Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 24 May 2026

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

20 November 2025

Data Plane of Source Address Validation Architecture-eXternal (SAVA-X) draft-xu-savax-data-09

Abstract

Due to the fact that the Internet forwards packets in accordance with the IP destination address, packet forwarding generally occurs without examination of the source address. As a result, malicious attacks have been initiated by utilizing spoofed source addresses. The inter-domain source address validation architecture represents an endeavor to enhance the Internet by employing state machines to generate consistent tags. When two end hosts at different address domains (ADs) of the IPv6 network communicate with each other, tags will be appended to the packets to identify the authenticity of the IPv6 source address.

This memo focuses 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 24 May 2026.

Copyright Notice

Table of Contents

1. Introduction

The Inter-Domain Source Address Validation-eXternal (SAVA-X) mechanism serves to establish a trust alliance among Address Domains (AD). It maintains a one-to-one state machine among ADs in conjunction with the AD Control Server (ACS). Moreover, it generates a consistent tag and deploys this tag to the ADs' border router (AER). The AER of the source AD appends a tag to packets originating from one AD and destined for another AD, thereby identifying the identity of the AD. The AER of the destination AD verifies the source address by validating the correctness of the tag to determine whether the packet has a forged source address.

In the packet forwarding process, if both the source address and the destination address of a packet belong to the trust alliance, the tag is either not added or added incorrectly. In such a case, the AER of the destination AD determines that the source address is forged and directly discards this packet. For packets with a source address outside the trust alliance, the destination AD forwards the packet directly.

This document mainly studies the relevant specifications of the data plane of the SAVA-X between ADs, which will protect IPv6 networks from being forged source addresses. 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.

2. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2.1. Terminology and Abbreviation

The following terms are used with a specific meaning:

ACS:
: AD Control Server. The server maintains the state machine with other ACS and distributes information to AER.

AD:
: Address Domain. The unit of a trust alliance. It is an address set consisting of all IPv6 addresses corresponding to an IPv6 address prefix.

ADID:
: The identity of an AD.

ADID_Rec:
: The record of the number of an AD.

AER:
: AD border router, which is placed at the boundary of an AD of STA.

API_Rec:
The record of the prefix of an AD or STA.

ARI_Rec:
The record with relevant information of an AD or STA.

SM:
State Machine, which is maintained by a pair of ACS to generate tags.

TA:
Trust Alliance. The IPv6 network that uses the SAVA-X mechanism.

Tag:
The authentic identification of the source address of a packet.

3. State Machine Mechanism

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

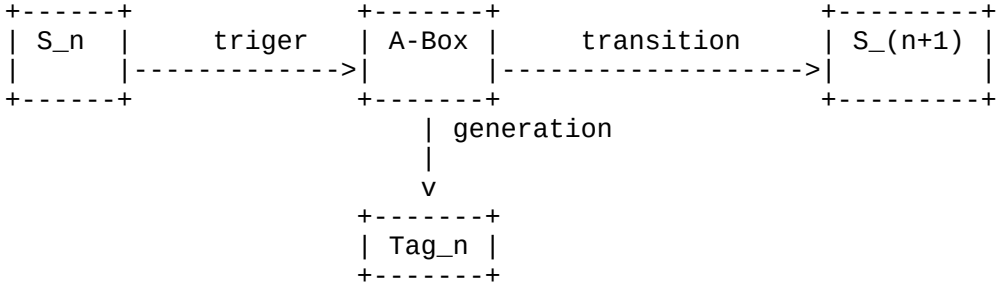

Figure 1: State machine mechanism.

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 transmit 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). The algorithm box (A-Box) is the core of the 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 trigger the transition of State.

Transition:
It reprents the progress of state transiting from S_n to S_(n+1).

Generation:
It represents the progress of calculating the current tag from the 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 unsigned long a = 698769069UL;
   unsigned long 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 the pseudo-random number generation algorithm is mainly a long cycle and pretty distribution, however, without or little consideration of safety factors. The backstepping security and prediction ability of the KISS algorithm have not been proved.

4.1.2. Hash Chain Algorithm

For the design of a 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 transit() is repsented as the invere 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 generate() are consistent, and Tag_n = S_n. In the following discussion, S_n is temporarily used instead of Tag_n for the convenience of expression.

The encryption end sends the initial state S_0 to the verification end, and maintains S_1 ~ 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) = transit(S_n). As explained above, 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 cannot backstep or predict, but also the verification end cannot do that. The disadvantage of the 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 the 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 Tag_1. In the subsequent state transition interval, the AER of the source AD uses the same tag, 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 perform 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 describe at Section 9, 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 the 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 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 the 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 the packet belongs to another 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 a 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 the 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 the 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 the 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, the 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 alliance, 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. The 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 another 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 a destination option header and add the 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 equal to the binary code of 00111011 in the destination header. It is detailed described at Section 9.

1. If the packet does not contain a destination option header or SAVA-X option. the packet SHOULD be discarded.

2. If the packet contains the SAVA-X option but the parameters or tag are incorrect, the packet SHOULD be discarded.

3. If the packet contains the 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 the 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 the SAVA-X option, Pad1 and PadN option in the destination option header, AER SHOULD remove the SAVA-X option and adjust the alignment of other options according to the relevant protocols of IPv6. In order to remove the SAVA-X option, the destination option header may also be filled, or some Pad1 and PadN may be removed, to make its length 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

The tag needs to be replaced when the packet passes through different sub-trust alliances. 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 the AER of each non-boundary address domain in the sub-trust alliance.

When the packet arrives 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 another 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 the 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 affecting at the same time in a period of time. But it may suffer from a replay attack. Therefore, a packet signature mechanism is proposed to prevent replay attacks and conceal the original tag.

The 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 attacks by simply copying tags.

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

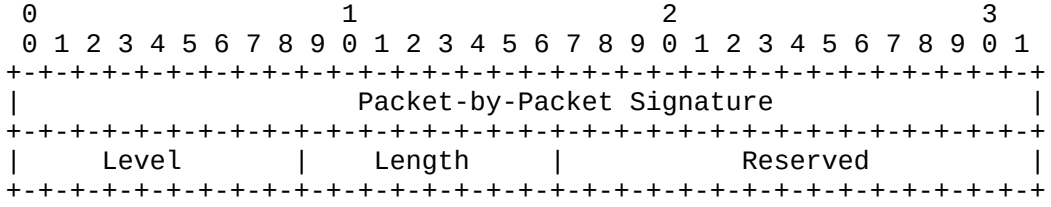

Figure 4: Format of the Packet-by-Packet signature

Packet-by-Packet Signature:
The hash value of the original tag, source address, destination address, first 8-bit of payload, credible level, and credible prefix length.

Level:
8-bit of credible level.

Length:
8-bit of credible prefix length.

Reserved:
16-bit of reserved field. 0 will be padded.

Firstly, it takes the source address, destination address and the first 8-bit of the data part of the data packet from the data packet, joins them in the way of (src-ip, dst-ip, first 8-bit of payload), and then joins the tag generated by the state machine at this time, the credible level of the SAVA architecture adopted by this AD and the length of the credible prefix to hash the concatenated string with the hash algorithm to get a new message digest. Then it is reduced to a 32-bit packet signature by clipping and folding algorithm. The AER adds the 32-bit packet signature together with the 2-bit credible level and the 7-bit credible prefix length to the SAVA-X option, fills the option into 64-bit, and forwards it. At the AER of the destination AD, the same splicing and the same hash operation are performed to verify whether the generated string is consistent with the signature of the data packet. If they are consistent, they are forwarded. Otherwise, it is considered that the source address is forged and the data packet is discarded.

Due to the problem of time synchronization, when both old and new tags are valid, both old and new tags need to be verified. As long as one of them passes the verification, the packet should be forwarded. The original tag generated by the state machine will not appear in the packet. The attackers do not know the tag generated by the state machine at this time, so they can not forge the packet signature in the same way, which ensures the security of the data communication plane.

7. MTU Consideration

As the AER adds an option to the packet, the length of this packet is increasing, causing the MTU problem. This problem could taken place in source AER or the link between source AER and destination AER. If it occurs on the source AER, the source AER returns the ICMP message of "packet too big" to the source host and informs the host to reduce the packet size. Otherwise, if it occurs on other links from the source AER to the destination AER, which means the packet size exceeds the MTU of other links from the source AER to the destination AER after adding the tag, the corresponding router will return the ICMP message of "packet too big" to the source host. However, after the source host adjusts its own MTU, the problem MAY still exist because the root cause is AER causing packet size to exceed MTU, and the host does not know it. This problem can be solved by the following two methods. First, the MTU of the external link is set to 1280 at the source AER as this is the minimum value of MTU under IPv6. Then the MTU of the source host end will be set to the minimum value of MTU subtracting the maximum value of the SAVA-X option. This method can solve the problem, but it greatly limits the packet size and wastes the available MTU. The second is to monitor the ICMP message of "packet too big" sent to the host in the domain at the source AER. If such a message is monitored, the expected MTU value in the message minus the maximum value of the SAVA-X option will be forwarded. This method makes good use of MTU value to a certain extent, but it causes a large monitoring cost.

8. Security Consideration

TBD.

9. IANA Considerations

SAVA-X is designed for IPv6-enabled networks. It takes a destination option, SAVA-X option, and header to carry the Tag. We recommend using 00111011, i.e. 59, for the SAVA-X option. Here we give our SAVA-X option format in use.

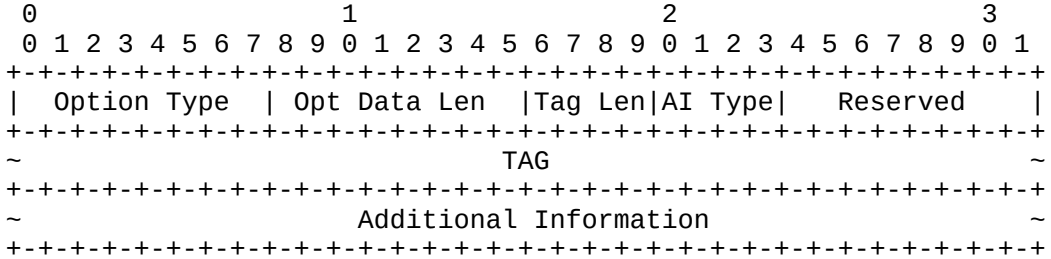

Figure 5: Format of SAVA-X option.

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

Opt Data Len:
8-bit field. The bytes length of the SAVA-X option. Its value is 2 + LenOfAI + (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 (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.

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

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

Additional Information:
As defined in the AI Type field.

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/rfc/rfc1760>.

[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/rfc/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/rfc/rfc8200>.

[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/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

Acknowledgments

TODO acknowledge.

Authors' Addresses

Ke Xu
Tsinghua University
China
Email: xuke@tsinghua.edu.cn

Xiaoliang Wang
Tsinghua University
China
Email: wangxiaoliang0623@foxmail.com

Yangfei Guo
Tsinghua University
China
Email: guoyangfei@zgclab.edu.cn

Jianping Wu
Tsinghua University
China
Email: jianping@cernet.edu.cn