July 2020

DESIGN AND IMPLEMENTATION OF PATH FINDING AND VERIFICATION IN THE INTERNET

Hao Cai

DESIGN AND IMPLEMENTATION OF PATH FINDING AND VERIFICATION IN THE INTERNET

A Dissertation Presented

by

HAO CAI

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2020

Electrical and Computer Engineering

DESIGN AND IMPLEMENTATION OF PATH FINDING AND VERIFICATION IN THE INTERNET

A Dissertation Presented

by

HAO CAI

Approved as to style and content by:

Tilman Wolf, Chair

Dennis L. Goeckel, Member

David Irwin, Member

Don Towsley, Member

Christopher V. Hollot, Department Head
Electrical and Computer Engineering

DEDICATION

To my parents.

ACKNOWLEDGMENTS

I would not have been able to complete this dissertation and achieve one of my dreams without the encouragement and guidance from people around me. While I use "we" out of deference to my co-authors, all mistakes contained herein are mine and mine alone.

I would like to express my deepest gratitude to my advisor, Professor Tilman Wolf, for his guidance and advice in the past years. During these years, I learned much from him, including rigorous scholarship and hard working. He taught me how to tackle new research problems, pitch ideas, and give presentations. I believe these skills will be valuable not only for my research but also for my future life.

I would like to thank my dissertation committee members, Professor Dennis L. Goeckel, Professor David Irwin, and Professor Don Towsley for their time of attending my presentation of the proposa and defensel. They also provided valuable feedback on the dissertation writing.

I would also like to thank my labmates and colleagues. Xinming Chen's programming skill has really impressed me. I feel fortunate to have worked with him on the credential and path finding projects. I have also worked closely with Abhishek Dwaraki. His rich engineering experience assured the successful implementation of my projects.

ABSTRACT

DESIGN AND IMPLEMENTATION OF PATH FINDING AND VERIFICATION IN THE INTERNET

MAY 2020

HAO CAI

B.Sc., SHANGHAI JIAO TONG UNIVERSITY

M.Sc., SHANGHAI JIAO TONG UNIVERSITY

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Tilman Wolf

In the Internet, network traffic between endpoints typically follows one path that is determined by the control plane. Endpoints have little control over the choice of which path their network traffic takes and little ability to verify if the traffic indeed follows a specific path. With the emergence of software-defined networking (SDN), more control over connections can be exercised, and thus the opportunity for novel solutions exists. However, there remain concerns about the attack surface exposed by fine-grained control, which may allow attackers to inject and redirect traffic.

To address these opportunities and concerns, we consider two specific challenges: (1) How can the network determine the choices of paths available to connect endpoints, especially when multiple criteria can be considered? And (2) how can endpoints verify the integrity of the path over which network traffic is sent. The latter consists of two subproblems, determining that the source of traffic is authentic and

determining that a specified path is traversed without deviation. In this dissertation, we investigate and present solutions for both the network path finding problem and the verification problem.

We first address path finding, or routing, which is a core functionality in the Internet. Existing approaches are either based on a single criterion (such as path length, delay, or an artificially defined "weight") or use a combinatorial optimization function when there are multiple criteria. We present a multi-criteria routing algorithm that can search the whole space of all possible paths. To achieve the scalability of our solution, we limit the search to only Pareto-optimal paths, which allows us to prune sub-optimal paths quickly and reduce computational complexity. We show that our approach is tractable on a variety of realistic topologies and the results Pareto-optimal paths can be clustered to present a few alternative options.

We then address path verification in the Internet, which consists of source authentication and path validation. Once a path has been selected, we show that an endpoint can validate that traffic indeed traverses along the chosen path. Prior work has relied on cryptographic approaches for such validation, which need significant computational resources. In contrast, we propose a lightweight and scalable technique to address this problem, which uses a set of orthogonal sequences as credentials in the packets. The verification of these orthogonal credentials is based on inner product computations, which can be easily implemented by basic bitwise operations in a processor. We show that the proposed approach can achieve the necessary security properties for both source authentication and path validation. Results from a prototype implementation show that the proposed technique can be implemented efficiently and only add a small computational overhead.

The results of our work enable novel uses of networks with fine-grained traffic control, such as enabling more path choices in networks where multiple performance criteria matter. In addition, our work contributes to efforts to make the Internet more

secure by presenting techniques that allow endpoints to validate the source and path of network traffic. We believe that these contributions help with improving both the current Internet and also future networks.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

LIST OF ALGORITHMS

CHAPTER 1

INTRODUCTION

Since its birth in the 1960s, the Internet has evolved in many aspects. As the diversity of uses for the Internet is increasing, many network protocols and architectures have been developed by the industry and the research community. In these proposed protocols and architectures, an important developing trend is that the network owners and operators have more controls on their infrastructures and data flows, allowing customization and optimization, and reducing the overall capital and operational costs. For example, Software Defined Networking (SDN) and OpenFlow have emerged as a new paradigm of networking, which transform the current Internet into an open and programmable component of a larger cloud infrastructure. The benefits of convenient control on the whole map of the entire network enable the introduction of new features in Internet to become less manual, less prone to error, and faster to implement.

Path finding or routing, i.e., determining a path for traffic to flow between communicating end-system, is a core functionality in such networks with the data flow control. In the current Internet, path finding is typically based on a single criterion, such as path length, delay, or an artificially defined weight. However, networks have grown in leaps and bounds so that single-criterion shortest paths no longer fit the whole spectrum of services that exist in today's networks. Multi-Criteria path problem has been addressed in several contexts, for example Quality of Service (QoS) routing. But when there are multiple optimization metrics, most approaches rely on an combinatorial optimization function, which combines all metrics into a single

metric (e.g., weighted sum). In contrast to these algorithms, our aim is to search the whole spectrum of all the possible optimal paths which have advantages even on any one metric, which can help the network owner or operator is able to take a more comprehensive consideration.

Once a selected data path is determined by a network owner or operator, how can they be sure that their preferences are truly enforced is another important concern in the future Internet. For instance, an enterprise might want the packets that it receives to pass through several services, such as an accounting service and a packet-cleaning service. Or a company might want fine-grained control over which providers carry which traffic between its branch offices, yet the network paths must respect the providers' pairwise business relationships. Or providers might want to make sure that they are carrying traffic only from friendly nations. These abilities of source authentication of packets and verification of an intended traffic path can help the Internet mitigate various attacks, such as DoS attack, address spoofing, flow redirection and etc. Unfortunately, the existing approaches either are unable to satisfy security requirements, or need a lot of computational resources (i.e., based on cryptographic techniques), which make these techniques expensive to implement in practice since potentially every packet needs to be checked at line rate. Therefore, in this work, we design a novel technique for the high-performance source authentication and path validation in the Internet, especially when considering that the further Internet may have billions of users with billions of services.

1.1 Path Finding with Multi-Criteria

While more and more network protocols and architectures enable the network owners and operators to decide intended paths that each data flow has been traversed on, there are usually a multiple choices of different paths that could be chosen. Widely used routing protocols, such as OSPF [61] and RIP [39], are designed to solve

this path-finding problem in the network. They use single routing metric and corresponding routing algorithms, such as Dijkstra's. However, networks have grown in the diversity of their use so that single-criterion shortest paths no longer fit the whole spectrum of services that exist in today's networks. For example, in networks where the user has the ability to choose paths [22, 83], the cost of a path and the quality of a path need to be represented by independent metrics. Also, load-balancing and use of alternate backup paths require multiple paths to be calculated [46]. In these cases, additional criteria beyond the shortest path metric may be used (e.g., available bandwidth, path reliability, etc.).

When only a single metric is used, a single optimal solution (i.e., shortest path) is enough. But when multiple metrics are used, a set of paths needs to be found to represent the trade-offs among criteria. A key challenge for realizing multi-criteria routing is the need to develop an efficient algorithm for determining suitable paths in the potentially very large space of all possible paths (exponential to the number of nodes). The multi-criteria path finding is an NP-hard [37] problem, but it is possible to develop solutions for typical-sized networks that work well in practice.

Previous work has addressed the multi-criteria optimal path problem in various contexts, for example Quality of Service (QoS) routing. A central problem in QoS routing is to find feasible paths between a source and a destination that satisfy multiple constraints (e.g., bandwidth, delay). Then, the best path among the feasible paths is selected based on a given optimization metric (e.g., delay-constrained least-cost path routing). When there are multiple optimization metrics, most approaches rely on an combinatorial optimization function [52], which combines all metrics into a single metric (e.g., weighted sum).

Using a single, combined metric simplifies the path finding problem, but also presents a fundamental limit on the ability to find solutions: a single optimization metric requires *a priori weighing* of each metric [29]. That is, before running the

path finding algorithm, the relative "value" between different metrics needs to be set. The result of the search is then optimal (only) for this fixed weighing of metrics. In practice, however, there are situations where this weighing cannot be done a priori. For example, a network may allow users to choose a specific path based on price and quality characteristics [22, 83]. In the marketplace of such a network, different paths need to be offered before knowing the users' sensitivity to the price and the quality (i.e., before knowing their weighting of criteria). Similarly, in SDN [63], an SDN controller may pre-compute available paths before knowing the weights of metrics that are desired by a specific northbound SDN application. In such cases, the weighing can only be done *a posteriori* and the multi-criteria optimal path problem needs to find the set of *all Pareto-optimal* paths. A path is Pareto-optimal if there is no other path that is better in all metrics. Since multiple metrics allow for the existence of paths that are better than others in one or more metric, but not all, there can be a large number of mutually Pareto-optimal paths. Based on the set of Pareto-optimal paths, one path can be chosen for any possible weighing of metrics (e.g., by an SDN application or by a network customer).

In our work, we address this multi-criteria path finding problem and design a high-performance algorithm to find all the possible Pareto-optimal paths, which is important in practice, especially in environments where different metric weights are unknown to the pathfinder. The key theoretical and practical research challenges this work in path finding with multi-criteria tries to address are:

- How to design a path finding algorithm that can find Pareto-optimal paths in multi-criteria networks?
- How to design it fast enough to find all the Pareto-optimal paths within 1 second on a typical sized network?

- How to find a subset of the Pareto-optimal paths in shorter time when the full Pareto-optimal set is not necessary?

1.2 Source Authentication and Path Validation

After the network owners and operators choose their intended paths for each data flow, the current Internet typically provides a simple delivery mechanism: we put destination addresses in packets and launch them into the network. We leave the network to decide the path that our packets take and the intermediate providers that the path passes through. Even network operators have little control over the paths that packets take toward them, or after leaving them. However, more and more endhosts and ISPs desire to validate service level agreement compliance regarding data delivery in the network: Did the packet truly originate from the claimed client? Did the client select a path that complies with the service provider's policy? Did the packet indeed travel through the path selected by the client?

The above discussed problems can be divided into two categories: *source authentication* and *path validation*, which we term "*path verification*" in this work. Source authentication and path validation are two important concepts in networking, which help construct higher-level security mechanisms, such as mitigating denial-of-service (DoS) attack, ensuring path compliance and packet attribution, and protecting against flow redirection. Source authentication is the verification of the source address of a host that sends a packet and is designed to determine whether this packet indeed originated from the claimed source. Path validation confirms that a packet indeed traversed the path known to (or selected by) the host (i.e., the source). The latter is used when senders, receivers, or operators want to ensure that a packet's path adheres to their preferences. For example, an enterprise might want to dictate that incoming traffic passes through certain services, such as deep packet inspection [53].

Path validation provides a way to verify this path compliance according to the policies of ISPs, enterprises, and data centers.

The current Internet does not provide any effective means for source authentication and path validation by routers or end-hosts. For example, a network provider cannot determine if traffic is sent by neighboring providers along paths that match service-level agreements; a receiver cannot be sure whether a packet is from a specific source, since an attacker can spoof source addresses in packets. Widely used end-to-end encryption and authentication schemes (e.g., TLS/SSL) are not able to solve these issues, since they are agnostic to which path their packets have been forwarded on. A stronger approach is needed, which enables routers and destinations to perform source authentication and path validation.

Most of existing approaches to implementing such source authentication and path validation are typically based on cryptographic schemes (e.g., digital signatures, HMAC [24] or UMAC [6] that uses a hash of the packet contents with a shared secret in it). They may work well in a domain-specific network with a limited number of users and dominated by strict security requirements. However, the high computational cost of cryptographic operations makes these techniques unsuitable for the data plane of the future Internet, where there are maybe up to billions of users with billions of services and potentially every packet needs to be checked at Gigabit per second link rates. Therefore, we design a novel technique for the high-performance path verification in the Internet, which use a set of orthogonal sequences as "credentials" in the packets. The key theoretical and practical research challenges this work in source authentication and path validation tries to address are:

- Where to place the credentials in a packet?
- How to design the credentials such that the size of the packet does not increase with the number of the hops.

- How to design the authentication mechanism that enables both source authentication and path validation simultaneously, while still providing the necessary security guarantees?
- How to decrease the router overhead as much as possible?
- How to test and implement such an authentication mechanism?
- How to compare the authentication mechanism with existing cryptographic approaches?

1.3 Organization and Contributions

This dissertation focuses on the research challenges discussed in Sections 1.1 and 1.2 to address the fundamental problems of the path finding and verification in the Internet. The rest of this dissertation is organized as follows, with the major contributions summarized in each.

Chapter 2 presents ParetoBFS, a new multi-criteria path finding algorithm to find all the possible Pareto-optimal paths for the Internet. This algorithm is a variant of the breadth-first search (BFS) algorithm and uses Pareto constraints to prune the traversal tree. Comparison with two existing algorithms shows ParetoBFS is tens to hundreds times faster and find more paths on typical sized networks. This chapter also shows a sampling heuristic to decreases the running time by only finding a subset of Pareto-optimal solutions.

Chapter 3 focuses on the source authentication in the Internet. An algorithm named OrthCredential is proposed to address this problem. OrthCredential uses a set of orthogonal sequences to verify packets along the path. It provides a fast and memory efficient method for source authentication. It is also resistant to DoS attacks.

Chapter 4 extends OrthCredential to a new algorithm named OSV (Orthogonal Sequence Verification) to further address the path validation problem in the Internet.

OSV also uses orthogonal capabilities that are carried in packets for verification, which can be implemented efficiently by basic bitwise operations on a processor. The experimental results show that the verification time in OSV is much lower than that existing approaches while providing the necessary security guarantees.

Chapter 5 evaluates the effectiveness and performance of our proposed path validation mechanism where "Docker" container [1] is used as the experiment tool.

Chapter 6 summarizes the previous chapters.

CHAPTER 2

PATH FINDING WITH MULTI-CRITERIA

Path finding, or routing, is a fundamental functionality in networking. Path finding in the current Internet uses a single criterion, such as hop count or link weight. Although there are proposed solutions to the multi-criteria optimal path selection problem for quality-of-service routing, since the routers eventually need to pick only one path, they usually combine all criteria into a single path optimization metric a priori. In contrast to these algorithms, our aim is to search the whole spectrum of all the possible optimal paths (i.e, Pareto-optimal path) that have advantages even on any one metric, which can help the network owner or operator is able to take a full consideration.

This chapter presents ParetoBFS, a variant of a breadth-first search that uses branch and bound techniques to find all the Pareto-optimal paths while effectively limiting the potentially very large search space. We present several sampling techniques to further increase the speed of the search while degrading the quality of the results only marginally. The simulation results show that existing multi-criteria combinatorial optimization approaches can only search a small fraction of all the Paretooptimal paths while our ParetoBFS can obtain the whole Pareto-optimal path set in shorter time. Some of the material in this chapter have been published in [24].

2.1 Introduction

Routing, which is determining a path for traffic to flow between communicating end-systems, is one of the essential functionalities of any computer network. In typical

networks, routing is based on a single criterion, such as path length, delay, or an artificially defined "weight." Widely used routing protocols, such as OSPF [61] and RIP [39], use single routing metric and corresponding routing algorithms, such as Dijkstra's algorithm [26] and the Bellman-Ford algorithm [13], to efficiently determine the optimal path between two network nodes.

However, networks have grown in the diversity of their use so that single-criterion shortest paths no longer fit the whole spectrum of services that exist in today's networks. For example, in networks where the user has the ability to choose paths [22, 83], the cost of a path and the quality of a path need to be represented by independent metrics. Also, load-balancing and use of alternate backup paths require multiple paths to be calculated [46]. In these cases, additional criteria beyond the shortest path metric may be used (e.g., available bandwidth, path reliability, etc.).

When only a single metric is used, a single optimal solution (i.e., shortest path) is enough. But when multiple metrics are used, a set of paths needs to be found to represent the trade-offs among criteria. A key challenge for realizing multi-criteria routing is the need to develop an efficient algorithm for determining suitable paths in the potentially very large space of all possible paths (exponential to the number of nodes). The multi-criteria path finding is an NP-hard [37] problem, but it is possible to develop solutions for typical-sized networks that work well in practice.

Previous work has addressed the multi-criteria optimal path problem in various contexts, for example Quality of Service (QoS) routing. A central problem in QoS routing is to find feasible paths between a source and a destination that satisfy multiple constraints (e.g., bandwidth, delay). Then, the best path among the feasible paths is selected based on a given optimization metric (e.g., delay-constrained least-cost path routing). When there are multiple optimization metrics, most approaches rely on an combinatorial optimization function [52], which combines all metrics into a single metric (e.g., weighted sum).

Using a single, combined metric simplifies the path finding problem, but also presents a fundamental limit on the ability to find solutions: a single optimization metric requires *a priori weighing* of each metric [29]. That is, before running the path finding algorithm, the relative "value" between different metrics needs to be set. The result of the search is then optimal (only) for this fixed weighing of metrics. In practice, however, there are situations where this weighing cannot be done a priori. For example, a network may allow users to choose a specific path based on price and quality characteristics [22, 83]. In the marketplace of such a network, different paths need to be offered before knowing the users' sensitivity to the price and the quality (i.e., before knowing their weighting of criteria). Similarly, in a Software-Defined Network (SDN) [63], an SDN controller may pre-compute available paths before knowing the weights of metrics that are desired by a specific northbound SDN application. In such cases, the weighing can only be done *a posteriori* and the multi-criteria optimal path problem needs to find the set of *all Pareto-optimal* paths. A path is Pareto-optimal if there is no other path that is better in all metrics. Since multiple metrics allow for the existence of paths that are better than others in one or more metric, but not all, there can be a large number of mutually Pareto-optimal paths. Based on the set of Pareto-optimal paths, one path can be chosen for any possible weighing of metrics (e.g., by an SDN application or by a network customer). In our work, we address this multi-criteria routing problem, which is important in practice, especially in environments where different metric weights are unknown to the pathfinder.

We present ParetoBFS, a variant of the breadth-first search (BFS) algorithm that uses Pareto constraints to prune the traversal tree. Experiments show that ParetoBFS can find *all* Pareto-optimal paths in a network in a reasonable time since typical-sized networks do not exhibit the characteristics that cause the problem space to become intractable. The specific contributions of this work are:

- The ParetoBFS algorithm that can find the entire set of Pareto-optimal paths in a network where the edges have an arbitrary number of metrics, both sum- and bottleneck-type. Comparison with two existing algorithms shows ParetoBFS is tens to hundreds times faster and finds more Pareto-optimal paths.

- A sampling heuristic for ParetoBFS that reduces the number of elements in the set of Pareto-optimal solutions and thus decreases the complexity of the path finding process. We show that despite not yielding all optimal solutions, this heuristic still yields solutions that are useful in practice.

We believe that this work provides a practical foundation for systematically using multi-criteria routing in networks to develop more effective network control applications in the future.

2.2 Background

Multi-criteria path finding has been studied extensively in the operations research community. This problem arises in many practical applications, including route planning in traffic networks [12] and QoS routing and traffic engineering in communication networks [78]. If the goal is to find the optimal path with some constraints on one or more metrics given a directed graph with edges that have a set of metrics, it is called multi-constrained path optimization (MCPO) [23, 30, 49, 52, 75, 79, 87]. Without the constraints on the metrics, this problem then becomes the multi-criteria optimization (MCO) problem [29, 37, 52, 67]. Solutions to MCPO and MCO are usually similar in that they use a combinatorial function on the multiple metrics (a priori) to find the optimal path.

The goal of ParetoBFS is to find all the Pareto-optimal paths, which is different from the prior work. Therefore, ParetoBFS is a broader solution to address both MCPO and MCO problems since the resulting paths from previous approaches are

usually a subset of the Pareto-optimal path set. These Pareto-optimal paths are important in many scenarios. For example, references [46] and [31] each describe a standalone routing service module that provides paths for other modules. Thus, the routing service module itself cannot make any choice for metric preferences. Also, in networks where paths are charged by their qualities, such as ChoiceNet [83], the cost and the quality of a path need to be represented by independent metrics. In these problems, there is no single objective function to select the best path, and it is impossible to give the paths an a priori ranking. Instead, the decision maker needs to see all the *Pareto-optimal* paths. Each Pareto-optimal path represents a trade-off between criteria, and may be equally important for the decision entity.

Section 2.7 compares the performance of ParetoBFS with some prior work in detail. The experiments show ParetoBFS is tens to hundreds times faster and can solve broader range of problems.

2.3 Preliminaries

Before describing the ParetoBFS algorithm in Section 2.4, we briefly introduce the network model and describe the formal definition of our path finding problem.

2.3.1 System Model

We model the network as a directed graph $G = (V, E)$, where V is the set of nodes and E is the set of edges interconnecting the nodes. n and m are the cardinalities of V and E, i.e., $n = |V|$, $m = |E|$, respectively.

To make the problem general enough, we consider that G is a multi-graph, which means there can be multiple edges between each node pair. (In practice, these multiple edges can correspond to different services that are offered on the same physical link, such as different QoS configurations.) In addition, we assume that each edge $\{e_{u,v}|u, v \in V\} \in E$ is associated with an edge criteria vector $w(u, v) =$

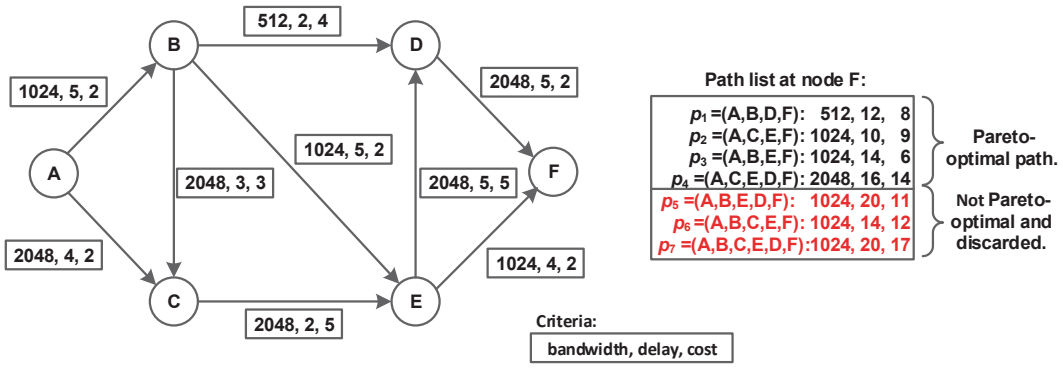


Figure 2.1. Example of Pareto-optimal path computation from node A to F.

$(w_1, w_2, ..., w_k)$, where k is the number of criteria. Each w_i corresponds to one of the independent criteria used in routing, such as bandwidth, latency, packet pass rate and cost. A path p from a source v_1^p to a destination v_r^p is defined as a finite sequence of edges that connects a sequence of vertices $(v_1^p, v_2^p, ..., v_r^p)$, $v_{i(i\leq r)}^p \in V$.

The path p can be assigned a path criteria vector $w^p = \{w_1^p, w_2^p, ..., w_k^p\}$. In this chapter, the calculation of the path criteria vector must satisfy the following property: when a hop is added to the path's end, the optimality of the new path does not increase on any criterion. Criteria satisfying this property can usually be classified into two types: sum-type criterion (e.g., delay) where $w_i^p = \sum_{e_{u,v}\in p} w_i(u,v)$; and bottleneck-type criterion (e.g., bandwidth) where $w_i^p = min(w_i(u,v))$[1].

2.3.2 Pareto-optimal Path

To define Pareto-optimality, we first define a *dominant path* as follows. We use the notation $\succeq$ to denote the left operand is more optimal than or equals to the right operand.

Definition 1 (Dominant path) *path p dominates path q if and only if*

[1]There are also multiplicative criteria (e.g. link reliability, packet loss rate), but they can be transformed into sum-type criteria by using a logarithm.

$$w_i^p \succeq w_i^q, \forall i \in \{1, 2, ..., k\}.$$

and the strict inequality holds at least once.

Then we can define Pareto-optimality as:

Definition 2 (Pareto-optimal path) *Path set P is called a Pareto-optimal set if and only if*

$$p \text{ does not dominate } q, \forall p, q \in P.$$

A path in a Pareto-optimal set is called a Pareto-optimal path.

In this chapter, the goal is to find all the Pareto-optimal paths from a source node to a target node in a given graph G. For instance, if each edge $e \in E$ has three metrics: bandwidth (w_1), delay (w_2) and cost (w_3), then the set of the Pareto-optimal paths P, which we finally find out, satisfies that, for $\forall p_i,\ p_j \in P, w_1^{p_i} > w_1^{p_j} \vee w_2^{p_i} < w_2^{p_i} \vee w_3^{p_i} < w_3^{p_j}$. This is different from the conventional multi-constrained optimal path problem [52], where a path optimization function f^p is used to combine all the metrics together and the optimal path is found by calculating the value of f^p on each path. As discussed above, the optimal path computed based on a single aggregated metric may not meet the multiple constraints being considered.

An example of the type of result we are aiming to obtain is shown in Figure 2.1. The edges of the graph are labeled with their respective metrics comprising of bandwidth (w_1), delay (w_2) and cost (w_3). There are seven paths $(p_1, p_2..., p_7)$ from source node A to destination node F. Among these paths, path $p_2 = (A, C, E, F)$ is strictly more optimal than path $p_5 = (A, B, E, D, F)$ because $w_2^{p_2} < w_2^{p_5}$ and $w_3^{p_2} < w_3^{p_5}$ while $w_1^{p_2} = w_1^{p_5}$. Therefore, path p_5 is not a Pareto-optimal path and would be discarded. Similarly, neither of the paths p_6 and p_7 are not Pareto-optimal paths because p_2 and p_3 is strictly more optimal than them. Finally, we get the Pareto-optimal paths $p_{1\sim4}$. (In the ParetoBFS algorithm, we maintain a list on each node to record all the

Pareto-optimal paths to this node and their corresponding parameters. Such a list is shown in black on node F in Figure 2.1.)

2.4 ParetoBFS : Pruning with Pareto Constraints

In this section, we first describe the plain breadth first search (BFS) solution to the multi-criteria path finding problem. Then, we describe how we use pruning to reduce the running time of the algorithm to a practical level.

2.4.1 Plain BFS to Find All Paths

A brute force solution to the multi-criteria path finding problem is to enumerate all the possible paths, then extract the Pareto-optimal set from them.

Algorithm 1 shows a variant of BFS algorithm that finds all the simple paths from the source node to a target node. Unlike the normal BFS, it does not maintain "visited" tags on the nodes, because a node may be visited multiple times when the algorithm examines different paths. Algorithm 1 starts from a source node and enqueues it into a path queue, i.e., *path_queue*. Then, the source node is dequeued and all the directed edges of it are enqueued into *path_queue* as new paths from the source node to some node in the graph. Each time a path is dequeued from *path_queue*, it is stored into the path set corresponding to its last node. Meanwhile, the out-edge neighbors of the dequeued path's last node are added to its end to form new paths. These new paths are further enqueued into *path_queue*. To prevent loops, Line 11 checks whether the neighbor is already in the path before appending it. After repeating this enqueue and dequeue process until *path_queue* is empty, *path_set* contains all the simple paths[2] from source node to all other nodes. Selecting a Pareto-optimal set from it is straightforward, as shown in function *pareto_add* of Algorithm 2.

[2]A simple path is a path which does not have repeating nodes.

Algorithm 1 BFS that finds Pareto-optimal paths by enumerating all the simple paths between two nodes.

```
 1: procedure BFS(G, source, target)
 2:     for all v ∈ G(v) do
 3:         path_set[v] ← ∅
 4:     end for
 5:     path_queue.push([source])
 6:     while path_queue.length > 0 do
 7:         path ← path_queue.pop()
 8:         s1 ← path.end()
 9:         for all edge ∈ s1.out_edges() do
10:             s2 ← edge.dest_node()
11:             if s2 ∉ path then
12:                 new_path ← path.append(edge)
13:                 path_set[s2] ← path_set[s2]∪
                                   {new_path}
14:                 if s2 ≠ target then
15:                     path_queue.push(new_path)
16:                 end if
17:             end if
18:         end for
19:     end while
20:     pareto_set ← ∅
21:     for all path ∈ path_set[target] do
22:         pareto_set ← pareto_add(pareto_set, path)
23:     end for
24:     return pareto_set
25: end procedure
```

Algorithm 1 can be easily extended to find the Pareto-optimal paths from one source node to all the other nodes, by replacing Line 13 with a *pareto_add* function, removing Line 14, and doing Lines 20 - 23 on each node.

The algorithm is obviously not scalable. In a directed graph, the number of possible paths is usually exponential to the number of nodes. Moreover, for a multi-graph with p parallel edges between each pair of nodes, the total number of paths increases with a factor of p^h, where h is the number of hops in a path. Figure 2.2(a) shows the number of paths traversed in Algorithm 1. It grows exponentially; enumerating all the possible paths is typically not feasible in both time and space. To make Al-

Algorithm 2 Pareto-optimal pruning process.

```
1: procedure PARETO_ADD(pareto_set, new_path)
2:     result_set ← ∅
3:     for all path ∈ pareto_set do
4:         if path is strictly more optimal than new_path then
5:             return pareto_set
6:         else if new_path is not strictly more optimal than path then
7:             result_set.append(path)
8:         end if
9:     end for
10:    result_set.append(new_path)
11:    return result_set
12: end procedure
```

gorithm 1 practical, it is necessary to prune the space of paths that are considered during the traversal.

2.4.2 ParetoBFS – Pruning While Searching

Since our goal is to find Pareto-optimal paths, we can stop considering a path if it is already strictly worse than other known paths. We call this process *pruning*. Formally, during the search process, a path ending with node v_i can be pruned if either of the following conditions satisfies:

1. The path is dominated by a path in the Pareto-optimal path set with destination node v_i.

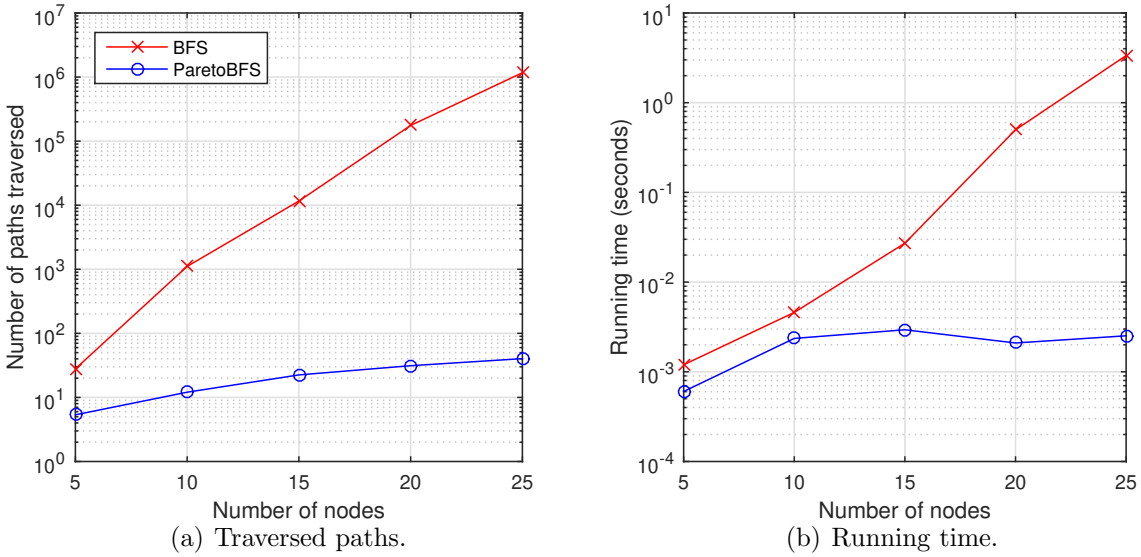
2. The path is dominated by a path in the Pareto-optimal path set with destination node $target$.

An algorithm with such pruning maintains the same theoretical worst-case time and space complexity. In practice, however, pruning reduces the size of the search tree dramatically. Note that pruning does not affect the correctness of the final solution, because extension cannot make a suboptimal path optimal.

Applying the pruning method to Algorithm 1, we can get the ParetoBFS algorithm as shown in Algorithm 3. Instead of saving all the paths, a set *pareto_set* is used

Algorithm 3 ParetoBFS

```
1: procedure ParetoBFS (G, source, target)
2:     for all v ∈ G(v) do
3:         pareto_set[v] ← ∅
4:     end for
5:     path_queue.push([source])
6:     while path_queue.length > 0 do
7:         path ← path_queue.pop()
8:         s1 ← path.end()
9:         if path is Pareto-optimal for pareto_set[target] and path ∈ pareto_set[s1]
   then
10:                 ▷ Check whether the path satisfies the Pareto-optimal conditions
11:             for all edge ∈ s1.out_edges() do
12:                 s2 ← edge.dest_node()
13:                 if s2 ∉ path then
14:                     new_path ← path.append(edge)
15:                     if new_path is Pareto-optimal to pareto_set[target] and
   pareto_set[s2] then
16:                                                 ▷ see if new_path can be added into the
   Pareto-optimal path set to node s2
17:                         pareto_add(pareto_set[s2],
                                       new_path)
18:                         if s2 ≠ target then
19:                             path_queue.push(new_path)
20:                         end if
21:                     end if
22:                 end if
23:             end for
24:         else
25:             continue
26:         end if
27:     end while
28:     return pareto_set[target]
29: end procedure
```



(a) Traversed paths. (b) Running time.

Figure 2.2. Comparison with respect to the number of traversed paths (a) and running time (b) for ParetoBFS and BFS: a BRITE- generated topology, 2 metrics, 1 parallel edge, and averaging over 60 runs with different graphs and source/target nodes.

to save the Pareto-optimal paths from the source node to each node. It differs from Algorithm 1 in Lines 9, 15 and 17. Lines 15 and 17 check the Pareto-optimality before the enqueue step, to eliminate any suboptimal path. There is another check after the dequeue step in Line 9, because the Pareto-optimal sets may have changed during the time that path stays in the queue. Figure 2.2(a) shows that the pruning method can effectively reduce the number of traversed paths by several orders of magnitude. The detailed performance and complexity analysis is shown in Section 2.5.

Algorithm 3 can be extended to find Pareto-optimal paths to all other nodes, by removing the condition checks involving the *target* Pareto-optimal set in Lines 9, 15 and 18. The running time increases because of the less strict pruning conditions.

2.5 Evaluation and Complexity Analysis

In this section, we discuss the effectiveness of our ParetoBFS algorithm in the context of network graphs to show that it is practically useful.

2.5.1 Methodology

To test the performance of the path finding algorithm, we use both generated topology and real-world topology. Although ParetoBFS can apply to both inter- and intra-AS topologies, most of the intelligent routing applications are used within private domains. So we focus on the intra-AS topology here. We use the BRITE topology generator [60] to generate router-level topologies. The sizes of the topologies range from 100 nodes to 10,000 nodes. BRITE provides three metrics for paths: length, bandwidth and latency. When testing with more than 3 metrics, we add extra random parameters besides these 3 metrics.

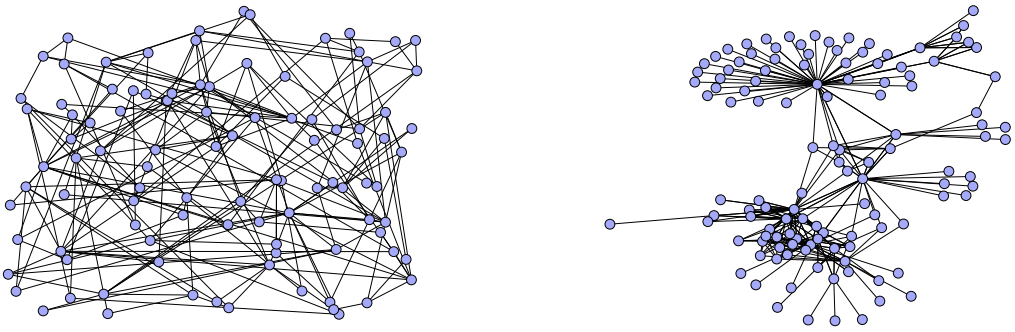
BRITE provides four generation models: Waxman [80], BA [10], BA-2 [4] and GLP [17](the GLP model is mainly for AS-level topologies). The node placement has two options: random and heavy-tailed. The bandwidth distribution has four options: constant, uniform, exponential and heavy-tailed. We test all the combinations and list the running time and the Pareto-optimal path count in Table 2.1. It can be observed that, except for the constant options, other combinations of parameters do not show significant difference in the path finding result. Therefore, we can arbitrarily pick these parameters. In the following experiments, the generation model is set to Waxman, a most commonly used intra-AS model, the node placement is set to random, and the bandwidth distribution is uniform distribution.

As for the real-world topology, we use Rocketfuel [73], an ISP topology data set measured by the University of Washington. Each Rocketfuel data file represents a topology of one AS, ranging from 100 nodes to 10,000 nodes. The data we use does not include any metric such as bandwidth or latency, so we randomly generate values for the metrics using a normal distribution.

Both the generated and the real-world topologies are uni-graphs, i.e., topologies with only one edge between the same pair of nodes. However, sometimes we need more than one edge between two nodes, these parallel edges can be either physical

Table 2.1. Different BRITE parameters does not have significant impact on Pareto-BFS running time and average Pareto-optimal path count. (1,000 nodes, 3 metrics, 1 parallel edge. Each result is an average over 100 runs. The unit of time is second.)

Node Placement	Bandwidth Distribution	Model							
		Waxman		BA		BA-2		GLP	
		time	paths	time	paths	time	paths	time	paths
Random	Constant	0.03	1.00	0.03	1.00	0.06	1.00	0.03	1.00
	Uniform	0.36	7.46	0.26	5.22	0.64	7.42	0.12	2.24
	Exponential	0.36	6.60	0.23	4.16	0.62	7.22	0.09	1.94
	HeavyTailed	0.42	6.64	0.28	4.84	0.68	7.40	0.12	2.36
Heavy Tailed	Constant	0.04	1.00	0.05	1.00	0.08	1.00	0.03	1.00
	Uniform	0.59	8.46	0.37	5.24	0.89	7.78	0.15	1.98
	Exponential	0.52	7.22	0.30	5.78	0.81	7.96	0.15	2.46
	HeavyTailed	0.48	7.62	0.25	4.68	0.84	7.76	0.13	2.70



(a) BRITE generated topology, 100 nodes.

(b) Rocketfuel topology, AS 4755, 121 nodes.

Figure 2.3. Examples of test topologies.

links with different metrics, or service offerings on the same link but with different QoS limits. To extend the uni-graphs to multi-graphs, each edge of the uni-graph is duplicated and assigned with Pareto-optimal metrics.

We use Python to implement our algorithms because of its convenient graph libraries, and the ability to integrate into the pox[3] SDN controller, which also uses Python. We use the pypy[4] interpreter to run the experiments, which can achieve

[3]http://www.noxrepo.org/pox/about-pox/

[4]A Python interpreter with JIT compiler. http://pypy.org/

performance close to the native code. One exception is the convex sampling in Section 2.6, we use CPython for that experiment because the convex hull calculation uses pyhull, which is not pypy compatible.

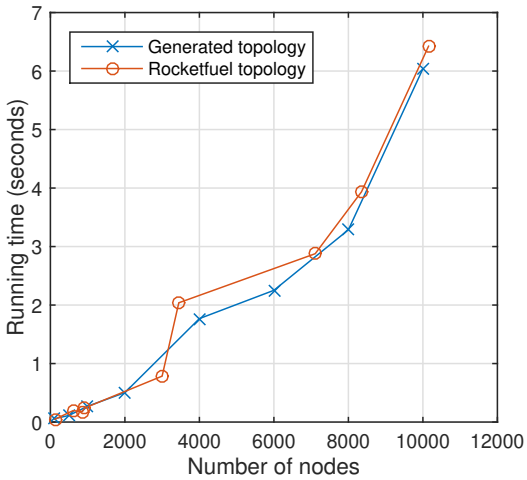
The processor we use is an Intel Core2 Quad CPU Q9400 running at 2.66 GHz. The software configuration is Ubuntu 14.04 64-bit with kernel version 3.13.0-24 and pypy 2.6.0.

2.5.2 Complexity Analysis

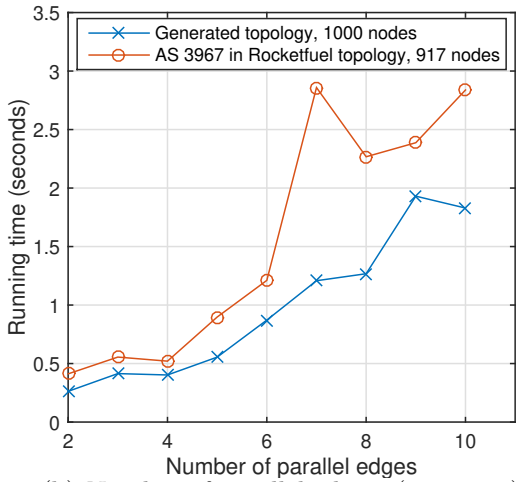
In this section, we provide a theoretical analysis on the plain BFS and Pareto-BFS algorithms (i.e., Algorithms 1 and 3). Let $G = (V, E)$ be the graph, where $V = (v_1, v_2, ..., v_n)$ is a set of all nodes of the graph and $E = (e_1, e_2, ..., e_m)$ is a set of all edges of the graph. The number of criteria is k. We assume the source node is v_1 and the target node is v_n.

Recall that Algorithm 1 first finds all possible paths and then the Pareto-optimal paths among all these paths. On the other hand, Algorithm 3 finds the Pareto-optimal path each time when it visits a node. We first analyze the time to find all the paths in Algorithm 1.

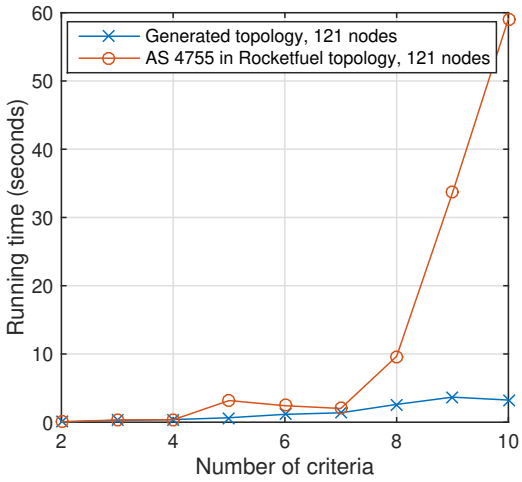
As discussed in Section 2.3, a suboptimal path cannot become optimal when a hop is added to its end. Therefore, all Pareto-optimal paths considered in this chapter are simple paths, which do not have repeating vertices. In a directed graph, for a simple path, we can order the vertices so that edges only point forward. E.g., if node u is a descendent of node v, then node u comes after node v in the sorted list of nodes. In Algorithm 1, the times that each node v_i $(i = 1, 2, ..., n)$ is visited are the number of the paths from source node v_1 to node v_i. Let v_2 be the next node. The number of paths from v_1 to v_2 is the number of parallel edges between them. Let v_3 be one of v_2's neighbours, the number of paths from v_1 to v_3 is the number of (direct) edges from v_1 to v_3, plus the paths that use v_2 as an intermediate vertex. More generally,



(a) Number of nodes. (2 criteria, 3 parallel edges)

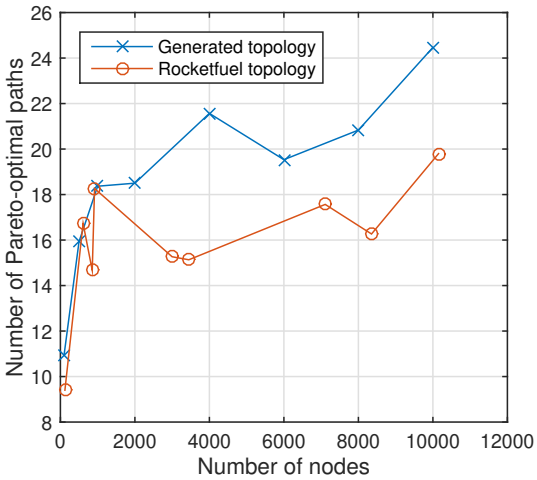


(b) Number of parallel edges. (2 criteria)

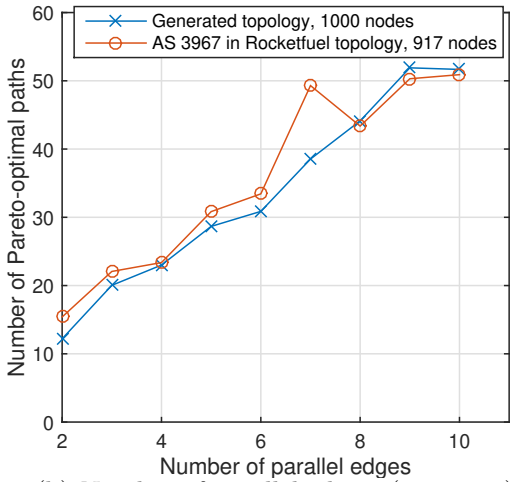


(c) Number of criteria. (3 parallel edges)

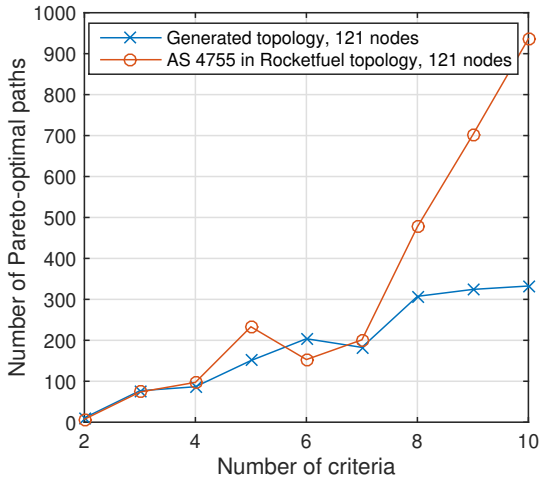
Figure 2.4. The time complexity of ParetoBFS to different variables. Each data point is an average of 30 runs.



(a) Number of nodes. (2 criteria, 3 parallel edges)



(b) Number of parallel edges. (2 criteria)



(c) Number of criteria. (3 parallel edges)

Figure 2.5. The number of Pareto-optimal paths found. Each data point is an average of 30 runs.

let $e(i,j)$ be the number of directed edges between node v_i and node v_j ($e(i,j) = 0$ if v_i and node v_j are not adjacent nodes), and $d(j)$ be the number of paths from v_1 to v_j, then we have:

$$d(j) = e(1,j) + \sum_{i=2}^{j} d(k)e(k,j).$$

For each node v_j, computing $d(j)$ takes time proportional to the in-degree of node v_j, and overall it will take $O(m)$ time. Therefore, Algorithm 1 visits each node $O(m)$ times, and the total time to find all the possible paths in Algorithm 1 is $O(nm)$ time. To calculate the complexity of the Pareto selection phase, we denote p as the number of all the paths from source node v_1 to target node v_n. p could be 1 if there is only 1 simple path from node v_1 to node v_n, however, p could also be $n!$ when graph G is full mesh (each node connects to every other node). The operation of Algorithm 2 takes $O(k)$ times computation for each path in the input *pareto_set*. The process of screening out the Pareto-optimal paths adds 1 Pareto-optimal path each time from the temporary *pareto_set*, and the number of paths in *pareto_set* goes from 0 to $p-1$. Therefore, the process will compute $O(k(1+2+\cdots+p)) = O(kp^2)$ times. Then, the running time for Algorithm 1 is $O(nm + kp^2)$.

In contrast to Algorithm 1, Algorithm 3 deletes the non-Pareto-optimal paths from source node v_1 to node v_j each time when it visits node v_j. Therefore, the number of paths saved in *path_queue* in Algorithm 3 will be less than that of Algorithm 1. The number could be the same when all paths are Pareto-optimal . Thus, in the worst case, Algorithm 3 also visits each node $O(m)$ times. We denote p^* as the Pareto-optimal paths between the source node v_1 and the target node v_n. The total running time for Algorithm 3 is $O(nmkp^*)$.

The time complexity of Algorithm 1 is dominated by the number of the paths p. In fact, in a typical network topology, p usually grows exponentially with the number of nodes n. We can take the graph in Figure 2.1 as an example. If we have

2 parallel edges between each connecting node pairs, then number of the paths from node A to node F becomes $3 \times 2^3 + 3 \times 2^4 + 1 \times 2^5 = 104$, which is much larger than n ($n = 6$). Besides, the number of the possible paths doubles when a new node is added into the graph. On the contrary, the time complexity of Algorithm 3 may not be dominated by the number of the Pareto-optimal paths p^* when p^* is just a small fraction of p. However, the optimal path fraction would grow rapidly when the number of considered metrics increases. In this case, the time complexity is dominated by p^*, and also grows approximate exponentially with n. The experimental results in the next section indicate the correctness of our analysis here.

2.5.3 Experimental Results

In this section, we present the experimental results of the ParetoBFS algorithm. We first present the running time of the plain BFS and ParetoBFS algorithms in Figure 2.2(b). It shows that the running time of plain BFS increases exponentially with the increase of the number of nodes. The complexity of ParetoBFS is sub-exponential, i.e., the running time may grow faster than any polynomial solution but is still significantly smaller than an exponential solution. This makes sense because ParetoBFS 's running time grows exponentially with the number of nodes in the worst case, which happens when the number of the Pareto-optimal paths makes up a large part of the paths between the source and target node. However, in a realistic network topology, the Pareto-optimal paths are usually a small fraction of the total paths. So the pruning method can prevent the curve from going too steep, because it keeps removing non-Pareto-optimal paths at each node, therefore it avoids unnecessary comparisons afterwards.

We then present the running time of ParetoBFS to find all the Pareto-optimal paths in graphs with different parameters. Here, we only focus on the running time. The memory consumption is proportional to the running time, because it depends on

the length of the *path_queue*. Figure 2.4 shows how the average running time of ParetoBFS grows with the increasing number of nodes, parallel edges and criteria, respectively. Figure 2.4(a) shows that ParetoBFS can find all the Pareto-optimal paths on a 10,000-node topology in 30 seconds. Figure 2.4(b) shows a similar complexity with the number of parallel edges as in Figure 2.4(a). This is also reasonable because increasing the number of parallel edges and increasing the number of nodes have the same effect on the traversal queue length, and the pruning methods also have similar effects on these two metrics. Figure 2.4(c), however, shows a steeper growth than the previous figures. For instance, if there are a number of k metrics $w_1, w_2..., w_k$ on each edge (the value of w_k is generated randomly), considering two neighboring nodes with two parallel edges connecting them, the probability that these two edges are Pareto-optimal is $1-\frac{1}{2^{k-1}}$. When k grows, the number of the Pareto-optimal paths between two nodes approaches the number of all the paths between them. This is the worst case for ParetoBFS which makes the running time grows exponentially. The large number of metrics also makes the Pareto pruning not working efficiently, which makes the running time grow faster than in Figure 2.4(a) and 2.4(b). In order to reduce the running time when the number of metrics is high, Section 2.6 proposes several sampling methods to reduce the size of the Pareto-optimal set.

Figure 2.5 shows how the number of the Pareto-optimal paths, p^*, grows with the increasing number of nodes, parallel edges and criteria, respectively. In Figure 2.5(a), the Rocketfuel-topology curve fluctuates, because each real topology has unique interior structure which is not as uniform as the generated topology. In Figure 2.5(b), it can be observed that the number of the Pareto-optimal paths varies linearly with the number of parallel edges when there are 2 criteria on each edge. The curves show the correctness of the analytic $O(nmkp^*)$ running time for ParetoBFS in the last section when compared with Figure 2.4(b). When the number of parallel edges doubles, p^* and m also double. Therefore, if the curves of p^* in Figure 2.5(b) can be considered as

linear, the curves are polynomial in Figure 2.4(b). Figure 2.5(c) shows how p^* varies with the number of criteria. It can be observed that p^* increases in Figure 2.5(c) more than in Figure 2.5(a) and in Figure 2.5(b). As discussed in Section 2.5.2, a large number of the criteria results in the number of the Pareto-optimal paths approaching the number of all the paths.

2.6 Sampling Pareto-optimal Paths

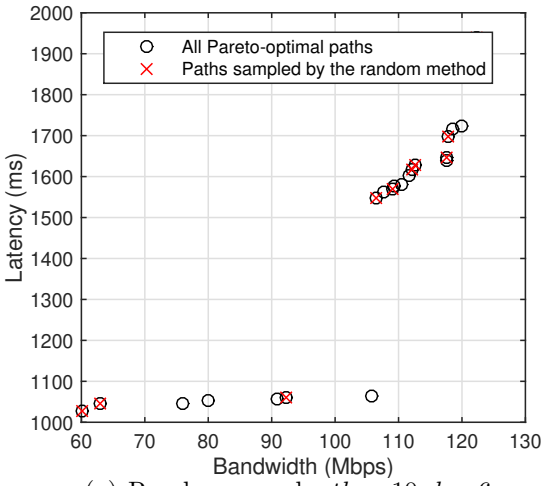
Algorithm 4 Sampling the Pareto-optimal set after adding a path.

```
1: procedure SAMPLING_ADD(pareto_set, new_path)
2:     result_set = pareto_add(pareto_set, new_path)
3:     if result_set.length > th then
4:         sampled_set = sampling(result_set)
5:         return sampled_set
6:     else
7:         return result_set
8:     end if
9: end procedure
```

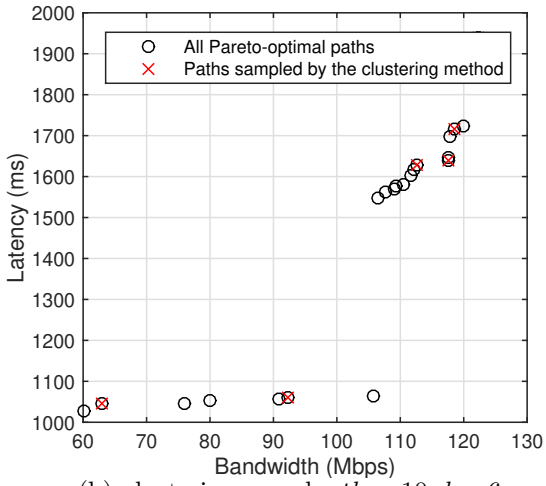
ParetoBFS finds all the Pareto-optimal paths. But as the number of criteria increases, the size of Pareto-optimal set may grow exponentially. Even for a small 1,000-node network with just 3 criteria, there may be hundreds of Pareto-optimal paths between two nodes.

Sometimes it is not necessary or too slow to find all the Pareto-optimal paths, so we introduce a heuristic based on sampling. Sampling the Pareto-optimal set can be useful in two ways: (1) sampling reduces the difficulty of choice for the entity selecting among Pareto-optimal paths; (2) if sampling happens during the search, the number of traversed paths can be further reduced, and the algorithm can find a set of paths that are close to the Pareto-optimal set in a shorter time.

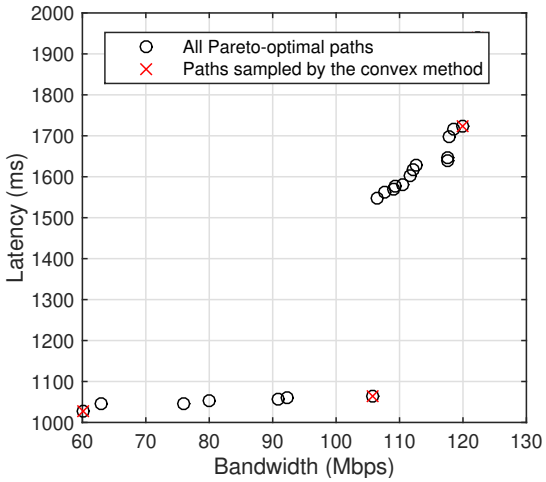
Algorithm 4 describes how to sample paths. Every time after a path is added to the Pareto-optimal set, the algorithm checks the Pareto-optimal set size. If the size is larger than a threshold th, a sampling method is used to reduce the Pareto-optimal set



(a) Random sample, $th = 10$, $l = 6$.



(b) clustering sample, $th = 10$, $l = 6$.



(c) Convex sample, $th = 10$.

Figure 2.6. The effect of different sampling methods. (2 criteria, 3 parallel edges, 10,000 nodes)

to l paths. It should be noted that sampling can discard some useful path halfway. It is possible that the final result is not a subset of the original ParetoBFS result.

Assuming $P = \{p_1, ..., p_m\}$ is the Pareto-optimal set found by ParetoBFS , and $Q = \{q_1, ..., q_n\}$ is the Pareto-optimal set found by ParetoBFS with sampling. To compare the effectiveness of the sampling methods, we propose the following metrics:

- Running Time Ratio (RT) is defined as the ratio of the running time to find Q to the running time to find P. This metric indicates how the sampling method affects the running time.

- Path Count Ratio (PC) is defined as the ratio of Q's size to P's size, that is, $PC = n/m$. This metric indicates how many Pareto-optimal paths can be found using this sampling method. It does not indicate the optimality of the paths.

- Path Quality (PQ) is defined as the average k-dimensional Euclidean distance between P's and Q's criteria vector sets $w^Q = \{w^{q_1}, ..., w^{q_n}\}$ and $w^P = \{w^{p_1}, ..., w^{p_m}\}$. Each w^{q_i} or w^{p_i} is a Pareto-optimal path's criteria vector. To calculate PQ, first normalize w^P and w^Q into w^P's range:

$$w_j^{p_i\prime} = \frac{w_j^{p_i} - \min(w_j^{p_1} \dots w_j^{p_m})}{\max(w_j^{p_1} \dots w_j^{p_m}) - \min(w_j^{p_1} \dots w_j^{p_m})}, \substack{i\in\{1...m\} \\ j\in\{1...k\}}$$

$$w_j^{q_i\prime} = \frac{w_j^{q_i} - \min(w_j^{p_1} \dots w_j^{p_m})}{\max(w_j^{p_1} \dots w_j^{p_m}) - \min(w_j^{p_1} \dots w_j^{p_m})}, \substack{i\in\{1...n\} \\ j\in\{1...k\}}$$

then for each $w^{q_i\prime}$, calculate the distance from its closest $w^{p_t\prime}$:

$$d^{q_i} = \min_{t\in\{1...m\}} \sqrt{\sum_{j\in\{1...k\}} (w_j^{q_i\prime} - w_j^{p_t\prime})^2}$$

Then PQ can be defined as: $PQ = \frac{1}{n}\sum_{i=1}^{n}(d^{q_i})$. It can be viewed as the average distance between w^P and w^Q, $PQ = 0$ means Q is a subset of P.

The sampling method must be fast and be able to process an arbitrary number of criteria. Assuming there is no preference over any criterion, the sampling methods should treat each criterion equally. In this section, three sampling techniques are investigated: random, clustering, and convex sampling.

2.6.1 Random Sampling

This method randomly samples l paths from the Pareto-optimal set. It is fast, but does not make use of any information of the data points. The result of a 2-criteria example is shown in Figure 2.6(a). Q mostly overlaps with P, which means that, after sampling, we can still find an approximate subset of the Pareto-optimal paths.

2.6.2 Clustering Sampling

It is an intuitive idea to cluster Pareto-optimal points that are close to each other in the k-dimensional space, especially when looking for redundant paths is not the goal. Here, we use Lloyd's clustering algorithm [57] to divide the points into l groups, and select the points closest to the center of each group.

Lloyd's algorithm's time complexity is $O(nkli)$ (n being the number of points; k being the dimension; l being the number of groups; i being the number of iterations). The example of a clustering result is shown in Figure 2.6(b). The points are more dispersed than Figure 2.6(a), thus they are more representative.

2.6.3 Convex Sampling

The assumption of convex sampling is that the points on the convex hull are better than the ones inside. This can be illustrated by Figure 2.7. Points 1-5 are Pareto-optimal points. Points 1, 2, 4, 5 and the *nadir point* (not a real data point) forms the convex hull. Point 3 is inside the hull. Compared to Point 2, Point 3 only improves a little in bandwidth, but sacrifices a lot in latency. The similar situation applies to Point 3 and 4. Therefore, Points 2 and 4 seems more preferable than Point 3. This

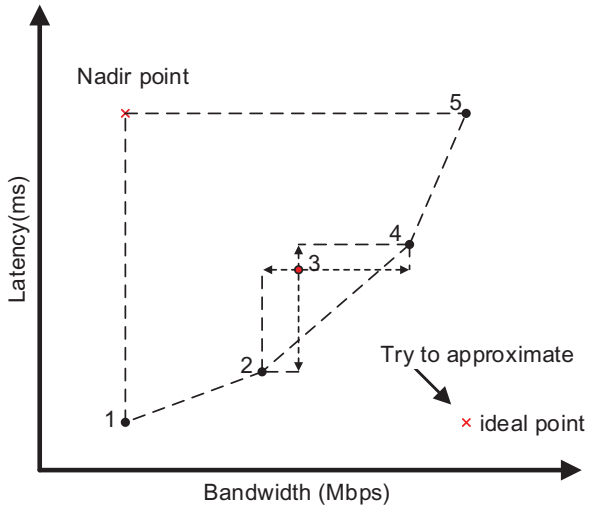


Figure 2.7. Example of convex sampling.

method works better if the criterion is sum-type, because the points on the convex hull are more likely to stay optimal when the path is extended.

We use the *qhull* library, which implements the Quickhull algorithm [11]. Its time complexity is $O(nlogv)$ in 2-d and 3-d, and $O(n^{\frac{v(\lfloor d/2 \rfloor - 1)}{\lfloor d/2 \rfloor}})$ for higher dimensions (n being the number of points; v being the number of points on the convex hull). The result in Figure 2.6(c) successfully eliminates the points inside the convex hull. For dimensions higher than 4, the performance of qhull degrades rapidly, it may no longer help speeding up the algorithm.

The advantage of the convex sampling is that it always reserves the corner points (e.g. Points 1 and 5 in Figure 2.6(c)), which represent the extreme values in one dimension, and they are more important if the decision maker wants to choose the highest value in one dimension. Another advantage is that the calculation of the convex hull does not require normalizing each dimension, thus improving the speed.

Table 2.2. The effectiveness of sampling methods.

k	th	l	random			clustering			convex		
			RT	PC	PQ	RT	PC	PQ	RT	PC	PQ
2	10	5	1.175	0.850	0.141	1.632	0.869	0.004	1.058	0.828	0.001
3	20	10	0.530	0.461	0.022	1.405	0.455	0.026	0.431	0.546	0.007
4	100	10	0.473	0.384	0.030	1.087	0.413	0.032	0.393	0.502	0.030

The disadvantage is that the convex sampling cannot control how many points are sampled. It is possible that too few or too many points are left, which brings uncertainty to the quality of the sampling result as well as the running time.

2.6.4 Comparison of Sampling Techniques

We test the three sampling techniques on 9 Rocketfuel topologies, whose sizes range from 121 to 10,152, and get the average RT, PC and PQ. The results are listed in Table 2.2. The sampling threshold th and sample size l also affect RT, PC and PQ. They are chosen from trial runs, to get a compromise between the running time and the result accuracy.

As for RT, the sampling techniques do not reduce the running speed when $k = 2$, but they tend to reduce the running time at higher dimensions. The random and convex sampling speeds are about the same. The clustering is much slower than the other two, thus is not recommended. As for PC, all the three techniques can find a similar amount of Pareto-optimal paths, even for 4 criteria problems, they can still find 40% to 50% of the Pareto-optimal paths. The convex sampling performs slightly better at higher dimensions. As for PQ, the convex sampling has the best path quality, but its PQ increases much faster than the other two, this may be because the convex sampling cannot control the sample size, so the result accuracy is less adjustable.

Overall, the convex sampling works the best among the three sampling methods, at least for $k = 2, 3, 4$. It is faster, finds more Pareto-optimal paths with higher path

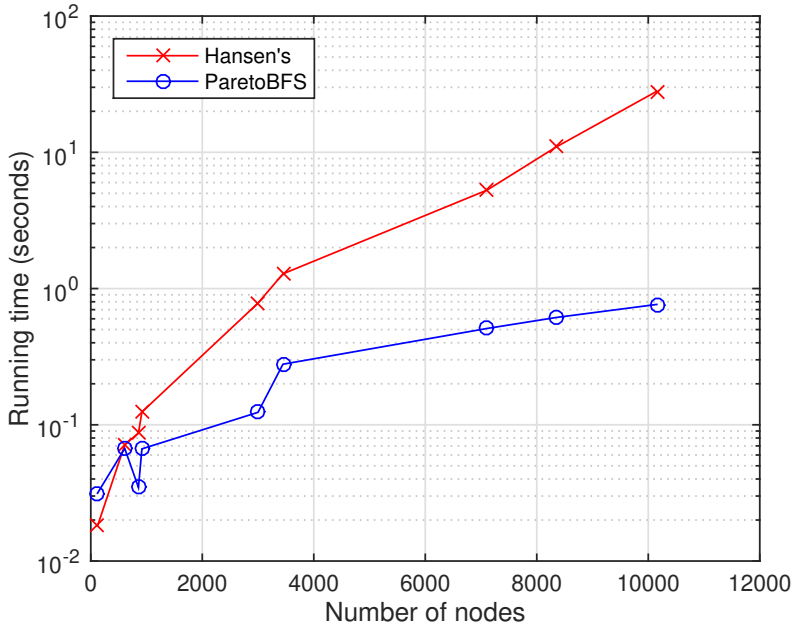


Figure 2.8. Running speed of Hansen's algorithm and ParetoBFS. (1 sum-type metric and 1 bottleneck-type metric, 1 parallel edge, Rocketfuel topology)

quality. Therefore, the convex sampling is recommended when dealing with 2, 3 and 4 criteria topologies.

2.7 Comparison with Related Work

As discussed in Section 2.2, much previous work has addressed the multi-criteria path finding problem. There are several survey papers and bibliographies [29, 35, 58, 76], which summarize more than 40 papers about the multi-criteria shortest path problem. Unfortunately, most of the papers only deal with sum-type metrics. Only two papers – Hansen [37] and Pelegrin et al. [67] – consider one sum-type and one bottleneck-type metric. Gandibleux et al. have a paper considering one bottleneck-type and an arbitrary number of sum-type metrics [34]. We have implemented Hansen's algorithm, and the comparison with ParetoBFS is shown in Figure 2.8.

Table 2.3. Comparison Martins' algorithm with ParetoBFS on 4 Rocketfuel topologies.

# of	k=2			k=3			k=4		
nodes	RT	PC	PQ	RT	PC	PQ	RT	PC	PQ
121	1.1	0.56	0.0000	1.3	0.41	0.0000	1.5	0.44	0.0000
609	23.7	0.42	0.0050	178.7	0.38	0.0018	121.2	0.38	0.0003
855	126.5	0.68	0.0000	233.4	0.61	0.0004	258.9	0.53	0.0007
917	34.4	0.41	0.0074	169.6	0.24	0.0008	279.1	0.37	0.0006

The Hansen's algorithm examined here is Algorithm 2 in reference [37]. It uses a multiple labeling scheme. Since Hansen's algorithm finds the exact Pareto-optimal set, we only compare the running time here.

Figure 2.8 shows the running speed between Hansen's algorithm and ParetoBFS , we can see that ParetoBFS 's running time grows slower with increasing nodes. Even for small topologies with a few hundred nodes, ParetoBFS is as fast as Hansen's algorithm. For the large topology with 10,000 nodes, ParetoBFS is about 40 times faster than Hansen's algorithm. Not to mention that Hansen's algorithm is only designed for the bi-criteria problem, while ParetoBFS is capable of dealing with more criteria.

Other than the *exact methods* (i.e. to find all the Pareto-optimal paths) like ParetoBFS and Hansen's algorithm, many papers propose *approximation methods* to find a subset of Pareto-optimal paths in an efficient manner. These are known as *fully polynomial approximation schemes* (FPAS). All the FPAS we investigated are only for sum-type metrics[5]. Here, we compare ParetoBFS with a popular FPAS – Martins' algorithm [59].

[5]In some work (e.g., [52]), it is suggested that bottleneck types can be converted to sum types by reciprocal. That is, define the optimal goal as: $f^p = \sum_e \frac{1}{bandwidth(e)}, e \in p$, where p is a path and e is an edge on p.

Table 2.4. Comparison of path finding algorithms. (p^* and p are the numbers of all the Pareto-optimal and possible paths between two nodes, respectively.)

Type	Number of criteria	Number of Pareto-Optimal paths	Complexity
Plain BFS	k	p^*	$O(mn + kp^2)$ $(p > p^*)$
ParetoBFS	k	p^*	$O(mnkp^*)$
Hasen's [37]	2	p^*	$O(p^{*2} \log n)$
Martins' [59]	k sum-type metrics	ω $(\omega < p^*)$	$O(k^2 \frac{m}{n} \omega^2 \log \omega)$

Martins' algorithm only gives an approximation of the Pareto-optimal set, which may differ from the exact Pareto-optimal set. Similarly, we compare the quality of results as in Section 2.6. The results on 4 Rocketfuel topologies are shown in Table 2.3. Even for graphs with hundreds of nodes, the running speed of Martins' algorithm is tens to hundreds times slower than ParetoBFS . On larger Rocketfuel topologies, Martins' algorithm becomes too slow to be feasible. Though Martins' algorithm finds a reasonable portion of the Pareto-optimal set (about 40% to 60%) and the quality of paths is very close to the exact Pareto-optimal set, Martins' algorithm is too slow compared to ParetoBFS . Besides, ParetoBFS can find all the Pareto-optimal paths while Martins' algorithm only finds a part of them.

Table 2.4 compares the complexity of ParetoBFS with Hansen's algorithm and Martins' algorithm. From the comparison, we can see that ParetoBFS is superior than prior work in various aspects: It is able to take an arbitrary number of sum-type and bottleneck-type metrics. Besides, it finds the full Pareto-optimal set faster than other exact methods. Our experiments also show that it is even faster than certain FPAS in practice.

2.8 Conclusions

In this chapter, we address the problem of finding multiple Pareto-optimal paths in a network where multiple criteria are used for routing. Such information is necessary

in networks where path choices need to be provided to consumers for a posteriori selection. We have described ParetoBFS , an algorithm to find all the Pareto-optimal paths in a network. The experiments show that the algorithm works well and can get a solution on a typical network in reasonable time. We have also proposed several sampling techniques to further reduce the running time when finding all the Pareto-optimal paths is not necessary or not feasible. We believe that this work presents an important step toward enabling novel routing techniques in modern networks.

CHAPTER 3

SOURCE AUTHENTICATION

In the Internet, an important problem is to make sure only authorized users (i.e., those who have paid for a particular network service) can access the service. Most existing authentication approaches are based on cryptographic techniques.Most existing authentication approaches are based on cryptographic techniques. However, cryptography has high computational cost, making it unsuitable for the data plane of the network, where potentially every packet needs to be checked at Gigabit per second link rates. This chapter describes a novel design for data plane access control, called OrthCredential. The main idea is to use a set of orthogonal sequences as credentials that can be verified easily to protect the data plane against various attacks. These orthogonal sequences can be constructed by Hadamard matrices. The evaluation shows that OrthCredential only requires less than 300 processor cycles for verification with 64-bit credentials, much less than existing access control schemes such as HMAC. And it provides reasonable security strength (e.g., less than 10^{-8} probability of successful attack). Some of the material in this chapter have been published in [20].

3.1 Introduction

Recently, there has been much interest in the networking community to explore new network architectures for the future Internet [64]. In many proposed architectures, functionalities in the network are viewed as "network services" [28, 47, 82]. These services can be simple paths between nodes, and they can be more complex protocol processing and content storage. Service-centric network architectures then

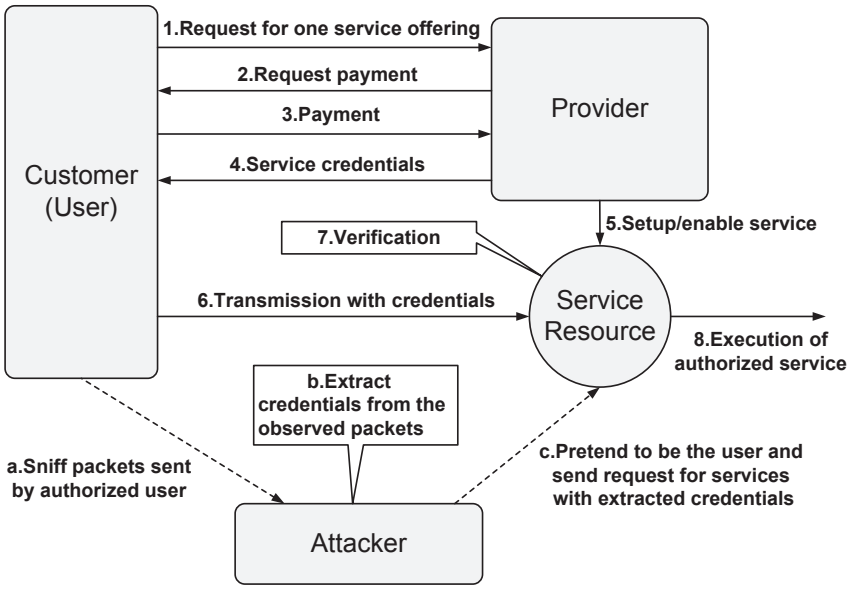


Figure 3.1. An example of interactions between a user and a provider.

describe the semantics of various network services and allow composition of more complex services based on users' needs. To incentivize the deployment of novel services by network providers, it has been envisioned that services can be offered in a marketplace [84]. In this space, users (or their applications) can choose and obtain necessary network services in return for payments to the providers who offer these services. A key technical challenge in such a network is to provide *access control* to these network services (if explicit payments are used or not). Only traffic sent by authorized users (i.e., those who have established an economic relationship with the provider) should be able to access network services that are not offered by default. Therefore, some form of checking must be applied to the traffic before granting them access to reserved resources. Figure 3.1 shows an example of the interactions within a service-oriented network. Once a service contract has established between a user and a provider, the provider delivers some credentials (or "capabilities" [6]) information to the user and sets up the service. These credentials (or seed material to generate multiple credentials) are created by the provider or an entity acting on behalf of the provider. The user attaches a credential to each request for service, which is created

via some method (e.g., cryptographic hash) with the secret information. The verifiers (i.e., edge routers) of providers can validate that a user (or network traffic sent by the user) is authorized for access by validating the credentials. Therefore, only the user who sends packets with the valid credentials can access the services.

It is important to note that, the "network" we consider in this chapter is not a domain-specific network with a limited number of users and dominated by strict security requirements, but is the future Internet with up to billions of users with billions of services. From traditional view, an effective checking mechanism may only require that authorized packets have some property that is hard for attackers to duplicate, while easy for legitimate users to create. However, when considering the common case that millions of packets on a link need to be checked by a router simultaneously, it is critical to develop authentication methods that can be checked with low performance impact, while providing sufficient security from access by unauthorized attackers.

This chapter proposes a novel design for data plane credential called OrthCredential(Orthogonal Credential), which enables access control and can be generated and verified at high data rates with low processing overhead and low storage requirements. The main idea of OrthCredential is that the user uses a sequence (credential) which is orthogonal to the verifier's sequences. And the verifier checks the inner product of the user's credential and the sequence on the verifier. The result of the inner product equals 0 means the credential is valid. These orthogonal sequences can be easily constructed from Hadamard matrices. While designing and enriching access control protocols has received much attention, our focus is on decreasing the cost to satisfy data plane devices' computational capability while guaranteeing an acceptable level of security. The advantage of OrthCredential is in two aspects: 1) the computation of inner product of two binary sequences can be done by fast integer operations on CPU; 2) the verifier only needs to save a few basic orthogonal credentials and a sum of received valid credentials to check the validity of multiple received packets. The

OrthCredential scheme has low verification time since inner product computations are much simpler than cryptographic operations. The main advantages of OrthCredential can be summarized as follows:

- **Low Verification Time:** We use an inner product computation to replace complicated cryptographic operations so that the verification time is significantly decreased (less than 350 clock cycles per packet if requiring less than 10^{-10} attack probability for an attacker to guess a valid credential).
- **Enable Powerful Denial-of-Service (DoS) Attack Defense:** Though verifying a valid credential requires a number of inner product computations between the credential and a set of saved orthogonal sequences, detection of an invalid credential can typically be done in a single inner product computation that requires very few operations (less than 50 clock cycles).
- **Low Memory Consumption in Router:** The verifier (i.e., the network router) only needs to save the sum of received valid credentials and a small part of the orthogonal sequences, which leads to very small storage consumption per flow. In the simulation, we will show that a space consumption of no more than 0.1 KB on the router can promise a random attack probability of less than 10^{-10} while preventing replay attacks simultaneously.
- **Small Overhead in Packets:** OrthCredential header in a packet is small (no more than 28 clock bytes) and thus does not cause significant overhead in packets in the data plane of the network.

3.2 Preliminaries

Before the introduction of OrthCredential system, we begin the discussion with a description of security requirements, attacker capabilities and incapabilities. Then, we list the performance metrics considered in our system.

3.2.1 Security Requirements

This work only considers the problem of the source authentication after a contract has been established between a user and a provider. In 7.1, we assume that the communications in steps 1-2 of the iterations are private and the service credentials (or credential seeds) can not be observed by a third party. This end-to-end security can be achieved by using existing protocols (e.g., TLS). The system must have the following security properties to be considered as a solution to our problem:

3.2.1.1 Security Requirements

The authentication system of access control must have the following security properties to be considered as a solution to our problem:

- Security Strength: In order to provide a secure network infrastructure, it is crucial that credentials are only available to authorized users in the network. Therefore, credentials should be difficult to be guessed or faked. Brute force methods must yield a sufficiently low probability of success that the packets sent by authorized users is unaffected.

- Verification without Trusting Hosts: The verifier should prevent malicious hosts spoofing packets. Authenticity should be determined solely from the packet contents and static per-flow information (e.g., public key or shared secret), and not from any other host information that changes per-packet.

- Replay Prevention: The authentication mechanism should include a means to detect reused credentials. This implies that state must be updated on a per-packet basis, but does not violate the previous requirement, provided the verification and anti-replay checks are separate, and update happens only after verification of authenticity. Given a fixed field size in the packet, this requirement implies that the number of packets that can be verified is bounded; however,

it should be large, so that resynchronization is required infrequently, even for high-data-rate links.

3.2.1.2 Attacker Capabilities

An illegal user, or an *attacker*, is trying to grant the access to some services (by sniffing packets from authorized users and extracting the credentials from the packets, or by some other methods). Figure 3.1 illustrates such attacker's behavior. Such attack may interfere with authorized packets, making the user failed to get the guaranteed service. In some scenarios, this may bring lost revenue to the provider. Our assumptions about the attacker's capabilities can be summarized to the following:

- Ability to eavesdrop at some point along the path from the user to the verifier. Sniffing legitimate packets traveling along the path can be accomplished by breaking into an end system (non-router) connected to a shared-medium network somewhere along the path.

- Ability to extract the credential information within the packets and pretend to be the valid users and transmit the packets under correct formats. The valid credentials can be derived through long-term observation and analysis of the credentials in the authorized packets.

- Ability to send arbitrary packets or flood a particular link, router, or host to which it connects, e.g., Denial of Attacks (DoS). By breaking into and taking control of many end systems within the network, and making them to transmit bursts of packets addressed to some target simultaneously, the attacker can cause packets arriving a verifier at a rate close to the capacity of the channel.

Additionally, we constrain the capabilities of the attacker as follows:

- An attacker does not have access to the secret capabilities materials associated with the credential information between users and providers. As discussed ear-

lier, we assume that the delivery of the secret capabilities materials is through strict encryption.

- An attacker cannot stop the legitimate packets along the path (i.e., cannot drop the network traffic on a router);

- If an attacker transmits a modified copy of an authorized packet, the packet cannot arrive before the original one.

It is conceivable that the above assumptions all might be violated, which Orth-Credential system does not defend against. However, the limitations on the attacker's capabilities are necessary to keep the discussion of attack scenarios and security requirements within scope.

3.2.2 Performance Requirements

Our model of verifier processing implies some performance constraints on the authentication check. We assume that authentication can be pipelined with other operations, and consider only requirements that follow from the basic architecture of the verifier (e.g., router). The performance constraints on the verification process are summarized in the following:

- Verification time: the time spent on verification of an arriving packet is vital for the verifier. Since credentials need to be validated for every packet that arrives in a verifier, they must be verified with low computational requirements. Furthermore, different credential system may perform differently on the verifying time for an invalid credential and valid credential. The second one is also important for the verifier, since there may be floods of DoS attacks within the network.

- Storage consumption: the storage consumption for the credentials is also crucial for the verifiers, since there are maybe millions of users who have the access

to the same service. Moreover, packets may arrive the verifier out-of-order. Verification mechanisms need to allow arbitrary packet order and save related information within a reserved window.

3.3 Related Work and Alternative Solutions

The problem of access control, or message authentication has received extensive attentions. Each proposed approach may hold a particular objective, however, cost is not the major concern of them. Therefore, they are perhaps incapable in a large-scale network with billions of users and services. In this section, we will discuss existing authentication schemes into different categories, and present some possible solutions to our problem and analyze their respective infeasibility.

3.3.1 Ingress Filtering

This mechanism is the early work that investigates DoS attacks, which discards packets that are not actually from the routes or networks that they claim to be from [32]. In [66], Paxson considers the fact that most of the traffic a router sees from a source comes in on the same interface, and use RBF to observe the interface of the packet sources. If it comes from a different interface, it filters the packet. An extended work is [27] by Duan, which proposes IDPF by using BGP information to build a relaxed RBF table. These checks have at least two drawbacks. First, ingress filtering cannot work well when there are asymmetric routes or protected paths in the network. Besides, these schemes are slow to respond to routing changes, and may drop legitimate packets.

3.3.2 IP Traceback

This mechanism is to identify attacks to their source and institute protection measures of the Internet by using routers to create state so that receivers can reconstruct the path of unwanted traffic [70–72]. Some of these mechanisms require

logging per-packet information in routers [71], or save the information in the packets instead [70]. However, this traceback process is a long and difficult process of identifying and punishing an attacker. In [69], a novel packet-marking scheme called Linear Packet Marking (LPM) is proposed which requires the number of packets equal to hop distance between attacker and the victim which is less than 31. It shows that the LPM algorithm performs much better in term of packets required for successful traceback and in handling large-scale DDoS attacks. LPM generates small storage overhead on routers; nevertheless, it also requires a long process of identifying an attacker. In [90], Yu et al. show that accurate traceback is possible within 20 seconds (approximately) in a large-scale attack network with thousands of "zombies".

3.3.3 HMAC/UMAC

The most popular methods for access control is Message Authentication Codes (MAC) (e.g., HMAC-MD5, HMAC-SHA1 and etc.), which usually involves a cryptographic hash function in combination with a secret cryptographic key, providing data integrity and the authentication of data origin. The authentication of data origin gives a confirmation that the message originates by the sender, who shares the used secret key with the receiver. There are a number of algorithms [15, 36, 56, 86] that have been developed in the last decade for the construction of "robust" MAC. UMAC [14] is claimed to be the fastest MAC that has been reported on in the cryptographic literature. Small algorithmic changes were made in 2004 and a number of UMAC options were eliminated for simplicity [50]. Many new Path verification Mechanisms (PVM) also generates proofs or credentials based on HMAC/UMAC (e.g., ICING [62], TVA [89]). However, it is inevitable that cryptographic operations require expensive computations even for small messages: larger than 150 clock cycles per block. Besides, in order to avoid replay packets, each router should keep a size of

window to record received authenticated credentials, which further increase the cost for each flow that the router supports.

3.3.4 Public-Key Cryptography

A message is encrypted with a recipient's public key and it cannot be decrypted by anyone who does not possess the matching private key, who is thus presumed to be the owner of that key and the person associated with the public key [77]. This is a classic method of verifying authenticity (e.g., digital signature). It has the advantage of not requiring any secret information to be distributed to or stored at the verifier. Only the key associated with the flow is required for verification. Unfortunately, even the latest signature verification is too computationally expensive. The arriving packets will trigger credential computations no matter they are valid or not. This increases the system's vulnerability to DoS attacks since encryption or decryption operations require several orders of magnitude more operations than conventional packet processing [21]. Therefore, digital signature cannot meet the performance requirements outlined above.

3.3.5 Hop-by-Hop Message Authentication

Some schemes like HCF [42] and HPPD [92], work by associating sources with hop counts instead of interfaces. The basic principle is that, if a packet comes from a source with a different hop count, the packet is dropped. The advantage of hop-by-hop authentication is that it does not require distribution of any shared secret to verifiers. However, if the attacker can guess the right hop count, spoofing the system would be easy. Besides, the scheme increases cost because the authentication code of each packet is both verified and re-generated at each hop.

3.3.6 Hidden Credentials

There are also many existing work that focuse on carrying out access control with hidden credentials [16, 40, 51]. In these schemes, the access control policies are used as keys to encrypt the data. Only the people who meet the conditions specified in the policies are able to generate the decryption keys. An important work [33] is by Frikken et al., which improves the performance of hidden credential schemes. However, all these schemes have very high running cost due to the complexity of the committed-integer based oblivious transfer protocols. For example, Frikken's scheme needs $O(\rho mn)$ encryption operations and $O(\rho^2 mn)$ communications where m is the number of credentials, n is the number of attributes in a policy, and ρ is the number of bits used to represent the attributes.

3.3.7 Path Verification

There are many routing security protocols [18,25,41,43,68,91] that ensure the authenticity and correctness of the topology propagation and route computation (e.g., S-BGP [43], RBF [68], SCION [91]). These approaches focus on computing the correct routes while they actually do not ensure that the resulting routes are used in packet forwarding. Recently, there are also some working focused on path forwarding and verification, e.g., ICING [62], TVA [89], SIFF [88]. In TVA [89], a source attaches capabilities to each packet, and each router verifies one capability. The capabilities are route-dependent, so if the route changes or multipath forwarding exists, legitimate traffic is dropped. In ICING, Naous et al. design a new networking primitive that enables path consent and compliance perfectly. However, the spoof generation requires encryption on the whole payload of the packet, therefore the verification time will also be large. We think these path forwarding or verification schemes are orthogonal to our OrthCredential mechanism, where low cost authentication for each flow in routers is our focus.

In the next, we analyze another two simple but possible solutions to our problem of source authentication, which helps the understanding of this chapter.

3.3.8 Constant Sequence Credentials

This is the simplest solution, which works as follows: the provider delivers one or a small number of constant credentials to the user, and the user will send packets with one of them each time. This method has a tiny cost on verifier since it needs only to save the constant credentials, and compare the received credential bit by bit when verifying packets. However, once the attacker has observed any valid credential then it can replay the credential and gain access to the services all the time.

3.3.9 Pseudorandom Sequence Credentials

This method uses dynamic credentials and is first described in [21] to prevent replay credentials. After the payment of a user, the provider will inform the user a seed for pseudorandom number generation (PRNG). Each time when the user wants to send packets, it uses the seed to generate a new credential. The same process is done by the verifiers. The potential problem is that in a real network, the packets may arrive out-of-order, which requires a size of window in verifier to save many credentials in advance. Other than that, this system is vulnerable to DoS attacks. Because the verifier will compare the incoming packet with every credential in the window, no matter it is valid or not. Besides, pure random sequence generation for hardware is nearly impossible and attackers are easy to obtain the pseudorandom sequence generation method after a long-term analysis of the packets.

3.4 Overview of OrthCredential

In this section, we describe OrthCredential system in a high level, deferring the design details to Section 3.5.

3.4.1 Goals and Non Goals

OrthCredential is an authentication code intended for use in authorizing access control to reserved network resources to address the requirements in Section 3.2.1.1. OrthCredential is different from conventional MACs that use cryptographic algorithms. It uses orthogonal sequences as credentials and determine validity by observing whether their inner product equals 0. The probability of counterfeiting it is higher than other crypt-schemes, but it is still low enough to be safe. The goal of OrthCredential is to achieve high performance and low cost for verification by eliminating some security guarantees that we have discussed above. It needs to be emphasized that OrthCredential is not designed as a replacement of conventional MAC algorithms. We believe many MAC schemes perform very well in end-to-end data transmission for high security demands, but cannot meet the requirements in a general service-oriented network with billions of services and users.

There are several functions that OrthCredential is not designed for: (i) OrthCredential does not guarantee the security and integrity of the payload in the packet during packet forwarding; (ii) OrthCredential does not guarantee the security of the path that the packet goes on. Actually, a secure path between two nodes can be also seemed as a service and OrthCredential can only provide the access to these secure paths which are set up by provides.

3.4.2 Deployment Scenario

The natural deployment scenario for OrthCredential is at layer 3 (the network layer). Consider a path service with given bandwidth guarantees as an example: The providers would deploy OrthCredential routers at the ingress of their networks (i.e., edge routers). Internal routers do not need to verify OrthCredential, just as internal forwarders may implement a different protocol from that of the edge routers. As shown in Figure 3.1, after the user established a contract with a provider, the

Table 3.1. A full description of the relevant notations in Chapter 3.

Variables	Description
$seed$	The secret information sent to the user and the verifier by the provider.
n	Length of each credential.
k	The index of generated Hadamard matrices ($k = 1,\ 2,\ 3...$).
$H(k)$	The k^{th} orthogonal matrix generated by user ($k = 1,\ 2,\ 3...$).
$h_i(k)$	The i^{th} row of $H(k)$ ($1 \leqslant i \leqslant n$).
$r_i(k)$	The random vector corresponding to $h_i(k)$.
$key(k)$	A secret key corresponding to $H(k)$ which is used to generate $r_i(k)$.
$c_i(k)$	The i^{th} credential and it satisfies $c_i(k) = h_i(k) + r_i(k)$
m	The number of the saved rows of $H(k)$ in the verifier ($1 \leqslant m \leqslant n$).
$H_m(k)$	A number of m rows in $H(k)$ that saved in the verifier (router).
$h_{m,i}(k)$	The i^{th} row of $H_m(k)$ in the verifier ($1 \leqslant i \leqslant m$).
c	The credential extracted from received packets by verifier.
h	The vector computed from c and it is expected to satisfy $h = c - r_i$ for some i.
$counter(k)$	The newest number of the credentials verified as valid, the value of *couter* resets when k is updated.
$sum(k)$	The sum vector of $v_i(k)$ which is saved in the verifier and $sum(k) = \sum_{i=1}^{n} v_i(k)$, the initial value of $sum(k) = 0$.
$sum\_bit[i]$	$sum\_bit[i]$ saves the i^{th} bit of each entry of $sum(k)$ ($1 \leqslant i \leqslant \log_2 n + 1$).

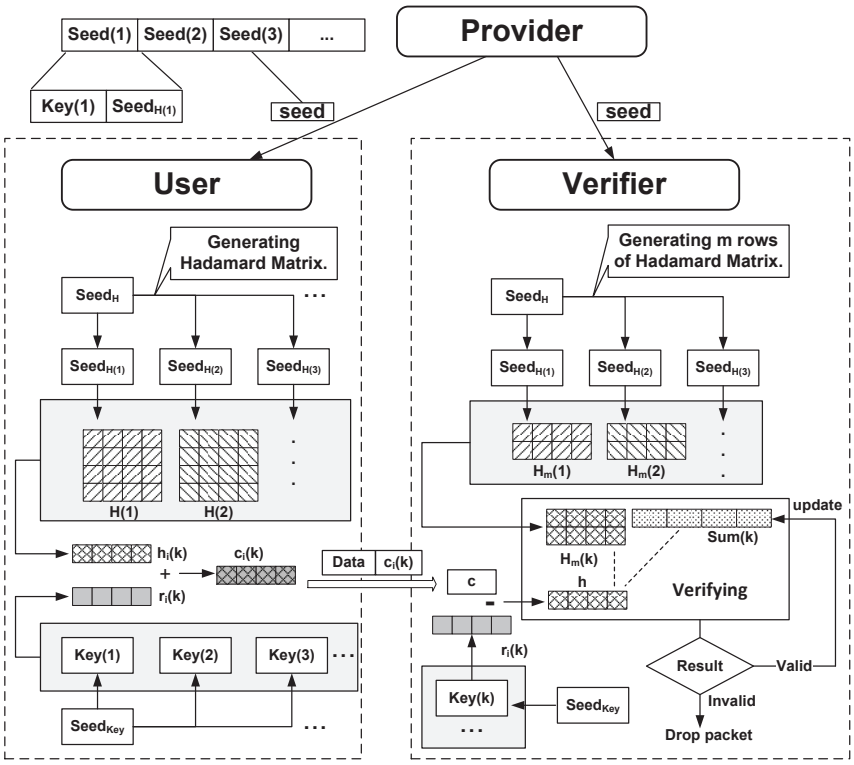


Figure 3.2. Credentials generation and verification in OrthCredential .

provider sets up the service and sends a secret generating seed to both the user and the edge routers along the transmission path. Each time the user uses the path service of the provider, the user sends packets with a credential generated by the secret seed. The edge router of the provider only forwards packets that contain a valid credential. Therefore, only an authorized user can access the bandwidth provided in the path service. The generation and verification mechanism of a credential is discussed further in Section 3.5.

3.4.3 Architecture and Components

OrthCredential is based on the technique of generating a series of sequences with mutual-orthogonal properties known to the user and the verifier. After a user has pursued some service from a provider, the provider will set up the service and send a secret generating seed to both the user and the devices along the transmission path.

The seed contains the secret information of generating different $n \times n$ orthogonal matrices $H(k)$ (k is a numerical order). Besides, the seed also contains the corresponding keys (denoted as $key(k)$) for each $H(k)$. Each time a packet is sent, the user chooses the first unused row vector $h_i(k)$ from $H(k)$ (i is the index of rows in c) and generates a random vector $r_i(k)$ by using $key(k)$, then the user can get a credential by:

$$c_i(k) = h_i(k) + r_i(k), \quad \text{where } k = 1,\ 2,\ 3, ...$$

A similar process is in the verifier: the verifier will generate $H_m(k)$ which includes a number of m rows of $H(k)$, and save it in memory. Once all rows from $H(k)$ are used, $H(k+1)$ will be generated from the seed. For every packet arriving at a router, the router extracts credential c and subtracts the corresponding $r_i(k)$ and results h. Then, h is used to compute the inner product with each $h_{m,i}(k)$, row vector of $H_m(k)$ ($\forall 1 \leqslant i \leqslant m$), to check its validity. Finally, to prevent replay attacks, h is used to compute the inner product with $sum(k)$, the saved sum of the received valid orthogonal credentials, to check if c has been used or not, since a used credential results in a non-zero inner product. If c is verified as valid, then the router adds h into $sum(k)$ to prevent replay attacks with credential c. Figure 3.2 shows the entire process that *OrthCredential* works, where the details will be discussed in Section 3.5. The concise definitions of the notations we used in this chapter can be found in Table 3.1.

The security of OrthCredential is based on a very low probability of an attacker obtaining a credential satisfying that the results of its inner product with each $h_{m,i}(k)$ and $sum(k)$ equal 0 simultaneously. Even for a replay packet, the result of the inner product of h ($h = c - r_i(k)$) and $sum(k)$ equals $||h^2||$ but not 0 either.

In our OrthCredential system, these orthogonal matrices $H(k)$ are Hadamard matrices. In the next section, we introduce the Hadamard matrices and their important properties.

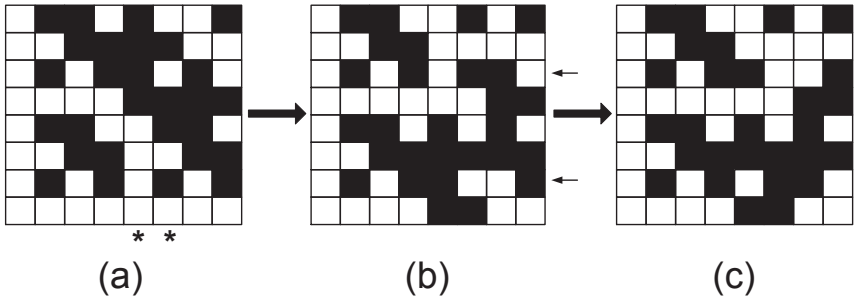


Figure 3.3. An illustration of Hadamard matrix and its properties.

Table 3.2. Number of Hadamard matrices of different types.

n	2	4	8	12	16	20	24	28	32
Types	1	1	1	1	5	3	60	487	13,707,126

3.4.4 Hadamard Matrix

3.4.4.1 Definition

A Hadamard matrix is an $n \times n$ matrix H containing entries from the set $\mathbf{Z_2} = \{-1, 1\}$, with the property of $HH^T = nI_n$.

This equation implies that all distinct rows or columns of a Hadamard matrix are linearly independent. Therefore, all rows or columns of a Hadamard matrix H are mutually orthogonal, i.e., have an inner product of 0. We let (v_1, v_2) denote the inner multiple computation, then vectors v_1 and v_2 are said to be "orthogonal" if and only if $(v_1, v_2) = 0$. For instance, $v_1 = (-1, -1, 1)$ is orthogonal to $v_2 = (-1, 1, 0)$ because $(v_1, v_2) = (-1) \times (-1) + (-1) \times 1 + 1 \times 0 = 0$. In this chapter, we use an "entry" to denote an element in a vector, for instance, each entry in a row of a Hadamard matrix is 1 or -1.

An explanation of a Hadamard matrix can be found in Figure 3.3(a), where black squares represent '-1's and white squares represent '1's. In our OrthCredential system, we use 0 to represent -1 in the Hadamard matrix. Then, the verification of checking whether the result of (c_1, c_2) equals 0 can be simply achieved by checking whether

the number of '0's equals the number of '1's in the result of bitwise AND operation $c_1 \& c_2$.

3.4.4.2 Properties

It is proved that, an $n \times n$ Hadamard matrix exists for $n = 1$, $n = 2$, and $n = 4k$ for any $k \in \mathbb{N}$ [45]. We further introduce a basic, but very important, property of Hadamard matrix:

Theorem 1 *Several operations on Hadamard matrices preserve its property: (i) Row permutation, or changing the sign of rows; (ii) Column permutation, or changing the sign of columns; (iii) Transposition.*

This is illustrated in Figure 3.3: Figure 3.3(a) illustrates an 8×8 Hadamard matrix; in Figure 3.3(b), we change the sign of its fifth and sixth column; in Figure 3.3(c), we further permute the third and seventh row. After these operations, the transformed matrices are still Hadamard matrices. Strictly speaking, two Hadamard matrices H, H' are said to be different if H' cannot be produced from H by operations (i)-(iii). Therefore, we say that there is only one Hadamard matrix of order 2, though it has eight different expressions. Table 3.2 shows the number of inequivalent classes of Hadamard matrices from $n = 2$ to $n = 32$ [45]. When n is larger than 32, the different types of Hadamard matrices are even much larger. These properties enable us to generate a random Hadamard matrix that the attacker cannot guess each time, which also limit the possibility of forging it. Even if the attacker can guess one credential accidentally, which will not be legitimate when the next Hadamard matrix is generated.

3.5 Design Details of OrthCredential

This section details OrthCredential's design, which aims to meet the requirements stated in Section 4.3.

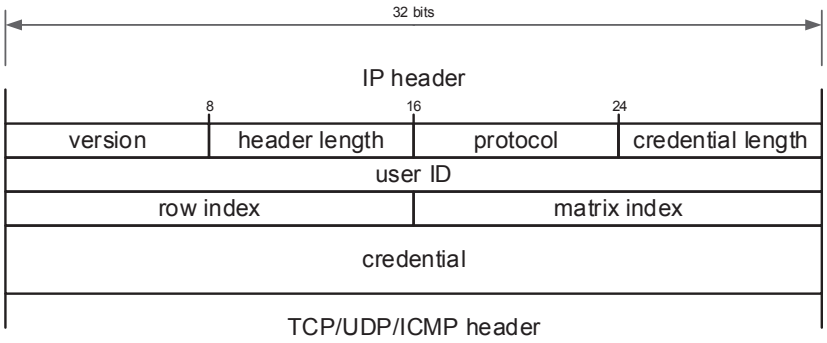


Figure 3.4. OrthCredential header, which total overhead in a packet is 16 to 28 bytes.

Table 3.1 describes the notation used throughout our design discussion and our pseudocode, and Figure 4.3 shows the OrthCredential header format. The header includes two types of information for each user. The first is the user's information, the provider will assign a unique ID for each user who has established a contract with. The verifier will turn to the corresponding verifying materials by checking this ID information. The second is the information used for verification: $\{n, i, k, c_i(k)\}$. The length of $c_i(k)$ (i.e., n) in our current system is 32, 64 or 128 bits. The verifier will tell the difference by checking n or the "header length" part.

3.5.1 Creating Credentials

As described in Section 3.4.3, a credential $c_i(k)$ is given by $c_i(k) = h_i(k) + r_i(k)$. We generate $r_i(k)$ by using a random() function with the seed $(key(k), k, i)$. Each $r_i(k)$ is different due to different sets of $(key(k), k, i)$. The reason why we add $r_i(k)$ with $h_i(k)$ in OrthCredential system is, if $h_i(k)$ can be easily observed by an attacker, the attacker can get a valid credential by generating lots of sequences that are orthogonal to $h_i(k)$. Therefore, $r_i(k)$ can help decrease the probability for attackers to forge a valid credential. We next describe how to construct an Hadamard matrix and thus get $h_i(k)$.

People have derived many construction methods to generate a Hadamard matrix for a given n [8]. The Hadamard construction method in our system is a simple, but efficient one - *Kronecker Product Construction*: if S, T are matrices, their Kronecker Product $S \otimes T$ is the matrix U constructed by replacing each $S_{i,j}$ in S by $S_{i,j}T$. It can be proved that the Kronecker Product $H_n \otimes H_m$ is a Hadamard matrix of order nm if H_n, H_m are Hadamard Matrices of orders n and m. This implies that we can get a $2n \otimes 2n$ Hadamard matrix by the product of an $n \times n$ Hadamard matrix with the basic 2×2 Hadamard matrix of

$$\begin{pmatrix} +1 & +1 \\ +1 & -1 \end{pmatrix}.$$

An illustration of Hadamard matrix construction in OrthCredential system by Kronecker Product is in Figure 3.5. Before doing Kronecker Product operation, the original $n \times n$ and the basic 2×2 Hadamard matrices will take several transform operations in Theorem 1 first, respectively. Then, the generated Hadamard matrix will also take several transform operations. All these transform operations are described in *seed* sent by the provider. For the original $n \times n$ Hadamard matrix, we use a Walsh-Hadamard transform to get it. For instance, in Figure 3.3(a), if we let the bottom row as 0^{th} row and the leftmost column as the 0^{th} column, then its $(i,j)^{\text{th}}$ entry $H_{i,j}$ can be written as $H_{i,j} = (-1)^{\sum_{p=1}^{n}(i_p \cdot j_p)}$, where $i = \sum_{p=1}^{n} i_p 2^p$ and $j = \sum_{p=1}^{n} j_p 2^p$. The Walsh-Hadamard transform is easy to be realized, since the computation of $\sum_{p=1}^{n}(i_p \cdot j_p)$ can be simply achieved by checking the number of '1's in the result of the bitwise operation i AND j. In real implementations, the verifier will save a number of basic Hadamard matrices beforehand, since they may be used for each flow's verification. In Section 3.6.1.1, we will show that this generation method can guarantee a very low repeating probability.

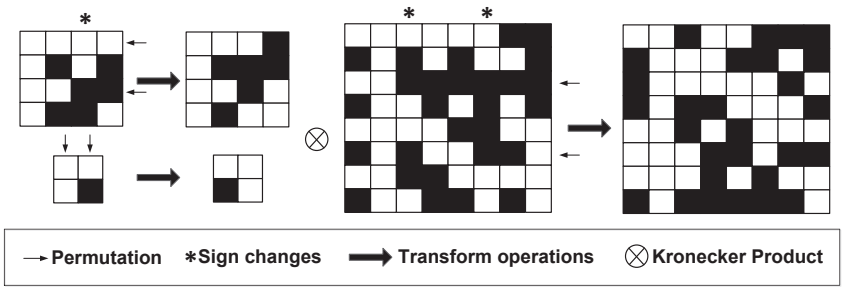


Figure 3.5. An illustration of an 8×8 Hadamard construction.

Each time a packet is sent, the user chooses an unused row vector $c_i(k)$ as the credential and places it together with its index k, i in the packet. Once all rows from $H(k)$ are used, $H(k+1)$ will be generated from the secret seed. We set the maximum k to 100, and the user will ask for a new seed from the provider when a number of k Hadamard matrices are used. Each n-bit credential $c_i(k)$ ($c_i(k) = h_i(k) + r_i(k)$) satisfies the following equations:

$$\begin{cases} \big(h_i(k),\ h_j(k)\big) = 0, & \text{when } i \neq j, \\ \big(h_i(k),\ h_j(k)\big) \neq 0, & \text{when } i = j. \end{cases} \tag{3.1}$$

3.5.2 Verifying Credentials

Algorithm 5 Operations of generating a credential

```
if i%(n + 1) = 0 then
    k← k + 1, i ← 1
    key(k) ← seed(k).key
    H_1 ← Hadamard_Transform(n/2)
    H_2 ← Hadamard_Transform(2)
    H_1, H_2 ← Transform_Operation(H_1, H_2, seed(k))
    H(k) ← Kronecker_Product(H_1, H_2)
    H(k) ← Transform_Operation(H(k), seed(k))
h_i(k) ← the i^th row of H(k)
r_i(k) ← Random(key(k), k, i)
c_i(k) ← h_i(k) + r_i(k)
return (c_i(k), k, i)
```

The verifier only generates and saves a number of m rows of $H(k)$ (i.e., $H_m(k)$), not the whole matrix. In an implementation, m is ususally $10\% - 30\%$ of n. The value of m depends on the security requirements of the service, and if the verifier saves the whole $H(k)$ (i.e., $m = n$) then it can achieve the best protection against forgery attacks. In Section 3.6, we will show that even a small m can also provide a very good protection for the access to the service.

For each packet arriving at provider devices, the verifier extracts credential c, then subtracts the corresponding row r_i and finally gets h. Then, h is used to compare with or compute the inner product with $h_{m,i}(k)$ to verify the validity of c. To avoid replay attacks, h is also used to compute the inner product with $sum(k)$ to check if c has been used or not, since a used credential will result in a non-zero inner product. A received credential c is verified as valid when the following conditions are satisfied:

$$\begin{aligned} &1.\quad \exists\ i\ (1 \leqslant i \leqslant m),\ h = h_{m,i}(k)\ \ \text{or} \\ &\qquad \forall\ i\ (1 \leqslant i \leqslant m),\ \big(h,\ h_{m,i}(k)\big) = 0; \end{aligned} \tag{3.2}$$

$$2.\quad \big(h,\ sum(k)\big) = 0. \tag{3.3}$$

If h satisfies the above equations (i.e., c is a valid credential), the local variable $couter(k)$ is incremented. Here, $couter(k)$ implies that h is the $(couter(k))^{\mathbf{th}}$ valid credential received by the verifier. In order to protect against a replay attack with credential c, the verifier will add h with $sum(k)$ and save the result as a new $sum(k)$. It must be emphasized that: (i) the addition here is between two vectors not two numbers; (ii) though we use 0 represent -1 in h during verification (as described in Section 3.4.4), but during this addition operation, h should be the original vector with entries 1 or -1. Thus, the value of each entry of $sum(k)$ is an integer in the range of $[-n, n]$. We next explain the verification details during real implementations:

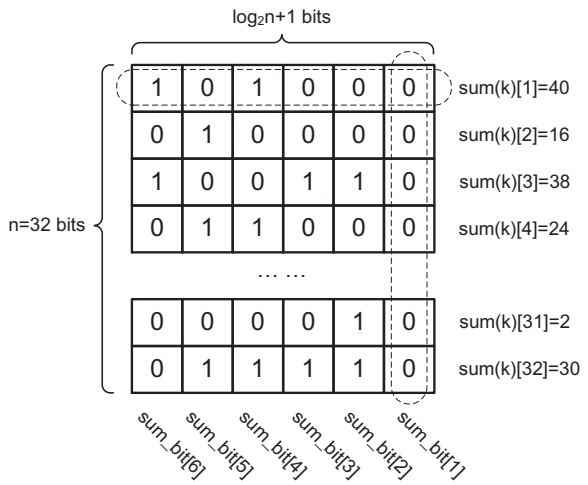


Figure 3.6. An example which shows the relations between $sum(k)$ and $\{sum\_bit[i]\}$ when $n = 32$.

Step 1: Verify Equation (3.2). The operations of verifying whether the result of $(h,\ h_{m,i}(k))$ equals 0 can be achieved by checking whether the number of '0's equals the number of '1's in the result of bitwise operation h AND $h_{m,i}(k)$.

Step 2: Verify Equation (3.3). The cost of the verification process mainly depends on the operations between h and $sum(k)$ when m is small. An intuitive solution would be represent $sum(k)$ as an array with n integers which are in the range of $[-n, n]$, and use n iterations to calculate the inner product. However, we can use the following schemes to decrease the cost to $O(log_2 n)$:

- Remove negatives in $sum(k)$: during the addition operation between h and $sum(k)$, we update $sum(k)$ with the result of $sum(k) + h + e_n$, where $e_n = (111...11)_n$. Therefore, the value range of each entry in $sum(k)$ becomes $[0, 2n]$.

- Recompose $sum(k)$: we use a set of $\{sum\_bit[i]\}$ to represent $sum(k)$, where $1 \leqslant i \leqslant \log_2 n + 1$. Each $sum\_bit[i]$ is an n-bit local variable in the verifier and the j^{th} bit of $sum\_bit[i]$ denotes the i^{th} bit of the j^{th} entry of $sum(k)$. Figure 3.6 is an example which shows the relations between $sum(k)$ and $sum\_bit[i]$ when $n = 32$. From the above discussion, we know that, each time $sum(k)$ updates by adding the result of $h + e_n$ whose each entry is 2 or 0, therefore each entry

of $sum\_bit[1]$ is always 0. In real implementations, the verifier does not save $sum\_bit[1]$ to decrease the storage consumption.

Algorithm 6 Operations of verifying a credential

$r_i(k) \leftarrow \text{Random}(key(k), k, i)$
$h \leftarrow c - r_i(k)$
▷ Step 1: Verify $h = h_{m,i}(k)$ or $(h, h_{m,i}(k)) = 0$
for $1 \leqslant i \leqslant m$ **do**
    if $h = h_{m,i}(k)$ **then**
        break
    else
        $result \leftarrow$ number of 1s in $(h \oplus h_{m,i}(k))$
        if $result \neq n/2$ **then**
            return InValid
▷ Step 2: Verify $(h, sum(k)) = 0$
$result \leftarrow 0$
$number\_h_0 \leftarrow$ number of 0s in h
$number\_h_1 \leftarrow$ number of 1s in h
for $1 \leqslant i \leqslant \log_2 n + 1$ **do**
    $number_0 \leftarrow$ number of '0's in $(\bar{h}$ & $sum\_bit[i])$
    $number_1 \leftarrow$ number of '1's in $(h$ & $sum\_bit[i])$
    $result \leftarrow result + 2^{i-1}(number_1 - number_0)$
if $result \neq couter(k) \cdot (number\_h_1 - number\_h_0)$ **then**
    return InValid
▷ Step 3: Update $sum(k)$
for $2 \leqslant i \leqslant \log_2 n + 1$ **do**
    if $i = 2$ **then**
        $carry \leftarrow h$
    else
        $temp \leftarrow carry \oplus sum\_bit[i]$
        $carry \leftarrow carry$ & $sum\_bit[i]$
        $sum\_bit[i] \leftarrow temp$
$couter(k) \leftarrow couter(k) + 1;$
return Valid

By the above schemes, Equation (3.3) could be written as:

$$couter(k) \cdot (h,\ e_n) = \sum_{i=1}^{log_2 n+1} 2^{i-1} \cdot (h,\ sum\_bit[i]).$$

The computations of inner product in the above equation can be realized by simple bitwise operations as the same as computing $(h,\ h_{m,i}(k))$. Therefore, we reduce the processing time of verifying Equation (3.3) from $O(n)$ to $O(\log_2 n)$.

Step 3: Update sum(k). An intuitive solution of updating $sum(k)$ with the result of $sum(k) + h$ would also need n iterations to accumulate the sum of each entry in $sum(k)$ and the corresponding entry (1 or -1) in h. However, with the above representation of sum_bits, we can also decease the processing time of updating $sum(k)$ from $O(n)$ to $O(\log_2 n)$. As described above, we actually update $sum(k)$ with $sum(k) + h + e_n$, where each entry of the result of $h + e_n$ is 2 or 0. Therefore, in real implementations, we will let each entry of $sum\_bit[2]$ plus 1 if the corresponding entry of this result is 2. If the addition on a entry of $sum\_bit[2]$ produces a carry, then reset this entry and transmit a carry to the corresponding entry of $sum\_bit[3]$, and so forth.

In addition to the verifying operations in Step 1, the whole verifying time (count in clock cycles) is in a scale of $O(m + \log_2 n)$. In Section 3.6, we will present the running time under real implementations. The pseudo codes of these algorithms for generation and verification of a credential are shown in Algorithms 5 and 6.

3.5.3 Attacks

The security of OrthCredential is based on a very low probability of an attacker to obtain credentials that satisfies Equations (3.2) and (3.3) simultaneously. We briefly analyze how OrthCredential counters various threats.

An attacker may try to obtain credentials by brute force. We use a part of n mutually orthogonal vectors (i.e., $H_m(k)$) to make these "grope around" attacks impossible. Though the probability that a random vector is (with entries "-1" or "1") orthogonal to $h_{m,i}(k)$ cannot be concluded theoretically, we verify the very low probability for random successful attack by experiments. In our implementation of

OrthCredential system, this random attack successful probability would be even lower because a valid credential also has to be orthogonal with $sum(k)$. The results of the experiments are shown in Section 3.6.

An attacker may try to obtain credentials by replaying the valid credentials sent by the user. We use a $sum(k)$ to perfectly prevent this kind of attack. It works since once h is verified to be valid, then it will be added into $sum(k)$. Thus, we have $(h, sum(k)) = (h, h) = |h|^2 \neq 0$. If h is expired for k, h will be also verified to be invalid due to a very low probability that h is orthogonal to each new $h_{m,i}(k)$.

An attacker may try to obtain credentials by generating Hadamard matrices. OrthCredential provides double protection against this attack. First, as discussed in Section 3.5.1, the properties of Hadamard matrices guarantee a very low probability that an attacker generates the same matrix as the user's. Besides, we use a dynamic random vector $r_i(k)$ to "encrypt" each row of the generated Hadamard matrix. If the attacker obtains the information of some $r_i(k)$ of a user, the attacker can derive $h_i(k)$ by observing the packets sent by the user, it is still very far away from breaking through the verifier. This is because $h_i(k)$ cannot be used directly, the attacker needs to generate vectors that are orthogonal to $h_i(k)$. However, it is still a very low probability that these generated vectors are orthogonal with all the $h_{m,i}(k)$ in the verifier.

An attacker may eavesdrop the communications between the user and the provider. Each time when the credentials generated from *seed* are run out of, the user will ask its service provider for a new credential seed. In OrthCredential system, any end-to-end traffic between the user and provider will be encrypted using existing protocols (e.g., TLS/SSL). There is no way for an attacker to obtain secret information by this behavior.

There are also other attacks described in Section 3.2.1.2 that are out of our assumption of the attackers' incapabilities and OrthCredential cannot defend against. Actually, it's a future work we will consider in OrthCredential.

3.6 Evaluation

In this section, we evaluate the performance of security, verification time and storage consumptions in OrthCredential. We have implemented the algorithm in C++. We use a PC with an Intel Core2 Quad CPU Q9400 running at 2.66GHz to test the algorithm's performance. The operating system is Ubuntu 14.04 64-bit with kernel version 3.13.0-24 and gcc version 4.8.2. Both the credential generation and verification codes are compiled in one program with gcc -O3 optimize level. We do not use platform specific instructions and assembly codes. The time consumed by each step is measured by CPU clock cycles.

3.6.1 Performance Evaluation

Now, we evaluate the OrthCredential system's performance on its security, verification latency and storage consumption, respectively.

3.6.1.1 Security

We first run simulations to ensure a very low probability of generating a same Hadamard matrix by the construction method which is described in Section 3.5.1. During the generating process, we take 3 basic transform operations each time (permutation, changing signs or transposition). We generate $100,000$ Hadamard matrices for $n = 32$, 64 and 128 and find the number of the same generated Hadamard matrices among them. The repeatability result is that, when $n = 32$, the repeatability is 0.18% and it is nearly 0 when $n = 64$ or 128. Considering that a dynamic random vector $r_i(k)$ will also "encrypt" each row of the Hadamard matrix, we believe that it is impossible for an attacker to guess a valid credential.

We then simulate a scenario that an attacker sends random credentials to brute force the verification process. As discussed in Section 4.5.3, the first step of verifying a valid credential is to determine whether this credential is orthogonal with each $h_{m,i}(k)$ while a random credential is hard to achieve this. Figure 3.7 shows the success probability of such random attack. It can be observed that, when the credential length (i.e., n) is 128, $m > 10$ can guarantee the breakthrough probability less than 10^{-9}, which can be considered safe enough. We can get that, even if the packet transmission rate of an attacker is as high as 1M pkt/s, then it still needs nearly an hour to guess a valid credential. Besides, in reality, the $H_{m,t}(k)$ may change many times during an hour. For some services with low-security requirements, we consider 32-bit or 64-bit credentials can also be used with a proper chosen m.

We also simulate a scenario that an attacker sends replay packets, the result is that all these replay packets are discarded by the verifier no matter the choose of n and m. There are also other attacks in reality that we cannot simulate, the discussions can be found in Section 3.5.3.

3.6.1.2 Verification Time

In OrthCredential system, we use computations of inner product to replace complicated cryptographic operations so that the verification time is significantly decreased, most of the possible computations are simplified to use the basic bitwise operations.

We have tested the different credential length (i.e., n) and different number of rows of a Hadamard matrix stored on the verifier (i.e., m). Figure 3.9(a) shows the time needed to verify a valid credential. When m is relatively smaller than n, the time increases almost linearly with m. That is because, in this case, the probability that h (calculated from the received credential c) equals a saved $h_{m,i}(k)$ is relatively small, thus a valid user's credential has to be calculated against all the m rows of $H_m(k)$ to get verified. When m approaches n, the verifier has a larger probability of

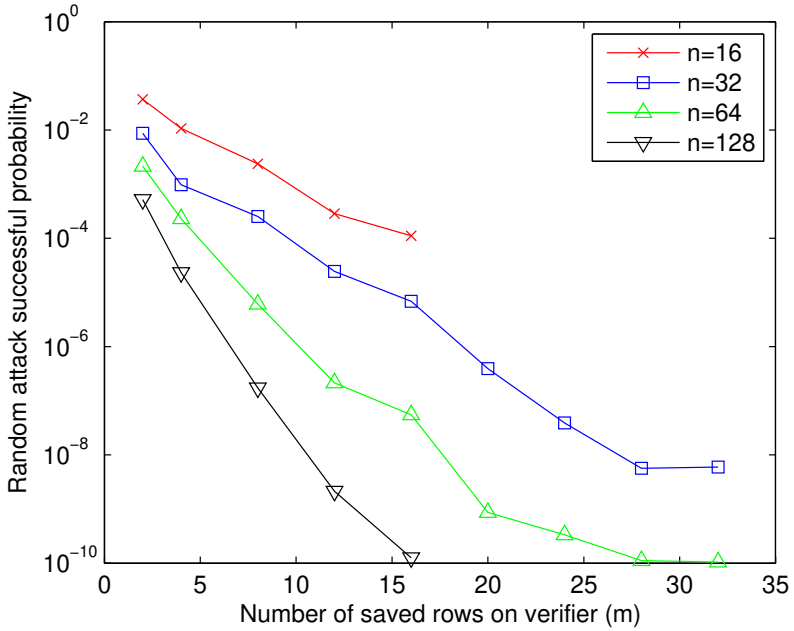


Figure 3.7. Success probability of attacks using random generated credentials.

saving a $h_{m,i}(k)$ that equals h, which makes the verification process jump to the sum verification, therefore the curve becomes gradual.

As discussed in Section 4.5.3, it requires three steps to verify a valid credential, which runs in a scale of $O(m + \log_2 n)$ time. The main cost depends on Step 1 and Step 2 since they both need an operation to count the number of '1's or '0's in a vector (i.e., POPCOUNT operation), while Step 3 only does the basic AND or XOR bitwise operations. 16-bit POPCOUNT is done by looking up twice in a 8-bit lookup table, while 32 and 64-bit POPCOUNT uses a variable-precision SWAR algorithm introduced in [5]. Because the lookup table method is faster, we can see the time for $n = 16$ is almost half of the time when $n = 32$. 32-bit and 64-bit credential cost almost the same time, because for a 64-bit CPU, operating a 64-bit integer is as fast as a 32-bit integer. 128-bit credential needs twice the time of 64-bit credential, because the CPU cannot do 128-bit integer calculation natively, all the calculations must be performed as two 64-bit operations. It is worth noting that although some new Intel x86 CPUs have SSSE3 and SSE4.2 instructions that can do fast POPCOUNT [38],

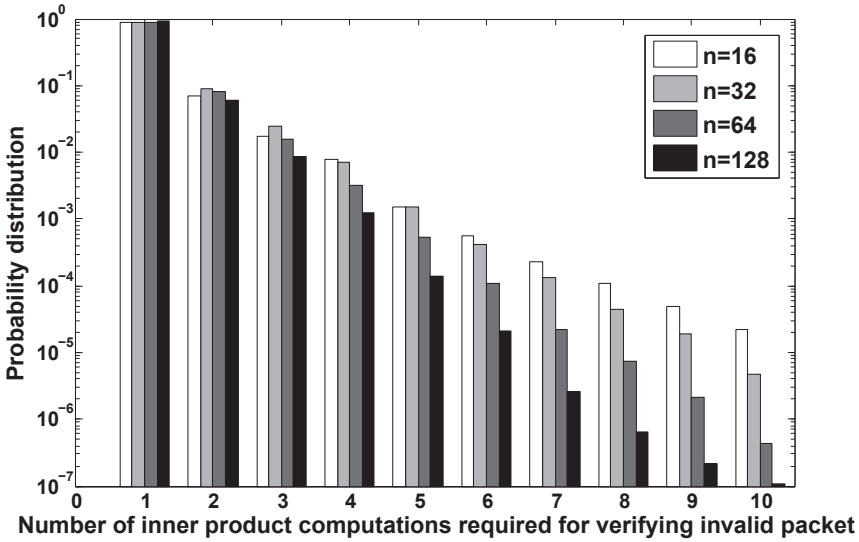
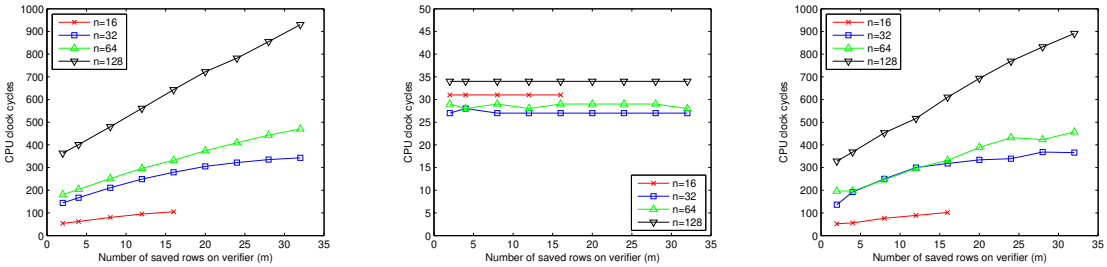


Figure 3.8. Probability distribution of the number of the inner product computations for an invalid packet.

we do not use it because OrthCredential system is platform dependent. If we use it, the speed can be further improved. Figure 3.9(b) shows the average time needed to verify a random attack credential. Theoretically, the verifier can discard the invalid credential within the first few inner product computations in Step 1 due to the low random attack successful probability. Therefore, the time is much less compared to verify a valid credential, and it is also nearly irrelevant to m. When $n = 16$, it takes a longer time to verify a random credential, this is due to the higher random attack successful probability, which will lead to a longer verification process. 3.8 shows the probability distribution of the number of the inner product computations required for a random credential before it is discarded by the verifier.

3.9(c) shows that the replay attack credentials take almost the same verification time as a valid credential, albeit none of them will be verified to be a valid one. This is because a replay credential will not be identified until completing Step 2 – inner product computation with $sum(k)$. While Step 3 only does serval basic AND or XOR bitwise operations that require less than 10 clock cycles to proceed, the error of experiments can cover up this slight difference between 3.9(a) and 3.9(c).



(a) A valid credential. (b) A random attack credential. (c) A replay attack credential.

Figure 3.9. The time for OrthCredential system to verify different types of packets.

Conventional cryptographic schemes will take much more verification time. Table 3.3 shows the experiment results of the average per-packet verification cost for some classical conventional cryptographic algorithms for 500-byte packets (an average packet length in Internet). These algorithms are also implemented in C++ without platform specific instructions and assembly codes. To make more comparisons, we also list the cost of some path verification mechanisms (ICING [62], TVA [89] and DPCP [81]) in the table. By comparing Table 3.3 with Figure 3.9, we can see the enormous advantages on the verification speed, especially considering the case that a DOS attacker floods a path with lots of random attack packets (the verification time of the schemes in Table 3.3 does not change for random attack packets).

3.6.1.3 Storage Consumption

As illustrated in 4.3, the total overhead of OrthCredential header in a packet is 16 to 28 bytes. In OrthCredential, the storage consumption of the verifier for each user depends on two parts: a number of m rows of an $n \times n$ Hadamard matrix and a set of $\{sum\_bit[i]\}$ where $2 \leqslant i \leqslant \log_2 n + 1$. Thus, the whole storage consumption in the verifier is $mn + n \log_2 n$ bits. It must be noted that this storage consumption is under a consideration of preventing replay packets. Many authentication schemes (e.g., HMAC/UMAC, ICING) cannot provide anti-replay protection naturally and have to keep a window to save received authorized credentials. In this point of view,

Table 3.3. Average verification costs of different schemes for 500-byte packets .

Algorithm	Cycles/Packet
HMAC (MD5)	5,335
HMAC (SHA-1)	8,931
AES/CTR (128 bit key)	7,277
DMAC (AES)	12,223
ICING (x-hop)	$2{,}080x + 19{,}520$
TVA (x-hop)	$3{,}264x$
DPCP (512-bit credential)	34,780

our OrthCredential does not need to save all the newest authenticated credentials, but uses a sum of the received authenticated credentials instead, which decreases the storage consumption efficiently. For instance, if OrthCredential uses 64-bit credentials with 5 rows of a 64×64 Hadamard matrix to implement verification, then the router only requires a space of 80 bytes to prevent all possible replay packets while other conventional cryptographic schemes may need a space of 512 bytes to achieve it.

3.6.2 Deployment on GENI

We have deployed OrthCredential on ExoGENI [9] to demonstrate that the protocol can work on a real network. We use Netfilter Queue [3] to modify the packets. The sender inserts a OrthCredential header between IP and TCP header, and sends it to the verifier. The verifier verifies and removes the header, then forwards the packets to the receiver. An attacker can send traffic with random credentials. The topology is shown in Figure 3.10.

We use iperf to generate TCP traffic from both the sender and the attacker to the receiver. The parameters of the TCP traffic are identical. So if the total traffic of the sender and the attacker exceeds 100Mbps, they will compete for the bandwidth to the receiver. The test is divided into 4 scenarios as shown in Table 3.4. When there is no verification and no attacker's traffic, the sender can occupy most of the 100Mbps bandwidth. When the attacker joins, it takes almost half of the 100Mbps

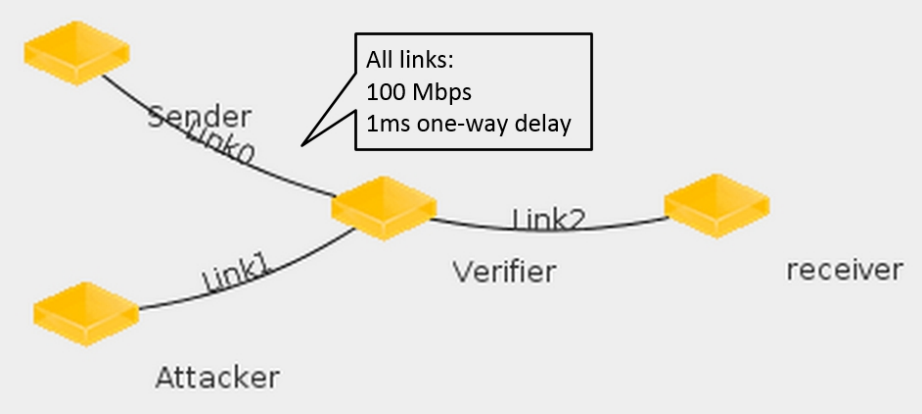


Figure 3.10. Test setup on ExoGENI using flukes.

Table 3.4. Throughput under different scenarios.

scenario	verification	has attacker	throughput	
			sender	attacker
1	disabled	no	89	-
2	disabled	yes	42	49
3	enabled	no	82	-
4	enabled	yes	80	0

bandwidth because of the fairness of TCP. When verification is enabled (with $n = 64$, $m = 5$), the attacker's traffic can not get through, so the sender can get most of the bandwidth again. This proves the access control ability of OrthCredential.

Comparing Scenarios 1 and 3, we can see that enabling or disabling the verification does not have noticeable effect on the sender's throughput. This is because the packet forwarding takes far more clock cycles than the verification process. This demonstrates the low verification time of OrthCredential. Varying n and m also has little impact on the throughput in our experiment.

3.7 Conclusions

This chapter introduces a novel credential design to provide efficient source authentication in the data plane of a network. A new credential design, OrthCredential, is presented to solve the problem of protecting reserved services with very low over-

head in terms of verification time and memory consumption, while guaranteeing good security performance. The prototype implementation has shown a small credential header (e.g., 20 bytes) can be checked in less than 300 processor cycles and require less than 800 bits of memory per flow on a router. We believe that OrthCredential presents an important contribution toward the future service-oriented Internet.

CHAPTER 4

PATH VALIDATION

In the last chapter, we presents OrthCredential, which is a novel technique for the source authentication with very low overhead in routers. This chapter will present Orthogonal Sequence Verification (OSV), which is a great extension of OrthCredential, to achieve both source authentication and path validation simultaneously. OSV also uses orthogonal credentials to enable source authentication and path verification simultaneously. The verification of these orthogonal credentials is based on inner product computations, which can be easily realized by basic bitwise operations in a processor. Therefore, OSV can also significantly reduce computational cost comparing the conventional cryptographic approaches. We present evaluation results which show that OSV is three orders of magnitude faster than the current approaches based on cryptographic operations. Therefore, we believe that our work presents an important contribution toward realizing high-performance, secure network protocols and network attack defenses in practice. Some of the material in this chapter have been published in [19].

4.1 Introduction

Source authentication and path validation are two important concepts in networking that help construct higher-level security mechanisms, such as mitigating denial-of-service (DoS) attack, ensuring path compliance and packet attribution, and protecting against flow redirection [48]. Source authentication is the verification of the source address of a host that sends a packet and is designed to determine whether

this packet indeed originated from the claimed source. Path validation, in particular, is to validate that the packet indeed traversed the path known to (or selected by) the host (i.e., the source). The latter is used when senders, receivers or operators would like to ensure that a packet's path adheres to their preferences. For example, an enterprise might want to dictate that incoming traffic passes through certain services, such as deep packet inspection [53]. Path validation provides a way to enforce this path compliance according to the policies of ISPs, enterprises, and data centers.

The current Internet, however, is unable to provide efficient means for source authentication and path validation by routers or end-hosts. For example, a network provider cannot determine if traffic is sent by neighboring providers along paths that match service-level agreements; a receiver cannot be sure whether a packet is from a specific source, since an attacker can spoof source addresses in packets. Widely used end-to-end encryption and authentication schemes (e.g., TLS/SSL) are not able to solve these issues, since they are agnostic to which path the packets have been forwarded on. A stronger approach is needed, which enables routers and destinations to perform source authentication and path validation.

In recent work, various methods have been proposed for source authentication (such as TVA [89], SNAPP [65], StopIt [54], AITF [7], NetFence [55], Passport [62], DPCP [81], and OrthCredential [20]), which cannot be used for path validation. Current proposed approaches for a combination of source authentication and path validation are ICING [53] and OPT [48]. However, they are both based on expensive cryptographic operations that require considerable computational resources. To make source authentication and path validation feasible for broad deployment, it is critical to develop methods that have low implementation cost while still providing reasonable security guarantees.

In this chapter, we present a novel technique for both source authentication and path verification, called Orthogonal Sequence Verification (OSV). OSV uses capabili-

ties, i.e., special tokens in the packet, that are checked by routers along the path of a flow. The main idea of OSV is that the sender and routers use "orthogonal sequences" of suitably chosen +1 and -1 values. The sender uses its orthogonal sequence as a credential that identifies the source, and routers add their own orthogonal sequence as credentials that capture which nodes a packet has traversed. The routers and destination check the result of the inner product of the sender's credential and their saved orthogonal sequences. A valid result (i.e., inner product of zero) indicates that the packet is from the claimed sender. Similarly, downstream routers and the destination can verify whether the packet has been forwarded by each of the nodes along the determined path based on the orthogonal sequence information that is incrementally updated in the packet. This technique provides an efficient and effective means to achieve source authentication and path verification. The main advantages of OSP can be summarized as follows:

- Low Verification Time: We use an inner product computation to replace complicated cryptographic operations so that the verification time is significantly decreased. The verification time of OSV is less than 0.01μs while ICING is usually needs more than 24.4μs. OPT could achieve the same verification time as OSV, but it needs at least 4ms for the key setup process beforehand.

- Low Packet Overhead: OSV header in a packet is small (about 125 bytes). ICING and OPT's header size are nearly the same as OSV's when the path length is below 4, but they increase rapidly as the path length increases. E.g., when the path length is 10, OSV's packet header is 128 bytes while ICING's and OPT's are 410 bytes and 196 bytes, respectively.

- Independence of Path Length: the properties of orthogonal sequences provides the advantage of OSV that the packet processing time in routers is almost

independent of the number of hops that are traversed by the packet. OSV's packet header overhead is also almost constant (2 bytes for each one more hop).

4.2 Related Work

Source authentication has been studied extensively, the related work on source authentication are discussed in Section 3.3.

Current techniques for both path validation and compliance have been proposed in ICING [53] and OPT [48]. In ICING, the source pre-computes a MAC, V_i, for each intermediate router R_i. When the packet arrives at router R_i, the router first reconstructs the MACs for the source and each upstream router using the secret keys shared with the source and each router and verifies whether the calculated results are equivalent to each V_i. Then, R_i computes new MACs for each downstream router using the shared secret key and updates each V_j ($j \geq i$). OPT designs a Dynamically Recreatable Key (DRKey) protocol that enables routers to create or recreate symmetric keys shared with end-hosts on the fly. OPT does not require routers to store keys shared with sources or other routers, nor perform a MAC computation for every router on the path. Therefore, this protocol requires each router R_i to compute much less MAC operation per packet compared to ICING. However, the key setup process of OPT will take a very long time instead (e.g., about 20ms processing time in destination node for an 8-hop path). Our evaluation in Section 4.6 shows that our proposed OSV can significantly reduce the processing latency for packets at the source, intermediate routers, and the destination.

4.3 Preliminaries

Before we present the design of OSV in Section 4.4, we first describe the design goals, our security model, and limitations.

4.3.1 Goals

OSV is designed to achieve both source authentication and path validation with very low overhead on routers, while guaranteeing an acceptable level of security. The destination and each intermediate router should be able to determine whether the packet indeed originated from the claimed source and traversed a specific path. While many cryptography-based techniques perform very well in terms of security, they do not meet the performance requirements for environments where source authentication and path validation are performed for every flow (e.g., service-oriented network with millions of services and billions of users [85]).

4.3.2 Security Model

In order to provide a rigorous discussion of the security properties of OSV, we now define the following security requirements and attacker capabilities.

4.3.2.1 Security Requirements

Any system that aims to achieve source authentication and path validation must have the following security properties to be considered as a solution to this problem:

1. Credential strength: Valid credentials (or capabilities) in packets can only be generated by an authorized sender in the network. Therefore, credentials must be based on secret information and be difficult to guess or fake. Brute force methods must yield a sufficiently low probability of success that they are not useful in practice.

2. Verification without trust chains: An entity verifying a packet must be able to do so without having to trust other nodes that have been visited along the path. Necessarily, the receiver needs to trust the sender since secret key material is exchanged. However, routers along the path do not need to trust other routers (and the receiver does not need to trust other routers). Authenticity should be

determined solely from the packet headers and local per-flow information (e.g., public key or shared secret).

3. Identification of denial-of-service (DoS) attack source: Sources of DoS attacks should be identifiable in order to isolate the attack.

4.3.2.2 Attacker Capabilities

The capabilities[1] of attackers that aim to disrupt the source authentication and path validation system are assumed to be following:

1. Ability to eavesdrop at any point along the path from the source to the destination. Sniffing legitimate packets traveling along the path can be accomplished by hacking into an end-host or router or by observing a shared-medium network link along the path.

2. Ability to send arbitrary packets or flood a particular link, router, or host. This attack can aim to create a DoS attack, e.g., using a botnet. A packet replay attack is a special case of this kind of packet injection that also utilizes the previously described capability.

3. Ability to cause path deviation, which leads to packets being forwarded along a path other than the path previously determined. For example, an attack can control an intermediate router and redirect the packet to skip other routers in the path.

4. Awareness of OSV operation. An attacker may be aware of the computation process of OSV such as the credential generation and verification.

Additionally, we constrain the capabilities of the attacker as follows:

[1]Unfortunately, the term "capabilities" is used for two different concepts: tokens carried in packets and abilities available to an entity. In this case, we refer to the latter.

5. An attacker does not have access to the secrets associated with the credential information between users and providers. We assume that the delivery of secret key material is done through established cryptographic protocols.

6. An attacker cannot stop legitimate packets along the path. Techniques for avoiding such black hole attacks are beyond the scope of this chapter and are addressed in related work.

7. If an attacker transmits a modified copy of an authorized packet, this packet cannot arrive before the original one. This ordering necessary to ensure that when detecting the replay of a valid credential, the original packet can be identified by the earlier arrival time. This constraint is typically easy to achieve since an attacker may require additional computation and potentially routing detours to create and deliver modified packets with replayed credentials.

In Section 4.6.1, we discuss how the proposed OSV achieves the stated security requirements.

4.3.3 Limitations

While OSV is effective for source authentication and path validation, there are several functions OSV is not designed for:

1. OSV does not guarantee the security and integrity of the packet during forwarding; After a packet departs a node, any downstream node can send it anywhere. It seems extremely hard to prevent such misbehavior. However, OSV can contain it: honest nodes drop packets that do not contain a proof of having passed through every intended prior node on the path.

2. OSV does not provide authenticated information about the location of silent errors or failures on the path. Intermediate routers or the destination can send

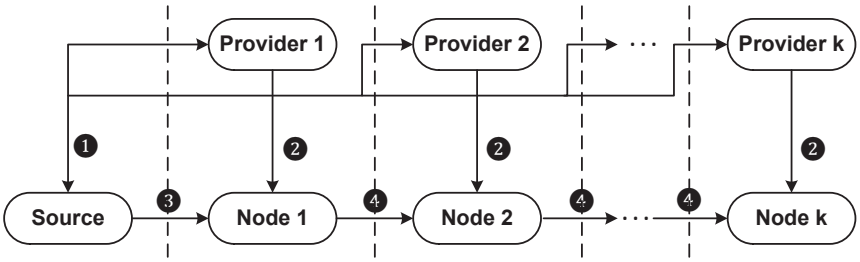


Figure 4.1. OSV's forwarding process.

an error message to the source, but the source cannot discover where this error happened if none of the upstream routers initiates the message.

3. OSV does not provide information about whether a packet received any contracted-for service at a node. For instance, if a sender chooses to send a packet through a particular node because the node advertised a virus-scanning service, the receiver can check if the packet was forwarded through the node, but cannot verify that the packet was actually scanned.

4. OSV nodes can copy packets and send them elsewhere, or pass packets through hidden nodes. This, too, seems very hard to prevent in a federated environment. However, unlike in the status quo, OSV senders and receivers can restrict the path to providers that they trust not to leak their packets.

4.4 OSV Overview

In this section, we describe OSV system in general terms, deferring some design details to Section 4.5.

OSV is a mechanism that allows each node on a selected path to verify that a packet indeed originated from the claimed source and followed the selected path. A deployment scenario for OSV is to implement this verification during layer 3 forwarding on OSV nodes at the ingress point of a provider's network. Internal routers (or forwarders) do not need to run OSV, just as the internal routers may implement a

Table 4.1. A full description of the relevant notations in Chapter 4.

Symbols	Description
k	The path length.
n	The length of the credentials, currently we choose n=128.
S	The source node.
N_i	The nodes in the path, including the intermediate routers and the destination. $(1 \leq i \leq k)$
c	The credential in the forwarded packet which is generated by S.
PVF_i	Field in the packet enabling N_i to verify the traversed path. PVF_i is updated when the packet is forwarded on each intermediate node.
OV_i	Field in the packet enabling N_i to verify the traversed path. OV_i is generated by S and remains the same during the forwarding process.
sum_i	A sum vector of the valid credentials c received by N_i to prevent replay attacks. The initial value of sum_i is 0.
H	The selected $n \times n$ Hadamard matrix.
m_S	The number of rows (i.e., h_S^j) saved in S.
m_i	The number of rows (i.e., $h_{N_i}^j$) saved in R_i.
K_S	Local secret of S.
K_{N_i}	Local secret of N_i.
$F(\cdot)$	Pseudorandom function.
h_S^j	The j^{th} orthogonal sequence in $\{h_S^1, ..., h_S^{m_S}\}$ that is saved in S. h_S^j is used to generate c, PVF_i and OV_i. $(1 \leq j \leq n)$.
$h_{N_i}^j$	The j^{th} orthogonal sequence in $\{h_{N_i}^1, ..., h_{N_i}^{m_i}\}$ that is saved in N_i. $h_{N_i}^j$ is used to both verify the packet source and path.
r_S^j	A pseudorandom sequence corresponding to the j^{th} orthogonal sequence in $\{h_S^1, ..., h_S^{m_S}\}$ using secret K_S, where $r_S^j = F(K_S \vert j)$.
$r_{N_i}^j$	A pseudorandom sequence corresponding to the j^{th} orthogonal sequence in $\{h_S^1, ..., h_S^{m_S}\}$ using secret K_{N_i}, where $r_{N_i}^j = F(K_{N_i} \vert j)$.

different protocol from that of the border router. In this chapter, each provider or node along the path is assumed that they do not trust each other, therefore they do not share secrets.

To communicate with a destination, a source node S chooses the path that it wants. In this chapter, we do not discuss how S finds that path since our focus is on the verification. The forwarding process of OSV is shown in Figure 4.1. We divide this process into 4 steps, which are described in the following. The notation we use in this chapter is summarized in Table 4.1.

After determining the path, the source node S first communicates with the providers of the nodes along the path to deliver the credential information (Step 1 in Figure 4.1). In OSV system, the credential information that is used for verification is generated by the source node S. During the communication process in Step 1, source node S will deliver different credential information to each node N_i, respectively. This credential information is based on the generation of a Hadamard matrix, which was introduced in Section 3.4.4. In Section 4.5.1, we describe what is the credential information and how it is generated and delivered to the providers. The generated credentials certify the provider's consent to carry packets along that path. In addition, S uses the credential information to construct the Path Validation Field (PVF) in the packet, which enables the nodes in the path to verify that the correct path has been followed so far. During this communication process, the providers also get necessary information for verification, which they pass on to their respective OSV nodes (Step 2 in Figure 4.1).

Before each packet's transmission, S uses the credential information to construct the verification fields in the packet header (Step 3 in Figure 4.1). The packet header format is shown in Figure 4.3 (the explanation of the lengths of PVF and OV_i are in Section 4.6). Credential c is used for source authentication, PVF and OV_i (Original Validation) are used for path validation.

When receiving a packet, node N_i performs the following verification steps (Step 4 in Figure 4.1):

- Source authentication: Verify credential c to check whether the packet was sent from S;
- Path Validation: Verify PVF_i and OV_i to check whether the packet is forwarded on the selected path;
- Field update: If the packet is verified as valid, (1) update the saved vector sum_i, which is the sum of the received valid credentials and used to prevent replay attacks; (2) update PVF_i to prove to the downstream nodes that N_i has forwarded the packet.

The details of the verfication is in Section 4.5.3. Finally, when the packet arrives the destination (node N_k), the same verification process is used to achieve end-to-end source authentication and path validation.

We note that OSV only considers the problems of the packet authentication in Steps 3 and 4. We assume that the communications in steps 1 and 2 for connection setup are private and the service credentials (or credential seeds) are not observed by a third party. This end-to-end security can be achieved by using existing protocols (e.g., TLS/SSL). Almost all the related work have made this assumption.

4.5 Design Details of OSV

In this section, we provide details on OSV's design. We first introduce the setup of the OSV protocol and then describe the details of packet initialization and verification.

Table 4.2. Credential information saved in source node S and each node N_i along the path.

	node S	node N_i
saved information	H, K_S, $\{K_{N_1}, ..., K_{N_k}\}$	$\{h_{N_i}^1, ..., h_{N_i}^{m_i}\}$, K_S, K_{N_i}

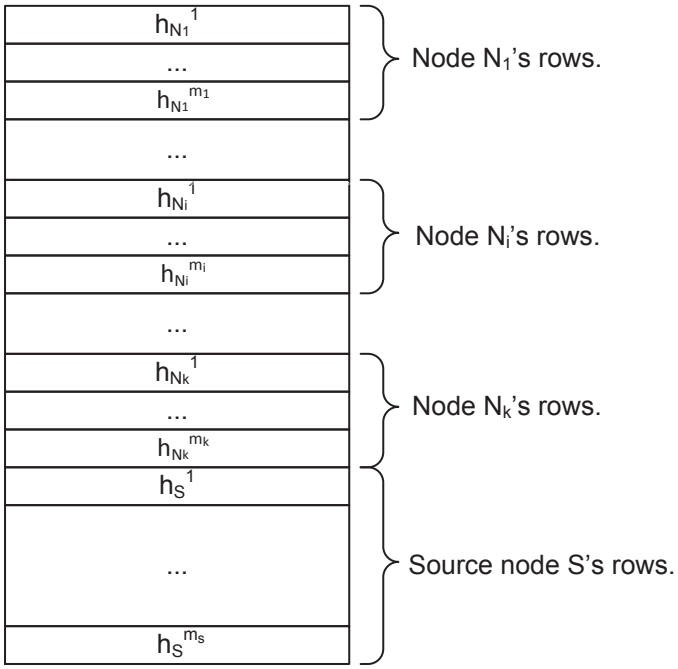


Figure 4.2. An illustration of the splitting of a generated Hadamard matrix.

4.5.1 OSV Initialization

We introduce what are the source/router credential "secrets" and how they are transmitted between the source node and all the routers, which are Steps 1 and 2 in Figure 4.1.

When a path of nodes (i.e., $< N_1, N_2, ..., N_k >$) is determined by the source node S, S generates a random $n \times n$ Hadamard matrix H locally. Related work has derived many construction methods for generating a Hadamard matrix for a given n [8]. The Hadamard construction method in our system is a simple, but efficient one – *Kronecker Product Construction* (a detailed description of this construction method

can be found in Section 3.6. H is different for each connection in the network since we use random transform operations during the construction process, where a proof by experiments can be also found in Section 3.6. In Step 1 in Figure 4.1, source node S delivers to each node N_i several different rows of H (through standard security protocols that ensure authenticity, integrity, and privacy). Node N_i saves these rows locally. The remaining rows of H are used by source node S to construct credentials. The different rows saved in each node N_i are used to verify the further packets sent from the source node S. An illustration of the splitting of the generated Hadamard matrix H is shown in Figure 4.2.

In our system, the number of rows, m_i, that each node N_i selects is a very small fraction of H, so that they cannot be used to construct the original H. Besides, each node N_i is not able to generate the rows saved in other nodes. For example, in our prototype implementation, $n = 128$ and $m_i = 3$. Only the source node S is aware of each node N_i's orthogonal sequences, while nodes in the network do not know other nodes' sequences. We denote the rows of H saved in N_i as $\{h^1_{N_i}, h^2_{N_i}, ..., h^{m_i}_{N_i}\}$. The rows used as credentials by S are denoted as $\{h^1_S, h^2_S, ..., h^{m_S}_S\}$. Therefore, the $n \times n$ Hadamard matrix H generated by source node S can be expressed as:

$$\begin{aligned} H &= \{h^1_{N_1}, h^2_{N_1}, ..., h^{m_1}_{N_1}, ..., h^1_{N_i}, h^2_{N_i}, ..., \\ &\quad h^{m_i}_{N_i}, ..., h^1_{N_k}, h^2_{N_k}, ...h^{m_k}_{N_k}, h^1_S, h^2_S, ..., h^{m_S}_S\}, \end{aligned} \tag{4.1}$$

where $m_S + \sum_{i=1}^{k} m_i = n$.

In addition, source node S also shares its local secret K_S with each node N_i, and N_i also shares its local secret K_{N_i} with S. But any given node does not know another node's local secret. These secrets K_S and K_{N_i} are used to generate the pseudorandom sequence r by a pseudorandom function (PRF) F.

In summary, in Step 1, (1) source node S generates a hadamard matrx H locally; (2) source node S then delivers specific rows $\{h^1_{N_i}, h^2_{N_i}, ..., h^{m_i}_{N_i}\}$ to each corresponding

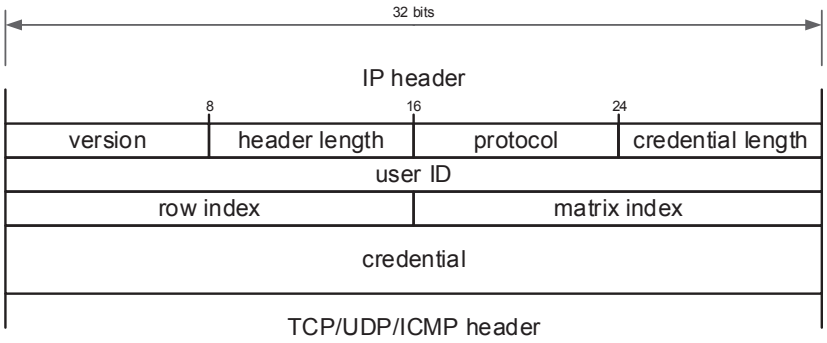


Figure 4.3. OSV header.

provider or node N_i, respectively; (3) source node S delivers its local secret K_S to each node N_i, and each node N_i also delivers its own secret K_{N_i} to source node S. In Step 2 in Figure 4.1, each provider delivers these credential information to their corresponding nodes for verification and forwarding. Table 4.2 summarizes the credential information saved in source node S and each node N_i.

4.5.2 Packet Initialization

After introducing the setup of credential information between S and N_i, we now describe the detailed computing process of an OSV packet initialization. which is the Step 3 in Figure 4.1:

4.5.2.1 Credential Generation

Each time when source node S wants to send a packet on the selected path, it puts a credential c in the packet header. Each node N_i along the path then uses the credential c to verify whether this packet comes from source node S. Each credential c is constructed by using an unused row in $\{h_S^1, h_S^2, ..., h_S^{m_S}\}$. When all the rows are used, source node S will generate a new Hadamard matrix and repeat the Step 1 in Figure 4.1. In real implementations, the source node generates multiple different Hadamard matrices locally at a time, and deliver multiple sets of $\{h_S^1, h_S^2, ..., h_{N_i}^{m_i}\}$ to each node N_i to avoid repeating step 1.

We use j to denote the index of an unused row. S chooses this unused row, denoted as h_S^j, and computes a pseudorandom sequence $r_S^j = F(K_S|j)$. The credential, c, then is

$$c = h_S^j + r_S^j = h_S^j + F(K_S|j). \tag{4.2}$$

The reason why we add r_S^j with h_S^j is that, if h_S^j is observed by an attacker, the attacker can get a valid sequence by generating lots of sequences that are orthogonal to h_S^j. Therefore, r_S^j can help decrease the probability for attackers to forge a valid credential. The pseudorandom sequence r_S^j can help decrease the probability for attackers to forge a valid credential significantly. Note that each N_i is able to regenerate r_S^j since K_S is shared with them and the index j can be found in OSV packet header.

4.5.2.2 OV_i Generation

For each node N_i, S first computes corresponding pseudorandom sequence $r_{N_i}^j$ by $r_{N_i}^j = F(K_{N_i}|j)$. Then, the corresponding OV_i for each node N_i is calculated by

$$OV_i = h_{N_i}^x \cdot \big(r_S^j + \sum_{p=1}^{i-1} r_{N_p}^j\big), \tag{4.3}$$

where x is a random chosen integer between 1 and m_i. In particular, $OV_1 = h_{N_1}^x \cdot r_S^j$. The symbol '$\cdot$' denotes an inner product computation.

We use a random x to increase the variability of OV_i in each packet sent from S leading to a higher security. Because the inner product result of $h_{N_i}^p$ $(1 \leqslant p \leqslant m_i)$ of node N_i and $h_{N_i'}^{p'}$ $(1 \leqslant p' \leqslant m_i')$ of node N_i' is always zero, no matter which row is chosen. More details can be found in Section 4.5.3.

4.5.2.3 PVF Generation

S generates PVF_0, the initial PVF value, as

$$PVF \leftarrow PVF_0 = h_S^j + r_S^j. \tag{4.4}$$

During the packet's forwarding process, PVF is updated by each node N_i along the path, while the credential c and each OV_i never change.

After the source node computes c, OV_i and PVF, it inserts these values into the packet header and transmits the packet toward the first node N_1. The pseudo code of the algorithm for packet initialization is shown in Algorithm 7.

4.5.3 Packet Verification and Forwarding

We now describe what each node N_i does when forwarding the packet, which is Step 4 in Figure 1. This step contains two parts – packet verification and update. The first part can be further divided into source authentication and path validation. The second part is performed only when the packet is verified as valid.

4.5.3.1 Source Authentication

This verification step uses credential c in the OSV header. Node N_i computes source node S's corresponding pseudorandom sequence $r_S^j = F(K_S|j)$, yielding the calculated orthogonal sequence (denoted as h, where $h = c - r_S^j$). The credential c is valid when it satisfies

$$\forall p \ (1 \leqslant p \leqslant m_i), \ h \cdot h_{N_i}^p = 0. \tag{4.5}$$

In Section 3.6, experiments show that it is a very low probability (below 10^{-5}) for a random h that is orthogonal to all $h_{N_i}^p$ $(1 \leqslant p \leqslant m_i)$. Furthermore, to prevent a replay attack, h is also used to compute the inner product with sum_i to check if c has been used or not. An unused and used c ($c = h + r_S^j$) always satisfies $h \cdot sum_i = 0$ and $h \cdot sum_i = |h|^2 \neq 0$, respectively.

4.5.3.2 Path Validation

This verification step uses PVF_{i-1} and OV_i in the OSV header. A packet that has passed through all the upstream nodes satisfies the following equation:

$$\exists p \ (1 \leqslant p \leqslant m_i), \ h^p_{N_i} \cdot PVF_{i-1} = OV_i. \tag{4.6}$$

The above equation is true since node N_{i-1} will update the value of PVF in the packet from PVF_{i-2} to PVF_{i-1} as the following:

$$PVF_{i-1} = PVF_{i-2} + h^x_{N_{i-1}} + r^j_{N_{i-1}}. \tag{4.7}$$

Note that $PVF_0 = h^j_S + r^j_S$, so we can get

$$\begin{aligned} PVF_{i-1} &= h^j_S + r^j_S + \sum_{p=1}^{i-2}(h^x_{N_p} + r^j_{N_p}) + h^x_{N_{i-1}} + r^j_{N_{i-1}} \\ &= h^j_S + r^j_S + \sum_{p=1}^{i-1}(h^x_{N_p} + r^j_{N_p}). \end{aligned} \tag{4.8}$$

As with Equation 4.3, x is a randomly chosen integer ranging from 1 to m_p. Assuming x' is an integer that satisfies $1 \leqslant x' \leqslant m_i$, Equation 4.6 is true since

$$\begin{aligned} & h^{x'}_{N_i} \cdot PVF_{i-1} \\ = \ & \underbrace{h^{x'}_{N_i} \cdot h^j_S}_{\text{equals } 0} + \sum_{p=1}^{i-1}(\underbrace{h^{x'}_{N_i} \cdot h^x_{N_p}}_{\text{equals } 0}) + h^{x'}_{N_i} \cdot r^j_S + \sum_{p=1}^{i-1}(h^{x'}_{N_i} \cdot r^j_{N_p}) \\ = \ & h^{x'}_{N_i} \cdot r^j_S + \sum_{p=1}^{i-1}(h^{x'}_{N_i} \cdot r^j_{N_p}). \end{aligned} \tag{4.9}$$

The right side of the above equation equals OV_i when $x' = x$ in Equation 4.3. Therefore, OSV can achieve the path validation through updating PVF by each node N_i along the path.

We describe the verification on node N_1 as an example. When a packet is sent from S to N_1, N_1 does the following computation using different $h^p_{N_1}$ ($p = 1, .., m_1$):

$$h^p_{N_1} \cdot PVF_0 = h^p_{N_1} \cdot (h^j_S + r^j_S) = h^p_{N_1} \cdot r^j_S. \tag{4.10}$$

N_1 finds a specific $h^p_{N_1}$ that makes the result equal to OV_1 ($OV_1 = h^x_{N_1} \cdot r^j_S$). Note that computations with different $h^j_{N_1}$ do not cause much overhead since m_i is typically very small in OSV (less than 5). After the verification, N_1 updates PVF from PVF_0 to PVF_1:

$$\begin{aligned} PVF_1 &= PVF_0 + h^x_{N_1} + r^j_{N_1} \\ &= h^j_S + h^x_{N_1} + r^j_S + r^j_{N_1}, \end{aligned} \tag{4.11}$$

where $r^j_{N_1} = F(K_{N_1}|j)$ is N_1's corresponding pseudorandom sequence. We can observe that $h^p_{N_2} \cdot PVF_1 = h^p_{N_2} \cdot (r^j_S + r^j_{N_1})$, where the result equals to OV_2 when p equals to the x chosen by source node S.

4.5.3.3 Field Update

When the packet passes both source authentication and path validation, node N_i update PVF in the packet to prove to downstream nodes that it has forwarded the packet. The operations are discussed above and are shown in the following equation:

$$PVF \leftarrow PVF_i = PVF_{i-1} + h^x_{N_i} + r^j_{N_i}, \tag{4.12}$$

where x is a random chosen integer ranging from 1 to m_i.

In addition, node N_i needs to update its local variable sum_i with $sum_i \leftarrow sum_i + h^j_S$. This sum_i can be used to identify replay attacks. This is because, for a replay packet, the result of the inner product of h ($h = c - r^j_S$) and sum_i equals $||h^2||$ but

Algorithm 7 OSV header initialization pseudo code.

1: **function** SOURCE INITIALIZATION

Require: (a) Generated Hadamard matrix H; (b) Secrets K_S and each K_{N_i} that N_i shares with S, respectively.

▷ The detailed method of generating H is in [20]

2: **if** all rows in $\{h_S^1, h_S^2, ..., h_S^{m_S}\}$ in H are used **then**

3: Generate a new $H = \{h_S^1, h_S^2, ..., h_S^{m_S}, h_{N_1}^1, h_{N_1}^2, ..., h_{N_1}^{m_1}, ..., h_{N_i}^1, h_{N_i}^2, ..., h_{N_i}^{m_i}, ..., h_{N_k}^1, h_{N_k}^2, ...h_{N_k}^{m_k}\}$

4: $j \leftarrow$ the index of an unused row in $\{h_S^1, h_S^2, ..., h_S^{m_S}\}$

5: $r_S^j \leftarrow F(K_S|j)$

6: $c \leftarrow h_S^j + r_S^j$

7: $PVF \leftarrow PVF_0 = h_S^j + r_S^j$

8: **for** each node N_i on the path, where $1 \leqslant i \leqslant k$ **do**

9: $r_{N_i}^j \leftarrow F(K_{N_i}|j)$

10: $x \leftarrow$ a random chosen integer between 1 and m_i

11: $OV_i \leftarrow h_{N_i}^x \cdot \left(r_S^j + \sum_{p=1}^{i-1} r_{N_p}^j\right)$

12: **end function**

not 0 either. When a new Hadamard matrix is used, N_i resets the value of sum_i to zero.

Using the above process, source authentication and path validation can be achieved. The pseudo code of the algorithm for packet validation and update is shown in Algorithm 8.

4.6 Evaluation

We present an evaluation of OSV consisting of three aspects: a security analysis based on our security model, a performance evaluation based on emulation of a broad design space, and a prototype evaluation based on an implementation in a testbed.

4.6.1 Security Analysis

We describe how OSV meets the security requirements described in Section 4.3.2. OSV achieves source authentication and path verification by merely requiring that the source and each node N_i along the path are trusted (since they exchange secret key information to generate matrices used in OSV). However, no entity along the path

Algorithm 8 OSV header validation and update (in Node N_i) pseudo code.

1: **function** OSV HEADER VALIDATION AND UPDATE
Require: (a) Received OSV packet header that contains c, j, OV_i, PVF and etc; (b) Secrets K_S and K_{N_i}; (c) Rows $\{h^1_{N_i}, ..., h^{m_i}_{N_i}\}$ from H.
2: $r^j_S \leftarrow F(K_S|j)$
3: $h \leftarrow c - r^j_S$
▷ 1) Source authentication (the detailed method is in [20])
4: **for** $1 \leqslant p \leqslant m_i$ **do**
5: **if** $h \cdot h^p_{N_i} \neq 0$ **then**
6: Drop the packet
7: **if** $h \cdot sum_i \neq 0$ **then**
8: Drop the packet
▷ 2) Path Validation
9: (Note PVF in OSV header = PVF_{i-1})
10: $flag \leftarrow$ FALSE
11: **for** $1 \leqslant p \leqslant m_i$ **do**
12: **if** $h^p_{N_i} \cdot PVF_{i-1} == OV_i$ **then**
13: $flag \leftarrow$ TRUE
14: **break**
15: **if** $flag ==$FALSE **then**
16: Drop the packet
▷ 3) Field Update
17: $x \leftarrow$ A random chosen integer between 1 and m_i
18: Update $PVF \leftarrow PVF_i = PVF_{i-1} + h^x_{N_i} + r^j_{N_i}$
19: Update $sum_i \leftarrow sum_i + h^j_S$
20: **end function**

needs to trust another entity. This property holds on any network configuration, including ones that have malicious routers. Thus, item 2 of the security requirements is met by design. To examine the other two security requirements, we consider various potential attacks.

Attack by brute force: The router uses orthogonal sequences $h^j_{N_i}$ to make these attacks very difficult. A randomly generated credential has a very low probability to be orthogonal with all $h^j_{N_i}$. For instance, when $n = 128$ and $m_i = 3$, this probability is below $10^{-4.5}$ (this results is described in more detail in Section VI of [20]). Besides, this kind DoS attack cannot exhaust computational resources on routers, either, since the packet is discarded within very few inner product computations, as shown in the

experiment results in Section 4.6.2.2. Thus, we can achieve item 1 of the security requirements in Section 4.3.2.

Replay attack: Replay attacks are prevented by use of the sum vector sum_i that is stored in routers. The detection of a replayed credential works because once one row from a Hadamard matrix (denote as h) has been verified to be valid, then it is added into sum_i by node N_i. Thus, we have $(h, sum_i) = (h, h) = |h|^2 \neq 0$ while the use of a valid, unused h still results in 0. Together with item 7 of attacker capabilities, we can thus ensure that item 1 of the security requirements is ensured under replay attacks.

Attack by controlling the intermediate router: It means that the attacker can compromise and learn some set of rows in the matrix H. However, it is impossible for the attacker to generate the original H since each router N_i only saves a very small fraction of H (less than 5 rows in a 128×128 H). It is also a very low probability (below 10^{-5}) for a random generated credetnial that is orthogonal to all $h^p_{N_i}$ $(1 \leqslant p \leqslant m_i)$ in any other Node N_i. Besides, the matrix H will change when the rows $\{h^1_S, h^2_S, ..., h^{m_S}_S\}$ in source node S are all used.

Path deviation attack: OSV ensures that a successful verification of PVF_{i-1} against OV_i implies that a received packet N_i has traversed all routers in the intended path in correct order. The attacker does not possess K_S and $h^j_{N_i}$, which is required to compute PVF_i and OV_i. The verification in Equation 4.6 does not succeed if PVF_i has not been updated by routers or has been updated in an incorrect order. This verification ensures that a malicious router cannot mount and attack where traffic skips routers or traverses them out-of-order. Besides, if a malicious router selects a path not intended by the source, an honest intermediary router rejects the packet. Thus, we can achieve item 3 of the security requirements since traffic must come from the specified source along the intended path.

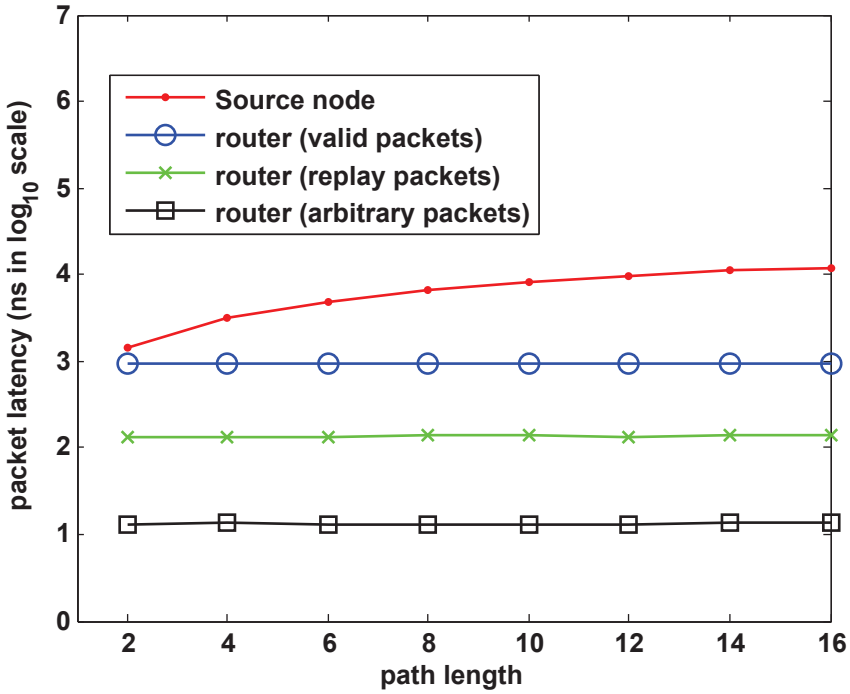


Figure 4.4. Packet processing latency of OSV (in a log_{10} scale).

Attack that compromises secrets: Even if an attacker knows every processing step taken by the source and routers, it cannot create valid credentials or PVF_i since it does not have the whole secret key material (item 5 of attacker capabilities), which is shared through established end-to-end security protocols (e.g., TLS/SSL). The Hadamard matrices are also difficult to be guessed due to their variability. Furthermore, during a packet's forwarding, we add a pseudorandom sequence r with orthogonal sequence h, which avoids that an attacker can observe h directly.

Based on these considerations, we conclude that OSV achieves the stated security requirements and thus provides effective source authentication and path validation.

4.6.2 Performance Evaluation

Next, we evaluate OSV with respect to the desired performance properties. We have implemented the credential generation and verification algorithms in C++. We use a PC with an Intel Core2 Quad CPU Q9400 running at 2.66GHz to obtain per-

formance results. The operating system is Ubuntu 14.04 64-bit with kernel version 3.13.0-24 and gcc version 4.8.2. For our evaluation, we choose $n = 128$ and $m_i = 3$.

4.6.2.1 Setup Latency

We first briefly evaluate the processing costs of Step 1 in Figure 4.1. The main cost in this setup process is the time for the source node S to generate the Hadamard matrix H. As mentioned in Section 4.5.2, S generates multiple Hadamard matrices at a time to avoid repeating Step 1 frequently. In our implementation, we let S generate 100 Hadamard matrices at a time, which can guarantee transmitting about 10,000 packets without repeating Step 1. We run the experiments by using paths with a number of 3, 5, 8 and 10 hops, respectively. The results have minor difference and concentrate around 0.15ms. This is much lower than OPT, e.g., for a 8-hop path, OPT needs about $20.68ms$ to setup the connection according to Table 3 in [48] (there is no result for the setup time for ICING).

4.6.2.2 Packet Processing Latency

We evaluate the processing costs of an OSV packet initialization in source node S, and the verification in nodes N_i along different hops in the path. The results of our experiments are shown in Figure 4.4 (note that the unit is ns and y-axis is in a log_{10} scale). We can see that the packet process latencies of OSV in source node and routers are all below 0.01 μs.

We first measure the time of a source S to initialize a packet. We let S initialize the packets with different Hadamard matrices H and different secret keys (K_S and K_{N_i}). The results in Figure 4.4 show the average time by running 1 million iterations. The latency for initializing a packet in source node S increases linearly with the path length, since a longer path requires the source to perform more calculations on OV_i.

For the verification time on routers (node N_i in the path). We use a kernel packet generator to generate different packets and send them through the software router

implemented on our experimental platform. For each run, the packet generator sends 1 million packets of each type to the router. We record the average time for the router to process each type of packet. We use 3 types of packets for this evaluation: packets with valid credentials, packets with replayed credentials, and packets with random credentials. The latter use arbitrary bit values for credentials, PVF and OV_i. The results show that the verification time for a replay packet or an arbitrary packet are much less than for a valid packet. The reason is that invalid packets are discarded in the credential verification process. In particular, arbitrary packets are discarded within very few inner product computations with $h^j_{N_i}$. We also observe that the verification time in routers is independent of the path length, because the router does not perform any computation that depends on the path length. (Actually, a router in OSV has no information about the path length in OSV.) The latency of verifying the packet in N_i is less than generating a packet in S. We believe that this slight difference does not affect of the efficiency of OSV, since the source node S typically has a lower traffic throughput than routers.

Since OSV uses inner product computations to replace expensive cryptography computations, its processing time is significantly lower compared to ICING and OPT. The verification time of ICING [53] in routers is $2.6k + 24.4\mu s$ (k is the path length) for each valid packet, which is hundreds of times more than in OSV. In OPT, the verification time in routers is in the same scale as OSV's. But the time for the setup packet is too long, for instance, for a 8-hop path, OPT needs about a time of $20.68ms$ to setup the connection.

4.6.2.3 Packet Overhead

We first examine how we decide the length of PVF and OV_i in the OSV header, which is shown in Figure 4.3. According to Equation 4.3 (which is an inner product computation), the maximum value of OV is $128 \log k$, where k is the path length.

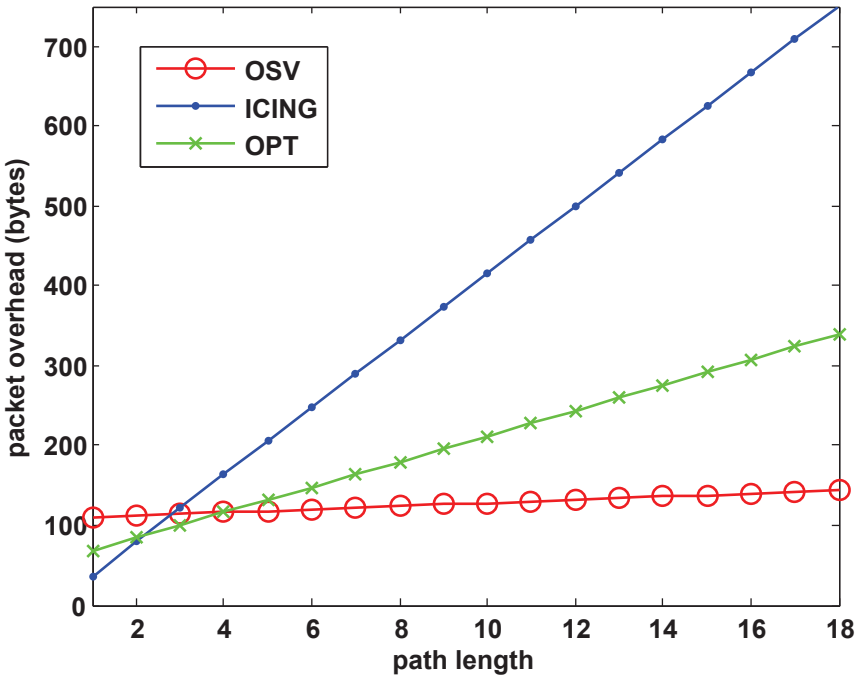


Figure 4.5. Packet overhead varying with increasing path length in bytes.

Therefore, we believe 2 bytes is sufficient to save this value. According to Equation 4.7 (which results in a vector), the maximum length of PVF is $128 \log k$ (note that it is the length, not the value).

Considering that paths with more than 30 hops are very rare in the Internet [44], we set the length of PVF as 80 bytes. Based on these assumptions, an OSV header includes 12+16+80=108 bytes that do not depend on the packet's path length. Only 2 bytes that are needed for each node N_i. An comparison of the packet overhead of OSV, ICING, and OPT is shown in Figure 4.5 for varying path lengths. Using the same pessimistic estimate of an average provider-level path length of 5, as in ICING [53], our packet header overhead is 118 bytes, which is 57.5% and 89.3% of the overhead of ICING (205 bytes) and OPT (132 bytes), respectively. Besides, in Figure 4.5, we can observe that the packet overhead of OSV is almost independent of the path length. Therefore, OSV has a smaller network bandwidth overhead compared to ICING and OPT.

4.6.2.4 Storage Consumption

We now evaluate the storage overhead in the source node and routers. As discussed in Section 4.5.1, the source node needs to save the Hadamard matrix H, a table of the indices of the selected rows in H by each node, and k secret keys K_{N_i} of each node N_i. Note that during the setup process of Step 1 in Figure 4.1, S generates multiple Hadamard matrices and deliver multiple sets of $\{h^1_{N_i}, ..., h^{m_i}_{N_i}\}$ to corresponding N_i. But, after that, S only saves one Hadamard matrix locally. When the rows of the current matrix are all used, it will generate a new one and also only save the new one.

Compared to ICING and OPT, OSV requires more space on the source node, because saving H and a table of indices required about 3 kilobytes. Each node N_i needs to store m_i orthogonal sequences ($16 \times 3 = 48$ bytes), the secret key K_S (16 bytes) and a sum vector sum_i ($16 \times \log 128 = 96$ bytes). This leads to a total requirement of 160 bytes. Note that the router storage overhead is under a consideration of preventing replay attacks. Many authentication scheme including ICING and OPT cannot provide anti-replay protection by default and have to keep a modestly sized cache at each node: From this point of view, OSV can decrease the storage consumption since it only saves the sum of the received valid credentials while other authentication schemes need to save all the received valid credentials. For example, when OSV uses a space of 96 bytes to prevent all replay packets, other authentication schemes need a $16 \times 16 = 256$ bytes space. Even if the additional cache for preventing replay attacks is not considered, the storage overhead for OSV in routers is almost the same as for ICING and OPT.

4.6.3 Deployment on Testbed

We have deployed OSV on ExoGENI [9] to demonstrate that OSV can work in a real network. We use the Netfilter Queue to insert the OSV header between TCP and IP headers. In the experiment, we let source node S generate traffic at a rate of

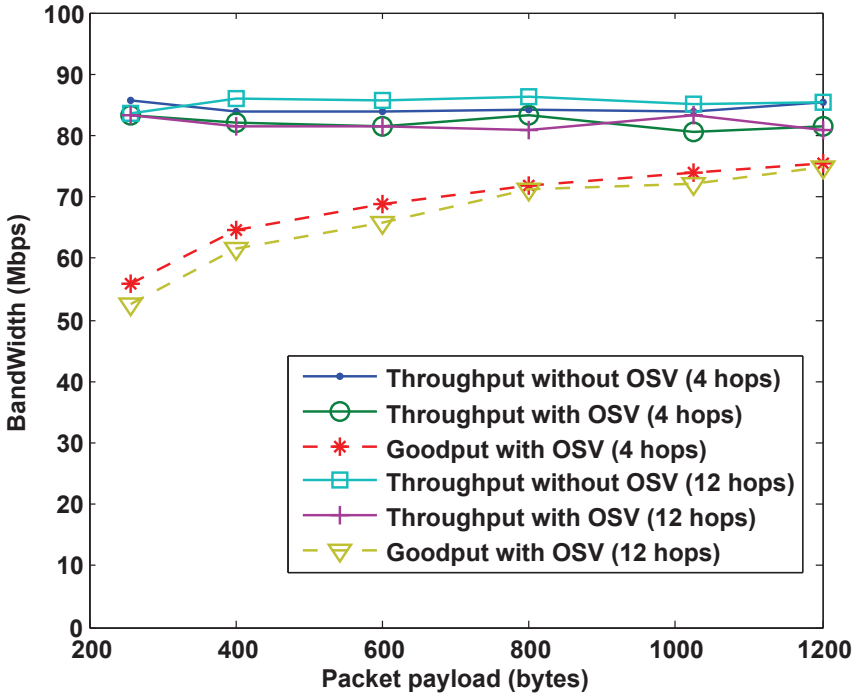


Figure 4.6. Throughput and goodput of OSV for 4-hop and 12-hop paths, in the context of varying payload sizes.

100Mbps and we observe the traffic sent from the intermediate routers. We compare the throughput when the nodes use and not use OSV protocol, and show OSV is a lightweight and scalable protocol.

We first examine OSV's overhead in terms of per-packet processing by measuring both the throughput and the goodput (the bandwidth used to transmit the payload of the packets, excluding OSV header). The results are shown in Figure 4.6. We can see that no matter what size of the payload is, enabling or disabling OSV protocol does not have noticeable effect on the throughput. This is because the packet forwarding takes far more time than the OSV process. This demonstrates the low verification time of OSV which is shown in Section 4.6.2.2. Cryptography schemes like ICING and OPT, in contrast, have lower throughput for small packet sizes due to computational demands. For instance, ICING can only achieve about 50% of the maximum throughput when the payload size is 256 bytes. We can also observe the goodput of

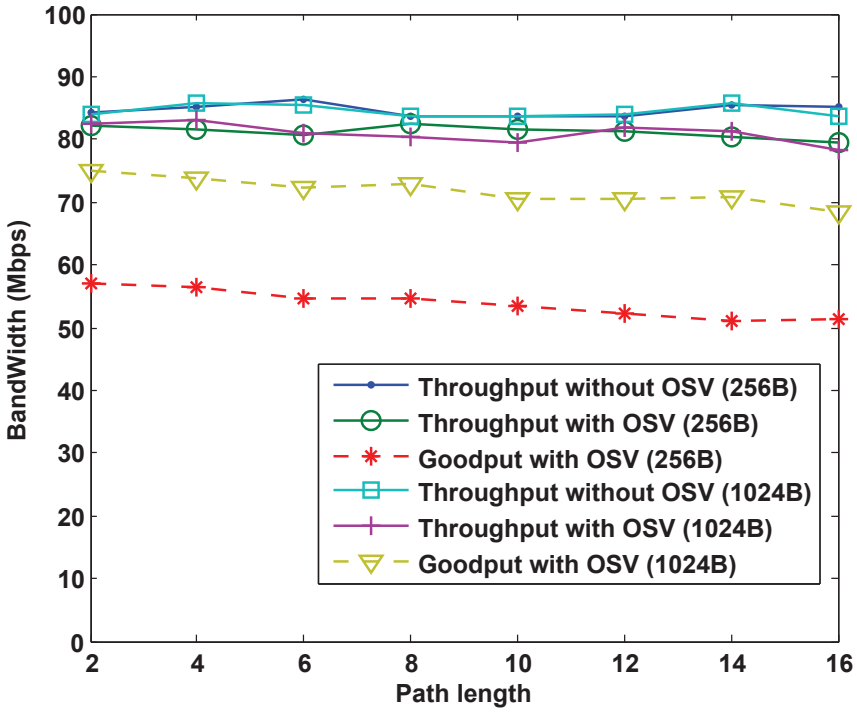


Figure 4.7. Throughput and goodput of OSV for 256B and 1024 packets, in the context of varying path lengths.

OSV increases as the packet size increases. This is because OSV header represents a smaller fraction of the total packet size as the payload size increases. The goodput of OSV with a 4-hop path is slightly higher than that with a 12-hop path is due to a smaller OSV header size. Besides, the source node needs to generate new Hadamard matrices and deliver the credential information to each router more frequently when the path length increases.

We then examine the OSV's scalability with respect to the path length. We perform tests with a minimum path length of 2 hops and a maximum path length of 16 hops. The results are shown in Figure 4.7. As the same in Figure 4.6, enabling or disabling OSV protocol does not have noticeable effect on the throughput. The goodput of OSV decreases slowly when the path length increases. This is because the OSV header size only increases 2 bytes for each additional hop, and the computing process of OSV in the router is irrelevant to the hops of the selected path. ICING

and OPT's goodput both show a faster decrease with increasing path length. We use the results from Figure 4.7 to compare with Figure 7 in [48]: when the packet size is 256 bytes, OSV's per-hop goodput degradation ratio is $\frac{57.8-51.0}{57.8 \cdot 14} = 0.84\%$, while OPT's is 3.03% and ICING's is 5.74%; when the packet size is 1024 bytes, OSV's per-hop goodput degradation ratio is about $\frac{74.7-69.2}{74.7 \cdot 14} = 0.53\%$, while OPT's is 1.25% and ICING's is 2.80%. Furthermore, when the path length is 2, OSV's goodput is $\frac{57.8}{83.0} = 69.6\%$ (for 256B packet size) and $\frac{74.7}{83.0} = 90.0\%$ (for 1024B packet size), respectively, which is also better than OPT and ICING (they are both 62% for 256B packet size and 85% for 1024B packet size). Therefore, OSV always has a better goodput compared to ICING and OPT.

4.7 Conclusions

In this chapter, we present a novel technique for both source authentication and path verification, called OSV (Orthogonal Sequence Verification). OSV uses orthogonal capabilities that are carried in packets for verification, which can be implemented efficiently by basic bitwise operations on a processor. Our experimental results show that the verification time in OSV is much lower than that existing approaches while providing the necessary security guarantees. We believe that Orthogonal Sequence Verification represents an important step toward more security in networks by providing efficient and effective source authentication and path validation.

CHAPTER 5

DEPLOYMENT ON CONTAINERS

In this chapter, we will use "Docker" [1] as the tool to evaluate the effectiveness and performance of our proposed path validation mechanism. Container is a form of lightweight virtualization, which provides an alternative means to partition hardware resources among users and expedite application deployment. Container technique is gaining increasing attention in recent years and has become an alternative to traditional virtual machines. Some of the primary motivations for the enterprise to adopt the container technology include its convenience to encapsulate and deploy applications, lightweight operations, as well as efficiency and flexibility in resources sharing [74].

5.1 Setup

A single host can have multiple containers running on it. The containers may have to communicate with each other and by using docker networking. There are multiple container networking modes on a single host, where we choose "bridge mode" in our deployment. As shown in Figure 5.1, Docker creates a bridge named *docker0* in the host OS once the Docker daemon *dockerd* is launched. When a new container is started, a pair of *veth* ports are created to connect the container to *docker0*. All containers connecting to *Docker0* belong to one virtual subnet and can communicate with each other using private IP addresses. Bridge mode alone does not connect containers to external networks and relies on other services, such as NAT and overlay, for inter-host communication. Bridge mode allows each container to own an isolated

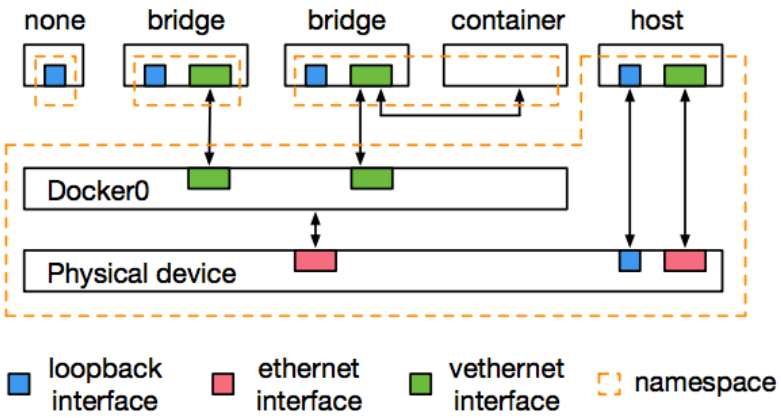


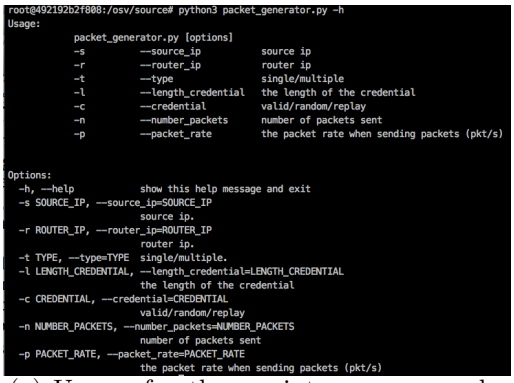
Figure 5.1. Container networking on a single host.

network namespace and an IP address, and all intercontainer communications need to go through the *docker0* bridge.

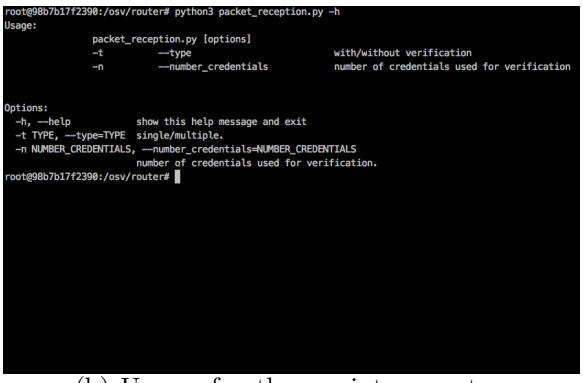
We deploy two different docker containers as a source node and a router on a single host. The host is a PC with an Intel Core i7 Quad CPU running at 2.7 GHz.These two containers are connected to one *Docker0* bridge and have no connection to external networks, so that we can evaluate their performance. These two containers both boot from a Ubuntu18 docker image and are allocated with a separate cpu core [2], respectively. The source node and route use *python* scripts to generate, send, receive packets and do verification on the credentials in them. Figure 5.2 shows the usage of the scripts on source node and router. In the deployment, we will evaluate the performance when the credential length is 32, 64 and 128.

5.2 Defense on Packets with Random Credentials

We first want to show the security effectiveness of our algorithm that can defend the attacks from the packets with random credentials. Figure 5.3 shows an example of using scripts to send and receive packets with random credentials between source and router (credential length is 64). We can see that the verification results are “false”.



(a) Usage of python script on source node.

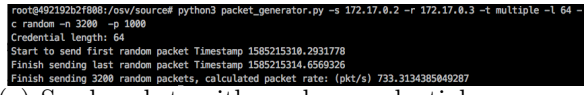


(b) Usage of python script on router.

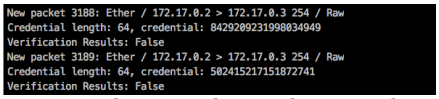
Figure 5.2. Usage of the python scripts on source node and router to generate, send, receive packets and do verification on the credentials in them.

We simulate a scenario when an attacker sends random credentials with different credential lengths (i.e., n) and the number of credentials used on the router (i.e., m). As explained in Section 4.5.3, a credential is verified as "valid" should satisfies two conditions. The first condition is that the credential passes the verification of the saved credentials in the router, another condition is that the credential passes the verification of the saved sum of received credentials in the router. If the router does not receive any valid credential yet, the sum will be zero and any credential can satisfy the second condition. So our simulation includes the cases of the router received valid credentials or not.

Figure 5.4 shows the success probability of such random attack varying with an increasing number of saved credentials in the router. We can see that, after the router receives valid credentials (labeled as "with sum" in the figure), the success probability becomes lower. It can be also observed that, when $n = 128$ and $m > 8$ or when $n = 64$ and $m > 12$, it can guarantee the breakthrough probability less than 10^{-6}, which can be considered safe enough. We can get that, even if the packet transmission rate of an attacker is as high as 10,000 pkt/s, then it still needs nearly 2 minutes to guess a valid credential. During the 2 minutes, the source node might already change a



(a) Send packets with random credentials on source node (credential length is 64).



(b) Receive packets with random credentials on router.

Figure 5.3. Send and receive packets with random credentials between source node and router (credential length is 64).

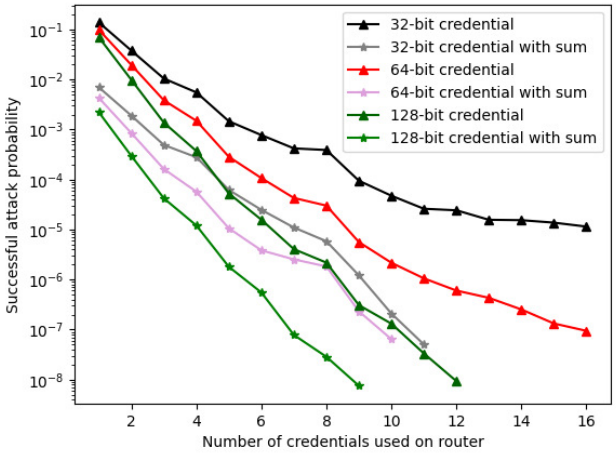
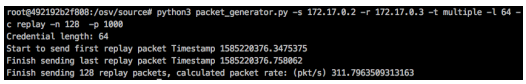


Figure 5.4. Successful probability of attacks of packets using random credentials, including the cases that the router received valid credentials or not.

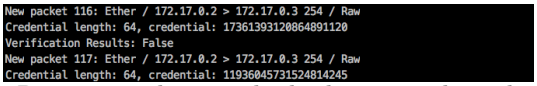
multiple of Hadamard Matrices. For some services with low-security requirements, we consider 32-bit or 64-bit credentials can also be used with a proper chosen m.

5.3 Defense on Packets with Duplicate Credentials

In this section, we simulate a scenario where an attacker sends replay packets. We send a number of packets with valid credentials to the router first. Then, we send the same packets to the router again and the result is that all these replay packets are discarded by the verifier no matter what the credential length is. Figure 5.5 shows an example of using scripts to send and receive packets with duplicate credentials between source and router (credential length is 64). We can see that the verification results are "false".

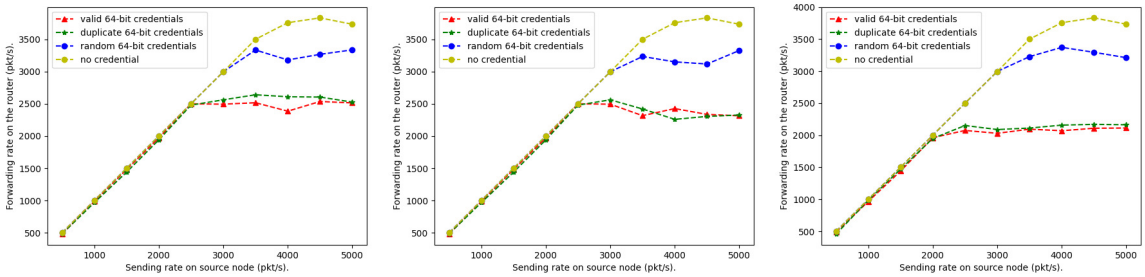


(a) Send packets with duplicate credentials on source node (credential length is 64).



(b) Receive packets with duplicate credentials on router.

Figure 5.5. Send and receive packets with duplicate credentials between source node and router (credential length is 64).



(a) When the number of credentials used on router is 3.

(b) When the number of credentials used on router is 10.

(c) When the number of credentials used on router is 20.

Figure 5.6. The forwarding rate of the router varies with increasing sending rate of the source node when the credential length is 64 and the number of credentials used on router is 3, 10 and 20, respectively.

5.4 Evaluation on Verification Overhead

In this section, we evaluate the overhead of the router that takes the verification on the receiving packets from the source node. Our experiments records the packet forwarding rate on the router with increasing packet sending rate from the source node with and without our verification mechanism. As mentioned in Section 5.1, the source node and router uses a different CPU core and both have enough memories, so we think they do not interfere with each other. When the experiment is running, we can use command *"docker stat"* to observe the cpu usages of the source node and the router, which are both nearly 100%.

Figure 5.6 shows the forwarding rate of the router varies with increasing sending rate of the source node when the credential length is 64 and the number of credentials used on router is 3, 10 and 20, respectively. The experiments record the results when the source node sends packets with valid, random and replay credentials, respectively.

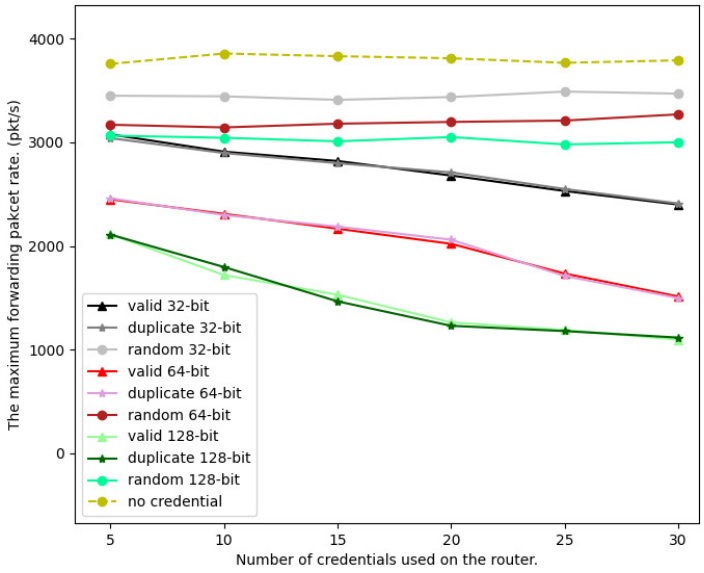


Figure 5.7. The maximum forwarding packet rates with increasing number of credentials used on the router with different credential length.

Besides, in order to analyze the overhead of the router due to the extra verification, we also record the results when the source node send packets without credentials. In this situation, the router does not need to do verifications on them. We can observe that, when the number of credentials used on the router is 3, the forwarding rate of the router goes to a limit that is around $2500 pkts/s$ for verifying the valid or replay packets. This number is around 2300 and $2000 pkts/s$ when the the number of credentials is 10 and 20, respectively. However, the forwarding rate is around $3150 pkts/s$ and does not change much when the the number of credentials changes. This is because the router can discard the invalid credential within the first very few inner product computations.

As shown in Figure 5.7, we compare the maximum forwarding packet rates with different credential length and different number of credentials used on the router. We can observe that the forwarding packet rates for packets with random credentials are very close when the credential length changes. The reason is because the inner product

computation does not take much resource no matter the length of the credential is. We can calculate the overhead is about extra 15% processing time for the router. For the packets with valid and replay credentials, the overheads vary with the increasing credential lengths. From Figure 5.4, we can know that it would be secure enough when number of credentials used in the router is 10 for the 128-bit credential. Therefore, we can calculate that the maximum possible overhead for verifying the valid and replay packets are about extra 50% processing time in the router.

CHAPTER 6

SUMMARY

The exponential growth of the current Internet has been extraordinary. An important developing trend of the current Internet is that the network owners and operators has more controls on their infrastructure and data flows, allowing customization and optimization, and reducing the overall capital and operational costs. The benefits of convenient controlling the whole map of the entire network enables the introduction of new features in Internet becomes less manual, less prone to error, and faster to implement.

This work addresses some fundamental problems in the current Internet. The first one is the path finding, i.e., determining a path for traffic to flow between communicating end-system, which is a core functionality in such networks with the data flow control. In typical networks, path finding is based on a single criterion, such as path length, delay, or an artificially defined weight. However, networks have grown in leaps and bounds so that single-criterion shortest paths no longer fit the whole spectrum of services that exist in todays networks. Multi-Criteria path problem has been addressed in several contexts, for example Quality of Service (QoS) routing. But when there are multiple optimization metrics, most approaches rely on an combinatorial optimization function, which combines all metrics into a single metric (e.g., weighted sum). In contrast to these algorithms, our aim is to search the whole spectrum of all the possible optimal paths that have advantages even on any one metric, which can help the network owner or operator is able to take a full consideration.

Another fundamental problem this work address is the path verification, which can be divided into source authentication and path validation further. Most of the existing approaches are either unable to satisfy security requirements or need significant computational resources due to cryptographic operations, thus limiting their suitability in practice where potentially every packet needs to be checked at line rate. This work presents OrthCredential(orthogonal vredentials) and OSV (orthogonal sequence verification), two lightweight and scalable technique to address the source authentication and path validation, respectively. OrthCredentialand OSV both use orthogonal capabilities to enable source authentication and path verification simultaneously. The verification of these orthogonal capabilities is based on inner product computations, which can be easily realized by basic bitwise operations in a processor. Therefore, OrthCredentialand OSV significantly reduces computational cost, while achieving the necessary security properties.

We believe the design and implementation of the path finding and verification algorithms in this work represents an important step toward more efficiency and security in networks.

BIBLIOGRAPHY

[1] docker. `https://www.docker.com`.

[2] Docker runtime options with Memory, CPUs, and GPUs. `https://docs.docker.com/config/containers/resource_constraints/`.

[3] libnetfilter_queue. `http://www.netfilter.org/projects/libnetfilter_queue/`.

[4] Albert, Réka, and Barabási, Albert-László. Topology of Evolving Networks: Local Events and Universality. *Phys. Rev. Lett. 85* (Dec 2000), 5234–5237.

[5] AMD. *AMD Athlon Processor x86 Code Optimization Guide*, 2002.

[6] Anderson, Tom, Roscoe, Timothy, and Wetherall, David. Preventing Internet denial-of-service with capabilities. *SIGCOMM Comput. Commun. Rev. 34*, 1 (2004), 39–44.

[7] Argyraki, K., and Cheriton, D.R. Scalable network-layer defense against internet bandwidth-flooding attacks. *IEEE/ACM Transactions on Networking 17*, 4 (Aug 2009), 1284–1297.

[8] Assmus Jr., E. F., and Key, J. D. *Designs and their codes*. Cambridge University Press, Cambridge, Great Britain, 1992.

[9] Baldine, Ilia, Xin, Yufeng, Mandal, Anirban, Ruth, Paul, Heerman, Chris, and Chase, Jeff. Exogeni: A multi-domain infrastructure-as-a-service testbed. In *Testbeds and Research Infrastructure. Development of Networks and Communities*, vol. 44 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*. Springer Berlin Heidelberg, 2012, pp. 97–113.

[10] Barabasi, Albert-Laszlo, and Albert, Reka. Emergence of Scaling in Random Networks. *Science 286*, 5439 (1999), 509–512.

[11] Barber, C. Bradford, Dobkin, David P., and Huhdanpaa, Hannu. The Quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software 22*, 4 (1996), 469–483.

[12] Barrett, Chris, Bisset, Keith, Holzer, Martin, Konjevod, Goran, Marathe, Madhav, and Wagner, Dorothea. Engineering Label-Constrained Shortest-Path Algorithms. In *Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management* (2008), AAIM '08, pp. 27–37.

[13] Bellman, Richard. On a routing problem. *Quarterly of Applied Mathematics 16*, 1 (Jan. 1958), 87–90.

[14] Black, John, Halevi, Shai, Krawczyk, Hugo, Krovetz, Ted, and Rogaway, Phillip. UMAC: Fast and secure message authentication. In *Proc. of (CRYPTO)* (Springer-Verlag, 1999), pp. 216–233.

[15] Boncelet, C.G., Jr. The NTMAC for authentication of noisy messages. *Information Forensics and Security, IEEE Transactions on 1*, 1 (2006), 35–42.

[16] Bradshaw, Robert W., Holt, Jason E., and Seamons, Kent E. Concealing complex policies with hidden credentials. In *Proc. of ACM CCS* (New York, NY, USA, 2004), pp. 146–157.

[17] Bu, T., and Towsley, D. On Distinguishing between Internet Power Law Topology Generators. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE* (2002), vol. 2, pp. 638–647 vol.2.

[18] Butler, K., Farley, T.R., McDaniel, P., and Rexford, J. A survey of BGP security issues and solutions. *Proc. of the IEEE 98*, 1 (January 2010), 100–122.

[19] Cai, H., and Wolf, T. Source authentication and path validation in networks using orthogonal sequences. In *2016 25th International Conference on Computer Communication and Networks (ICCCN)* (2016), pp. 1–10.

[20] Cai, Hao, Chen, Xinming, and Wolf, T. OrthCredential: A new network capability design for high-performance access control. In *Network Protocols (ICNP), 2014 IEEE 22nd International Conference on* (Oct 2014), pp. 233–244.

[21] Calvert, K.L., Venkatraman, S., and Griffioen, J.N. FPAC: fast, fixed-cost authentication for access to reserved resources. In *Proc. of IEEE INFOCOM* (March 2002), pp. 1049–1058.

[22] Castro, Ignacio, Panda, Aurojit, Raghavan, Barath, Shenker, Scott, and Gorinsky, Sergey. Route Bazaar: Automatic Interdomain Contract Negotiation. In *15th Workshop on Hot Topics in Operating Systems (HotOS XV)* (Kartause Ittingen, Switzerland, May 2015), USENIX Association.

[23] Chen, Shigang, and Nahrstedt, K. On Finding Multi-constrained Paths. In *Communications, 1998. ICC 98. Conference Record. 1998 IEEE International Conference on* (Jun 1998), vol. 2, pp. 874–879 vol.2.

[24] Chen, X., Cai, H., and Wolf, T. Multi-criteria routing in networks with path choices. In *2015 IEEE 23rd International Conference on Network Protocols (ICNP)* (Nov 2015), pp. 334–344.

[25] chun Hu, Yih. Efficient security mechanisms for routing protocols. In *Proc. of Network and Distributed System Security Symposium (NDSS)* (2003), pp. 57–73.

[26] Dijkstra, Edsger W. A note on two problems in connexion with graphs. *Numerische Mathematik 1* (Dec. 1959), 269–271.

[27] Duan, Zhenhai. Constructing inter-domain packet filters to control IP spoofing based on BGP updates. In *Proc. of IEEE Infocom* (2006), pp. 1–12.

[28] Dutta, Rudra, Rouskas, George N., Baldine, Ilia, Bragg, Arnold, and Stevenson, Dan. The SILO architecture for services integration, control, and optimization for the future Internet. In *Proc. of IEEE ICC* (2007), pp. 1899–1904.

[29] Ehrgott, Matthias, and Gandibleux, Xavier. A Survey and Annotated Bibliography of Multiobjective Combinatorial Optimization. *OR-Spektrum 22*, 4 (2000), 425–460.

[30] Ergun, Funda, Sinha, Rakesh, and Zhang, Lisa. An Improved FPTAS for Restricted Shortest Path. *Inf. Process. Lett. 83*, 5 (Sept. 2002), 287–291.

[31] Farrel, Adrian, Vasseur, Jean-Philippe, and Ash, Jerry. A Path Computation Element (PCE)-Based Architecture. RFC 4655, Network Working Group, Aug. 2006.

[32] Ferguson, P., and Senie, D. Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing. RFC Editor.

[33] Frikken, K., Atallah, M.J., and Li, Jiangtao. Attribute-based access control with hidden policies and hidden credentials. *Computers, IEEE Transactions on 55*, 10 (October 2006), 1259–1270.

[34] Gandibleux, Xavier, Beugnies, and Randriamasy, Sabine. Martins' Algorithm Revisited for Multi-objective Shortest Path Problems with a MaxMin Cost Function. *4OR 4*, 1 (2006), 47–59.

[35] Garroppo, Rosario G., Giordano, Stefano, and Tavanti, Luca. A Survey on Multi-constrained Optimal Path Computation: Exact and Approximate Algorithms. *Comput. Netw. 54*, 17 (Dec. 2010), 3081–3107.

[36] Ge, R., Arce, G.R., and Di Crescenzo, G. Approximate message authentication codes for N-ary alphabets. *Information Forensics and Security, IEEE Transactions on 1*, 1 (2006), 56–67.

[37] Hansen, Pierre. Bicriterion Path Problems. In *Multiple Criteria Decision Making Theory and Application*, Gnter Fandel and Tomas Gal, Eds., vol. 177 of *Lecture Notes in Economics and Mathematical Systems*. Springer Berlin Heidelberg, 1980, pp. 109–127.

[38] Haque, Imran S., Pande, Vijay S., and Walters, W. Patrick. Anatomy of high-performance 2d similarity calculations. *Journal of Chemical Information and Modeling 51*, 9 (2011), 2345–2351.

[39] Hedrick, C. Routing Information Protocol. RFC 1058, Network Working Group, June 1988.

[40] Holt, Jason E., Bradshaw, Robert W., Seamons, Kent E., and Orman, Hilarie. Hidden credentials. In *Proc. of the ACM Workshop on Privacy in the Electronic Society (WPES)* (New York, NY, USA, 2003), pp. 1–8.

[41] Hu, Yih-Chun, Perrig, Adrian, and Sirbu, Marvin. SPV: Secure path vector routing for securing BGP. In *Proc. of ACM SIGCOMM* (New York, NY, USA, 2004), pp. 179–192.

[42] Jin, Cheng, Wang, Haining, and Shin, Kang G. Hop-count filtering: an effective defense against spoofed DDoS traffic. In *Proc. of ACM CCS* (2003), pp. 30–41.

[43] Kent, S., Lynn, C., and Seo, K. Secure border gateway protocol (S-BGP). *Selected Areas in Communications, IEEE Journal on 18*, 4 (April 2000), 582–592.

[44] Kent, Stephen, Lynn, Charles, Mikkelson, Joanne, and Seo, Karen. Secure Border Gateway Protocol (S-BGP). *IEEE Journal on Selected Areas in Communications (JSAC) 18* (2000), 103–116.

[45] Kharaghani, H., and Tayfeh-Rezaie, B. Hadamard matrices of order 32. *Journal of Combinatorial Designs 21*, 5 (May 2012), 212–221.

[46] Khetrapal, Gautam, and Sharma, Saurabh Kumar. Demystifying Routing Services in Software-Defined Networking. Tech. rep., Aricent Inc., 2013.

[47] Khondoker, R., Reuther, B., Schwerdel, D., Siddiqui, A., and Muller, P. Describing and selecting communication services in a service oriented network architecture. In *Kaleidoscope: Beyond the Internet? - Innovations for Future Networks and Services, 2010 ITU-T* (2010), pp. 1–8.

[48] Kim, Tiffany Hyun-Jin, Basescu, Cristina, Jia, Limin, Lee, Soo Bum, Hu, Yih-Chun, and Perrig, Adrian. Lightweight source authentication and path validation. In *Proceedings of ACM Conference on SIGCOMM* (2014), pp. 271–282.

[49] Korkmaz, Turgay., and Krunz, Marwan. Multi-constrained Optimal Path Selection. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE* (2001), vol. 2, pp. 834–843 vol.2.

[50] Krovetz, T. UMAC: Message Authentication Code using Universal Hashing. RFC 4418 (Informational), March 2006.

[51] Li, Jiangtao, and Li, Ninghui. Policy-hiding access control in open environment. In *Proc. of ACM PODC* (New York, NY, USA, 2005), pp. 29–38.

[52] Li, Zhenjiang, and Garcia-Luna-Aceves, J. J. A Distributed Approach for Multi-constrained Path Selection and Routing Optimization. In *Proceedings of the 3rd International Conference on Quality of Service in Heterogeneous Wired/Wireless Networks* (New York, NY, USA, 2006), QShine '06, ACM.

[53] Liu, Xin, Li, Ang, Yang, Xiaowei, and Wetherall, David. Passport: Secure and adoptable source authentication. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (2008), pp. 365–378.

[54] Liu, Xin, Yang, Xiaowei, and Lu, Yanbin. To filter or to authorize: Network-layer dos defense against multimillion-node botnets. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication* (Seattle, WA, USA, Aug. 2008), SIGCOMM '08, pp. 195–206.

[55] Liu, Xin, Yang, Xiaowei, and Xia, Yong. NetFence: preventing internet denial of service from inside out. *SIGCOMM Comput. Commun. Rev. 41*, 4 (Aug. 2010).

[56] Liu, Yu, and Boncelet, C.G. The CRC-NTMAC for noisy message authentication. *Information Forensics and Security, IEEE Transactions on 1*, 4 (2006), 517–523.

[57] Lloyd, Stuart P. Least Squares Quantization in PCM. *Information Theory, IEEE Transactions on 28*, 2 (Mar 1982), 129–137.

[58] Martins, E. Q. V. *Bibliography of papers on Multiobjective Optimal Path Problems*. http://www.mat.uc.pt/∼eqvm/cientificos/biblio/mo.ps.Z, 1996.

[59] Martins, Ernesto Queirs Vieira. On a Multicriteria Shortest Path Problem. *European Journal of Operational Research 16*, 2 (1984), 236 – 245.

[60] Medina, Alberto, Lakhina, Anukool, Matta, Ibrahim, and Byers, John. BRITE: An Approach to Universal Topology Generation. In *Proceedings of the Ninth International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (Washington, DC, USA, 2001), MASCOTS '01, IEEE Computer Society, pp. 346–.

[61] Moy, John. OSPF version 2. RFC 2328, Network Working Group, Apr. 1998.

[62] Naous, Jad, Walfish, Michael, Nicolosi, Antonio, Mazières, David, Miller, Michael, and Seehra, Arun. Verifying and enforcing network paths with ICING. In *Proc. of ACM CoNEXT* (New York, NY, USA, 2011), pp. 30:1–30:12.

[63] Open Networking Foundation. *Software-defined Networking: The New Norm for Networks*, 2012.

[64] Pan, Jianli, Paul, Subharthi, and Jain, Raj. A survey of the research on future internet architectures. *IEEE Communications Magazine 49*, 7 (2011), 26–36.

[65] Parno, Bryan, Perrig, Adrian, and Andersen, Dave. SNAPP: Stateless network-authenticated path pinning. In *Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security (ASIACCS)* (2008), pp. 168–178.

[66] Paxson, Vern. An analysis of using reflectors for distributed denial-of-service attacks. *SIGCOMM Comput. Commun. Rev. 31*, 3 (July 2001), 38–47.

[67] Pelegrin, Blas, and andez, Pascual Fern. On the sum-max bicriterion path problem. *Computers & Operations Research 25*, 12 (1998), 1043 – 1054.

[68] Popa, Lucian, Egi, Norbert, Ratnasamy, Sylvia, and Stoica, Ion. Building extensible networks with rule-based forwarding. In *Proc. of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI)* (Berkeley, CA, USA, 2010), pp. 1–6.

[69] Saurabh, S., and Sairam, A.S. Linear and remainder packet marking for fast IP traceback. In *Fourth International Conference on Communication Systems and Networks (COMSNETS)* (2012), pp. 1–8.

[70] Savage, Stefan, Wetherall, David, Karlin, Anna, and Anderson, Tom. Practical network support for IP traceback. *SIGCOMM Comput. Commun. Rev. 30*, 4 (2000), 295–306.

[71] Snoeren, Alex C., Partridge, Craig, Sanchez, Luis A., Jones, Christine E., Tchakountio, Fabrice, Kent, Stephen T., and Strayer, W. Timothy. Hash-based IP traceback. In *Proc. of ACM SIGCOMM* (New York, NY, USA, 2001), pp. 3–14.

[72] Song, Dawn Xiaoding, and Perrig, A. Advanced and authenticated marking schemes for IP traceback. In *Proc. of IEEE INFOCOM* (2001), vol. 2, pp. 878–886.

[73] Spring, N, Mahajan, R, Wetherall, D, and Anderson, T. Measuring ISP topologies with Rocketfuel. *Networking, IEEE/ACM Transactions on 12*, 1 (2004), 2–16.

[74] Suo, K., Zhao, Y., Chen, W., and Rao, J. An analysis and empirical study of container networks. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications* (2018), pp. 189–197.

[75] Tsaggouris, George, and Zaroliagis, Christos. Multiobjective Optimization: Improved FPTAS for Shortest Paths and Non-linear Objectives with Applications. In *Algorithms and Computation*, Tetsuo Asano, Ed., vol. 4288 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2006, pp. 389–398.

[76] Ulungu, E. L., and Teghem, J. Multi-objective Combinatorial Optimization Problems: A Survey. *Journal of Multi-Criteria Decision Analysis 3*, 2 (1994), 83–104.

[77] Wang, Haodong, Sheng, Bo, Tan, C.C., and Li, Qun. Comparing symmetric-key and public-key based security schemes in sensor networks: A case study of user access control. In *Proc. of the 28th IEEE International Conference on Distributed Computing Systems (ICDCS)* (June 2008), pp. 11–18.

[78] Wang, Z., and Crowcroft, J. Quality-of-service routing for supporting multimedia applications. *Selected Areas in Communications, IEEE Journal on 14*, 7 (Sep 1996), 1228–1234.

[79] Warburton, A. Approximation of Pareto Optima in Multiple-objective, Shortest-path Problems. *Oper. Res. 35*, 1 (Feb. 1987), 70–79.

[80] Waxman, B.M. Routing of multipoint connections. *Selected Areas in Communications, IEEE Journal on 6*, 9 (Dec 1988), 1617–1622.

[81] Wolf, T., Natarajan, S., and Vasudevan, K. T. High-performance capabilities for 1-hop containment of network attacks. *IEEE/ACM Transactions on Networking 21*, 6 (December 2013), 1931–1946.

[82] Wolf, Tilman. Service-centric end-to-end abstractions in next-generation networks. In *Proc. of IEEE ICCCN* (Arlington, VA, 2006), pp. 79–86.

[83] Wolf, Tilman, Griffioen, James, Calvert, Kenneth L., Dutta, Rudra, Rouskas, George N., Baldin, Ilya, and Nagurney, Anna. ChoiceNet: Toward an Economy Plane for the Internet. *SIGCOMM Comput. Commun. Rev. 44*, 3 (July 2014), 58–65.

[84] Wolf, Tilman, Griffioen, James, Calvert, Kenneth L., Dutta, Rudra, Rouskas, George N., Baldine, Ilia, and Nagurney, Anna. Choice as a principle in network architecture. In *Proc. of ACM Annual Conference of the Special Interest Group on Data Communication (SIGCOMM)* (Helsinki, Finland, Aug. 2012), pp. 105–106. (Poster).

[85] Wolf, Tilman, Griffioen, James, Calvert, Kenneth L., Dutta, Rudra, Rouskas, George N., Baldine, Ilia, and Nagurney, Anna. ChoiceNet: toward an economy plane for the Internet. *ACM SIGCOMM Computer Communication Review 44*, 3 (July 2014), 58–65.

[86] Xie, Liehua, Arce, G. R., and Graveman, R. F. Approximate image message authentication codes. *Multimedia, IEEE Transactions on 3*, 2 (2001), 242–252.

[87] Xue, Guoliang, Zhang, Weiyi, Tang, Jian, and Thulasiraman, K. Polynomial Time Approximation Algorithms for Multi-Constrained QoS Routing. *Networking, IEEE/ACM Transactions on 16*, 3 (June 2008), 656–669.

[88] Yaar, Abraham, Perrig, Adrian, and Song, Dawn. SIFF: A stateless internet flow filter to mitigate ddos flooding attacks. In *IEEE Symposium on Security and Privacy* (May 2004), pp. 130–143.

[89] Yang, Xiaowei, Wetherall, David, and Anderson, Thomas. TVA: a DoS-limiting network architecture. *IEEE/ACM Transactions on Networking 16*, 6 (December 2008), 1267–1280.

[90] Yu, Shui, Zhou, Wanlei, Doss, R., and Jia, Weijia. Traceback of ddos attacks using entropy variations. *Parallel and Distributed Systems, IEEE Transactions on 22*, 3 (March 2011), 412–425.

[91] Zhang, Xin, Hsiao, Hsu-Chun, Hasker, Geoffrey, Chan, Haowen, Perrig, Adrian, and Andersen, David G. SCION: Scalability, control, and isolation on next-generation networks. In *Proc. of IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2011), pp. 212–227.

[92] Zhou, Xiaobo, Ippoliti, Dennis, and Boult, Terrance. Hop-count based probabilistic packet dropping: Congestion mitigation with loss rate differentiation. *Comput. Commun. 30*, 18 (2007), 3859–3869.