Toward Practical Inter-Domain Source Address Validation

Xiaoliang Wang, Ke Xu, *Fellow, IEEE, Member, ACM*, Yangfei Guo, Haiyang Wang, Songtao Fu, Qi Li, *Senior Member, IEEE*, Bin Wu, and Jianping Wu, *Fellow, IEEE*

***Abstract*— The Internet Protocol (IP) is the most fundamental building block of the Internet. However, it provides no explicit notion of packet-level authenticity. Such a weakness allows malicious actors to spoof IP packet headers and launch a wide variety of attacks. Meanwhile, the highly decentralized management of Internet infrastructure makes large-scale source address validation challenging in terms of overhead, validity, and flexibility. This paper presents a practical anti-spoofing approach, Source Address Validation Architecture eXternal (SAVA-X). SAVA-X introduces the concept of Address Domain to enable address validation in finer, prefix-level granularity. The address domains are organized in nested hierarchies to provide higher scalability and lower maintenance costs for partial deployment. We implement SAVA-X on commercial backbone routers and the P4 platform. The experiments indicate that the hardware implementation of SAVA-X can achieve 98% throughput on 100 Gbps links and close to the native IP forwarding in per-packet overhead, with less than 10 microseconds additional processing latency.**

***Index Terms*— Network security, source address validation, hierarchical validation.**

I. Introduction

THE enormous success of the Internet has made the Internet Protocol (IP) a primary building block in modern network architecture. However, this protocol does not provide any address, prefix, or even Autonomous System(AS)-level authentication for security. There is no guarantee that a received IP packet is from the claimed sender. Such a weakness allows malicious actors to forge IP packet headers [1], [2] and triggers various attacks in Internet communications. For example, DHCP spoofing, resource exhaustion, DNS reflection DDoS, DNS cache poisoning, TCP session hijacking, and TCP SYN Flooding, [3], [4], [5] to name just a few. For large-scale DoS attacks, IP spoofing remains the primary attack vector and keeps growing[1,2] [6], [7], [8]. Especially for a BitTorrent DDoS attack [9] that emerged in 2007. It is still attacking by spoofing victim IP in 2022 and keeps growing rapidly.[3]

Although we already have plenty of technical means to deal with similar attacks, such as DNS verification and intrusion detection systems (IDS), forwarding such malicious traffic instead of filtering it out as early as possible may cause the Internet to suffer from a severe waste of network resources. Moreover, source address spoofing, a vulnerability at the network layer, will likely result in new security challenges for upper-layer applications. For example, [10], [11], and [12], in these attacks, source address spoofing is used to impersonate the victim server or client to accomplish DoS or side-channel attacks.

The lack of source address validation (SAV) is a critical architectural limitation and the principal cause of the above issues. To address this problem, the Internet Engineering Task Force (IETF) has introduced source address validation architecture (SAVA) and best current practices (BCP) recommending that networks block IP packets with spoofed source addresses [13], [14], [15], [16]. Although SAV for edge networks has also made corresponding gains for detecting spoofed and reflected traffic [17], inter-domain schemes are still the primary mechanism for the Internet to perform anti-spoofing due to better cost-effectiveness. Several approaches have also been proposed to detect and filter spoofed traffic [16], [18], [19], [20], [21], [22], [23], [24]. These existing schemes use a consistent paradigm that establishes a binding relationship between a specific source IP address and detectable packet characteristics (e.g., routing direction, number of hops, packet tag). When traffic arrives with mismatched values, it is considered spoofed and filtered out. However, with the large scale of the Internet, practical implementation of SAV still faces several important challenges:

Manuscript received 25 October 2022; revised 18 April 2023, 31 August 2023, and 12 January 2024; accepted 5 March 2024; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor K. Chen. Date of publication 19 March 2024; date of current version 20 August 2024. This work was supported in part by the National Key Research and Development Program of China under Grant 2022YFB3102303; in part by the National Natural Science Foundation of China (NSFC) under Grant U22B2031, Grant 61932016, Grant 62202258, and Grant 62132011; in part by the National Science Foundation for Distinguished Young Scholars of China under Grant 61825204; and in part by the Beijing Outstanding Young Scientist Program under Grant BJJWZYJH01201910003011. *(Corresponding author: Ke Xu.)*

Xiaoliang Wang is with the Information Engineering College, Capital Normal University, Beijing 100048, China (e-mail: wangxiaoliang@cnu.edu.cn).

Ke Xu and Jianping Wu are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China, and also with the Zhongguancun Laboratory, Beijing 100094, China (e-mail: xuke@tsinghua.edu.cn; jianping@cernet.edu.cn).

Yangfei Guo is with the Zhongguancun Laboratory, Beijing 100094, China (e-mail: guoyangfei@zgclab.edu.cn).

Haiyang Wang is with the Department of Computer Science, University of Minnesota Duluth, Duluth, MN 55904 USA (e-mail: haiyang@d.umn.edu).

Songtao Fu is with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: fust18@mails.tsinghua.edu.cn).

Qi Li is with the Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing 100084, China, and also with the Zhongguancun Laboratory, Beijing 100094, China (e-mail: qli01@tsinghua.edu.cn).

Bin Wu is with H3C, Hangzhou 310052, China (e-mail: bin_wu@h3c.com).

Digital Object Identifier 10.1109/TNET.2024.3377116

[1]https://www.wired.com/story/github-ddos-memcached/

[2]https://krebsonsecurity.com/2016/09/the-democratization-of-censorship/

[3]https://blog.cloudflare.com/cloudflare-ddos-threat-report-2022-q3/

1) Clear security benefits. Lack of clear security benefits will prevent network operators from deploying SAV. For instance, BCP 38 [15] only guarantees that no spoofed traffic is generated in deployed ASes, but non-deployed ASes still could spoof source addresses belonging to the deployed range.

2) Scalability of deployment. The efficacy of any source address validation (SAV) system hinges on its ability to excel across a spectrum of deployment scenarios. Ideally, it should function seamlessly in environments with limited nodes, while gracefully scaling to encompass the intricacies of large-scale deployments. Moreover, it should provide a smooth and reliable migration path from smaller setups to their larger counterparts. Cryptography-based approaches, while effective in guaranteeing end-to-end SAV by embedding verifiable tags within packets, present significant challenges in large-scale implementations. Maintaining packet signature keys across numerous ASes becomes a non-trivial hurdle, compounded by the overhead of global state synchronization caused by dynamic membership changes. In contrast, routing-based schemes, which leverage the inherent association between source addresses and specific network interfaces, boast minimal overhead. However, this efficiency comes at the expense of compromised accuracy, particularly in smaller deployments where the absence of collaborative detection mechanisms hampers effectiveness.

3) Flexibility of verification granularity. The granularity of source address validation (SAV) may be flexible to meet different deployment requirements. For example, in some countries or regions where ASN are not abundant, a single AS often covers a large physical area, such as a city or a group of different universities. In this case, the need for SAV may be limited to a specific subnet within the AS, such as a government department (due to public safety concerns, it is necessary to ensure the authenticity of the source address of the traffic interacting with it). For other inter-ISP settlement scenarios, it could treat multiple ASes as a unit, without concern for possible source address spoofing between them.

As a result, the real-world deployment of SAV in the past decades is far from being satisfactory [25], [26]. Providing a practical SAV design remains one of the most challenging issues in modern network architecture.

This paper proposes the design and implementation of a practical Inter-domain anti-spoofing solution: Source Address Validation Architecture eXternal (SAVA-X), where eXternal refers to inter-domain. SAVA-X is a cryptography-based SAV that guarantees the authenticity of packet source addresses by maintaining verifiable packet tags between ASes in the SAVA-X ecosystem. Meanwhile, unlike legacy AS-based SAV designs, SAVA-X introduces the concept of Address Domain (*AD*) to enable address validation in finer, prefix-level granularity.

We worked with major equipment manufacturers and implemented the complete SAVA-X solution on commercial routing platforms (H3C CR16K and CR19K series, Huawei NE8000 series) and the P4 platform. The experiments indicate that the hardware implementation of SAVA-X can achieve 98% throughput on 100 Gbps links and close to the native IP forwarding in per-packet overhead, with less than 10 microseconds of additional processing latency.

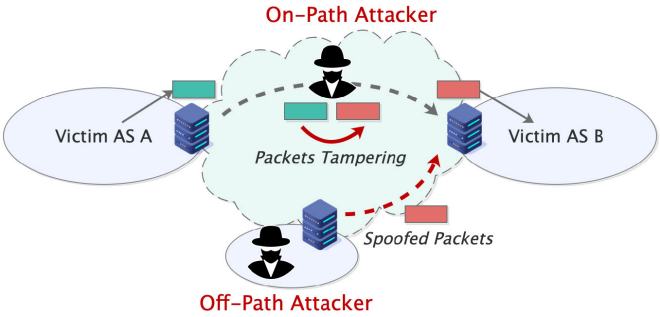

Fig. 1. Inter-domain source address spoofing scenarios addressed by SAVA-X.

The structure of this paper is as follows: in Section II, we defined the security challenges and desired goals for SAVA-X. In Section III, we give a detailed description of the design of SAVA-X. In Section IV, we provide the complete implementation of SAVA-X. In Section V, we evaluated the hardware forwarding performance of SAVA-X and gave a comparative analysis of the performance of SAVA-X on the P4 platform. In Section VI, we investigate the scalability and security of SAVA-X. After that, Section VII summarises the related works. Finally, in Section VIII, we conclude the paper.

II. Problem Definition

In this section, we discuss the threat model for inter-domain source address validation, the assumptions, and the design goals of SAVA-X.

A. Threat Model

As shown in Figure 1, the threat model for inter-domain source address validation can be described as follows.

Victim AS. Source AS A and destination AS B are interconnected through an unreliable Internet region, and their border routers cannot confirm the authenticity of the source addresses in received packets.

On-path attacker. An attacker is on the forwarding path of A and B. It can access all packets forwarded between A and B, with packet header and payload tampering capabilities. It generates traffic that spoofs the source address of A or B.

Off-path attacker. The attacker is outside the forwarding path of A and B. It generates traffic that spoofs the source address of A or B but cannot obtain information about the packets forwarded between A and B.

B. Assumption

To address the above threat model, the design of SAVA-X relies on the following assumptions:

1) There are global consistent mappings between an AS and all its legal IP prefixes. We assume that all ASes that join SAVA-X can effectively confirm each other's legitimate IP prefixes. This can be achieved through address assignment authorities or existing Internet infrastructure (e.g., RPKI [27]). Meanwhile, ASes joining the SAVA-X ecosystem are required to deploy Ingress/Egress Filtering at their borders to filter

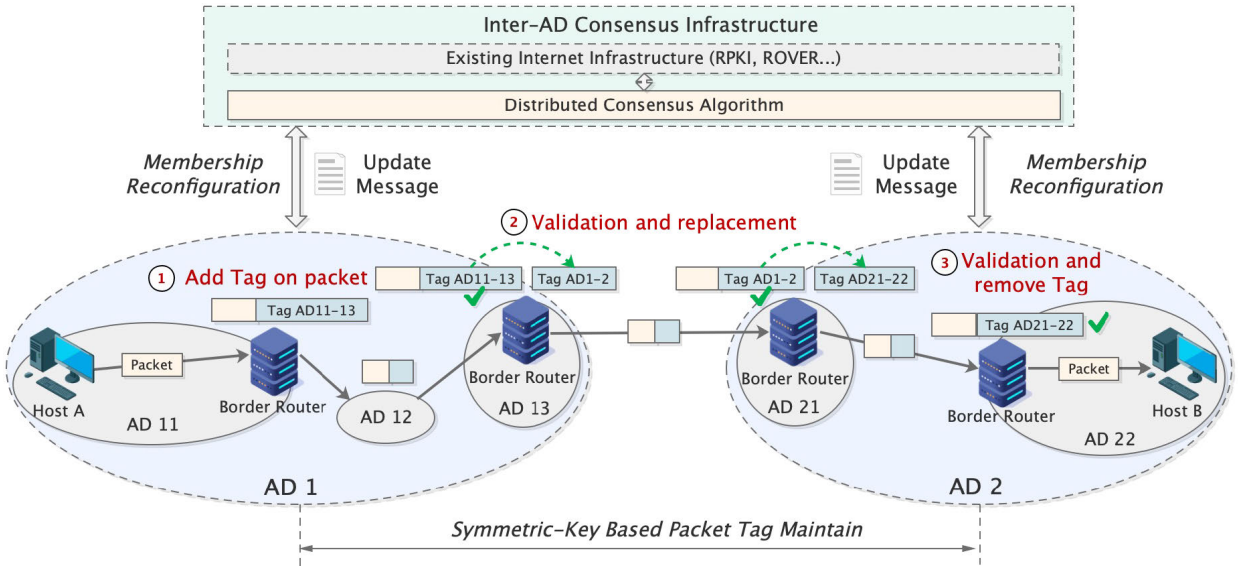

Fig. 2. Overview of SAVA-X. *AD*s with a hierarchical structure, packets carry *Tags* for replacement when forwarded across layers, and packet validation is achieved between *AD*s through key-based packet *Tags*. The consensus infrastructure maintains global consistency to achieve dynamic management of participants.

packets whose source address is not part of their own address space.

2) There is a global AS authentication. The authentication here refers to the digital certificates of AS. All the ASes that join SAVA-X can verify each other's identity through the digital certificates between them (e.g., X.509 certificate of AS).

3) The ASes can authorize the sub-prefixes assigned (e.g., an end-entity certificate [28] containing public and private keys derived from the AS). With this assumption, the ASes can provide prefix authorization in finer granularity.

C. Design Goals

Based on the above assumptions, the design goals of SAVA-X can be summarized as follows.

1) Linear security benefits. SAVA-X guarantees clear and linearly increasing security benefits with deployment scope.

2) Low key maintenance cost. SAVA-X effectively limits the number of *Key*s maintained between *AD* pairs and reduces maintenance overhead.

3) High system scalability. Local changes (such as the membership reconfiguration, the replacement of keys, etc.) do not have a global impact on the SAVA-X ecosystem.

4) Early spoof detection. SAVA-X identifies, and filters spoofed traffic close to the attacker.

5) Flexible verification granularity. SAVA-X provides prefix-level validation and clarifies the security responsibilities corresponding to IP address allocation authority.

III. Design

In this section, we present the design of SAVA-X and highlight how its hierarchical architecture addresses the problems in the legacy source address validation approaches.

A. Building Blocks

Address Domains. SAVA-X splits or merges ASes into different domains that have a set of prefixes. We call these sets as Address Domains (*AD*s). Based on the above assumptions defined in Section II-B, the SAVA-X *AD*s have the following important features:

1) The mapping between *AD* and its IP prefixes is cryptographically verifiable.

2) The *AD* has a globally consistent authentication derived from the corresponding AS authentication mechanism.

Hierarchical Address Domains. A number of *AD*s can form a new *AD* with a larger scope and a higher level of hierarchy. We refer to the larger *AD* as the *parent AD*, the smaller *AD*s contained within the *parent AD* as *child AD*s, and the *child AD*s as siblings. There is no additional trust required between brother *AD*s within the same *parent AD*. This is because SAVA-X will utilize cryptographic mechanisms to ensure that these *AD*s cannot forge each other's source addresses. However, *child AD*s need to trust their *parent AD*s (usually a provider or an upper management organization). Because all packets from *child AD*s sending to the outside of their parent need to be further processed by the *parent AD*. They need to trust the *parent AD* to perform this process honestly. If *AD*s do not trust this *parent AD*, they can just choose not to join it, at the cost of maintaining more packet keys, which we will discuss later.

The hierarchical *AD* structure satisfies the following constraints.

1) Any AD belongs to at most one parent AD directly and any parent AD contains at least one child AD.

2) The shape of a parent AD must be convex. The convex ensures that packets forwarded inside an *parent AD* (from a *child AD* to another *parent AD*) do not pass through an outside *AD*. This is important to guarantee that the packet validation logic is clear and concise (For further discussion, please see Section VI-B).

3) An AD can have at most one parent AD, and its network must be completely contained within the parent AD. Otherwise, the *AD* cannot be added as a *child AD* to the *parent AD*. This

will ensure that after a packet is sent from a particular *AD*, it will enter only one *parent AD*.

Inter-AD Consensus Infrastructure. the Inter-*AD* Consensus Infrastructure is used to manage SAVA-X membership (e.g., join or leave an alliance) and hierarchical structure of *AD*s. The infrastructure does not bind any specific method to achieve global consensus. The design of a consensus mechanism is not in the scope of this paper. See Section IV for more details about our implementation.

B. Overview

As shown in Figure 2, an AS can be divided into several hierarchical SAVA-X *AD*s (e.g., *AD*1 and *AD*11). The legitimacy mapping between *AD*s and their prefixes are provided by existing infrastructures such as RPKI [27], ROVER [29]. In this case, *AD*13 and 21 are border *AD*s of their *parent AD*s. The border *AD*s are utilized for cross-layer validation and *Tag* replacement(see Section III-E).

To explain the interactions between SAVA-X *AD*s, we will discuss the steps when *AD*11 is trying to join the SAVA-X alliance and communicate with *AD*22. It is assumed that all other *AD*s, except *AD*1, are in their places.

*Step1: AD*11 sends a join message to Inter-*AD* Consensus Infrastructure. After authentication, *AD*11 negotiates initial *Key* and maintains packet *Key* with *AD*12 and *AD*13.

Step2: Host A in *AD*11 sends a packet, the border router of *AD*11 adds a *Tag* of "`AD11-AD13`" and sends it to *AD*12. The *Tag* is generated with the packet *Key* maintained between *AD*11 and *AD*13. The detailed *Tag* generation process is attached in Section IV-C.

*Step3: AD*12 receives the packet. Since *AD*12 is not a border *AD* and the destination IP prefix of the packet does not belong to *AD*12, the packet will be directly forwarded to *AD*13.

*Step4: AD*13 receives the packet and can determine that the destination IP prefix is outside the *parent AD*1 and belongs to *AD*2. The border router of *AD*13 will validate the *Tag* "`AD11-AD13`" carried by the packet. After that, the router will replace the *Tag* "`AD11-AD13`" with "`AD1-AD2`" and send the packet to *AD*2.

*Step5: AD*21 receives the packet. Its border router will validate the packet *Tag* "`AD1-AD2`", determine that the destination IP prefix belongs to its *brother AD*22, and replace the *Tag* "`AD1-AD2`" with "`AD21-AD22`".

*Step6: AD*22 receives the packet. Its border router will validate the legality of the *Tag* "`AD21-AD22`" and confirm that the source address in *AD*11 is valid.

C. Hierarchical Key Maintenance

As shown in Figure 3, for any *AD* pairs in SAVA-X, they only need to maintain a *Key* when:

1) They are brother *AD*s within the same *parent AD*, e.g., *AD*11, *AD*12 and *AD*13.

2) They are parent-child relationship, e.g., *AD*11 and *AD*111 or *AD*112.

An *AD* within a *parent AD* can leave at any time, e.g. lack of trust in the *parent AD* or a change in the management structure. In this case, the keys with other external *AD*s that were previously maintained by the *parent AD* need to be additionally maintained by the *AD* itself. At the same time, the *AD* will no longer maintain packet keys with other *AD*s within the *parent AD*, and these internal *AD*s will no longer be visible, and the *parent AD* as a whole will establish key maintenance with the separated *AD*.

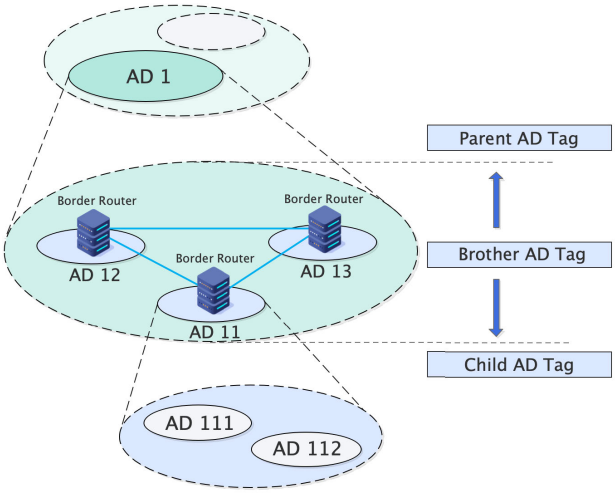

Fig. 3. Hierarchical *AD* Structure. *Tag* is maintained between *AD* pairs only in the case of parent-child or brother-brother relationship.

It is easy to see that the hierarchical *AD* can largely reduce the total number of keys in SAVA-X. This will optimize the system maintenance overhead, making supporting a larger deployment range possible. Moreover, such a hierarchy will block the external changes outside *parent AD* and provide scalability in large-scale deployment. In this case, the malicious traffic will also be filtered earlier in stub *AD*s to provide better efficiency.

D. Management of Address Domains

The Inter-*AD* Distributed Consensus Infrastructure is used to maintain the global consistency of SAVA-X membership and *AD* configurations. When a new *AD*, such as *AD*11 in Figure 3, joins SAVA-X, it needs to generate a request message to the Inter-*AD* Consensus Infrastructure and gather the signatures of the relevant *AD*s. The Inter-*AD* Consensus Infrastructure will then perform the message checking and adjust SAVA-X's latest deployment structure (e.g., *AD* connection relationships and keys' maintenance). For example, *AD*11 to join the SAVA-X ecosystem, a request message needs to be generated in the following format:

$$
\begin{aligned}
&Message_{join} \\
&= Sign_{relativeAD}\{Type = AD_{join} \\
&\quad \| ADNumber \| PublicKey_{AD11} \\
&\quad \| \{AD\ 1(AD11[BorderServerIP](AD111, AD\ 112))\}\}
\end{aligned}
$$

The first field *Type* indicates that the current message is used to verify an *AD*'s join request. The second field is the ID of the currently joined *AD*. The third field carries the public key of the *AD*11, which could be generated by itself or derived from RPKI's X.509 certificates. The fourth field shows the parent-child relationships of the currently joined *AD*. In this

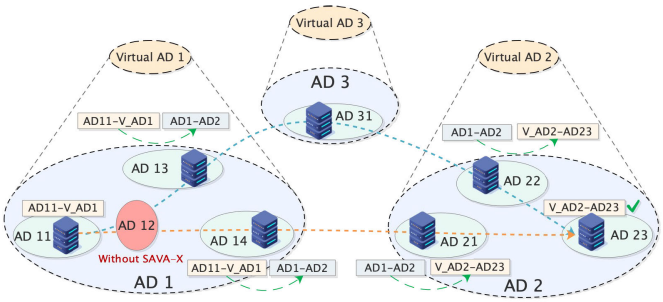

Fig. 4. Segment validation and *Tag* replacement. Simplify the processing logic of *Tag* replacement by *virtual AD* to decouple *Tag* replacement from the forwarding path selection.

example, *AD*11 is a child *AD* of *AD*1. *AD*111 and *AD*112 are both the child *AD*s of *AD*11.

After receiving an application message, the relevant *AD*s verify and sign the message, and then send it to all participating nodes through the Inter-*AD* Distributed Consensus Infrastructure and perform consensus. *AD* 11 formally joins the SAVA-X ecosystem when consensus is completed. Its public key takes effect and it can negotiate the packet *Key* with other *AD*s. The relevant *AD*s will be notified to make corresponding adjustments, and the Inter-*AD* Distributed Consensus Infrastructure will record the new *AD* structure. After *Key* negotiation is completed, *AD* 11 starts to send traffic carrying SAVA-X packet tags.

The *Type* field in the above message also supports exit, merge, split, and other types. All *AD* structure change messages require the tag of the associated *AD*s and a consensus process through the Inter-domain Distributed Consensus Infrastructure. The global consensus is used to ensure the consistency of the *AD* structure in the SAVA-X ecosystem.

E. Segment Validation

In SAVA-X, *AD*s form an onion-like structure. Tag replacement in SAVA-X should only require the structure information of the *AD*s, independent of the actual forwarding path between. To this end, SAVA-X adopted a design called *virtual border AD* (Noted as V_*AD*). The *virtual border AD* is a logic *AD* representing all the border *AD*s in a *parent AD*. This design will bring two benefits to SAVA-X. First, routers support SAVA-X only need to check if the destination IP prefix is inside or outside their *parent AD*. If the destination IP prefix is inside, add a *Tag* to the destination *AD*; if the destination IP prefix is outside, add a *Tag* to the *virtual border AD*. Second, The *parent AD*s handle member *AD*s that do not support SAVA-X. It only requires all the border *AD*s to deploy SAVA-X.

Take Figure 4 as an example. When *AD*11 sends a packet to *AD*23, the SAVA-X router in *AD*11 will check the destination IP prefix in the packet. The router will find that the destination IP prefix is outside *parent AD*1. Therefore, the router will add the *Tag* "AD11-V_AD1" indicating that the packet is validated by a SAVA-X router in *AD*11. Therefore, this tagged packet can be verified by other SAVA-X routers in *parent AD* 1 (e.g., *AD*13 *AD*14), *AD*11 also does not need to be aware of the actual subsequent forwarding path. The packet will be forwarded without *Tag*-checking for routers that do not support SAVA-X (e.g., *AD*12). After that, one of the border *AD*s (e.g., *AD*13 *AD*14) will check and replace the *Tag* with a new *Tag* "AD1-AD2" and send the packet out of *parent AD*1. This *Tag* shows that the packet is already verified by *AD*1. In Figure 4, if *AD*31 received this packet, it will directly forward the tagged packet to *AD*2 without validating the *Tag*, because the destination IP prefix does not belong to *AD*31. When *AD*2's border *AD* (e.g., *AD*21 or *AD*22) receives this packet, the *Tag* "AD1-AD2" will be verified and replaced by "V_AD2-AD23". Finally, *AD*23 will receive the packet, no matter from which border *AD*, verify the *Tag* "V_AD2-AD23", and confirm that the packet is without source address spoofing. The packet would be dropped if any check fails.

Algorithm 1 Segment Validation

Input: *AD* A receives a packet with S_{IP},D_{IP},*Tag*, $Tag_{outgoing}$,$Tag_{incoming}$

```
For the sake of clarity, all validation is considered passed, and if not, the packet is dropped;
if A is a border AD in the parent AD then
    if S_IP is in this parent AD and D_IP is out of the parent AD then
        Validation of Tag;
        Replacement Tag with Tag_outgoing ;
        Forwarding outside parent AD ;
    if S_IP is not in this parent AD and D_IP is inside of the parent AD then
        Validation of Tag;
        Replacement Tag with Tag_incoming ;
        Forwarding inside parent AD ;
    Forwarding without validation;
else
    if D_IP is inside of A then
        validation of Tag;
        Forwarding to host;
    else
        Forwarding without validation;
```

The detailed segment validation algorithm is shown in the Algorithm 1. In Algorithm 1, S_{IP} is the source IP address, D_{IP} is the destination IP address, *Tag* is carried by current packet, $Tag_{outgoing}$ is maintained by *virtual border AD*s between source *AD* and its *parent AD*, $Tag_{incoming}$ is maintained by *virtual border AD*s between the destination *AD* and its *parent AD*.

IV. Implementation

This section presents the implementation details of SAVA-X. SAVA-X comprises two primary components: the AD Control Server (ACS) and several AD Border Routers (ADBR) within each *AD*. The ACS is responsible for managing the hierarchical *AD* structure and key management, while the ADBRs are responsible for storing these keys and handling tag-processing tasks.

A. AD Control Server

We have developed a versatile AD Control Server (ACS) capable of running on any POSIX-compliant platform. This robust server facilitates the hierarchical organization of *AD* layers, seamless management of *AD* information, hierarchical key management, and efficient communication between ACS instances and ADBR devices, enabling seamless *AD* information distribution across the network.

1) Hierarchical AD Layers Division: In SAVA-X implementation, RPKI [27] serves as a prerequisite infrastructure, establishing a hierarchical binding between AS and IP address prefixes. However, the RPKI hierarchy doesn't perfectly align with SAVA-X's layered *AD* structure. *AD*s act as logical entities without disrupting existing routing policies or infrastructure. Network division into nested *AD*s is based on diverse criteria such as routing policies, geography, culture, economics, contracts, and legal considerations. To ensure compatibility and enable flexible IP address prefix composition and decomposition, it is recommended to adhere to existing routing policies, creating a layered *AD* structure. This approach achieves scalability, encapsulates *AD* information, and eliminates the need for higher layers to possess in-depth knowledge of lower layers. SAVA-X could seamlessly coexist with the current Internet infrastructure, providing hierarchical management and encapsulation of *AD* information.

2) AD Information Management: The essential information about an *AD* includes the AD number, IPv6 address prefix, and public key. However, managing *AD* information becomes more complex in a layered structure compared to a flat mode where tag processing is straightforward like source adding, destination verifying, and none of the business of forwarding nodes. Each *AD* in SAVA-X has a unique *parent AD*, and this parent AD number is stored to facilitate the identification of *ancestor AD*s. SAVA-X maintains *AD* information in two tables: the **ADID-Info Table** containing *AD Number*, *parent AD Number*, and *ACS Address* and the **ADID-Prefix Mapping Table** containing *ADID*, *Prefix Length*, and *IPv6 Address*. The generation of the *Public Key* is crucial before communication and should be securely stored. For a visual representation of these tables, refer to Figure 6.

- *AD Number (ADID)* is a 32-bit integer that serves as the unique identity within an *AD*.
- *Parent AD Number (parent ADID)* is also a 32-bit field used for searching all the *ancestor AD*s of a specified *AD*. It allows the Address Configuration Service (ACS) to determine which ACS it should connect to.
- *ACS Address* is a 128-bit IPv6 address used for establishing connections with other ACSes, with which the current ACS maintains hierarchical keys.
- *Prefix Length* is an 8-bit field that specifies the length of the IPv6 prefix.
- *IPv6 Address* is a 128-bit IPv6 address that determines which packet should be checked and protected by ADBR.
- *Public Key* - The ACS generates a public/private key pair for communication security. This key pair is protected by the current Public Key Infrastructure system and should be signed by the hierarchical Certificate Authorities [30].

3) Hierarchical Key Management: SAVA-X employs a **Time-based State Machine Key Generation** mechanism illustrated in Algorithm 2 to minimize network connection and communication costs. *AD*s maintain synchronized state machines that periodically generate keys for mutual source address validation. Unlike the flat mode, SAVA-X doesn't require a full-mesh connection among all *AD*s. Instead, *AD*s establish limited connections with essential counterparts including its *parent AD*, *brother AD*s within the same *parent AD*, and directly subordinate *child AD*s. When it starts, the ACS initiates these connections immediately based on *AD* information to configure the state machines. Two types of keys are used in SAVA-X.V_AD key is generated by the current *AD*'s state machine in conjunction with the *parent AD* for cross-layer communication and direction-specific keys are generated by separate state machine pairs for communication with *brother AD*s. The State Machine Information includes the following fields.

- *Source ADID* is an ADID that identifies the packets received from.
- *Destination ADID* is an ADID that identifies the packets sent to.
- *State Machine Algorithm* is an 8-bit field that specifies several predefined algorithms used to generate keys. We now recommend and implement using the One Time Password System based on the S/Key system [31].
- *Key Length* is an 8-bit field that defines the length of generated keys in octets.
- *IS Length* is an 8-bit field that specifies the length of *Initial State* in octets.
- *Initial State* is the negotiated bit-string based on which the state machine runs and generates keys periodically. It's a variable length field whose length relies on the *IS Length* field.
- *Interval* is a 32-bit state transition interval that defines the interval in seconds when the state machine transits its states.
- *Effecting Time* is the time to start this state machine. This is in the UNIX timestamp format.
- *Expired Time* is the time to stop this state machine. This is in the UNIX timestamp format.

4) ACS-ACS Communication Functions: The communication between ACSes utilizes TCP [32] and SSL/TLS [33]. When an ACS establishes a connection with another ACS, the ACS with a bigger ADID initiates the negotiation process for the state machine. The peer ACS determines the state machine information, as specified in Section IV-A3. Once the negotiation is completed, both ACS systems activate the state machine at the designated *Effecting Time* and terminate it promptly at the *Expired Time* if it is already running. To ensure continuous service availability, ACS should implement the **Keep-Alive** mechanism. In the event of failure detection by one ACS, the peer ACS is considered dead. Subsequently, the ACS will re-establish the connection with the peer ACS and report the failure after reaching the maximum number of retry attempts.

Algorithm 2 Time-Based State Machine Key Generation Algorithm

Input: $SourceADID$, $DestinationADID$, $Algorithm$, $KeyLength$, $ISLength$, $InitialState$, $Interval$, $EffectingTime$, $ExpiredTime$

State Machine SM is ready after ACSes negotiation.
Function *startSM()*
    $S_n = InitialState$;
    $Key_n = None$;
    $time_{effect} = EffectingTime$;
    wait until $time_{effect} <= NowTime$;
    do
        if $time_{effect} <= NowTime$ **then**
            $S_{n+1}, Key_{n+1} = Algorithm(S_n, Key_n)$;
            $S_n = S_{n+1}$;
            $Key_n = Key_{n+1}$;
            $time_{effect} = time_{effect} + Interval$;
        sleep 1 second;
    while $NowTime < ExpiredTime$;

B. AD Border Router

The AD Border Router (ADBR) is responsible for providing prefix-checking and tag-processing functions. We have successfully implemented these functions of ADBR in a series of commercial backbone routers produced by major manufacturers. Additionally, ADBR needs to establish communication with ACS to receive *AD* information and hierarchical keys. ADBRs should be strategically positioned at the borders of nested *AD*s to enable smooth communication and secure data exchange between the different layers of the network architecture.

1) ACS-ADBR Communication Functions: The ADBR establishes connections with ACS using TCP and SSL/TLS protocols. It incorporates the **Keep-Alive** mechanism with ACS to maintain an active connection. ADBR maintains two tables for SAVA-X. The **ADID-Key Mapping Table** stores information about the corresponding *AD*, including the *ADID*, two keys, and their *Expiration Time*s. Due to the slight time difference between devices on the Internet, NTP is used to synchronize time, ensuring that both keys are updated simultaneously. The newer key is used for adding and verifying *Tag*s, while the older one is only employed for verifying *Tag*s. The **ADID-Prefix Mapping Table** contains the *IPv6 Prefix* information of the corresponding *AD*. Within the data plane, the fields *Level*, *ADs List*, and *Common Ancestor Level* are utilized to determine which key should be applied to packets with specific source and destination IPv6 addresses.

- *ADID*, *IPv6 Prefix*, and *Prefix Length* are the same as described in Section IV-A2.
- *Level* means the level of the corresponding *AD*. It starts from 0 for root *AD*. A larger value means the *AD* is at a lower level.
- *ADs List* is an array that lists all the ADIDs from the current *AD* to its *parent AD*.
- The *Common Ancestor Level* serves a similar purpose to the *Level* field, as it indicates the common ancestor level between the corresponding *AD* and the *AD* where the ADBR is located.

In SAVA-X, tags have directions, and they could be determined using the pair <*Source ADID, Destination ADID*>. The ADID range is divided into two parts, and each *AD* has two ADIDs in each part. The only difference between these ADIDs is the reversal of their highest bits. An ADID with the highest bit of 0 represents the *AD* itself. An ADID with the highest bit of 1 is used for traversing the *AD* to its *parent AD* or *child AD*s, which are *virtual AD*s explained in Section III-E. A packet adopts the virtual ADID when crossing layers.

2) ADBR Interface Types: Prefix checking and tag processing functions in an ADBR are determined based on the roles of different ADBR interfaces. Each interface in an ADBR can be categorized as either *ingress*, *egress*, or *trust*. The ingress port is responsible for connecting to devices within the *AD*, while the egress port connects to devices outside the *AD*. The trust port serves as the link to the ADBR within the same *AD*. To specify the level at which an interface sits in the hierarchical *AD*s, a router command requires an egress level field, i.e., $Level_L$. This manually assigned $Level_L$ field is utilized in packet processing operations.

3) Prefix Checking and Tag Processing: The packet processing in an ADBR involves two modules: the inward processing module and the outward processing module. Algorithm 3 describes the process on ADBR.

Upon arrival at the ADBR, a packet enters the inward processing module and undergoes classification based on its destination address. If the destination address does not belong to any SAVA-X deployed *AD*s, the ADBR directly forwards the packet. However, if the destination address belongs to a member *AD*, the ADBR further classifies the packet based on the type of its arriving port. In the case of an ingress port, the ADBR forwards the packet directly if its source address is not associated with the *AD* where the ADBR is located. Otherwise, the ADBR forwards the packet for outward processing. For packets received from an egress port, the ADBR determines the *Common Ancestor Level* ($level_D$) between the destination address and the ADBR by consulting the **ADID-Prefix Mapping Table**. The ADBR also obtains the local egress port level ($level_L$). If $level_L \leq level_D$, the ADBR checks the packet's tag for validation. The tag can be accessed from the **ADID-Key Mapping Table** using $level_L$ to obtain the ADID from the *ADs List* in the *ADID-Prefix Mapping Table*.

If the packet fails the tag validation, the ADBR drops it. However, if the tag is correct, the ADBR determines whether the packet needs a new tag based on the comparison between the corresponding *Level* and *Common Ancestor Level* in the **ADID-Prefix Mapping Table**. If *Level* == *Common Ancestor Level*, a new tag is added to replace the old tag using the **ADID-Key Mapping Table** with $level_D + 1$ to obtain the ADID. Finally, the packet is forwarded to the outward processing module.

In the outward processing module, before forwarding a packet that has undergone inward processing, the ADBR

Algorithm 3 Process on ADBR

Input: Interface I of ADBR R of *AD* A receives a packet with S_{IP}, D_{IP}, *Tag*

For the sake of clarity, the packet rises from and sinks to two different ADs deployed with SAVA-X, and if not, the packet is forwarded directly;
if I *is a trust port* **then**
  Forwarding without checking and validation;
else
  if *Packet direction is Inward* **then**
    if I *is an ingress port* **then**
      if S_{IP} *is inside of* A and D_{IP} *is not inside of* A **then**
        Adding *Tag*;
    if I *is an egress port* **then**
      if D_{IP} *is inside of* A or A *is the border* AD *of* parent AD **then**
        Validation of *Tag*;
    Forwarding to outward processing module;
  else
    if I *is an ingress port* **then**
      if S_{IP} *is inside of* A and D_{IP} *is not inside of* A **then**
        Validation of *Tag*;
        Removing *Tag*;
      Forwarding to Host;
    if I *is an egress port* **then**
      if D_{IP} *is inside of* A or A *is the border* AD *of* parent AD **then**
        Validation of *Tag*;
        if D_{IP} *is not inside of* A **then**
          Replacement of *Tag*;
          Forwarding out;
        else
          Forwarding to Host;

compares the *Level* ($level_L$) of the outgoing interface with the *Common Ancestor Level* between the ADBR and either the source address ($level_S$) or destination address ($level_D$). If $level_L > level_S \&\& level_L > level_D$, the ADBR directly forwards the packet without requiring a tag.

However, if $level_S \geq level_L$ and the packet is marked with a tag, the ADBR verifies the tag by extracting the ADID using $level_S + 1$ from the *ADs List* in order to determine the validation key from the **ADID-Key Mapping Table**. Subsequently, the packet is assigned a new tag generated from the **ADID-Key Mapping Table** using $level_L$, and then forwarded out.

C. Tag Generation

The tag generation mechanism in SAVA-X, inspired by Passport [23] and DIA [24], involves placing the 64-bit tag in the IPv6 destination option header. The tag consists of a 32-bit packet message authentication code (*PktMAC*, hash digest), an 8-bit prefix length, and 24 reserved bits. This design enables SAVA-X to protect address prefixes of various lengths and authenticate full-length 128-bit IPv6 source addresses when deployed on the access device. However, due to the complexity of SAVA-X, it is generally recommended to use a coarser-grained protection scope. The reserved 24 bits in the tag can be utilized for future protocol extensions, such as supporting special services or calibrating the security level of authenticated source address traffic.

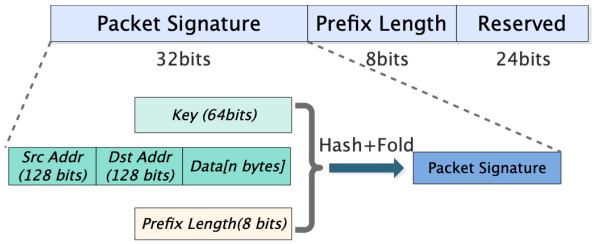

Fig. 5. Tag generation procedure.

The *PktMAC* serves the purpose of ensuring the authenticity of the packet source and the integrity of the packet itself. It follows the existing cryptography-based inter-domain SAV approach. Initially, the source *AD* generates a key between two *AD*s by maintaining a state machine mechanism at the ACS. By default, the generated key has a length of 64 bits, which can be specified using the *Key Length* field during negotiation. The 128-bit source address, 128-bit destination address, the first n bytes of the packet payload (Data[n]), and the 8-bit prefix length are combined and hashed using a hash function. The appropriate folding algorithm is then applied, resulting in the final 32-bit *PktMAC*. The hash function can be uhash or CRC32, both of which produce a 32-bit output. Additionally, other hash algorithms with longer output lengths, such as SipHash, MD5, or SHA256, can be used by collapsing the output to obtain the 32-bit *PktMAC*, given that the generated keys' length is adjusted accordingly.

Unlike some existing cryptographic schemes, such as using AES to encrypt the hashed data in DIA, SAVA-X relies solely on hash algorithms for tag generation. This choice strikes a balance between security and performance in SAVA-X. Although using only hash functions may increase the collision probability, the focus in inter-domain forwarding scenarios is more on minimizing performance loss. Experimental results indicate that SAVA-X's tag generation method and hierarchical *AD* design can support commercial routing platforms to achieve high-performance forwarding capabilities.

D. Example of SAVA-X

In Figure 6, emAD1 is positioned at the same level as *AD*2. Within *AD*1, we have *AD*3 and *AD*4, while *AD*5 is a direct subordinate of *AD*2. Each *AD* contains its own ACS, ADBRs, and several end hosts.

Initially, all *AD*s should be deployed with SAVA-X enabled in their ADBRs. Subsequently, the ACS of each *AD* will analyze the ACSes with which it should maintain a state machine

and generate corresponding keys. As discussed earlier, the state machine should be maintained by *AD*s at the same level or in a direct parent-child relationship. The ACS will deliver these keys, along with the *AD*s' information, to their respective ADBRs to form the **ADID-Prefix Mapping Table** and **ADID-Key Mapping Table**. Since keys are generated periodically, the ACS will periodically deliver the keys, while announcing only the updated part when referring to *AD* information. Once these steps are completed, communication can commence among the end hosts located in different *AD*s.

Communication between AD1 and AD2: *AD*1 and *AD*2 are positioned at the same level, thus requiring the maintenance of a state machine to generate keys. When the SAVA-X feature is enabled, an end host H1 located in *AD*1 sends a packet to an end host H2 located in *AD*2. The ADBR of *AD*1 receives the packet sent by H1 and determines that the packet requires a tag by checking if the source address belongs to *AD*1 and if it is destined for *AD*2. Consequently, the outward interface of *AD*1's ADBR adds a tag "`AD1-AD2`" within the packet's destination option header. Then, *AD*1's ADBR sends out the packet. When *AD*2's ADBR receives the packet, it identifies that the packet carries a tag. It then determines which tag the packet should have by checking the packet's source and destination addresses, as well as the inward interface level of this ADBR. The ADBR compares the carried tag with the expected tag and drops the packet if the comparison fails. Otherwise, it removes the tag and forwards the packet to the outward interface for further processing. As the outward interface of *AD*2's ADBR is an ingress port, it directly forwards the packet.

Communication between AD3 and AD5: *AD*3 and *AD*5 are located in different superior *AD*s, requiring tag replacement when an end host H3 in *AD*3 communicates with an end host H5 in *AD*5. When *AD*3's ADBR receives the packet sent by H3, it first checks the packet's address to confirm that the source address prefix belongs to *AD*3 and the destination address prefix belongs to *AD*5, indicating that their communication needs SAVA-X. It then determines that the packet's destination address is not within *AD*1, requiring the tag attached to the packet to indicate that it will be sent out of *AD*1. The tag "`AD3-V_AD1`" is used for this purpose. The left ADBR of *AD*4 does nothing to the packet since both the source and destination address prefixes do not belong to *AD*4. The right ADBR of *AD*4 checks the carried tag and replaces it with "`AD1-AD2`". When *AD*2's ADBR receives the packet, it processes it as described in IV-D. At this point, the packet sent from *AD*3 to *AD*5 is officially terminated.

V. Evaluation

In this section, we assess the forwarding performance of SAVA-X using commercial inter-domain routers and P4 programmable switches, respectively.

A. Evaluation on Commercial Router

As mentioned in Section IV, the data plane of SAVA-X is responsible for generating, verifying, and replacing tags.

ADID-Prefix Mapping Table of R1

ADID	Prefix
AD1	P1
AD2	P2
AD3	P3
AD4	P4

ADID-Info Table of R1

ADID	Parent ADID	Public Key	ACS Address
AD1	—	PK1	ACS1
AD2	—	PK2	ACS2
AD3	AD1	PK3	ACS3
AD4	AD1	PK4	ACS4
AD5	AD2	PK5	ACS5

ADID-Key Mapping Table of R1

Src ADID	Dst ADID	Key
AD1	AD2	K12
AD2	AD1	K21
AD3	AD4	K34
AD4	AD3	K43
AD1	AD4	K14
AD4	AD1	K41

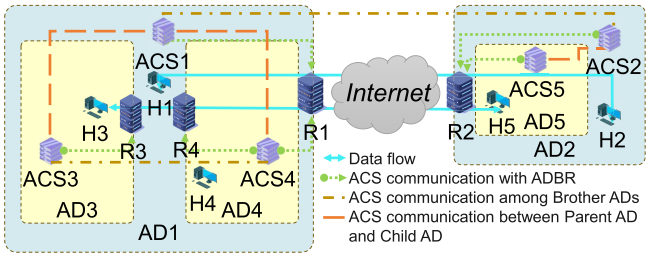

Fig. 6. An Example of SAVA-X hierarchical deployment and packet tag processing.

On the other hand, the control plane of SAVA-X is implemented on a side-mounted server. The control plane manages the AD structure, negotiates keys within the SAVA-X ecosystem, and distributes keys to the data plane.

For our experiments, we focused on the H3C CR19K routers depicted in Figure 7-d. We constructed a nested AD hierarchy using nine routers following the pattern R1-R2-R3-R4-R5-R6-R7-R8-R9, as shown in Figure 8. Each interface of these routers has a capacity of 100 Gbps. By leveraging the hardware virtualization capabilities of the H3C CR19K routers, R1-R4 were derived from one physical router with eight ports, while R5-R9 were derived from another physical router with ten ports. In this topology, R1 and R9 functioned as normal *AD*s, R5 represented an independent *AD*3, and the remaining routers served as border *AD*s in different layers. We organized *AD*1, *AD*2, and *AD*3 as the tier 1 layer, establishing a symmetric network topology with four additional layers on each side. To gather experimental data, we utilized traffic meters to measure specific router connections within the topology. This included 1-hop (R1-R2), 2-hops (R4-R5-R6), 4-hops (R3-R4-R5-R6-R7), 6-hops (R2-R3-R4-R5-R6-R7-R8), and 8-hops (R1-R2-R3-R4-R5-R6-R7-R8-R9) scenarios.

The testing traffic is generated using an ISP's real-world Internet traffic statistics. In this dataset, over 85% of the data packets have a size smaller than 500 bytes. The average packet size is 424 bytes, and the median packet size is 280 bytes.

1) One-Way Delay With Different Packet Sizes in 1-hop: One-way delay is a standard for IP performance measurement defined in [34]. We measured the per-packet one-way delay of SAVA-X and the baseline (native IP forwarding) on the link between R1 and R2. The measurements were taken under light load conditions, i.e. 10 Gbps, with no queuing, and the results are depicted in Figure 7-a. From the figure, it can be observed that the per-packet one-way delay under light load exhibits a slight increase as the packet size increases. While SAVA-X introduces additional processing overhead compared to native IP forwarding, the resulting latency increase is minimal. It is important to note that these increased latency

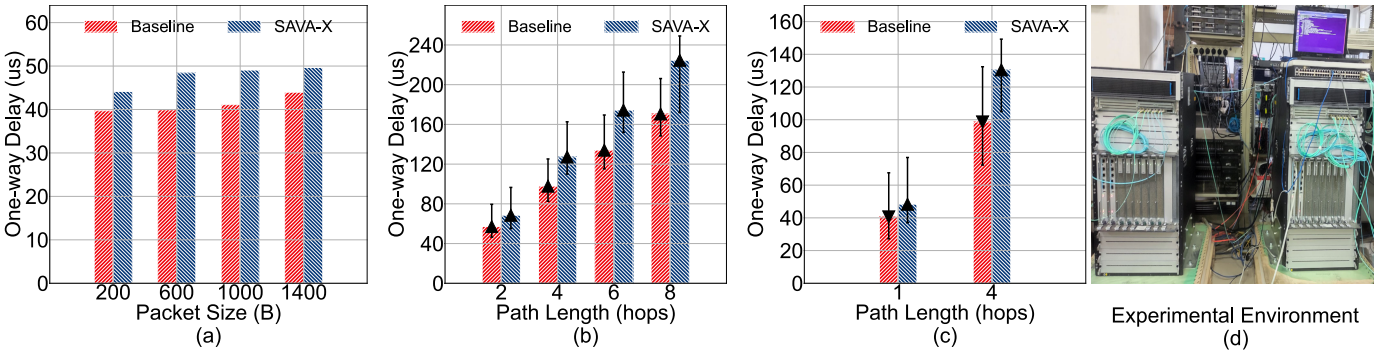

Fig. 7. a. One-way delay with different packet sizes in 1-hop. b. One-way delay of fixed packet size with different path lengths. c. One-way delay of mixed packet size with different path lengths. d. The commercial router and experimental environment.

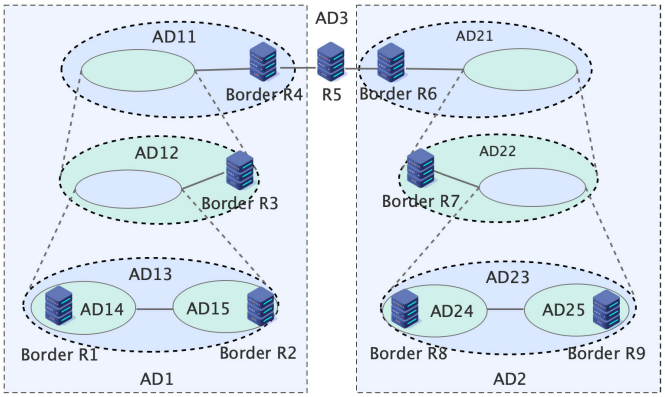

Fig. 8. Topology of Evaluation.

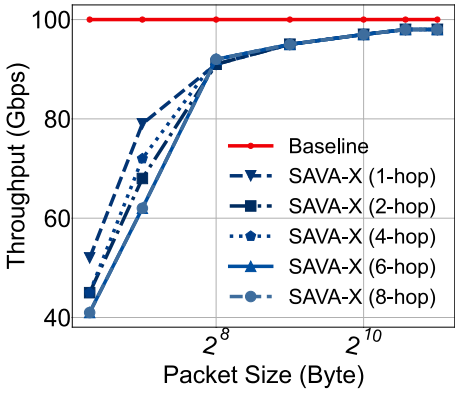

Fig. 9. Throughput with fixed packet size.

values are very small when compared to the typical end-to-end latency of the Internet, which is usually in the tens of milliseconds.

2) One-Way Delay With Different Path Lengths: Considering that SAVA-X requires per-packet processing, we measured the one-way delay in two scenarios: fixed packet size and mixed packet size. According to CAIDA's measurement data, the average Internet packet size is around 600 bytes.[4] For this experiment, we fixed the packet size to 512 bytes. We measured the latency under different incoming traffic rates, ranging from 1 Gbps to the maximum throughput, for each topology. We recorded the average performance of SAVA-X and the baseline (native IP forwarding). The results are shown in Figure 7-b. In the 2-hop topology (i.e., R4-R5-R6), the additional latency introduced by SAVA-X is minimal. This is because *AD*1 and *AD*2 are in the same layer, eliminating the need for segment validation. However, as the number of hops increases to four (R3-R4-R5-R6-R7), there is a significant difference between SAVA-X and the baseline. This is because both R4 and R6 require segment validation. To evaluate the performance using real-world traffic, we conducted a latency test with mixed packet sizes on the 1-hop and 4-hop topologies. The results are presented in Figure 7-c. In the 1-hop experiment, the additional latency introduced by SAVA-X is only 7.79 μs. However, in the 4-hop experiment, which requires two rounds of segment validation, the additional latency introduced by SAVA-X reaches 31.88 μs.

[4]https://www.caida.org/catalog/datasets/passive_trace_statistics/

3) Throughput With Fixed Packet Size: This experiment aims to evaluate the throughput variation of SAVA-X for different packet sizes (ranging from 78 bytes to 2048 bytes) and different hop counts (1-8 hops). As shown in Figure 9, the experimental results show that the throughput rate of SAVA-X increases rapidly as the packet size grows. It starts from around 45 Gbps for the smallest packet size of 78 bytes and reaches the maximum throughput of 98 Gbps at a packet size close to 1500 bytes. Beyond this point, the throughput remains constant at 98 Gbps without further changes with the increase in packet size. For packet lengths below 256 bytes, the throughput gradually decreases as the number of hops in the test topology increases. For example, the throughput decreases from 79 Gbps (1 hop) to 62 Gbps (8 hops) at a packet size of 128 bytes. However, when the packet size exceeds 256 bytes, the throughput no longer varies with the number of hops. For instance, at packet sizes of 256 bytes, 512 bytes, 1024 bytes, and 1500 bytes, the throughput remains stable at 91-92 Gbps, 95 Gbps, 97 Gbps, and 98 Gbps, respectively, regardless of the number of hops. These findings suggest that when the packet size is less than 256 bytes, the high number of packets and the per-packet processing of SAVA-X impose significant stress on the router. As a result, each additional hop has a notable impact on the end-to-end throughput. However, when the packet size exceeds 256 bytes, the per-packet processing becomes less of a bottleneck and the increase in hop count at this packet size does not significantly affect the end-to-end throughput. The primary factor is the additional 18 bytes added to each packet, which includes

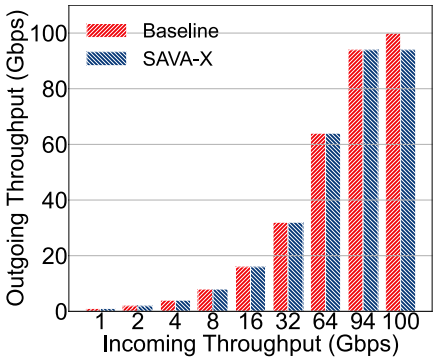

Fig. 10. Throughput with mixed packet size.

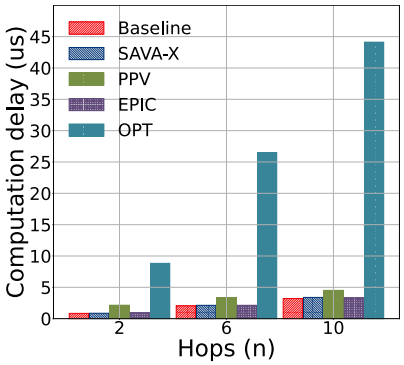

Fig. 11. The computation delay with packet size of 512 Bytes on P4.

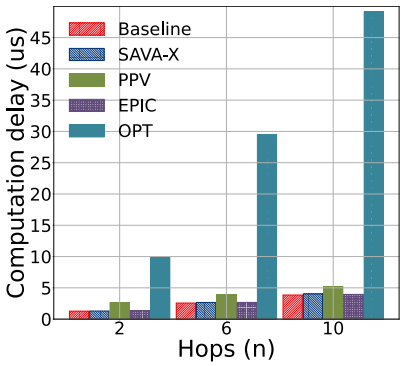

Fig. 12. The computation delay with packet size of 1024 Bytes on P4.

the standard destination options header (4B), tag length (1B), reserved field (1B), Tag (8B), and source ADID (4B).

4) Throughput With Mixed Packet Size: This experiment uses the dataset provided by the ISP to test the maximum throughput that SAVA-X can achieve under real network traffic. Corresponding to the latency experiment, we tested the throughput of SAVA-X under 1-hop as well as 4-hop topology conditions, and the experimental results are shown in Figure 10. The experimental results show that SAVA-X achieves the same throughput under 1-hop and 4-hop conditions, and can continuously increase the output traffic bandwidth as the incoming traffic bandwidth increases until SAVA-X reaches the maximum throughput when the incoming traffic reaches 94 Gbps. There is no difference in throughput performance of SAVA-X between 1-hop and 4-hop, so we no longer differentiate them in Figure 10. As a comparison, the baseline can achieve 100 Gbps throughput. Combined with the experimental results for fixed packet length, it can be found that SAVA-X can reach a maximum throughput of 98 Gbps regardless of the number of hops and packet size variations, which means that about 2% of bandwidth is lost due to the packet operations introduced by SAVA-X. In addition, the additional bandwidth is consumed due to the increase of 18 bytes in packets. The average size of mixed packets is 424 bytes, and the increase of 18 bytes also causes about 4% bandwidth loss, which, together with the about 2% bandwidth lost by SAVA-X packet operation, is the origin of the 6% bandwidth loss in this experiment.

B. Evaluation on P4 Switch

We compared the performance of SAVA-X and the state-of-the-art source validation mechanism on the data plane using a programmable switch (Barefoot Tofino programmable switch S9180-32X). Considering that the forwarding path validation mechanism can be used to a large extent to guarantee the authenticity of source addresses, we selected the following schemes for comparison, including OPT [35], a representative of early schemes, PPV [36] for probabilistic validation, and EPIC [37] for hierarchical security, with native IPv6 as baseline. We implement different hash algorithms on P4 to meet the computational requirements of each mechanism for packet tagging, including AES-128 for OPT, 2EM for EPIC and PPV, and CRC32 for SAVA-X (in line with the commercial router implementation). Considering the lack of comparison basis due to differences in design goals and implementation of each mechanism in the control plane, the additional overhead caused by key updates and other control plane management is not considered in this experiment. Only the data plane performance is compared. In experiments, SAVA-X, OPT, and EPIC verify packets at each hop, and PPV is based on the probability of 1/hops. We set the packet size to 512 and 1024 bytes and the forwarding path to 2, 6, and 10 hops to measure the computation delay during end-to-end forwarding. The experimental results are shown in Figure 11 and Figure 12.

The experimental results indicate that SAVA-X and EPIC exhibit relatively lower data-plane forwarding overhead, with minimal additional computational delay compared to the baseline, about 1 μs. In this experimental setting, OPT demonstrates a relatively larger computational delay due to the S9180-32X programmable switch necessitating 4 recirculations to complete an AES-128 operation (10 rounds). In contrast, OPT requires two AES operations per route. Specifically, the 2EMs closely resembled native IP forwarding on the experimental device. This results in a larger computational overhead for OPT compared to other schemes, and the gap increases with path length. It should be noted that if experiments are conducted with devices that have better support for the AES algorithm, the computational overhead of OPT will likely be significantly lower. PPV selects only two routers at each hop for tagging, necessitating two 2EM operations for each router. Regrettably, similar to OPT, the 12 pipelines on the device in this experiment cannot simultaneously complete two 2EM operations and require an additional recirculation.

These additional recirculation processes slightly increase the delay of PPV in comparison to EPIC and SAVA-X.

VI. Analysis

In this section, we analyze the performance and security properties in turn, to demonstrate that SAVA-X is a practical SAV approach. The practicality here is reflected in maintaining clear security benefits with lightweight overhead. The most important characteristic of SAVA-X is that it only deploys on the *AD*, the legacy end host could enjoy the benefit without upgradation. In the following, we first analyze the qualitative and quantitative overhead incurred by SAVA-X, to demonstrate the feasibility of SAVA-X to be deployed in commercial hardware. Then we analyze the security properties of SAVA-X, to explain why SAVA-X could defend against attacks, such as source spoofing and tag replay attacks, and further explain that SAVA-X does not incur new attacks.

A. Performance Analysis

The evolution of the Internet has proven that SAV deployment is a long process. From this perspective, backward compatibility, lightweight overhead, and scalability are crucial features for the deployability of an SAV approach. We first explain that SAVA-X is an SAV that keeps backward compatibility. Besides that, the key maintenance and per-packet processing in SAVA-X is also lightweight. Furthermore, the scalability could also be guaranteed by the hierarchical *AD*s, to facilitate the evolution from partial to large-scale deployments.

1) Backward Compatibility: SAVA-X is designed as a segmented end-to-end cryptography validation scheme that operates independently of the forwarding path. The *virtual AD* ensures that SAVA-X achieves the validation with the tag. Therefore, SAVA-X does not require routing information and does not impact existing routing systems. One of the advantages of SAVA-X is its end-to-end validation feature, which allows packets to traverse networks where SAVA-X is not yet deployed without compromising the validation process. This means that even if SAVA-X is only deployed in certain parts of the network, early adopters can still benefit from the security provided by the validation scheme without requiring a high deployment ratio across the entire network.

2) Key Maintenance Overhead: The key is negotiated by the ACS in each *AD*, and then each ADBR receives and locally stores the keys.

We first analyze the number of *AD*s in SAVA-X. SAVA-X is a hierarchical architecture, it splits the Internet into nested *AD*s. As the logical topology of hierarchical structure is a tree, suppose that there is N *AD* left after splitting the whole Internet via criteria such as routing, culture, business agreements, and so forth in the SAVA-X ecosystem. At the ideal situation, excluding the lowest level, the *AD*s obey uniform distribution, hence the logical structure of SAVA-X is a complete M-ary tree whose height is L and each node has M children nodes. The second lowest level *AD* would equally divide the N leaves *AD* into $\lceil \frac{N}{M^{L-1}} \rceil$. In this way, the total number of *AD*s could be listed as Equation (1).

$$NUM_{AD} = 1 + M + M^2 + \ldots + M^{L-2} + N \quad (1)$$

Then we analyze the storage overhead in terms of *AD* information. According to the CIDR report,[5] there are 32527 ASes running IPv6 with 201023 prefixes. $L = 4$ and $M = 24$ could meet the requirement to contain all the ASes in SAVA-X. For *AD* relationship information, each parent ADID takes $5B$, each ADID takes $5B$, the located level of each *AD* takes $1B$, each local *AD* takes $1B$, and the public key takes $128B$, then the total storage overhead is $(5 + 5 + 1 + 1 + 128) \times 32527 = 4.554MB$. *AD*'s prefix information includes the IPv6 prefix and its length ($17B$), the ADID ($5B$), the effective time ($8B$), and the sequence of the hierarchical level ($4 \times 5 = 20B$), so it takes $10.051MB$ for 201023 prefixes. Therefore, the storage overhead of *AD* information is sustainable.

Next, we introduce the storage overhead of keys in ADBR. The implementation of SAVA-X uses a clock-synchronized state machine. Considering the existence of small-time differences and the bi-directions between the same source and destination *AD* pairs, we should take 4 keys in effect simultaneously. Then each ADBR at least store $4 \times (M-1)$ keys for $M-1$ *brother AD*s at the same level, and 4×1 keys for its *parent AD*, $4 \times (M-1)$ keys for $M-1$ *child AD*s, thus has $4 \times (2M-1)$ keys. In the extreme case, one ADBR is settled at a whole different $L+1$ level boundary *AD*s, the key storage overhead at most could be represented by Equation (2).

$$NUM_{key} = 4 \times ((L-1) \times (M-1)) + 4 \times \left(\left\lceil \frac{N}{M^{L-1}} \right\rceil \times \left(\left\lceil \frac{N}{M^{L-1}} \right\rceil - 1 \right) \right) \quad (2)$$

To negotiate the keys with each other, each ADBR at most communicates with 4 ACSs with SSL/TLS, each communication takes $10KB$, and each ACS at most communicates with $1 + 23 + 24 = 48$ ACSes. Therefore, the storage overhead for the secure communication of each ACS is $0.48MB$, while each ADBR takes $0.04MB$, which is also sustainable.

3) Per-Packet Processing Overhead: As per-packet processing overhead in ADBR is the most significant indicator of an SAV scheme, we analyze the lightweight processing in ADBR. Based on the key stored in local ADBR, the source ADBR and the destination ADBR only process one *PktMAC* for each packet, while in the transition ADBR, it will validate the tag with one *PktMAC* and update the tag with another *PktMAC*. As each *PktMAC* takes one CRC32 operation (or another lightweight cryptographic operation [38]) in SAVA-X, which satisfies the packet processing at line rate, the per-packet processing overhead is sustainable with today's commercial hardware.

4) Expansion Management: Another characteristic of SAVA-X is scalability. Hierarchical *AD* effectively isolates structural changes outside the *AD* from internal nodes, minimizing the impact of changes on the rest of the system. On the other hand, SAVA-X can be implemented with any existing distributed consensus algorithm for Inter-*AD* Consensus Infrastructure. It has no special requirements for the efficiency

[5] https://www.cidr-report.org/v6/as2.0/#General_Status

of this process based on the assumption that *AD* information does not change frequently. Therefore, the decentralized maintenance approach can further reduce the management complexity of the expansion.

B. Security Analysis

We first analyze the correctness of SAVA-X, and tag replay attack does not hurt SAVA-X with the per-packet *PktMAC*. Besides that, SAVA-X could also defend against attacks on *AD* information and key negotiation.

1) Correctness: The correctness of SAVA-X relies on the cryptography of *PktMAC*s. The key for packet MAC is derived from the key negotiation between *AD*s. The base for the negotiated key comes from the public key exchanged during consensus when the *AD* joins the SAVA-X ecosystem and the corresponding private key held by the *AD* itself. The security of distributed consensus algorithms and public key cryptosystems is beyond the scope of this paper. With the protection of the aforementioned process, the security of SAVA-X's packet tagging key can be ensured. However, to meet the performance demands of inter-domain forwarding, SAVA-X opts for a relatively simple hash algorithm. To strike a balance between mechanism security and forwarding performance, SAVA-X employs independently renewable keys (based on a time-synchronized state machine). By significantly reducing the key update period, higher security can be achieved at a smaller cost of forwarding performance.

The ADBR generates the tag by hashing a five-tuple in the packet as depicted in Figure 5. An attacker might attempt to guess the tag and crack that key using brute force. Nevertheless, the tag is a 64-bit integer and the *PktMAC* is 32-bit long. Therefore, the feasible solution space is approximately 2^{32}. This makes it highly unlikely for attackers to achieve packet falsification. As the five-tuple includes the source address and payload, SAVA-X could effectively defend against various attacks, such as source spoofing or other off-path attacks [11].

Moreover, similar to other tag-based mechanisms, SAVA-X requires that the border routers of the participating parties can perform packet inspection and tag processing honestly. However, this requirement may not be satisfied in some special cases. When an *AD* is intentionally evil, it can only harm itself because it does not have the keys of other *AD*s. Even when it cooperates with other malicious *AD*s, it can only spoof each other and cannot affect the validation of normal traffic between honest nodes. At this point, however, malicious *AD*s can indeed spoof address prefixes in undeployed regions, which may facilitate the deployment of SAVA-X. Unfortunately, the threat is even more severe when an *AD* is intentionally malicious, and it will likely spoof the traffic of all its sub-*AD*s. However, as the upper layer *AD*, it is usually the network provider or management organization of the lower layer *AD*. The upper layer *AD* does not have the motivation to be evil from the perspective of business interests or management authority. What's more, unlike end hosts and servers, the connectivity of inter-domain routers and their special operating systems make it very difficult for attackers to control them. If the worst case happens and an attacker breaches all the border routers of upper layer *AD*, it does cause traffic spoofing to the sub-*AD*, which is the loss from the reduced management overhead of the hierarchical structure.

2) Guaranteed convex AD: A non-convex *AD* implies that the forwarding paths between internal *AD*s may have to cross other *AD*s, which will not guarantee that no tag replacement is required between *AD*s within the same *parent AD*. This might result in tag validation failures among *AD*s. In engineering practice, we ensure that *AD*s are convex by examining the forwarding paths among all *AD*s within the same *parent AD*. If the forwarding paths among all *AD*s in the same *parent AD* only exist within the *parent AD* (with no *AD*s outside the *parent AD* on the paths), we can confirm that the current *AD* is convex. If the check fails, the relevant *AD*s should reject the join request, and the joining *AD* should be divided into smaller *AD*s. Importantly, if BGP hijacking or other path manipulation causes the *AD* to no longer be convex, some of the *AD*s within the same *parent AD* may fail validation. It's worth noting that failure does not necessarily mean dropping it; it can be flagged as suspicious traffic. Additionally, this could potentially be a way to detect path manipulation attacks.

3) Replay attack: The tag is tied to a specific partial payload of the packet, giving each packet a unique tag, thus preventing attackers from reusing the observed tag. Meanwhile, as SAVA-X's keys have an expiration time, historical packets that surpass this time cannot be utilized for replay attacks, even if they are legitimate. But legitimate packets within the expiration time can indeed cause replay attacks. For off-path attackers, choosing a shorter key update period based on the round-trip time (RTT) between the source and destination is a viable option, but this method is not effective for on-path attackers. SAVA-X does not protect against traffic amplification attacks by on-path nodes with legitimate traffic. However, it is worth mentioning that experiments show that SAVA-X has little impact on router throughput, so attackers cannot exploit it to consume additional resources of links or routing nodes. Moreover, attackers cannot arbitrarily choose the attack object or collocate specific attack packets and must launch the above attack when there is continuous legitimate traffic and fresh legitimate packets are accessible.

4) Attacks towards AD information: In SAVA-X, the consistency guarantee of *AD* information relies on distributed consensus infrastructure among *AD*s. Therefore, the security challenges encountered by existing consensus algorithms will also be introduced into SAVA-X. However, the situation in SAVA-X differs from the typical distributed consensus scenario. Due to the hierarchical structure of SAVA-X, the breakdown of consensus on local *AD* information does not impact other non-adjacent *AD*s in the system. Simultaneously, the updated *AD* information only requires the signature confirmation from its parent, brother, and *child AD*s, indicating that an attack on the special node needs to hold specific resources, thereby heightening the complexity of the attack.

5) Attacks towards initial status negotiation: We have selected the clock-synchronized state machine to produce the key, which will operate locally with the initial status. To thwart SAVA-X, attackers could target the negotiation of the initial status. However, when the ACS-ACS pair or ACS-ADBR pair

is about to connect, SSL/TLS will be employed to ensure security in communication. SAVA-X can prevent attackers from obtaining the initial status, even if they can eavesdrop on the negotiation packet.

VII. Related Work

Inter-domain source address validation has been a long-standing problem in computer networking and related areas. Existing mechanisms for handling IP spoofing traffic are designed to create an expectation vector for each source, containing specific traffic parameters. These approaches can be categorized into routing-based, cryptography-based, and other schemes based on different traffic parameters.

A. Routing-Based Schemes

Routing-based source address validations rely on BGP to carry source address attributes on the forwarding path, including AS-level information (e.g., AS path, path length, etc.) to help ISPs validate their traffic. For example, Distributed Packet Filtering (DPF) [39] describes the filter method by constructing source address binding with the router's port and dropping packets whose source address is out of the binding relationship. As a consequent mechanism, the Inter-domain Distributed Packet Filter (IDPF) [21] uses BGP updates to infer feasible paths between a source and a destination. While this approach is only designed to mitigate the degree of IP spoofing, it can successfully reduce the number of IP addresses an attacker can spoof.

To prevent Inter-domain IP spoofing, IETF has recommended ISPs deploy ingress/egress filtering, also known as BCP 38 [15]. However, this approach has failed to provide deployment incentives that make the ratio of deployment stay at 80 percent now [26]. The Unicast Reverse Path Forwarding (uRPF) [40], also known as BCP 84, introduces a design to check the consistency between in and out ports as well as the reachability of prefixes. This protocol assumes symmetric inter-domain routing and utilizes this feature to detect spoofing traffic. To reduce its false positive detection rate, the enhanced feasible-path uRPF (EFP-uRPF) [16], [41] is then proposed to enable better accuracy and flexibility. However, due to the asymmetric and dynamic nature of inter-domain routing, the detection accuracy of the above uRPF protocols is still far from being satisfactory.

Therefore, ISPs should explore other high-accuracy options like BGP Anti-Spoofing Extension (BASE) [19] and Source Address Validity Enforcement (SAVE) [20]. The BASE protocol divides the deployed nodes into logical neighbors and achieves more accurate filtering on the forwarding path. On the contrary, SAVE constructs an independent interaction protocol to share routing decisions. It also enables the source router probe destination address space to learn valid previous hops for the source prefixes. As a trade-off, these protocols also require very high deployment rates to achieve the desired security gain. While the consequent iSAVE [42] improves the incremental deployment, the implementation of iSAVE is still too complex. The elevating deployment cost largely reduced their practicality in real-world inter-domain networks.

Selection Notice (SN) [43] adds a BGP option to announce the routers' selection of a route to the destination address prefix's owner AS i.e. all the routers of the packet forwarding path would know the packet sent from the source would pass which path. Though this is a good idea, the coupling of SN and BGP would make the dynamic routing affect the robustness of SN deeply.

B. Cryptography-Based Schemes

To address accuracy and deployment issues, cryptography-based source address validation has been widely recommended in recent years. This method aligns with the classic end-to-end design philosophy, where the end here refers to an AS. Specifically, a secret key will be shared between each ordered pair of ASes (e.g., AS s and AS d) before communication. Subsequently, a mark will be generated based on the secret key and applied to tag the traffic between s and d. When a tagged packet arrives at the border routers of AS d, the attached mark will be verified and used to detect possible spoofing traffic.

Based on this design, some existing protocols, such as IPSec [22], can be extended and adjusted to support inter-domain source address validation. For example, when IPSec AH [44] is used in tunnel mode, it will protect the encapsulated inner IP packet, including the entire inner IP header. Therefore, IPSec AH can be adopted to prevent Inter-domain IP spoofing. Unfortunately, this protocol requires the peer AS border routers to generate different secret keys, making it hard to implement across Internet ASes due to the maintenance of enormous keys. To reduce the maintenance cost, the Spoofing Prevention Method (SPM) [18] is proposed to construct a Trust Alliance System (TA) and allow ASes to join voluntarily. The ASes joined to the TA will be responsible for their egress traffic and checking the ingress traffic. However, the centralized management of trust alliances also brings severe deployment and collaboration challenges across distributed governance ASes.

Passport [23] introduces a new path validation mechanism to enable ASes to independently verify the source addresses. Passport achieves this by utilizing message authentication codes (MAC) computed with a secret key shared between the source AS and ASes on the path, which offers strong security and deployment incentives even in small-scale deployments. Inspired by the MAC generation of Passport, the Deployable Inter-domain Anti-spoofing (DIA) [24] offers an end-to-end source address validation method. In this setup, a MAC replaces the packet's IP identification field. The MAC is produced using symmetric keys and packet digests via the UMAC algorithm [45]. Once verified, the MAC is replaced with a random number and then sent to the destination host. Both Passport and DIA are challenged with growing key management overhead and struggled to handle large-scale networks.

The source and path verification approach can also be employed to safeguard the source address against spoofing. OPT [35] utilizes DRKey for the distribution of host keys and router keys, but the encryption of the entire path results in significant overhead. On the other hand, PPV [36] utilizes

probabilistic marking to reduce computation and communication costs to just two MACs per packet, regardless of the path length. EPIC [37] enhances verification efficiency by dividing the verification granularity into four levels, namely L0-L3.

Based on the Cryptographically Generated Address (CGA), Tan et al. have proposed a hierarchical technique SAV [46], [47]. However, CGA is incompatible with the current routing infrastructure, making it challenging to deploy.

C. Other Schemes

With the Time-to-live field of the IP header, Hop Count Filtering (HCF) [48] can roughly estimate whether the source address belongs to the correct region. It is ingenious but hardly accurate. Moreover, the Time-to-live field still could be spoofed as no router will check its correctness. Based on the model of SN route selection, IPVF [49] could shrink the size of filtering rules by referencing the hop limit of HCF. But IPVF is still not high-precision. CatchIt [50] combines the strength of SN and BASE. It announces route selection via TCP connection at the deployment area. However, CatchIt fails to provide guidelines for deployment in the long run.

Traceback is a series of post-disposal methods. It records the path information when packet forwarding and tries to backtrace the origin of the packet from the destination end. These methods including SPIE [51] which records the path information at the routers, iTrace-CP [52] which fully uses the ICMP to record the path information, and PPM [53] which directly uses the IP packet to record the path information. However, the most significant deficiency of these methods is that the algorithm of traceback is too heavy and complex.

VIII. Conclusion

IP spoofing poses a persistent and significant security challenge, rooted in fundamental design limitations of the internet architecture. Several source address validation mechanisms have emerged to address this vulnerability, yet their widespread adoption and efficacy in real-world internet conditions remain unsatisfactory. This paper introduces SAVA-X, a practical, inter-domain Source Address Validation solution designed for effective implementation.

Leveraging verifiable packet tags and a hierarchical address domain structure, SAVA-X delivers the following key advantages. Firstly, Linear Security Benefits. SAVA-X implements cryptographic source address validation capabilities from small-scale to large-scale deployments, proportionally enhancing security based on network size. This is achieved through the scalability provided by the hierarchical design, significantly reducing the maintenance overhead of traditional data plane solutions. Secondly, Adaptable Resolution. The flexibility of nested address domains enables diverse resolution configurations, tailoring security measures to specific network needs. The third one is Practical Scalability. Rigorous evaluations on commercial routers and the P4 platform demonstrate SAVA-X's high efficiency, incurring negligible latency increases and minimal throughput losses.

SAVA-X offers a compelling solution for mitigating IP spoofing, its hierarchical architecture and adaptable features align with the demands of diverse network environments.

References

[1] S. M. Bellovin, "Security problems in the TCP/IP protocol suite," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 19, no. 2, pp. 32–48, Apr. 1989.

[2] R. T. Morris, "A weakness in the 4.2 BSD unix TCP/IP software," Comput. Sci., AT T Bell Laboratories, Murray Hill, NJ, USA, Tech. Rep. 117, 1985.

[3] W. M. Eddy, "TCP SYN flooding attacks and common mitigations," IETF RFC Editor, Tech. Rep., pp. 1–25, 2007, vol. 4987, doi: 10.17487/RFC4987.

[4] C. Morrow, "BLS fastaccess internal tech needed," NANOG Arch., 2006. [Online]. Available: http://www.merit.edu/mail.archives/nanog/2006-01/msg00220.html

[5] B. Stock, G. Pellegrino, C. Rossow, M. Johns, and M. Backes, "Hey, you have a problem: On the feasibility of large-scale web vulnerability notification," in *Proc. 25th USENIX Secur. Symp.*, T. Holz and S. Savage, Eds. Austin, TX, USA, 2016, pp. 1015–1032.

[6] D. Anstee, D. Bussiere, G. Sockrider, and C. Morales, "Worldwide infrastructure security report," Arbor Netw., Westford, MA, USA, Tech. Rep., 2017. [Online]. Available: https://www.businesswire.com/news/home/20170124005370/en/

[7] M. Jonker, A. King, J. Krupp, C. Rossow, A. Sperotto, and A. Dainotti, "Millions of targets under attack: A macroscopic characterization of the DoS ecosystem," in *Proc. ACM IMC*, S. Uhlig and O. Maennel, Eds. London, U.K., 2017, pp. 100–113.

[8] E. Osterweil, A. Stavrou, and L. Zhang, "21 years of distributed denial-of-service: A call to action," *Computer*, vol. 53, no. 8, pp. 94–99, Aug. 2020.

[9] K. E. Defrawy and M. Gjoka, "BotTorrent: Misusing BitTorrent to launch DDoS attacks," in *Proc. 3rd Workshop Steps Reducing Unwanted Traffic Internet*, S. M. Bellovin, Ed. Santa Clara, CA, USA, 2007, pp. 1–6.

[10] X. Feng et al., "PMTUD is not panacea: Revisiting IP fragmentation attacks against TCP," in *Proc. NDSS*, 2022, pp. 1–18.

[11] X. Feng, C. Fu, Q. Li, K. Sun, and K. Xu, "Off-path TCP exploits of the mixed IPID assignment," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, J. Ligatti, X. Ou, J. Katz, and G. Vigna, Eds. 2020, pp. 1323–1335.

[12] X. Feng et al., "Off-Path network traffic manipulation via revitalized ICMP redirect attacks," in *Proc. 31st USENIX Secur. Symp.*, 2022, pp. 2619–2636.

[13] J. Wu, G. Ren, and X. Li, "Source address validation: Architecture and protocol design," in *Proc. IEEE Int. Conf. Netw. Protocols*, 2007, pp. 276–283.

[14] J. Wu, J. Bi, X. Li, G. Ren, K. Xu, and M. I. Williams, "A source address validation architecture (SAVA) testbed and deployment experience," IETF RFC Editor, Tech. Rep., RFC 5210, 2008, pp. 1–25, doi: 10.17487/RFC5210.

[15] P. Ferguson and D. Senie, "Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing," IETF RFC Editor, Tech. Rep., 2827, 2000, pp. 1–10, doi: 10.17487/RFC2827.

[16] F. Baker and P. Savola, "Ingress filtering for multihomed networks," Tech. Rep., 2004, pp. 1–16.

[17] J. Wu, J. Bi, M. Bagnulo, F. Baker, and C. Vogt, "Source address validation improvement (SAVI) framework," IETF RFC Editor, Tech. Rep., 7039, 2013, pp. 1–14, doi: 10.17487/RFC7039.

[18] A. Bremler-Barr and H. Levy, "Spoofing prevention method," in *Proc. INFOCOM 24th Annu. Joint Conf. IEEE Comput. Commun. Societies*, Miami, FL, USA, Mar. 2005, pp. 536–547.

[19] H. Lee, M. Kwon, G. Hasker, and A. Perrig, "BASE: An incrementally deployable mechanism for viable IP spoofing prevention," in *Proc. ACM Symp. Inf., Comput. Commun. Secur.*, F. Bao and S. Miller, Eds. Singapore, Mar. 2007, pp. 20–31.

[20] J. Li, J. Mirkovic, M. Wang, P. L. Reiher, and L. Zhang, "SAVE: Source address validity enforcement protocol," in *Proc. IEEE INFOCOM 21st Annu. Joint Conf. IEEE Comput. Commun. Societies*, New York, NY, USA, Jun. 2002, pp. 1557–1566.

[21] Z. Duan, X. Yuan, and J. Chandrashekar, "Controlling IP spoofing through interdomain packet filters," *IEEE Trans. Dependable Secure Comput.*, vol. 5, no. 1, pp. 22–36, Jan./Mar. 2008.

[22] S. T. Kent and K. Seo, "Security architecture for the internet protocol," IETF RFC Editor, Tech. Rep., RFC 4301, 2005, pp. 1–101, doi: 10.17487/RFC4301.

[23] X. Liu, A. Li, X. Yang, and D. Wetherall, "Passport: Secure and adoptable source authentication," in *Proc. 5th USENIX Symp. Networked Syst. Design Implement.*, J. Crowcroft and M. Dahlin, Eds. San Francisco, CA, USA, 2008, pp. 365–376.

[24] B. Liu, J. Bi, and Y. Zhu, "A deployable approach for inter-AS anti-spoofing," in *Proc. 19th Annu. IEEE Int. Conf. Netw. Protocols*, Vancouver, BC, Canada, 2011, pp. 19–24.

[25] R. Beverly, A. W. Berger, Y. Hyun, and K. Claffy, "Understanding the efficacy of deployed internet source address validation filtering," in *Proc. 9th ACM SIGCOMM Internet Meas. Conf.* A. Feldmann and L. Mathy, Eds. Chicago, IL, USA, 2009, pp. 356–369.

[26] M. J. Luckie, R. Beverly, R. Koga, K. Keys, and J. A. Kroll, "Network hygiene, incentives, and regulation: Deployment of source address validation in the internet," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, L. Cavallaro, J. Kinder, X. Wang, and J. Katz, Eds. London, U.K., 2019, pp. 465–480.

[27] R. Austein, G. Huston, S. T. Kent, and M. Lepinski, "Manifests for the resource public key infrastructure (RPKI)," IETF RFC Editor, Tech. Rep., pp. 1–19, 2012, vol. 6486, doi: 10.17487/RFC6486.

[28] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. T. Polk, "Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile," IETF RFC Editor, Tech. Rep., RFC 5280, 2008, pp. 1–151, doi: 10.17487/RFC5280.

[29] J. Gersch and D. Massey, "ROVER: Route origin verification using DNS," in *Proc. 22nd Int. Conf. Comput. Commun. Netw.*, Nassau, Bahamas, Aug. 2013, pp. 1–9.

[30] M. Nyström and B. Kaliski, "PKCS #10: Certification request syntax specification version 1.7," IETF RFC Editor, Tech. Rep., RFC 2986, 2000, pp. 1–14, doi: 10.17487/RFC2986.

[31] N. Haller, C. Metz, P. J. N. II, and M. Straw, "A one-time password system," IETF RFC Editor, Tech. Rep., pp. 1–25, 1998, vol. 2289, doi: 10.17487/RFC2289.

[32] W. M. Eddy, "Transmission control protocol (TCP)," IETF RFC Editor, Tech. Rep., RFC 9293, 2022, pp. 1–98, doi: 10.17487/RFC9293.

[33] E. Rescorla, "The transport layer security (TLS) protocol version 1.3," IETF RFC Editor, Tech. Rep., RFC 8446, 2018, pp. 1–160, doi: 10.17487/RFC8446.

[34] G. Almes, S. Kalidindi, M. J. Zekauskas, and A. Morton, "A one-way delay metric for IP performance metrics (IPPM)," IETF RFC Editor, Tech. Rep., RFC 7679, 2016, pp. 1–27, doi: 10.17487/RFC7679.

[35] T. H.-J. Kim, C. Basescu, L. Jia, S. B. Lee, Y.-C. Hu, and A. Perrig, "Lightweight source authentication and path validation," in *Proc. ACM Conf. SIGCOMM*, 2014, pp. 271–282.

[36] B. Wu et al., "Enabling efficient source and path verification via probabilistic packet marking," in *Proc. IEEE/ACM 26th Int. Symp. Quality Service (IWQoS)*, Jun. 2018, pp. 1–10.

[37] M. Legner, T. Klenze, M. Wyss, C. Sprenger, and A. Perrig, "EPIC: Every packet is checked in the data plane of a path-aware internet," in *Proc. 29th USENIX Conf. Secur. Symp.*, 2020, pp. 541–558.

[38] L. Wang, H. Kim, P. Mittal, and J. Rexford, "Programmable in-network obfuscation of traffic," 2020, *arXiv:2006.00097*.

[39] K. Park and H. Lee, "On the effectiveness of route-based packet filtering for distributed dos attack prevention in power-law internets," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 4, pp. 15–26, 2001.

[40] W. Kumari and D. McPherson, "Remote triggered black hole filtering with unicast reverse path forwarding (URPF)," IETF RFC Editor, Tech. Rep., RFC 5635, 2009, pp. 1–15, doi: 10.17487/RFC5635.

[41] K. Sriram, D. Montgomery, and J. Haas, "Enhanced feasible-path unicast reverse path forwarding," IETF RFC Editor, Tech. Rep., pp. 1–17, 2020, vol. 8704, doi: 10.17487/RFC8704.

[42] T. Ehrenkranz, J. Li, and P. D. McDaniel, "Realizing a source authentic internet," in *Security and Privacy in Communication Network* (Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering), vol. 50, S. Jajodia and J. Zhou, Eds., Singapore. Berlin, Germany: Springer, 2010, pp. 217–234.

[43] L. Wang, J. Wu, and K. Xu, "Bgp extension to support interdomain distributed packets filtering," *J. Softw.*, vol. 18, no. 12, pp. 3048–3059, 2007.

[44] S. T. Kent, "IP authentication header," IETF RFC Editor, Tech. Rep., RFC 4302, 2005, pp. 1–34, doi: 10.17487/RFC4302.

[45] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, and P. Rogaway, "UMAC: Fast and secure message authentication," in *Advances in Cryptology—CRYPTO* (Lecture Notes in Computer Science), vol. 1666, M. J. Wiener, Ed. Santa Barbara, CA, USA. Berlin, Germany: Springer, 1999, pp. 216–233.

[46] P. Tan, H. Jia, Y. Chen, and J. Mao, "A hierarchical source address validation technique based on cryptographically generated address," in *Proc. IEEE Int. Conf. Comput. Sci. Automat. Eng.*, vol. 2, Jun. 2011, pp. 33–37.

[47] T. Aura, "Cryptographically generated addresses (CGA)," IETF RFC Editor, Tech. Rep., RFC 3972, 2005, pp. 1–22, doi: 10.17487/RFC3972.

[48] H. Wang, C. Jin, and K. G. Shin, "Defense against spoofed IP traffic using hop-count filtering," *IEEE/ACM Trans. Netw.*, vol. 15, no. 1, pp. 40–53, Feb. 2007.

[49] Z. Zhang, Y. Liu, J. Wu, G. Ren, and J. Bi, "An inter-AS path vector filter: Towards elimination of false negatives," in *Proc. IEEE Int. Workshop Local Metropolitan Area Netw.*, Beijing, China, Apr. 2015, pp. 1–2.

[50] J. Li, J. Bi, and J. Wu, "Umbrella: A routing choice feedback based distributed inter-domain anti-spoofing solution," in *Proc. 20th IEEE Int. Conf. Netw. Protocols*, Austin, TX, USA, Oct. 2012, pp. 1–2.

[51] A. C. Snoeren, "Hash-based IP traceback," in *Proc. ACM SIGCOMM Conf. Appl., Technol., Architectures, Protocols Comput. Commun.*, R. L. Cruz and G. Varghese, Eds. San Diego, CA, USA, 2001, pp. 3–14.

[52] H. C. J. Lee, V. L. L. Thing, Y. Xu, and M. Ma, "ICMP traceback with cumulative path, an efficient solution for IP traceback," in *Proc. Inf. Commun. Secur., 5th Int. Conf.* (Lecture Notes in Computer Science), vol. 2836, S. Qing, D. Gollmann, and J. Zhou, Eds., Huhehaote, China. Berlin, Germany: Springer, 2003, pp. 124–135.

[53] S. Savage, D. Wetherall, A. R. Karlin, and T. E. Anderson, "Practical network support for IP traceback," in *Proc. ACM SIGCOMM Conf. Appl., Technol., Architectures, Protocols Comput. Commun.*, C. Partridge, Ed. Stockholm, Sweden, 2000, pp. 295–306.