Privacy-Preserving Quantification of Cross-Domain Network Reachability

Fei Chen, Bruhadeshwar Bezawada, and Alex X. Liu

Abstract—Network reachability is an important characteristic for understanding end-to-end network behavior and helps in detecting violations of security policies across the network. While quantifying network reachability within one administrative domain is a difficult problem in itself, performing the same computation across a network spanning multiple administrative domains presents a novel challenge. The problem of quantifying network reachability across multiple administrative domains is more difficult because the privacy of security policies of individual domains is a serious concern and needs to be protected through this process. In this paper, we propose the first cross-domain privacy-preserving protocol for quantifying network reachability. Our protocol constructs equivalent representations of the Access Control List (ACL) rules and determines network reachability while preserving the privacy of the individual ACLs. This protocol can accurately determine the network reachability along a network path through different administrative domains. We have implemented and evaluated our protocol on both real and synthetic ACLs. The experimental results show that the online processing time of an ACL containing thousands of rules is less than 25 s. Given two ACLs, each containing thousands of rules, the comparison time is less than 6 s, and the total communication cost is less than 2100 kB.

Index Terms—Cross domain, network reachability quantification, privacy preserving.

I. INTRODUCTION

A. Background and Motivation

NETWORK reachability for a given network path from the source subnet to the destination subnet is defined as the set of packets that are allowed by all network devices on the path. Network reachability quantification is important for understanding end-to-end network behavior and detecting the violation of security policies. Several critical concerns like router

misconfiguration, policy violations, and service availability can be verified through an accurate quantification. Quantifying network reachability is a difficult and challenging problem for two reasons. First, various complex mechanisms, such as Access Control Lists (ACLs), dynamic routing, and network address translation (NAT), have been deployed on network devices for restricting network reachability. To perform an accurate analysis, administrators need to collect all the reachability restriction information from these network devices. However, collecting such information is very difficult due to the privacy and security concerns. Second, the explosion of the Internet has caused an increase in the complexity and sophistication of these devices, thus making reachability analysis computationally expensive and error-prone.

The current practice of reachability management is still “trial and error” due to the lack of network reachability analysis and quantification tools. This approach leads to significant number of configuration errors and has been shown to be the major cause of failure for Internet services [2]. Industry research also shows that a significant percentage of human effort and monetary resources are employed in maintaining the operational status of the network. Reports place the revenue losses per hour for media, banking, and brokerage industry to \$1.2 million, \$2.6 million, and \$4.5 million, respectively [3]. Several critical business applications and sensitive communications are severely affected due to network outages caused by misconfiguration errors. These events place a tremendous amount of pressure on network operators to debug the problems quickly. Thus, systematic analysis and tools for quantification of network reachability are needed for understanding end-to-end network behavior and detecting configuration errors.

B. Limitation of Prior Art

The current practice of determining network reachability through probing has two major drawbacks. First, probing is expensive because it needs to generate and send a significant amount of probe packets. Second, probing is inaccurate, e.g., it cannot probe the open ports with no server listening on them. Due to these drawbacks, many approaches were proposed to address the reachability problem [4]–[9], [15]. The main assumption in all these approaches is that the reachability restriction information of each network device and configuration state are known to a central network analyst, who is quantifying the network reachability. However, in practice, it is common that the network devices along a given path belong to different domains where the reachability restriction information cannot be shared with others including the network analyst. Fig. 1 shows a typical scenario of network reachability, where User₁ wants to know what packets he can send to User₂

Manuscript received August 23, 2013; revised February 24, 2014; accepted March 04, 2014; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor S. Fahmy. Date of publication June 06, 2014; date of current version June 12, 2015. This work was supported in part by the National Science Foundation under Grant No. CNS-1017598. The preliminary version of this paper titled “Privacy-Preserving Cross-Domain Network Reachability Quantification” was published in the Proceedings of the 19th IEEE International Conference on Network Protocols (ICNP), Vancouver, BC, Canada, October 17–20, 2011.

F. Chen is with BloomReach, Inc., Mountain View, CA 94041 USA (e-mail: fei@bloomreach.com).

B. Bezawada is with the Department of Computer Science and Technology, Nanjing University, Nanjing 500032, China (e-mail: bru@nju.edu.cn).

A. X. Liu is with the Department of Computer Science and Engineering, Michigan State University, East Lansing, MI 48824 USA, and also with the Department of Computer Science and Technology, Nanjing University, Nanjing 500032, China (e-mail: alexliu@cse.msu.edu).

Color versions of one or more of the figures in this paper are available online at <http://ieeexplore.ieee.org>.

Digital Object Identifier 10.1109/TNET.2014.2320981

Fig. 1. Example of end-to-end network reachability.

through the given path. The network devices deployed along this path belong to three different parties, i.e., S_1 and FW_1 belong to Subnet₁; FW_2 , FW_3 , and R_1 belong to the Internet service provider (ISP); and FW_4 and S_2 belong to Subnet₂.

Keeping the reachability restriction information private is important for two reasons. First, such information is often misconfigured and has security holes that can be exploited by attackers if it is disclosed. In reality, most firewall policies have security holes [10]. Disclosing ACLs allows attackers to analyze and utilize the vulnerabilities of subnets along a given path. For example, if ACLs along a path from Subnet₁ to Subnet₂ do not block some worm traffic, attackers can break into Subnet₂ from Subnet₁. In practice, neither ISPs nor private networks disclose their ACLs. Second, the reachability restriction information of a network device contains private information, e.g., the ACLs of a network device contain the IP addresses of servers, which can be used by an attacker to launch more targeted attacks. In practice, even within an organization, often no employees other than the firewall administrators are allowed to access their firewall policies. Hence, safeguarding the privacy of network devices is a critical requirement in network reachability quantification.

C. Cross-Domain Quantification of Reachability

To our best knowledge, no prior work has addressed the problem of privacy-preserving network reachability quantification. We propose the first privacy-preserving protocol for quantifying network reachability for a given network path across multiple domains. First, for the network devices belonging to each party, we convert the reachability restriction information of these devices to an ACL by leveraging an existing network reachability quantification tool [6]. This tool takes as input the reachability restriction information, including ACLs, all possible network transforms (e.g., NAT and PAT), and protocol states (e.g., connection-oriented and stateless), and outputs an ACL. Note that ACLs are actually the most important security components for network devices, such as network routers, firewalls, and layer-3 switches, to filter the network traffic. The network devices in each party may include multiple ACLs and also other reachability restriction information. Considering the example in Fig. 1, Fig. 2 shows the three resulting ACLs, A_1 , A_2 , and A_3 , for Subnet₁, ISP, and Subnet₂, respectively. For ease of presentation, in the rest of paper, we use “ACL” to denote the resulting ACL converted from multiple ACLs as well as other reachability restriction information in one party. Second, we calculate the set of packets that are accepted by all the resulting ACLs on a given network path in a privacy-preserving manner. Typically, an ACL consists of multiple rules, and each rule has several

Fig. 2. Three resulting ACLs converted from Fig. 1.

fields, such as source and destination IP addresses, etc., with an accept or discard decision. Given this, the network reachability quantification requires the comparison of the rules of adjacent ACLs. However, a direct comparison of the ACL rules is complex and error-prone as there may be several overlapping rules that require further analysis. For example, a single rule in one ACL may overlap with multiple rules in another ACL. Determining the exact nature of the overlap is important for accurate quantification of network reachability.

Our proposed cross-domain quantification approach of network reachability can be very useful for many applications. We illustrate this using two example scenarios. First, a global view of the network reachability can help ISPs to define better QoS policies to improve traffic management. For example, the knowledge of the different paths through which a particular type of traffic is allowed by the ACLs can help the ISPs to maintain a rated list of the best-quality paths in case of path failures. Second, since network reachability is crucial for many companies that provide their services over the Internet, performing a privacy-preserving computation of the network reachability could become a new business for the ISPs and other parties involved in this computation. The ISPs can answer the reachability queries of these companies using this global knowledge and even provide some information regarding the quality of various paths.

Note that ACLs (also called stateless firewalls), stateful firewalls, and Intrusion Prevention Systems (IPSs) are three different mechanisms for packet filtering. ACLs are stateless by definition as they make the decision for accepting or discarding a packet based on the packet itself and do not consider the packets that they see in the past. They mainly check the well-known five-tuple of the packet header—source IP address, destination IP address, source port number, destination port number, and protocol type—and make a decision regarding the packet. Stateful firewalls make the decision for accepting or discarding a packet based on both the current packet and the packets that they have seen in the past. They consider previous packets by checking whether the connection has been established and allow only those packets that have a matching connection in the past. IPSs make decisions on accepting or discarding packets based on packet payload. They check packet payload mainly by performing regular expression matching where the regular expressions correspond to malicious packet signatures. The scope of this paper is on network reachability based on packet headers. Thus, the technical discussion of this paper is primarily based on ACLs. We will also discuss how to adapt our solutions to deal with stateful firewalls in Section VI.

D. Technical Challenges

There are four key challenges in the privacy-preserving quantification of network reachability.

- 1) It is computationally expensive. An ACL may consist of many rules, and each rule consists of multiple fields. Therefore, comparing multiple ACLs with a large number of rules can be quite expensive, even if only a few ACLs are involved in the process. Furthermore, the complexity of comparison can be expensive due to overlapping rules resulting in many comparisons.
- 2) Protecting the privacy of the ACL rules is crucial. Since a rule has to be sent to other parties to enable comparison, it is necessary to propose a protocol that will not reveal the rule but still allows the different ACLs to calculate the intersection.
- 3) Communication cost is high as even calculating the intersection of a small number of ACLs is a tedious process and requires a number of messages to be exchanged among different parties.
- 4) Computing the reachability information when ACLs are updated is an important performance-related issue. It is necessary to explore optimization approaches in such scenarios without sacrificing the privacy of individual ACLs.

E. Our Approach

We propose the first cross-domain privacy-preserving protocol for quantifying network reachability. We note that the domains could be connected through multiple ISPs, or they could be independently administered domains within the same ISP. We consider n ACLs, $A_1, \dots, A_n, n \geq 2$, in a given network path where each ACL belongs to a distinct party. Our protocol consists of three phases: ACL preprocessing, ACL encoding and encryption, and ACL comparison.

In the first phase, we transform all the ACLs into an equivalent representation, firewall decision diagram (FDD) [11], [12], and then extract the nonoverlapping rules with accept decisions. In the second phase, to perform privacy-preserving comparison, we reduce the problem to that of computing privacy-preserving intersection of two numerical ranges. Accordingly, we first transform the rules, which are represented as ranges, into a sequence of prefix numbers, and then encrypt these numbers with secret keys of different parties. This phase enables different parties to compute the intersection of nonoverlapping rules in their ACLs without revealing these rules.

In the third phase, the destination ACL computes the intersection of its nonoverlapping rules with the rules from its adjacent ACL, and then the adjacent ACL further repeats this computation with its adjacent ACL until the source ACL is reached. Finally, all the ACLs collaboratively decrypt the encrypted intersection of the nonoverlapping rules, but only the first party (with the source ACL) obtains the result.

To reduce the computation and communication cost, we use the divide-and-conquer strategy to divide the problem of computing reachability of n ACLs to the problem of computing reachability of three ACLs. The initial intersection is performed among the rules of three adjacent ACLs that are located in a sequence along the network path. Subsequent comparisons are grouped in a similar manner, i.e., the intersection of three ACLs can be treated as a new ACL, and the process is repeated among three new ACLs. This optimization technique reduces the number of ACL encryptions and the number of messages in our protocol from $O(n^2)$ to $O(n)$.

To handle the issue of changes to ACL rules, we analyze various plausible scenarios and use the approach from [22] to localize the reachability computation. We show that our approach is effective in reducing reachability computation across the entire network in the event of such ACL updates.

F. Summary of Experimental Results

We performed extensive experiments over real and synthetic ACLs. Our experimental results show that the core operation of our protocol is efficient and suitable for real applications. The online processing time of an ACL with thousands of rules is less than 25 s, and the comparison time of two ACLs is less than 5 s. The communication cost between two ACLs with thousands of rules is less than 2100 kB.

G. Key Contributions

The following are our key contributions.

- 1) We propose the first cross-domain privacy-preserving protocol to quantify network reachability across multiple domains. Our protocol can accurately compute the intersection of the rules among the ACLs along a given network path without the need to share these rules across those domains. This is the first step toward privacy-preserving quantification of network reachability, and it can be extended to other network metric measurements that are sensitive in nature. Furthermore, we propose an optimization technique to reduce computation and communication cost of our protocol. It reduces the number of ACL encryptions and the number of messages from $O(n^2)$ to $O(n)$.
- 2) We describe an approach to handle ACL updates in an effective manner.
- 3) We conducted extensive experiments on both real and synthetic ACLs, and the results show that our protocol is efficient and suitable for real applications.

II. RELATED WORK

A. Network Reachability

Several existing approaches [4]–[9], [15] have studied the problem of quantifying network reachability. The key challenges in network reachability include misconfiguration of ACLs, changes of routing policies, and link failures, which could prevent accessibility to essential network services. To estimate reachability, existing approaches analyze ACLs while considering other critical parameters like dynamic routing policies, packet transforms, and variations in protocol operations. To estimate the bounds on reachability, Xie *et al.* [9] defined union and intersection operations over ACLs while taking into account the routing decisions, packet transforms, and link failures. This approach, however, overapproximates and does not yield exact bounds. Ingols *et al.* [5] used binary decision diagrams (BDDs) to reduce the complexity of handling ACLs and to estimate reachability more accurately. Matousek *et al.* [7] described a formal model using interval decision diagrams (IDDs) to analyze network reachability under all possible network failure conditions. However, the approach is not scalable as it performs an exhaustive evaluation of failure scenarios that may or may not occur. Al-Shaer *et al.* [4] proposed a more accurate model using BDDs. They applied symbolic model checking techniques on properties specified in computation tree logic

(CTL) [13] to verify reachability across the network for any given packet. Sung *et al.* [8] studied the effect of reachability constraints on class-of-service flows, where the packets are subjected to an additional constraint based on their class of service. Liu *et al.* [6] used FDDs to quantify reachability while considering all possible network transforms like NAT and PAT, as well as protocol states like connection-oriented, stateless, and so on.

While most solutions operate on static configurations, some work has been proposed for estimating reachability in an online manner. Bandhakavi *et al.* [14] analyzed the network reachability using a simulated routing environment, i.e., they constructed a routing graph that represents the possible paths that could be taken by routing advertisements under the current router configurations. Analysis of this graph helps to identify violations in security policies and in verifying reachability. Zhang *et al.* [15] described a real-time monitoring and verification system for network reachability. A monitoring software runs on all the routers and collects up-to-date ACL and forwarding state information, which enables the network administrator to determine instantaneous reachability between any source–destination pair.

Several other works have been proposed to reduce the complexity of managing networks and to verify network configurations. Casado *et al.* [16] described a novel architecture for enterprise, Secure Architecture for the Networked Enterprise (SANE), which comprises a centralized authentication server that allows authorized users to access services. In SANE, the ACLs can be specified in a natural way so as to capture the semantics clearly. Le *et al.* [17] used data mining techniques to analyze security policies and to detect possible misconfigurations in the policies. They considered the notion of association rule mining to extract usable safe configurations of routers and detect anomalies in other routers using the extracted patterns. Benson *et al.* [18] described complexity metrics to evaluate relative complexity among alternate network designs. The metrics allow network operators to compare configurations with standard configurations and identify errors.

All these approaches are based on the same assumption that there is a central network analyst who has the complete knowledge of the network configuration and other critical information. However, in reality, this assumption is not true for a network where network devices belong to different independent domains and whose network configuration cannot be shared with other domains due to privacy concerns. Therefore, these approaches cannot quantify network reachability across a network spanning different domains.

B. Privacy-Preserving Set Operation

The other work partly related to our work is on privacy-preserving set operations. These solutions enable n parties, where each party has a private set S_i , to collaboratively compute the intersection of all sets, $S_1 \cap \dots \cap S_n$, without disclosing more information of one party's private set beyond the intersection to other parties [23]–[25]. Although it is possible to explore the application of these solutions to solve the problem of privacy-preserving network reachability, the communication cost of these solutions is prohibitive due to the messages exchanged during the privacy-preserving set operations. In privacy-preserving set operations, after the computation, every party needs to know the

final result, i.e., $s_1 \cap \dots \cap s_n$, while in privacy-preserving network reachability, only the first party needs to know the result. This requirement significantly alters the communication cost of the resulting protocol. Under the semi-honest model [26], the state-of-the-art solution for privacy-preserving set operations [24] can achieve $O(ndw^2m^d)$ for n parties, each party with an ACL of m rules over d w -bit domains, while our solution can achieve $\min(O(ndwm^d), O(n2^w))$, which is far more efficient when w is large. Such complexity analysis will be discussed in Section VII-B.

C. Privacy-Preserving Collaborative Firewall Enforcement in VPN

Previous work on collaborative firewall enforcement in virtual private networks (VPNs) [19]–[21] differs from our work from three perspectives. First and most importantly, the problems being addressed are different. Such work focuses on enforcing firewall policies over encrypted VPN tunnels without leaking the privacy of the remote network's firewall policy, whereas our work focuses on privacy-preserving calculation of the intersection of multiple ACLs, which is a new problem. Second, the privacy requirements are different. Such work preserves only the privacy of the remote network's policy, whereas our work preserves the privacy of all parties that are involved in the quantification of network reachability. Last but not least, previous approaches do not require decryption or decoding because their comparison result is whether a packet matches a rule, whereas our work requires decryption of network reachability that is represented by a set of ACLs.

III. PROBLEM STATEMENT AND THREAT MODEL

A. ACLs

We consider quantifying network reachability using the ACLs of different domains that are converted to enable this computation. Each ACL A is an ordered list of *rules*, and each rule is composed of a *predicate* over d *fields*, $F_1, \dots, F_d$ and a *decision* for the packets that match the predicate. Typically, an ACL checks five fields, source IP (32 bits), destination IP (32 bits), source port (16 bits), destination port (16 bits), and protocol type (8 bits). A packet over the d fields $F_1, \dots, F_d$ is a d -tuple $(p_1, \dots, p_d)$, where each p_i is an element of the domain of field F_i , denoted as $D(F_i)$. A *predicate* specifies a set of packets over the d fields, $F_1 \in S_1 \wedge \dots \wedge F_d \in S_d$, where each $S_i \subseteq D(F_i)$ and is specified as either a prefix or a range. A packet $(p_1, \dots, p_d)$ *matches* a rule $F_1 \in S_1 \wedge \dots \wedge F_d \in S_d \rightarrow \langle \text{decision} \rangle$ if and only if the condition $p_1 \in S_1 \wedge \dots \wedge p_d \in S_d$ holds. Typical decisions include: accept, discard, accept with logging, and discard with logging. Without loss of generality, we only consider *accepting rules*, i.e., rules with accept decisions, and *discarding rules*, i.e., rules with discard decisions. Two rules in an ACL may conflict, i.e., they have different decisions and there is at least one packet that matches both rules. To resolve these conflicts, ACLs usually employ a first-match semantics, i.e., the decision of the first-matching rule is enforced on the packet. The matching set of r_i , $M(r_i)$, is the set of all possible packets that match the rule r_i [28]. Table I shows an example ACL, the format of which is based upon the format of Cisco Access Control Lists.

TABLE I
EXAMPLE ACL

Rule	Src. IP	Dest. IP	Src. Port	Dest. Port	Prot.	Action
r_1	123.24.*.*	242.168.0.1	*	80	TCP	accept
r_2	219.27.*.*	242.168.0.2	*	25	TCP	discard
r_3	*	*	*	*	*	discard

B. Problem Statement

We focus on quantifying the end-to-end network reachability for a given network path with multiple network devices belonging to different parties. The network devices are connected with physical interfaces for filtering outgoing packets and incoming packets. A network path is a unidirectional path for transferring packets from the source to the destination. Along the given network path, there are multiple ACLs for filtering these packets. There may be multiple ACLs that are enforced by the same domain. To convert them to a single ACL for one party, we first employ the existing network reachability approach, Quarnet [6], and generate reachability matrices. Second, we use the query language of Quarnet to obtain a set of packets that can pass through the given path within the party. Finally, we convert the set of packets to a single ACL. Without loss of generality, in the rest of paper, we use the term “ACL” to denote the resulting ACL converted from multiple ACLs in one party. Given an ACL A , let $M(A)$ denote the set packets that are accepted by A . Given a network path with n ACLs $A_1, A_2, \dots, A_n$ for transferring packets from A_1 to A_n , where A_i belongs to the party P_i ($1 \leq i \leq n$), quantifying the network reachability is computing the intersection among $M(A_1), \dots, M(A_n)$, i.e., $M(A_1) \cap M(A_2) \cap \dots \cap M(A_n)$.

In our context, we aim to design a privacy-preserving protocol that enables the first party P_1 to compute the intersection of n ACLs, $n \geq 2$, $M(A_1) \cap M(A_2) \cap \dots \cap M(A_n)$ without revealing rules in an ACL A_i ($1 \leq i \leq n$) to any other party P_j ($j \neq i$). We make the following assumptions: 1) The destination of the network path cannot be an intermediate network device. In other words, the destination ACL A_n should filter the packets to end users but not to another network device. 2) All parties follow the privacy-preserving protocol for quantifying the network reachability.

Note that if one party does not want to involve or only wants to provide part of its ACL rules, the party P_1 can still run the protocol to compute network reachability among the remaining ACLs. This requirement is very important especially for the party who really cares about the security of its private network, e.g., a bank that will not share with other parties the information about which packets are allowed to enter into its private network.

C. Threat Model

We consider the semi-honest model, where each party follows our protocol correctly but it may try to learn the ACL rules of other parties [26], [27]. For example, the party P_1 may use the intermediate results to reveal the ACL rules of other parties. The semi-honest model is realistic in our context because a malicious party cannot gain benefits by providing a forged ACL or not following our protocol. For instance, in [27], the benefit of launching active collusion attacks is almost negligible, and even damaging to the individual parties. A similar scenario is

present in our protocol, where launching such attacks neutralizes the benefits of computing network reachability, especially if the other ACL might undergo changes in a short period of time.

IV. PRIVACY-PRESERVING QUANTIFICATION OF NETWORK REACHABILITY

To compute the network reachability from A_1 to A_n , our privacy-preserving protocol consists of three phases, *ACL preprocessing*, *ACL encoding and encryption*, and *ACL comparison*. In the first phase, ACL preprocessing, each party converts its ACL to a sequence of accepting rules. The union of the matching sets of these accepting rules is equal to the set of packets that are accepted by the ACL. In the second phase, ACL encoding and encryption, each party encodes and encrypts each field of its accepting rules for preserving the privacy of its ACL. In the third phase, ACL comparison, all parties compare their ACLs, and finally the party P_1 finds out the set of packets that are accepted by all ACLs. Particularly, P_{n-1} compares the encoded and encrypted accepting rules from A_{n-1} to those from A_n , and finds out the multiple accepting rules whose union is equal to the intersection of $M(A_n)$ and $M(A_{n-1})$. Then, P_{n-2} compares the accepting rules from ACL A_{n-2} to the resulting accepting rules in the first step, and finds out the multiple accepting rules whose union is equal to $M(A_n) \cap M(A_{n-1}) \cap M(A_{n-2})$. This step is repeated until P_1 finds out the multiple accepting rules whose union is equal to $M(A_1) \cap \dots \cap M(A_n)$. Note that the resulting accepting rules of each step are in an encrypted format that prevents any party from revealing these rules by itself. To reveal the final accepting rules, P_1 requires all other parties to decrypt these rules with their private keys, and then P_1 decrypts these rules.

We assume that each S_i in an ACL rule $S_1 \wedge \dots \wedge S_i \wedge \dots \wedge S_n$ is represented in the range format. Therefore, the basic problem of privacy-preserving network reachability is to compute the intersection among the multiple ranges of the ACL rules belonging to different parties in a privacy-preserving manner. This problem boils down to the problem of computing intersection of two ranges $[a, b]$ and $[a', b']$, denoted as $[a, b] \cap [a', b']$. Thus, we first describe the privacy-preserving protocol for computing $[a, b] \cap [a', b']$, and then describe the three phases in our network reachability protocol.

A. Privacy-Preserving Range Intersection

To compute the intersection of a range $[a, b]$ from A_i and a range $[a', b']$ from A_j , our basic idea is to check which range among $[\min, a - 1]$, $[a, b]$, and $[b + 1, \max]$ includes a' or b' , where $\min$ and $\max$ are the minimum and maximum numbers, respectively. Thus, the problem of computing $[a, b] \cap [a', b']$ reduces to the problem of checking whether a number is in a range, e.g., $a' \in [\min, a - 1]$, which can be solved by leveraging the prefix membership verification scheme in [20]. Our scheme consists of five steps. We label each step with the party that is performing that step.

1) $[P_i]$: *Range Transformation*: The party P_i transforms each range $[a, b]$ to three ranges $[\min, a - 1]$, $[a, b]$, and $[b + 1, \max]$, where $\min$ and $\max$ are the minimum and maximum numbers of the corresponding field's domain, respectively. For example, $[5, 7]$ is converted to $[0, 4]$, $[5, 7]$, and $[8, 15]$, where 0 and 15 are the minimum and maximum

numbers. Note that $[\min, a - 1]$ and $[b + 1, \max]$ may not exist. If $a = \min$, then $[\min, a - 1]$ does not exist; if $b = \max$, then $[b + 1, \max]$ does not exist.

2) $[P_i]$: *Range to Prefix Set Conversion*: The party P_i converts each range to a minimum set of prefixes, whose union corresponds to the range. Let $\mathcal{S}([\min, a - 1])$, $\mathcal{S}([a, b])$, and $\mathcal{S}([b + 1, \max])$ denote the resulting sets of prefixes for the three ranges, respectively. For example, $\mathcal{S}([5, 7]) = \{0101, 011^*\}$, where “ * ” denotes that this bit can be 0 or 1.

3) $[P_j]$: *Prefix Family Generation*: The party P_j generates the *prefix families* of a and b , denoted as $\mathcal{F}(a)$ and $\mathcal{F}(b)$. The *prefix family* $\mathcal{F}(a)$ consists of a and all the prefixes that contains a . Assuming w is the bit length of a , $\mathcal{F}(a)$ consists of $w + 1$ prefixes where the l th prefix is obtained by replacing the last $l - 1$ bits of a by * . For example, as the binary representation of 6 is 0110, we have $\mathcal{F}(6) = \{0110, 011^*, 01^{**}, 0^{***}, 0^{****}\}$. It is easy to prove that $a' \in [a, b]$ if and only if $\mathcal{F}(a') \cap \mathcal{S}([a, b]) \neq \emptyset$.

$[P_i, P_j]$: *Prefix Numericalization*: P_i and P_j convert the resulting prefixes to numbers so that they can encrypt them in the next step. We use the prefix numericalization scheme in [29]. This scheme basically inserts 1 before * 's in a prefix and then replaces every * by 0. For example, 01^{**} is converted to 01100. If the prefix does not contain * 's, we place 1 at the end of the prefix. For example, 1100 is converted to 11001. Given a set of prefixes S , we use $\mathcal{N}(S)$ to denote the resulting set of numericalized prefixes. Thus, $a' \in [a, b]$ if and only if $\mathcal{N}(\mathcal{F}(a')) \cap \mathcal{N}(\mathcal{S}([a, b])) \neq \emptyset$.

4) $[P_i, P_j]$: *Private Set Intersection*: Checking whether $\mathcal{N}(\mathcal{F}(a')) \cap \mathcal{N}(\mathcal{S}([a, b])) \neq \emptyset$ is basically checking whether an element from $\mathcal{N}(\mathcal{F}(a'))$ is equal to an element from $\mathcal{N}(\mathcal{S}([a, b]))$. We use commutative encryption (e.g., [30] and [31]) to do this check. Given a number x and two encryption keys K_i and K_j , a commutative encryption satisfies the property $((x)_{K_i})_{K_j} = ((x)_{K_j})_{K_i}$, i.e., encryption with K_i and then K_j is equivalent to encryption with K_j and then K_i . For ease of presentation, we use $(x)_{K_{ij}}$ to denote $((x)_{K_i})_{K_j}$.

5) $[P_i]$: *Commutative Encryption of $\mathcal{N}(\mathcal{S}([a, b]))$* : In our scheme, to check whether $\mathcal{N}(\mathcal{F}(a')) \cap \mathcal{N}(\mathcal{S}([a, b])) \neq \emptyset$, P_i encrypts numbers in $\mathcal{N}(\mathcal{S}([a, b]))$ with its private key K_i and sends it to P_j . Then, P_j further encrypts them by its private key K_j and sends them back to P_i . Let $\mathcal{N}(\mathcal{S}([a, b]))_{K_{ij}}$ denote the result.

6) $[P_j]$: *Commutative Encryption of $\mathcal{N}(\mathcal{F}(a'))$* : P_j encrypts numbers in $\mathcal{N}(\mathcal{F}(a'))$ with K_j , sends it to P_i . Then, P_i encrypts them by K_i . Let $\mathcal{F}(a')_{K_{ji}}$ denote the result. Finally, P_i can check whether there is a common element in two sets $\mathcal{N}(\mathcal{S}([a, b]))_{K_{ij}}$ and $\mathcal{F}(a')_{K_{ji}}$. Through the previous steps, P_i knows which range among $[\min, a - 1]$, $[a, b]$, and $[b + 1, \max]$ includes a' or b' . Then, P_i can compute $[a, b] \cap [a', b']$. For example, if $a' \in [\min, a - 1]$ and $b' \in [a, b]$, $[a, b] \cap [a', b'] = [a, b']$. Note that a' and b' are in the form of $\mathcal{F}(a')_{K_{ji}}$ and $\mathcal{F}(b')_{K_{ji}}$. P_i cannot reveal a' and b' without knowing P_j 's private key K_j . Fig. 3 illustrates the intersection computation of $[5, 7]$ (in A_1) and $[6, 15]$ (in A_2).

B. ACL Preprocessing

In the ACL preprocessing phase, each party P_i ($1 \leq i \leq n$) computes the set of packets $M(A_i)$ that are accepted by its ACL A_i . To achieve this purpose, P_i first converts its ACL

Fig. 3. Privacy-preserving range intersection.

Fig. 4. (a)–(c) Conversion of A_1 .

to an equivalent sequence of nonoverlapping rules. Nonoverlapping rules have an important property, that is, for any two nonoverlapping rules nr and nr' , the intersection of the two corresponding matching sets is empty, i.e., $M(nr) \cap M(nr') = \emptyset$. Thus, any packet p matches one and only one nonoverlapping rule converted from A_i and the decision of this nonoverlapping rule is the decision of A_i for the packet p . Therefore, instead of computing the set $M(A_i)$, P_i only needs to retrieve all nonoverlapping accepting rules because the union of matching sets of these rules is equal to $M(A_i)$. The preprocessing of each A_i includes three steps as follows.

1) P_i converts its ACL A_i to an equivalent acyclic directed graph, called FDD [11]. An FDD construction algorithm, which converts an ACL to an equivalent FDD, is presented in [32]. Fig. 4(b) shows the FDD constructed from Fig. 4(a).

2) P_i extracts nonoverlapping accepting rules from the FDD. We do not consider nonoverlapping discarding rules because the packets discarded by any ACL A_i cannot pass through the path. Fig. 4(c) shows the nonoverlapping accepting rules extracted from the FDD in Fig. 4(b).

Next, P_1 needs to compare its accepting rules from A_1 to those from the other $n - 1$ ACLs. Without loss of generality, in Sections IV-C and IV-D, we use a simplified example in Fig. 5 to show that how to compute the network reachability among three ACLs. Each ACL has only one field, and the domain for the field is $[0, 15]$. Note that in Fig. 5, $nr_1^{(i)}$ and $nr_2^{(i)}$ denote two nonoverlapping accepting rules for ACL A_i ($1 \leq i \leq 3$). Clearly, the network reachability among these three

Fig. 5. Example of three adjacent ACLs.

ACLs can be denoted as two accepting rules $F_1 \in [0, 2] \rightarrow a$ and $F_1 \in [6, 7] \rightarrow a$. Next, we will show how to compute these two rules in a privacy-preserving manner.

C. ACL Encoding and Encryption

In the ACL encoding and encryption phase, all parties need to convert their nonoverlapping accepting rules to another format such that they can collaboratively compute the network reachability while one party cannot reveal the ACL of any other party. Recall the privacy-preserving range intersection scheme in Section IV-A, two parties employ different encoding methods: One converts a range $[a, b]$ to a set of prefixes $\mathcal{S}([a, b])$, and another converts a number a' to its prefix family $\mathcal{F}(a')$. Assume that each party P_i ($1 \leq i \leq n$) has a private key K_i . Let $\mathcal{H}$ denote the encoding function used by the party P_j ($1 \leq j \leq n-1$). Let $F_1 \in [a_1, b_1] \wedge \dots \wedge F_d \in [a_d, b_d]$ denote the predicate of an accepting rule over d fields. The encoding and encryption result of this accepting rule for A_j is $\mathcal{H}_{K_j, \dots, n}([a_1, b_1]) \wedge \dots \wedge \mathcal{H}_{K_j, \dots, n}([a_d, b_d])$. Let $\mathcal{L}$ denote the encoding function used by the party P_n . Considering the above accepting rule, the result is $\mathcal{L}_{K_n}([a_1, b_1]) \wedge \dots \wedge \mathcal{L}_{K_n}([a_d, b_d])$. We discuss the procedure of these two encoding and encryption methods in detail as follows.

1) Encoding and Encryption of ACL A_j ($1 \leq j \leq n-1$):

a) For each nonoverlapping accepting rule $F_1 \in [a_1, b_1] \wedge \dots \wedge F_d \in [a_d, b_d]$, P_j performs *Range Transformation* as described in Section IV-A.1. Fig. 6(b) shows the ranges generated from Fig. 6(a).

b) P_j uses the approach in Section IV-A.2 to convert each range into sets of prefixes. Fig. 6(c) shows the prefixes generated from Fig. 6(b). That is, for the three ranges converted from $[a_l, b_l]$, compute $\mathcal{S}([\min_l, a_l - 1])$, $\mathcal{S}([a_l, b_l])$, $\mathcal{S}([b_l + 1, \max_l])$.

c) P_j unions all these prefix sets and permutes these prefixes. Fig. 6(d) shows the resulting prefix set. This step has two benefits. First, it avoids encrypting and sending duplicate prefixes, and hence significantly reduces the computation and communication costs for the next two steps. Second, it enhances the security; any other parties except P_j cannot reconstruct the nonoverlapping accepting rules because it is difficult to correlate the prefixes to their corresponding rules without the knowledge of the original ACL.

d) P_j uses the *Prefix Numericalization* approach of Section IV-A.4 to numericalize and encrypt each number using K_j . Fig. 6(e) and (f) shows the numericalized and encrypted prefixes.

e) P_j sends the resulting encrypted prefixes to P_{j+1} , which further encrypts them with its private key K_{j+1} . Then, P_{j+1} sends the resulting values to P_{j+2} , which further encrypts them with K_{j+2} . This process is repeated until P_n encrypts them. Finally, P_n sends to P_j the resulting prefixes that are encrypted $n - j + 1$ times. Fig. 6(f) shows the result after encrypting by

Fig. 6. (a)–(h) Encoding and encryption of ACL A_1 . Fig. 7. (a)–(c) Encoding and encryption of ACL A_3 .

P_1 , and Fig. 6(g) shows the result after encrypting by P_2 and P_3 .

f) P_j reconstructs the nonoverlapping accepting rules from the multiple encrypted prefixes because P_j knows which prefix belongs to which field of which rule.

Based on the above steps, the encoding and encryption function used by P_j ($1 \leq j \leq n-1$) is defined as $\mathcal{H}_{K_j, \dots, n}([a_l, b_l]) = (\mathcal{N}(\mathcal{S}(\min_l, a_l - 1))_{K_j, \dots, n}, \mathcal{N}(\mathcal{S}(a_l, b_l))_{K_j, \dots, n}, \mathcal{N}(\mathcal{S}(b_l + 1, \max_l))_{K_j, \dots, n})$, where $[a_l, b_l]$ is the range in the l th field of a rule in A_j . Fig. 8(a) illustrates the encoding and encryption result of ACL A_2 in Fig. 5. The only difference between operations for A_1 and A_2 is that A_1 's numericalized prefixes are encrypted by all the three parties, while A_2 's numericalized prefixes are only encrypted by two parties P_2 and P_3 .

2) Encoding and Encryption of ACL A_n :

a) For each range $[a_l, b_l]$ of a nonoverlapping rule, P_n generates two prefix families $\mathcal{F}(a_l)$ and $\mathcal{F}(b_l)$ using the *Prefix Family Generation* approach from Section IV-A.3. Fig. 7(b) shows the result from Fig. 7(a).

b) P_n uses the *Prefix Numericalization* approach from Section IV-A.4 to numericalize, and then encrypts the prefixes using its private key K_n . Fig. 7(c) shows the resulting prefixes.

Fig. 8. (a)–(f) Comparison of ACLs A_2 and A_3 .

Based on the above steps, the encoding and encryption function used by A_n is defined as $\mathcal{L}_{K_n}([a_l, b_l]) = (\mathcal{N}(\mathcal{F}(a_l))_{K_n}, \mathcal{N}(\mathcal{F}(b_l))_{K_n})$, where $[a_l, b_l]$ is the range in the l th field of a rule.

D. ACL Comparison

In the ACL comparison phase, we need to compare the n sequences of encrypted nonoverlapping accepting rules or encrypted numbers from every two adjacent ACLs. Without loss of generality, we only present the comparison between A_{n-1} and A_n . Fig. 8(a) shows the encrypted nonoverlapping rules converted from A_2 in Fig. 5, and Fig. 8(b) shows the encrypted numbers converted from A_3 in Fig. 5. The comparison includes the following four steps.

1) P_n sends the resulting sequence to P_{n-1} , which further encrypts them with its private key K_{n-1} . Let $\mathcal{L}_{K_n}([a'_l, b'_l]) \wedge \dots \wedge \mathcal{L}_{K_n}([a'_d, b'_d])$ denote the encoded and encrypted result of the accepting rule nr' from A_n . P_{n-1} encrypts it with K_{n-1} , i.e., computes $\mathcal{L}_{K_{n(n-1)}}([a'_l, b'_l]) \wedge \dots \wedge \mathcal{L}_{K_{n(n-1)}}([a'_d, b'_d])$. Fig. 8(c) shows the encrypted result from Fig. 8(b).

2) For each nonoverlapping accepting rule nr from A_{n-1} , P_{n-1} computes $nr \cap nr'$. Let $\mathcal{H}_{K_{(n-1)n}}([a_l, b_l]) \wedge \dots \wedge \mathcal{H}_{K_{(n-1)n}}([a_d, b_d])$ denote the encoded and encrypted result of nr . To compute $nr \cap nr'$, for each field l ($1 \leq l \leq d$), P_{n-1} compares $\mathcal{H}_{K_{(n-1)n}}([a_l, b_l])$ to $\mathcal{L}_{K_{n(n-1)}}([a'_l, b'_l])$, where $\mathcal{H}_{K_{(n-1)n}}([a_l, b_l]) = (\mathcal{N}(\mathcal{S}(\min_l, a_l - 1))_{K_{(n-1)n}}, \mathcal{N}(\mathcal{S}(a_l, b_l))_{K_{(n-1)n}}, \mathcal{N}(\mathcal{S}(b_l + 1, \max_l))_{K_{(n-1)n}}) \mathcal{L}_{K_{n(n-1)}}([a'_l, b'_l]) = (\mathcal{N}(\mathcal{F}(a'_l))_{K_{n(n-1)}}, \mathcal{N}(\mathcal{F}(b'_l))_{K_{n(n-1)}})$. According to the privacy-preserving range intersection, to check whether $a'_l \in [\min_l, a_l - 1]$, P_{n-1} checks whether $\mathcal{N}(\mathcal{S}(\min_l, a_l - 1))_{K_{(n-1)n}} \cap \mathcal{N}(\mathcal{F}(a'_l))_{K_{n(n-1)}} = \emptyset$.

Similarly, P_{n-1} checks whether $a'_l \in [a_l, b_l]$, $a'_l \in [b_l + 1, \max_l]$, and whether $b'_l \in [\min_l, a_l - 1]$, $b'_l \in [a_l, b_l]$, $b'_l \in [b_l + 1, \max_l]$. Based on the result, P_{n-1} computes the intersection between $[a_l, b_l]$ and $[a'_l, b'_l]$, i.e., $[a_l, b_l] \cap [a'_l, b'_l]$.

Let T_l denote $[a_l, b_l] \cap [a'_l, b'_l]$. For example, if $a'_l \in [a_l, b_l]$ and $b'_l \in [b_l + 1, \max_l]$, the condition $a_l \leq a'_l \leq b_l < b'_l$ holds and hence $T_l = [a_l, b_l]$. If for any T_l ($1 \leq l \leq d$) the condition $T_l \neq \emptyset$ holds, then $nr \cap nr' = T_1 \wedge \dots \wedge T_d$.

Fig. 9. Decryption process of the comparison result.

Note that the party P_{n-1} cannot reveal a'_l and b'_l through this comparison because P_{n-1} does not know P_n 's private key K_n . Thus, if $T_l = [a'_l, b'_l]$, P_{n-1} only knows $\mathcal{N}(\mathcal{F}(a'_l))_{K_{n(n-1)}}$ and b_l . We denote the information that P_{n-1} knows about T_l as $\{\mathcal{N}(\mathcal{F}(a'_l))_{K_{n(n-1)}}, b_l\}$. Fig. 8(d) shows the result after comparing A_2 and A_3 . Note that the result may contain the field values from A_{n-1} 's nonoverlapping accepting rules, which are not encrypted, e.g., the field value 6 in Fig. 8(d). This is caused due to only part of the rule in A_{n-1} intersecting with the rule from the A_n . In such cases, the privacy of these values is important, and P_{n-1} needs to protect these values as well. To preserve the privacy of A_{n-1} , the party P_{n-1} encodes and encrypts such values from A_{n-1} 's nonoverlapping accepting rules and before sending the final intersection results to P_n . To ensure correctness of the protocol, for a number a_l , P_{n-1} computes its corresponding prefix family and encrypts it as follows: $\mathcal{N}(\mathcal{F}(a_l))_{K_{(n-1)n}}$. Fig. 8(e) shows the result after encoding and encrypting the value 6.

3) To facilitate the next comparison to A_{n-2} , P_{n-1} sends the comparison result to P_n , and then P_n encrypts the numbers from A_{n-1} 's nonoverlapping accepting rules. Fig. 8(f) shows the encryption result.

These three steps are repeated to further compare A_{n-2} with the result stored in P_n . The comparison phase of the protocol terminates after all parties finish the comparisons, where P_n has the comparison end result. Let $\{\mathcal{N}(\mathcal{F}(a_l))_{K_{1\dots n}}, \mathcal{N}(\mathcal{F}(b_l))_{K_{1\dots n}}\}$ denote the l th field of an encrypted rule in the result. To decrypt this rule, P_n first decrypts it with K_n and sends the values to P_{n-1} . Then, P_{n-1} decrypts it with K_{n-1} and sends the values to P_{n-2} and so on. This step is executed until P_1 decrypts the rule. Finally, P_1 has the result $F_1 \in [a_1, b_1] \wedge \dots \wedge F_d \in [a_d, b_d]$. The comparison result of three ACLs in Fig. 5 is $\{\mathcal{N}(\mathcal{F}(0))_{K_{123}}, \mathcal{N}(\mathcal{F}(2))_{K_{123}}\}$, and $\{\mathcal{N}(\mathcal{F}(6))_{K_{123}}, \mathcal{N}(\mathcal{F}(7))_{K_{123}}\}$. Fig. 9 shows the decryption process of the comparison result.

V. INCREMENTAL UPDATES OF ACLS

An update to an ACL implies that either a rule is added, removed, or modified. Due to this, it may be necessary to update the network reachability accordingly. Note that, in practice, these updates are usually performed within a network defined interval, i.e., they need to be done manually, and it is not possible to automate this process. Regardless of the type of ACL updates, without loss of generality, there are two outcomes of ACL updates. The affected rule either causes additional packets to be accepted or to be discarded at the router. Note that this discussion can be extended to multiple rule updates in a straight-

forward manner. To optimize the process, we consider the effect of all the rule updates, i.e., batch updates, prior to processing the ACL.

A. Addition of Rules With Accept Decision

If the new rule is an accepting rule, then there is no need to update the reachability information for a given source since this rule does not affect the existing set of accepted packets. Note that this rule may increase the set of accepted packets along this path. However, calculating the updated set of accepted packets requires the execution of the privacy-preserving protocol along the path since it is possible that there are discarding rules, either upstream or downstream, that might neutralize the effect of the newly added rule(s). Hence, for all practical purposes, we assume that the nodes will not repeat the reachability protocol execution in this case, and that this information will be reflected during the next reachability computation.

B. Addition of Rules With Discard Decision

If the new rule is a discarding rule, it will result in the following situations. First, the discard rule can overlap with any of the ACLs of its upstream nodes. In this case, adding this rule will not affect the current reachability information as the set of discarding packets remains constant.

Second, the new rule starts discarding packets that were allowed earlier. In this case, the source node needs to update the network reachability information. Note that the new discarding rule affects the set of accepting rules of the intermediate firewall. Since the packets allowed by the intermediate firewall are dependent on its accepting rules, it is necessary to see if any change in these accepting rules affects the reachability information of the source node. To check this, the source node initiates a privacy-preserving firewall comparison [22] with the affected intermediate node using the accepting rules calculated during the reachability analysis. Using this computation, the source node computes the updated reachability information as follows. All rules that are not in the intersection of the two ACLs but are present in the previous reachability accepting rule set are discarded. The removed rules are the rules that are affected due to the addition of the discarding rule of the intermediate rule. We note that this approach, however, cannot scale well if more than one node updates its firewall. This localized approach ensures that all the nodes along the path need not participate in the privacy-preserving reachability computation and, hence, saves bandwidth and computational overhead.

C. Addition of New Routers

If new routers with ACLs are introduced in the network, our protocol checks if the existing reachability is affected due to the introduction of the new ACL. This case is equivalent to the case where new rules are added to an existing ACL. Therefore, the rules of the new router are treated in a similar fashion as described earlier, and the update protocol is run on this new network configuration.

VI. STATEFUL FIREWALLS

We have discussed privacy-preserving quantification of intersection of multiple ACLs across different domains. However, a party on the path may have a stateful firewall to check not

only its ACL, but also its state table. The state table includes the information of whether the connection from source to destination or from destination to source has been established. When a packet does not match any entry in the state table, the packet is discarded, and the connection will fail.

We assume that the network is path-coupled on stateful firewalls. Path-coupled network means that the path from source to destination and the path from destination to source contain the same set of network devices. Most networks are path-coupled. If a party P_i ($1 \leq i \leq n$) has a stateful firewall with its ACL A_i and its state table A_{S_i} , at any moment, P_i can calculate $M(A_i \cap A_{S_i})$, which denotes the set of TCP packets that can pass through the network device of P_i . Thus, if any party P_i ($1 \leq i \leq n$) on the path has a stateful firewall, for any given moment, we can use $A_i \cap A_{S_i}$ to denote its ACL rules instead of A_i . Then, we use our proposed approach to quantify the network reachability from source to destination.

VII. SECURITY AND COMPLEXITY ANALYSIS

A. Security Analysis

The security of our protocol is based on the two important properties of the commutative encryption.

- 1) *Secrecy*: For any x and key K , given $(x)_K$, it is computationally infeasible to compute K .
- 2) *Indistinguishability*: the distribution of $(x)_K$ is indistinguishable from the distribution of x .

Based on the first property, without knowing P_j 's secret key K_j , the party P_i ($i \neq j$) cannot decrypt the encrypted numbers from P_i . Furthermore, one party P_i cannot statistically analyze encrypted numbers from A_j ($i \neq j$) because each party P_j ($1 \leq j \leq n-1$) unions the encrypted prefix numbers into one set before sending them to P_i for further encryption. Therefore, after the first and second phases of our protocol, i.e., *ACL preprocessing* and *ACL encoding and encryption*, the party P_i cannot reveal the ACL of any other party. Based on the second property, we can prove that after the third phase, i.e., *ACL comparison*, the party P_i only learns the limited information of the ACLs of other parties, but such information cannot help it reveal them.

Without loss of generality, we consider the comparison between ACLs A_{n-1} and A_n . For each nonoverlapping rule nr from A_{n-1} , let $V_{F_l,h}(nr)$ denote the h th ($1 \leq h \leq 3$) prefix set for the field F_l ($1 \leq l \leq d$), e.g., in Fig. 8(a), $V_{F_l,1}(nr_1^{(2)})$ denotes $\{00010, 00101\}$. Let $\mathcal{N}(\mathcal{F}(a_l))$ denote one set of prefixes for the field F_l , e.g., $\mathcal{N}(\mathcal{F}(0))$ in Fig. 8(c). The basic operation of the third phase is to compare whether two sets from different ACLs, e.g., $V_{F_l,h}(nr)$ and $\mathcal{N}(\mathcal{F}(a_l))$, have a common element. According to the theorems in multiparty secure computation [33], [34] and the theorem in [22], we can prove that after the three phases, the party P_{n-1} only learns $V_{F_l,h}(nr) \cap \mathcal{N}(\mathcal{F}(a_l))$ and the size of $\mathcal{N}(\mathcal{F}(a_l))$.

To prove this claim, based on the theorems of multiparty secure computation [33], [34], we only need to prove that the distribution of the P_{n-1} 's view of our protocol cannot be distinguished from a simulation that uses only $V_{F_l,h}(nr)$,

$V_{F_l, h}(nr) \cap \mathcal{N}(\mathcal{F}(a_l))$, and the size of $\mathcal{N}(\mathcal{F}(a_l))$. The theorem in [22] proves that P_{n-1} 's view of our protocol

$$Y_R = \left\{ \underbrace{(x_1)_{K_{n-1}}, \dots, (x_m)_{K_{n-1}}}_{x_i \in V_{F_l, h}(nr) \cap \mathcal{N}(\mathcal{F}(a_l))}, \underbrace{(x_{m+1})_{K_{n-1}}, \dots, (x_t)_{K_{n-1}}}_{x_i \in V_{F_l, h}(nr) - \mathcal{N}(\mathcal{F}(a_l))} \right\}$$

cannot be distinguished from the simulation

$$Y_S = \left\{ \underbrace{(x_1)_{K_{n-1}}, \dots, (x_m)_{K_{n-1}}}_{x_i \in V_{F_l, h}(nr) \cap \mathcal{N}(\mathcal{F}(a_l))}, \underbrace{z_{m+1}, \dots, z_t}_{t-m=|V_{F_l, h}(nr) - \mathcal{N}(\mathcal{F}(a_l))|} \right\}$$

where $z_{m+1}, \dots, z_t$ are random values and uniformly distributed in the domain of encrypted numbers.

Knowing $V_{F_l, h}(nr) \cap \mathcal{N}(\mathcal{F}(a_l))$ and the size of $\mathcal{N}(\mathcal{F}(a_l))$, P_{n-1} cannot reveal the rules in A_n for two reasons. First, a numericalized prefix in $V_{F_l, h}(nr) \cap \mathcal{N}(\mathcal{F}(a_l))$ can be generated from many numbers. Considering a numericalized prefix of 32-bit IP addresses $b_1 b_2 \dots b_k 10 \dots 0$, the number of possible IP addresses that can generate such a prefix is 2^{32-k} . Furthermore, after the comparison, P_{n-1} sends to P_n the comparison result that is encrypted with P_{n-1} 's secret key K_{n-1} . Without knowing K_{n-1} , P_n cannot reveal the comparison result and, hence, cannot reveal the values from A_{n-1} . Second, the size of $\mathcal{N}(\mathcal{F}(a_l))$ cannot be used to reveal the rules in A_n because for any a_l or b_l in the field F_l of A_n , the size of $\mathcal{N}(\mathcal{F}(a_l))$ or $\mathcal{N}(\mathcal{F}(b_l))$ is constant.

At the end of our protocol, only P_1 knows the intersection of n ACLs, which includes some information (i.e., numbers) from other ACLs. However, our goal is to preserve the privacy of ACLs, not the privacy of the intersection result. Knowing such numbers cannot help P_1 to reveal an ACL rule of other parties for two reasons. First, a real ACL typically consists of hundreds of rules, and no one consists of only one rule. Second, P_1 does not know which numbers belong to A_j ($2 \leq j \leq n$), which two numbers form an interval, and which d intervals form a rule in A_j . The number of possible combinations can be extremely large. Considering the intersection in Fig. 9, P_1 cannot know which ACL, A_2 or A_3 , contains the number 2 or the number 6.

B. Complexity Analysis

In this section, we analyze the computation, space, and communication costs in our protocol. Let m_i be the number of rules in ACL A_i ($1 \leq i \leq n$) and d be the number of fields. For ease of presentation, assume that different fields have the same length, i.e., w bits. We first analyze the complexity of processing ACLs $A_1, \dots, A_{n-1}$, and then analyze the complexity of processing ACL A_n . The maximum number of nonoverlapping rules generated from the FDD is $(2m_i - 1)^d$ [32]. Each nonoverlapping rule consists of d w -bit intervals, each interval can be converted to at most three ranges, and each range can be converted to at most $2w - 2$ prefixes [35]. Thus, the maximum number of prefixes generated from these nonoverlapping rules is $3d(2w - 2)(2m_i - 1)^d$. Recall that P_i ($1 \leq i \leq n - 1$) unions all prefixes into one set. Then, the number of prefixes cannot exceed 2^{w+1} . Therefore, for processing A_i ($1 \leq i \leq$

$n - 1$), the computation cost of encryption by $P_i, \dots, P_n$ is $\min(3d(2w - 2)(2m_i - 1)^d, 2^{w+1}) = \min(O(dw m_i^d), O(2^w))$, the space cost of P_i is $O(dw(m_i)^d)$, and the communication cost is $\min(O(dw(m_i)^d), O(2^w))$. For processing P_n , each interval of the nonoverlapping rules is converted to two prefix families, and each prefix family includes $w + 1$ prefixes. Thus, the maximum number of prefixes converted from A_n is $2d(w + 1)(2m_n - 1)^d$. Therefore, for processing A_n , the computation, space, and communication cost of P_n is $O(dw(m_n)^d)$.

VIII. PROTOCOL OPTIMIZATION

To reduce the computation and communication costs, we use the divide-and-conquer strategy to reduce the problem of computing reachability of n ACLs to the problem of computing reachability of two ACLs. Then, the intermediate results are aggregated hierarchically to obtain the final reachability result. Let Q_i ($1 \leq i \leq n$) denote the set of nonoverlapping accepting rules from ACL A_i . In the ACL encoding and encryption phase, Q_i is encrypted $n - i + 1$ times, i.e., encrypted by $P_i, P_{i+1}, \dots, P_n$. Thus, the number of encryptions for $Q_1, \dots, Q_n$ is $n + (n - 1) + \dots + 1 = O(n^2)$. Similarly, the number of messages in this phase is $O(n^2)$. To reduce the number of encryptions and messages, we first divide n ACLs into $\lfloor n/2 \rfloor$ groups. The j th ($1 \leq j \leq \lfloor n/2 \rfloor$) group includes two adjacent ACLs A_{2j-1} and A_{2j} . The last group includes adjacent ACLs $A_{2\lfloor n/2 \rfloor - 1}, \dots, A_n$. For example, five ACLs can be divided into two groups $\{A_1, A_2\}$ and $\{A_3, A_4, A_5\}$. Second, for the ACLs in each group, we run the proposed protocol to compute the network reachability. The result for each group is actually a new set of nonoverlapping accepting rules. Therefore, we obtain $\lfloor n/2 \rfloor$ sets of nonoverlapping accepting rules. Repeat these two steps until we obtain the reachability for all n ACLs. Overall, there are $\lfloor n/2 \rfloor + \lfloor n/2^2 \rfloor + \dots + 1 = O(n)$ groups, and for each group with two ACLs, the number of ACL encryptions and messages is $2 + 1 = 3$. Thus, the number of ACL encryptions and messages is reduced from $O(n^2)$ to $O(n)$.

To further reduce the computation and communication costs, we compress the sequence of prefix families from A_n and then encrypt and send them to other parties in the ACL comparison phase. The sequence of prefix families from A_n could have many duplicate numbers. For example, in Fig. 7(c), the sequence of prefix families, $\mathcal{F}(0)$, $\mathcal{F}(2)$, $\mathcal{F}(4)$, and $\mathcal{F}(8)$, has duplicated prefixes 00^{**} , 0^{***} , and $****$. To reduce the costs, P_n first unions all the prefix families from A_n to one set, i.e., removes all duplicated prefixes. Second, P_n replaces each prefix in a prefix family converted from A_n with the index of this number in the set. Let m denote the total number of prefixes in the set. Then, k ($1 \leq k \leq m$) is the index of the k th encrypted number in the set. The sequence of prefix families from A_n is represented as a set of prefixes with the sequence of indexes. Next, P_n only needs to encrypt the set of prefixes, and then sends to other parties the encrypted result with the sequence of indexes.

IX. EXPERIMENTAL RESULTS

We evaluated the efficiency and effectiveness of our protocol on 10 real ACLs and 100 synthetic ACLs. Both real and synthetic ACLs examine five fields: source IP, destination IP, source port, destination port, and protocol type. For real ACLs, the

number of rules ranges from hundreds to thousands, and the average number of rules is 806. Due to security concerns, it is difficult to obtain many real ACLs. Thus, we generated a large number of synthetic ACLs based on Singh *et al.*'s technique [36]. To create a rule in a synthetic ACL, we randomly pick two prefixes from a set of values that follow the same distribution as in real ACLs, one for the source IP and one for the destination IP. Then, we randomly pick source port, destination port, and protocol type from a set of all the values in real ACLs. Finally, we randomly add the decision, accept or discard, to the rule. For synthetic ACLs, the number of rules ranges from 200 to 2000, and for each number, we generated 10 synthetic ACLs. In implementing the commutative encryption, we used the Pohlig–Hellman algorithm [30] with a 1024-bit prime modulus and 160-bit encryption keys. Our experiments were implemented using Java 1.6.0 and carried out on a PC running Linux with 2 Intel Xeon cores and 16 GB of memory.

To evaluate the effectiveness, we verified the correctness of our protocol because we knew all the ACLs in the experiments. The results show that our protocol is deterministic and accurate with the given ACLs. Thus, in this section, we focus on the efficiency of our protocol. Recall that processing ACL A_i ($1 \leq i \leq n-1$) is different from processing the last destination ACL A_n . Therefore, we evaluate the computation and communication costs of the core operations of our protocol, processing ACL A_i ($1 \leq i \leq n-1$), processing the destination ACL A_n , and comparing A_i and A_n . The computation cost is the execution time for each step in our approach and the communication cost is the size of messages exchanged between two parties. We note that the computation cost includes the cost of converting the ACLs to FDDs, which is relatively smaller compared to the cost incurred due to the commutative encryption. Knowing the performance of our protocol, we can easily estimate time and space consumption for a given network path with n ACLs belonging to n parties.

A. Efficiency on Real ACLs

Our protocol is efficient for processing real ACL A_i ($1 \leq i \leq n-1$). Fig. 10(a) shows, for processing A_i , the computation cost of P_i and the average computation cost of other parties $P_{i+1}, \dots, P_n$. The computation cost of P_i is less than 2 s, and the computation cost of P_j ($i+1 \leq j \leq n$) is less than 1.5 s. Note that for processing A_i , the computation cost of P_i is one-time offline cost because P_i knows A_i , while the computation cost of P_j ($i+1 \leq j \leq n$) is online cost. Fig. 10(b) shows the average communication cost between any two adjacent parties P_j and P_{j+1} ($i \leq j \leq n$) for processing ACL A_i , which is less than 60 kB. Note that the computation costs of different parties P_j ($i+1 \leq j \leq n$) are similar because they encrypt the same number of prefixes from A_i . Hence, we only show the average computation cost of parties $P_{i+1}, \dots, P_n$. Similarly, the communication costs between every two adjacent parties P_j and P_{j+1} are the same.

Our protocol is efficient for processing real ACL A_n . Fig. 11(a) shows, for processing A_n , the computation cost of P_n and the average computation cost of other parties. The computation cost of P_n is less than 10 s. The average computation cost of other parties is less than 6 s. Similarly, for processing A_n , the computation cost of P_n is one-time offline

Fig. 10. (a) Computation and (b) communication costs for processing real ACL A_i ($1 \leq i \leq n-1$).

Fig. 11. (a) Computation and (b) communication costs for processing real ACL A_n .

cost, while the computation costs of other parties is online cost. Fig. 11(b) shows the average communication cost between P_n and P_i ($1 \leq i \leq n-1$), which is less than 410 kB.

Our protocol is efficient for real ACL comparison. The comparison time between two ACLs is less than 1 s, which is much less than the computation cost of processing ACLs. The commutative encryption is more expensive than checking whether two sets have a common element.

B. Efficiency on Synthetic ACLs

To further evaluate the efficiency, we executed our protocol over every 10 synthetic ACLs with the same number of rules, and then measured the computation and communication costs of operations on synthetic ACLs A_1 to A_n for different parties. Particularly, we measured computation and communication costs for processing each synthetic ACL A_i ($1 \leq i \leq n-1$), processing synthetic ACL A_n , and the comparison time for every two ACLs. Note that the vertical axis of Figs. 12(a) and 13(a) are in a logarithmic scale.

For processing synthetic ACL A_i ($1 \leq i \leq n-1$), Fig. 12(a) shows the computation cost of P_i and the average computation cost of parties $P_{i+1}, \dots, P_n$, and Fig. 12(b) shows the average communication cost between P_j and P_{j+1} ($i \leq j \leq n$). The one-time offline computation cost, i.e., the computation cost of P_i , is less than 400 s, and the online computation cost, i.e., the average computation cost of other parties P_j , is less than 5 s. The average communication cost between any two adjacent parties P_j and P_{j+1} is less than 450 kB.

For processing synthetic ACL A_n , Fig. 13(a) shows the computation cost of P_n and the average computation cost of other parties, and Fig. 13(b) shows the average communication cost between P_n and P_i ($1 \leq i \leq n-1$). The one-time offline computation cost, i.e., the computation cost of P_n , is less than 550 s, and the online computation cost, i.e., the average computation

Fig. 12. (a) Average computation and (b) average communication costs for processing synthetic ACL A_i ($1 \leq i \leq n-1$).

Fig. 13. (a) Average computation and (b) average communication for processing synthetic ACL A_n .

cost of other parties $P_1, \dots, P_{n-1}$, is less than 25 s. The average communication cost between P_n and P_i is less than 2100 kB.

C. Efficiency of Incremental Updates of ACLs

To evaluate the efficiency of our approach to handle incremental updates of ACLs, we employed the technique of mutation testing [37] to introduce incremental updates of real and synthetic ACLs. This is a well-accepted technique for carrying out testing experiments in real practice. Given a real and a synthetic ACL, we used mutation testing to randomly create an incremental update of the ACL. Particularly, we randomly changed the some ranges of a rule in this ACL. Note that, using this technique, the incremental update of the ACL may result in the equivalent to the original ACL, that is, the FDD of the new ACL is the same as that of the original ACL. In such a case, we created another incremental update of this ACL until the FDD of the new ACL was different from the original ACL. Overall, for each real and synthetic ACL, we generated an incremental update of this ACL in our experiments. Then, we applied our approach proposed in Section V to process each incremental update of an ACL.

Our protocol is efficient for processing incremental update of real ACL A_i ($1 \leq i \leq n-1$). Fig. 14(a) shows, for processing incremental update of A_i , the computation cost of P_i and the computation cost of P_n . Note that only two parties, P_i and P_n , are involved in this process. The computation cost of P_i is less than 0.4 s, and the computation cost of P_n is less than 0.2 s. Both costs are at least 5 times less than those of running the entire protocol in Section IV. Fig. 14(b) shows the average communication cost between two parties P_i and P_n for processing incremental update of ACL A_i , which is less than 20 kB. These communication costs also show an equivalent reduction to that of the computational cost in terms of magnitude.

Similarly, our protocol is efficient for processing incremental update of synthetic ACL A_i ($1 \leq i \leq n-1$). Fig. 15(a) shows

Fig. 14. (a) Average computation and (b) average communication costs for processing updates of real ACL A_i .

Fig. 15. (a) Average computation and (b) average communication costs for processing updates of synthetic ACL A_i .

the computation cost for processing incremental update of A_i for P_i and the computation cost for P_n . The computation cost of P_i is less than 100 s, and the computation cost of P_n is less than 2 s. Fig. 15(b) shows the average communication cost between two parties P_i and P_n for processing incremental update of ACL A_i . The communication cost is less than 120 kB. These computation and communication costs are far less than those of running the entire protocol.

X. CONCLUSION

In this paper, we addressed the problem of privacy-preserving quantification of network reachability across different domains. Protecting the privacy of access control configuration is important as the information can be easily abused. We propose an efficient and secure protocol to quantify the network reachability accurately while protecting the privacy of ACLs. We use the divide-and-conquer strategy to decompose the reachability computation, which results in a magnitude reduction of the computation and communication costs. To validate our protocol, we conducted the experiments on both real and synthetic ACLs, which demonstrate that our protocol has the benefits of fast computation and low communication overhead and is suitable for deployed networks. We showed that ACL modifications can be handled effectively. The experimental results demonstrate that our approach is practical for real applications.

Network reachability quantification is an important requirement for effective network management. It has a number of applications like configuration verification and discovery of security vulnerabilities in network. The process might have to be repeated periodically in a time period to ensure that all services are accessible. This requirement implies that quantification process must be secure as well as efficient, i.e., no information leaks due to the possibility of abuse.

REFERENCES

- [1] F. Chen, B. Bruhadeshwar, and A. X. Liu, "Privacy-preserving cross-domain network reachability quantification," in *Proc. IEEE ICNP*, 2011, pp. 155–164.
- [2] D. Oppenheimer, A. Ganapathi, and D. A. Patterson, "Why do Internet services fail, what can be done about it?," in *Proc. USITS*, 2003, pp. 1–15.
- [3] Z. Kerravala, "As the value of enterprise networks escalates, so does the need for configuration management," *Enterprise Computing & Networking*, The Yankee Group Report, 2004.
- [4] E. Al-Shaer, W. Marrero, A. El-Atawy, and K. ElBadawi, "Network configuration in a box: Towards end-to-end verification of network reachability and security," in *Proc. IEEE ICNP*, 2009, pp. 123–132.
- [5] K. Ingols, R. Lippmann, and K. Piwowarski, "Practical attack graph generation for network defense," in *Proc. ACSAC*, 2006, pp. 121–130.
- [6] A. X. Liu and A. R. Khakpour, "Quantifying and verifying reachability for access controlled networks," *IEEE/ACM Trans. Netw.*, vol. 21, no. 2, pp. 551–565, Apr. 2013.
- [7] P. Matousek, J. Rab, O. Rysavy, and M. Sveda, "A formal model for network-wide security analysis," in *Proc. IEEE Int. Conf. Workshop Eng. Comput. Based Syst.*, 2008, pp. 171–181.
- [8] Y.-W. E. Sung, C. Lund, M. Lyn, S. Rao, and S. Sen, "Modeling and understanding end-to-end class of service policies in operational networks," in *Proc. SIGCOMM*, 2009, pp. 219–230.
- [9] G. G. Xie *et al.*, "On static reachability analysis of IP networks," in *Proc. IEEE INFOCOM*, 2005, pp. 2170–2183.
- [10] A. Wool, "A quantitative study of firewall configuration errors," *Computer*, vol. 37, no. 6, pp. 62–67, Jun. 2004.
- [11] M. G. Gouda and A. X. Liu, "Firewall design: Consistency, completeness and compactness," in *Proc. ICDCS*, 2004, pp. 320–327.
- [12] M. G. Gouda and A. X. Liu, "Structured firewall design," *Comput. Netw. J.*, vol. 51, no. 4, pp. 1106–1120, 2007.
- [13] E. A. Emerson, "Temporal and modal logic," in *Handbook of Theoretical Computer Science*. Cambridge, MA, USA: MIT Press, 1990, pp. 995–1072.
- [14] S. Bandhakavi, S. Bhatt, C. Okita, and P. Rao, "Analyzing end-to-end network reachability," in *Proc. IFIP/IEEE Int. Symp. Integrated Netw. Manage.*, 2009, pp. 585–590.
- [15] B. Zhang, T. S. E. Ng, and G. Wang, "Reachability monitoring and verification in enterprise networks," presented at the SIGCOMM, 2008, (poster).
- [16] M. Casado *et al.*, "Sane: A protection architecture for enterprise networks," in *Proc. USENIX Security Symp.*, 2006, pp. 137–151.
- [17] F. Le, S. Lee, T. Wong, H. S. Kim, and D. Newcomb, "Detecting network-wide and router-specific misconfigurations through data mining," *IEEE/ACM Trans. Netw.*, vol. 17, no. 1, pp. 66–79, Feb. 2009.
- [18] T. Benson, A. Akella, and D. Maltz, "Unraveling the complexity of network management," in *Proc. NSDI*, 2009, pp. 335–348.
- [19] J. Cheng, H. Yang, S. H. Wong, and S. Lu, "Design and implementation of cross-domain cooperative firewall," in *Proc. IEEE ICNP*, 2007, pp. 284–293.
- [20] A. X. Liu and F. Chen, "Collaborative enforcement of firewall policies in virtual private networks," in *Proc. PODC*, 2008, pp. 95–104.
- [21] A. X. Liu and F. Chen, "Privacy preserving collaborative enforcement of firewall policies in virtual private networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 5, pp. 887–895, May 2011.
- [22] F. Chen, B. Bruhadeshwar, and A. X. Liu, "Cross-domain privacy-preserving cooperative firewall optimization," *IEEE/ACM Trans. Netw.*, vol. 21, no. 3, pp. 857–868, Jun. 2013.
- [23] M. Freedman, K. Nissim, and B. Pinkas, "Efficient private matching and set intersection," in *Proc. EUROCRYPT*, 2004, pp. 1–19.
- [24] L. Kissner and D. Song, "Privacy-preserving set operations," in *Proc. CRYPTO*, 2005, pp. 241–257.
- [25] Y. Sang and H. Shen, "Efficient and secure protocols for privacy-preserving set operations," *Trans. Inf. Syst. Security*, vol. 13, no. 1, pp. 9:1–9:35, 2009.
- [26] O. Goldreich, *Foundations of Cryptography: Volume II (Basic Applications)*. Cambridge, U.K.: Cambridge Univ. Press, 2004.
- [27] R. Chen, I. E. Akkus, and P. Francis, "SplitX: High-performance private analytics," in *Proc. SIGCOMM*, 2013, pp. 315–326.
- [28] A. X. Liu and M. G. Gouda, "Complete redundancy detection in firewalls," in *Proc. DBSec*, 2005, pp. 196–209.
- [29] Y.-K. Chang, "Fast binary and multiway prefix searches for packet forwarding," *Comput. Netw.*, vol. 51, no. 3, pp. 588–605, 2007.
- [30] S. C. Pohlig and M. E. Hellman, "An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance," *IEEE Trans. Inf. Theory*, vol. IT-24, no. 1, pp. 106–110, Jan. 1978.
- [31] D. K. H. D. R. Safford and D. L. Schales, "Secure RPC authentication (SRA) for TELNET and FTP," Tech. Rep., 1993.
- [32] A. X. Liu and M. G. Gouda, "Diverse firewall design," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 8, pp. 1237–1251, Aug. 2008.
- [33] R. Agrawal, A. Evfimievski, and R. Srikant, "Information sharing across private databases," in *Proc. SIGMOD*, 2003, pp. 86–97.
- [34] O. Goldreich, "Secure multi-party computations," ver. 1.4, Working draft, 2002.
- [35] P. Gupta and N. McKeown, "Algorithms for packet classification," *IEEE Netw.*, vol. 15, no. 2, pp. 24–32, Apr. 2001.
- [36] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet classification using multidimensional cutting," in *Proc. SIGCOMM*, 2003, pp. 213–224.
- [37] R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "Hints on test data selection: Help for the practicing programmer," *Comput.*, vol. 11, no. 4, pp. 34–41, Apr. 1978.

Fei Chen received the B.S. and M.S. degrees in automation from Tsinghua University, Beijing, China, in 2005 and 2007, respectively, and the Ph.D. degree in computer science from Michigan State University, East Lansing, MI, USA, in 2011.

He is currently a Member of Technical Staff with BloomReach, Inc, Mountain View, CA, USA. His research interests focus on networking, privacy, and security.

Bruhadeshwar Bezawada received the B.E. degree in electronics and communications from Osmania University, Hyderabad, India, in 1998, the M.S. degree in electrical and computer engineering and Ph.D. degree in computer science and engineering from Michigan State University, East Lansing, MI, USA, in 2000 and 2005, respectively.

He is currently a Research Associate with Nanjing University, Nanjing, China. His research interests are in security, networking, and dependable systems.

Alex X. Liu received the Ph.D. degree in computer science from The University of Texas at Austin, Austin, TX, USA, in 2006.

His research interests focus on networking and security.

Dr. Liu is an Associate Editor of the IEEE/ACM TRANSACTIONS ON NETWORKING. He is the TPC Co-Chair of ICNP 2014. He received the IEEE & IFIP William C. Carter Award in 2004, an NSF CAREER Award in 2009, and the Michigan State University Withrow Distinguished Scholar Award

in 2011. He received Best Paper awards from ICNP 2012, SRDS 2012, LISA 2010, and TSP 2009.