Structure-Aware CNF Obfuscation for Privacy-Preserving SAT Solving

Ying Qin*, ShengYu Shen* and Yan Jia*
*College of computer
National University of Defense Technology, ChangSha, China 410073
Email: {yingqin,syshen,yjia}@nudt.edu.cn

***Abstract*—In this paper, we study the problem of solving hard propositional satisfiability problem in Cloud or computing grid, which is honest-but-curious. We propose an approach to preserve input and output privacy based on CNF obfuscation, and present obfuscation algorithm and its corresponding solution recovery algorithm. By obfuscation, the CNF formula is transformed into another formula, with different circuit structure and over-approximated solution space. Solution of the original CNF can be extracted from the solution of obfuscated CNF by solution recovery. Theoretical analysis demonstrates that, obfuscation algorithm can significantly change the structure of CNF formula with polynomial time complexity, while solution recovery algorithm can filter out solution with linear time complexity. Experimental results on ISCAS89 benchmark show that obfuscation incurs acceptable negative impact on the performance of SAT solving.**

***Keywords*-component; formatting; style; styling;**

I. INTRODUCTION

Propositional satisfiability [1] has been widely used in hardware and software verification [2], [3], cryptography [4] etc. 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 and grid can provide elastic computing resource, which make outsourcing hard SAT problem to public Cloud or computation grid [5], [24], [25] very attractive.

However, security issues make many customers reluctant to move their critical computation tasks to grid or Cloud. Since the grid is constructed by loosely connecting a large number of readily available, high-end, heterogeneous computing facilities [5], the risks posed by hoarding participants in grid computing environments are real and immediate [19]. While the Cloud vendors can be trusted and the Cloud infrastructure (i.e., the virtualization layer) can be assumed to be secure, the virtual machines cannot be trusted to be always honest and loyal. Literature [13] points out security vulnerability that Amazon EC2 suffered from: Since Amazon Machine Image (AMI) are widely shared among the EC2 community, a malicious AMI could flood the community with hundreds of infected virtual instances. Literature [14] also points out the possibility of attacking virtual machine through another virtual machine in the same physical machine.

These facts show that input and output data of SAT problem may be exposed to untrusted third party, who may inspect valuable information from these data. For example, SAT program originated from verification may suffer from leakage of privacy, such as circuit structure information. Works carried out by Roy [10] and Fu [11] suggest the possibility of extracting circuit information from CNF formula. Furthermore, Du [19] called solution of hard SAT problem generating from cryptograph etc as high-value rare events, and point out that the solution of SAT problem should also be treated as privacy, because it may be leaked to third party by hoarding participants. Moreover, the SAT solver deployed in grid or Cloud may also be compromised by adversary, who may compel SAT solver to return incorrect result to mislead verification.

These threats put customers who plan to outsource SAT solving in a dilemma: using public Cloud or grid in open environment is cost-efficiently, but may suffer from leakage of privacy or incorrect result. In order to meet this challenge, we develop novel techniques for outsourcing SAT solving in Cloud or in computing grid securely, which can preserve privacy of input and output, without changing the solution.

The major contribution of this paper is: For the first time, we give a structure aware obfuscation algorithm, which can preserve input and output privacy during outsourcing SAT solving. This algorithm has the following four advantages: First, by obfuscation, 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, solution of the original CNF can be recoverd from over-approximated solution of the obfuscated CNF, which keep the solution a privacy even for the SAT solver; Finally, obfuscation algorithm is polynomial complexity, and solution recovery algorithm is linear complexity, which reduce the impact on the overall performance of SAT solving.

The remainder of this paper is organized as follows. Preliminaries are presented in Section 2. Section 3 describes of the threat model, while the implementation of privacy persevering SAT solver based on obfuscation is presented in Section 4. Section 5 analyzes correctness, effectiveness and algorithms complexity of obfuscation; Section 6 describes the related work; Section 7 gives the experimental results, while Section 8 concludes this paper.

II. Preliminaries

A. SAT solving

The Boolean value set is denoted as $B = \{T, F\}$. For a Boolean formula F_C over a variable set V, the propositional satisfiability problem (abbreviated as SAT) is to find a satisfying assignment $A : V \rightarrow B$, so that F_C evaluates to T. If such a satisfying assignment exists, then F_C is satisfiable, and the saitifying assignment is called solution of F_C; otherwise, it is unsatisfiable. An unsatisfiable subset of formula is an unsatisfiable core. A computer program that decides the existence of a satisfying assignment is SAT solver [12].

Normally, a SAT solver requires the formula to be in 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 three clauses $x_1 \vee \neg x_2$, $x_2 \vee x_3$, $x_2 \vee \neg x_4$. 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.

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

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 $|\Phi| \equiv 3$.

Variable set of CNF formula F_C is denoted as V_{F_C}.

When variables in CNF formula F_C are assigned with solution S_C, we denote it as $F_C(S_C/V_{F_C})$.

B. Tseitin encoding

In hardware verification, circuits and properties are converted into CNF formula by Tseitin encoding [6], 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 list 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).

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

III. Threat model

A. System assumption

We will take Cloud computing as example to illustrate possible threats when outsourcing SAT solving, since grid is same as Cloud as for threat model.

In our research, there are two types of Cloud, private Cloud and public Cloud. Private Cloud is trusted but has only limited computation power and memory to handle simple computation. While public Cloud can provide elastic computation and memory resource to deal with complex computation. CNF formula will be generated from netlist or program in private Cloud, while SAT solover is deployed in public Cloud to solve the CNF formula and return the solution to the private Cloud.

B. Attack model

Algorithms [8]–[11] have been proposed to extract and utilize circuit structure in CNF formula. Circuit structure extraction algorithms, presented by Roy et al. [10] and Fu et al. [11], are based on subgraph isomorphism and pattern matching technique, and can recover lots of circuits from CNF formula. These pattern matching or subgraph isomorphism techniques are available freely. In public Cloud computing environment, adversary, who has controled VM [13], [14], may use these algorithms to recover the circuit structure from the CNF formula.

As pointed out by literature [19], since solutions to difficult instances of NP-complete problems are rare events, with the practical importance of many of these problems, hoarding participants may keep these solutions for economic value.

Therefore both CNF formula and its solutions should be treated as privacy that should be preserved.

As a result, in our research, we assume there are curious and hoarding participants [19] in public Cloud. That means, the participants conduct all the required computations to get solutions of SAT problem; But they may try to get information from CNF formula as much as possible, such as circuit structure information; And if the solution are valuable, they may keep the computation results and leak it to third party.

IV. System design

A. Privacy-preserving SAT solving framework based on obfuscation

When we design the Cloud or grid oriented SAT solving framework, the following four goals are taken into consideration:

1) As for the portability, current SAT solvers with conflict analysis [12] are very efficient. So we would like to use them directly instead of developing new algorithms like [17].
2) As for the stealth [26], the framework should be able to prevent circuit structure from being recovered from CNF formula.
3) As for the resilence [26], the framework should prevent accurate solution from being known even by the SAT solver, which is deployed in public Cloud.

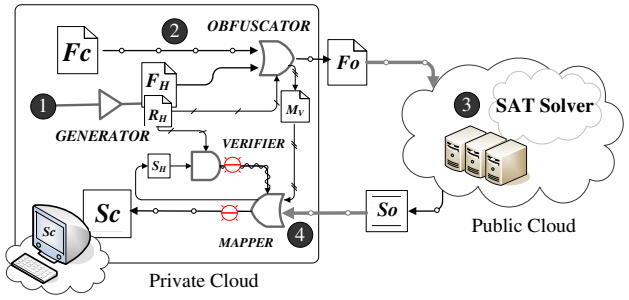

Figure 1. Privacy-preserving SAT solving framework based on CNF formula obfuscation.

4) As for the cost, the framework should not incur too much overhead.

According to these goals, we present a privacy-preserving SAT solving framework based on CNF formula obfuscation. To obfuscate the CNF formula, according to SSH rules and CSA strategies described below, we embed extra literals into CNF clauses and insert extra clauses into CNF formula. The extra literals and part of new clauses are from another CNF formula, which we called Husks formula. SSH rules and CSA strategies ensure the original CNF formula to be blended with Husks formula seamless, to attain goals 2) and 3). SSH rules and CSA strategies are described in *IV-C3)* and *IV-C4)*.

***Definition** 1 (Singular Husk formula):* Singular Husk formula is a CNF formula with only one solution, and assignments of variables in this solution is non-uniform, that is, not all 0 or all 1.

***Definition** 2 (Husks formula):* Husks formula is a satisfiable CNF formula with more than one solution, and assignments of variables in each of its solutions are non-uniform, that is, not all 0 or all 1.

Detailed implementation of this framework is shown in Figure 1. The SAT problem is solved in 4 steps.

Step 1, GENERATOR algorithm generates a Husks formula F_H and one of its solution R_H.

Step 2, OBFUSCATOR algorithm obfuscates the Original CNF F_C to obtain a new CNF formula F_O.

Step 3, F_O is solved by SAT Solver deployed in public Cloud, which returns solution S_O.

Step 4, MAPPER and VERIFIER algorithm maps S_O to S_C, and check if F_C is satisfied under S_C.

The **Step 3** run in public Cloud, while other steps run in trusted private Cloud.

The GENERATOR, OBFUSCATOR, MAPPER and VERIFIER algorithms will be described in Subsection IV-B, IV-C and IV-D respectively.

B. Husks formula Generation

There are several methods to generate satisfiable CNF formula [15], [16]. In this paper, Husks formula (in Definition 2) is constructed based on prime factorization method, as described in literature [16].

GENERATOR algorithm to generate Husks formula is shown in Algorithm 1.

First, given two primes $p_A \neq p_B$ (at line 2), we assign $p_A \cdot p_B$ to the output of a multiplier M with constraint $I_1 \neq 1$ and $I_2 \neq 1$ (at line 3). I_1 and I_2 are inputs of M.

Second, we convert the multiplier M into CNF formula $Tseitin(M)$ (at line 4).

To satisfy $Tseitin(M)$, the two inputs of M must be $\{I_1 = p_A, I_2 = p_B\}$ or $\{I_1 = p_B, I_2 = p_A\}$, which makes the $p_A|p_B$ or $p_B|p_A$ the two solutions of $Tseitin(M)$. We take one of them as R_H.

Singular Husk formula with unique solution (in Definition 1) can also be constructed by similar method described in Algorithm 1, by constraining $p_A \equiv p_B$.

C. Circuit Structure aware obfuscation

1) Circuit structure in CNF formula: Since we want to protect circuit structure in CNF formula, let's first study how the circuit can be recovered from CNF formula. Literatures [10], [11] have proposed algorithms to recover circuit structure from CNF formula in details. Before discussing them, some concepts should be introduced first.

***Definition** 3 (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**. Variable corresponding 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.

As metioned in [10], gates with the same characteristic functions will be encoded into the same CNF signature.

With these definitions, Roy et al. [10] 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 instance to represent the recovered circuit.

Fu et al. [11] presents another algorithm that first detects all possible gates with key clause and CNF signature based pattern matching, and then constructs a maximum acyclic

Algorithm 1: GENERATOR

Data: NULL
Result: Husks CNF F_H and Husks result R_H
1 **begin**
2 Generating prime numbers p_A and p_B ;
3 $\Phi = M(I_1 \neq 1, I_2 \neq 1, O = p_A * p_B)$;
4 $F_H = Tseitin(\Phi)$;
5 $R_H = p_A \mid p_B$;

combinational circuit by selecting a maximum subset of matched gate.

Potential attackers can exploit these knowledge to recover the circuit structure. Thus, CNF signature and key clause are important information that should be protected.

2) Input and output privacy-preserving scheme: To prevent information carried by CNF formula and its solution from leakage, a privacy-preserving scheme is proposed, the scheme is based on the following facts and anticipations:

Fact 1: Changing CNF signature and key clause in CNF formula will make circuit recovering based on pattern matching or subgraph isomorphism impossible.

Fact 2: Solution space should not be under-approximated after obfuscation, otherwise the result will be misleading even for real user.

Anticipation 1: According to Fact 2, solution space have to be over-approximated after obfuscation, so as to mislead hoarding participants in public Cloud.

Anticipation 2: The solution of obfuscated CNF formula should be easily mapped back to the original formula.

The proposed scheme, denoted as OBFUSCATOR, generates a new CNF formula F_O, by embedding Husks formula F_H into the original formula F_C, with Circuit Structure Aware(CSA) strategy and Solution Space Hold(SSH) rules . By CSA strategy, the scheme changes the clause set and literal set in clauses of F_C, to prevent its structure from being recovered. By SSH rules, the solution space is over-approximated after obfuscation, so as to prevent its accurate solutions from being known even by SAT solver in public Cloud. We will describe them below in Subsection *IV-C3*) and *IV-C4*) respectively.

3) Solution Space Hold rules: Let's consider an interesting problem: We outsource CNF formula generated from SAT problem, and wish SAT solver deployed in Cloud to give solutions to the SAT problem, without knowing exactly what the SAT problem is and what the exactly solution is.

A simple approach is to blend CNF formula of real SAT problem with that of another satifiable SAT problem, and outsource the blended CNF formula. Unfortunately, partition based technique [27] can easily separate the two undependent formulas.

Let's consider an incremental approach: An arbitrary formula F_C, and a satisfiable formula F_H with $\boldsymbol{R_H}$ as one of its solutions, and there is no common variable between F_C and F_H, viz.$V_{F_C} \cap V_{F_H} = \phi$. We blend F_C with F_H seamlessly, so as to hide F_C. At the same time, we keep all solutions of F_C still in the new formula.

To blend F_C and F_H seamlessly, an intuitive approach is to insert variables of F_H into clauses of F_C, and generate new clauses with variables in F_C and F_H. According to property of CNF, for any CNF formula F_C, inserting new variables into its clauses may expand its solution space; On the contrary, adding new clauses which consist variables in F_C, may narrow down its solution space. How can we ensure all the solutions of F_C still in new formula? Before giving answer to this problem, following concepts should be clarified.

***Definition** 4 (Solution $S_C \subseteq$ Solution S_O):* CNF formula F_C and F_O have n_{F_C} common variables $x_1, ..., x_{n_{F_C}}$ and $|V_{F_C}| \equiv n_{F_C}$, $|V_{F_O}| \equiv n_{F_O}$, $n_{F_O} \geqslant n_{F_C} > 0$. S_C and S_O are solutions of F_C and F_O respectively, and assignments to n_{F_C} common variables are same in S_C and S_O, viz. $S_C = \{x_1 = B_1, ..., x_{n_{F_C}} = B_{n_{F_C}} | B_i \in \{T, F\},\ 1 \leqslant i \leqslant n_{F_C}\}$,
$S_O = \{x_1 = B_1, ..., x_{n_{F_C}} = B_{n_{F_C}}, ..., x_{n_{F_O}} = B_{n_{F_O}} | B_i \in \{T, F\},\ 1 \leqslant i \leqslant n_{F_O}\}$. Then Solution S_C is subset of Solution S_O, denoted as $S_C \subseteq S_O$.

***Definition** 5 (Solution Space Equation(SSE)):* CNF formula F_C has n solutions $\{S_{C_1}, ..., S_{C_n}\}$; By obfuscation, F_C has been transformed into F_O, which also has n solutions $\{S_{O_1}, ..., S_{O_n}\}$, and for $i \in [1, n]$, $S_{C_i} \subseteq S_{O_i}$. Then, we say F_O is solution space equated with F_C, denoted as $F_C \equiv_{SSE} F_O$.

***Definition** 6 (Solution Space Over-approximation(SSO)):* CNF formula F_C has n solutions $\{S_{C_1}, ..., S_{C_n}\}$; By obfuscation, F_C has been transformed into F_O, which has m solution, $\{S_{O_1}, ..., S_{O_n}, ..., S_{O_m}\}$, $m \geqslant n$, while for $i \in [1, n]$, $S_{C_i} \subseteq S_{O_i}$. Then, we say F_O is solution space over-approximated as F_C, denoted as $F_C \vdash_{SSO} F_O$.

In order to keep all solutions of F_C in new formula, we proposed Solution Space Hold(SSH) rules to obfuscate F_C with F_H and one of its solutions R_H, so as to make the solution space over-approximated after obfuscation.

Solution Space Hold Rules (SSH Rules):

1) **Rule 1**: For any clause $c \in F_C$, take one variable from R_H, and insert it into c according to the following rule: If assignment of variable is T in R_H, insert its negative literal; If assignment of variable is F in R_H, insert its positive literal. Then clause c is replaced with the resulted clause.
2) **Rule 2**: Generating new clauses with literals from R_H and variables in F_C according to the following rule: If assignment of variable is T in R_H, insert positive literal into clause; If assignment of variable is F in R_H, insert negative literal into clause.

***Definition** 7 (**Obf**(F_C, F_H, R_H)):* For arbitrary formula F_C, and satisfiable formula F_H with R_H as one of its assignments, $Obf(F_C, F_H, R_H)$ is the result of applying SSH Rules when blending F_C with F_H. If F_H is a Singular Husk formula and R_H is its unique solution, $Obf(F_C, F_H, R_H)$ is called **SSE obfuscation**. If F_H is Husks formula and R_H is one of its solutions, $Obf(F_C, F_H, R_H)$ is called **SSO obfuscation**.

For SSH based obfuscation, we have following theorems.

***Theorem** 1 (SSE Obfuscation):* For arbitrary CNF formula F_C, and Singular Husk formula F_{sH}, if $V_{F_C} \cap V_{F_{sH}} \equiv \phi$, and R_{sH} is the unique solution of F_{sH}, **then** $Obf(F_C, F_{sH}, R_{sH}) \equiv F_C \wedge F_{sH}$.

***Theorem** 2 (SSO Obfuscation):* For arbitrary CNF formula F_C, Husks formula F_H, if $V_{F_C} \cap V_{F_H} \equiv \phi$, and R_H is one of solutions of F_H. **then** $F_C \vdash_{SSO} Obf(F_C, F_H, R_H)$.

According to Theorem 1 and Definition 5, we have:

***Inference** 1:* For arbitrary CNF formula F_C, Singular Husk formula F_{SH}, if $V_{F_C} \cap V_{F_{SH}} \equiv \phi$, and R_{SH} is the unique solution of F_{SH}, **then** $Obf(F_C, F_{SH}, R_{SH}) \equiv_{SSE} F_C$.

Theorem 1 and 2 will be proved in Subsection V-A.

An obfuscated CNF formula $F_O = Obf(F_C, F_H, R_H)$ generated by SSO obfuscation consists of all the variables of F_C and F_H.

If variables from F_H are assigned with R_H, then we have: $F_O(R_H/V_{F_O}) = Obf(F_C, F_H(R_H/V_{F_H}), R_H)$

Accdording to Lemma 1 in Subsection V-A, $F_H(R_H/V_{F_H})$ can be expressed as a Singular Husk formula with unique solution R_H. According to Inference 1, for CNF formulas F_C and its obfuscated formula F_O, we have:

1) F_C is unsatisfiable iff F_O is unsatisfiable. And the unsatisfiable core of F_C can be obtained from unsatisfiable core of F_O by deleting literals in F_H.
2) F_C is satisfiable iff F_O is satisfiable. And the solution of F_C can be obtained by projecting solution of F_O into variables set of F_C .

As a result, each solution of F_O can be partitioned into solutions of F_C and F_H. So the solution of F_C can be extracted from that of F_O by projection on variables set of F_C.

If variables from F_H are assigned with other solution except R_H, that is $(S_H \neq R_H)$, then we have:

$F_O(S_H/V_{F_O}) = Obf(F_C, F_H(S_H/V_{F_H}), R_H)$.

Since R_H is not solution of $F_H(S_H/V_{F_H})$, obfuscation may expand solution space of F_C, that is: If F_O is satisfied, solution acquired by projection on variables set of F_C may be false solution. We can rule out this situation by confining F_H being assigned with R_H when recovering solution.

In conclusion, the solution space of F_O is overapproximation of F_C. As a result, F_O can be solved with the same SAT solver as F_C, but solution of F_C can not be recovered from that of F_O without knowing R_H.

4) Circuit Structure Aware strategy: Through SSH, OBFUSCATOR can insert new literals into clauses and generate new clauses, while ensuring the solution space under control. But which clause should be inserted with literal and what forms of new clause should be generated remain a question.

Since gates are basic blocks to construct circuit, and CNF signature and key clause are clues to detect circuit structure, as mentioned in Subsection *IV-C1)*. We try to change the CNF signature and key clause of gate by adding literals and clauses. Furthermore, in order to mislead adversary, literals and clauses added may construct new legal CNF signature with clauses in original CNF signature, so as to hide original circuit structure seamlessly.

Algorithm 2: OBFUSCATOR

Data: The original CNF F_C, Husks CNF F_H, Husks result R_H

Result: The obfuscated CNF F_O, variable mapping M

```
1  begin
2      mark(F_C);
3      foreach c ∈ F_C do
4          if c ∈ Key Clause Set then
5              lit =get literal ∈ R_H;
6              c = c ∪ ¬lit;
7              nc = generate_new_clause(c, lit);
8              F_C = F_C ∪ nc;
9      foreach c ∈ F_C do
10         averagelen = (σ_{c'∈F_C} |c'|) / |F_C| ;
11         while |c| < averagelen do
12             lit =get literal ∈ R_H ;
13             while ¬lit ∈ c do
14                 lit=get literal ∈ R_H ;
15             c = c ∪ ¬lit;
16         M =remap all variable in F_C ∪ F_H ;
17         F_O =reorder all clause in F_C ∪ F_H ;
```

$$a = \wedge(b,c) \rightarrow \begin{Bmatrix} c1:(a \vee \neg b \vee \neg c) \\ c2:(\neg a \vee b) \\ c3:(\neg a \vee c) \end{Bmatrix} \quad a = \wedge(b,\ c,\ \mathrm{A}) \rightarrow \begin{Bmatrix} c1:(a \vee \neg b \vee \neg c \vee \neg \mathrm{A}) \\ c2:(\neg a \vee b) \\ c3:(\neg a \vee c) \\ c4:(\neg a \vee A) \end{Bmatrix}$$

a)AND2 gate a *b)AND2 gate a after obfuscation*

Figure 2. Obfuscating AND2 into AND3.

For example, Figure2a) is CNF signature of AND2 gate a. By inserting A into key clause c_1 and generating clause c_4 with A and a, we transform gate a from AND2 into AND3, with a new input variable A, which is distinguishable with b and c, the input variables of AND2. The clauses for OR, NAND, and NOR gates, which are quite similar to that of AND gates, can also be transformed in this way.

5) All in One: OBFUSCATOR algorithm

The proposed OBUFSCATOR algorithm obfuscates CNF formula F_C with SSH rules and CSA strategies, so as to prevent structure of F_C and its accurate solution from being known by adversary.

To achieve these goals, OBFUSCATOR detect gates in CNF formula, then transform them into gates with different CNF signature. Detailed implementation of OBFUSCATOR is in Algorithm 2, which use $mark$ (line 2) to detect key clauses and output variables in CNF formula, and use $generate\_new\_clause$ (line 7) to generate new clause. As all circuits can be represented by a combination of AND2 and INV, and the $mark$ algorithm for INV is trivial, so

Algorithm 3: **mark** and **generate_new_clause**

```
1  mark;
   Data: CNF formula S
   Result: marked S
2  begin
3      foreach (C ∈ S) & (|C| ≡ 3) do
4          foreach l ∈ C do
5              foreach
               (C_1 ∈ S) & (¬l ∈ C_1) & (|C_1| ≡ 2) do
6                  foreach l_1 ∈ C_1 do
7                      if (¬l_1 ∈ C) & (l_1 ≠ l) then
8                          match++ ;
9      if match ≡ 2 then
10         mark l as output literal ;
11         mark C as Key Clause;
12 generate_new_clause;
   Data: key clause C in AND2, Husk literal lit
   Result: new clause C_1
13 begin
14     olit=Getting output literal from C ;
15     C_1 = lit ∪ ¬olit ;
```

we only present the implementation of $mark$ for AND2 in Algorithm 3. Similarly, we also present only the implementation of **generate_new_clause** for AND2 in Algorithm 3. These two algorithms can transform a CNF signature of AND2 to that of AND3.

SSH and CSA based obfuscation procedure implemented in Algorithm 2 and 3 is described below.

***Procedure** 1 (Obf_{SSH_CSA}):*

1) Input: Formula F_C, Husks formula F_H, solution R_H.
2) Output: Formula F_O.

According to Algorithm 2, F_C consists of **key clause**(line 4) and **non-key clause**, corresponding clause sets denoted as F_{Ck} and F_{Cn}.

Step 1: For key clause $c \in F_{Ck}$, take one literal lit from R_H, and insert $\neg lit$ into c (at line 6, 15 in Algorithm 2) according to SSH rule 1. The resulting clause set is denoted as S_3 .

Step 2: Generating new clauses (line 7 in Algorithm 2) with literal lit from R_H and output variable of c in F_C according to SSH rule 2 (line 15 in Algorithm 3). New clauses set generated in this way is denoted as S_4.

Step 3: Combining and randomly reordering S_3, S_4, F_H, and F_{Cn}, to produce F_O (line 6, 15, 8 17 in Algorithm 2).

end Procedure.

D. Solution recovery

After SAT Solving finished in public Cloud, S_O, the solution of F_O, will be returned to the private Cloud. In accordance with OBFUSCATOR, MAPPER and VERIFIER are used to filter solution of F_C out from S_O. MAPPER and VERIFIER are implemented in Algorithm 4.

Algorithm 4: MAPPER and VERIFIER

```
   Data: Obfuscated result S_O, variable mapping table
         M, Husk result R_H
   Result: Result S_C
1  begin
2      if S_O is UNSAT return UNSAT ;
3      foreach lit ∈ S_O do
4          var = abs(lit);
5          rvar = M[var].variable;
6          if M[var].formula is F_C then
7              S_C[rvar] = lit > 0?rvar : ¬rvar ;
8          else
9              Hlit = lit > 0?rvar : ¬rvar;
10             if R_H[rvar] ≠ Hlit then
11                 alert("Get another Solution from SAT
                   Solver");
12                 break;
13     printf(" SAT solution is S_C ");
```

According to Theorem 2, If result is UNSAT, then the original CNF formula is UNSAT (line 2). If result is SAT, MAPPER (line 5-7) projects solution into variables of F_C and F_H, to get S_C and S_H, which are the candidate solution of F_C and F_H respectively. VERIFIER (line 9-8) checks if S_H is equal to R_H, if yes, S_C is real solution of F_C. Otherwise, S_C may be false solution, hence, it is necessary to ask for a new solution from SAT Solver(at line 11).

The solution projection is done according to the variable mapping table M, generated by OBFUSCATOR(at Line 16 in Algorithm 2). $M[var].variable$ (at Line 5) represents the original variables name of var, and $M[var].formula$ (at Line 6) may be F_C or F_H, which the var belongs to.

V. Correctness, Effectiveness and Complexity

A. Correctness

According to Theorems 1 and 2, under SSH rules, original CNF formula can be blended with Husks formula seamless, without narrowing down the solution space. In this section, we prove these theorems. First of all, let's introduce some lemmas.

***Lemma** 1 (Husks Equation(HE)):* For Husks formula F_H and $|V_{F_H}| = n$, with its all m solutions $\{S_{H_l} | 1 \leqslant l \leqslant m\}$, and $S_{H_l} = \{y_k = B_{l_k} | B_{l_k} \in \{T, F\}, 1 \leqslant k \leqslant n\}$.

For each S_{H_l}, let $F_{slH} = (\bigwedge_{1 \leqslant i \leqslant n}^{B_{l_i} \equiv T} y_i) \wedge (\bigwedge_{1 \leqslant j \leqslant n}^{B_{l_j} \equiv F} \neg y_j)$, and $F_{sH} = \bigvee_{1 \leqslant l \leqslant m} F_{slH}$,

then $F_{sH} \equiv F_H$.

Proof:

1) Since $F_{sH} \equiv T$ then there must exist

$$F_{slH} = (\bigwedge_{1 \leqslant i \leqslant n}^{B_{l_i} \equiv T} y_i) \wedge (\bigwedge_{1 \leqslant j \leqslant n}^{B_{l_j} \equiv F} \neg y_j) \equiv T \tag{3}$$

Then we have:

$$S_{H_l} = \{y_i = T, y_j = F | B_{l_i} \equiv T, B_{l_j} \equiv F, 1 \leqslant i, j \leqslant n\} \tag{4}$$

Let B_i, B_j substitutes T in $y_i = T$ and F in $y_j = F$ respectively, then we have

$$S_{H_l} = \{(y_i = B_{l_i}, y_j = B_{l_j}) | B_{l_i} \equiv T, B_{l_j} \equiv F, 1 \leqslant i, j \leqslant n\} \tag{5}$$

Since S_{H_l} is one solution of F_H, then $F_H(S_H/V_{F_H})$ is true. thus we have:

$$F_{sH} \vdash F_H \tag{6}$$

2) Since $F_H \equiv T$, then there exists a solution of F_H, shown in Equation (7), that make $F_H(S_{H_1}/V_{F_H})$ true.

$$S_{H_1} = \{y_k = B_{1_k} | B_{1_k} \in \{T, F\}, 1 \leqslant k \leqslant n\}. \tag{7}$$

Construct F_{s1H} according to (7), and we have Equation (9):

$$F_{s1H} = (\bigwedge_{1 \leqslant i \leqslant n}^{B_{1_i} \equiv T} y_i) \wedge (\bigwedge_{1 \leqslant j \leqslant n}^{B_{1_j} \equiv F} \neg y_j) \tag{8}$$

$$F_{sH} = F_{s1H} \vee (\bigvee_{2 \leqslant l \leqslant m} F_{slH}). \tag{9}$$

Since $F_{s1H} \equiv T$, with Equation (8) and (9), we have:

$$F_{sH} \equiv T \tag{10}$$

$$F_H \vdash F_{sH} \tag{11}$$

According to Equation (6) and (11), we have:

$$F_{sH} \equiv F_H \tag{12}$$

■

***Lemma** 2 (Singular Husk Equation(SHE)):* For singular Husk formula F_H with $|V_{F_H}| = n$, and unique solutions S_H=$\{(y_i = B_i, y_j = B_j) | B_i \equiv T, B_j \equiv F, 1 \leqslant i, j \leqslant n\}$.

let $F_{sH} = F_H \wedge (\bigwedge_{1 \leqslant i \leqslant n}^{B_i \equiv T} y_i) \wedge (\bigwedge_{1 \leqslant j \leqslant n}^{B_j \equiv F} \neg y_j)$

then $F_H \equiv F_{sH}$.

Proof: Abbr. Similar to Lemma 1. ■

According to Lemma 1 and 2, a singular Husk formula is equivalent to conjunction of all its solution literals, and a Husks formula is equivalent to disjunction of its solutions clauses, while each solution clause is conjunction of literals in the solution.

***Lemma** 3 (OR Hold Obfuscation):* For formula F_C and $F_{RH} \vee F_{SH}$, with R_H is an assignment of $F_{RH} \vee F_{SH}$, then

$Obf(F_C, F_{RH} \vee F_{SH}, R_H) \equiv Obf(F_C, F_{RH}, R_H) \vee Obf(F_C, F_{SH}, R_H)$

Proof: Assume

- F_C=$F_{Ck} \wedge F_{Cn}$, F_{Ck}= $\bigwedge_1^m (a_i \vee X_i)$.
- F_H=$F_{RH} \vee F_{SH}$
- R_H=$\{y_j = B_j | B_j \in \{T, F\}, 1 \leqslant j \leqslant n\}$.
- Let $F_O = Obf(F_C, F_H, R_H)$

According to **Procedure** 1, construct F_O as following 3 **steps**.

1) With $(y_j \equiv B_j) \in R_H$, $(a_i \vee X_i) \in F_{Ck}$ and Rule 1:
   if $B_j \equiv T$, then clause $C_{ij} = (a_i \vee X_i) \wedge \neg y_j$.
   if $B_j \equiv F$, then clause $C_{ij} = (a_i \vee X_i) \wedge y_j$.
   and $S_3 = \bigwedge_{1 \leqslant i \leqslant m}^{1 \leqslant j \leqslant n} C_{ij}$
2) With $(y_j \equiv B_j) \in R_H$, $(a_i \vee X_i) \in F_{Ck}$ and Rule 2:
   if $B_j \equiv T$, then clause $D_{ij} = \neg a_i \wedge y_j$.
   if $B_j \equiv F$, then clause $D_{ij} = \neg a_i \wedge \neg y_j$.
   and $S_4 = \bigwedge_{1 \leqslant i \leqslant m}^{1 \leqslant j \leqslant n} D_{ij}$.
3) Let $F_{aC} = S_3 \wedge S_4 \wedge F_{Cn}$, then $F_O = F_H \wedge F_{aC}$.

According to Step 3):

$$\begin{aligned} F_O = F_H \wedge F_{aC} \qquad F_H = F_{RH} \vee F_{SH} \quad &\models \\ F_O = (F_{RH} \vee F_{SH}) \wedge F_{aC} \quad &\models \\ F_O = (F_{RH} \wedge F_{aC}) \vee (F_{SH} \wedge F_{aC}) & \end{aligned} \tag{13}$$

According to **Procedure** 1, F_{aC} is only relevent to F_C and R_H, then

$$F_{SH} \wedge F_{aC} \equiv Obf(F_C, F_{SH}, R_H) \tag{14}$$

$$F_{RH} \wedge F_{aC} \equiv Obf(F_C, F_{RH}, R_H) \tag{15}$$

According to Equation (13), (14), (15), we have:

$$F_O \equiv Obf(F_C, F_{RH}, R_H) \vee Obf(F_C, F_{SH}, R_H) \tag{16}$$

■

***Lemma** 4 (AND Hold Obfuscation):* For formula F_C and $F_{RH} \wedge F_{SH}$, with R_H is an assignment of $F_{RH} \wedge F_{SH}$, then

$Obf(F_C, F_{RH} \wedge F_{SH}, R_H)$
$\equiv Obf(F_C, F_{RH}, R_H) \wedge Obf(F_C, F_{SH}, R_H)$

Proof: Abbr. Similar to Lemma 3. ■

According to Lemma 3 and 4, for Husk formula F_H, AND and OR relation are true after obfuscation.

***Lemma** 5 (Unique Positive literal SSE Obfuscation):* For any CNF formula F_C, we have

$Obf(F_C, B = b, b \equiv T) \equiv F_C \wedge b$

Proof: Assume

- F_C=$F_{Ck} \wedge F_{Cn}$, $F_{Ck} = A$, $A = a \vee X$.
- F_H=B, $B = b$ while $b \notin X$, R_H=$\{b \equiv T\}$.
- Let $F_O = Obf(F_C, F_H, R_H)$

According to **Procedure** 1, Construct F_O as following 3 **steps**.

1) With $(b \equiv T) \in R_H$ and Rule 1, we have clause $C = A \vee \neg b$, and $S_3 = C$.
2) With $(b \equiv T) \in R_H$, literal $a \in A$ and Rule 2, we have clause $D = \neg a \vee b$, and $S_4 = D$.
3) let $F_O = F_H \wedge S_3 \wedge S_4 \wedge F_{Cn}$.

According to Step 3):

$$\begin{array}{ccc} F_O = F_H \wedge S_3 \wedge S_4 \wedge F_{Cn} & S_3 = C\ S_4 = D & \models \\ F_O = F_H \wedge C \wedge D \wedge F_{Cn} & F_H = B\ B = b & \models \\ F_O = b \wedge C \wedge D \wedge F_{Cn} & C = A \vee \neg b & \models \\ F_O = b \wedge (A \vee \neg b) \wedge D \wedge F_{Cn} & & \models \\ F_O = b \wedge A \wedge D \wedge F_{Cn} & D = \neg a \vee b & \models \\ F_O = b \wedge A \wedge (\neg a \vee b) \wedge F_{Cn} & & \models \\ F_O = b \wedge A \wedge F_{Cn} & F_{Ck} = A & \models \\ F_O = b \wedge F_{Ck} \wedge F_{Cn} & F_C = F_{Ck} \wedge F_{Cn} & \models \\ F_O = F_C \wedge b & & \end{array} \tag{17}$$

■

***Lemma** 6 (Unique Negative literal SSE Obfuscation):* For any CNF formula F_C, we have

$Obf(F_C, B = \neg b, b = F) = F_C \wedge \neg b$

Proof: Abbr. Similar to Lemma 5. ■

According to Lemma 5 and 6, after unique literal obfuscation, solution space is unchanged.

Then let's discuss SSE Obfuscation based on singular Husk formula, and SSO Obfuscation based on Husks formula.

Theorem 1 Solution Space Equated (SSE) Obfuscation

For arbitrary CNF formula F_C, and Singular Husk formula $F_{_S}H$, if

- $V_{F_C} \cap V_{F_H} = \phi$, R_H is unique solutions of F_H.
- $F_O = Obf(F_C, F_H, R_H)$.

then $F_C \wedge F_H \equiv F_O$.

Proof:

Assume R_H=$\{y_k = B_k | B_k \in \{T, F\}, 1 \leqslant k \leqslant n\}$.

According to **Procedure** 1, constuct F_O as following **steps**:

1) Let $F_{Op} = F_C$.
   for $y_i \in \{y_i | (y_i = B_i) \in R_H \parallel B_i \equiv T)\}$, let
   F_{Op}=$Obf(F_{Op}, B = y_i, y_i \equiv B_i)$.
2) Let $F_{On} = F_{Op}$.
   for $y_j \in \{y_j | (y_j = B_j) \in R_H \parallel B_j \equiv F)\}$, let
   F_{On}= $Obf(F_{On}, B = \neg y_j, y_j \equiv B_j)$.
3) $F_O = F_{On} \wedge F_H$.

According to Lemma 4, 5 and Step 1), we have:

$$F_{Op} \equiv F_C \wedge (\bigwedge_{1 \leqslant i \leqslant n}^{B_i \equiv T} y_i) \tag{18}$$

According to Lemma 4, 6 and Step 2), we have:

$$F_{On} \equiv F_{Op} \wedge (\bigwedge_{1 \leqslant j \leqslant n}^{B_j \equiv F} \neg y_j). \tag{19}$$

According to Step 3) and Equation (18) (19), we have:

$$F_O \equiv F_C \wedge (\bigwedge_{1 \leqslant i \leqslant n}^{B_i \equiv T} y_i) \wedge (\bigwedge_{1 \leqslant j \leqslant n}^{B_j \equiv F} \neg y_j) \wedge F_H \tag{20}$$

Since R_H is the unique satisfied solution of F_H, according to Lemma 2, we have:

$$F_H \wedge (\bigwedge_{1 \leqslant i \leqslant n}^{B_i \equiv T} y_i) \wedge (\bigwedge_{1 \leqslant j \leqslant n}^{B_j \equiv F} \neg y_j) \equiv F_H \tag{21}$$

According to Equation (20), (21), we have:

$$F_O \equiv F_C \wedge F_H \tag{22}$$

Since F_H is satisfiable, $V_{F_C} \cap V_{F_H} = \phi$, we have Inference:

$$F_O \equiv_{SSE} F_C \tag{23}$$

■

Theorem 2 Solution Space Overapproximated (SSO) Obfuscation

For arbitrary CNF formula F_C, Husks formula F_H, if

1) $V_{F_C} \cap V_{F_H} = \phi$, R_H is one of m solutions of F_H.
2) $F_O = Obf(F_C, F_H, R_H)$.

Then $F_C \vdash_{SSO} F_O$.

Proof:

Assume one solution of F_H is R_H=$\{y_i = B_{R_k} | B_{R_k} \in \{T, F\}, 1 \leqslant k \leqslant n\}$, and its all other $m-1$ solutions $\{S_{H_l} | 1 \leqslant l \leqslant m-1\}$. $S_{H_l} = \{y_k = B_{l_k} | B_{l_k} \in \{T, F\},\ 1 \leqslant k \leqslant n\}$.

According to R_H and S_H, let's define $F_{_R}H$ and $F_{_S}H$:

$$F_{_R}H = (\bigwedge_{1 \leqslant i \leqslant n}^{B_{R_i} \equiv T} y_i) \wedge (\bigwedge_{1 \leqslant j \leqslant n}^{B_{R_j} \equiv F} \neg y_j) \tag{24}$$

$$F_{_S}H = \bigvee_{1 \leqslant l \leqslant m-1} ((\bigwedge_{1 \leqslant i \leqslant n}^{B_{l_i} \equiv T} y_i) \wedge (\bigwedge_{1 \leqslant j \leqslant n}^{B_{l_j} \equiv F} \neg y_j)) \tag{25}$$

According to Lemma 1, we have:

$$F_{_R}H \vee F_{_S}H \equiv F_H \tag{26}$$

With $F_O = Obf(F_C, F_H, R_H)$ and Equation (26), we have:

$$F_O \equiv Obf(F_C, F_{_R}H \vee F_{_S}H, R_H) \tag{27}$$

According to Lemma 3 and Equation (27), we have:

$$F_O = Obf(F_C, F_{_R}H, R_H) \vee Obf(F_C, F_{_S}H, R_H) \tag{28}$$

According to Theorem 1, we have:

$$Obf(F_C, F_{_R}H, R_H) \equiv F_C \wedge F_{_R}H \tag{29}$$

With Equation (28) and (29), we have:

$$F_O \equiv (F_C \wedge F_{_R}H) \vee Obf(F_C, F_{_S}H, R_H) \tag{30}$$

With Equation (30), we have:

$$F_C \wedge F_{_R}H \vdash_{SSO} F_O \tag{31}$$

Since F_{RH} is satisfiable, $V_{F_C} \cap V_{F_H} = \phi$, with Equation (31):

$$F_C \vdash_{SSO} F_O \tag{32}$$

■

B. Effectiveness

1) Input Obfuscation through Transforming circuit structure: By appending redundant literals and clauses, OBFUSCATOR can change signatures in CNF formula into other legal signatures. After obfuscation, the original CNF formula is transformed into another formula, mixed with noisy circuit structure. Since obfuscated CNF formula is outsourced as input of SAT solver, circuit structure in original CNF formula will not be exposed to adversary.

Figure 3a) and 3b) shows the CNF signatures of two AND2 gates a and e, while their CNF signatures after obfuscation are shown in Figure 3c) and 3d).

There are three types of changes:

1) The length of key clauses c_1 and c_5 are changed from 3 to 4, this defeats structure detection techniques [11] based on key clause oriented pattern matching;
2) CNF signatures of a (characteristic clauses c_1-c_3) and e (characteristic clauses c_5-c_7) are changed into different forms, and there are new clauses added in formula, such as c_4 and c_8, This defeats structure detection techniques [10] based on sub-graph isomorphic;
3) By inserting proper literals in key clauses and generating new clause, CNF signature of gate a is changed from AND2 to AND3, shown in Figure 3a) and 3c). Husk variable A, which becomes an input variable of gate AND3, is indistinguishable with b and c, which are original input variables of AND2. This makes it impossible to distinguish gates AND2 and AND3.

2) Output Camouflage by over-approximating solution space: According to Theorem 2, the solution space after SSO obfuscation is an over-approximated version of the original one. That means, even SAT solver can't find out the real solution. First, they can not tell real valuable variables from variables of Husks formula, which are meaningless to verification. Second, they cannot tell if a satisfied solution

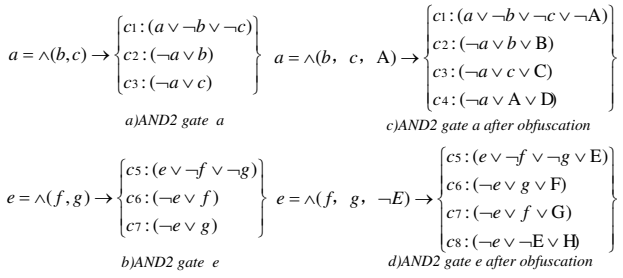

Figure 3. CNF signature of a and e before and after obfuscation

means whether the original SAT problem is also satisfiable, because some false solutions are produced by obfuscation. Through overapproximation, We just turn an obvious Rare Events into a Camouflaged Rare Events, as anticipation in literature [19] .

C. Complexity

1) Obfuscation algorithm complexity: Obfuscation is implemented in Algorithm 2. The main procedure of Algorithm 2 consists only one layer of loop, but one of it sub-procedure **mark** (Algorithm 3) consists 4 layers of loop, and the runtimes of the 2 inner loops are bounded by length of clauses. So the complexity of the obfuscation algorithm is $O(n^2)$.

2) Solution recovery algorithm complexity: Solution recovery is implemented in Algorithm 4, which only consists one layer of loop, its complexity is $O(n)$. According to Theorem 2, result from SAT solver may consist false solution, so Algorithm 4 may be run more than one time to get correct solution. Since Algorithm 4 is of linear complexity, it incurs minor impact on performance of SAT Solving.

VI. Related work

Secure Computation Outsourcing based on encryption: R. Gennaro et al. [18] 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. [17] 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. [20] multiplied data with random diagonal matrix before outsourcing. and recovered results by reversible matrix operations. Paper [21] discussed secure outsourcing of numerical and scientific computation, by disguising with a set of problem dependent techniques. C.Wang [7] 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. [19] injected a number of chaff items into the workloads so as to confuse dishonest participants. Golle et al. [22] proposed to

insert some pre-computed results images of ringers into the computation workload to defeat untrusted or lazy workers. Szada et al. [23] extended the ringer scheme and propose methods to deal with cheating detection.

VII. Experiments

Algorithms presented in this paper are 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 100 times and transform them into CNF formulas, and generate Husks formula with variables number $vn = 675$ and clauses number $cn = 2309$, and then obfuscate the CNF formula by transform 2 input gates into 3 input gates. We use MiniSat as solver.

Table I presents experiments result on benchmarks, meaning of parameters are listed below.

vn/cn:variable and clause number of CNF formula.

Marked Gate:number of gates being changed in obfuscation.

Solve Times:SAT Solver time before and after obfuscation.

Obfuscation Times:obfuscation time.

Map Time:solution recovery time.

Acccording to Algorithm 2, Obfusaction time is up to number of gates being changed, while solution recovery time is up to size of CNF formula, experiments manifest the fact.

As for Asymmetric Speedup [7], for more than 60 % of circuits, the value is more than 260%, It indicates the necessity of outsourcing complex SAT solving. But for some small size circuits, Asymmetric Speedup is less than 1. Especially for circuit s3384, since the obfuscation takes lots of time to transform 139860 gates, Asymmetric Speedup is only 5.22%.

$$Asymmetric\ Speedup = \frac{Solving\ Time}{Obfuscation\ Times + Map\ Time} \tag{33}$$

The experiments also show that overhead of SAT solving time, incurred by obfuscation, are different among circuits. For more than 60 % of circuits, overhead is less than 30%; But for the other 40% circuits, overhead exceed 100%.

These facts remind us at least two things: First, since obfuscation time is up to gates being changed, delicate obfuscation algorithm which change less gates but still can mislead adversary should be studied. Second, since overhead on SAT solving incurred by obfuscation are different among circuit benchmarks, much more attention should be pay on the impact on solving time, when designing obfuscation algorithm.

VIII. Conclusion

This paper proposes a circuit aware CNF obfuscation algorithm, that can prevent the confidential information from being recovered by adversary, when outsourcing SAT problem in Cloud or grid. Theoretical analysis and experimental results show that algorithms can significantly change structure of CNF formula, with polynomial complexity and without narrowing down its solution space.

Acknowledgment

This work was funded by projects 61070132 and 61133007 supported by National Natural Science Foundation of China.

References

[1] M. Davis, H. Putnam: A Computing Procedure for Quantification Theory. J. ACM 7(3): 201-215 (1960)

[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, Helmut Veith: Counterexample-Guided Abstraction Refinement. CAV 2000: 154-169

[4] M. Soos, K. Nohl, C. Castelluccia: Extending SAT Solvers to Cryptographic Problems. SAT 2009: 244-257

[5] A. Hyvärinen, T. Junttila, I. Niemelä: Grid-Based SAT Solving with Iterative Partitioning and Clause Learning. CP 2011: 385-399

[6] G. Tseitin. On the complexity of derivation in propositional calculus. Studies in Constr. Math. and Math. Logic, 1968.

[7] C. Wang, K. Ren, J. Wang: Secure and practical outsourcing of linear programming in cloud computing. INFOCOM 2011: 820-828

[8] C. Li: Integrating Equivalency Reasoning into Davis-Putnam Procedure. AAAI/IAAI 2000: 291-296

[9] R. Ostrowski, É. Grégoire, B. Mazure, L. Sais: Recovering and Exploiting Structural Knowledge from CNF Formulas. CP 2002: 185-199

[10] J. Roy, I. Markov,V. Bertacco: Restoring Circuit Structure from SAT Instances. IWLS'04, pp. 361-368.

[11] Z. Fu, S. Malik: Extracting Logic Circuit Structure from Conjunctive Normal Form Descriptions. VLSI Design 2007: 37-42

[12] MiniSat-SAT Algorithms and Applications Invited talk given by Niklas Sorensson at the CADE-20 workshop ESCAR. http://minisat.se/Papers.html

[13] M. Balduzzi, J. Zaddach, D. Balzarotti, E. Kirda, S. Loureiro: A security analysis of amazon's elastic compute cloud service. SAC 2012: 1427-1434

[14] T. Ristenpart, E. Tromer, H. Shacham, S. Savage: Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. CCS 2009: 199-212

[15] D. Achlioptas, C. Gomes, H. Kautz, B. Selman: Generating Satisfiable Problem Instances. AAAI/IAAI 2000: 256-261

Table I
RUNTIME OF CNF FORMULA GENERATED FROM DIFFERENT CIRCUIT

circuit	vn	cn	Marked Gates	Solve Time(s)			Obfuscation Time(s)	Map Time(s)	Asymmetric Speedup
				Before obfuscation	After obfuscation	Solve Overhead			
s1196a	13970	50011	4397	0.056003 s	0.068004 s	21.43%	0.076004 s	0.004000 s	***70.00%***
s1196b	13970	50011	4397	0.048003 s	0.060003 s	25.00%	0.064004 s	0.012000 s	***63.16%***
s1196	13970	50011	4397	0.048003 s	0.080005 s	66.67%	0.072004 s	0.000000 s	***66.67%***
s13207	34423	117739	6221	0.280017 s	0.316019 s	12.86%	0.284017 s	0.016001 s	***93.33%***
s3384	60503	210546	*139860*	5.04031 s	12.4368 s	*146.75%*	*96.450027s*	0.132008 s	***5.22%***
s15850	181772	668093	41471	16.361 s	46.0069 s	*181.20%*	1.392087 s	0.048003 s	1136.11%
s3271	57976	219807	14597	5.97237 s	6.4044 s	7.23%	1.396087 s	0.020001 s	421.75%
s3330	49635	159813	9679	7.9405 s	7.28045 s	**-8.31%**	0.264016 s	0.016001 s	2835.72%
s38417	492876	1939718	9563	652.685 s	3538.12 s	442.09%	0.412025 s	0.012000 s	153926.07%
s38584	593381	2227470	66491	496.847 s	1092.3 s	119.85%	9.756609 s	0.120007 s	5030.54%
s4863	85963	320633	*128896*	352.746 s	352.606 s	**-0.04%**	*133.788361s*	0.140008 s	263.38%
s5378	59462	209372	14096	11.9407 s	8.73655 s	**-26.83%**	0.428026 s	0.020001 s	2665.17%
s6669	115303	408306	10868	1.31608 s	1.6081 s	22.19%	0.296018 s	0.024001 s	411.25%
s9234	87484	337020	15392	244.627 s	292.058 s	19.39%	0.728045 s	0.032002 s	32185.77%
s35932	584534	2135163	23418	282.17 s	714.573 s	*153.24%*	0.676042 s	0.024001 s	40307.52%

[16] M Jarvisalo. Equivalence checking hardware multiplier designs. SAT Competition 2007 benchmark description.

[17] Z. Brakerski, G. Rothblum: Black-box obfuscation for d-CNFs. ITCS 2014: 235-250

[18] R. Gennaro, C. Gentry, B. Parno: Non-interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers. CRYPTO 2010: 465-482

[19] W. Du, M. Goodrich: Searching for High-Value Rare Events with Uncheatable Grid Computing. ACNS 2005: 122-137

[20] M. Atallah, K. Pantazopoulos, J. Rice, E. Spafford: Secure outsourcing of scientific computations. Advances in Computers 54: 215-272 (2001)

[21] M. Atallah, J. Li: Secure outsourcing of sequence comparisons . Int. J. Inf. Sec. 4(4): 277-287 (2005)

[22] P. Golle, I. Mironov: Uncheatable Distributed Computations. CT-RSA 2001: 425-440

[23] D. Szajda, B. Lawson, J. Owen: Hardening Functions for Large Scale Distributed Computations. IEEE Symposium on Security and Privacy 2003: 216-224

[24] Paralleling OpenSMT Towards Cloud Computing http://www.inf.usi.ch/urop-Tsitovich-2-127208.pdf

[25] Formal in the Cloud OneSpin: New Spin on Cloud Computing.http://www. eejournal.com/archives/articles/20130627-onespin/?printView=true

[26] X.S. Zhang, F.L. He and W.l. Zuo. Theory and Practice of Program Obfuscation. Convergence and Hybrid Information Technologies, Book edited by: Marius Crisan, ISBN 978-953-307-068-1, pp. 426, March 2010, INTECH, Croatia.

[27] G. Karypis, R. Aggarwal, V. Kumar, S. Shekhar: Multilevel Hypergraph Partitioning: Application in VLSI Domain. DAC 1997: 526-529