Cloud-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. With the emerging cloud computing paradigm, it becomes increasingly motivated to outsource complex SAT problem to the commercial public cloud for larger computation demand and greater flexibility. But outsourcing SAT solving to cloud also bring in new security challenge, that is, some confidential information encoded in CNF formula, such as information of 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 CNF formula with proper rules. **Second**, the obfuscated CNF 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; Obfuscate; 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, the size of SAT problem generated from verification also increases rapidly.

On the other hand, cloud computing paradigm can provide elastic computing resource to meet the workload demand of different application. It becomes increasingly motivated to outsource complex SAT solving procedure from local sites to the commercial public cloud [23][24]. However, when outsourcing SAT solvingcomputing data shall be send into remote server which is shared among different client. Threat of information leakage resulted from authorized access to outsourcing data make an obstacle to widely adopt the new computing paradigm.

In formal verification, circuit, code and property will be converted into CNF (Conjunctive Normal Form) formula through Tsentin[4] coding before SAT solving. After Tsentin encoded, circuit structure and other sensitive information are still existed in CNF formula. To prevent unauthorized user get sensitive information from CNF formula, it is necessary to obfuscate CNF formula before outsourcing into cloud.

This paper presents a novel Cloud-SAT-solver based on Obfuscating CNF. **First**, the original CNF formula S_1 is obfuscated into a new CNF formula S, by embedded through proper rules with a CNF formula S_2 which has unique solution, embedding rules guarantee S has different graph structure compared with S_1. **Second**,S is sent to SAT-solver Ω which deployed in Cloud, Ω generates Solution of R_O and send it back. **Third**, the solution R is obtained from the solution of R_O by projection, the correctness is guaranteed by embedding rules in first step.

The advantage of this method is that: firstly, through obfuscation, sensitive information in original CNF formula, such as circuit structure, will not be available in Obfuscated CNF formula which is outsourced into cloud; secondly, Obfuscated CNF formula can be solved by SAT solver without any modification. Finally, the theoretical analysis and experiments shows that obfuscating and solution recovery algorithms linear complexity, reducing the impact on the overall performance of SAT solving.

In the following sections: Chapter II describes the background; Chapter III gives the description of the problem; Chapter IV gives the implementation of cloud-SAT-solver based on obfuscation; Chapter V analyzes correctness, effectiveness and algorithm complexity of the obfuscation algorithm; Chapter VI describes the related work; Chapter VI gives the experimental results; Chapter VII summarizes.

2 Background

2.1 Preliminaries

A problem handled by SAT solver is usually specified in a conjunctive normal form (CNF) formula of propositional logic. A CNF formula is represented using Boolean variables that can take the values F (false) or T (true). Clauses are disjunction of literals, which are either a variable or its complement, and a CNF formula is a conjunction of clauses.

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

Φ is a typical example of CNF formula, which contains four variables x_1, x_2, x_3, x_4, and four clauses $x_1 \vee \neg x_2$, $x_2 \vee x_3$, $x_2 \vee \neg x_4$, $\neg x_1 \vee \neg x_3 \vee x_4$. In clause $x_1 \vee \neg x_2$, there are two literals x_1 and $\neg x_2$. literal x_1 is called positive literal of variable x_1. while $\neg x_2$ is called negative literal of variable x_2.

The number of literals in clause C is length of C, denoted as $|C|$.For example $|x_1 \vee \neg x_2| = 2$

Solving SAT problem is to determine if there exists an assignment to the variables of CNF formula so that the formula can be satisfied. CNF formula is satisfied SAT if and only if each clause in the CNF formula takes value 1, that means at least one literal in the clause takes value 1. If there does not exist assignment to the variables so that the formula can be satisfied, CNF formula is UNSAT, whereas Clauses which is conflicted with each other is called unsatisfied core.

2.2 Tseitin coding

In hardware verification, Circuits and property are converted into CNF formula through Tseitin coding[4],then CNF formula is handled by SAT solver. Every circuit can be expressed by combination of gate AND2 and NOT, so here lists Tseitin coding of them.

For gate AND2 $z = NOT(x)$, its CNF formula generated by Tseitin encoding, is $(x \vee z) \wedge (\neg x \vee \neg z)$. While for gate AND2 $z = AND2(x_1, x_2)$, 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 other type gate, its according CNF formula can be generated through Tseitin encoding. For a complex circuit consists of basic gate, such as AND2 and NOT, the CNF formula of the circuit is conjunctive of formulas of basic gate.

As illustrated in Fig. 1, Fig 1a) circuit consists of a gate of AND and a gate of OR. Through Tseitin encoding, CNF formula of 1a) circuit is expressed as conjunctive of clauses in Fig.1b) and Fig.1c). Formally, for any circuit C, it can

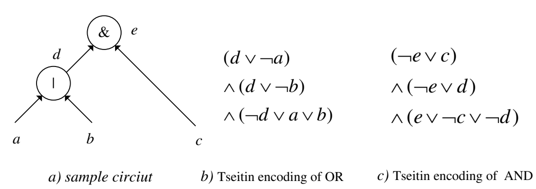


Fig. 1. Tseitin form of circuit

be converted into CNF formula ψ through Tseitin, note as ψ=Tseitin(C).

This paper focuses on obfuscation of CNF formula generated from circuit in formal verification, the obfuscation algorithm is also suitable for CNF formula generated from software code.

3 Problem Definition

3.1 Model of SAT solving in cloud

In cloud computind paradigm, there is three step involved in verification oriented SAT solving[24]. As illustrated in Fig. 2.

1. Upload computing data: User converts circuit into CNF formula and upload to cloud server through client.

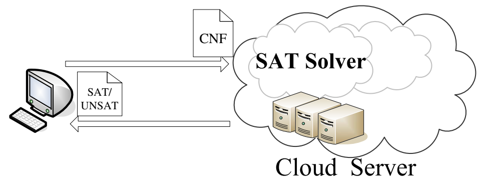


Fig. 2. SAT solving in cloud

2. SAT solve: SAT solver deployed in cloud server handles CNF formula, and gives solution.If CNF formula is SAT, solver will give an assignment of the variables; While if CNF formula is UNSAT, solver will give an unsatisfied core.
3. Download solution: User get solution from cloud through client.

3.2 Threat Model

In cloud computing paradigm, CNF formula will be uploaded and handled in public cloud; Cloud are multi-tenant, unauthorized access[11] to CNF formula may result in leakage of sensitive information.

On the other hand,result verification of SAT problem is simple: if CNF formula is SAT, just check whether CNF formula is satisfied under the solution given by cloud; while if CNF formula is UNSAT, just get unsatisfied core from cloud. Result verifications for both conditions are linear complexity.

According to facts listed above, in this paperwe assume that cloud computing servers are honest but curious: cloud servers will complete the SAT solving tasks correctly, but CNF formula may be analyzed to obtain additional information, such as all or part of the circuit structure.

The information of circuit structure is not lost during encoding from circuit to CNF. Li [6] and Ostrowski [7] discuss the techniques to extract circuit structures from CNF formula. Furthmore , Roy [8] and Fu [9] present algorithm of CNF-to-circuit decoding to recover circuit structure from CNF formula.

Before we discuss algorithms of CNF-to-circuit decoding, concepts in algorithms are introduced first.

Definition 1 (CNF signature). *A CNF signature of a logic gate is a CNF formula representing the characteristic function of the gate. Clause in CNF signature is called characteristic clause. If a characteristic clause contains all variables in CNF-signature, the clause is named as* **key clause**. *Variable represents Output of a logical gate is called* **output variable**.

Take gate AND as an example, after Tseitin encoding , three input AND gate (AND3) is converted into the set of clauses C as shown in figure 3. C is CNF signature of gate AND3. $c_1 \sim c_4$ is characteristic clause of gate AND3. Clause c_1 that contains all the variables in gate AND3, is key clause. a is the output variable.

$$a = AND\ (b,c,d) \xrightarrow{\text{Clause Form}} C = \begin{Bmatrix} c_1 : (a \vee \neg b \vee \neg c \vee \neg d) \\ c_2 : (\neg a \vee b) \\ c_3 : (\neg a \vee c) \\ c_4 : (\neg a \vee d) \end{Bmatrix}$$

Fig. 3. CNF signature of gate AND3

Under encoding rules, gate with the same characteristics function will be encoded into the same set of clauses. Potential attackers can exploit structural knowledge from CNF formulas to restore the circuit structure. Some restoring circuit structure algorithms are based on concept of directed hyper-graph and bipartite graph.

Definition 2 (Hypergraph of CNF). *Let Σ be a CNF formula. A graph of clauses $G = (V, E)$ is associated to Σ s.t.*

1. *each vertex of V corresponds to a clause of Σ*
2. *each edge (c_1 , c_2) of E corresponds to a variable of Σ,while c_1 c_2 contain the same variable or complement.*
3. *each edge is labeled by the variable.*

Definition 3 (Directed Hypergraph of CNF). *A Directed Hypergraph is a Hypergraph with each endpoint of edge labeled by :*

1. *$\uparrow$ when clause contains variable*
2. *$\dagger$ when clause contains complement of variable*

Definition 4 (Bipartite graph of CNF). *Let Σ be a CNF formula. A graph of clauses $G = (V, E)$ is associated to Σ s.t.*

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

Fig.4a) gives the corresponding hypergraph of gate AND3 in Fig.3. Fig.4b) gives the corresponding Directed Hyper-Graph of gate AND3 in Fig.3. while $\uparrow$ represents positive, - represents negative variable. Fig.4c) gives the corresponding Bipartite Graph of gate AND3 in Fig.3.

Based on definitions list above, Roy, etc.[8] articulats that an arbitrary combinational circuit can be encoded as a CNF-SAT instance so that its circuit structure is preserved and can readily be extracted. they describe algorithms for restoring circuit structure from CNF formulas and empirically show its success and scalability on very large benchmarks.

Fu[9] presents a tool CNF2CKT to implement CNF-to-circuit decoding. The CNF2CKT algorithm effciently extracts a maximum circuit structure from any given CNF instance.

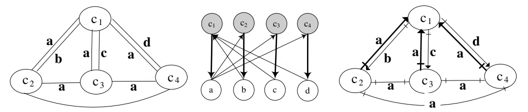


Fig. 4. Graph representation of gate AND3

These circuit structure extraction algorithms are based on subgraph isomorphism and pattern matching, exploiting the graph structure characteristics of CNF formula. In cloud computing paradigm threats can use these algorithms to obtain the circuit structure information carried by CNF formula. Therefore, It is essential to prevent information leakage when outsourcing CNF formula to cloud.

4 Cloud-SAT-solver with privacy-preserving

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

First, since CNF signature is key in circuit extraction, altering CNF signature of gate in CNF formula will make circuit extraction algortigm losing efficacy. Second, the current classical SAT solver [10] is efficient with integration of mechanisms such as conflict detection,Unit propergation. Distinct from obfuscating algorithm in literature[11] which develops a brand-new SAT-solving algorithm, we wish to take advantage of classical SAT solver. Third,we anticipate the solution of obfuscating CNF formula may be much easily mapped into original CNF formula.

The proposed algorithm is also dependent on the following definition.

Definition 5 (Husk formula). *Husk formula is a CNF formula with a unique solution,in which assignment of variables is non-specific(Not all 0 or all 1).*

The cloud SAT-solver based on obfuscating algorithm is shown intuitively in Figure 5, and formally in Algorithm 1.

With the original CNF formula S_1, Algorithm 1 compute its solution with the following four steps. Steps 1, 2 and 4 are done locally, while step 3 is on the cloud server.

In subsequent sections 4.1, 4.2 and 4.3,the GENERATOR, OBFUSCATOR and MAPPER will be described in detail. It need be emphasized that, S_1 and S_2 have disjoint set of variables.

4.1 Generate Husk formula

From the point of view of cryptology, Husk formula is a secret key used to encrypt the CNF formula. Our obfuscating algorithm requires Husk formula

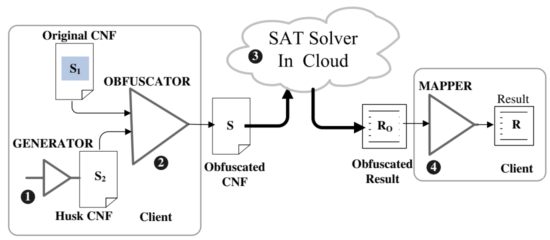


Fig. 5. The cloud SAT-solver based on obfuscating CNF

Data: The original CNF formula S_1
Result: The solution of S_1
begin
  GENERATOR generates a husk formula with unique solution r_h ;
  OBFUSCATOR obfuscates s_1 to obtain a new cnf formula s ;
  Solving S on cloud server ;
  MAPPER maps solution of S to that of S_1 ;
end

Algorithm 1: The general framework

has a unique solution. In this paper, Husk formula is constructed based on prime factorization method: given prime X with a binary vector representation $X =< x_1, x_2, \ldots, x_n >$, taking square of X as the of the output of multiplier M, while banning X equals 1. Then converting the multiplier M into CNF formula Φ. Through construction above, the two inputs of M must all be $X =< x_1, x_2, \ldots, x_n >$can CNF formula of M be satisfied. That means assignment of input variable is unique. Since assignment of all variables in CNF formula of M are decided by assignment of inputs variable.So,Φ has a unique solution. Assuming multiplier $M(I_1, I_2, O)$, in which there are two inputs I_1 and I_2, and one output O. GENERATOR algorithm to generate Husk formula is shown in Algorithm 2.

Data: NULL
Result: Husk CNF S_2 and Husk result R_H
begin
  generate a prime number p ;
  $sq = p \times p$;
  $\phi = M(I_1 \neq 1, I_2 \neq 1, O = sq)$;
  $S_2 = Tseitin(\phi)$;
  $R_H = p \mid p$;
end

Algorithm 2: GENERATOR

4.2 Construct obfuscating formula

The proposed obfuscating algorithm is to generates a new CNF formula S, by embedding clauses and variables(as literal) of Husk formula S_2 into Original formula S_1 with proper rules ,(there is no intersection between variables set of S_2 and S_1). Through adding new clauses and new literals, the algorithm alters the clause set and literal set in clauses of S_1,so as to obfuscating CNF signature in S_1. Proper rules guarantees solution space is invariant, that means, S and S_1 can be solved with the same SAT-solver. There is following relationship between solutions of S and that of S_1: S is unsatisfied iff S_1 is unsatisfied; S is satisfied iff S_1 is satisfied, and solutions of S_1 can be extracted from solutions of S by projection on variables set of S_1. Details of implementation is in OBUFSCATOR algoritms.

Data: The original CNF S_1Husk CNF S_2Husk result R_H
Result: The obfuscated CNF S, variable mapping M
1 **begin**
2     $markS_1$;
3   **for** $c \in S_1$ **do**
4   lit =get literal $\in R_H$;
5         $c = c \cup lit$;
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   **for** $c \in S_1$ **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   insert $\neg lit$ into c ;
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**

Algorithm 3: OBFUSCATOR

In order to keep solution space invariant, when insert variable of S_2 into clauses of S_1, rules must be followed (OBFUSCATOR algorithm,line 4,16): variables,which is assigned F in R_H (unique solution of S_2)are as positive literals;

variables,which is assigned T in R_H (unique solution of S_2)are as negtive literals. Rationality of the rules will be explained in section 4.1

In algorithm(lines 1 , line 5-8), there is two procedure defined, *mark* (line 1) and *generate_new_clause* (line 6) . Procedure *mark* marks key clauses and output variables of some kind of gate in CNF formula. Procedure *generate_new_clause* generates some new clauses matching the key clauses, so as to assemble new CNF signature(such as assemble CNF signature of AND3 based that of AND2, details presented in section 5.2), therefore improve the hardness of distinguish gates.

Gate of different type has different forms in key clauses and output variable, so that different mark algorithms are needed ,but complexity of these mark algorithms is of the same. Here take AND2 gate as example ,detailed implementation is presented in MARK algoritm.

```
Data: S, gateType AND2
Result: marked S
begin
    for ((C ∈ S) && (|C| == 3)) do
        for L ∈ C do
            for ((C1 ∈ S) && (−L ∈ C1) && (|C1| == 2)) do
                for L1 ∈ C1 do
                    if ((−L1 ∈ C) && (L1! = L)) then
                        match + + ;
                    end
                end
            end
        end
    end
    if match ≡ 2 then
        mark L as output literal ;
        mark C as key clause ;
    end
end
```

Algorithm 4: MARK

Same as MARK presented in Algorithm 4, we take AND2 gate as example here, more details are presented in GENAND2CLAUSE algoritm.

Algorithm *mark* need to analyze structure in CNF formula, times to run is up to number of clauses and variables in CNF formula.Since,Obfuscator runs in client, such as panel or PC.Whether to use mark algorithm or not is depended on computation capabity of client.

Inserting positive/negative literal into clause or appending new clause into formula will result in the transformation of key clause and CNF signature in formula, therefore bringing corresponding changes in hypergraph and bipartite-

Data: key clause C in AND2, Husk Literal lit
Result: new clause C_1
begin
  $olit$=get output literal from C ;
  $C_1 = lit \cup \neg olit$;
end

Algorithm 5: GENAND2CLAUSE

graph of CNF formula. These transformations reflect the effectiveness of obfuscation.

4.3 Recover the original solution

The variables in S is superset of that in S_1 .In obfuscating algorithm, Mapping table M is used to map variable name from S_1 to S. Therefore, to get solution of S_1 ,we need 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,given by SAT-solver located in the cloud. Procedure to get solution of S_1 is implemented by MAPPER algorithm.

Data: obfuscated result R_O, variable maping table M, Husk result R_H
Result: result R
begin
  for $lit \in R_H$ **do**
    $var = lit > 0?lit : -lit$;
    $rvar = M[var].var$;
    if $M[var].S == S_1$ **then**
      $R[rvar] = lit > 0?rvar : -rvar$;
    else
      $Hlit = lit > 0?rvar : -rvar$;
      if $Hr[rvar]! = Hlit$ **then**
        printf(something wrong with CloudSAT solver);
      end
    end
  end
end

Algorithm 6: MAPPER

5 Correctness, effectiveness and performance analysis

The obfuscating, algorithm blends the original formula seamless with Husk formula, correctness and effectiveness are basic requirements of the algorithm.

The Correctness means that the algorithm should keep the solution space of the original CNF formula, that is, for CNF formulas S_1 and its according obfuscated formula S, the following facts are hold: If S_1 is unsatisfied, S is unsatisfied either, vice versa, and the unsatisfied core of S_1 can be obtained by deleting literal in S_2 from clauses in unsatisfied core of S; If S_1 is satisfied, S is satisfied too, vice versa, and the solution of S_1 can be obtained by projecting solution of S into variables of S_1 .

The effectiveness of the algorithm refers to that changes brought to the CNF formula by obfuscating, make the circuit structure extraction work more difficult, or circuit structures can not be extracted anymore.

Obfuscating algorithm and solution recovery algorithms are required to run on the client with weak computing power, algorithmic complexity should not be too high.

5.1 Correctness proof

For illustration, OBFUSCATOR algorithm is simplified as follows. Since step 4 only remaps variable name, which will not affect assignment of variable, it will be ignored when discussed correctness.

1. Step1 : For each clause in S_1 , take one or more variables in S_2 , insert them into clauses in S_1, abided by $mathbfrule1$: if assignment of variable is T in R_H, insert negative literal of variable into clause; if assignment of variable is F in R_H, insert positive literal of variable into clause. New clauses generated by Step 1 constitutes clause set S_3.
2. Step2: generate new clause with literals in R_H and output variables in S_1 , abided by $mathbfrule2$: if assignment of variable is T in R_H, insert positive literal of variable into clause; if assignment of variable is F in R_H, insert negative literal of variable into clause. Literal forms of output variable is up to the key clause to which the output variable belongs. New clauses generated by Step2 constitutes clause set S_4.
3. Step3: combine clauses in S_2,S_3,S_4, disorder clauses and produce new formula S.
4. Step4: remap variables name, log mapping information into table M

For illustration, the following definitions and settings are given.

Definition 6 (Original variable). *Variable in original CNF formula is called* **original variable***; Clause in original CNF formula is called* **original clause***. Variable in Husk formula is called* **Husk variable***. Clause in Husk formula is called* **Husk clause***. Literal constitutes result of Husk formula is called* **Husk literal***.*

According to Step 3 in algorithm listed above, there are two type of variables in obfuscated formula: original variable and Husk variable. While there are three type of clauses in obfuscated formula: Husk clause(clauses in S_2), original clause blended with Husk variable(clauses in S_3), clauses consist of Husk literal and output variable (clauses in S_4) .

As we know, Husk formula is a unique solution, when GENERATOR generate Husk formula S_2, it also give the result $R_H = \{b_1, b_2, \ldots, b_m\}$. When assign R_H to variable, each clause in S_2 will be T.

According to Step 1 in algorithm, each clause in S_3 can be expressed in form $C = A \vee B$, while A is clause from S_1 , $B = B_i$, $B_i = z_i \vee B_{i-1}$, $B_1 = z_1$. Under constrains of Husk clause, Husk variable must be assigned $H = b_1, b_2, \ldots, b_m$, then z_i must be F. So $B = B_i = z_i \vee z_{i-1} \vee \ldots \vee z_1$ must be F. That means $C = A \vee B$Value of C is up to A. So solution space of S_3 is same as that of S_1.

Clauses in S_4 are consisted of Husk literal and output variables from S_1. These clauses can be represented as $C = z_i \vee A$. while if $b_i == F$,then $z_i = \neg y_i$,if $b_i == T$,then $z_i = y_i$. because under constraint from Husk clauses, Husk variables must be assigned as following: $R_H = \{b_1, b_2, \ldots, b_m\}$Thus $z_i = TC = z_i \vee A = T$So clause C will not constrain the assignment of variable in A. Take following example to illustrate.

Theorem 1. *With the following hypothesis:*

1. *Clause $A = a \vee X$ and clause $B = b$, while b is not variable in X.*
2. *let $S_1 = A$, $S_2 = B$,then $R_H = \{(T) \times (b)\}$*

Obfuscating procedure:

1. *Step1. according to R_H and rule 1, get clause $C = A \vee \neg b$,let $S_3 = C$*
2. *Step2. according to R_H and rule 2, take b as literal,take literal a in clause A,get clause $D = a \vee b$,let $S_4 = D$*
3. *Step3. let $S = S_2 \wedge S_3 \wedge S_4$, $S = B \wedge C \wedge D$*
4. *Step4. omit*

Proof. By substituting the following equations

1. $B = b$
2. $C = A \vee \neg b$
3. $D = a \vee b$
4. $S_1 = A$
5. $S_2 = B$

into $S = B \wedge C \wedge D$, we have

$$\begin{aligned} S &= b \wedge (A \vee \neg b) \wedge D \\ &= b \wedge A \wedge D \\ &= b \wedge A \wedge (a \vee b) \\ &= b \wedge A \\ &= B \wedge A \\ &= S_1 \wedge S_2 \end{aligned} \tag{2}$$

In conclusion, obfuscated formula S is equivalent to $S_1 \wedge S_2$.

Assuming O, H and Z are solutions of formula S_1 , S_2 and S respectively, With $O = \{(a_1, a_2, \dots, a_m) \times (x_1, x_2, \dots, x_m)\}$ and $H = \{(b_1, b_2, \dots, b_n) \times (y_1, y_2, \dots, y_n)\}$. We have :

$$\begin{aligned} Z &= O|H \\ &= \{(a_1, a_2, \dots, a_m, b_1, b_2, \dots, b_n) \times (x_1, x_2, \dots, x_m, y_1, y_2, \dots, y_n)\} \end{aligned} \quad (3)$$

Under the obfuscating algorithm in this paper, the following theorem holds.

Theorem 2. *S_1 , S_2 ,S is CNF formula, S_2 has only a unique solution,$S=$ OBFUSCATOR (S_1 , S_2); Assume Z is solution of S, and Y is the solution of S_2 , then $X = MAPPER(Z, S_1)$ is solution of S_1 .*

Theorem 3. *S_1 , S_2 ,S is CNF formula, S_2 has only a unique solution,$S=$ OBFUSCATOR (S_1 , S_2), S_1 is unsatisfied, if and only if S is unsatisfied.*

These theorems ensure the correctness of the obfuscating algorithm, solution space before and after obfuscating did not change. Appendix 1 gives a formal proof of these theorems.

5.2 Effectiveness analysis

As discussed in chapter 2, in the threat model, analysis of hyper-graph and bipartite graph is mean to extract circuit structure. In order to verify the effectiveness of the algorithm, we give the qualitative and quantitative analysis of changes to hyper-graph and bipartite graph brought by obfuscating.

Qualitative analysis Assume there are instances a and e of gate AND2. Fig 6 give CNF signature and hyper-graph of a and e. Assume Husk result

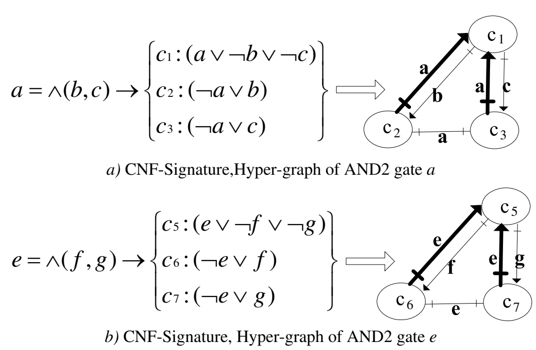
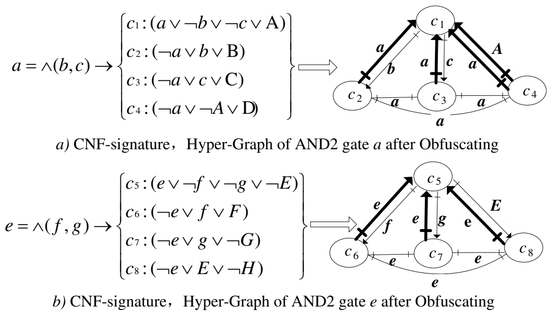


Fig. 6. CNF signature and Hyper-Graph of AND2 gate a and e

is $\{FFFFTFTT \times ABCDEFGH\}$, after obfuscating, CNF signature and hypergraph of a and e are shown in Fig 7. Both a and e are instances of gate



a) CNF-signature，Hyper-Graph of AND2 gate *a* after Obfuscating

b) CNF-signature，Hyper-Graph of AND2 gate *e* after Obfuscating

Fig. 7. CNF signature and Hyper-Graph of AND2 gate a and e after obfuscating

AND2. Changes to CNF signatures of a and e by obfuscating result in four effects. First, the length of c_1 and c_5, key clauses of a and e, is changed from 3 to 4, this change disables structure detection techniques of key clause based pattern matching presented in literature [9]. Second, after obfuscating, CNF signatures of a and e are different, thus Hypergraph of them are not isomorphic anymore, this change disables structure detection techniques based on sub-graph isomorphic presented in literature [8]. Third, there are new clauses added in formula, such as c_4 in a and c_8 in e, this change also disables structure detection techniques based on sub-graph matching [8].

Furthermore, by inserting proper literal in key clauses and generating new clause, CNF signature of instance e is changed from AND2 to AND3, shown in Fig 7b), 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 change makes it impossible to distinguish structure of AND2 and AND3.

The changes to bipartite graphs of a and e, before and after obfuscating,is same as Hypergraph.

5.3 Complexity of the algorithm analysis

In OBFUSCATOR algorithmmain procedure consists only one layer of loops,Although procedure of marking key clause and output variable consists 4 layer of loops,but times of 3 inner loops is limited by length of clauses so complexity is still $O(n)$, other part of algorithm consists only 1 layer of loop at least, so the complexity of the OBFUSCATOR algorithm is $O(n)$. The complexity of the MAPPER algorithm is O(n).

6 Related works

6.1 Secure Computation Outsourcing

With the popularity of cloud computing, secure outsourcing scientific computing become a hot research topic. According to the methods used, it can be classified into encryption based method, disguising based method. The related research will introduce in the following section.

Encryption based techniques R. Gennaro [14]presented the concept of verifiable computation scheme, which is based on Yaos Garbled-Circuit[25] and fully homomorphic encryption(FHE)[13] , this scheme shows the secure computation outsourcing is viable in theory. Due to the extremely high complexity of FHE operation and the pessimistic circuit sizes that can hardly be handled in practice, attempts to apply the scheme into real applications is still in progress. Zvika [10]etc constructed an obfuscated program for d-CNFs that preserves the input-output functionality of the function, but reveals nothing 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, such as computation cost by multi-linear map.

Disguising based techniques Instead of outsourcing general functions, in the security community, Atallah et al. explore a list of customized solutions [16][17][18][19] for securely outsourcing specific computations. Orienting linear algebra algorithms in scientific computing, M. J. Atallah[16] multiply data with random diagonal matrix before outsourcing. The results can be obtained by reversible matrix operations. The paper also discussed the extension problem domain and reduction, in order to further disguise computation. One problem with the approach is not discussed how to verify the correctness of the results returned.In [17]., they give the first investigation of secure outsourcing of numerical and scientific computation, including LE. Though a set of problem dependent disguising techniques are proposed, they explicitly allow private information leakage. C.Wang[5] discussed the problem of securely outsourcing LP computations in cloud computing, By explicitly decomposing LP computation outsourcing into public LP solvers and private data, our and provide such a practical mechanism design which fulfills input/output privacy, cheating resilience, and efficiency.

6.2 Verifiable computation delegation

Verifiable computation delegation, where a computationally weak customer can verify the correctness of the delegated computation results from a powerful but untrusted server without investing too much resources, has found great interests in theoretical computer science community.

In[15], Du. et al. propose a method of protecting high value rare events for general computation outsourcing in grid computing. To prevent participants from keeping the rare events, they injects a number of chaff items into the workloads so as to confuse dishonest participants.

In distributed computing and targeting the specific computation delegation of one-way function inversion, Golle et al. [21] propose to insert some precomputed results (images of ringers) along with the computation workload to defeat untrusted (or lazy) workers.

Szada et al. [22] extend the ringer scheme and propose methods to deal with cheating detection of other classes of computation outsourcing, including optimization tasks and Monte Carlo simulations.

Although these work mainly focus on results verification, the idea of addition of ringer, garbled circuit or chaff instances into computation workload without investing too much resources illuminates us.

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, GB RAM. We unloop circuits in iscas89-benchmark into 30 times and transform them into CNF formulas to simulate real verification. Generating Husk formula with variables numerber vn=675 and clauses number cn=2309and obfuscating the CNF formula. Fig 8 presents size of CNF formula after obfuscatingrepresented with vn-variables number and cn-clauses number).SAT Solver time(SAT Time) before and after obfuscating(CloudSAT Time), obfuscating time(Obfuscating Time),and result recovery time(Mapping time).

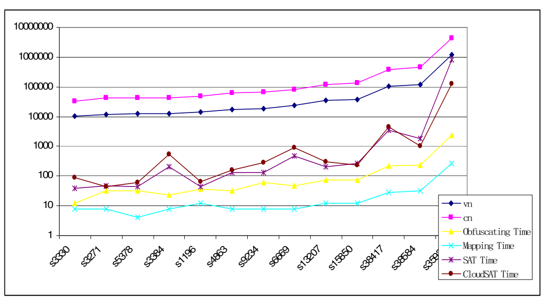


Fig. 8. Relationship between Runtime and Size of CNF

Experiments show that, although the size of CNF formula(cn/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 size of the added Husk formula, SAT Solver time is likely to be equal before and after obfuscating. SAT solving time depends on the internal structure of CNF formula. shown in Figure 8. On the other hand, obfuscating time and result recovery time increase linearly with the size of circuit (variables number and clauses number) shown in Figure 8.

8 Conclusion

This paper is intended to meet the privacy-preserving needs of SAT solving in cloud computing with disguising based techniques. A obfuscating algorithm changes the clauses set of CNF formula and literals set of clause so as to transform CNF signature in formula. In the obfuscating process, with proper embedding rulesthe algorithm remain the solution space of CNF formula unchanged, namely: if the formula S_1 is unsatisfied, the formula S, which is gernerated by obfuscating S_1 , is unsatisfied either; If the formula S_1 is satisfied, the formula$S$can also be satisfied, and furthermore, the solution of formula S_1 can be obtained by projecting solution of$S$ on set of variables of S_1 .

Due to obfuscating has transformed the characteristics clauses set of gate in formula, characteristics clauses based pattern matching can not restore the gate structure from CNF formula anymore. Since the obfuscating algorithm herein can not guaranteed all instances with the same type of CNF signature being transformed into different form, the use of subgraph isomorphism methods may find a part of gate instances with same new signature, but it is still impossible to recover the full circuit structure. Thorough security analysis and experiments demonstrate the effectiveness and practicality of the proposed mechanism.

9 Acknowledge

This work was supported in part by the National Science Foundation under grant.

10 REFERENCES

References

1. Davis, M.; Putnam, H. (1960). "A Computing Procedure for Quantification Theory". Journal of the ACM 7 (3): 201.
2. Gary D. Hachtel, Fabio Somenzi: Logic synthesis and verification algorithms. Springer 2006: I-XXIII, 1-564
3. Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, Helmut Veith: Counterexample-Guided Abstraction Refinement. CAV 2000: 154-169
4. G. Tseitin. On the complexity of derivation in propositional calculus. Studies in Constr. Math. and Math. Logic, 1968.
5. Cong Wang, Kui Ren, Jia Wang: Secure and practical outsourcing of linear programming in cloud computing. INFOCOM 2011: 820-828
6. C. M. Li, Integrating equivalency reasoning into Davis-Putnam procedure in Proceedings of the 7th National Conference on Artificial Intelligence (AAAI'00), 2000, pp. 291-296.
7. R. Ostrowski etc. Recovering and exploiting structural knowledge from cnf formulas, in Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP'02), 2002, pp. 185-199

8. Jarrod A.Roy Igor L.Markov. Valeria Bertacco. Restoring Circuit Structure from SAT Instances IWLS, Temecula Creek CA, June 2004 , pp. 361-368.
9. Zhaohui Fu. Sharad Malik Extracting Logic Circuit Structure from Conjunctive Normal Form Descriptions. 20th International Conference on VLSI Design (VLSI Design 2007), Sixth International Conference on Embedded Systems (ICES 2007), 6-10 January 2007, Bangalore, India. IEEE Computer Society 2007 Pages 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. Hey, You, Get Off of My Cloud:Exploring Information Leakage in Third-Party Compute Clouds, CCS 2009
12. Zvika Brakerski and Guy N.Rothblum Black-Box Obfuscation for d-CNFs Proceeding ITCS '14 Proceedings of the 5th conference on Innovations in theoretical computer science Pages 235-250
13. C. Gentry, Fully homomorphic encryption using ideal lattices, in ACM STOC, Bethesda, MD, USA, 2009, pp. 169-178.
14. Rosario Gennaro, Craig Gentry, Bryan Parno: Non-interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers. CRYPTO 2010: 465-482
15. Wenliang Du and Michael T. Goodrich. Searching for High-Value Rare Events with Uncheatable Grid Computing 2004
16. Mikhail J. Atallah, K. N. Pantazopoulos, John R. Rice, Eugene H. Spafford: Secure outsourcing of scientific computations. Advances in Computers 54: 215-272 (2001)
17. Mikhail J. Atallah, Jiangtao Li: Secure outsourcing of sequence comparisons . Int. J. Inf. Sec. 4(4): 277-287 (2005)
18. David Benjamin, Mikhail J. Atallah: Private and Cheating-Free Outsourcing of Algebraic Computations. PST 2008: 240-245
19. Mikhail J. Atallah, Keith B. Frikken: Securely outsourcing linear algebra computations. ASIACCS 2010: 48-59
20. Tuomo. Factoring benchmark for SAT-solover
21. Pascal Paillier: Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. EUROCRYPT 1999: 223-238
22. D. Szajda, B. G. Lawson, and J. Owen, Hardening functions for large scale distributed computations, in Proc. of IEEE Symposium on Security and Privacy, 2003, pp. 216-224
23. Paralleling OpenSMT Towards Cloud Computing http://www.inf.usi.ch/urop-Tsitovich-2-127208.pdf
24. Formal in the Cloud OneSpins New Spin on Cloud Computing.http://www.eejournal.com/archives/articles/20130627-onespin/?printView=true
25. A. C.-C. Yao, Protocols for secure computations (extended abstract), in Proc. of FOCS, 1982, pp. 160-164.