Privacy-Preserving Cross-Domain Network Reachability Quantification

Fei Chen
Computer Science and Engineering
Michigan State University, USA
feichen@cse.msu.edu

Bezawada Bruhadeshwar
Center for Security Research
Institute of Information Technology, India
bezawada@iiit.ac.in

Alex X. Liu
Computer Science and Engineering
Michigan State University, USA
alexliu@cse.msu.edu

Abstract—Network reachability is one of the key factors for capturing end-to-end network behavior and detecting the violation of security policies. While quantifying network reachability within one administrative domain is already difficult, quantifying network reachability across multiple administrative domains is more difficult because the privacy of security policies becomes 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 with thousands of rules is less than 25 seconds, the comparison time of two ACLs is less than 6 seconds, and the communication cost between two ACLs with thousands of rules is less than 2100 KB.

I. INTRODUCTION

A. Background and Motivation

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. 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. 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. Therefore, to perform an accurate analysis, administrators need to collect all the reachability restriction information from these network devices. Collecting such information could be 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. Such practice leads to significant number of configuration errors, which has been shown to be the major cause of failure for Internet services [19]. Industry research also shows that a significant percentage of human effort and monetary resources are employed in maintaining the operational status of the network [12]. Several critical business applications and sensitive communication are affected severely 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 quantification tools of network reachability are needed for understanding end-to-end network behavior and detecting configuration errors.

B. Limitation of Prior Art

Performing network reachability analysis is a complex task that involves aggregating and analyzing the reachability restriction information from all the devices along a given network path. The current practice of verifying reachability is to send probing packets. However, probing has two major drawbacks. First, probing is expensive to quantify network reachability because it needs to generate and send significant amount of packets. Second, probing is inaccurate, *e.g.*, it cannot probe the open ports with no server listening on them. Due to these drawbacks of probing, many approaches were proposed to address the reachability problem [2], [11], [13], [18], [24], [26]. The main assumption in all these approaches is that the reachability restriction information of each network device and other configuration state are known to a central network analyst, who is quantifying the network reachability. However, in reality, it is common that the network devices along a given path belong to different parties where the reachability restriction information cannot be shared with others including the network analyst. Fig. 1 shows a typical scenario of network reachability, where $User_1$ wants to know what packets he can send to $User_2$ 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 $Subnet_1$, FW_2, FW_3, and R_1 belong to ISP, FW_4 and S_2 belong to $Subnet_2$.

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 [25]. Disclosing ACLs allows attackers to analyze and utilize the vulnerabilities of subnets along a given path. For example, if ACLs along a path from $Subnet_1$
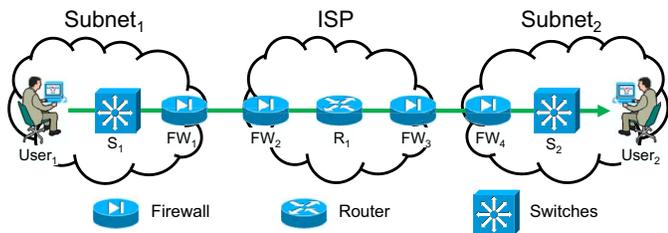
Fig. 1. An example of end-to-end network reachability

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

to Subnet$_2$ do not block some worm traffic, attackers can break into Subnet$_2$ from Subnet$_1$. 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 attacker to launch more targeted attacks. If such information of one device can be shared with other devices, an attacker needs to capture only a single device (or a small subset) to know the security profile of the entire network, *i.e.*, the sensitive information in the ACLs can be abused for gaining profit or to disrupt important services across the network. In practice, even within an organization, often no employees other than the firewall administrators are allowed to access their firewall policies.

C. Cross-Domain Quantification of Network 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 parties. First, for the network devices belonging to each party, we convert the reachability restriction information of these devices to an access control list (ACL) by leveraging the existing network reachability quantification tool [13]. This tool takes as the input the reachability restriction information, including ACLs, all possible network transforms (*e.g.*, NAT and PAT), and protocol states (*e.g.*, connection-oriented and state-less), and outputs an ACL. Note that ACLs are the most important security component for network devices to filter the traffic. Considering the example in Fig. 1, Fig. 2 shows the three resulting ACLs, A_1, A_2, and A_3, for Subnet$_1$, ISP, and Subnet$_2$, 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 the given network path in a privacy-preserving manner. This calculation requires the comparison of the rules in all the ACLs, which is complex and error-prone.

Our proposed cross-domain quantification approach of network reachability can be very useful for many applications. Here we give two examples. First, a global view of the network reachability can help internet service providers (ISPs) to define better QoS policies. 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 the network reachability is crucial for many internet
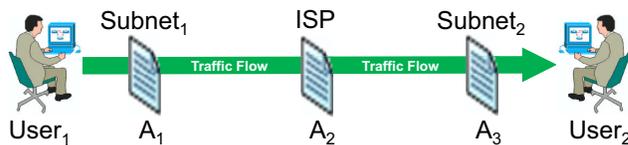
companies, performing a privacy-preserving computation of the network reachability could become a new business for the ISPs and other parties that involve in this computation.

D. Technical Challenges

There are three 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) 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. (3) 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.

E. Our Approach

We propose the first cross-domain privacy-preserving protocol for quantifying network reachability. We consider n ACLs ($n \geq 2$) in a given network path and each ACL belongs to a distinct party. Starting with the ACL A_n, our protocol calculates the intersection of these rules with the rules of the adjacent ACL. Next, using the results of this comparison, the adjacent ACL repeats the process with the next adjacent ACL until the ACL A_1 is involved. Finally, the first party obtains the intersection of the rules from all these ACLs. Briefly, 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) [9], and then extract the non-overlapping rules with accept decisions. In the second phase, to perform privacy preserving comparison, we first transform the rules 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 non-overlapping rules in their ACLs without revealing these rules. In the third phase, the destination ACL computes the intersection of its non-overlapping 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. Note that the comparison result of every two adjacent ACLs is encrypted with multiple secret keys so that no party can reveal the comparison result independently. Finally, all the ACLs collaboratively decrypt the encrypted intersection of the non-overlapping rules, but only the first party (with the source ACL) obtains the result.

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 seconds and the comparison time of two ACLs is less than 5 seconds. The communication cost between two ACLs with thousands of rules is less than 2100 KB.

G. Key Contributions

We make three key contributions. (1) We propose the first cross-domain privacy-preserving protocol to quantify network reachability across multiple parties. 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 parties. This is the first step towards privacy-preserving quantification of network reachability and it can be extended to other network metric measurements that are sensitive in nature. (2) We propose an optimization technique to reduce computation and communication costs. It reduces the number of ACL encryptions and the number of messages from $O(n^2)$ to $O(n)$. (3) We conducted extensive experiments on both real and synthetic ACLs and the result shows that our protocol is efficient and suitable for real applications.

II. RELATED WORK

A. Network Reachability

The challenges in network reachability include, misconfiguration of ACLs, changes of routing policies, and link failures, that 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 [2], [11], [13], [18], [24], [26]. To estimate the bounds on reachability, Xie *et al.* defined union and intersection operations over ACLs while taking into account the routing decisions, packet transforms, and link failures [26]. This approach, however, over approximates and does not yield exact bounds. Ingols *et al.* used Binary Decision Diagrams (BDDs) to reduce the complexity of handling ACLs and to estimate reachability more accurately [11]. Matousek *et al.* described a formal model using Interval Decision Diagrams (IDDs) to analyze network reachability under all possible network failure conditions [18]. 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.* proposed a more accurate model using BDDs and applied symbolic model checking techniques on properties specified in computation tree logic (CTL) [5], to verify reachability across the network for any given packet [2]. Sung *et al.* 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 [24]. Khakpour *et al.* used Firewall Decision Diagrams (FDDs) to quantify reachability while considering all possible network transforms like NAT, PAT as well as protocol states like connection-oriented, state-less and so on

[13]. They also described a query language to enable network operators to execute reachability queries.

All these approaches are based on the same assumption, that is, 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 parties whose network configuration cannot be shared with other parties. Therefore, these approaches cannot quantify network reachability across different parties.

B. Privacy Preserving Set Operation

Other related work is privacy preserving set operations, which enables n parties, each party with its private set s_i, to collaboratively compute the intersection of all sets, $s_1 \cap \cdots \cap s_n$, without disclosing more information of one party's private set beyond the intersection to other parties [6], [14], [22]. Although we could apply these solutions to solve the problem of privacy preserving network reachability, the communication cost of these solutions is prohibitive due to the unnecessary requirement of privacy preserving set operations. In privacy preserving set operations, after computation, every party needs to know the result, *i.e.*, $s_1 \cap \cdots \cap s_n$, while in privacy preserving network reachability, only the first party needs to know the result. This difference significantly affects the communication cost. Under the semi-honest model [8], the state-of-the-art solution for privacy preserving set operations [14] 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. The complexity analysis in Section V-B will discuss how to calculate $O(dwm^d)$ and $O(2^w)$.

III. PROBLEM STATEMENT AND THREAD MODEL

A. Access Control Lists (ACLs)

We consider quantifying network reachability of the ACLs converted from different parties. Each ACL A is an ordered list of *rules* and each rule is composed of a *predicate* over d *fields*, $F_1, \cdots, F_d$ 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 $F_1, \cdots, F_d$ is a d-tuple $(p_1, \cdots, p_d)$ where each p_i is an element of the domain of field F_i, denoted as $D(F_i)$. A $\langle predicate \rangle$ specifies a set of packets over the d fields, $F_1 \in S_1 \wedge \cdots \wedge F_d \in S_d$, where each $S_i \subseteq D(F_i)$ and is specified as either a prefix or a range. A packet $(p_1, \cdots, p_d)$ *matches* a rule $F_1 \in S_1 \wedge \cdots \wedge F_d \in S_d \rightarrow \langle decision \rangle$ if and only if the condition $p_1 \in S_1 \wedge \cdots \wedge p_d \in S_d$ 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 [16]. Table I shows an example ACL. The format of these rules is based upon the format used in Cisco Access Control Lists.

Rule	Src. IP	Dest. IP	Src. Port	Dest. Port	Prot.	Action
r_1	123.24.*.*	192.168.0.1	*	80	TCP	accept
r_2	*	*	*	*	*	discard

TABLE I
AN EXAMPLE ACL

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 and other restriction information for filtering these packets. Multiple ACLs and other restriction information may belong to the same party. To convert them to a single ACL for one party, we first employ the existing network reachability approach, Quarnet [13], to convert them to 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 as well as other reachability restriction information 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 $A_1, A_2, ..., A_n$ 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 $M(A_1), \cdots, M(A_n)$, i.e., $M(A_1) \cap M(A_2) \cdots \cap M(A_n)$.

In our context, we aim to design a privacy preserving protocol which enables the first party P_1 to compute the intersection of n ACLs ($n \geq 2$), $M(A_1) \cap M(A_2) \cdots \cap M(A_n)$ 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 two 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) The source ACL A_1 is not allowed to compute the intersection of a subset of all ACLs along the given network path. Because A_1 can easily reveal some rules in one ACL A_i by three steps. First, compute $M(A_1) \cap \cdots \cap M(A_{i-1})$. Second, compute $M(A_1) \cap \cdots \cap M(A_i)$. Third, compute $M(A_1) \cap \cdots \cap M(A_{i-1}) - M(A_1) \cap \cdots \cap M(A_i)$.

Note that if one party does not want to involve in this process 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 who will not share with other parties the information that what packets can 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 [8]. 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.

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} with those from A_n, and finds out the multiple accepting rules whose union is equal to the intersection of $M(A_n)$ and $M(A_{n-1})$, $M(A_n) \cap M(A_{n-1})$. Then, P_{n-2} compares the accepting rules from ACL A_{n-2} with the resulting accepting rules in the first step, and finds out the multiple accepting rules whose union is equal to $M(A_n) \cap M(A_{n-1}) \cap M(A_{n-2})$. Repeat this step until P_1 finds out the multiple accepting rules whose union is equal to $M(A_1) \cap \cdots \cap M(A_n)$. Note that, the resulting accepting rules of each step are in an encrypted format which 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.

The basic problem of privacy-preserving network reachability is how to compute the intersection among multiple range 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 $[a, b] \cap [a', b']$ boils down 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 [15]. The idea of prefix membership verification is to convert the problem of checking whether a number is in

a range to the problem of checking whether two sets have common elements. Our scheme consists of six steps:

(1) The party P_i converts 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) The party P_i converts each range to a 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 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) 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***, ****}. It is easy to prove that $a' \in [a,b]$ if and only if $\mathcal{F}(a') \cap \mathcal{S}([a,b]) \neq \emptyset$.

(4) 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 [3]. 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 $\mathcal{N}(\mathcal{F}(a')) \cap \mathcal{N}(\mathcal{S}([a,b])) \neq \emptyset$.

(5) 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.*, [20], [21]) to do this checking. 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 $((x)_{K_i})_{K_j}$. In our scheme, to check whether $\mathcal{N}(\mathcal{F}(a')) \cap \mathcal{N}(\mathcal{S}([a,b])) \neq \emptyset$, P_i first encrypts numbers in $\mathcal{N}(\mathcal{S}([a,b]))$ with its private key K_i, 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. Second, P_j encrypts numbers in $\mathcal{N}(\mathcal{F}(a'))$ with K_j and 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).
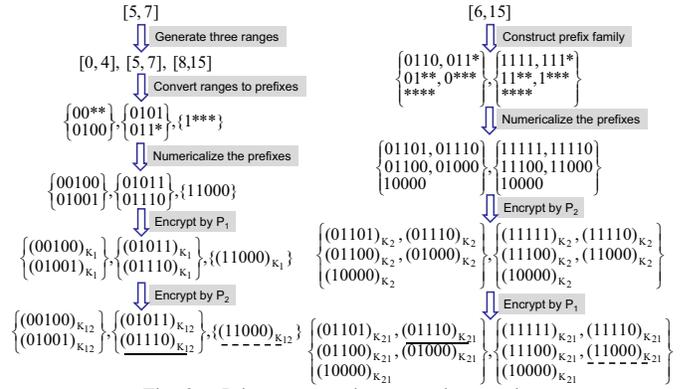

Fig. 3. Privacy-preserving range intersection

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 to an equivalent sequence of non-overlapping rules. Non-overlapping rules have an important property, that is, for any two non-overlapping 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 non-overlapping rule converted from A_i and the decision of this non-overlapping 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 the non-overlapping accepting rules because the union of the matching sets of these rules is equal to $M(A_i)$. The preprocessing of each A_i includes three steps:

(1) P_i converts its ACL A_i to an equivalent acyclic directed graph, called firewall decision diagram (FDD) [9]. An FDD construction algorithm, which converts an ACL to an equivalent FDD, is presented in [17]. Fig. 4(b) shows the FDD constructed from Fig. 4(a).
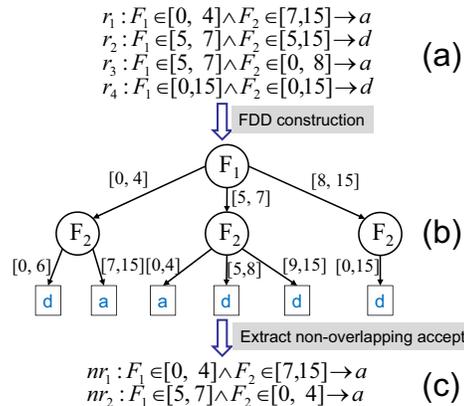

$r_1 : F_1 \in [0, 4] \wedge F_2 \in [7,15] \rightarrow a$
$r_2 : F_1 \in [5, 7] \wedge F_2 \in [5,15] \rightarrow d$
$r_3 : F_1 \in [5, 7] \wedge F_2 \in [0, 8] \rightarrow a$
$r_4 : F_1 \in [0,15] \wedge F_2 \in [0,15] \rightarrow d$ (a)

$nr_1 : F_1 \in [0, 4] \wedge F_2 \in [7,15] \rightarrow a$
$nr_2 : F_1 \in [5,7] \wedge F_2 \in [0, 4] \rightarrow a$ (c)

Fig. 4. The Conversion of A_1

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

After ACL preprocessing, the problem of privacy preserving quantification of network reachability boils down to the problem of privacy preserving range intersection, which is in fact the basic problem in this paper. Next, P_1 needs to compare

its accepting rules from A_1 with those from other $n-1$ ACLs. Without loss of generality, in the next two subsections, 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 non-overlapping accepting rules for ACL A_i ($1 \leq i \leq 3$). Obviously, the network reachability among these three 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 that how to compute these two rules in a privacy preserving manner.
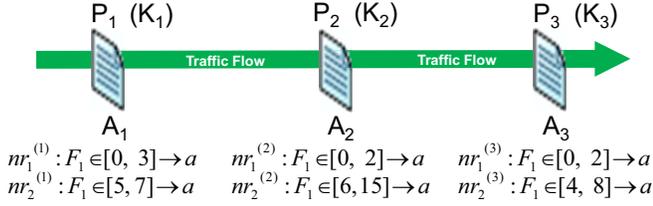

Fig. 5. The example three adjacent ACLs

C. ACL Encoding and Encryption

In the ACL encoding and encryption phase, all parties need to convert their non-overlapping 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. To achieve this purpose, each party first encodes its non-overlapping accepting rules and then encrypts each field of these rules. 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 $\mathcal{F}(a')$. Thus, in this phase, there are two different encoding and encryption methods for different ACLs. Assume that each party P_i ($1 \leq i \leq n$) has a private key K_i. Each party P_j ($1 \leq j \leq n-1$) encodes the non-overlapping accepting rules from A_j by converting each range to a set of prefixes and then encrypts each numericalized prefix by other parties P_j, $P_{j+1}, \cdots, P_n$. Let $\mathcal{H}$ denote the encoding function used by the party P_j ($1 \leq j \leq n-1$). Let $F_1 \in [a_1, b_1] \wedge \cdots \wedge F_d \in [a_d, b_d]$ denote the predicate of an accepting rule over d fields. The encoding and encryption result of this accepting rule for A_j is $\mathcal{H}_{K_{j\cdots n}}([a_1,b_1]) \wedge \cdots \wedge \mathcal{H}_{K_{j\cdots n}}([a_d,b_d])$. The party P_n encodes the non-overlapping accepting rules from A_n by converting each range to two prefix families and then encrypts each numericalized prefix by itself. Let $\mathcal{L}$ denote the encoding function used by the party P_n. Considering the above accepting rule, the result is $\mathcal{L}_{K_n}([a_1,b_1]) \wedge \cdots \wedge \mathcal{L}_{K_n}([a_d,b_d])$. We discuss the procedure of these two encoding and encryption methods in detail as follows.

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

(1) For each non-overlapping accepting rule $F_1 \in [a_1,b_1] \wedge \cdots \wedge F_d \in [a_d, b_d]$, P_j converts each range $[a_l, b_l]$ ($1 \leq l \leq d$) to three ranges $[min_l, a_l-1]$, $[a_l, b_l]$, $[b_l+1, max_l]$, where min_l and max_l are the minimum and maximum values for the l-th field, respectively. Fig. 6(b) shows the ranges generated from Fig. 6(a).

(2) P_j converts each range to a set of prefixes. Fig. 6(c) shows the prefixes generated from Fig. 6(b). That is, for the
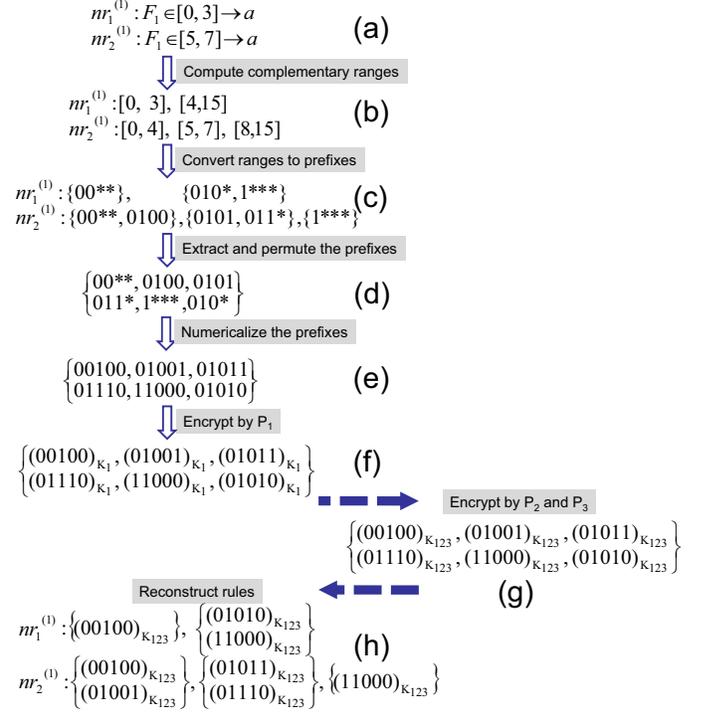

Fig. 6. Encoding and encryption of ACL A_1

three ranges converted from $[a_l, b_l]$, compute $\mathcal{S}([min_l, a_l-1])$, $\mathcal{S}([a_l,b_l])$, $\mathcal{S}([b_l+1, max_l])$.

(3) 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 non-overlapping accepting rules, because it is difficult to correlate the prefixes to their corresponding rules without the knowledge of the original ACL.

(4) P_j numericalizes and encrypts each prefix using K_j. Fig. 6(e) and 6(f) show the numericalized and encrypted prefixes.

(5) P_j sends the resulting prefixes to P_{j+1} which further encrypts them with its private key K_{j+1}. Then, P_{j+1} sends the result prefixes 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 P_1 and Fig. 6(g) shows the result after encrypting by P_2 and P_3.

(6) P_j reconstructs the non-overlapping 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 $\mathcal{H}_{K_{j\cdots n}}([a_l, b_l]) = (\mathcal{N}(\mathcal{S}(min_l, a_l - 1))_{K_{j\cdots n}}, \mathcal{N}(\mathcal{S}(a_l,b_l))_{K_{j\cdots n}}, \mathcal{N}(\mathcal{S}(b_l + 1, max_l))_{K_{j\cdots n}})$, 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.
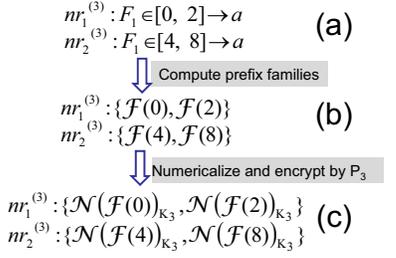
$nr_1^{(3)}: F_1 \in [0,2] \to a$
$nr_2^{(3)}: F_1 \in [4,8] \to a$ **(a)**

⇩ Compute prefix families

$nr_1^{(3)}: \{\mathcal{F}(0), \mathcal{F}(2)\}$
$nr_2^{(3)}: \{\mathcal{F}(4), \mathcal{F}(8)\}$ **(b)**

⇩ Numericalize and encrypt by P_3

$nr_1^{(3)}: \{\mathcal{N}(\mathcal{F}(0))_{K_3}, \mathcal{N}(\mathcal{F}(2))_{K_3}\}$
$nr_2^{(3)}: \{\mathcal{N}(\mathcal{F}(4))_{K_3}, \mathcal{N}(\mathcal{F}(8))_{K_3}\}$ **(c)**

Fig. 7. Encoding and encryption of ACL A_3

Encoding and Encryption of ACL A_n

(1) For each range $[a_l, b_l]$ of a non-overlapping rule, P_n generates two prefix families $\mathcal{F}(a_l)$ and $\mathcal{F}(b_l)$. Fig. 7(b) shows the result from Fig. 7(a).

(2) P_n numericalizes and encrypts the prefixes using its private key K_n. Fig. 7(c) shows the resulting prefixes.

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

After the first two phases, each party P_j $(1 \le j \le n-1)$ converts its ACL A_j to a sequence of encrypted non-overlapping accepting rules and P_n converts its ACL A_n to a sequence of encrypted numbers. Fig. 8(a) shows the encrypted non-overlapping rules converted from A_2 in Fig. 5 and Fig. 8(b) shows the encrypted numbers converted from A_3 in Fig. 5.

In the ACL comparison phase, we need to compare the n sequences of encrypted non-overlapping 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. This comparison includes 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 $\mathcal{L}_{K_n}([a_1', b_1']) \wedge \cdots \wedge \mathcal{L}_{K_n}([a_d', b_d'])$ 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 $\mathcal{L}_{K_{n(n-1)}}([a_1', b_1']) \wedge \cdots \wedge \mathcal{L}_{K_{n(n-1)}}([a_d', b_d'])$. Fig. 8(c) shows the encrypted result from Fig. 8(b).

(2) For each non-overlapping accepting rule nr from A_{n-1}, P_{n-1} computes $nr \cap nr'$. Let $\mathcal{H}_{K_{(n-1)n}}([a_1, b_1]) \wedge \cdots \wedge \mathcal{H}_{K_{(n-1)n}}([a_d, b_d])$ denote the encoded and encrypted result of nr. To compute $nr \cap nr'$, for each field l $(1 \le l \le d)$, P_{n-1} compares $\mathcal{H}_{K_{(n-1)n}}([a_l, b_l])$ with $\mathcal{L}_{K_{n(n-1)}}([a_l', b_l'])$ 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 above 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 $a_l \le a_l' \le b_l < b_l'$ holds and hence $T_l = [a_l', b_l]$. If for any T_l $(1 \le l \le d)$ the condition $T_l \ne \emptyset$ holds, then $nr \cap nr' = T_1 \wedge \cdots \wedge T_d$.
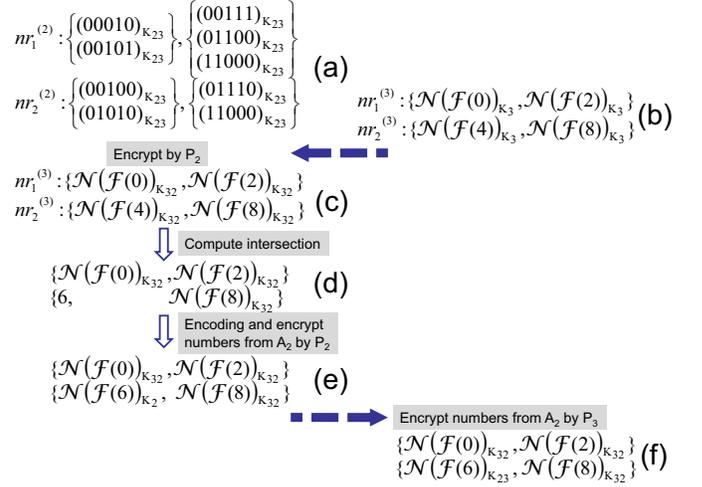
$nr_1^{(2)}: \{(00010)_{K_{23}}, (00101)_{K_{23}}\}, \{(00111)_{K_{23}}, (01100)_{K_{23}}, (11000)_{K_{23}}\}$
$nr_2^{(2)}: \{(00100)_{K_{23}}, (01010)_{K_{23}}\}, \{(01110)_{K_{23}}, (11000)_{K_{23}}\}$ **(a)**

$nr_1^{(3)}: \{\mathcal{N}(\mathcal{F}(0))_{K_3}, \mathcal{N}(\mathcal{F}(2))_{K_3}\}$
$nr_2^{(3)}: \{\mathcal{N}(\mathcal{F}(4))_{K_3}, \mathcal{N}(\mathcal{F}(8))_{K_3}\}$ **(b)**

Encrypt by P_2 ⬅

$nr_1^{(3)}: \{\mathcal{N}(\mathcal{F}(0))_{K_{32}}, \mathcal{N}(\mathcal{F}(2))_{K_{32}}\}$
$nr_2^{(3)}: \{\mathcal{N}(\mathcal{F}(4))_{K_{32}}, \mathcal{N}(\mathcal{F}(8))_{K_{32}}\}$ **(c)**

⇩ Compute intersection

$\{\mathcal{N}(\mathcal{F}(0))_{K_{32}}, \mathcal{N}(\mathcal{F}(2))_{K_{32}}\}$
$\{6, \mathcal{N}(\mathcal{F}(8))_{K_{32}}\}$ **(d)**

⇩ Encoding and encrypt numbers from A_2 by P_2

$\{\mathcal{N}(\mathcal{F}(0))_{K_{32}}, \mathcal{N}(\mathcal{F}(2))_{K_{32}}\}$
$\{\mathcal{N}(\mathcal{F}(6))_{K_2}, \mathcal{N}(\mathcal{F}(8))_{K_{32}}\}$ **(e)**

Encrypt numbers from A_2 by P_3 ➡

$\{\mathcal{N}(\mathcal{F}(0))_{K_{32}}, \mathcal{N}(\mathcal{F}(2))_{K_{32}}\}$
$\{\mathcal{N}(\mathcal{F}(6))_{K_{23}}, \mathcal{N}(\mathcal{F}(8))_{K_{32}}\}$ **(f)**

Fig. 8. Comparison of ACLs A_2 and A_3

Note that the party P_{n-1} cannot reveal a_l' and b_l' through this comparison because P_{n-1} doesn't 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 numbers from A_{n-1}'s non-overlapping accepting rules, which are not encrypted, *e.g.*, the number 6 in Fig. 8(d).

(3) To preserve the privacy of A_{n-1}, the party P_{n-1} encodes and encrypts the numbers from A_{n-1}'s non-overlapping accepting rules, and then sends the result to P_n. For example, for a number a_l, P_{n-1} computes $\mathcal{N}(\mathcal{F}(a_l))_{K_{(n-1)}}$. Fig. 8(e) shows the result after encoding and encrypting 6.

(4) To facilitate the next comparison with 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 non-overlapping accepting rules. Fig. 8(f) shows the encryption result.

$\{\mathcal{N}(\mathcal{F}(0))_{K_{321}}, \mathcal{N}(\mathcal{F}(2))_{K_{321}}\}$
$\{\mathcal{N}(\mathcal{F}(6))_{K_{231}}, \mathcal{N}(\mathcal{F}(7))_{K_{123}}\}$

⇩ Decrypt by P_3

$\{\mathcal{N}(\mathcal{F}(0))_{K_{21}}, \mathcal{N}(\mathcal{F}(2))_{K_{21}}\}$
$\{\mathcal{N}(\mathcal{F}(6))_{K_{21}}, \mathcal{N}(\mathcal{F}(7))_{K_{12}}\}$

Decrypt by P_2 ⬅

$\{\mathcal{N}(\mathcal{F}(0))_{K_1}, \mathcal{N}(\mathcal{F}(2))_{K_1}\}$
$\{\mathcal{N}(\mathcal{F}(6))_{K_1}, \mathcal{N}(\mathcal{F}(7))_{K_1}\}$

Decrypt by P_1 ⬅

$\{\mathcal{N}(\mathcal{F}(0)), \mathcal{N}(\mathcal{F}(2))\}$
$\{\mathcal{N}(\mathcal{F}(6)), \mathcal{N}(\mathcal{F}(7))\}$
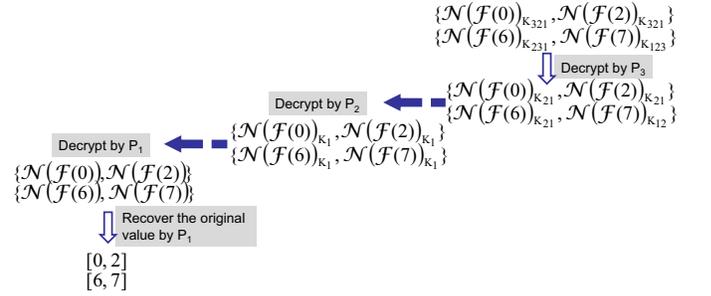
⇩ Recover the original value by P_1

[0, 2]
[6, 7]

Fig. 9. Decryption process of the comparison result

Repeat these four steps to further compare A_{n-2} with the result stored in P_n. After all parties finish the comparison, P_n has the comparison result. Let $\{\mathcal{N}(\mathcal{F}(a_l))_{K_{1 \cdots n}}, \mathcal{N}(\mathcal{F}(b_l))_{K_{1 \cdots n}}\}$ 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 to P_{n-1}. Then, P_{n-1} decrypts it with K_{n-1} and sends to P_{n-2}. Repeat this step until P_1 decrypts the rule. Finally, P_1 have the result $F_1 \in [a_1, b_1] \wedge \cdots \wedge F_d \in [a_d, b_d]$. 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. 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 non-overlapping 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 multi-party secure computation [1], [7] and the theorem in [4], 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 multi-party secure computation [1], [7], 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 [4] proves that P_{n-1}'s view of our protocol

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

cannot be distinguished from the simulation

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

where $z_{m+1}, \cdots, z_t$ 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 $b_1 b_2 \cdots b_k 1 0 \cdots 0$, the number of possible IP addresses that can generate such prefix is 2^{32-k}. Furthermore, after the comparison, P_{n-1} sends to P_n the comparison result which 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 $A_1, \cdots, A_{n-1}$ and then analyze the complexity of processing ACL A_n. The maximum number of non-overlapping rules generated from the FDD is $(2m_i-1)^d$ [17]. Each non-overlapping 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 [10]. Thus, the maximum number of prefixes generated from these non-overlapping 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 $P_i, \cdots, P_n$ is $\min(3d(2w-2)(2m_i-1)^d, 2^{w+1}) = \min(O(dwm_i^d), O(2^w))$, the space cost of P_i is $O(dwm_i^d)$, and the communication cost is $\min(O(dwm_i^d), O(2^w))$. For processing P_n, each interval of the non-overlapping 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 P_n, the computation, space, and communication costs of P_n is $O(dwm_n^d)$.

VI. OPTIMIZATION

To reduce the computation and communication costs, we use the divide-and-conquer strategy to divide 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 final reachability result. Let Q_i $(1 \leq i \leq n)$ denote the set of non-overlapping 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, $P_{i+1}, \cdots, P_n$. Thus, the number of encryptions for $Q_1, \cdots, Q_n$

is $n + (n-1) + \cdots + 1 = O(n^2)$. Similarly, the number of messages in this phase is $O(n^2)$. To reduce the 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 $A_{2\lfloor n/2 \rfloor - 1}, \cdots, A_n$. For example, 5 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 non-overlapping accepting rules. Therefore, we obtain $\lfloor n/2 \rfloor$ sets of non-overlapping accepting rules. Repeat these two steps until we obtain the network reachability for all n ACLs. Through this process, there are $\lfloor n/2 \rfloor + \lfloor n/2^2 \rfloor + \ldots + 1 = O(n)$ 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 $O(n)$.

VII. 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 [23]. 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 [20] 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 16GB 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. 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 $P_{i+1}, \cdots, P_n$. The computation cost of P_i is less than 2 seconds and the computation cost of P_j ($i + 1 \leq j \leq n$) is less than 1.5 seconds. 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 $P_{i+1}, \cdots, P_n$. 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 that 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 seconds. The average computation cost of other parties is less than 6 seconds. Similarly, for processing A_n, the computation cost of P_n is one-time offline cost, while the computation costs of other party 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 second, which is much less than the computation cost of processing ACLs. Because 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.
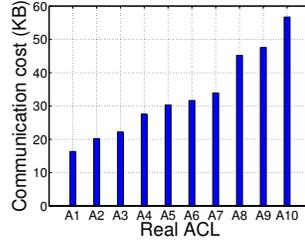
For processing synthetic ACL A_i ($1 \leq i \leq n-1$), Fig. 12(a) shows the computation cost of P_i and the average computation cost of parties $P_{i+1}, \cdots, P_n$ and Fig. 12(b) shows the average communication cost between P_j and P_{j+1} ($i \leq j \leq n$). The one-time offline computation cost (*i.e.*, the computation cost of P_i) is less than 400 seconds, and the online computation cost (*i.e.*, the average computation cost of other parties P_j) is less than 5 second. 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 seconds, and the online computation cost (*i.e.*, the average computation cost of other parties $P_1, \cdots, P_{n-1}$) is less than 25 seconds. The average communication cost between P_n and P_i is less than 2100 KB.
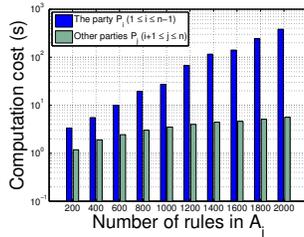
Fig. 14 shows the average comparison time for every two synthetic ACLs. The comparison time between any two synthetic ACLs is less than 5 seconds, which is much more efficient than processing ACLs.
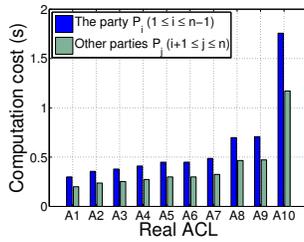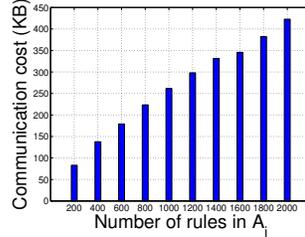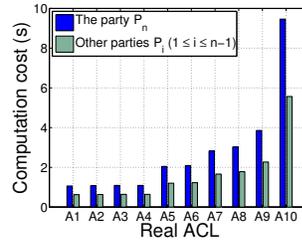
(a) Computation cost

(b) Communication cost

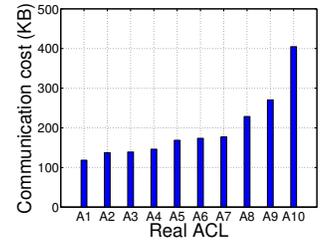Fig. 10. Comp. & comm. costs for processing real ACL A_i $(1 \le i \le n-1)$

(a) Computation cost
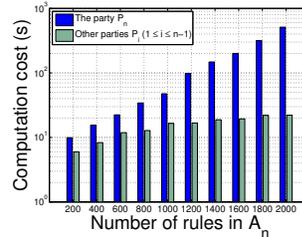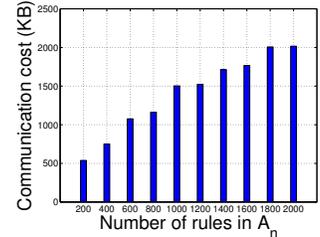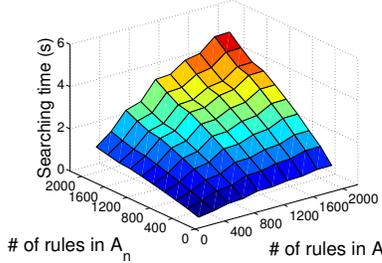

(b) Communication cost

Fig. 11. Comp. & comm. costs for processing real ACL A_n

(a) Ave. computation cost

(b) Ave. communication cost

Fig. 12. Comp. & comm. costs for processing synthetic ACL A_i $(1 \le i \le n-1)$

(a) Ave. computation cost

(b) Ave. communication cost

Fig. 13. Comp. & comm. costs for processing synthetic ACL A_n

Fig. 14. Comparison time of synthetic ACLs A_i and A_n

VIII. CONCLUSIONS

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 as well as low communication overhead, and is suitable for use in deployed networks.

Our future work will consider dynamic routing information and topological variations where links go down or new links get added to the network resulting in new paths for data propagation. In our protocol, we have considered the routing information partially by taking snapshots of the routing state and encoding it as an ACL rule. However, for a more finer-grained analysis, the instantaneous forwarding information state along with the local routing policies and other service level agreements also need to be considered.

Acknowledgment

REFERENCES

[1] R. Agrawal, A. Evfimievski, and R. Srikant. Information sharing across private databases. In *SIGMOD*, 2003.
[2] 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 *ICNP*, 2009.
[3] Y.-K. Chang. Fast binary and multiway prefix searches for packet forwarding. *Computer Networks*, 51(3), 2007.
[4] F. Chen, B. Bruhadeshwar, and A. X. Liu. A cross-domain privacy-preserving protocol for cooperative firewall optimization. In *INFOCOM*, 2011.
[5] E. A. Emerson. Temporal and modal logic. 1990.
[6] M. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *EUROCRYPT*, 2004.
[7] O. Goldreich. *Secure multi-party computations*. Working draft. Version 1.4, 2002.
[8] O. Goldreich. *Foundations of Cryptography: Volume II (Basic Applications)*. Cambridge University Press, 2004.
[9] M. G. Gouda and A. X. Liu. Firewall design: consistency, completeness and compactness. In *ICDCS*, 2004.
[10] P. Gupta and N. McKeown. Algorithms for packet classification. *IEEE Network*, 15(2), 2001.
[11] K. Ingols, R. Lippmann, and K. Piwowarski. Practical attack graph generation for network defense. In *ACSAC*, 2006.
[12] Z. Kerravala. As the value of enterprise networks escalates, so does the need for configuration management. Enterprise Computing & Networking, The Yankee Group Report, 2004.
[13] A. R. Khakpour and A. X. Liu. Quantifying and querying network reachability. In *ICDCS*, 2010.
[14] L. Kissner and D. Song. Privacy-preserving set operations. In *CRYPTO*, 2005.
[15] A. X. Liu and F. Chen. Collaborative enforcement of firewall policies in virtual private networks. In *PODC*, 2008.
[16] A. X. Liu and M. G. Gouda. Complete redundancy detection in firewalls. In *DBSec*, 2005.
[17] A. X. Liu and M. G. Gouda. Diverse firewall design. *IEEE TPDS*, 19(8), 2008.
[18] P. Matousek, J. Rab, O. Rysavy, and M. Sveda. A formal model for network-wide security analysis. In *Proc. IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, 2008.
[19] D. Oppenheimer, A. Ganapathi, and D. A. Patterson. Why do internet services fail, and what can be done about it? In *USENIX USITS*, 2003.
[20] S. C. Pohlig and M. E. Hellman. An improved algorithm for computing logarithms over gf(p) and its cryptographic significance. *IEEE TIT*, IT-24, 1978.
[21] D. K. H. D. R. Safford and D. L. Schales. Secure RPC authentication (SRA) for TELNET and FTP. Technical report, 1993.
[22] Y. Sang and H. Shen. Efficient and secure protocols for privacy-preserving set operations. *ACM TISSEC*, 13(9), 2009.
[23] S. Singh, F. Baboescu, G. Varghese, and J. Wang. Packet classification using multidimensional cutting. In *SIGCOMM*, 2003.
[24] 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 *SIGCOMM*, 2009.
[25] A. Wool. A quantitative study of firewall configuration errors. *IEEE Computer*, 37(6), 2004.
[26] G. G. Xie, J. Khan, D. A. Maltz, H. Zhang, A. Greenberg, G. Hjálmtýsson, and J. Rexford. On static reachability analysis of ip networks. In *INFOCOM*, 2005.