

Workgroup: Network Working Group  
Internet-Draft: draft-xu-savax-data-00  
Published: 18 November 2021  
Intended Status: Informational  
Expires: 22 May 2022

Authors: K. Xu	X. Wang
Tsinghua University	Tsinghua University
Y. Guo	J. Wu
Tsinghua University	Tsinghua University

## **Data Plane of Inter-Domain Source Address Validation Architecture**

### **Abstract**

Because the Internet forwards packets according to the IP destination address, packet forwarding typically takes place without inspection of the source address and malicious attacks have been launched using spoofed source addresses. The inter-domain source address validation architecture is an effort to enhance the Internet by using state machine to generate consistent tags. When communicating between two end hosts at different ADs of IPv6 network, tags will be added to the packets to identify the authenticity of the IPv6 source address.

This memo focus on the data plane of the SAVA-X mechanism.

### **Status of This Memo**

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 22 May 2022.

### **Copyright Notice**

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

(<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

## Table of Contents

- [1. Introduction](#)
- [2. Terminology and Abbreviation](#)
- [3. State Machine Mechanism](#)
- [4. Tag](#)
  - [4.1. Tag Generation Algorithm](#)
    - [4.1.1. Pseudo-Random Number Algorithm](#)
    - [4.1.2. Hash Chain Algorithm](#)
  - [4.2. Tag Update](#)
- [5. Packet Processing at AER](#)
  - [5.1. Port Classification](#)
  - [5.2. Source Address Validation](#)
  - [5.3. Packet Classification](#)
  - [5.4. Tag Addition](#)
  - [5.5. Tag Verification](#)
  - [5.6. Tag Replacement](#)
- [6. Packet Signature](#)
- [7. Security Consideration](#)
- [8. IANA Considerations](#)
- [9. Acknowledgements](#)
- [10. Normative References](#)
- [Authors' Addresses](#)

## 1. Introduction

The Inter-Domain Source Address Validation (SAVA-X) mechanism establishes a trust alliance among Address Domains (AD), maintains a one-to-one state machine among ADs, generates a consistent tag, and deploys the tag to the ADs' border router (AER). The AER of the source AD adds a tag to identify the identity of the AD to the packet originating from one AD and sinking in another AD. The AER of the destination AD verifies the source address by validating the correctness of the tag to determine whether it is a packet with a forged source address.

In the process of packet forwarding, if the source address and the destination address of this packet both are addresses in the trust alliance, however the tag is not added or incorrectly added, AER of the destination AD determines that the source address is forged and directly discards this packet. The destination AD forwards the

packet directly for packets whose source address is an address outside the trust alliance.

This document mainly studies the relevant specifications of the data plane of the inter-domain source address validation architecture mechanism between ADs, which will protect IPv6 networks from being forged source address. You could see [[RFC8200](#)] for more details about IPv6. It stipulates the state machine, tag generation and update, tag processing in AER, and packet signature Its promotion and application can realize the standardization of the data plane in the SAVA-X to facilitate the related equipment developed by different manufacturers and organizations to cooperate to accomplish the inter-domain source address validation jointly.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119, BCP 14 [[RFC2119](#)] and indicate requirement levels for compliant CoAP implementations.

## 2. Terminology and Abbreviation

Abbreviation	Description
AD	Address Domain, the unit of a trust alliance, which is an address set consisting of all IPv6 addresses corresponding to an IPv6 address prefix.
TA	Trust Alliance, the IPv6 network that uses the SAVA-X mechanism.
ACS	AD Control Server, the server that maintains state machine with other ACS and distribute information to AER.
AER	AD border router, which is placed at the boundary of an AD or STA.
ADID	The identity of an AD.
ADID_Rec	The record of a number of an AD.
ARI_Rec	The record with relevant information of an AD or STA.
API_Rec	The record of prefix of an AD or STA.
SM	State Machine, which is maintained by a pair of ACS to generate tags.
Tag	The authentic identification of source address of a packet.

Table 1

## 3. State Machine Mechanism

In SAVA-X, state machine mechanism is used to generate, update, and manage the tags.

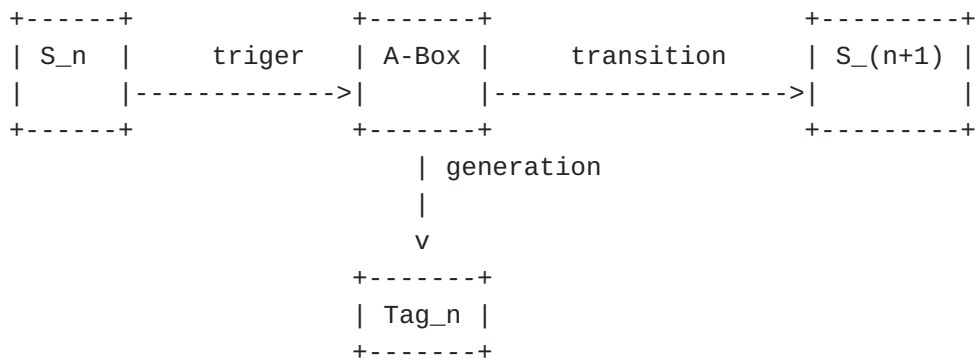


Figure 1: State machine mechanism.

**State:**  $S_n$  and  $S_{(n+1)}$  represent the current state and next state of the SM respectively.

**Tag:**  $Tag_n$  is generated in the progress of state transiting from  $S_n$  to  $S_{(n+1)}$ .

**Algorithm Box:** A-Box is Alogorithm Box. It is used to transite the State and generate the tag. It takes the current State as the input and the following State and current tag as the output. The algorithm box consists of two parts: one is the transition function  $transit()$ ,  $S_{(n+1)} = transit(S_n)$ ; the second is the function  $generate()$  to generate tags.  $Tag_n = generate(S_n)$ . Algorithm box (A-Box) is the core of state machine. It determines the data structure of state and tag, the specific mode of state machine implementation, as well as its security and complexity.

**Trigger:** It is used to trig the transition of State.

**Transition:** It reprints the progress of state transiting from  $S_n$  to  $S_{(n+1)}$ .

**Generation:** It reprints the progress of calculating the current tag from current State.

## 4. Tag

### 4.1. Tag Generation Algorithm

There are two ways to generate tags: pseudo-random number algorithm and hash chain algorithm.

#### 4.1.1. Pseudo-Random Number Algorithm

In the pseudo-random number generation algorithm, an initial number or string is usually used as the "seed", which corresponds to the initial state of the state machine. Using seeds, a pseudo-random number sequence is generated as a tag sequence through some

algorithm. Next, we would take KISS (keep it simple stub), a pseudo-random number generation algorithm, as an example to introduce how to apply it to the state machine mechanism. For the algorithm details of KISS, you could refer to the following reference pseudo code:

```
/* Seed variables */
uint x = 123456789, y = 362436000, z = 521288629, c = 7654321;
uint KISS(){
    const ulong a = 698769069UL;
    ulong t;
    x = 69069*x+12345;
    y ^= (y<<13); y ^= (y>>17); y ^= (y<<5);
    t = a*z+c; c = (t>>32);
    z=cast(uint)t;
    return x+y+z;
}
```

Figure 2: KISS99: Pseudo-random number generatation

In this algorithm, State  $S$  can be expressed as  $(x, y, z, c)$ . The algorithm box is `KISS()`. After each calculation, the state undergoes a transition from  $S_n$  to  $S_{(n+1)}$ , that is, the four variables  $x, y, z$  and  $c$  are all changed. At the same time, a pseudo-rng number  $(x + y + z)$  is generated.

As the state machine shown above, the initial state is  $S_0 = (123456789, 362436000, 521288629, 7654321)$ . In fact, the initial state can be arbitrarily selected by the algorithm shown below:

```
void init_KISS() {
    x = devrand();
    while (!(y = devrand())); /* y must not be zero */
    z = devrand();
    /* Don't really need to seed c as well
       but if you really want to... */
    c = devrand() % 698769069; /* Should be less than 698769069 */
}
```

Figure 3: KISS99: Initial state selection

The basic design goal of pseudo-random number generation algorithm is mainly long cycle and pretty distribution, however, without or little consideration of safety factors. The backstepping security and prediction ability of KISS algorithm have not been proved.

#### 4.1.2. Hash Chain Algorithm

For the design of hash chain based tag generating algorithm, one can see S/Key in [RFC1760]. In the S/Key system, there is an encryption end and an authentication end. The encryption end generates an initial state  $W$ , and then uses some hash algorithm  $H()$  to iterate on  $W$  to obtain a string sequence:  $H_0(W)$ ,  $H_1(W)$ , ...,  $H_N(W)$ , where  $H_n(W)$  represents the iterative operation of  $H()$  on  $W$   $n$  times,  $H_0(W) = W$ . The state sequence  $\{S\}$  is defined as the reverse order of the hash chain, that is,  $S_n = H_{(N-n)}(W)$ . For example, the initial state  $S_0 = H_N(W)$  and the final state  $S_N = H_0(W) = W$ , so the transfer function  $\text{transit}()$  is represented as the inverse  $H()$ . Different from the KISS pseudo-random number generation algorithm mentioned in the previous section, in the hash chain, the tag is the state itself, that is, the output and input of  $\text{generate}()$  are consistent, and  $\text{Tag}_n = S_n$ . In the following discussion,  $S_n$  is temporarily used instead of  $\text{Tag}_n$  for the convenience of expression.

The encryption end sends the initial state  $S_0$  to the verification end, and maintains  $S_1 \sim S_n$ , which is also the tag sequence used. The encryption end sends  $S_{(n+1)}$  to the verification end every time. The verification end uses the  $S_n$  maintained by itself to verify the tag correctness of the encryption end by calculating  $S_{(n+1)} = \text{transit}(S_n)$ . As explained above,  $\text{transit}()$  is the inversion of  $H()$ . In practice, a secure hash algorithm is usually used as  $H()$ , such as SHA-256. For these hash algorithms, it is easy to calculate  $H()$ , but it is difficult to calculate the inversion of  $H()$ . Therefore, the actual operation is as follows: after receiving  $S_{(n+1)}$ , the verification end calculates whether  $H(S_{(n+1)})$  is equal to  $S_n$ . If it is equal, the verification is successful, otherwise it fails.

Hash chain algorithm has high security. It can prevent backstepping and prediction well. Not only the attacker can't backstep or predict, but also the verification end cannot do that. The disadvantage of hash chain algorithm is that before using tags, the encryption end needs to calculate all tag sequences, and then send the last of the sequence to the verification end as the initial state. At the same time, the encryption end needs to save a complete tag sequence, although it can be deleted after each tag is used up. The cost of storage of hash chain algorithm can not be ignored

#### 4.2. Tag Update

After the state machine is enabled, the source AD uses the initial state  $S_0$  to transfer to the state  $S_1$  through the algorithm box, and generates the tag  $\text{Tag}_1$ . In the subsequent state transition interval, the AER of the source AD uses the same tag,  $\text{Tag}_1$ , to add to the message sent from this AD to the destination AD. The source AD does not transfer from the state  $S_1$  to the state  $S_2$  until the

transition interval passes, and starts to use tag Tag\_2. In this cycle, the state sequence S\_1 ~ S\_N and tag sequence Tag\_1 ~ TAG\_N are experienced, in which Tag\_1 ~ Tag\_N are used as tags in turn and added to the message by the source AER. Similarly, the destination AER uses the same state machine to calculate the tag sequence, so as to verify the tag in the message. If the source AD and the destination AD can ensure the synchronization of the state machine, it would guarantee the synchronization of the tags. So the tags can be verified correctly.

Each state machine has an activation time and an Expiration Time. After the expiration time comes, the current state machine is deactivated. If a new state machine is available, the new state machine will be used and performs the same label verification process.

## **5. Packet Processing at AER**

SAVA-X does not require the intermediate router to recognize and process the SAVA-X option, which we will described at [Section 8](#), as long as the intermediate router correctly implements the extension header and option processing method described in IPv6 protocol [[RFC8200](#)]. The intermediate router could correctly forward the packet regardless of its specific content even if it does not recognize the SAVA-X option well.

The border router, AER, needs to handle tag correctly. The AER of the source AD judges whether the IPv6 destination address belongs to the trust alliance. If no, the packet will be forwarded directly. If yes, the AER continues to judge the hierarchical relationship between the the source AD and the member ADs to which the packet's destination IP address belongs. If the source AD and the destination AD are under the same sub-trust alliance, the AER would add the tag between the two ADs, otherwise add the AD\_V tag.

Note that the packet will not be processed at other AERs in the sub-trust alliance.

At the AER of the boundary of sub-trust alliance, the packet is classified according to the IPv6 destination address. If the destination address is not within the trust alliance, it will be forwarded directly. If the destination address belongs to this sub-trust alliance, it will be classified according to the source IP address. If the source address also belongs to this sub-trust alliance, it will be forwarded directly. If the source address does not belong to this sub-trust alliance, the AER needs to verify the sub-trust alliance tag and replace it with the AD\_V tag in this sub-trust alliance for following forwarding. If the destination IP address of packet belongs to other sub-trust alliance, it SHALL be

classified according to the source address. If the source address belongs to this sub-trust alliance, verify the AD\_V tag. If consistent, replace with sub-trust alliance tag. If the source address is not in this sub-trust alliance, it will be forwarded directly. Otherwise, the packet will be discarded.

The AER of the destination AD classifies packet according to the source address of the packet to be forwarded to determine whether it originates from a member AD. If yes, enter the label check. Otherwise it will be forwarded directly. Tag verification process: if the tag carried by the packet is consistent with the tag used by the source AD, remove the tag and forward the packet. Otherwise the packet will be discarded.

### **5.1. Port Classification**

In order to classify packets correctly to complete tag addition, inspection and packet forwarding, it is necessary to classify the ports (interfaces) of AER. Any connected port of AER must belong to and only belong to the following types of ports:

- \*Ingress Port: Connect to the port of non-SAVA-X router in this AD. Generally connected to IGP router in the domain.

- \*Egress Port: Connect to other AD ports.

- \*Trust Port: Connect to the port of SAVA-X router in this AD.

### **5.2. Source Address Validation**

In SAVA-X, AER must check the source address of the packet. Only the packet passing the check will be subject to the [Section 5.3](#) step, and the packet using the fake source IP address will be discarded. The source address is checked using the ingress filtering method. AER only checks the source address according to the following three rules:

- \*The packet entering an AER from the Ingress Port SHALL only carry the source address prefix belonging to this AD.

- \*The packet entering an AER from the Egress Port SHALL NOT carry the source address prefix belonging to this AD.

- \*Packets entering an AER from Trust Port are not checked.

The prefix of IP address owned by one AD SHALL be configured by the administrator or obtained from the control plane, and deployed to AER by ACS of this AD.



### 5.3. Packet Classification

It SHALL be classified after the packet entering an AER passes the source address validation. There are three types of packets: packets that SHOULD be tagged, packets that SHOULD check tags, and other messages. The judgment rules of the three packets are as follows:

\*Packets entering AER from Ingress Port. If the source address belongs to this AD and the IPv6 destination address belongs to another AD in the same sub-trust alliance, tag must be added. If the source IP address belongs to another AD in the same sub-trust alliance and the IPv6 destination address belongs to another sub-trust alliances, the tag must be verified and replaced with the sub-trust alliance tag. Other packets are forwarded directly.

\*Packets entering AER from the Egress Port. Tag must be checked if the source address belongs to another AD in the same sub-trust alliance and the IPv6 destination address belongs to this AD. If the source address belongs to other sub-trust alliance and the IPv6 destination address belongs to another AD in the same sub-trust alliance, the tag must be checked and replaced. And other packets can be forwarded directly.

\*Packets entering AER from Trust Port. These packets SHOULD be forwarded directly.

The relationship between IP address and ADs SHALL be obtained from the control plane and deployed to the AER by the ACS of the AD. When the SAVA-X option of the packet received from the progress port carries the active AD number, you can skip the "mapping from address to AD number" process and directly use the AD number carried in the message.

### 5.4. Tag Addition

AER SHOULD add destination option header and add SAVA-X option into the packet according to the requirements of IETF [[RFC8200](#)].

According to [[RFC8200](#)], the destination option header SHOULD be filled so that its length is an integer multiple of 8 bytes, including the Next Hader and Hdr Ext Len fields of the destination option header, the Next Header and Payload Length fields of the IPv6 packet header, and the upper protocol header (such as TCP, UDP, etc.). If it is necessary, AER SHOULD recalculate the Checksum field.

## 5.5. Tag Verification

AER will process the first option with Option Type equals to the binary code of 00111011 in the destination header. We would talk more about that at [Section 8](#).

1. If the packet does not contain destination option header or SAVA-X option. the packet SHOULD be discarded.
2. If the packet contains SAVA-X option but the parameters or tag are incorrect, the packet SHOULD be discarded.
3. If the packet contains SAVA-X option, and the parameters and tag are correct, AER must replace the tag or remove the tag when needed before forwarding the message.

In the following scenarios, the tag needs to be removed. If there are only SAVA-X option, Pad1 and PadN options in the destination option header of the message, AER SHOULD remove the whole destination option header. If there are other options besides SAVA-X option, Pad1 and PadN option in the destination option header, AER SHOULD remove SAVA-X option and adjust the alignment of other options according to the relevant protocols of IPv6. In order to removing the sava-x option, the destination option header may also be filled, or some Pad1 and PadN may be removed, to make its length be multiple of 8 bytes. At the same time, the Next Header field and Payload Length field deployed in the IPv6 message header, and the Checksum field of the upper protocol header (such as TCP, UDP, etc.) SHALL be rewritten as necessary.

## 5.6. Tag Replacement

Tag needs to be replaced when packet pass through different sub-trust alliance. Tag replacement needs to be done on the AER of the boundary address domain of the sub-trust alliance. This feature is not necessary to realize on the AER of each non-boundary address domain in the sub-trust alliance.

When packet is arrievied at the AER of the sub-trust alliance boundary, it is classified according to the destination address.

1. If the destination address does not belong to the trust alliance, it will be forwarded directly.
2. If the destination address belongs to this sub-trust alliance, it will be classified according to the source address of the packet.

\*If the source address also belongs to this sub-trust alliance, the packet will be forwarded directly.

\*If the source address does not belong to this sub-trust alliance, AER should verify the sub-trust alliance tag and replace it with the AD\_V tag in this sub-trust alliance for forwarding.

3. If the destination address of the packet belongs to other sub-trust alliance, it shall be classified according to the source address.

\*If the source address belongs to this sub-trust alliance, AER should verify the AD\_V tag and replace the tag with sub-trust alliance tag when it is consistent.

\*If the source address is not in this sub-trust alliance, it will be forwarded directly.

4. Otherwise, the packet will be discarded.

Alliance tag will be used when the packet crosses the upper AD which is at the higher level of source AD and destination AD. Alliance tag is the tag maintained between the source AD corresponding to the AD in the parent AD and the destination AD corresponding to the address domain in the parent AD.

## **6. Packet Signature**

It is difficult to accurately synchronize time among the trust alliance members. So we propose a shared time slice, which means that there are two tags effecting at the same time in a period of time. But it may suffer from replay attack. Therefore, a packet signature mechanism is proposed to prevent replay attack and cancel the original tag.

Tag is time-dependent. The state machine triggers state transition by time and generates a new tag. In a short period of time, all data packets are labeled with the same tag. Moreover, due to the subtle differences in time synchronization, both old and new tags can be used for this short period of time, so attackers can reuse tags for replay attack by simply copying tags.

The packet signature mechanism joins 8-bit part of the payload in the packet and the tags generated by the state machine. And then it calculates hash value with parameters above to achieve the effect of packet by packet signature and resist the attackers reuse of tags. Its processing flow is shown below.



## 8. IANA Considerations

[illegible]

**Option Type:** 8-bit field. The destination option type of SAVA-X = 59.

**Opt Data Len:** 8-bit field. The bytes length of SAVA-X option. Its value is  $2 + \text{LenOfAI} + (\text{TagLen} + 1)$ , where LenOfAI is 2 when AI Type is 1, or 4 when AI Type is 2, or 0 default.

**Tag Len:** 4-bit field. The bytes length of TAG equals to  $(\text{Tag Len} + 1) * 8$ , e.g. if Tag Len = 7, it means SAVA-X uses 64 bits long TAG. It guarantees the length of TAG would be an integral multiple of 8 bits. The maximum length of TAG is 128 bits and the minimum length of TAG is 32 bits.

**AI Type:** 4-bit field. The type of Additional Information. 0 for no Additional Information, 1 for 16-bit long Additional Information and 2 for 32-bit long Additional Information. Others are not assigned.

**Reserverd:** These bits are not used now and must be zero.

**TAG:** Variable-length field The actual tag, its length is determined by Tag Len field.

**Additional Information:** As defined in AI Type field.

## 9. Acknowledgements

Much of the content of this document is the expansion of the IETF [RFC5210] in inter-domain level. Thanks to the effort of pioneers.

## 10. Normative References

- [RFC1760] Haller, N., "The S/KEY One-Time Password System", RFC 1760, DOI 10.17487/RFC1760, February 1995, <<https://www.rfc-editor.org/info/rfc1760>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC5210] Wu, J., Bi, J., Li, X., Ren, G., Xu, K., and M. Williams, "A Source Address Validation Architecture (SAVA) Testbed and Deployment Experience", RFC 5210, DOI 10.17487/RFC5210, June 2008, <<https://www.rfc-editor.org/info/rfc5210>>.
- [RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <<https://www.rfc-editor.org/info/rfc8200>>.

## Authors' Addresses

Ke Xu  
Computer Science, Tsinghua University  
Qinghuayuan street, Haidian District  
Beijing  
100084  
China

Email: [xuke@tsinghua.edu.cn](mailto:xuke@tsinghua.edu.cn)

Xiaoliang Wang  
Computer Science, Tsinghua University  
Qinghuayuan street, Haidian District  
Beijing  
100084  
China

Email: [wangxiaoliang0623@foxmail.com](mailto:wangxiaoliang0623@foxmail.com)

Yangfei Guo  
Institute for Network Sciences and Cyberspace, Tsinghua University  
Qinghuayuan street, Haidian District  
Beijing  
100084  
China

Email: [guoyangf19@mails.tsinghua.edu.cn](mailto:guoyangf19@mails.tsinghua.edu.cn)

Jianping Wu  
Computer Science, Tsinghua University  
Qinghuayuan street, Haidian District  
Beijing  
100084  
China

Email: [jianping@cernet.edu.cn](mailto:jianping@cernet.edu.cn)