IVeri: Privacy-Preserving Interdomain Verification

Technical Report

Ning Luo[‡], Qiao Xiang[†‡], Timos Antonopoulos[‡], Ruzica Piskac[‡], Y. Richard Yang[‡], Franck Le[*]
[†]Xiamen University, [‡]Yale University, [*]IBM Watson Research Center

***Abstract*—In an interdomain network, autonomous systems (ASes) often establish peering agreements, so that one AS (agreement consumer) can influence the routing policies of the other AS (agreement provider). Peering agreements are implemented in the BGP configuration of the agreement provider. It is crucial to verify their implementation because one error can lead to disastrous consequences. However, the fundamental challenge for peering agreement verification is how to preserve the *privacy* of both ASes involved in the agreement. To this end, this paper presents IVeri, the first privacy-preserving interdomain agreement verification system. IVeri models the interdomain agreement verification problem as a SAT formula, and develops a novel, efficient, privacy-serving SAT solver, which uses oblivious shuffling and garbled circuits as the key building blocks to let the agreement consumer and provider collaboratively verify the implementation of interdomain peering agreements without exposing their private information. A prototype of IVeri is implemented and evaluated extensively. Results show that IVeri achieves accurate, privacy-preserving interdomain agreement verification with reasonable overhead.**

I. Introduction

An interdomain network (*e.g.*, the Internet [1] and the Large Hadron Collider science network [2]) connects multiple autonomous systems (ASes) and exchanges traffic among them. The *de facto* interdomain routing protocol is the Border Gateway Protocol [3]. ASes use BGP to exchange routing information. In addition, BGP also allows each AS to independently apply its local routing policies to select and advertise interdomain routes.

Due to economic or collaborative incentives, two peering ASes often establish *peering agreements* so that one AS (*agreement consumer*) can influence the routing policies of the other AS (*agreement provider*). Such agreements are implemented in the configurations of the provider's BGP routers. Consider an interdomain network in Figure 1. An example of a peering agreement between ASes A and B is that B will send all A's traffic towards F through D. In this example, A is the agreement consumer, and B is the agreement provider, who implements this agreement in its BGP configurations.

The correct implementation of peering agreements is crucial for ASes to realize their network management goals, *e.g.*, defense against distributed denial-of-service (DDoS) attacks, inbound and outbound traffic engineering, and fulfillment of regulations on data traffic traversal. Nevertheless, technology news have been repeatedly reporting about peering agreements not being implemented correctly, leading to disastrous results [4]–[7]. For example, due to an error in a BGP configuration in 2018, Google lost control of several millions of IP addresses for more than one hour and rerouted the internet traffic through China [8].

Ning Luo and Qiao Xiang are co-primary authors. The corresponding author is Qiao Xiang (qiaoxiang@xmu.edu.cn).

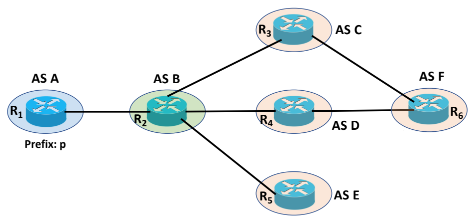


Fig. 1: An example of interdomain network to illustrate BGP and interdomain peering agreements.

As such, it is crucial to verify whether a peering agreement is implemented correctly at the provider's BGP configurations proactively, before the buggy configurations are deployed or before the agreement is actual invoked (*i.e.*, via a route announcement with community tag). However, this problem remains unsolved and non-trivial, despite the fact that many recent tools are focused on finding configuration errors in routing protocols [9]–[13]. The fundamental challenge for this problem is how to preserve the *privacy* of both ASes involved in the agreement. On one hand, the BGP configurations of the agreement provider is private for operational security [14]. On the other hand, our private conversations with some of the largest content service providers, who play the role of agreement consumer, show that if possible, they would also prefer their agreements-to-verify being kept private, for the reason that exposing these may cause their sensitive information (*e.g.*, internal operational policies and offline negotiation with other ASes) being inferred. With the privacy concerns of both parties, strawman solutions such as the client-server design and the trusted third-party design to apply recent network configuration tools are not desired. A related approach [15]–[17] is to design collaborative verification protocols in which an agreement consumer can verify how the agreement provider processes an actual route announcement with the help of all other neighboring ASes of the provider. However, this approach verifies peering agreement on per route announcement basis, *i.e.*, it can only find errors after a route announcement is sent.

In this paper, we systematically study this problem and design IVeri, the first privacy-preserving interdomain agreement verification system. In particular, we first leverage the formalism from recent network configuration verification tools [9]–

[11] to let the agreement consumer model the agreement-to-verify and the agreement provider model the BGP configurations as SMT (satisfiability modulo theories) formulas and transform to SAT formulas, respectively. Specifically, let F^B represent the BGP configurations of the agreement provider and F^C represent the SAT formula describing the agreement-to-verify of the agreement consumer. To verify that provider's configurations correctly implement the agreement-to-verify is to check whether $F^B \Rightarrow F^C$ is a tautology, which is equivalent to check whether $F^B \wedge \neg F^C$ is unsatisfiable [9]–[11]. To simplify notation, we denote $\neg F^C$ as F^A and thus the problem of privacy-preserving interdomain agreement verification is defined as to *determine the satisfiability of $F^A \wedge F^B$, while F^A and F^B are held privately by the agreement consumer and provider, respectively.*

To this end, we developed a novel, efficient, privacy-preserving SAT solver in IVeri. The basic idea of this solver consists of (1) an oblivious shuffling algorithm [18] that enables two parties to shuffle an encrypted version of the verification formula, and further ensures that the non-encrypted configuration and agreement formulas are always held private by the respective parties, and (2) the standard DPLL algorithm encoded with inexpensive garbled circuits [19], denoted by GC-DPLL. The solver determines the satisfiability of the verification formula contributed by the agreement consumer and provider without loss of their privacy on their contribution.

In summary, this paper presents the following **main contributions**:

- We systematically study an important, real problem of interdomain peering agreement verification, and present IVeri, a privacy-preserving interdomain agreement verification system, which is, to the best of our knowledge, the first system to verify the agreement implementation in BGP configurations;
- IVeri leverages and expands the modeling technique in recent network verification tools and establishes the interdomain peering agreement verification problem as a privacy-preserving SAT problem;
- IVeri designs a novel, efficient privacy-preserving SAT solver that determines the satisfiability of the verification formula without leaking the agreement consumer or provider sensitive information. In addition to interdomain verification, this SAT solver also has other application prospectives such as multi-domain resource orchestration via constraint programming [20];
- A prototype of IVeri is implemented and evaluated with extensive experiments. Results show that IVeri achieves accurate, privacy-preserving interdomain agreement verification with reasonable overhead.

II. Background, Motivation and Challenge

This section provides a brief background on BGP and interdomain peering agreements, demonstrates the importance of interdomain peering agreement verification, and elaborates on its fundamental challenges.

A. Background

BGP in a nutshell. BGP [3] is the *de facto* interdomain routing protocol interconnecting ASes. Specifically, each AS owns a set of IP addresses, and assigns some of its routers as BGP border routers, which are connected to BGP border routers in neighboring ASes. Figure 1 shows an interdomain network with 6 ASes, where AS A owns IP prefix p. For simplicity of the presentation, we assume that each of them has one BGP border router.

Two ASes whose BGP routers are connected to each other are called BGP peers, *e.g.*, A and B are BGP peers. BGP peers exchange routing information on how to reach destination IP prefixes through *route announcements*. Abstractly, each route announcement carries a destination prefix and a sequence of ASes to traverse to reach the destination prefix, which we call an *AS path*. In addition, a route announcement can also carry other attributes, such as *origin*, which is the origin AS of the destination prefix, or *community tags*, which are numerical values whose semantics are predefined between two BGP peers via offline communication. For example, R_2 can send a route announcement $(p, [B, A], comtag : 10 : 30)$ to R_3, which includes an destination prefix p, an AS path $[B, A]$ to reach p, and a community tag attribute of value $10 : 30$.

In BGP, each AS can make and execute its own policies to decide which routes to use (*i.e.*, selection policy), and whether or not to announce the to peers (*i.e.*, export policy). For example, an AS may prefer to selecting shorter routes for better latency, and may choose not to announce a route to a peer for business reason. In reality, these policies are implemented in the configurations of BGP routers.

BGP peering agreements. For economical or collaborative reasons, two BGP peering ASes often reach peering agreements to allow one AS (agreement consumer) to influence the routing policies of the other AS (agreement provider) [21]. Such agreements are implemented as part of the route selection/export polices in the BGP configurations of the agreement provider. Their correct implementation is essential for ASes to realize their network management goals, *e.g.*, defense against DoS attacks, inbound and outbound traffic engineering, and fulfillment of regulations on data traffic traversal. Some representative peering agreements [15], [21] are:

- **Selective export**: let the agreement provider not propagate the route announced by the consumer to certain peers of the providerr;
- **Set local preference**: let the agreement provider set its local preference of the route announced by the consumer to certain value;
- **Prefer/avoid certain AS**: let the agreement provider prefer to /avoid selecting a route containing certain AS when receiving multiple route announcement;

B. Motivation

Verifying whether an interdomain peering agreement is correctly implemented in BGP configurations is of great importance because one error can lead to harmful consequences [4]–[7]. We use the following example to demonstrate that.

A motivating example (DoS attack). Consider the interdomain network given in Figure 1, and assume that AS E launches a DoS attack to A, *i.e.*, keeps sending high volumes of malicious traffic to prefix p in A within a short time period. When A detects a DoS attack, and suspects that E is the source of this attack, A can defend by reaching a *selective-export* agreement with B: *B will not export any route announced by A to E*. If such an agreement is implemented correctly by B, B will send a new route announcement $(p, [])$ to E, also called a withdraw announcement, indicating that B cannot reach p. As a result, E cannot continue the attack because it has no route to reach p, and A can observe the stop of the attack.

However, if B does not implement this agreement correctly, E will continue having a route $B \rightarrow A$ to reach prefix p, and use this route for the attack. As such, A will observe that the attack does not stop, and shift its suspicion to another AS, *e.g.*, C, and reaches another selective-export agreement with B to not export any route announced by A to C. If this new agreement is implemented by B correctly this time, not only does A still suffer from the DoS attack, the normal traffic from C to A is also completely blocked, exacerbating the damage of this attack.

C. Challenges

Many tools have been developed to find configurations errors in routing protocols [9]–[13], [22]. As such, a strawman solution to verify whether an peering agreement is correctly implemented is to treat the agreement the customer wants to verify as a network property, let a trusted third party collect the agreement from the consumer and the configurations from the provider and run an existing configuration verification tool to verify it. However, this is not the case.

The reason, and the fundamental challenge of verifying interdomain peering agreements, is *privacy*. On one hand, given an agreement provider, its implementation of peering agreements (*i.e.*, the BGP router configuration files) is private for operational security and commercial reasons [14]. Although some work has shown it is possible to infer some aspects of ASes' BGP configurations [23], the inference results have limited accuracy and cannot be utilized for peering agreement verification.

On the other hand, for an agreement consumer, the peering agreements are also considered private. One may find this argument a little odd at a first glance, however, it is justified. Many agreements (*e.g.*, selective export, set local preference and AS path prepending) are invoked "on-demand" to achieve certain network management goals (*e.g.*, inbound traffic engineering). When to invoke which agreements is related to the operation policy of agreement consumers. As such, revealing which agreements to verify risk exposing its operation policy. For example, consider the not-export-to-E agreement between A and B. Assume additionally that it is on-demand requiring A attaching community tags in its route announcements to B. If A reveals to B that it wants to verify that B implements this agreement correctly, but does not attach any community tags in its route announcements, B may deduce that either A does not suspect E as a DoS attack, or A and E reach additional commercial collaboration that allows E to send traffic to A, both of which are private to A.

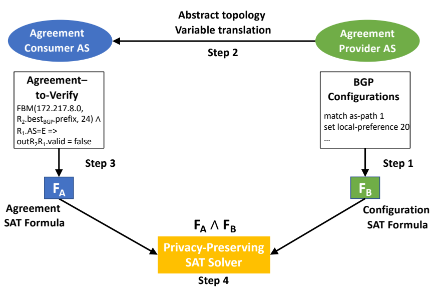


Fig. 2: The basic architecture and workflow of IVeri.

III. System Model and Problem Formulation

IVeri operates between the agreement consumer and the agreement provider (Figure 2). This section presents the system model of IVeri and the formulation of the privacy-preserving interdomain agreement verification problem.

A. System Model

We leverage the modeling technique of Minesweeper [9], a state-of-the-art intradomain network verification tool, to model the BGP configurations of the agreement provider, but go beyond their focus on intradomain network properties (*e.g.*, reachability and black-hole freeness) and safety properties of interdomain network (*e.g.*, stability), to develop new models for interdomain peering agreements.

Agreement provider: encode BGP configurations as SAT formulas. Similar as Minesweeper, in IVeri, the agreement provider extracts its BGP configurations into an SMT formula. Due to space limit, we refer readers to [9] for more details of this model. As an example, the following SMT formula specifies that a BGP router will not announce any route toward the prefix 172.217.8.0/24 to the BGP router whose IP address is 65.124.208.93.

$$FBM(172.217.8.0, best_{BGP}.prefix, 24) \wedge \mathsf{R_i.IP} = \mathsf{65.124.208.93} \Rightarrow \mathsf{out_{R_4R_i}.valid} = \mathsf{false}. \quad (1)$$

Because all the variables of the provider's SMT formula are finite-bounded, it can be easily translated to a SAT formula in CNF (*i.e.*, using bit blasting and Tseytin transformation). We call the derived formula the *configuration SAT formula*.

Agreement provider: provide abstract BGP router topology and SMT/SAT variables. The agreement provider exposes to the consumer an abstract topology of a full-mesh of the provider's BGP routers and their connections to other BGP peers, and a SMT/SAT variable translation list. The abstract topology provides sufficient topology information (*i.e.*, all BGP connections of the agreement provider) for the consumer to specify peering agreements to verify. The exposed list of SMT/SAT variable translation ensures that the variable semantics are consistent between the configuration SAT formula and the agreement SAT formula. It also allows the agreement provider to decide the scope of peering agreements that the agreement consumer can verify, so as to restrict the consumer from snooping-around with arbitrary verification requests.

Agreement consumer: encode peering agreements as SAT formulas. With the abstract topology and the SMT/SAT variable list, the agreement consumer expresses the agreement-to-verify as an SMT formula. Consider the selective-export agreement from the motivating example given in Section II-B, it can be modeled as the following SMT formula:

$$\forall \mathsf{i} = 3,4,5.\ \mathsf{FBM}(172.217.8.0, \mathsf{R}_2.\mathsf{best}_{\mathsf{BGP}}.\mathsf{prefix}, 24) \wedge \mathsf{R}_\mathsf{i}.\mathsf{AS} = \mathsf{E} \Rightarrow out_{R_2 R_i}.valid = false, \quad (2)$$

where prefix p is 172.217.8.0/24. Similarly, other common peering agreements can also be expressed as SMT formulas. However, recall that we need to check the unsatisfiability of the configuration SAT formula and the negation of the peering agreement. For the same reasons as when encoding BGP configurations, the negation of the consumer's SMT formula can also be transformed to a SAT formula in CNF. The resulting formula we call the *agreement SAT formula*.

B. Problem Formulation

After introducing the system model of IVeri, we next formally define the privacy-preserving interdomain agreement verification problem.

Notations. The following conventions are followed to differentiate scalar, vector and matrix variables. We use v to represent a scalar and $\mathbf{v} = \{v_1, v_2, \dots, v_{|\mathbf{v}|}\}$ to represent a row vector of size $|\mathbf{v}|$. We use $\mathbf{V} = \{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_{|\mathbf{V}|}\}$, with $|\mathbf{v}_i| = |\mathbf{v}_j|$, for all $i \leq j \leq |\mathbf{V}|$ to represent a matrix of $|\mathbf{V}|$ rows. The element on the i-th row and j-th column of $\mathbf{V}$ is denoted by $\mathbf{V}_{ij}$. Unless explicitly specified, operators on vectors or matrices indicate component-wise operations.

A permutation π is a bijective function: $\{1, \dots, n\} \rightarrow \{1, \dots, n\}$, for $n \in \mathbb{N}$. A *row permutation* of a matrix $\mathbf{V}$ is represented as $\{\mathbf{v}_{\pi^{-1}(1)}, \mathbf{v}_{\pi^{-1}(2)}, \dots, \mathbf{v}_{\pi^{-1}(|\mathbf{V}|)}\}$ and written as $\pi(\mathbf{V})$.

A SAT formula in CNF of m clauses (*i.e.*, $c_1 \wedge c_2 \wedge, \dots, \wedge c_m$) over n Boolean variables $x_1, x_2, \dots, x_n$, is represented by an n-by-m matrix F, where each element $\mathbf{F}_{ij}$ of $\mathbf{F}$, consists of two bits. The first bit, denoted as $\mathbf{F}_{ij}.O$, is 1 if variable x_i occurs in clause c_j, and is 0 otherwise. The second bit, denoted as $\mathbf{F}_{ij}.P$, is 1 if and only if the variable x_i occurs as a positive literal (*i.e.*, x_i) in clause c_j.

Given a pair of peering ASes, A and B denote the agreement consumer and the agreement provider, respectively. As stated in Section I, the agreement SAT formula (*i.e.*, the negation of the agreement-to-verify) generated by A is represented as $\mathbf{F}^A$, and the configuration SAT formula generated by the provider B is represented as $\mathbf{F}^B$. The privacy-preserving interdomain peering agreement verification problem is defined as follows:

Problem 1 (Privacy-Preserving Interdomain Peering Agreement Verification Problem)**.**Given an agreement consumer A with associated agreement SAT formula $\mathbf{F}^A$ and an agreement provider B with an associated configuration SAT formula $\mathbf{F}^B$, the two parties A and B should decide if the formula $\mathbf{F}^A \wedge \mathbf{F}^B$ is satisfiable in a way such that A (respectively B) does not know what $\mathbf{F}^B$ (respectively $\mathbf{F}^A$) is. The agreement provider B satisfies the agreement requirements iff formula $\mathbf{F}^A \wedge \mathbf{F}^B$ is unsatisfiable.

Security model. This paper assumes a *semi-honest* security model [19], *i.e.*, the agreement customer and the agreement provider will not deviate from the workflow specified in IVeri, but merely try to gather information during its execution [24]. This is sufficient for multiple scenarios of interdomain routing, including commercial Internet [16], collaboration science networks where ASes collaboratively conduct common tasks such as data transfers [2], and military coalition networks [25].

IV. Privacy-Preserving SAT Solver

This section presents details on our instantiation of privacy-preserving SAT solver in IVeri to verify interdomain peering agreements in a privacy-preserving manner. We first provide a short review on the theoretical foundations this solver is built upon, and then present the basic idea and details of our solver. We also conduct rigorous analysis on the correctness, efficiency and privacy-preservingness of the solver.

A. Preliminaries

DPLL algorithm. IVeri's privacy-preserving SAT solver is based on the classic DPLL algorithm, as many modern SAT solvers do [26]–[29]. In essence, DPLL is a backtracking search algorithm. Given a SAT formula in CNF represented by a matrix $\mathbf{F}$, the algorithm runs by choosing a Boolean variable x_i, assigning a truth value to it, simplifying the formula by removing all clauses that are satisfied by the truth value assignment of x_i as well as removing all literals involving the variable x_i that are false under this assignment, and then recursively checking if the simplified formula is satisfiable. If so, $\mathbf{F}$ is satisfiable. Otherwise, the algorithm uses the same recursive check by assigning the opposite truth value to x_i. To improve the efficiency of this backtracking search process, many optimizations are introduced [26]–[28]. One key optimization that most modern SAT solvers adopt into the original DPLL is *unit literal search and resolution*, which attempts to find a unit clause that contains only one unassigned variable, and assigns a truth value to this variable to satisfy this clause, before attempting anything else. This optimization substantially reduces the search space, leading to significant improvement in search efficiency.

Secure multi-party computation (SMPC). SMPC refers to a set of frameworks for multiple parties to jointly compute a function over their inputs while keeping those inputs private [19]. Although there is a rich set of SMPC related techniques, this paper focuses on the following two that are adopted in the proposed privacy-preserving SAT solver.

- **Garbled circuits**. This is a cryptographic protocol, introduced in [30], that allows two mistrusting parties to jointly evaluate any function that can be expressed as a Boolean circuit over their private inputs. An example application of this protocol is the Millionaires' Problem, where the goal is for two parties to decide who has more money without exposing how much money they each have [19]. The output of a garbled circuit can be revealed to both parties, only one specific party, or no party but be taken as inputs of other garbled circuits. For more details, readers may refer to [31].
- **Oblivious shuffling**. This technique [18] enables two par-

ties to collaboratively decide a permutation π over a vector of cipher texts $\mathbf{\Phi} = \{\phi_1, \phi_2, \ldots, \phi_{|\mathbf{v}|}\})$ such that ϕ_i is the cipher text of some v_i. The protocol provides that (1) the output vector of this protocol is the vector $\{\phi_{\pi(1)}, \phi_{\pi(2)}, \ldots, \phi_{\pi(|\mathbf{v}|)}\}$ where $\phi_{\pi(i)}$ is the cipher texts of $v_{\pi(i)}$; (2) the permutation π is unknown to both parties unless they collaborate; (3) the plain text $v_1, \cdots, v_n$ is never revealed to any party during the protocol.

B. Privacy-Preserving SAT Solver: Details

This subsection first describes the basic idea behind the privacy-preserving SAT solver, followed by the technical details of its key components.

Limitations of garbled circuits. A strawman solution to Problem 1 is to construct a garbled circuit of the entire DPLL algorithm, because it can be expressed as a Boolean circuit. However, the resulting circuit can blow up, especially for the algorithms such as DPLL of which the circuit size is exponential to the size of inputs. Such an explosion of complexity can be overcome by revealing a part of the information, exposing the history of backtracking-search, or leaving the access pattern of private inputs of $\mathbf{A}$ and $\mathbf{B}$ outside the garbled circuit. However, the privacy leak of this solution can be substantial and lead to the complete exposure of $\mathbf{F^A}$ and $\mathbf{F^B}$.

Basic idea: The proposed SAT solver addresses this tradeoff between the high overhead of accessing private inputs in the garbled circuit and the severe privacy leakage of accessing private inputs outside the garbled circuit with a simple, yet elegant idea: instead of hiding the search history with high overhead, expose a disguised search history that no party can recover independently.

Specifically, the solver consists of two key steps: *permute* and *garble*. First, two parties construct and permute a matrix corresponding to $\mathbf{F^A} \wedge \mathbf{F^B}$ through the oblivious shuffling algorithm (Algorithm 2), after which both permutation and entries of the matrix are shared. Then, two parties use garbled circuit to implement the computation operations in DPLL, while all indexes being accessed are revealed and become public (Algorithm 3). As such, the high overhead of accessing private inputs in the garbled circuit is avoided. In addition, only *the structure of the backtracking-search steps* is exposed as the permutation is unknown to any single party. More precisely, the history of the search is exposed, but it is a history of permuted Boolean literals and thus no party knows which literal of original formula is being accessed. It is more difficult for $\mathbf{A}$ and $\mathbf{B}$ to infer $\mathbf{F^B}$ and $\mathbf{F^A}$ by observing this permuted search pattern. The steps of the privacy-preserving SAT solver are presented as follows:

Step 0: Initialization (Algorithm 1). The solver starts by zero-padding and aligning $\mathbf{F^A}$ and $\mathbf{F}^B$. This is because during the modeling phase, $\mathbf{A}$ and $\mathbf{B}$ each independently introduce auxiliary Boolean variables to transform their SAT formulas into CNF. As such, $\mathbf{A}$ and $\mathbf{B}$ each reveal to each other the number of auxiliary Boolean variables they introduced. Then $\mathbf{F^A}$ and $\mathbf{F^B}$ are both "stretched" to the same number n of rows such that the row vectors $\mathbf{f}_i^{\mathbf{A}}$ and $\mathbf{f}_i^{\mathbf{B}}$ record the occurrence and polarity of the same Boolean variable x_i in $\mathbf{F^A}$ and $\mathbf{F^B}$, separately. Given an auxiliary Boolean variable introduced by $\mathbf{A}$, its corresponding row vector in $\mathbf{F^B}$ is a zero vector. We denote $m^{\mathbf{A}}$ and $m^{\mathbf{B}}$ the number of clause of $F^{\mathbf{A}}$ and $F^{\mathbf{B}}$ respectively, assume $\mathbf{F^A}$ is of n-by-$m^{\mathbf{A}}$, $\mathbf{F^B}$ is of n-by-$m^{\mathbf{B}}$ and let $m = m^{\mathbf{A}} + m^{\mathbf{B}}$.

After $\mathbf{F^A}$ and $\mathbf{F}^B$ are aligned, the solver uses a simple procedure as shown in Algorithm 1 to store an encrypted version of the matrix $\{\mathbf{F^A} \ \ \mathbf{F^B}\}$ privately at $\mathbf{A}$ where that is the concatenation of matrix $\mathbf{F^A}$ and matrix $\mathbf{F^B}$. At the same time, it stores the encryption key k_B used privately at $\mathbf{B}$. In this work, we explicitly for encryption use one-time pad with pseudorandom function (**PRF**) that takes the a key and inputs and outputs an almost random string of the same size as the input. We use element-wise application of **PRF** when input is a matrix. That is $(\mathbf{PRF}(M))_{ij} = \mathbf{PRF}(M_{ij})$. We refer interested readers to [31] for more details about **PRF** and the proof security of this encryption scheme.

Algorithm 1: Preparing for oblivious shuffling.

```
Input: $A$ and $B$ each holds its private formula $\mathbf{F^A}$ and $\mathbf{F^B}$
Output: $\mathbf{A}$ holds an encrypted version of $\{\mathbf{F^A} \ \ \mathbf{F^B}\}$, and $\mathbf{B}$ holds
        the key $k_{\mathbf{B}}$
1 $\mathbf{A}$ locally a generates key $k_{\mathbf{A}}$ and an $n$-by-$m^{\mathbf{A}}$ random matrix $\mathbf{R^A}$;
2 $\mathbf{A}$ sends an $n$-by-$m^{\mathbf{A}}$ matrix $\mathbf{\Psi} = \mathbf{F^A} \oplus \mathbf{PRF}(k_{\mathbf{A}}, \mathbf{R^A})$, to $\mathbf{B}$;
3 $\mathbf{B}$ locally a generates key $k_{\mathbf{B}}$, an $n$-by-$m^{\mathbf{A}}$ random matrix $\mathbf{R}^{\mathbf{B}}_{\mathbf{A}}$,
    and an $n$-by-$m^{\mathbf{B}}$ random matrix $\mathbf{R}^{\mathbf{B}}_{\mathbf{B}}$;
4 $\mathbf{B}$ sends $\mathbf{\Phi^A} = \mathbf{\Psi} \oplus \mathbf{PRF}(k_{\mathbf{B}}, \mathbf{R}^{\mathbf{B}}_{\mathbf{A}})$,
    $\mathbf{\Phi^B} = \mathbf{F^B} \oplus \mathbf{PRF}(k_B, R^{\mathbf{B}}_{\mathbf{B}})$, $\mathbf{R}^{\mathbf{B}}_{\mathbf{A}}$ and $\mathbf{R}^{\mathbf{B}}_{\mathbf{B}}$ then sends it back to
    $\mathbf{A}$;
5 $\mathbf{A}$ computes $\mathbf{\Phi^A} \oplus \mathbf{PRF}(k_{\mathbf{A}}, \mathbf{R^A}) = \mathbf{F^A} \oplus \mathbf{PRF}(k_{\mathbf{B}}, \mathbf{R}^B_A)$;
6 $\mathbf{A}$ stores $\mathbf{R^B} = \{\mathbf{R}^B_A \ \ \mathbf{R}^B_B\}$ and
    $\mathbf{F}_{k_{\mathbf{B}}, \mathbf{R^B}} = \{\mathbf{F^A} \oplus \mathbf{PRF}(k_{\mathbf{B}}, \mathbf{R}^B_A) \ \ \mathbf{F^B} \oplus \mathbf{PRF}(k_{\mathbf{B}}, \mathbf{R}^B_B)\} =$
    $\{\mathbf{F^A} \ \ \mathbf{F^B}\} \oplus \mathbf{PRF}(k_{\mathbf{B}}, \mathbf{R^B})$, and $\mathbf{B}$ stores $k_B$;
```

Step 1: Oblivious shuffling of private inputs (Algorithm 2). Given a matrix $\mathbf{V}$, the oblivious shuffling algorithm in the proposed solver (Algorithm 2) decides a row permutation π on $\mathbf{V}$, and distributes the secret $(\pi, \pi(\mathbf{V})$ between $\mathbf{A}$ and $\mathbf{B}$. Specifically, it takes $\mathbf{R}$ and $\mathbf{V} \oplus \mathbf{PRF}(k_B, \mathbf{R})$, the information stored at A at the end of the preparation (Line 6 of Algorithm 1), where $\mathbf{V} = \mathbf{F} = \{\mathbf{F^A} \ \ \mathbf{F^B}\}$ and $\mathbf{R} = \mathbf{R}^B$; and $k_{\mathbf{B}}$, the information stored at B (Line 7 of Algorithm 1), as the input. It then has $\mathbf{A}$ and $\mathbf{B}$ each introduce private random matrices (*i.e.*, $\mathbf{R^A}$ by $\mathbf{A}$ and $\mathbf{R}^1$, $\mathbf{R}^2$ by $\mathbf{B}$) and private permutations (*i.e.*, $\pi_{\mathbf{A}}$ by $\mathbf{A}$ and $\pi_{\mathbf{B}}$ by $\mathbf{B}$), and achieves the goal of oblivious shuffling by local permutations and secure evaluation of **PRF** via garbled circuits.

The key ingredient of this shuffling is the math behind Line 8, where it can be derived that $\mathbf{\Theta} = \pi_{\mathbf{A}}(\mathbf{V} \oplus \mathbf{\Gamma} \oplus \mathbf{\Omega})$. With this result, in Line 11-12, $\mathbf{A}$ can store a piece of secret $s_{\mathbf{A}} = \pi_{\mathbf{B}}\pi_{\mathbf{A}}(\mathbf{V}) \oplus \pi_{\mathbf{B}}(\mathbf{PRF}(k_{\mathbf{B}}, \mathbf{R}^1))$, and $\mathbf{B}$ can store another piece $s_{\mathbf{B}} = \pi_{\mathbf{B}}(\mathbf{PRF}(k_{\mathbf{B}}, \mathbf{R}^1))$. As such, the secret can only be recovered as $\pi = s_{\mathbf{A}} \oplus s_{\mathbf{B}} = \pi_{\mathbf{B}}\pi_{\mathbf{A}}$, and $\pi(\mathbf{V}) = \pi_{\mathbf{B}}\pi_{\mathbf{A}}(\mathbf{V}$ when both pieces are combined together. During this shuffling, no input from A (*i.e.*, $\mathbf{R}$ and $\mathbf{V} \oplus \mathbf{PRF}(k_B, \mathbf{R})$) or B ($k_{\mathbf{B}}$) is exposed to the other party.

```
Algorithm 2: Oblivious shuffling over private inputs.
Input: $\mathbf{A}$ holds $\mathbf{R}$ and $\mathbf{V} \oplus \mathbf{PRF}(k_B, \mathbf{R})$; $\mathbf{B}$ holds $k_\mathbf{B}$;
Output: $\mathbf{A}$ holds $\pi_\mathbf{B}\pi_\mathbf{A}(\mathbf{V}) \oplus \pi_\mathbf{B}(\mathbf{PRF}(k_\mathbf{B}, \mathbf{R}^1))$;
B holds $\pi_\mathbf{B}(\mathbf{PRF}(k_\mathbf{B}, \mathbf{R}^1))$;
1  $\mathbf{A}$ generates a key $k_\mathbf{A}$, a random matrix $\mathbf{R}^\mathbf{A}$, and a permutation
     $\pi_\mathbf{A}$;
2  $\mathbf{A}$ computes $\mathbf{\Sigma} = \pi_\mathbf{A}(\mathbf{V} \oplus \mathbf{PRF}(k_B, \mathbf{R}) \oplus \mathbf{PRF}(k_\mathbf{A}, \mathbf{R}^\mathbf{A}))$,
     $\pi_\mathbf{A}(\mathbf{R}^\mathbf{A})$ and $\pi_\mathbf{A}(\mathbf{R})$;
3  $\mathbf{A}$ sends $\Sigma$ and $\pi_\mathbf{A}(\mathbf{R}^\mathbf{A})$ to $B$;
4  $\mathbf{B}$ generates two random matrices $\mathbf{R}^1$ and $\mathbf{R}^2$;
5  $\mathbf{A}$ and $\mathbf{B}$ construct two garbled circuits to collectively compute
     $\mathbf{\Delta} = \mathbf{PRF}(k_\mathbf{B}, \pi_A(\mathbf{R}))$ and $\mathbf{\Gamma} = \mathbf{PRF}(k_\mathbf{A}, \mathbf{R}^2)$, respectively,
     without revealing the input to each other; and the circuits do not
     release the results to any party;
6  $\mathbf{A}$ computes $\mathbf{\Lambda} = \mathbf{PRF}(k_\mathbf{A}, \pi_A(\mathbf{R}^\mathbf{A}))$;
7  $\mathbf{B}$ computes $\mathbf{\Omega} = \mathbf{PRF}(k_\mathbf{B}, \mathbf{R}^1)$;
8  $\mathbf{A}$ and $\mathbf{B}$ construct a garbled circuit taking the output of the previous
     circuits to collectively compute $\mathbf{\Theta} = \Sigma \oplus \mathbf{\Delta} \oplus \mathbf{\Gamma} \oplus \mathbf{\Lambda} \oplus \mathbf{\Omega}$, which
     equals to $\pi_\mathbf{A}(\mathbf{V}) \oplus \mathbf{\Gamma} \oplus \mathbf{\Omega}$, and the circuit only release $\mathbf{\Theta}$ to $\mathbf{B}$;
9  $\mathbf{B}$ generates a permutation $\pi_\mathbf{B}$, computes
     $\pi_\mathbf{B}(\mathbf{\Theta}) = \pi_\mathbf{B}\pi_\mathbf{A}(\mathbf{V}) \oplus \pi_\mathbf{B}(\mathbf{\Gamma}) \oplus \pi_\mathbf{B}(\mathbf{\Omega}))$, $\pi_\mathbf{B}(\mathbf{R}^1)$, and
     $\pi_\mathbf{B}(\mathbf{R}^2)$, and sends them to $\mathbf{A}$;
10 $\mathbf{A}$ computes $\mathbf{PRF}(k_\mathbf{A}, \pi_\mathbf{B}(\mathbf{R}^2)) = \pi_\mathbf{B}(\Gamma)$;
11 $\mathbf{A}$ stores
     $s_\mathbf{A}(\mathbf{V}) = \pi_\mathbf{B}(\mathbf{\Theta}) \oplus \pi_\mathbf{B}(\mathbf{\Omega}) = \pi_\mathbf{B}\pi_\mathbf{A}(\mathbf{V}) \oplus \pi_\mathbf{B}(\mathbf{PRF}(k_\mathbf{B}, \mathbf{R}^1))$;
12 $\mathbf{B}$ stores $s_\mathbf{B}(\mathbf{V}) = \pi_\mathbf{B}(\mathbf{PRF}(k_\mathbf{B}, \mathbf{R}^1))$;
```

In addition to permuting the SAT formula matrix $\mathbf{F} = \{\mathbf{F}^\mathbf{A}\ \mathbf{F}^\mathbf{B}\}$ by row with π and distributing this secret between $\mathbf{A}$ and $\mathbf{B}$, The proposed SAT solver also uses Algorithms 1 and 2 to permute two row vectors $\mathbf{prior}$ and $\mathbf{assign}$ with π and distribute each resulting secret between $\mathbf{A}$ and $\mathbf{B}$. These two vectors represent the branching search strategy specified by the agreement provider $\mathbf{B}$. Specifically, $\mathbf{prior}$ represents the priorities of all Boolean variables in the verification formula $\mathbf{F}$. During the DPLL search, when there is no more unit literal in the formula, the variable that is not decided and has the highest priority will be selected as the next variable x_i and first be assigned a truth value of $assign_i$ to simplify the formula. How $\mathbf{prior}$ and $\mathbf{assign}$ are computed at $\mathbf{B}$ is out of the scope of the proposed solver. [1]

Step 2: A DPLL garbled circuit with exposed, permuted search history (Algorithm 3). After using an oblivious shuffling algorithm to permute the verification formula $\mathbf{F}$ and the branching search strategy $(\mathbf{prior}, \mathbf{assign})$, the proposed SAT solver designs a garbled circuit that encodes the DPLL algorithm with an exposed, permuted search history (Algorithm 3). Specifically, Algorithm 3 uses the same backtrack-searching process and the unit literal search optimization as in the original DPLL.[2]

Compared with the DPLL algorithm, the key differences in Algorithm 3 are the use of oblivious subroutines, *i.e.*, resolution (Line 8) 4, contradiction detection and checking (Line 9) 5, unit literal search (Line 25) 6, and branching (Line 31) 7, which are all encoded by garbled circuits. Specifically, the operator $[\cdot]$ is used in Algorithm 3 to indicate that the bracketed expression is computed using a garbled circuit and the result is not released to either $\mathbf{A}$ or $\mathbf{B}$ unless explicitly specified using the *release* keyword.

Specifically, the algorithm takes $\pi(\mathbf{F})$, $\pi(\mathbf{prior})$ and $\pi(\mathbf{assign})$, the permuted verficiation formula and search strategy from the oblivious shuffling step, as input. These

[1]We note that the proposed solver may be more efficient if the search strategy is decided by $\mathbf{A}$ and $\mathbf{B}$ collectively, and this is left for future work.

[2]The proposed solver does not use pure literal elimination because it is rarely used by modern DPLL-based SAT solvers.

```
Algorithm 3: Garbled-Circuit DPLL
Input: Secrets $\pi(\mathbf{F})$, $\pi(\mathbf{prior})$ and $\pi(\mathbf{assign})$ that are distributed
       stored at $\mathbf{A}$ and $\mathbf{B}$;
1  $\mathbf{u} = \{\text{false}, \cdots, \text{false}\}$ ;
2  $\mathbf{d} = \{\text{false}, \cdots, \text{false}\}$;
3  $\mathbf{c} = \{\text{false}, \cdots, \text{false}\}$;
   // $\mathbf{u}, \mathbf{d}, \in \{\text{true}, \text{false}\}^n$, $\mathbf{c} \in \{\text{true}, \text{false}\}^m$
4  $T$ is an empty stack;
   // $T$ is the stack that is initialized to be
      empty
5  $i = 0$ ;
6  while true do
7      if $i \neq 0$ then
8          OblivRes($i, \pi(\mathbf{F}), \pi(\mathbf{assign}), \mathbf{c}$);
9          $([b_s], [b_c]) =$ OblivCC($\pi(\mathbf{F}),$ );
10         $[b_c]$.release($\mathbf{A}, \mathbf{B}$);
11         $[b_s]$.release($\mathbf{A}, \mathbf{B}$);
12         if $b_c$ then
13             while $\neg T.empty$ do
14                 $(\pi(F), i, c, d, \pi(\mathbf{assign}), state) = T$.pop();
15                 if $state == FIRST$ then
16                     break
17             if $T.empty$ then
18                 return false
19             $[\pi(\mathbf{assign})_i] = [\neg\pi(\mathbf{assign})_i]$;
20             $T$.push($\pi(\mathbf{F}), i, \mathbf{c}, \mathbf{d}, \pi(\mathbf{assign})$, SECOND);
21             continue
22         else
23             if $b_s$ then
24                 return true
25     [ind] = OblivULS($\pi(\mathbf{F}), \pi(\mathbf{prior}), \mathbf{u}, \pi(\mathbf{assign})$);
26     [ind].release($\mathbf{A}, \mathbf{B}$);
27     if $ind \neq 0$ then
28         $i =$ ind ;
29         $d_i =$ true
30     else
31         $[i] =$ OblivBranch($\pi(\mathbf{prior}), \mathbf{d}$);
32         $[i]$.release($\mathbf{A}, \mathbf{B}$);
33         $d_i =$ true;
34         $T$.push($\pi(\mathbf{F}), i, \mathbf{c}, \mathbf{d}, \pi(\mathbf{assign})$, FIRST);
35     $i = 1$ ;
```

```
Algorithm 4: OblivRes: oblivious resolution.
Input: $i_0, [\pi(\mathbf{F})], [\pi(\mathbf{assign})], [\mathbf{c}]$
Output: None
1  for $j$ = 1 to m do
2      $[b] = [\pi(\mathbf{assign})_{i_0}]$;
3      $[\text{cond}_0] = [\neg\pi(\mathbf{F})_{i_0,j}.O]$;
4      $[\text{cond}_1] = [(b \neq \pi(\mathbf{F})_{i_0,j}.P) \cdot \pi(\mathbf{F})_{i_0,j}.O]$;
5      $[\text{cond}_2] = [(b == \pi(\mathbf{F})_{i_0,j}.P) \cdot \pi(\mathbf{F})_{i_0,j}.O]$;
6      for $i$ = 1 to n do
7          $[\text{cond}_3] = [i == i_0]$;
8          $[\pi(\mathbf{F})_{i,j}.O] = [\text{cond}_0 \cdot \pi(\mathbf{F})_{i,j}.O + \text{cond}_2.0 + \text{cond}_1 \cdot$
             $\text{cond}_3 \cdot 0 + \text{cond}_1 \cdot (1 - \text{cond}_3) \cdot \pi(\mathbf{F})_{i,j}.O]$;
9      end
10     $\mathbf{c}_j = \text{cond}_2$
11 end
```

Algorithm 5: OblivCC: oblivious contradiction checking.

```
Input: $[\pi(\mathbf{F})]$, $[\mathbf{c}]$
Output: $[b_s]$, $[b_c]$
$[b]$ = false;
$[s] = 0$;
for $j$ = 1 to $m$ do
    $[z] = 0$ ;
    for $i$ = 1 to $n$ do
        $[z] = [z + \pi(\mathbf{F})_{i,j}.O \cdot 1]$;
    end
    $[b] = [b \vee (z == 0 \wedge \neg \mathbf{c}_j)]$ ;
    $[s] = [s + z]$
end
return $([s == 0], [b])$
```

inputs are split into two secrets and distributed over $\mathbf{A}$ and $\mathbf{B}$, As such, for the simplicity of presentation without causing confusion, when these inputs show in the psuedocode, they represent the operations of combining the secrets held at $\mathbf{A}$ and $\mathbf{B}$. For example, if $\pi(\mathbf{F})$ is distributed as $s_A(\mathbf{F})$ to $\mathbf{A}$ and $s_B(\mathbf{F})$ to $\mathbf{B}$, where $s_A(\mathbf{F}) \oplus s_B(\mathbf{F})$, then $[\pi(\mathbf{F})]$ represents $[s_A(\mathbf{F}) \oplus s_B(\mathbf{F})]$.

Moreover, two parties also initialize other three vectors: secret shared **u** that represents whether the literals are unit or not; public **d** that records if the values of the literals have been decided, and **c** that records if the clauses have been removed (satisfied). Last but not least, a stack T is also instantiated for backtracking later.

C. Privacy-Preserving SAT Solver: Analysis

We conduct rigorous analysis on the correctness, privacy and efficiency on the proposed privacy-preserving SAT solver. The security and correctness of this solver rely on the security and correctness of the oblivious shuffling phase (*i.e.*, Algorithm 2) and the way it performs permutation. We hereby state them first here:

Theorem 1 (**Correctness and Privacy-Preserving of Oblivious Shuffling**)**.**Given a pseudorandom function (**PRF**) and input $(\mathbf{V} \oplus \mathbf{PRF}(k_{\mathbf{B}}, R), R)$, Algorithm 2:

- outputs $s_{\mathbf{A}}(\mathbf{V})$ and $s_{\mathbf{B}}(\mathbf{V})$ satisfying $s_{\mathbf{A}}(\mathbf{V}) \oplus s_{\mathbf{B}}(\mathbf{V}) = \pi_{\mathbf{B}}\pi_{\mathbf{A}}(\mathbf{V})$
- where $\pi_{\mathbf{A}}$ (respectively $\pi_{\mathbf{B}}$) is unknown to $\mathbf{B}$ (respectively $\mathbf{A}$);
- keeps the v_i unrevealed. In particular, given two vectors $\mathbf{V}$ and $\mathbf{V}'$, there is no probabilistic polynomial time $\mathbf{A}$ or $\mathbf{B}$ that is able to individually distinguish the algorithm is performed with $(\mathbf{V} \oplus \mathbf{PRF}(k_{\mathbf{B}}, R), R)$ or $(\mathbf{V}' \oplus \mathbf{PRF}(k_{\mathbf{B}}, R), R)$ as an input.

Proof. (Sketch) For the first point on correctness, it can be shown as

$$s_{\mathbf{A}}(\mathbf{V}) \oplus s_{\mathbf{B}}(\mathbf{V}) = \pi_{\mathbf{B}}(\mathbf{PRF}(k_{\mathbf{B}}, \mathbf{R}^1)) \oplus \pi_{\mathbf{B}}\pi_{\mathbf{A}}(\mathbf{V}) = \pi_{\mathbf{B}}\pi_{\mathbf{A}}(\mathbf{V})$$

For the last two points on privacy, the basic idea is that the security of this oblivious shuffling relies on security of one-time pad encryption and garbled circuits that is proved in [31]. The central idea of our proof is reduction from the attacks against the privacy of this oblivious shuffling scheme to the attacks against either garbled circuits or the one-time pad encryption. □

Algorithm 6: OblivULS: oblivious unit literal search.

```
Input: $\pi(\mathbf{F})$, $[\pi(\mathbf{prior})]$, $[\pi(\mathbf{assign})]$, $[\mathbf{u}]$
Output: [ind] such that ind $\in \{1, \cdots, n\}$
for $j = 1$ to $m$ do
    $[b_j] = 0$;
    for $i = 1$ to $n$ do
        $[b_j] = [\ b_j + \pi(\mathbf{F})_{i,j}.O]$
    end
    for $i = 1$ to $n$ do
        [cond] $= [(b_j == 1) \wedge \pi(\mathbf{F})_{i,j}.O == 1)]$;
        $[\mathbf{u}_i]$ = [cond $\cdot 1 + \neg$cond $\cdot \mathbf{u}_i]$;
        $[\pi(\mathbf{assign})_i] = [\mathbf{u}_i \cdot \pi(\mathbf{F})_{i,j}.P + (1 - \mathbf{u}_i) \cdot \pi(\mathbf{assign})_i]$;
    end
end
[ind] = [0];
[pri] = [0];
for $i$ = 1 to $n$ do
    [cond] = $[(\mathbf{u}_i == 1) \cdot (\pi(\mathbf{prior})_i >$ pri)];
    [ind] = [cond $\cdot i +$ ind $\cdot (1 -$ cond)] ;
    [pri] = [cond $\cdot \pi(\mathbf{prior})_i +$ pri $\cdot (1 -$ cond)];
end
return [ind]
```

Correctness: The correctness of our SAT solver is a natural result of the basic idea behind our algorithm. Notice that permuting the literals will not change the satisfiablity of a formula, to prove the correctness of our protocol, it is sufficient to show that algorithm 3 indeed implements DPLL.

Theorem 2 (**Correctness**)**.**Given any SAT propositional formula F in CNF as input, it is satisfiable if and only if algorithm 3 returns $true$.

Proof. (Sketch) The basic idea of the proof is to show that Algorithm 3 implements DPLL. Essentially, we conduct as many **OblivULS** (Algorithm 6 as possible until no unit literal is available (corresponding to searching unit literal and removing clauses). Then, **OblivBranch** (Algorithm 7) is called and current state is pushed into stack, after which **OblivRes** (Algorithm 4)and **OblivCC** (Algorithm 5) are called to simplify $\pi(\mathbf{F})$ and test validity of our guess (corresponding to the branching). Whenever a failure (b_c = false) arises, we backtrack the searching tree until a *first* guess is found by popping out the stack (corresponding backtracking). On the other hand, if a success is met, *i.e.* b_s = true, the algorithm terminates and outputs true. Since each variable has at most 2 possible assignments, the algorithm either eventually returns true at some point, or it exhausts all possible assignments and reaches the bottom of stack T then returns false. With this sketch, a formal proof can be easily derived. □

Security. We give the following theorem on the information leakage of our solver.

Theorem 3 (**Information Leakage**)**.**The algorithm 3 realizes secure MPC for deciding the satisfiability of $F^{\mathbf{A}} \wedge F^{\mathbf{B}}$ with a leakage profile of: (1) the number of common variables of $F^{\mathbf{A}}$ and $F^{\mathbf{B}}$, (2) the number of variables of $F^{\mathbf{A}}$ and $F^{\mathbf{B}}$, (3) the size of the input formulas, and (4) the searching pattern.

Algorithm 7: OblivBranch: oblivious branching.

```
Input: $[\pi(\mathbf{prior})], \mathbf{d}$
Output: $[\text{ind}']$ such that ind $\in \{1, \cdots, n\}$
1 $[\text{ind}'] = [0]$;
2 $[\text{pri}] = [0]$;
3 for $i$ = 1 to $n$ do
4     $[\text{cond}] = [\neg\mathbf{d}_i \cdot (\pi(\mathbf{prior})_i > \text{pri})]$ ;
5     $[\text{ind'}] = [\text{cond} \cdot i + \text{ind}' \cdot (1 - \text{cond})]$;
6     $[\text{pri}] = [\text{cond} \cdot \pi(\mathbf{prior})_i + \text{ind}' \cdot (1 - \text{cond})]$ ;
7 end
8 return ind$'$ ;
```

Essentially, we allow $\mathbf{A}$ and $\mathbf{B}$ to learn searching pattern, *i.e.*, the structure of the decision tree in DPLL, and the searching time. Given a formula, the backtrack search algorithm organizes the search for a satisfying assignment by maintaining a decision tree. By searching pattern, we mean the structure of the traversal tree during the execution of DPLL. The searching pattern essentially only shows when branching and backtracking occurs but does not reveals what the branching literal is. We stress that we can still show security of the resulting system by formally proving the leakage is nothing more than this. We accept this certain minimal amount of leakage is unavoidable for the sake of efficiency. The formal proof of the theorem is based on simulation paradigm of cryptography [32] and omitted due to space limit.

We conclude our analysis with the following theorem on the efficiency of the proposed solver.

Theorem 4 (**Efficiency**).Given a formula f, denote T the number of steps in DPLL to determine if f is satisfiable with the heuristics that can be expressed as two vectors $\mathbf{assign}$ and $\mathbf{prior}$, about which we discuss in IV-B, then the computational complexity as well as the communication complexity of GC-DPLL is $O(T \cdot mn)$.

Proof. The proof can be directly derived from two facts (a) the overhead from garbled circuits is constant [19] so the computational complexity and communication complexity of each subroutine are both bounded by $O(mn)$. (b) the computation and communication for oblivious shuffling is $O(mn)$. The other parts of our protocol are exactly the same as DPLL, therefore the total computation and communication will be bounded by $O(T \cdot mn)$. □

V. Evaluation

This section presents a prototype implementation of IVeri, and evaluate its performance via extensive experiments.

A. Prototype Implementation

To model the interdomain peering agreements verification problem, the IVeri prototype uses Batfish [33] and Minesweeper [9] to parse vendor-specific BGP configurations to an SMT formula, and then transforms the SMT formula to the configuration SAT formula $\mathbf{F^B}$ using bit blasting [26] and Tseytin transformation [34]. The prototype also uses a similar transformation procedure to transform the SMT formula of peering agreements to the agreement SAT formula $\mathbf{F^A}$.

The privacy-preserving SAT solver of this prototype is implemented using OpenSSL [35] and Obliv-C [36], a GCC wrapper to embed secure computation protocols inside regular C programs. The prototype uses AES-128 as the choice of PRF (pseudorandom function). One implementation optimization in the prototype is that, during the oblivious shuffling phase, instead of using component-wise operations on each element in the matrix, we divide elements in the same row of matrix into blocks of 128 bits, and use block-wise operations. Because each element in a SAT formula matrix has only 2 bits, this optimization improves the efficiency (in terms of computation, storage and communication) of oblivious shuffling by approximately 128/2=64 times. Overall, the IVeri prototype has ∼2700 lines of C and Obliv-C code.

B. Experiment Methodology

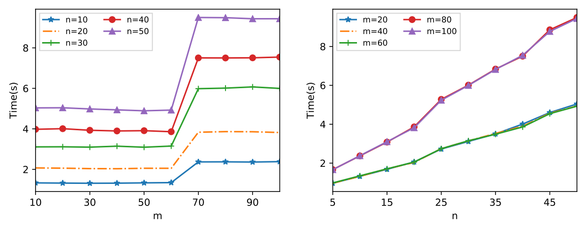
The goal is to examine the efficiency of IVeri on interdomain peering agreement verification, and evaluate its overhead. To this end, two sets of experiments are designed. The first set is the functionality check experiment. Specifically, we setup the example abstract topology in Figure 1 as the evaluation scenario, where the two BGP routers of the agreement provider B are configured using the Cisco IOS configuration language, and the agreement consumer specifies five representative peering agreements [15], [21] (including the three introduced in Section II). The second set of experiment uses a standard SAT datasets (*i.e.*, the SATLIB datasets [37]) to evaluate the overhead of the privacy-preserving SAT solver more extensively. In both sets of experiments, the agreement consumer and provider each runs its IVeri process on an Ubuntu server with Intel(R) Xeon(R) Platinum 8168 CPU and 376GB RAM.

C. Results

Functionality check: In the first experiment, five representative peering agreements are specified. For each agreement, we create different interdomain agreement verification problems by using three different sets of correct BGP configurations of B. As such, a total of 15 experiments are executed. In all 15 experiments, IVeri successfully determines that the problem of $\mathbf{F^A} \wedge \mathbf{F^B}$ is unsatisfiable, indicating that the agreement-to-verify is implemented correctly at B. For each agreement, we then deliberately change one correct BGP configuration to be incorrect, and executes IVeri again. Results show that in all these 5 experiments, IVeri successfully finds a satisfiable solution to the problem of $\mathbf{F^A} \wedge \mathbf{F^B}$, indicating that these agreements are not correctly implemented. With this functionality check result, we conclude that IVeri provides accurate results for interdomain peering agreement verification.

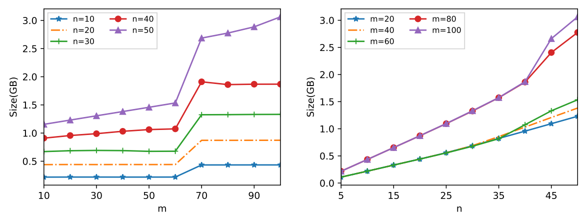
Overhead study via SAT benchmark simulation: In the second experiment, we select an open SAT dataset [37] whose input SAT formulas have 50 variables and 100 clauses. In the simulation, we vary the number of variables (n) to be from 5 to 50, with a step size of 5, and vary the number of clauses (m) to be from 10 to 100, with a step size of 10. For each combination, 30 SAT formulas are generated from the dataset. In each experiment, each formula is approximately separated in half and held by A and B, respectively.

The first metric we study is *oblivious shuffling delay*, *i.e.*, the time consumption of the oblivious shuffling phase of the proposed solver. Figure 3a shows that (1) the worst case of oblivious shuffling delay is less than 10 seconds; and (2)



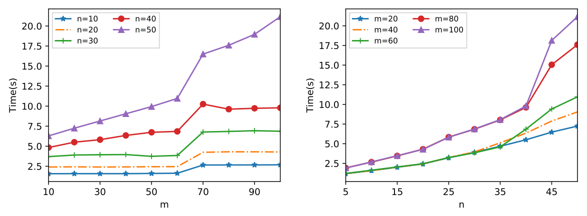
(a) Varying number of clauses m. (b) Varying number of variables n.

Fig. 3: Oblivious shuffling delay of IVeri.



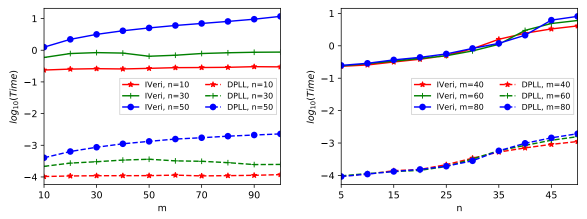
(a) Varying number of clauses m. (b) Varying number of variables n.

Fig. 5: Data transmission overhead of IVeri: total message size.



(a) Varying number of clauses m. (b) Varying number of variables n.

Fig. 4: Verification delay of IVeri.



(a) Varying number of clauses m (b) Varying number of variables n.

Fig. 6: Computation delay: GC-DPLL vs. DPLL.

the delay increases sharply when m reaches 64. This sharp increase is the result of the implementation optimization, where the oblivious shuffling operations in the same row are block-wise with 128/2=64 elements, not component-wise. Figure 3b shows that the oblivious shuffling delay increases linearly as the number of variables increases. The separation of two lines in this figure is again the result of our block-wise shuffling operation.

Second, we study the *verification delay*, which is the sum of the oblivious shuffling delay and the delay of the GC-DPLL algorithm. Figure 4 shows that the oblivious shuffling delay takes about 50% of the verification delay, and that the impact of m on the verification delay increases as n increases.

Third, we study the size of transmitted data during the verification. Figure 5 shows that the size of total transmitted data is up to 3 GB. Although this seems like large, in modern networks where the interdomain bandwidth is often of hundreds of GBs, this data transmission overhead is negligible.

Fourth, we compare the performance of the GC-DPLL algorithm with the original DPLL algorithm. Figure 6 compares their time consumption. We observe that the GC-DPLL algorithm increases the verification delay of the DPLL algorithm by three orders of magnitude. Considering that the absolute verification delay of GC-DPLL is of tens of seconds, this increase is acceptable. With these overhead study, we conclude that IVeri is an efficient solution for privacy-preserving interdomain agreement verification.

VI. Related Work

Network verification. Network verification tools are roughly categorized in two classes. First, network configuration verification tools (*e.g.*, [9]–[11]) focus on finding errors in the configurations of routing protocols (*i.e.*, the control plane of network) within the same network by building and analyzing a symbolic network model (*e.g.*, an SMT model). IVeri heavily leverages this approach to model BGP configurations, but goes beyond to build new models for interdomain peering agreements. Second, data plane verification tools (*e.g.*, [38]–[40]) focus on verifying that the data plane of a network (*e.g.*, the forwarding tables and access control lists) satisfies certain properties (*e.g.*, loop-freeness). Although tools such as Looking Glass [41] and RouteView [42] expose certain BGP data plane configurations (*e.g.*, selected routes) of participating ASes, such incomplete information makes it non-trivial to extend data plane verification to interdomain networks.

Verification in interdomain networks. There are some efforts in interdomain network verification [12], [13], [15]–[17], [43], [44]. FSR [12], Bagpipe [13] and biNode [45] focus on verifying the basic property (*i.e.*, stability and reachability) of BGP in interdomain networks and require the complete exposure of the BGP configurations of all ASes. SIDR [43] focuses on verifying the safety property of data plane configurations of multiple SDXes, and sacrifices accuracy for privacy and safety. NetReview [16] and VPerf [15] focus on the errors in BGP configurations in BGP peers and are work most related to IVeri. However, they operate on a per route announcement basis, *i.e.*, it can only find errors after a route announcement is sent. To the best of our knowledge, IVeri is the first to verify the agreement implementation in BGP configurations proactively while preserving the privacy of ASes.

SAT solver. SAT solver is an active research area with a rich literature [26]–[28], [46], [47]. [47] studies how to disguise a SAT formula held by one party from the other party who holds an SAT solver However, to the best of our knowledge, we are the first SAT solver to decide the satisfiability of the conjunction of two private SAT formulas held by two parties.

SMPC in networks. SMPC-based routing systems have been proposed to let ASes collectively compute interdomain routes for better network performance [48], [49]. They are orthogonal to this paper, as IVeri focuses on verifying interdomain peering agreements instead of designing new routing systems.

VII. Conclusion

We investigate the important problem of interdomain peering agreement verification. We identify privacy as the fundamental challenge, and design IVeri, the first privacy-preserving interdomain agreement verification system, whose core is a novel privacy-serving SAT solver. Extensive evaluation on a IVeri prototype demonstrates its efficiency and efficacy.

References

[1] L. Gao and J. Rexford, "Stable internet routing without global coordination," *IEEE/ACM Transactions on Networking (TON)*, vol. 9, no. 6, pp. 681–692, 2001.

[2] "The Large Hadron Collider (LHC) Experiment," https://home.cern/topics/large-hadron-collider.

[3] Y. Rekhter, S. Hares, and D. T. Li, "A Border Gateway Protocol 4 (BGP-4)," RFC 4271, Jan. 2006. [Online]. Available: https://rfc-editor.org/rfc/rfc4271.txt

[4] "Time warner cable says outages largely resolved," http://www.seattletimes.com/business/time-warner-cable-says-outages-largely-resolved, 2014.

[5] "Bgpmon: Recent news and updates: leak causing internet outages in japan, brazil, india and etc." https://www.bgpmon.net/, 2019.

[6] "Bgp route leak causes cloudflare and amazon aws problems," https://www.bleepingcomputer.com/news/technology/bgp-route-leak-causes-cloudflare-and-amazon-aws-problems/, 2019.

[7] R. Mahajan, D. Wetherall, and T. Anderson, "Understanding bgp misconfiguration," in *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 4. ACM, 2002, pp. 3–16.

[8] "Google goes down due to bgp errors," https://arstechnica.com/information-technology/2018/11/major-bgp-mishap-takes-down-google-as-traffic-improperly-travels-to-china/, 2018.

[9] R. Beckett, A. Gupta, R. Mahajan, and D. Walker, "A general approach to network configuration verification," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 2017, pp. 155–168.

[10] ——, "Control plane compression," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. ACM, 2018, pp. 476–489.

[11] S. K. Fayaz, T. Sharma, A. Fogel, R. Mahajan, T. Millstein, V. Sekar, and G. Varghese, "Efficient network reachability analysis using a succinct control plane representation," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 217–232.

[12] A. Wang, L. Jia, W. Zhou, Y. Ren, B. T. Loo, J. Rexford, V. Nigam, A. Scedrov, and C. Talcott, "Fsr: Formal analysis and implementation toolkit for safe interdomain routing," *IEEE/ACM Transactions on Networking*, vol. 20, no. 6, pp. 1814–1827, Dec 2012.

[13] K. Weitz, D. Woos, E. Torlak, M. D. Ernst, A. Krishnamurthy, and Z. Tatlock, "Formal semantics and automated verification for the border gateway protocol."

[14] R. Anwar, H. Niaz, D. Choffnes, Í. Cunha, P. Gill, and E. Katz-Bassett, "Investigating interdomain routing policies in the wild," in *Proceedings of the 2015 Internet Measurement Conference*. ACM, 2015, pp. 71–77.

[15] M. Zhao, W. Zhou, A. J. Gurney, A. Haeberlen, M. Sherr, and B. T. Loo, "Private and verifiable interdomain routing decisions," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 383–394, 2012.

[16] A. Haeberlen, I. C. Avramopoulos, J. Rexford, and P. Druschel, "Netreview: Detecting when interdomain routing goes wrong." in *NSDI*, vol. 2009, 2009, pp. 437–452.

[17] H. Yang and S. S. Lam, "Collaborative verification of forward and reverse reachability in the internet data plane," in *2014 IEEE 22nd International Conference on Network Protocols*. IEEE, 2014, pp. 320–331.

[18] Y. Huang, D. Evans, and J. Katz, "Private set intersection: Are garbled circuits better than custom protocols?" in *NDSS*, 2012.

[19] A. C. Yao, "Protocols for secure computations," in *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, 1982.

[20] Q. Xiang, J. J. Zhang, X. T. Wang, Y. J. Liu, C. Guok, F. Le, J. MacAuley, H. Newman, and Y. R. Yang, "Toward fine-grained, privacy-preserving, efficient multi-domain network resource discovery," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 8, pp. 1924–1940, 2019.

[21] I. One Step Consulting, "BGP community guides," https://onestep.net/communities/, 2019.

[22] K. Jayaraman, N. Bjørner, J. Padhye, A. Agrawal, A. Bhargava, P.-A. C. Bissonnette, S. Foster, A. Helwer, M. Kasten, I. Lee *et al.*, "Validating datacenters at scale," in *Proceedings of the ACM Special Interest Group on Data Communication*. ACM, 2019, pp. 200–213.

[23] L. Gao, "On inferring autonomous system relationships in the internet," *IEEE/ACM Transactions on Networking (ToN)*, vol. 9, no. 6, pp. 733–745, 2001.

[24] M. Raykova, *Secure Computation in Heterogeneous Environments: How to Bring Multiparty Computation Closer to Practice?* Columbia University, 2012.

[25] V. Mishra, D. Verma, C. Williams, and K. Marcus, "Comparing software defined architectures for coalition operations," in *ICMCIS 2017*.

[26] S. Malik and L. Zhang, "Boolean satisfiability from theoretical hardness to practical success," *Communications of the ACM*, vol. 52, no. 8, pp. 76–82, 2009.

[27] C. P. Gomes, H. Kautz, A. Sabharwal, and B. Selman, "Satisfiability solvers," *Foundations of Artificial Intelligence*, vol. 3, pp. 89–134, 2008.

[28] N. Sorensson and N. Een, "Minisat v1. 13-a sat solver with conflict-clause minimization," *SAT*, vol. 2005, no. 53, pp. 1–2, 2005.

[29] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

[30] A. C.-C. Yao, "How to generate and exchange secrets," in *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*. IEEE, 1986, pp. 162–167.

[31] Y. Lindell and B. Pinkas, "A proof of security of yao&#x2019;s protocol for two-party computation," *J. Cryptol.*, vol. 22, no. 2, pp. 161–188, Apr. 2009. [Online]. Available: https://doi.org/10.1007/s00145-008-9036-8

[32] Y. Lindell, *How to Simulate It – A Tutorial on the Simulation Proof Technique*. Cham: Springer International Publishing, 2017, pp. 277–346.

[33] A. Fogel, S. Fung, L. Pedrosa, M. Walraed-Sullivan, R. Govindan, R. Mahajan, and T. Millstein, "A general approach to network configuration analysis," in *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, 2015, pp. 469–483.

[34] G. S. Tseitin, "On the complexity of derivation in propositional calculus," in *Automation of reasoning*. Springer, 1983, pp. 466–483.

[35] J. Viega, M. Messier, and P. Chandra, *Network security with openSSL: cryptography for secure communications.* " O'Reilly Media, Inc.", 2002.

[36] S. Zahur and D. Evans, "Obliv-c: A language for extensible data-oblivious computation." *IACR Cryptology ePrint Archive*, vol. 2015, p. 1153, 2015.

[37] "Satlib - benchmark problems," aug 2011. [Online]. Available: https://www.cs.ubc.ca/~hoos/SATLIB/benchm.html

[38] P. Kazemian, G. Varghese, and N. McKeown, "Header space analysis: Static checking for networks." in *NSDI*, vol. 12, 2012, pp. 113–126.

[39] H. Yang and S. S. Lam, "Real-time verification of network properties using atomic predicates," *IEEE/ACM Transactions on Networking*, vol. 24, no. 2, pp. 887–900, 2015.

[40] A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. B. Godfrey, "Veriflow: Verifying network-wide invariants in real time," in *Presented as part of the 10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13)*, 2013, pp. 15–27.

[41] A. Khan, T. Kwon, H.-c. Kim, and Y. Choi, "As-level topology collection through looking glass servers," in *Proceedings of the 2013 conference on Internet measurement conference*. ACM, 2013, pp. 235–242.

[42] O. RouteViews, "University of oregon routeviews project," *Eugene, OR.[Online]. Available: http://www. routeviews. org*.

[43] R. Birkner, A. Gupta, N. Feamster, and L. Vanbever, "Sdx-based flexibility or internet correctness?: Pick two!" in *Proceedings of the Symposium on SDN Research*. ACM, 2017, pp. 1–7.

[44] A. Shieh, E. G. Sirer, and F. B. Schneider, "Netquery: A knowledge plane for reasoning about network properties," in *ACM SIGCOMM*

Computer Communication Review, vol. 41, no. 4. ACM, 2011, pp. 278–289.

[45] X. Shao and L. Gao, “Verifying policy-based routing at internet scale,” in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020, pp. 2293–2302.

[46] M. Davis, G. Logemann, and D. W. Loveland, *A machine program for theorem-proving*. New York University, 1961.

[47] Y. Qin, S. Shen, and Y. Jia, “Structure-aware cnf obfuscation for privacy-preserving sat solving,” in *2014 Twelfth ACM/IEEE Conference on Formal Methods and Models for Codesign (MEMOCODE)*.

[48] Q. Chen, S. Shi, X. Li, C. Qian, and S. Zhong, “Sdn-based privacy preserving cross domain routing,” *IEEE Transactions on Dependable and Secure Computing*, 2018.

[49] G. Asharov, D. Demmler, M. Schapira, T. Schneider, G. Segev, S. Shenker, and M. Zohner, “Privacy-preserving interdomain routing at internet scale,” *Proceedings on Privacy Enhancing Technologies*, vol. 2017, no. 3, pp. 147–167, 2017.