Cross-Domain Privacy-Preserving Cooperative Firewall Optimization

Fei Chen
Department of CSE
Michigan State University
East Lansing, MI
feichen@cse.msu.edu

Bezawada Bruhadeshwar
Center for Security Research
Institute of Information Technology
Hyderabad - 500032, India
bezawada@iiit.ac.in

Alex X. Liu
Department of CSE
Michigan State University
East Lansing, MI
alexliu@cse.msu.edu

***Abstract*—Firewalls have been widely deployed on the Internet for securing private networks. A firewall checks each incoming or outgoing packet to decide whether to accept or discard the packet based on its policy. Optimizing firewall policies is crucial for improving network performance. Prior work on firewall optimization focuses on either intra-firewall or inter-firewall optimization within one administrative domain where the privacy of firewall policies is not a concern. This paper explores inter-firewall optimization across administrative domains for the first time. The key technical challenge is that firewall policies cannot be shared across domains because a firewall policy contains confidential information and even potential security holes, which can be exploited by attackers. In this paper, we propose the first cross-domain privacy-preserving cooperative firewall policy optimization protocol. Specifically, for any two adjacent firewalls belonging to two different administrative domains, our protocol can identify in each firewall the rules that can be removed because of the other firewall. The optimization process involves cooperative computation between the two firewalls without any party disclosing its policy to the other. We implemented our protocol and conducted extensive experiments. The results on real firewall policies show that our protocol can remove as many as 49% of the rules in a firewall whereas the average is 19.4%. The communication cost is less than a few hundred KBs. Our protocol incurs no extra online packet processing overhead and the offline processing time is less than a few hundred seconds.**

I. Introduction

A. Background and Motivation

Firewalls are critical in securing private networks of businesses, institutions, and home networks. A firewall is often placed at the entrance between a private network and the external network so that it can check each incoming or outgoing packet and decide whether to accept or discard the packet based on its policy. A firewall policy is usually specified as a sequence of rules, called Access Control List (ACL), and each rule has a predicate over multiple packet header fields (*i.e.*, source IP, destination IP, source port, destination port, and protocol type) and a decision (*i.e.*, accept and discard) for the packets that match the predicate. The rules in a firewall policy typically follow the first-match semantics where the decision for a packet is the decision of the first rule that the packet matches in the policy. Each physical interface of a router/firewall is configured with two ACLs: one for filtering outgoing packets and the other one for filtering incoming packets. In this paper, we use the terms *firewalls*, *firewall policies*, and *ACLs* interchangeably.

The number of rules in a firewall significantly affects its throughput. Fig. 1 shows the result of the performance test of iptables conducted by HiPAC [1]. It shows that increasing the number of rules in a firewall policy dramatically reduces the firewall throughput. Unfortunately, with the explosive growth of services deployed on the Internet, firewall policies are growing rapidly in size. Thus, optimizing firewall policies is crucial for improving network performance.

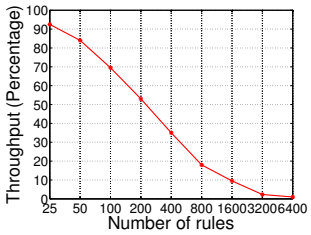


Fig. 1. The effect of the number of rules on the throughput with frame size 128 bytes in [1]

B. Limitation of Prior Work

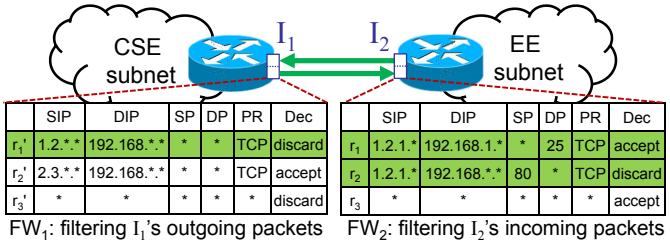
Prior work on firewall optimization focuses on either intra-firewall optimization [7], [15], [16], [17], [18], [19], [20], [21] or inter-firewall optimization [3], [27] within one administrative domain where the privacy of firewall policies is not a concern. Intra-firewall optimization means optimizing a single firewall. It is achieved by either removing redundant rules [15], [17] or rewriting rules [7], [16], [18], [19], [20], [21]. Prior work on inter-firewall optimization requires two firewall policies without any privacy protection, and thus can only be used within one administrative domain. However, in reality, it is common that two firewalls belong to different administrative domains where firewall policies cannot be shared with each other. Keeping firewall policies confidential is very important for two reasons. First, a firewall policy may have security holes that can be exploited by attackers. Quantitative studies have shown that most firewalls are misconfigured and have security holes [25]. Second, a firewall policy often contains private information, *e.g.*, the IP addresses of servers, which can be used by attackers to launch more precise and targeted attacks.

C. Cross-domain Inter-firewall Optimization

To our best knowledge, no prior work focuses on cross-domain privacy-preserving inter-firewall optimization. This paper represents the first step in exploring this unknown space. Specifically, we focus on removing inter-firewall policy redundancies in a privacy-preserving manner. Consider two adjacent firewalls 1 and 2 belonging to different administrative domains Net_1 and Net_2. Let FW_1 denote the policy on firewall 1's outgoing interface to firewall 2 and FW_2 denote the policy on firewall 2's incoming interface from firewall 1. For a rule r in FW_2, if all the packets that match r but do not match any rule above r in FW_2 are discarded by FW_1, rule r can be removed because such packets never come to FW_2. We call rule r an *inter-firewall redundant rule* with respect to FW_1. Note that FW_1 and FW_2 only filter the traffic from FW_1 to FW_2; the traffic from firewall 2's outgoing interface

to firewall 1's incoming interface is guarded by other two separate policies. For simplicity, we assume that FW_1 and FW_2 have no intra-firewall redundancy as such redundancy can be removed using the proposed solutions [15], [17].

Fig. 2 illustrates inter-firewall redundancy, where two adjacent routers belong to different administrative domains CSE and EE. The physical interfaces connecting two routers are denoted as I_1 and I_2, respectively. The rules of the two firewall policies FW_1 and FW_2, that are used to filter the traffic flowing from CSE to EE, are listed in two tables following the format used in Cisco Access Control Lists. Note that SIP, DIP, SP, DP, PR, and Dec denote source IP, destination IP, source port, destination port, protocol type, and decision, respectively. Clearly, all the packets that match r_1 and r_2 in FW_2 are discarded by r_1' in FW_1. Thus, r_1 and r_2 of FW_2 are inter-firewall redundant with respect to r_1' in FW_1.



	SIP	DIP	SP	DP	PR	Dec
r_1'	1.2.*.*	192.168.*.*	*	*	TCP	discard
r_2'	2.3.*.*	192.168.*.*	*	*	TCP	accept
r_3'	*	*	*	*	*	discard

FW_1: filtering I_1's outgoing packets

	SIP	DIP	SP	DP	PR	Dec
r_1	1.2.1.*	192.168.1.*	*	25	TCP	accept
r_2	1.2.1.*	192.168.*.*	80	*	TCP	discard
r_3	*	*	*	*	*	accept

FW_2: filtering I_2's incoming packets

Fig. 2. Example inter-firewall redundant rules

D. Technical Challenges and Our Approach

The key challenge is to design a protocol that allows two adjacent firewalls to identify the inter-firewall redundancy with respect to each other without knowing the policy of the other firewall. While intra-firewall redundancy removal is already complex [15], [17], inter-firewall redundancy removal with the privacy-preserving requirement is even harder. To determine whether a rule in FW_2 is inter-firewall redundant with respect to FW_1, Net_2 certainly needs some information about FW_1; yet, Net_2 cannot reveal FW_1 from such information.

A straightforward solution is to perform a privacy preserving comparison between two rules from two adjacent firewalls. Particularly, for each rule r in FW_2, this solution checks whether all possible packets that match rule r in FW_2 match a rule r' with the discard decision in FW_1. If rule r' exists, r is inter-firewall redundant with respect to r' in FW_1. However, because firewalls follow the first-match semantics and the rules in a firewall typically overlap, this solution is not only incorrect but also incomplete. Incorrect means that wrong redundant rules could be identified in FW_2. Suppose this solution identifies r as a redundant rule in FW_2 with respect to r_2' in FW_1. However, if some packets that match rule r also match rule r_1' (r_1' is above r_2') with the accept decision in FW_1, these packets will pass through FW_1 and then FW_2 needs to filter them with r. In this case, r is actually not redundant. Incomplete means that a portion of redundant rules could be identified in FW_2. If all possible packets that match rule r in FW_2 are discarded by not only one rule but multiple rules in FW_1, r is also redundant. However, the direct comparison solution cannot identify such redundancies.

Our basic idea is as follows. For each rule r in FW_2, we first compute a set of compact predicates representing the set of packets that match r but do not match the rules above r in FW_2. Then, for each predicate, we check whether all the packets that match the predicate are discarded by FW_1. If this condition holds for all the predicates computed from rule r, then rule r is redundant. To efficiently compute these predicates, we convert firewalls to firewall decision diagrams [17]. To allow the two firewalls to detect the redundant rules in FW_2 in a privacy-preserving manner, we develop a protocol so that two firewalls can detect such redundant rules without disclosing their policies to each other.

Our protocol applies to both stateful and stateless firewalls. The main difference between stateful and stateless firewalls is that stateful firewalls maintain a connection table. Upon receiving a packet, if it belongs to an established connection, it is automatically accepted without checking against the rules. Having the connection table or not does not affect our protocol.

E. Key Contributions

We make two key contributions. First, we propose a novel privacy-preserving protocol for detecting inter-firewall redundant rules in one firewall with respect to another firewall. This paper represents the first effort along this unexplored direction. Second, we implemented our protocol and conducted extensive experiments on both real and synthetic firewall policies. The results on real firewall policies show that our protocol can remove as many as 49% of the rules in a firewall whereas the average is 19.4%. The communication cost is less than a few hundred KBs. Our protocol incurs no extra online packet processing overhead and the offline processing time is less than a few hundred seconds.

II. Related Work

A. Firewall Redundancy Removal

Prior work on intra-firewall redundancy removal aims to detect redundant rules within a single firewall [12], [15], [17]. Gupta identified backward and forward redundant rules in a firewall [12]. Later, Liu *et al.* pointed out that the redundant rules identified by Gupta are incomplete, and proposed two methods for detecting all redundant rules [15], [17]. Prior work on inter-firewall redundancy removal requires the knowledge of two firewall policies and therefore is only applicable within one administrative domain [3], [27].

B. Collaborative Firewall Enforcement in VPN

Prior work on collaborative firewall enforcement in virtual private networks (VPNs) aims to enforce firewall policies over encrypted VPN tunnels without leaking the privacy of the remote network's policy [6], [13]. The problems of collaborative firewall enforcement in VPNs and privacy-preserving inter-firewall optimization are fundamentally different. First, their purposes are different. The former focuses on enforcing a firewall policy over VPN tunnels in a privacy-preserving manner, whereas the latter focuses on removing inter-firewall redundant rules without disclosing their policies to each other. Second, their requirements are different. The former preserves the privacy of the remote network's policy, whereas the latter preserves the privacy of both policies.

III. System and Threat Models

A. System Model

A firewall FW is an ordered list of $rules$. Each $rule$ has a $predicate$ over d $fields$ $F_1, \cdots, F_d$ and a $decision$ for the packets that match the predicate. Firewalls usually check five fields: source IP, destination IP, source port, destination port, and protocol type. The length of these fields are 32, 32, 16, 16, and 8 bits, respectively. A $predicate$ defines a set of packets

over the d fields, and is specified as $F_1 \in S_1 \wedge \cdots \wedge F_d \in S_d$ where each S_i is a subset of F_i's domain $D(F_i)$. A packet over the d fields $F_1, \cdots, F_d$ is a d-tuple $(p_1, \cdots, p_d)$ where each p_i $(1 \leq i \leq d)$ is an element of $D(F_i)$. 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 firewall decisions include accept, discard, accept with logging, and discard with logging. Without loss of generality, we only consider accept and discard in this paper. We call a rule with the accept decision an *accepting rule* and a rule with the discard decision a *discarding rule*. In a firewall policy, a packet may match multiple rules whose decisions are different. To resolve these conflicts, firewalls typically employ a first-match semantics where the decision for a packet p is the decision of the first rule that p matches. A *matching set* of r_i, $M(r_i)$, is the set of all possible packets that match the rule r_i [15]. A *resolving set* of r_i, $R(r_i)$, is the set of packets that match r_i but do not match any rule r_j above r_i $(j<i)$, and $R(r_i)$ is equal to $M(r_i) - M(r_1) \cup \cdots \cup M(r_{i-1})$ [15].

Based on above concepts, we define inter-firewall redundant rules. Given two adjacent firewalls FW_1 and FW_2, where the traffic flow is from FW_1 to FW_2, a rule r in FW_2 is inter-firewall redundant with respect to FW_1 if and only if all the packets in r's resolving set are discarded by FW_1.

B. Threat Model

We adopt the semi-honest model in [9]. For two adjacent firewalls, we assume that they are semi-honest, *i.e.*, each firewall follows our protocol correctly but each firewall may try to reveal the policy of the other firewall. The semi-honest model is realistic and well adopted [4], [26]. For example, this model is appropriate for large organizations that have many independent branches as well as for loosely connected alliances composed by multiple parties. While we are confident that all administrative domains follow mandate protocols, we may not guarantee that no corrupted employees are trying to reveal the private firewall policies of other parties. We leave investigation of privacy-preserving firewall optimization in the model with malicious participants to future work.

IV. Privacy-Preserving Inter-Firewall Redundancy Removal

In this section, we present our privacy-preserving protocol for detecting inter-firewall redundant rules in FW_2 with respect to FW_1. To do this, we first converts each firewall to an equivalent sequence of non-overlapping rules. Because for any non-overlapping rule nr, the matching set of nr is equal to the resolving set of nr, *i.e.*, $M(nr) = R(nr)$, we only need to compare non-overlapping rules generated from the two firewalls for detecting inter-firewall redundancy. Second, we divide this problem into two subproblems, *single-rule coverage redundancy detection* and *multi-rule coverage redundancy detection*, and then propose our privacy-preserving protocol for solving each subproblem. A rule nr is *covered* by one or multiple rules $nr'_{i_1} \cdots nr'_{i_k}$ $(k \geq 1)$ if and only if $M(nr) \subseteq M(nr'_{i_1}) \cup \cdots \cup M(nr'_{i_k})$. The first subproblem checks whether a non-overlapping rule nr in FW_2 is covered by a non-overlapping discarding rule nr' in FW_1, *i.e.*, $M(nr) \subseteq M(nr')$. The second subproblem checks whether a non-overlapping rule nr in FW_2 is covered by multiple non-overlapping discarding rules $nr'_{i_1} \cdots nr'_{i_k}$ $(k \geq 2)$ in FW_1, *i.e.*, $M(nr) \subseteq M(nr'_{i_1}) \cup \cdots \cup M(nr'_{i_k})$. Finally, after the redundant non-overlapping rules generated from FW_2 are identified, we map them back to the original rules in FW_2 and then identify the redundant ones.

The problem of checking whether $M(nr) \subseteq M(nr')$ boils down to the problem of checking whether one range $[a, b]$ in nr is contained by another range $[a', b']$ in nr', which further boils down to the problem of checking whether $a \in [a', b']$ and $b \in [a', b']$. Thus, we first describe the privacy-preserving protocol for comparing a number and a range.

A. Privacy-Preserving Range Comparison

To check whether a number a from FW_2 is in a range $[a', b']$ from FW_1, we use a method similar to the prefix membership verification scheme in [13]. The basic idea is to convert the problem of checking whether $a \in [a', b']$ to the problem of checking whether two sets converted from a and $[a', b']$ have a common element. Our method consists of four steps:

(1) *Prefix conversion*. It converts $[a', b']$ to a minimum number of prefixes, denoted as $\mathcal{S}([a', b'])$, whose union corresponds to $[a', b']$. For example, $\mathcal{S}([11, 15])$=$\{1011, 11**\}$.

(2) *Prefix family construction*. It generates all the prefixes that contains a including a itself. This set of prefixes is called the prefix family of a, denoted as $\mathcal{F}(a)$. Let k be the bit length of a. The prefix family $\mathcal{F}(a)$ consists of $k+1$ prefixes where the i-th prefix is obtained by replacing the last $i-1$ bits of a by $*$. For example, as the binary representation of 12 is 1100, we have $\mathcal{F}(12)$=$\{1100, 110*, 11**, 1***, ****\}$. It is not difficult to prove that $a \in [a', b']$ if and only if $\mathcal{F}(a) \cap \mathcal{S}([a', b']) \neq \emptyset$.

(3) *Prefix numericalization*. It converts the prefixes generated in the previous steps to concrete numbers such that we can encrypt them in the next step. We use the prefix numericalization scheme in [5]. Given a prefix $b_1 b_2 \cdots b_k * \cdots *$ of w bits, we first insert 1 after b_k. The bit 1 represents a separator between $b_1 b_2 \cdots b_k$ and $*\cdots*$. Then we replace every $*$ by 0. For example, 11** is converted to 11100. If the prefix does not contain *s, we place 1 at the end of the prefix. For example, 1100 is converted to 11001.

(4) *Comparison*. It checks whether $a \in [a', b']$ by checking whether $\mathcal{F}(a) \cap \mathcal{S}([a', b']) \neq \emptyset$, which boils down to checking whether two numbers are equal. We use commutative encryption to do this checking in a privacy-preserving manner. Given a number x and two encryption keys K_1 and K_2, a commutative encryption is a function that satisfies the property $((x)_{K_1})_{K_2} = ((x)_{K_2})_{K_1}$, *i.e.*, encryption with key K_1 first and then K_2 is equivalent to encryption with key K_2 first and then K_1. Example commutative encryption algorithms are the Pohlig-Hellman Exponentiation Cipher [22] and Secure RPC Authentication (SRA) [23]. In our scheme, each domain chooses a private key. Let K_1, K_2 be the private keys chosen by Net_1 and Net_2, respectively. To check whether number v_1 from Net_1 is equal to number v_2 from Net_2 without disclosing the value of each number to the other party, Net_1 can first encrypt v_1 using key K_1 and sends $(x)_{K_1}$ to Net_2; similarly, Net_2 can first encrypt v_2 using key K_2 and sends $(x)_{K_2}$ to Net_1. Then, each party checks whether $v_1 = v_2$ by checking whether $((v_1)_{K_1})_{K_2} = ((v_2)_{K_2})_{K_1}$. Note that $((v_1)_{K_1})_{K_2}$ =$((v_2)_{K_2})_{K_1}$ if and only if $v_1 = v_2$. If $v_1 \neq v_2$, neither party can learn anything about the numbers being compared. Fig. 3 illustrates the process of checking whether 12 from FW_2 is in the range $[11, 15]$ from FW_1.

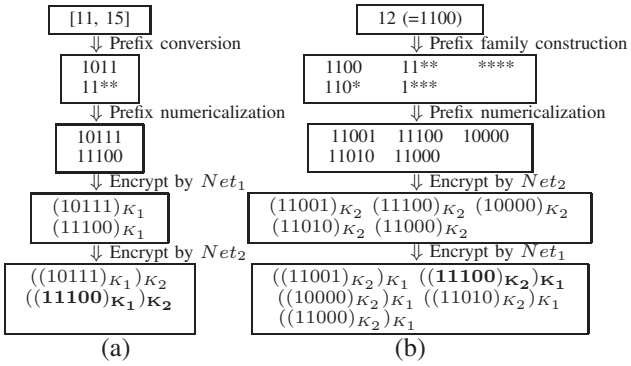


Fig. 3. Prefix membership verification

B. Processing Firewall FW_1

To detect the redundant rules in FW_2, Net_1 converts its firewall FW_1 to a set of non-overlapping rules. To preserve the privacy of FW_1, Net_1 first converts each range of a non-overlapping discarding rules from FW_1 to a set of prefixes. Second, Net_1 and Net_2 encrypt these prefixes using commutative encryption. The conversion of FW_1 includes nine steps:

(1) Net_1 first converts FW_1 to an equivalent firewall decision diagram (FDD) [10], [11]. An FDD for a firewall FW of a sequence of rules $\langle r_1, \cdots, r_n \rangle$ over fields $F_1, \cdots, F_d$ is an acyclic and directed graph that has five properties. (a) There is exactly one node that has no incoming edges. This node is called the *root*. The nodes that have no outgoing edge are called *terminal* nodes. (b) Each node v has a label, denoted $F(v)$. If v is a nonterminal node, then $F(v) \in \{F_1, \cdots, F_d\}$. If v is a terminal node, then $F(v)$ is a decision. (c) Each edge e, $u \rightarrow v$, is labeled with a non-empty set of integers, denoted $I(e)$, where $I(e)$ is a subset of the domain of u's label (*i.e.*, $I(e) \subseteq D(F(u))$). (d) The set of all outgoing edges of a node v, denoted $E(v)$, satisfies two conditions: (a) *consistency*: $I(e) \cap I(e') = \emptyset$ for any two distinct edges e and e' in $E(v)$; (b) *completeness*: $\bigcup_{e \in E(v)} I(e) = D(F(v))$. (e) A directed path from the root to a terminal node is called a *decision path*. No two nodes on a decision path have the same label. Each path in the FDD corresponds to a non-overlapping rule. A *full-length ordered FDD* is an FDD where in each decision path all fields appear exactly once and in the same order. For ease of presentation, we use the term "FDD" to denote "full-length ordered FDD". An FDD construction algorithm, which converts a firewall policy to an equivalent FDD, is presented in [14]. Fig. 4(b) shows the FDD constructed from Fig. 4(a).

(2) Net_1 reduces the FDD's size by merging isomorphic subgraphs. An FDD f is *reduced* if and only if it satisfies two conditions: (a) no two nodes in f are isomorphic; (b) no two nodes have more than one edge between them. Two nodes v and v' in an FDD are *isomorphic* if and only if v and v' satisfy one of the following two conditions: (a) both v and v' are terminal nodes with identical labels; (b) both v and v' are nonterminal nodes and there is a one-to-one correspondence between the outgoing edges of v and the outgoing edges of v' such that every two corresponding edges have identical labels and they both point to the same node. Fig. 4(c) shows the FDD reduced from the FDD in Fig. 4(b).

(3) Net_1 extracts non-overlapping discarding rules. We do not need to consider the non-overlapping accepting rules from FW_1 because the packets accepted by FW_1 are passed to FW_2. Note that a non-overlapping rule from FW_2 that is

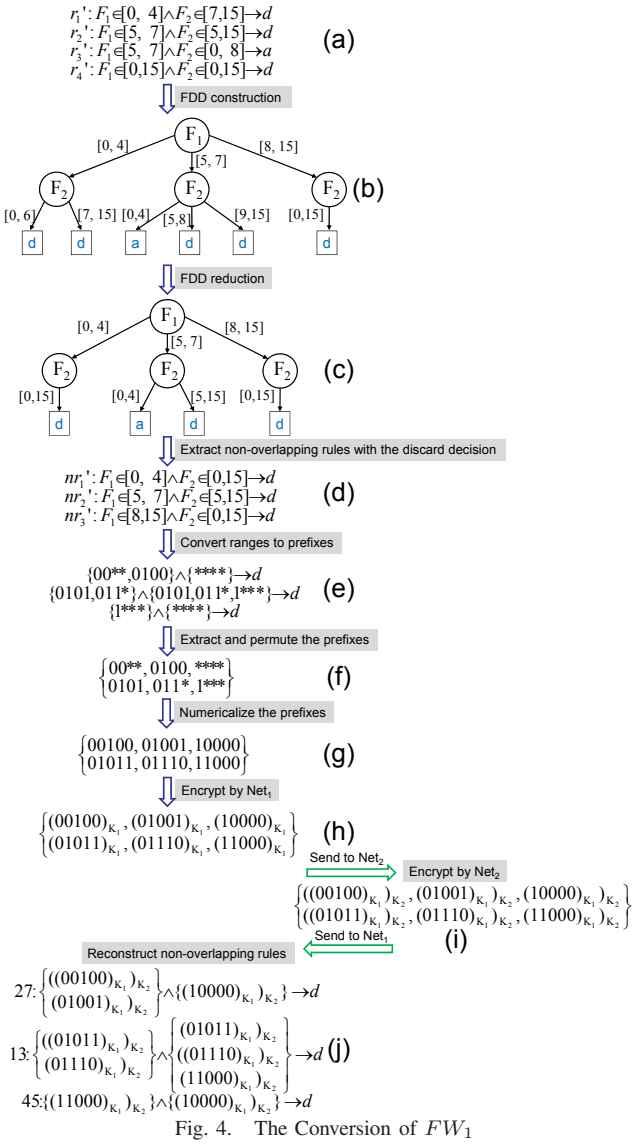


Fig. 4. The Conversion of FW_1

covered by these discarding rules from FW_1 is redundant. Fig. 4(d) shows the discarding non-overlapping rules extracted from the reduced FDD in Fig. 4(c).

(4) Net_1 converts each range to a set of prefixes. Fig. 4(e) shows the prefixes generated from Fig. 4(d).

(5) Net_1 unions all these prefix sets and then permutes the prefixes. Fig. 4(f) shows the resulting prefix set. Note that the resulting set does not include duplicate prefixes. The benefits are two-fold. In terms of efficiency, it avoids encrypting and sending duplicate prefixes for both parties, and hence, significantly reduces computation and communication costs. In terms of security, Net_2 cannot reconstruct the non-overlapping rules from FW_1, because Net_2 does not know which prefix belongs to which field of which rule. However, Net_1 knows such information and it can reconstruct these non-overlapping rules.

(6) Net_1 numericalizes and encrypts each prefix using K_1, and then sends to Net_2. Figures 4(g) and (h) show the numericalized and encrypted prefixes, respectively.

(7) Net_2 further encrypts these prefixes with K_2 and sends them back to Net_1 as shown in Fig. 4(i).

(8) Net_1 reconstructs its non-overlapping discarding rules

from the double encrypted prefixes because Net_1 knows which prefix belongs to which field of which rule.

(9) For each reconstructed non-overlapping rule, Net_1 assigns a distinct random index to it. These indices are used for Net_2 to identify the redundant non-overlapping rules from FW_2. For example, in Fig. 6(a), Net_1 assigns its three rules with three random indices: 27, 13, and 45. The detailed discussion is in Section IV-D.

C. Processing Firewall FW_2

In order to compare two firewalls in a privacy-preserving manner, Net_2 and Net_1 convert firewall FW_2 to d sets of double encrypted numbers, where d is the number of fields. The conversion of FW_2 includes five steps:

(1) Net_2 converts FW_2 to an equivalent all-match FDD. All-match FDDs differ from FDDs on terminal nodes. In an all-match FDD, each terminal node is labeled with a nonempty set of rule sequence numbers, whereas in an FDD each terminal node is labeled with a decision. For rule r_i $(1{\leq}i{\leq}n)$, we call i the *sequence number* of r_i. The set of rule sequence numbers labeled on a terminal node consists of the sequence numbers of all the rules that overlaps with the decision path ending with this terminal node. Given a decision path $\mathcal{P}$, $(v_1e_1\cdots v_de_dv_{d+1})$, the matching set of $\mathcal{P}$ is defined as the set of all packets that satisfy $F(v_1){\in}I(e_1){\wedge}\cdots{\wedge}F(v_d){\in}I(e_d)$. We use $M(\mathcal{P})$ to denote the matching set of $\mathcal{P}$. More formally, in an all-match FDD, for any decision path $\mathcal{P}$: $(v_1e_1\cdots v_de_dv_{d+1})$, if $M(\mathcal{P}){\cap}M(r_i){\neq}\emptyset$, then $M(\mathcal{P}){\subseteq}M(r_i)$ and $i{\in}F(v_{d+1})$. Fig. 4(b) shows the all-match FDD generated from Fig. 4(a). Considering the terminal node of the fourth path in Fig. 4(b), its label $\{2, 4\}$ means that $M(\mathcal{P}){\subseteq}M(r_2)$, $M(\mathcal{P}){\subseteq}M(r_4)$, $M(\mathcal{P}){\cap}M(r_1){=}\emptyset$, and $M(\mathcal{P}){\cap}M(r_3){=}\emptyset$. An all-match FDD not only represents a firewall in a non-overlapping fashion but also represents the overlapping relationship among rules. The reason of converting FW_2 to an all-match FDD is that later Net_2 needs the rule sequence numbers to identify inter-firewall redundant rules in FW_2.

(2) Net_2 extracts all non-overlapping rules from the all-match FDD. Fig. 5(c) shows the non-overlapping rules extracted from Fig. 5(b). For each range $[a, b]$ of a non-overlapping rule, Net_2 generates two prefix families $\mathcal{F}(a)$ and $\mathcal{F}(b)$. Fig. 5(d) shows the result from Fig. 5(c).

(3) For every field F_k, Net_2 unions all prefix families of all the non-overlapping rules into one prefix set and permutes the prefixes. Considering the first field F_1 in Fig. 5(d), Net_2 unions $\mathcal{F}(0)$, $\mathcal{F}(2)$, $\mathcal{F}(3)$, $\mathcal{F}(5)$, $\mathcal{F}(6)$ and $\mathcal{F}(15)$ to the first prefix set in Fig. 5(e). The benefits of this step are similar to those of Step (5) in FW_1's conversion. In terms of efficiency, it avoids encrypting and sending the duplicate prefixes for each field, and hence, significantly reduces the computation and communication costs. In terms of security, Net_1 cannot reconstruct the non-overlapping rules of FW_2, because it Net_1 does not know which prefix belongs to which rule in FW_2. However, Net_2 knows such information, which will be used to identify redundant non-overlapping rules later. Note that, the ordering of the fields cannot be permuted because Net_1 needs to perform comparison of the prefixes with only those prefixes from the corresponding fields.

(4) Net_2 numericalizes and encrypts the prefixes using its private K_2, and sends to Net_1. Fig. 5(f) shows the prefixes.

(5) Net_1 further encrypts these prefixes using its key K_1.

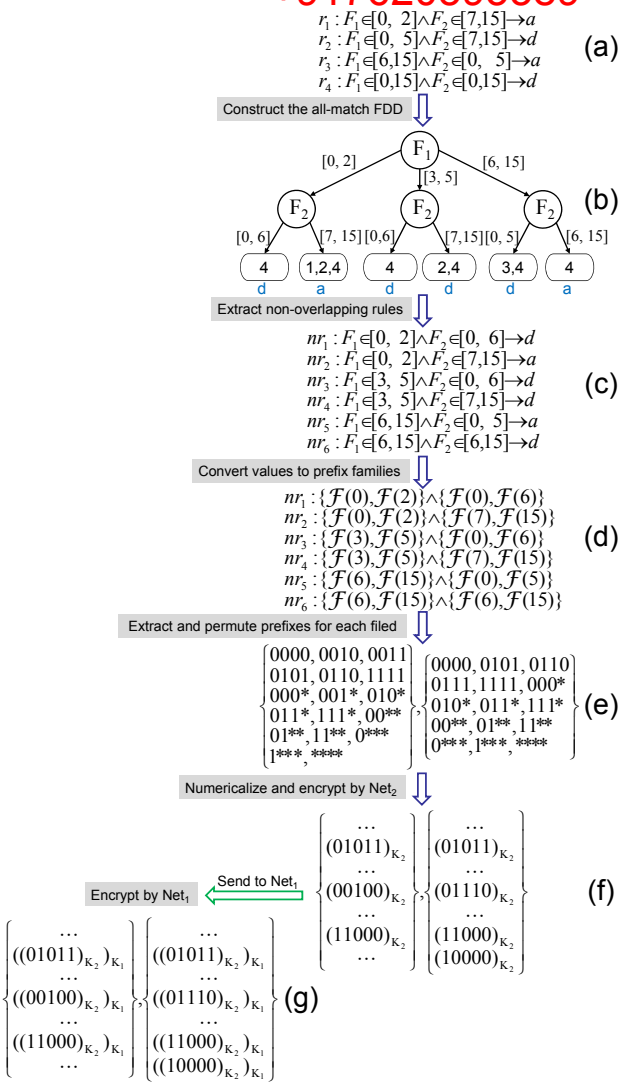


Fig. 5. The Conversion of FW_2

D. Single-Rule Coverage Redundancy Detection

After processing the two firewalls, Net_1 has a sequence of double encrypted non-overlapping rules obtained from FW_1 and d sets of double encrypted numbers obtained from FW_2. Let $(F_1{\in}\mathcal{S}_1){\wedge}\cdots{\wedge}(F_d{\in}\mathcal{S}_d){\rightarrow}discard$ denote a double encrypted rule, where $\mathcal{S}_i$ is a set of double encrypted numbers. Let $\mathbb{S}_1,\cdots,\mathbb{S}_d$ denote the d sets of double encrypted numbers from FW_2. Fig. 6(a) shows the double encrypted non-overlapping rules generated from Fig. 4 and Fig. 6(b) shows the double encrypted numbers generated from Fig. 5. For each field F_i $(1{\leq}i{\leq}d)$ and for each number a in $\mathbb{S}_i$, Net_1 checks whether there exists a double encrypted rule $(F_1{\in}\mathcal{S}_1){\wedge}\cdots{\wedge}(F_d{\in}\mathcal{S}_d){\rightarrow}discard$ such that $a \in \mathcal{S}_i$. If rule r_i satisfies this condition, then Net_1 associates the rule index i with a. As there maybe multiple rules that satisfy this condition, eventually Net_1 associates a set of rule indices with a. If no rule satisfies this condition, Net_1 associates an empty set with a. Considering the number $((01011)_{K_2})_{K_1}$, only the rule with index 13 contains it because $((01011)_{K_2})_{K_1}{=}((01011)_{K_1})_{K_2}$; thus, Net_1 associates $((01011)_{K_2})_{K_1}$ with $\{13\}$. Finally, Net_1 replaces each number in $\mathbb{S}_1,\cdots,\mathbb{S}_d$ with its corresponding set of rule indices, and sends them to Net_2.

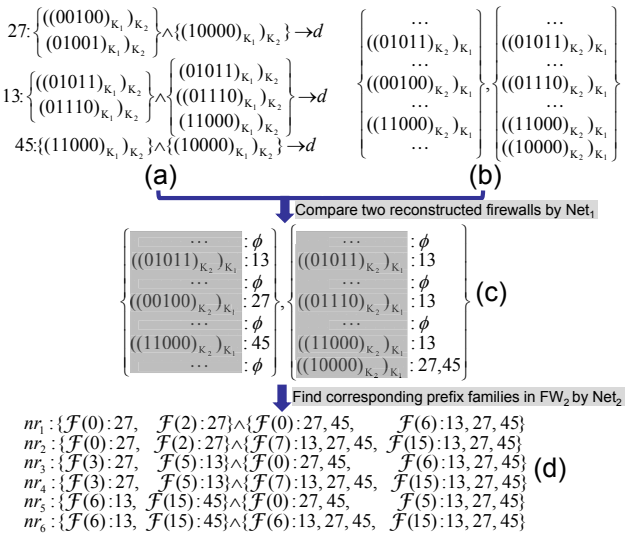


Fig. 6. Comparison of Two Firewalls

Upon receiving the sets from Net_1, for each prefix family, Net_2 finds the index of the rule that overlaps with the prefix family. For a non-overlapping rule nr from FW_2, if all its prefix families overlap with the same discarding rule nr' from FW_1, nr is covered by nr' and hence, nr is redundant. For example, in Fig. 6(d), nr_1 is redundant, because $\mathcal{F}(0)$, $\mathcal{F}(2)$, $\mathcal{F}(0)$ and $\mathcal{F}(6)$ overlap with rule 27 from FW_1. Similarly, nr_2 is redundant. Note that $\mathcal{F}(v)$:$j_1, ..., j_k$ denotes that $\mathcal{F}(v)$ overlaps with non-overlapping rules $j_1, ..., j_k$ from FW_1.

E. Multi-Rule Coverage Redundancy Detection

To detect multi-rule coverage redundancy, Net_2 keeps track of partial overlapping between its non-overlapping rules and those from FW_1. A non-overlapping rule nr from FW_2 is *partial overlapping* with a non-overlapping rule nr' from FW_1 if and only if $M(nr)\cap M(nr')\neq\emptyset$. Let $[a_1,b_1]\wedge\cdots\wedge[a_d,b_d]$ and $[a'_1,b'_1]\wedge\cdots\wedge[a'_d,b'_d]$ denote the predicates of nr and nr', respectively. The condition $M(nr)\cap M(nr')\neq\emptyset$ holds if and only if for any field F_k $(1\leq k\leq d)$, one of the three conditions holds: (1) $a_k\in[a'_k,b'_k]$; (2) $b_k\in[a'_k,b'_k]$; (3) $[a_k,b_k]\supset[a'_k,b'_k]$. The first two conditions are easy to check for Net_2 based on the privacy-preserving range comparison scheme in Section IV-A. Checking the third condition can be done by reversing the roles of Net_1 and Net_2 in the single-rule coverage redundancy detection. Considering the comparison result in Fig. 6(d), Net_2 knows that nr_4 is partial overlapping with two rules 27 and 13 in Fig. 5(j). Using this information, for each non-overlapping rule nr from FW_2 that overlaps with multiple non-overlapping rules from FW_1, Net_2 asks Net_1 to compute a new set of rules by combining these non-overlapping rules from FW_1. For example, after receiving the list with indices 27 and 13, Net_1 computes a new rule nr'_{new}: $F_1\in[0,7]\wedge F_2\in[5,15]\rightarrow d$ by combing these two rules, $F_1\in[0,4]\wedge F_2\in[0,15]\rightarrow d$ and $F_1\in[5,7]\wedge F_2\in[5,15]\rightarrow d$. If no new rule exists, nr is not multi-rule coverage redundant. Otherwise, both parties run the single-rule redundancy detection protocol again to check whether nr is covered by a new rule. If true, rule nr is multi-rule coverage redundant. For example, after comparing nr_4 with nr'_{new}, Net_2 can identify that nr_4 is redundant.

Note that, if the protocol is executed with the same keys in both single-rule and multi-rule redundancy detection, one firewall could learn considerable information of the other by comparing the new reconstructed firewall with that generated in the single-rule redundancy detection. To avoid this problem, Net_1 and Net_2 choose two new keys K'_1 and K'_2 in the multi-rule redundancy detection. Further note that in the multi-rule redundancy detection, the number of rules to be compared could be much smaller than those in the single-rule redundancy detection. Thus, the computation and communication costs are much less than those in the single-rule redundancy detection.

F. Identification and Removal of Redundant Rules

After single-rule and multi-rule coverage redundancy detection, Net_2 identifies the redundant non-overlapping rules in FW_2. Next, Net_2 needs to identify which original rules are inter-firewall redundant. As each path in the all-match FDD of FW_2 corresponds to a non-overlapping rule, we call the paths that correspond to the redundant non-overlapping rules *redundant paths* and the remaining paths *effective paths*. For example, in Fig. 7, the dashed paths are the redundant paths that correspond to nr_1, nr_2 and nr_4 in Fig. 5(c), respectively. Finally, Net_2 identifies redundant rules based on Theorem 4.1.

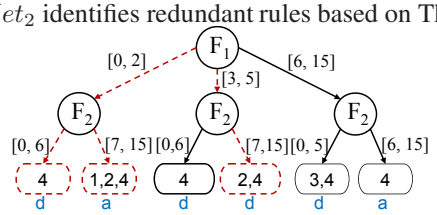


Fig. 7. Identification of redundant rules in FW_2

Theorem 4.1: Given firewall FW_2:$\langle r_1,\cdots,r_n\rangle$ with no intra-firewall redundancy and its all-match FDD, rule r_i is inter-firewall redundant with respect to FW_1 if and only if two conditions hold: (1) there is a redundant path whose terminal node contains sequence number i; (2) there is no effective path whose terminal node contains i as the smallest element.

Proof: Let $\{\mathcal{P}_1,\cdots,\mathcal{P}_m\}$ denote all paths in FW_2's all-match FDD. According to the theorems in [15], [17], the resolving set of each rule r_i $(1\leq i\leq n)$ in firewall FW_2 satisfies the condition $R(r_i)=\cup_{k=1}^{k=t}M(\mathcal{P}_{j_k})$ $(1\leq j_k\leq m)$, where $\mathcal{P}_{j_1},\cdots,\mathcal{P}_{j_t}$ are all the paths whose terminal nodes contain i as the smallest element. Based on the definition of inter-firewall redundant rules in Section III-A, rule r_i is inter-firewall redundant if and only if all the packets in $\cup_{k=1}^{k=t}M(\mathcal{P}_{j_k})$ are discarded by FW_1. Thus, each path $\mathcal{P}_{j_k}$ $(1\leq k\leq t)$ is a redundant path. In other words, all the paths $\mathcal{P}_{j_1},\cdots,\mathcal{P}_{j_t}$ whose terminal nodes contain i as the smallest element are redundant paths. ■

Considering redundant paths in Fig. 7, Net_2 identifies that r_1 and r_2 are inter-firewall redundant with respect to FW_1.

Theorem 4.2: The privacy-preserving inter-firewall redundancy removal protocol is a *complete* inter-firewall redundancy removal scheme.

Proof: Suppose that our proposed protocol is not a complete scheme and hence it cannot detect all inter-firewall redundant rules in FW_2. Assume that rule r_i is inter-firewall redundant in FW_2 but it is not detected by our protocol. According to Theorem 4.1, $R(r_i)=\cup_{k=1}^{k=t}M(\mathcal{P}_{j_k})$ $(1\leq j_k\leq m)$ and $\mathcal{P}_{j_1},\cdots,\mathcal{P}_{j_t}$ are redundant paths in FW_2's all-match FDD. Thus, some paths in $\{\mathcal{P}_{j_1},\cdots,\mathcal{P}_{j_t}\}$ cannot be identified as redundant paths by our protocol. This conclusion violates the fact that our protocol can identify all the redundant paths in FW_2's all-match FDD. ■

V. Firewall Update After Optimization

If FW_1 or FW_2 changes after inter-firewall optimization, the inter-firewall redundant rules identified by the optimization may not be inter-firewall redundant anymore. In this section, we discuss our solution to address firewall update. There are five possible cases under this scenario.

(1) Net_1 changes the decisions of some rules from discard to accept in FW_1. In this case, Net_1 needs to notify Net_2 that which non-overlapping rules (indices of these rules) from FW_1 are changed. Using this information, Net_2 checks if there were any rules in FW_2 that were removed due to these rules, and then adds the affected rules back into FW_2.

(2) Net_1 changes the decisions of some rules from accept to discard in FW_1. In this case, Net_2 can run our cooperative optimization protocol again to identify more inter-firewall redundant rules in FW_2.

(3) Net_2 changes the decisions of some rules in FW_2. In this case, neither party needs to take actions because the inter-firewall redundancy detection does not consider the decisions of the rules in FW_2.

(4) Net_1 adds or removes some rules in FW_1. In this case, since the resolving sets of some rules in FW_1 may change, a rule in FW_2 that used to be inter-firewall redundant maybe not redundant anymore. It is important for Net_2 to run our cooperative optimization protocol again.

(5) Net_2 adds or removes some rules in FW_2. Similar to the fourth case, since the resolving sets of some rules in FW_2 may change, it is important for Net_2 to run our protocol again.

VI. Security and Complexity Analysis

A. Security Analysis

To analyze the security of our protocol, we first describe the commutative encryption and its properties. Let Key denote a set of private keys and Dom denote a finite domain. A commutative encryption f is a computable function f:$Key\times Dom\rightarrow Dom$ that satisfies the following four properties. (1) Secrecy: For any x and key K, given $(x)_K$, it is computationally infeasible to compute K. (2) Commutativity: For any x, K_1, and K_2, we have $((x)_{K_1})_{K_2} = ((x)_{K_2})_{K_1}$. (3) For any x, y, and K, if $x \neq y$, we have $(x)_K \neq (y)_K$. (4) The distribution of $(x)_K$ is indistinguishable from the distribution of x. Note that, for ease of presentation, in the rest of this section, any x, y, or z is an element of Dom, any K, K_1, or K_2 is an element of Key, and $(x)_K$ denotes $f(x, K)$.

In the conversion of FW_1, for each non-overlapping rule nr' from FW_1, let $V_{F_j}(nr')$ denote the prefix set for the field F_j, *e.g.*, in Fig. 4(e), $V_{F_1}(nr'_1)$ denotes {00**, 0100}. In the conversion of FW_2, let T_{F_j} denote the prefix set for the field F_j after removing duplicate prefixes, *e.g.*, in Fig. 5(e), T_{F_1} denotes the first prefix set. Our cooperative optimization protocol essentially compares $V_{F_j}(nr')$ and T_{F_j} in a privacy-preserving manner. We have the following theorem.

Theorem 6.1: If both parties Net_1 and Net_2 are semi-honest, after comparing two sets $V_{F_j}(nr')$ and T_{F_j} using our protocol, Net_1 learns only the size $|T_{F_j}|$ and the intersection $V_{F_j}(nr') \cap T_{F_j}$, and Net_2 learns only the size $|V_{F_j}(nr')|$ and the intersection $V_{F_j}(nr') \cap T_{F_j}$.

Proof: According to the theorems in multi-party secure computation [2], [8], if we can prove that the distribution of the Net_1's view of our protocol cannot be distinguished from a simulation that uses only $V_{F_j}(nr')$, $V_{F_j}(nr') \cap T_{F_j}$, and $|T_{F_j}|$, then Net_1 cannot learn anything else except $V_{F_j}(nr') \cap T_{F_j}$ and $|T_{F_j}|$. Note that Net_1's view of our protocol is the information that Net_1 gains from FW_2.

Without loss of generality, we only prove that Net_1 learns only the size $|T_{F_j}|$ and the intersection $V_{F_j}(nr') \cap T_{F_j}$. The simulator for Net_1 uses key K_1 to create a set from $V_{F_j}(nr')$ and $V_{F_j}(nr') \cap T_{F_j}$ as follows

$$Y_S = \{\underbrace{(x_1)_{K_1}, \cdots, (x_m)_{K_1}}_{x_i \in V_{F_j}(nr') \cap T_{F_j}}, \quad \underbrace{z_{m+1}, \cdots, z_n}_{n-m=|V_{F_j}(nr') - T_{F_j}|} \}$$

where $z_{m+1}, \cdots, z_n$ are random values generated by the simulator and they are uniformly distributed in the finite domain Dom. According to the theorems in [2], Net_1 cannot distinguish the distribution of Y_S's elements from that in

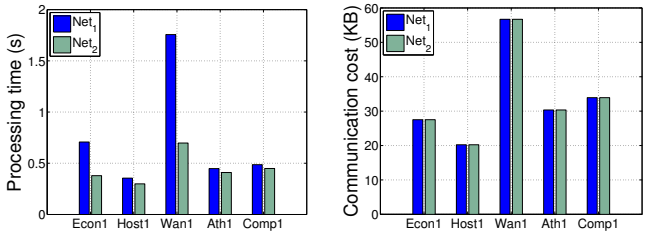
$$Y_R = \{\underbrace{(x_1)_{K_1}, \cdots, (x_m)_{K_1}}_{x_i \in V_{F_j}(nr') \cap T_{F_j}}, \underbrace{(x_{m+1})_{K_1}, \cdots, (x_n)_{K_1}}_{x_i \in V_{F_j}(nr') - T_{F_j}}\}$$

The Net_1's view of our protocol corresponds to Y_R. Therefore, the distribution of the Net_1's view of our protocol cannot be distinguished from this simulation. ■

Next, we analyze the information learned by Net_1 and Net_2. After implementing our protocol, Net_1 knows the converted firewalls of FW_1 and FW_2, *e.g.*, Fig. 4(j) and Fig. 5(g), and Net_2 knows the comparison result, *e.g.*, Fig. 6(d). On Net_1 side, for each field F_j ($1 \leq j \leq d$), it knows only $|T_{F_j}|$ and $V_{F_j}(nr') \cap T_{F_j}$, and it cannot reveal the rules of FW_2 for two reasons. First, in $V_{F_j}(nr') \cap T_{F_j}$, a numericalized prefix can be generated from many different numbers. For example, a prefix of IP addresses (32 bits) $b_1 b_2 \cdots b_k * \cdots *$ can be generated from 2^{32-k} different IP addresses. Second, even if Net_1 finds the number for a prefix in $V_{F_j}(nr') \cap T_{F_j}$, Net_1 doesn't know which rule in FW_2 contains that number. On Net_2 side, it only knows that the prefix x in $\mathcal{F}(x)$ belongs to which non-overlapping rules in FW_1. But such information is not enough to reveal the rules in FW_1.

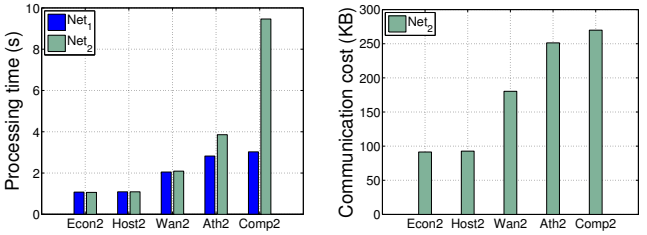
B. Complexity Analysis

Let n_1 and n_2 be the number of rules in two adjacent firewalls FW_1 and FW_2, respectively, and d be the number of fields in both firewalls. For simplicity, we assume that the numbers in different fields have the same length, say w bits. We first analyze the computation, space, and communication costs for the conversion of FW_1. Based on the theorem in [14], the maximum number of non-overlapping rules generated from the FDD is $(2n_1-1)^d$. Each non-overlapping rule consists of d w-bit intervals and each interval can be converted to at most $2w-2$ prefixes. Thus, the maximum number of prefixes generated from these non-overlapping rules is $d(2w-2)(2n_1-1)^d$. Note that the total number of prefixes cannot exceed 2^{w+1} because Net_1 puts all prefixes into one set. Thus, the computation cost of encryption by Net_1 is $\min(d(2w-2)(2n_1-1)^d, 2^{w+1})$. Therefore, for the conversion of FW_1, the computation cost of Net_1 is $\min(O(dwn_1^d), O(n_1^d + 2^w))$, the space cost of Net_1 is $O(dwn_1^d)$, the communication cost is $\min(O(dwn_1^d), O(2^w))$, and the computation cost of Net_2 is $\min(O(dwn_1^d), O(2^w))$. Similarly, for the conversion of FW_2, the computation cost of Net_2 is $\min(O(dwn_2^d), O(n_2^d + 2^w d))$, the space cost of Net_2 is $O(dwn_2^d)$, the communication cost is $\min(O(dwn_2^d), O(2^w d))$, and the computation cost of Net_1 is $\min(O(dwn_2^d), O(2^w d))$.



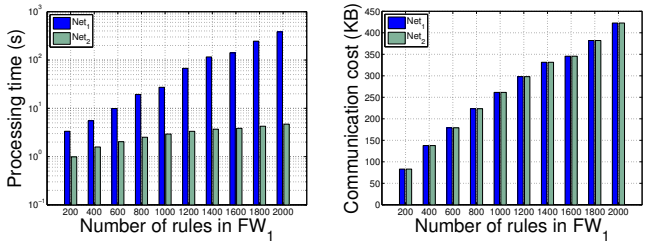
(a) Processing time (b) Communication cost

Fig. 8. Processing FW_1 on real firewalls



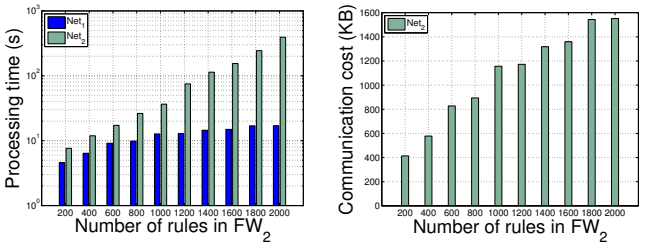
(a) Processing time (b) Communication cost

Fig. 9. Processing FW_2 on real firewalls



(a) Ave. processing time (b) Ave. communication cost

Fig. 10. Processing FW_1 on synthetic firewalls



(a) Ave. processing time (b) Ave. communication cost

Fig. 11. Processing FW_2 on synthetic firewalls

VII. Experimental Results

We evaluate the effectiveness of our protocol on real firewalls and evaluate the efficiency of our protocol on both real and synthetic firewalls. We implemented our protocol using Java 1.6.0. Our experiments were carried out on a PC running Linux with 2 Intel Xeon cores and 16GB of memory.

A. Evaluation Setup

We conducted experiments over five groups of two real adjacent firewalls. Each firewall examines five fields, source IP, destination IP, source port, destination port, and protocol. The number of rules ranges from dozens to thousands. In implementing the commutative encryption, we used the Pohlig-Hellman algorithm [22] with a 1024-bit prime modulus and 160-bit encryption keys. To evaluate the effectiveness, we conducted our experiments over these five groups of adjacent firewalls. To evaluate the efficiency, for two firewalls in each group, we measured the processing time, the comparison time, and the communication cost of both parties.

Due to security concerns, it is difficult to obtain a large number of real adjacent firewalls. To further evaluate the efficiency, we generated a large number of synthetic firewalls based on Singh *et al.* 's method [24]. The synthetic firewalls also examine the same five fields as real firewalls. The number of rules in the synthetic firewalls ranges from 200 to 2000, and for each number, we generated 10 synthetic firewalls. To measure the efficiency, we first processed each synthetic firewall as FW_1 and then measured the processing time and communication cost of two parties. Second, we processed each synthetic firewall as FW_2 and measured the processing time and communication cost. Third, we measured the comparison time for every two synthetic firewalls. We did not evaluate the effectiveness of our protocol on synthetic firewalls because they are generated randomly and independently without considering whether two firewalls are adjacent or not.

B. Methodology

In this section, we define the metrics to measure the effectiveness of our protocol. Given our firewall optimization algorithm A, and two adjacent firewalls FW_1 and FW_2, we use $A(FW_1, FW_2)$ to denote a set of inter-firewall redundant rules in FW_2. Let $|FW|$ denote the number of rules in FW and $|A(FW_1, FW_2)|$ denote the number of inter-firewall redundant rules in FW_2. To evaluate the effectiveness, we define a *redundancy ratio* $\beta(A(FW_1, FW_2)) = \frac{|A(FW_1,FW_2)|}{|FW_2|}$. This ratio $\beta(A(FW_1, FW_2))$ measures what percentage of rules are inter-firewall redundant in FW_2.

C. Effectiveness and Efficiency on Real Firewall Policies

Table I shows the redundancy ratios for 5 real firewall groups. Column 1 shows the names of five real firewall groups. Columns 2 and 3 show the names of firewalls FW_1 and FW_2, respectively. Column 4 shows the number of rules in firewall FW_2. Fig. 8 shows the processing time and communication cost of two parties Net_1 and Net_2 when processing FW_1; Fig. 9 shows the processing time and communication cost of the two parties when processing FW_2. Fig. 12 shows the comparison time of each group.

Group	FW_1	FW_2	$\|FW_2\|$	redundancy ratio
Econ	Econ1	Econ2	129	17.1%
Host	Host1	Host2	139	49.6%
Wan	Wan1	Wan2	511	1.0%
Ath	Ath1	Ath2	1308	14.4%
Comp	Comp1	Comp2	3928	14.7%

TABLE I
Redundancy ratios for 5 real firewall groups

Our protocol achieves significant redundancy ratio on four real firewall groups. For 5 real firewall groups, our protocol achieves an average redundancy ratio of 19.4%. Particularly, for the firewall group Host, our protocol achieves 49.6% redundancy ratio, which implies that almost half of rules in Host2 are inter-firewall redundant rules. For firewall groups Econ, Ath, and Comp, our protocol achieves 14.4%-17.1% redundancy ratios, which implies that about 15% of rules in FW_2 are redundant in these three groups. Only for one firewall group Wan, our protocol achieves 1.0% redundancy ratio. From these results, we observed that most adjacent real firewalls, have many inter-firewall redundant rules. Thus, our protocol could effectively remove inter-firewall redundant rules and significantly improve the network performance.

Our protocol is efficient for processing and comparing two real firewalls. When processing FW_1 in the 5 real firewall

groups, the processing time of Net_1 is less than 2 seconds and the processing time of Net_2 is less than 1 second. When processing FW_2 in those real firewall groups, the processing time of Net_1 is less than 4 seconds and the processing time of Net_2 is less than 10 seconds. The comparison time of two firewalls is less than 0.07 seconds. The total processing time of two parties is less than 15 seconds, which demonstrates the efficiency of our protocol.

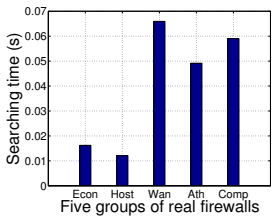


Fig. 12. Comparing two real firewalls

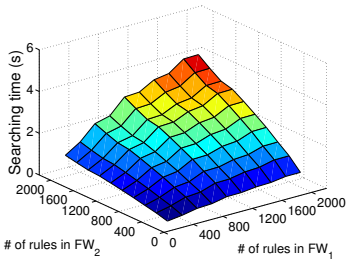


Fig. 13. Comparing two synthetic firewalls

Our protocol is efficient for the communication cost between two parties. When processing firewall FW_1 in the 5 real firewall groups, the communication cost from Net_1 to Net_2 and that from Net_2 to Net_1 are less than 60 KB. Note that the communication cost from Net_1 to Net_2 and that from Net_2 to Net_1 are the same because Net_1 and Net_2 encrypt the same number of values and the encrypted values have the same length, *i.e.*, 1024 bits in our experiments. When processing FW_2 in those real firewall groups, the communication cost from Net_2 to Net_1 is less than 300 KB. The total communication cost between two parties is less than 500 KB, which can be sent through the current network (*e.g.*, DSL network) around 10 seconds.

D. Efficiency on Synthetic Firewall Policies

For the synthetic firewalls, Fig. 10 and Fig. 11 show the average processing time and communication cost of two parties Net_1 and Net_2 for processing FW_1 and FW_2, respectively. Fig. 13 shows the average comparison time for every two synthetic firewalls. Note that the vertical axis of two figures 10(a) and 11(a) are in a logarithmic scale.

Our protocol is efficient for processing and comparing two synthetic firewalls. When processing the synthetic firewalls as FW_1, the processing time of Net_1 is less than 400 seconds and the processing time of Net_2 is less than 5 seconds. When processing the synthetic firewalls as FW_2, the processing time of Net_1 is less than 400 seconds and the processing time of Net_2 is less than 20 seconds. The comparison time of two synthetic firewalls is less than 4 seconds.

Our protocol is efficient for the communication cost between two synthetic firewalls. When processing the synthetic firewalls as FW_1, the communication cost from Net_1 to Net_2 and that from Net_2 to Net_1 grow linearly with the number of rules in FW_1, and both costs are less than 450 KB. Similarly, when processing synthetic firewalls as FW_2, the communication cost from Net_2 to Net_1 grows linearly with the number of rules in FW_2, and the communication cost from Net_2 to Net_1 is less than 1600 KB.

VIII. Conclusions and Future Work

In this work, we identified an important problem, cross-domain privacy-preserving inter-firewall redundancy detection. We propose a novel privacy-preserving protocol for detecting such redundancy. We implemented our protocol in Java and conducted extensive evaluation. The results on real firewall policies show that our protocol can remove as many as 49% of the rules in a firewall whereas the average is 19.4%.

There are many special cases that could be explored based on our current protocol. For example, there may be hosts or Network Address Translation (NAT) devices between two adjacent firewalls. Our current protocol cannot be directly applied to such cases. Extending our protocol to these cases could be an interesting topic, and requires further investigation.

Acknowledgment

This material is based in part upon work supported by National Science Foundation under Grant Numbers CNS-1017598.

References

[1] Firewall throughput test, www.hipac.org/performance_tests/results.html.
[2] R. Agrawal, A. Evfimievski, and R. Srikant. Information sharing across private databases. In *ACM SIGMOD*, pages 86–97, 2003.
[3] E. Al-Shaer and H. Hamed. Discovery of policy anomalies in distributed firewalls. In *IEEE INFOCOM*, pages 2605–2616, 2004.
[4] J. Brickell and V. Shmatikov. Privacy-preserving graph algorithms in the semi-honest model. In *ASIACRYPT*, pages 236–252, 2010.
[5] Y.-K. Chang. Fast binary and multiway prefix searches for packet forwarding. *Computer Networks*, 51(3):588–605, 2007.
[6] J. Cheng, H. Yang, S. H. Wong, and S. Lu. Design and implementation of cross-domain cooperative firewall. In *IEEE ICNP*, 2007.
[7] Q. Dong, S. Banerjee, J. Wang, D. Agrawal, and A. Shukla. Packet classifiers in ternary CAMs can be smaller. In *ACM SIGMETRICS*, pages 311–322, 2006.
[8] O. Goldreich. *Secure multi-party computations*. Working draft. Version 1.4 edition, 2002.
[9] O. Goldreich. *Foundations of Cryptography: Volume II (Basic Applications)*. Cambridge University Press, 2004.
[10] M. G. Gouda and A. X. Liu. Firewall design: consistency, completeness and compactness. In *IEEE ICDCS*, pages 320–327, 2004.
[11] M. G. Gouda and A. X. Liu. Structured firewall design. *Computer Networks Journal (Elsevier)*, 51(4):1106–1120, 2007.
[12] P. Gupta. *Algorithms for Routing Lookups and Packet Classification*. PhD thesis, Stanford University, 2000.
[13] A. X. Liu and F. Chen. Collaborative enforcement of firewall policies in virtual private networks. In *ACM PODC*, pages 95–104, 2008.
[14] A. X. Liu and M. G. Gouda. Diverse firewall design. *IEEE TPDS*, 19(8), 2008.
[15] A. X. Liu and M. G. Gouda. Complete redundancy removal for packet classifiers in tcams. *IEEE TPDS*, in press.
[16] A. X. Liu, C. R. Meiners, and E. Torng. Tcam razor: A systematic approach towards minimizing packet classifiers in tcams. *IEEE/ACM Trans. on Networking*, in press.
[17] A. X. Liu, C. R. Meiners, and Y. Zhou. All-match based complete redundancy removal for packet classifiers in TCAMs. In *IEEE INFOCOM*, pages 574–582, 2008.
[18] A. X. Liu, E. Torng, and C. Meiners. Firewall compressor: An algorithm for minimizing firewall policies. In *IEEE INFOCOM*, 2008.
[19] C. R. Meiners, A. X. Liu, and E. Torng. TCAM Razor: A systematic approach towards minimizing packet classifiers in TCAMs. In *IEEE ICNP*, pages 266–275, 2007.
[20] C. R. Meiners, A. X. Liu, and E. Torng. Bit weaving: A non-prefix approach to compressing packet classifiers in TCAMs. In *IEEE ICNP*, pages 93–102, 2009.
[21] C. R. Meiners, A. X. Liu, and E. Torng. Topological transformation approaches to optimizing tcam-based packet processing systems. In *ACM SIGMETRICS*, pages 73–84, 2009.
[22] S. C. Pohlig and M. E. Hellman. An improved algorithm for computing logarithms over gf(p) and its cryptographic significance. *IEEE Trans. on Info. Theory*, IT-24:106–110, 1978.
[23] D. K. H. D. R. Safford and D. L. Schales. Secure RPC authentication (SRA) for TELNET and FTP. Techn. Rep., 1993.
[24] S. Singh, F. Baboescu, G. Varghese, and J. Wang. Packet classification using multidimensional cutting. In *ACM SIGCOMM*, 2003.
[25] A. Wool. A quantitative study of firewall configuration errors. *IEEE Computer*, 37(6):62–67, 2004.
[26] Z. Yang, S. Zhong, and R. N. Wright. Privacy-preserving classification of customer data without loss of accuracy. In *SIAM*, 2005.
[27] L. Yuan, H. Chen, J. Mai, C.-N. Chuah, Z. Su, and P. Mohapatra. Fireman: a toolkit for firewall modeling and analysis. In *IEEE S&P*, pages 199 – 213, 2006.