as follows. Background material is presented in Section 2. Section 3 gives the description of the problem, while the implementation of Cloud SAT solver based on obfuscating is presented in Section 4. Section 5 analyzes correctness, effectiveness and complexity; Section 6 describes the related work; Section 7 gives the experimental results, while Section 8 concludes this paper.

2 Preliminaries

2.1 SAT solving

The Boolean value set is denoted as $B = \{T, F\}$. For a Boolean formula S over a variable set V, the propositional satisfiability problem (abbreviated as SAT) is to find a satisfying assignment $A : V \to B$, so that S evaluates to T. If such a satisfying assignment exists, then S is satisfiable; otherwise, it is unsatisfiable. A clause subset of an unsatisfiable formula is an unsatisfied core. A computer program that decides the existence of a satisfying assignment is SAT solver[10].

Normally, a SAT solver requires the formula to be conjunctive normal form (CNF), in which a formula is a conjunction of its clause set, and a clause is a disjunction of its literal set, while a literal is a variable or its negation.

Φ in Equation (1) is a CNF formula with four variables x_1, x_2, x_3, x_4, and four clauses $x_1 \vee \neg x_2$, $x_2 \vee x_3$, $x_2 \vee \neg x_4$, $\neg x_1 \vee \neg x_3 \vee x_4$. Literal x_1 is positive literal of variable x_1 in clause $x_1 \vee \neg x_2$, while $\neg x_2$ is a negative literal.

$$\Phi = (x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (x_2 \vee \neg x_4) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \tag{1}$$

The number of literals in clause C is denoted as $|C|$. The number of clauses in a CNF formula F is denoted as $|F|$. For example $|x_1 \vee \neg x_2| \equiv 2$, while $|\phi| \equiv 4$

2.2 Tseitin encoding

In hardware verification, circuits and properties are converted into CNF formula by Tseitin encoding[4], and then CNF formula is solved by SAT solver. Circuits can all be expressed by a combination of gate AND2 and INV, so we only lists Tseitin encoding of gate AND2 and INV here.

For gate INV $z = \neg x$, its Tseitin encoding is $(x \vee z) \wedge (\neg x \vee \neg z)$. For gate AND2 $z = x_1 \wedge x_2$, its CNF formula is $(\neg x_1 \vee \neg x_2 \vee z) \wedge (x_1 \vee \neg z) \wedge (x_2 \vee \neg z)$. For a complex circuit C expressed by a combination of AND2 and INV, its Tseitin encoding $Tseitin(C)$ is a conjunctive of all these gates' Tseitin encoding.

For a circuit C with an INV $d = \neg a$ and an AND2 $e = d \wedge c$, its Tseitin encoding is shown in Equation (2).

$$Tseitin(C) = \left\{ \begin{array}{c} (a \vee d) \\ \wedge\ (\neg a \vee \neg d) \end{array} \right\} \wedge \left\{ \begin{array}{c} (\neg e \vee c) \\ \wedge \quad (\neg e \vee d) \\ \wedge\ (e \vee \neg c \vee \neg d) \end{array} \right\} \tag{2}$$

3 Threat Model of Cloud-based SAT solving

Solving SAT in Cloud includes three steps: **first** generating and uploading CNF formula to Cloud, **then** solving CNF in Cloud, **finally** downloading solution. Thus, unauthorized access[11] to CNF formula in Cloud may result in leakage of confidential information.

On the other hand, verifying the result of SAT is simple. If CNF formula is satisfiable, the solution can be substituted into CNF to check if it is satisfiable; If CNF formula is unsatisfiable, an unsatisfiable core returned by the SAT solver can be verified by resolution. In both case, verifying the result are linear complexity.

Thus, in this paper we assume that Cloud servers are honest but curious. That is, the CNF will be solved correctly, but also may be analyzed to recover confidential information. Algorithms [6–9] have been proposed to recover circuit from CNF. Before discussing them, some concepts should be introduced first.

Definition 1 (CNF signature). *CNF signature of gate g is its Tseitin encoding $Tseitin(g)$. Each clause in CNF signature is called characteristic clause. A characteristic clause containing all variables in CNF-signature is a **key clause**. Variables correspond to output of a gate is called **output variable**.*

For AND2 in Equation (2), $\neg e \vee c$ is a characteristic clause. Clause $e \vee \neg c \vee \neg d$ is a key clause. e is an output variable.

With such encoding rules, gates with the same CNF signature will be encoded into the same set of clauses. Potential attackers can exploit structural knowledge to recover the circuit structure. Some such algorithms[6–9] are based on concept of directed hyper-graph and bipartite graph described below.

Definition 2 (Hypergraph of CNF and Directed Hypergraph of CNF). *A Hypergraph $G = (V, E)$ of a CNF formula Σ is*

1. *each vertex of V corresponds to a clause of Σ;*
2. *each edge $(c_1, c_2) \in E$ corresponds to two clauses c_1 and c_2 containing the same variable or its negation;*
3. *each edge is labeled by the variable;*

A Directed Hypergraph is a Hypergraph with each endpoint of edge labeled by $\uparrow$ *when clause contains non-negative variable, or* $\dagger$ *when clause contains negative variable.*

Definition 3 (Bipartite graph of CNF). *A Bipartite graph* $G = (V, E)$ *of a CNF formula* Σ *is*

1. *each vertex of* V *corresponds to clause and variable of* Σ*;*
2. *each edge* $(c, v) \in E$ *corresponds to a clauses* c *that contains variable* v*;*
3. *each edge is labeled by* $\uparrow$ *when clause contains non-negative variable, or* $\dagger$ *when clause contains negative variable.*

With these definitions, Roy et al. [8] first converts the CNF to an Hypergraph G, and then matches the CNF signatures of all types of gates in G to recover gates by subgraph isomorphism, finally creates a maximal independent set (MIS) instance to represent the recovered circuit.

Fu et al.[9] presents another algorithm that first detects all possible gates with pattern matching, and then constructs a maximum acyclic combinational circuit by selecting a maximum subset of matched gate.

In Cloud computing paradigm, attacker may use these algorithms to recover the circuit structure from CNF formula. Therefore, it is essential to prevent information leakage before outsourcing CNF formula to Cloud.

4 Cloud-based SAT solver preserving privacy

In this paper, we present a Cloud-based SAT solver based on obfuscating, which prevent information leakage by hiding the structure in CNF. The obfuscating algorithm is based on the following three facts:

1. First, changing CNF signature of gates in CNF formula will make circuit recovering based on pattern matching or subgraph isomorphism impossible.
2. Second, current SAT solver with conflict analysis [10] is very efficient. So we would like to use them directly, instead of developing a new one like [11].
3. Third, the solution of obfuscated CNF formula should be easily mapped back to original formula.

The proposed algorithm is based on the following definition.

Definition 4 (Husk formula). *Husk formula is a CNF formula with a unique solution, and assignment of variables is non-uniform, that is, not all 0 or all 1.*

The overall framework of the Cloud SAT solver is shown intuitively in Figure 1. In Step 1, GENERATOR algorithm generates a husk formula with unique solution R_H; In Step 2, OBFUSCATOR algorithm obfuscates S_1 to obtain a new CNF formula S; In Step 3, S is solved in Cloud; In Step 4, MAPPER algorithm maps solution of S to that of S_1.

The GENERATOR, OBFUSCATOR and MAPPER algorithm will be described in Subsection 4.1,4.2 and 4.3 respectively.

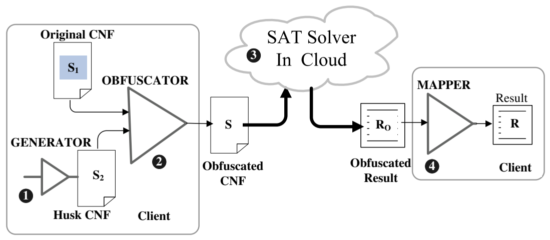


Fig. 1. The Cloud SAT solver based on obfuscating CNF

4.1 Generating Husk formula

Husk formula is constructed based on prime factorization method: **First**, given a prime p represented by a binary vector $p =< p_1, p_2, \dots, p_n >$, we assigning its square p^2 to the output of a multiplier M, with constraint $p \neq 1$. **Second**, We convert the multiplier M into CNF formula $Tseitin(M)$.

To satisfy $Tseitin(M)$, the two inputs of M must all be $p =< p_1, p_2, \dots, p_n >$, which makes the p the unique solution of $Tseitin(M)$.

With these discussion, GENERATOR algorithm to generate Husk formula is shown in Algorithm 1.

4.2 Constructing obfuscated formula

The proposed OBUFSCATOR algorithm generates a new CNF formula S by embedding Husk formula S_2 into the original formula S_1 with proper rules. By adding new clauses and new literals, OBUFSCATOR algorithm changes the clause set and literal set in clauses of S_1, to prevent its structure from being recovered.

In order to keep solution space unchanged, when inserting variables of S_2 into clauses of S_1, the following rules must be followed, as depicted in Line 5 and 18 of OBFUSCATOR algorithm: Variables assigned F in R_H will be inserted as positive literals; While variables assigned T will be inserted as negative literals.

Algorithm 1: GENERATOR

```
  Data: NULL
  Result: Husk CNF $S_2$ and Husk result $R_H$
1 begin
2     Generating a prime number $p$ ;
3     $\Phi = M(I_1 \neq 1, I_2 \neq 1, O = p^2)$ ;
4     $S_2 = Tseitin(\Phi)$ ;
5     $R_H = p \mid p$ ;
6 end
```

```
Algorithm 2: OBFUSCATOR
Data: The original CNF $S_1$,Husk CNF $S_2$,Husk result $R_H$
Result: The obfuscated CNF $S$, variable mapping $M$
1  begin
2      $mark(S_1)$;
3      foreach $c \in S_1$ do
4          lit =get literal $\in R_H$;
5          $c = c \cup \neg lit$;
6          if $c \in$ Key Clause Set KCS then
7              $nc = generate\_new\_clause(c, lit)$;
8              $S_2 = S_2 \cup nc$;
9          end
10     end
11     foreach $c \in S_1$ do
12         $averagelen = \frac{\sigma_{c' \in S_1} |c'|}{|S_1|}$ ;
13         while $|c| < averagelen$ do
14             $lit$ =get literal $\in R_H$ ;
15             while $\neg lit \in c$ do
16                 lit=get literal $\in R_H$ ;
17             end
18             $c = c \cup \neg lit$;
19         end
20         $M$ =remap all variable in $S_1 \cup S_2$ ;
21         $S$ =reorder all clause in $S_1 \cup S_2$ ;
22     end
23 end
```

Thus, S and S_1 can be solved with the same SAT solver, and the solution of S_1 can be extracted from that of S by projection on variables set of S_1. More details of OBUFSCATOR algorithm is presented in Algorithm 2.

In algorithm 2, Procedure **mark** at Line 2 marks key clauses and output variables of some kind of gate in CNF formula. Procedure **generate_new_clause** at Line 7 generates some new clauses matching the key clauses, such that identifying gates become more difficult.

Different mark algorithms are needed for different gate types with different CNF signatures. But their complexity are of the same. As all circuits can be represented by a combination of AND2 and INV, and the *mark* algorithm for INV is trivial, so we only present the implementation of *mark* for AND2 in Algorithm 3.

Similarly, we also only present the implementation of **generate_new_clause** for AND2 in Algorithm 3.

Algorithm 3: mark and **generate_new_clause**

```
 1 mark;
   Data: CNF formula $S$
   Result: marked $S$
 2 begin
 3     foreach $(C \in S)$ & $(|C| \equiv 3)$ do
 4         foreach $l \in C$ do
 5             foreach $(C_1 \in S)$ & $(\neg l \in C_1)$ & $(|C_1| \equiv 2)$ do
 6                 foreach $l_1 \in C_1$ do
 7                     if $(\neg l_1 \in C)$ & $(l_1 \neq l)$ then
 8                         $match + +$ ;
 9                 end
10             end
11         end
12     end
13     if $match \equiv 2$ then
14         mark $L$ as output literal ;
15         mark $C$ as key clause ;
16     end
17 end
18 generate_new_clause;
   Data: key clause $C$ in AND2, Husk literal $lit$
   Result: new clause $C_1$
19 begin
20     $olit$=Getting output literal from $C$ ;
21     $C_1 = lit \cup \neg olit$ ;
22 end
```

4.3 Recover the original solution

The variables set of S is a superset of S_1. Therefore, to get solution of S_1, we need to filter assignment of variables in S_1 from R_O according to the variable name mapping table M. R_O is the solution of S returned by SAT solver in Cloud. This is implemented by MAPPER in Algorithm 4.

5 Correctness, effectiveness and performance analysis

5.1 Correctness proof

The Correctness means that the algorithm should keep the solution space unchanged, that is, for CNF formulas S_1 and its obfuscated formula S:

1. If S_1 is unsatisfied iff S is unsatisfied. And the unsatisfied core of S_1 can be obtained from unsatisfied core of S by deleting literal in S_2;
2. If S_1 is satisfied iff S is satisfied. And the solution of S_1 can be obtained by projecting solution of S into variables set of S_1 .

Algorithm 4: MAPPER

```
Data: Obfuscated result $R_O$, variable mapping table $M$, Husk result $R_H$
Result: Result $R$
begin
    foreach $lit \in R_H$ do
        $var = abs(lit)$;
        $rvar = M[var].var$;
        if $M[var].S \equiv S_1$ then
            $R[rvar] = lit > 0?rvar : \neg rvar$ ;
        else
            $Hlit = lit > 0?rvar : \neg rvar$;
            if $Hr[rvar] \neq Hlit$ then
                printf(something wrong with CloudSAT solver);
        end
    end
end
```

OBFUSCATOR algorithm is simplified to the follows rules.

1. **Rule 1**: For each clause $c \in S_1$, we take one or more variables from S_2, and insert them into c according to the following rule: if variable is T in R_H, insert its negative literal; if variable is F in R_H, insert its positive literal. The resulting clause set is denoted as S_3.
2. **Rule 2**: We generate new clauses with literals from R_H and output variables in S_1 according to the following rule: if variable is T in R_H, insert positive literal into clause; if variable is F in R_H, insert negative literal into clause; Literal of output variable is extracted directly from the key clause and inverted. New clauses set generated in this way is denoted as S_4.
3. **Rule 3**: Combining and randomly reordering S_2,S_3 and S_4 to produce S.

Definition 5 (Original variable). *Variable and clause in original CNF formula is denoted* ***original variable*** *and* ***original clause****. Variable and clause in Husk formula is called* ***Husk variable*** *and* ***Husk clause****. Literal of Husk result is called* ***Husk literal****.*

Husk formula's solution is unique. Let the result be $R_H = \{b_1, b_2, \ldots, b_m\}$. S_2 can be satisfied only when assigning R_H to it.

According to Rule 1, each clause in S_3 can be expressed as $C = A \vee B$, where A is clause from S_1 , and $B = B_i$, $B_i = z_i \vee B_{i-1}$, $B_1 = z_1$. If $b_j \equiv F$, then $z_i = y_i$. if $b_i \equiv T$, then $z_i = \neg y_i$. With constrains of Husk clause, Husk variable must be assigned $H = b_1, b_2, \ldots, b_m$, then z_i must be F. So $B = B_i = z_i \vee z_{i-1} \vee \cdots \vee z_1$ must be F. This means $C = A \vee B$, whose value is decided by A.

Clauses in S_4 consist of Husk literals and output variables from S_1. These clauses can be represented as $C = z_i \vee A$. If $b_i \equiv F$, then $z_i = \neg y_i$, otherwise $z_i = y_i$. With constraint from Husk clauses, Husk variables must be assigned as following: $R_H = \{b_1, b_2, \ldots, b_m\}$. Thus $z_i = T$,$C = z_i \vee A = T$. So clause C will not constrain the assignment of variables in A.

Theorem 1. *With the following hypothesis and facts:*

1. *Clause $A = a \vee X$ and clause $B = b$, while $b \notin X$;*
2. *Let $S_1 = A$, $S_2 = B$,then $R_H = \{(T) \times (b)\}$;*
3. *With R_H and Rule 1, we have clause $C = A \vee \neg b$ and $S_3 = C$;*
4. *With R_H and Rule 2, with literal $a \in A$, we have clause $D = a \vee b$ and $S_4 = D$;*
5. *let $S = S_2 \wedge S_3 \wedge S_4$ and $S = B \wedge C \wedge D$.*

we can prove $S = S_1 \wedge S_2$.

Proof. We have

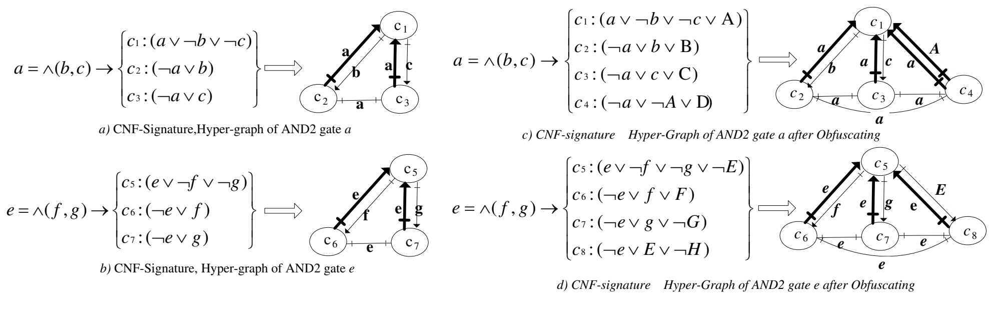
$$\begin{array}{ccl}
S = B \wedge C \wedge D & C = A \vee \neg b & \models \\
S = b \wedge (A \vee \neg b) \wedge D & & \models \\
S = b \wedge A \wedge D & D = a \vee b & \models \\
S = b \wedge A \wedge (a \vee b) & & \models \\
S = b \wedge A & B = b & \models \\
S = B \wedge A & S_1 = A\ ,\ S_2 = B & \models \\
S = S_1 \wedge S_2 & &
\end{array} \tag{3}$$

□

5.2 Effectiveness analysis

According to Section 3, analyzing hyper-graph and bipartite graph is major approach to recover circuit structure. To show the effectiveness of our algorithm, we present qualitative analysis of its changes to hyper-graph and bipartite graph.

Figure 2a) and 2b) shows the CNF signature and hyper-graph of two AND2 gate a and e. While their CNF signature and hypergraph after obfuscating are shown in Figure 2c) and 2d). There are four types of changes:



$a = \wedge(b,c) \rightarrow$ $c_1 : (a \vee \neg b \vee \neg c)$, $c_2 : (\neg a \vee b)$, $c_3 : (\neg a \vee c)$
a) CNF-Signature,Hyper-graph of AND2 gate a
$a = \wedge(b,c) \rightarrow$ $c_1 : (a \vee \neg b \vee \neg c \vee A)$, $c_2 : (\neg a \vee b \vee B)$, $c_3 : (\neg a \vee c \vee C)$, $c_4 : (\neg a \vee \neg A \vee D)$
c) CNF-signature Hyper-Graph of AND2 gate a after Obfuscating
$e = \wedge(f,g) \rightarrow$ $c_5 : (e \vee \neg f \vee \neg g)$, $c_6 : (\neg e \vee f)$, $c_7 : (\neg e \vee g)$
b) CNF-Signature, Hyper-graph of AND2 gate e
$e = \wedge(f,g) \rightarrow$ $c_5 : (e \vee \neg f \vee \neg g \vee \neg E)$, $c_6 : (\neg e \vee f \vee F)$, $c_7 : (\neg e \vee g \vee \neg G)$, $c_8 : (\neg e \vee E \vee \neg H)$
d) CNF-signature Hyper-Graph of AND2 gate e after Obfuscating

Fig. 2. CNF signature and Hyper-Graph of a and e before and after obfuscating

1. First, the length of key clauses c_1 and c_5 are changed from 3 to 4, this defeats structure detection techniques [9] based on pattern matching key clause;
2. Second, CNF signatures of a and e are changed. This defeats structure detection techniques[8] based on sub-graph isomorphic;
3. Third, there are new clauses added in formula, such as c_4 and c_8, this change also defeats structure detection techniques[8] based on sub-graph matching;
4. Furthermore, by inserting proper literal in key clauses and generating new clause, CNF signature of instance e is changed from AND2 to AND3, as shown in Figure 2b). Husk variable E, which becomes a input variable of gate AND3, is indistinguishable with f and g, which are original input variables of AND2. this makes it impossible to distinguish AND2 and AND3.

The changes to bipartite graphs of a and e is similar.

5.3 Complexity of the algorithm analysis

The main procedure of OBFUSCATOR algorithm consists only one layer of loop. Although **mark** consists 4 layers of loop, the runtimes of the 3 inner loops are bounded by length of clauses. So the complexity of the OBFUSCATOR algorithm is $O(n)$. The complexity of the MAPPER algorithm is obvious $O(n)$.

6 Related works

Secure Computation Outsourcing based on encryption: R. Gennaro et al.[13] presented the concept of verifiable computation scheme, which shows the secure computation outsourcing is viable in theory. But the extremely high complexity of FHE operation and the pessimistic circuit sizes make it impractical. Zvika et al.[12] constructed an obfuscated program for d-CNFs that preserves its functionality without revealing anything else. The construction is based on a generic multi-linear group model and graded encoding schemes, along with randomizing sub-assignments to enforce input consistency. But the scheme incurs large overhead caused by their fundamental primitives.

Secure Computation Outsourcing based on disguising: For linear algebra algorithms, Atallah et al. [15] multiplied data with random diagonal matrix before outsourcing. and recovered results by reversible matrix operations. But it had not discuss how to verify the result. Paper [16] discussed secure outsourcing of numerical and scientific computation, by disguising with a set of problem dependent techniques. C.Wang[5] presented securely outsourcing linear programming(LP) in Cloud, by explicitly decomposing LP computation into public LP solvers and private data, and provide a practical mechanism which fulfills input/output privacy, cheating resilience, and efficiency.

Verifiable computation delegation: Verifiable computation delegation is the technique to enable a computationally weak customer to verify the correctness of the delegated computation results from a powerful but untrusted server without investing too much resources. To prevent participants from keeping the rare events, Du. et al. [14] injected a number of chaff items into the workloads so as to confuse dishonest participants. Golle et al. [17] proposed to insert some pre-computed results (images of ringers) into the computation workload to defeat untrusted or lazy workers. Szada et al. [18] extended the ringer scheme and propose methods to deal with cheating detection.

7 Experiments

Algorithms presented in this paper are all implemented in language C. The experiments is conducted on a laptop with Intel Core(TM) i7-3667U CPU @ 2.00GHz, 8GB RAM. We unroll circuits in iscas89 benchmark for 30 times and transform them into CNF formulas, and generate Husk formula with variables number $vn = 675$ and clauses number $cn = 2309$, and then obfuscate the CNF formula. Figure 3 presents size of CNF formula after obfuscating with vn and cn, SAT Solver time(SAT Time) before and after obfuscating(CloudSAT Time), obfuscating time(Obfuscating Time), and result recovery time(Mapping time).

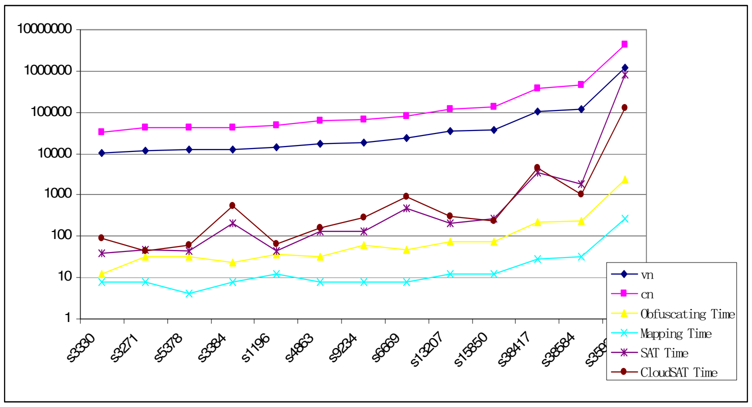


Fig. 3. Relationship between Runtime and Size of CNF

Experiments show that, although the size of CNF formula(cn and vn) after obfuscating is increased, SAT solver time does not follow a linear growth after obfuscating. When the original CNF formula is much larger than the Husk formula, SAT Solver time is likely to be the same between before and after obfuscating. At the same time, obfuscating time and result recovering time increase linearly with the size of circuit (variables number and clauses number).

8 Conclusion

This paper propose the first Cloud SAT solver algorithm that can prevent the confidential information in the CNF formula from being recovered by untrusted

Cloud server. Theoretical analysis and experimental result show that our algorithms can significantly improve security of the SAT solver with linear complexity while keeping its solution space unchanged.

References

1. D.Putnam. A Computing Procedure for Quantification Theory. Journal of the ACM 7 (3): 201.
2. G. Hachtel, F. Somenzi. Logic synthesis and verification algorithms. Springer 2006: I-XXIII, 1-564.
3. E. Clarke, O. Grumberg, S. Jha, Y. Lu, H. Veith. Counterexample-Guided Abstraction Refinement. CAV'00, pp 154-169.
4. G. Tseitin. On the complexity of derivation in propositional calculus. Studies in Constr. Math. and Math. Logic, 1968.
5. C. Wang, K. Ren, J. Wang. Secure and practical outsourcing of linear programming in Cloud computing. INFOCOM'11: 820-828.
6. C. Li. Integrating equivalency reasoning into Davis-Putnam procedure. in AAAI'00, pp. 291-296.
7. R. Ostrowski. Recovering and exploiting structural knowledge from cnf formulas, CP'02, pp. 185-199.
8. J.Roy, I.Markov,V. Bertacco. Restoring Circuit Structure from SAT Instances IWLS'04, pp. 361-368.
9. Z. Fu, S. Malik. Extracting Logic Circuit Structure from Conjunctive Normal Form Descriptions. VLSI Design'07, pp 37-42.
10. MiniSat SAT Algorithms and Applications Invited talk given by Niklas Srensson at the CADE-20 workshop ESCAR. http://minisat.se/Papers.html
11. H. You. Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds, CCS'09
12. Z. Brakerski and G. Rothblum. Black-Box Obfuscation for d-CNFs. ITCS'14,pp. 235-250.
13. R. Gennaro, C. Gentry, B. Parno. Non-interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers. CRYPTO'10, pp 465-482.
14. W. Du and M. Goodrich. Searching for High-Value Rare Events with Uncheatable Grid Computing.2004.
15. M. Atallah, K. Pantazopoulos, J. Rice, E. Spafford. Secure outsourcing of scientific computations. Advances in Computers 54: 215-272 (2001)
16. M. Atallah, J. Li. Secure outsourcing of sequence comparisons. Int. J. Inf. Sec. 4(4): 277-287 (2005).
17. P. Golle and I. Mironov. Uncheatable distributed computations. CT-RSA'01, pp. 425440.
18. D. Szajda, B. G. Lawson, and J. Owen. Hardening functions for large scale distributed computations. ISSP'03, pp. 216-224.
19. Paralleling OpenSMT Towards Cloud Computing http://www.inf.usi.ch/urop-Tsitovich-2-127208.pdf
20. Formal in the Cloud OneSpins New Spin on Cloud Computing.http://www.eejournal.com/archives/articles/20130627-onespin/?printView=true