IEICE TRANS. FUNDAMENTALS, VOL.E106–A, NO.3 MARCH 2023

PAPER Special Section on Cryptography and Information Security

APVAS: Reducing the Memory Requirement of AS_PATH Validation by Introducing Aggregate Signatures into BGPsec*

Ouyang JUNJIE[†], ***Nonmember***, **Naoto YANAI**[†a)], ***Member***, **Tatsuya TAKEMURA**[†], ***Nonmember***, **Masayuki OKADA**[††], **Shingo OKAMURA**[†††], ***Members***, ***and*** **Jason Paul CRUZ**[†], ***Nonmember***

SUMMARY The *BGPsec* protocol, which is an extension of the border gateway protocol (BGP) for Internet routing known as BGPsec, uses digital signatures to guarantee the validity of routing information. However, the use of digital signatures in routing information on BGPsec causes a lack of memory in BGP routers, creating a gaping security hole in today's Internet. This problem hinders the practical realization and implementation of BGPsec. In this paper, we present *APVAS (AS path validation based on aggregate signatures)*, a new protocol that reduces the memory consumption of routers running BGPsec when validating paths in routing information. APVAS relies on a novel aggregate signature scheme that compresses individually generated signatures into a single signature. Furthermore, we implement a prototype of APVAS on *BIRD Internet Routing Daemon* and demonstrate its efficiency on actual BGP connections. Our results show that the routing tables of the routers running BGPsec with APVAS have 20% lower memory consumption than those running the conventional BGPsec. We also confirm the effectiveness of APVAS in the real world by using 800,000 routes, which are equivalent to the full route information on a global scale.

key words: *BGPsec, path validation, aggregate signatures, internet routing, memory consumption*

1. Introduction

1.1 Backgrounds

The *Border Gateway Protocol (BGP)* [2] enables networks, such as an Internet service provider (ISP), to exchange routing information at the level of *autonomous system (AS)* by assigning a unique number to each AS. BGP is also the primary routing protocol used in the backbone of the Internet. However, BGP does not verify the validity of routing information being exchanged, and thus an AS always registers routing information received from other ASes as valid even when an adversary manipulates the routing information. This fundamental flaw in BGP has caused many incidents that resulted in heavy and severe damages, e.g., YouTube hijacking[**] and Ethereum hijacking[***]. According to some measurement results [3], such a hijack happens about four times a day on average. Therefore, guaranteeing the validity of routing information in BGP is an urgent and significant issue.

To tackle the issue mentioned above, technologies that guarantee the security of BGP in a *cryptographic* fashion have attracted attention. Loosely speaking, these technologies aim to verify the validity of routing information via generation and verification of digital signatures in the routing information. Specifically, signatures can be used in two ways, namely, *route origin validation* that only allows advertisements for an IP prefix by the legitimate AS as a prefix owner and *path validation* that guarantees all members of an AS path which is a connection of ASes from a source to a destination. Route origin validation is almost practical by virtue of the practical realizations of RPKI [4] and ROA [4], [5] as related protocols. In contrast, path validation has no practical realization even though it is instantiated by BGPsec [6] because its use of digital signatures significantly increases the memory consumption of BGP routers. For instance, according to a recent estimation [7], BGPsec is required to have memory storage of several tens of gigabytes because routers running BGPsec need to store all signatures according to the current specification of BGPsec [6]. The issue related to memory consumption is known as the *memory consumption problem*. Moreover, BGPsec lacks experimental evaluations, and thus a precise evaluation of the memory consumption problem remains incomplete.

BGP hijacking has also given rise to the hijacking of cryptocurrencies [8], [9], such as Bitcoin, as a new aspect of cybercrime. A recent finding has shown that the use of BGPsec can only prevent BGP hijacking [10]. Therefore, an essential issue in BGP security can be solved by making BGPsec practical, i.e., *by reducing memory consumption of routers and solving the memory consumption problem*.

1.2 Contribution

In this paper, we present a new path validation protocol named *APVAS (AS path validation based on aggregate signatures)*, which utilizes aggregate signatures [11] to combine

Manuscript received March 15, 2022.
Manuscript revised August 19, 2022.
Manuscript publicized January 11, 2023.

[†]The authors are with the Graduate School of Information Science and Technology, Osaka University, Suita-shi, 565-0871 Japan.
[††]The author is with University of Nagasaki, Sasebo-shi, 856-8580 Japan.
[†††]The author is with National Institute of Technology, Nara College, Yamatokoriyama-shi, 639-1080 Japan.

*This paper is a full version of our previous work published in arXiv [1].

a) E-mail: yanai@ist.osaka-u.ac.jp

DOI: 10.1587/transfun.2022CIP0024

**YouTube Hijacking: https://www.ripe.net/publications/news/industry-developments/youtube-hijacking-a-ripe-ncc-ris-case-study

***Ethereum Hijacking: https://www.internetsociety.org/blog/2018/04/amazons-route-53-bgp-hijack/

individual signatures into a single short signature and solve the memory consumption problem. Moreover, we implement a prototype of APVAS on a router daemon software. Our prototype is a first attempt to measure the memory consumption of routers using state-of-the-art cryptography in actual devices. Our experimental results show that APVAS can reduce the memory consumption of routers by 20% compared to the conventional BGPsec. Furthermore, we confirm via experiments whether APVAS can be used in the real world by using 800,000 routes, which are equivalent to the full route information on a global scale.

This paper presents two technical contributions. The first contribution is the proposal of a novel aggregate signature scheme named *bimodal aggregate signatures*. Aggregate signatures are expected to apply to BGPsec in cryptographic theory, but aggregate signatures in early literature [11], [12] are unsuitable for the current specifications of BGPsec. More precisely, *when the original aggregate signatures are trivially deployed in BGPsec, either the capability for signature aggregation or the security will be lost*. In contrast, APVAS can decrease the memory requirement of path validation as well as keep the security of BGPsec by the use of bimodal aggregate signatures (see Sect. 4 for details).

The second contribution is the implementation of a prototype of APVAS by extending *BIRD Internet Routing Daemon (BIRD)*[†], which is a software that virtualizes a BGP router. We succeeded in *measuring the performance of APVAS in an actual environment* by leveraging BIRD. Although our experiments were conducted on linear network topology, as far as we know, this is the first time that aggregate signatures are evaluated in an actual environment. Moreover, by extending our prototype, we can potentially evaluate protocols in future works (see Sect. 6 for details). The source codes of our APVAS prototype are available on GitHub[††].

2. Related Work

This section describes related works on BGP security and aggregate signatures.

2.1 BGP Security

The closest works to APVAS are the aggregate path authentication [13], APAT [14], and APVAS+ [15]. These works introduced aggregate signatures [11] in BGPsec (and S-BGP [16]) to aggregate individually generated signatures into a single short signature. More precisely, the aggregate path authentication is a trivial use of aggregate signatures, and APAT does not provide signature chains, which is the primary motivation for proposing bimodal aggregate signatures (see Sect. III-C for details). In contrast, we propose the bimodal aggregate signatures by improving the algorithms of aggregate signatures [11], and thus our work is significantly different from the works described above. APVAS can also be introduced into versatile extensions of BGP [17], [18] and experimental platforms [19], [20].

In recent years, APVAS+ [15] was proposed as an improved construction of APVAS. APVAS+ successfully provides the withdrawn process of routes, which is outside the scope of APVAS. Readers interested in our work are advised to read APVAS+, which provides state-of-the-art results on aggregate signatures. To the best of our knowledge, there is no commercial implementation of the conventional BGPsec itself [21] although BGPsec fully overcomes security concerns according to recent works [10], [22], [23]. Meanwhile, partial deployment of BGPsec is rather meaningless [24], and BGP oscillation will also be introduced [25].

In the past years, BGP security research [26]–[28] aimed to serve a quick response with "decent" security by utilizing *filtering* instead of digital signatures. For example, the use of filtering can prevent 85% of hijacking [27], whereas paths can be repaired from a hijacking within a minute [28]. These results show how threats are mitigated in the real world. However, a filtering-based approach makes the BGP security difficult to distinguish hijacking for a defense to DDoS or that by an adversary from a third party's standpoint.

2.2 Aggregate Signatures

Aggregate signatures were originally proposed in [11] as a cryptographic scheme to compress individually generated signatures into a single short signature. Current schemes are classified into two types, i.e., general aggregate signatures [11], [29]–[35] and sequential aggregate signatures [12], [36]–[39]. Informally, while sequential aggregate signatures support signature chains whereby each signer signs the signatures generated by the previous signer, general aggregate signatures do not support such chains because each signer generates signatures anytime. According to the early literature [40], sequential aggregate signatures are suitable for BGPsec because BGPsec requires each AS to sign both the route information and the signatures generated by previous ASes. However, sequential aggregate signatures do not support aggregation of individual signatures, and hence the data size of signatures increases linearly in proportion to the number of paths [41]. Therefore, an extension of the general aggregate signatures may be desirable for BGPsec.

The bimodal aggregate signatures presented in this paper are a new kind of aggregate signatures that combine signature chains by using the sequential aggregate signatures as used similarly in general aggregate signatures. The bimodal aggregate signatures are seen as an extension of history-free sequential aggregate signatures [39] whereby the signatures are aggregated by the aggregation algorithm in [11]. The security of bimodal aggregate signatures can be proven formally.

Meanwhile, there is the collateral signature problem [14], [42] in which verification of signatures fails when a fault signature is aggregated, as a common problem of

[†]BIRD: https://bird.network.cz/
[††]https://github.com/fseclab-osaka/apvas

aggregate signatures. Countermeasures to the collateral signature problem have been proposed in the existing methods [14], [42], [43] and we note that bimodal aggregate signatures can be extended in the same manner.

3. Border Gateway Protocol Security Extension (BGPsec)

This section provides backgrounds on BGP hijacking, path validation, and the main problem of BGPsec.

3.1 Motivating Example: BGP Hijacking

As a motivating example of BGPsec, we explain route hijacking on BGP below. First, we describe the protocol specification of BGP when finding a route to a destination on the entire Internet per AS unit. Each AS is assigned a unique AS number, and BGP uses these AS numbers to distinguish ASes and exchanges routing information via TCP. After a TCP connection is established, ASes exchange the routing information with each other as an update message, which contains `Network Layer Reachability Information (NLRI)`, the IP prefix of a destination, and the `AS_PATH` which is the route to a destination. Figure 1 shows an example of a route advertisement on BGP. BGP routers decide the best path to each IP prefix under the received route information and the operators' static policy if there are multiple routes for the IP prefix. In doing so, ASes append their respective AS number to `AS_PATH` and advertise the IP prefix and the `AS_PATH` to the neighbors as the best path. As a result, chains of AS numbers to reach each IP prefix as `AS_PATH` are configured. Following the best path selection algorithm, BGP chooses the route with the shortest length of `AS_PATH` as the best path, e.g., AS4 in Fig. 1 registers the `AS_PATH` of `AS3 AS2 AS1` as the best path to AS1 with 192.0.2.0/24 because the length of the `AS_PATH` of `AS3 AS2 AS1` is shorter than that of the `AS_PATH` of `AS5 AS3 AS2 AS1`.

However, the `AS_PATH` included in the update message can be intentionally rewritten by an AS to launch a route hijacking. As shown in Fig. 2, AS5 advertises AS4 a shorter route to AS1, i.e., `AS5 AS1`. Despite the nonexistence of the route information, AS4 registers this information in its routing table according to the best path selection algorithm. BGPsec [6] prevents route hijacking by validating the `AS_PATH` with the use of digital signatures on the routing information. Hereafter, we focus on the path validation provided by BGPsec.

3.2 Path Validation

Figure 3 shows an example of a route advertisement on BGPsec. BGPsec is required to have memory storage of several tens of gigabytes because routers running BGPsec need to store all signatures according to the current specification of BGPsec [6]. In BGPsec [6], `BGPsec_PATH` attributes are defined instead of the `AS_PATH` attributes of

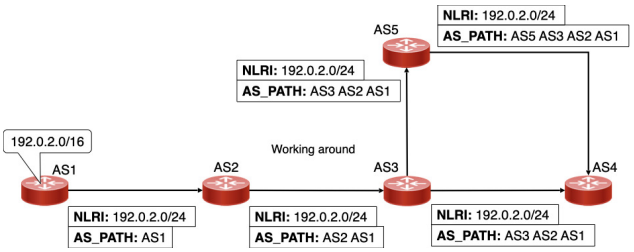

Fig. 1 Route advertisement on BGP.

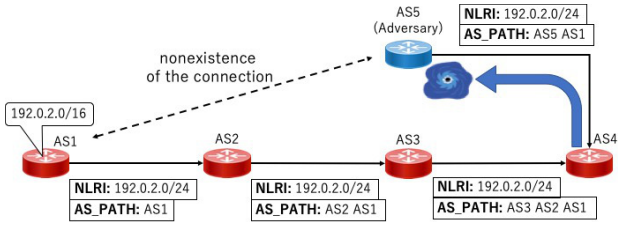

Fig. 2 Example of route hijacking.

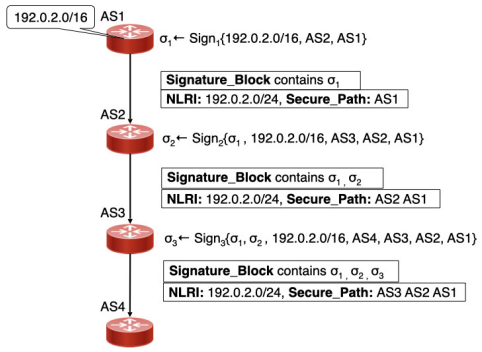

Fig. 3 Route advertisement on BGPsec.

BGP. `BGPsec_PATH` attributes contain `Secure_PATH` and `Signature_Blocks`. `Secure_PATH` lists the AS numbers of all ASes that the routing information passed through, and it is identical to `AS_PATH` of the conventional BGP. In contrast, `Signature_Block` stores digital signatures generated by each AS specified in `Secure_PATH`. The signature length varies according to the signature algorithm specified by `Algorithm Suite Identifier`. The number of Signature Blocks varies depending on the type of signature scheme, i.e., `Algorithm Suite Identifier`. Therefore, the higher the number of Signature Blocks, the more signature schemes can be supported.

Each AS sends and receives an update message containing these parameters. We describe the important parameters below:

Target AS Number: AS number of the destination of routing information.

Signature Segment: Digital signatures, where the number of the signatures is identical to the number of ASes that the routing information passed through excluding the route origin.

Secure_Path Segments: AS numbers of ASes that the rout-

Table 1 Existing evaluation [7] of memory requirement for BGP and BGPsec.

	BGP		BGPsec	
Year	Number of Paths	Memory Requirement	Number of Paths	Memory Requirement
2020	6332177	0.13 GB	6332177	2.79 GB
2021	4433446	0.09 GB	10130562	4.47 GB
2022	2547235	0.05 GB	14201374	6.62 GB
2023	1149812	0.02 GB	18111088	7.99 GB
2024	355617	0.01 GB	21794419	9.61 GB
2025	0	0 GB	25472541	11.23 GB

The memory requirement of BGPsec is evaluated by multiplying the number of paths in each year with the data size of an update message with ECDSA-256, i.e., 420 bytes. In addition to the update message, the data size of routing tables is further included in the evaluation. Interested readers are advised to read [7] for more details about this evaluation. Meanwhile, the memory requirement of BGP is evaluated only by the data size of routing tables because the update message of BGP does not include signatures.

ing information passed through, where at least the AS number of the route origin is required.
Algorithm Suite Identifier: An identifier for specifying the signature algorithm used for signature generation.
NLRI: Values of network addresses and their subnet mask managed by the route origin.

3.3 Problem Setting

Since an update message on BGPsec contains digital signatures, the update message balloons and a big part of which comes from the digital signatures. National Institute of Standards and Technology (NIST) [7] has shown the estimation results for memory consumption of routers on BGPsec as shown in Table 1. According to NIST, a BGP update message has an average size of 78 bytes, while a BGPsec update message is 388 bytes to 1188 bytes in size, depending on the signature algorithms. In Table 1, the columns of Memory Requirement show total values for routing tables registering destinations and that `BGPsec_PATH` attributes. Route information on BGPsec for the worldwide level, i.e., full route, requires a router to own more than 10 gigabytes of memory. Discussions and deployment of BGPsec began in 2016, and the deployment is estimated to finish in 2025. However, *the complete deployment is nowhere in sight due to the memory consumption problem.*

Solving the problem described above is non-trivial. The use of the general aggregate signatures [11], [29]–[31], [33], [34] to compress individual signatures into a single short signature seems to be a potential approach to solve the memory consumption problem. However, the general aggregate signatures do not provide a function that allows each AS to sign signatures generated by the previous ASes. We call such a function *signature chain* for the sake of convenience. A signature chain strictly guarantees the connection between an origin AS and the current AS for each `AS_PATH`. Intuitively, signature chains are necessary for the security of BGPsec, but the general aggregate signatures cannot provide them. Thus, the security guarantee by BGPsec might be lost if general aggregate signatures are deployed trivially. A signature chain can be constructed by gathering the signatures from all ASes, indicating that the number of signatures is linear for the number of ASes for each `AS_PATH`. Therefore, in the scenario described above, general aggregate signatures are useless. It also indicates that either the security guarantee or the efficiency, i.e., reducing memory consumption, will be lost.

4. Bimodal Aggregate Signatures

This section discusses the bimodal aggregate signature scheme used as a new building block in APVAS. We first describe the central concept of the bimodal aggregate signature scheme. We then show formal definitions and the construction of the scheme. Finally, we show the correctness of the scheme and security analysis via formal proof.

4.1 Main Concept

In the bimodal aggregate signatures, when n users generate n individual signatures, the signature aggregation is executed in two ways, i.e., an interactive style of general aggregate signatures [11], [29]–[34] and a signature-chain style of sequential aggregate signatures [12], [36]–[40].

Compared to the general aggregate signatures [11], [29]–[34], bimodal aggregate signatures can strictly provide the security of BGPsec under signature chains as well as the efficiency of aggregating individual signatures even on `Secure_PATH`s. Specifically, the security of BGPsec depends on signature chains, and only the sequential aggregate signatures [12], [36]–[40] provide such chains via signature aggregation.

On the other hand, to improve efficiency, i.e., reducing memory consumption, signatures even on different `Secure_PATH`s should be aggregated after route advertisement has converged. The reason for NOT aggregating signatures between different `Secure_PATH`s until route convergence is that the process would be complicated if a shorter length of `Secure_PATH` is found after aggregating the signatures of some `Secure_PATH`. More concretely, to update the shorter length of `Secure_PATH`, we need to remove the previous `Secure_PATH` and then store the shorter `Secure_PATH` in a routing table. In doing so, extracting an original signature for the previous `Secure_PATH` from the aggregated signature is hard [44], and thus the previous `Secure_PATH` cannot be removed after aggregation. Consequently, aggregating signatures is desirable only after `Secure_PATH` is no longer removed. Note that even after route convergence, there may be an incident that causes routes to be withdrawn. In this case, as described in Sect. 2.1, it can be addressed by introducing the APVAS extension [15]. Although the above function is outside the scope of the BGPsec specification [6] in the sense that `Secure_PATH`s whose signatures are aggregated cannot be advertised anymore, the memory consumption can be significantly reduced. Such capability is inspired only by signature aggregation of the general aggregate signatures. In other words, both the

security and the efficiency can be achieved if the two ways for signature aggregation described above are executed simultaneously.

Based on the above observation, bimodal aggregate signatures are designed to provide both the signature chain of sequential aggregate signatures and the signature aggregation of the general aggregate signatures. The main idea is to sign the verification equations instead of the signatures themselves. Generally speaking, the verification represents the validity of signatures and can be used independently of the signature aggregation. Hence, bimodal aggregate signatures can solve the memory consumption problem without sacrificing security.

4.2 Assumption

The proposed scheme is based on pairings defined as follows. Let $\mathbb{G}$ and $\mathbb{G}_T$ be groups with a prime order p. Then, a bilinear map $\boldsymbol{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is a map with the following conditions: for any $U, V \in \mathbb{G}$ and $a, b \in \mathbb{Z}_p^*$, $\boldsymbol{e}(aU, bV) = \boldsymbol{e}(U, V)^{ab}$ holds; for any generator $P \in \mathbb{G}$, $\boldsymbol{e}(P, P) \neq 1_{\mathbb{G}_T}$ holds, where $1_{\mathbb{G}_T}$ is an identity element in $\mathbb{G}_T$; and, for any $U, V \in \mathbb{G}$, $\boldsymbol{e}(U, V)$ can be computed efficiently. We assume that solving the discrete logarithm problems in $\mathbb{G}$ and $\mathbb{G}_T$ is computationally hard. We call the parameter $(p, \mathbb{G}, \mathbb{G}_T, \boldsymbol{e})$ achieving the conditions above as *pairing parameter*.

The security of the proposed scheme is proven under the computational Diffie-Hellman (CDH) assumption in $\mathbb{G}$ defined below.

Definition 1 (ϵ-CDH Assumption in $\mathbb{G}$): We define a CDH problem with a security parameter 1^κ as follows: for a pairing parameter $(p, \mathbb{G}, \mathbb{G}_T, \boldsymbol{e})$ and a given tuple (P, aP, bP) with uniformly random $(a, b) \leftarrow \mathbb{Z}_p^2$ as input, compute abP. We say that a ϵ-CDH assumption in $\mathbb{G}$ holds if there is no probabilistic polynomial-time algorithm that can solve the CDH problem with a probability greater than ϵ.

4.3 Formal Definitions of Bimodal Aggregate Signatures

We define the syntax of a bimodal aggregate signature scheme and its correctness below.

4.3.1 Syntax

The algorithms of a bimodal aggregate signature scheme are defined as follows. Here, let $\mathcal{M}$ be a message space and $\mathcal{PK}$ be a public key space.

Setup: Given a security parameter 1^κ, output a public parameter $para$.

UserKeyGen: Given $para$, output a secret key sk and a public key pk.

SeqAggSign: Given $para$, a secret key sk_i, a public key pk_i, a message $m_i \in \mathcal{M}$ to be signed, a set $L = \{(pk_j, m_j)\}_{j \in S}$ of pairs of public keys and messages for a set S of the previous signers, and an aggregate signature σ, return a new aggregate signature σ' on a new set $L' = L \cup \{(pk_i, m_i)\}$ or $\perp$ to indicate an error.

AggSign: Given $para$, sets $L_1 = \{(pk_j, m_j)\}_{j \in S_1}$ and $L_2 = \{(pk_j, m_j)\}_{j \in S_2}$, and aggregate signatures σ_1 and σ_2, return a new aggregate signature σ' on a new set $L' = L_1 \cup L_2$ or $\perp$ to indicate an error.

Verify: Given $para$, a set $L = \{(pk_j, m_j)\}_{j \in S}$ of pairs of public keys and messages, and an aggregate signature σ, output True or False.

The correctness of the bimodal aggregate signature scheme is then defined as follows.

Definition 2 (Correctness): For all $para \leftarrow \mathbf{Setup}(1^\kappa)$, all $(sk, pk) \leftarrow \mathbf{UserKeyGen}(para)$, all $m_i \in \mathcal{M}$, and all $L \subseteq \mathcal{PK} \times \mathcal{M}$, the following condition holds:

$$\text{True} = \mathbf{Verify}\left(\begin{array}{c} para, L', \\ \mathbf{SeqAggSign}\left(\begin{array}{c} para, \\ \mathbf{UserKeyGen}\left(para\right), \\ m_i, L, \sigma \end{array}\right) \end{array}\right),$$

where $L' = L \cup \{(pk_i, m_i)\}$. We say that a bimodal aggregate signature scheme is correct if the condition described above holds. In the above equation, L' and $\mathbf{SeqAggSign}\left(para, \mathbf{UserKeyGen}\left(para\right), m_i, L, \sigma\right)$ can be related with $L* = L_1 \cup L_2$ for any $L_1, L_2 \subseteq \mathcal{PK} \times \mathcal{M}$ and the following equation, respectively:

$$\mathbf{AggSign}\left(\begin{array}{c} para, \\ \mathbf{SeqAggSign}\left(\begin{array}{c} para, \\ \mathbf{UserKeyGen}\left(para\right), \\ m_i, L_1, \sigma \end{array}\right), \\ \mathbf{SeqAggSign}\left(\begin{array}{c} para, \\ \mathbf{UserKeyGen}\left(para\right), \\ m_i, L_2, \sigma \end{array}\right) \end{array}\right).$$

4.3.2 Security Definitions

We define the unforgeability of signatures for preventing a signature forgery via received signatures as the security of a bimodal aggregate signature scheme. The unforgeability of a bimodal aggregate signature scheme is proven via the following game between a challenger C and an adversary $\mathcal{A}$. An advantage of $\mathcal{A}$ is defined as a probability whereby C outputs *accept* in the game. The game follows the certified key model [45] of sequential aggregate signature scheme [46] but is slightly different in the sense that the aggregation function can be utilized by $\mathcal{A}$ in local. Hereafter, we denote by $x^{(i)}$ a value of the i-th query for all x.

Initial Phase: The challenger C generates $para \leftarrow \mathbf{Setup}(1^\kappa)$ and $(sk^*, pk^*) \leftarrow \mathbf{UserKeyGen}(para)$ as a challenge. C then runs $\mathcal{A}$ with $(para, pk^*)$ as input.

Certify Query: $\mathcal{A}$ sends a pair of $(sk^{(h)}, pk^{(h)})$ to C, and then C registers $(sk^{(h)}, pk^{(h)})$ if they are a valid pair.

Sign Query: $\mathcal{A}$ sends a signing query $(para, m^{(h)}, L^{(h)}, \sigma)$ to C, and then C returns a signature σ.

Output After q_c iterations of the **Certify Query**, and q_s iterations of the **Sign Query**, $\mathcal{A}$ outputs (L^*, σ^*), where $L^* = \{(pk_i^*, m_i^*)\}_{i=1}^N$ with $N \in \mathbb{N}$ and the following conditions hold: the **Verify** algorithm outputs True; there is exactly one

$pk^*_{i^*}$ such that $pk^*_{i^*} = pk^*$ holds; for any i, pk^*_i in L appears in $\{pk^{(h)}_i\}_{h=1}^{q_c}$ holds for the **Certify Query** except for pk^*; for the $pk^*_{i^*}$, a tuple of $(m^*_{i^*}, L^*_{i^*-1})$ where $L^*_{i^*-1} = \{(pk^*_j, m^*_j)\}_{j \in S}$ has never been queried to the **Sign Query**; and all pk^*_i in L^* are distinct. If all the conditions hold, then C outputs $accept$. Otherwise, C outputs $reject$.

Definition 3: We say that a bimodal aggregate signature scheme is $(q_c, q_s, q_h, N, \epsilon)$-unforgeable if there is no probabilistic polynomial-time adversary $\mathcal{A}$ who forges with $(q_c, q_s, q_h, N, \epsilon)$. Here, we say that $\mathcal{A}$ forges the scheme with $(q_c, q_s, q_h, N, \epsilon)$ if a challenger C outputs $accept$ with a probability greater than ϵ in the security game described above. Here, $\mathcal{A}$ can generate at most q_c key certification queries, at most q_s signing queries, and at most q_h random oracle queries, and N is the number of signers in $\mathcal{A}$'s output.

4.4 Algorithms

Hereafter, each signer is represented by a unique index i in the algorithms for convenience. We denote by S a set of signers for any signature and by S_i a set of signers who join a chain of signatures from 1 to i for any i. We also denote by $\|$ a concatenation of any string and by $\|_{j \in S_i}$ concatenations of strings for any signer $j \in S_i$.

We show the construction in **Algorithms** 1–5. The fourth line of **Algorithm** 3 and the third line of **Algorithm** 5 are identical to a signature chain, i.e., σ indicates a signature. More precisely, σ indicates a signature, and $\mathbf{e}(\sigma, P)$ in **Algorithm** 3 and $\mathbf{e}(c_j, X_i)$ in **Algorithm** 5 are identical to computations which are the main core for verification of signatures. Intuitively, by inputting signatures and their verifications to a hash function, the same signature chains are constructed. Even after σs themselves are aggregated and lost, the corresponding $\mathbf{e}(c_j, X_i)$'s can be computed. Therefore, both a signature chain of sequential aggregate signatures and an interactive aggregation of general aggregate signatures can be provided simultaneously.

Meanwhile, **Algorithm 4** aggregates only two signatures, but any number of signatures can be aggregated by iterating the algorithm.

4.5 Correctness of the Scheme

In this section, we briefly show that our scheme in Sect. 4 is correct in the meaning of Definition 2. In particular, we show that the **Verify** algorithm outputs True for any signature output by **Sign**.

Theorem 1: The proposed scheme is correct.

Proof. First, for any (sk, pk), the following equation holds on the **Verify** algorithm by the conditions of pairings:

$$\begin{aligned} \mathbf{e}(\sigma, P) &= \mathbf{e}\left(\sum_{i \in S} x_i \cdot H(c_i), P\right) \\ &= \prod_{i \in S} \mathbf{e}\left(x_i \cdot H(c_i), P\right)) \\ &= \prod_{i \in S} \mathbf{e}\left(H(c_i), x_i \cdot P\right), \end{aligned}$$

where $c_i = H\left(\left(\prod_{j \in S_i} \mathbf{e}(c_j, X_i)\right) \|_{j \in S_i} (pk_j \| m_j)\right)$ for any i. Therefore, the algorithm returns True and thus the proposed scheme is correct. □

Algorithm 1 Setup

Ensure: Public parameter $para$
1: Generate pairing parameter $(p, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$
2: $P \leftarrow \mathbb{G}$
3: Choose a hash function $H : \{0,1\}^* \rightarrow \mathbb{G}$
4: $para = (p, \mathbb{G}, \mathbb{G}_T, \mathbf{e}, P, H)$

Algorithm 2 UserKeyGen

Require: Public parameter $para$
Ensure: Secret key sk, public key pk
1: $x \leftarrow \mathbb{Z}_p$
2: $X = xP$
3: $sk = x$, $pk = X$

Algorithm 3 SeqAggSign

Require: public parameter $para$, secret key sk_i, public key pk_i, plaintext $m_i \in \{0,1\}^*$, list $L = \{(pk_j, m_j)\}_{j \in S}$ of public keys and plaintexts, signature σ
Ensure: Signature σ, list $L' = \{(pk_j, m_j)\}_{j \in S} \cup \{(pk_i, m_i)\}$ of public keys and plaintexts
1: **if** $L = \emptyset$ **then**
2: set $\sigma = 0$
3: **end if**
4: $c = H\left(\mathbf{e}(\sigma, P) \| pk_i \| m_i \|_{j \in S_i} (pk_j \| m_j)\right)$
5: $\sigma = \sigma + x \cdot H(c)$

Algorithm 4 AggSign

Require: public parameter $para$, list $L_1 = \{(pk_j, m_j)\}_{j \in S}$ of public keys and plaintexts, list $L_2 = \{(pk_j, m_j)\}_{j \in S'}$ of public keys and plaintexts, signature σ_1, signature σ_2
Ensure: signature σ, list $L' = L_1 \cup L_2$
1: $\sigma = \sigma_1 + \sigma_2$

Algorithm 5 Verify

Require: public parameter $para$, list $L = \{(pk_j, m_j)\}_{j \in S}$ of public keys and plaintexts, signature σ
Ensure: True or False
1: For any $i \in S$, parse pk_i as X_i
2: **if** all $(pk_i, m_i) \in S$ are distinct **then**
3: $\forall i, c_i = H\left(\left(\prod_{j \in S_i} \mathbf{e}(c_j, X_i)\right) \|_{j \in S_i} (pk_j \| m_j)\right)$
4: **if** $\mathbf{e}(\sigma, P) = \prod_{i \in S} \mathbf{e}(H(c_i), X_i)$ **then**
5: **return** True
6: **end if**
7: **end if**
8: **return** False

4.6 Security Analysis of the Scheme

We analyze the security of the proposed scheme via a formal proof. The security of the proposed scheme can be proven under the CDH assumption in $\mathbb{G}$ in the random ora-

cle model [47]. We give the details of the proof below.

Theorem 2: Suppose that a hash function H is modeled as a random oracle. The proposed scheme is then $(q_c, q_s, q_h, N, \epsilon)$-unforgeable under the ϵ-CDH assumption in $\mathbb{G}$, where

$$\epsilon' \geq \epsilon \cdot \left(\frac{1}{e(q_s+1)}\right)$$

with e as the base of the natural logarithm.

Proof. In this proof, a reduction algorithm $\mathcal{B}$ that solves the CDH problem is shown by an existence of an adversary $\mathcal{A}$ which forges the proposed scheme. Here, let $\mathcal{A}$ be an adversary who breaks the proposed scheme with $(q_c, q_s, q_h, N, \epsilon)$. $\mathcal{B}$ is given a challenge (P, aP, bP) in $\mathbb{G}$ and interacts with $\mathcal{A}$ as follows.

$\mathcal{B}$ sets $(p, \mathbb{G}, \mathbb{G}_T, \boldsymbol{e}, P)$ as $para$ and aP as pk^*. Next, $\mathcal{B}$ sets $\mathcal{PK}[-,-]$, $\sigma[-,-,-,-,-]$, $H[-,-,-]$ as a $\mathcal{PK}$-list for key certification queries, σ-list for signing queries, and a H-list for random oracle queries, respectively. Here, $\mathcal{B}$ utilizes a random coin $B \in \{0,1\}$ to set 1 with a probability ε to set 1, and we finally determine ε to complete the proof:

H **Query** $(c^{(h)})$**:** If the given query c has been registered in the list, $\mathcal{B}$ retrieves B_i from the list. Otherwise, $\mathcal{B}$ sets $B_i = 1$ with a probability δ, which is decided at the end of this proof, or $B_i = 0$ with $1-\delta$. Next, $\mathcal{B}$ generates $\alpha_i \leftarrow \mathbb{Z}_p^*$ and then sets $\mathbf{H}_1(c^{(h)}) = \alpha_i P$ for $B_i = 1$ or $\mathbf{H}_1(c^{(h)}) = \alpha_i P + bP$ for $B_i = 0$. $\mathcal{B}$ registers $(c^{(h)}, \alpha_i, B_i)$ in the H-list and returns the $\mathbf{H}_1(c^{(h)})$.

Certify Query $\left(sk^{(h)}, pk^{(h)}\right)$**:** $\mathcal{B}$ checks if $X^{(h)} = x^{(h)} \cdot P$ for the given $(sk^{(h)}, pk^{(h)})$ holds. If so, $\mathcal{B}$ registers $\left(sk^{(h)}, pk^{(h)}\right)$ in the $\mathcal{PK}$-list. Otherwise, $\mathcal{B}$ discards the given query.

Sign Query $\left(m_i^{(h)}, L^{(h)}, \sigma'\right)$**:** $\mathcal{B}$ computes $c_i^{(h)}$ for the given query via the random oracle and then checks if $B_i = 1$ for $c_i^{(h)}$ on the H-list. If so, $\mathcal{B}$ aborts the process. Otherwise, i.e., $B_i = 0$ for $c_i^{(h)}$, $\mathcal{B}$ retrieves α_i corresponding to $c_i^{(h)}$ from the H-list and then computes $\sigma = \alpha_i \cdot aP + \sigma'$ as a signature. Since the signature σ is accepted on the verification algorithm, the distribution in the simulation described above is indistinguishable from the standpoint of $\mathcal{A}$. $\mathcal{B}$ then returns a signature σ and registers $(m_i^{(h)}, L^{(h)}, c_i^{(h)}\sigma', \sigma)$ in the σ-list.

Output: After the simulation described above, $\mathcal{A}$ outputs a forgery (L^*, σ^*) where there exists exactly a single $pk_{i^*}^*$, which is identical to pk^*, from the definition of the forgery. $\mathcal{B}$ retrieves the values corresponding to c_{i^*} for pk^* from the H-list and checks if $B_{i^*} = 1$ holds. If not, $\mathcal{B}$ aborts the process. Otherwise, the signature can be written as follows because the verification holds:

$$\sigma = a(\alpha_{i^*} P + bP) + \sum_{i \in [1,n] \setminus \{i^*\}} (x_i \alpha_i P),$$

where x_i is a secret key registered in the $\mathcal{PK}$-list for any i. $\mathcal{B}$ can then extract a solution to the problem by the following computation:

$$abP = \sigma - \alpha_{i^*} aP - \sum_{i \in [1,n] \setminus i^*} (\alpha_i x_i P).$$

To complete the proof, we analyze the success probability ϵ' of $\mathcal{B}$. In this proof, there are two cases where $\mathcal{B}$ aborts the process; the first case $abort_S$ is in the **Sign Query** where $B_i = 1$ in the $\mathbf{H}_1$-list; and the second case $abort_{out}$ is in the **Output** where $B_{i^*=0}$ holds. The success probability can be then estimated as follows:

$$\begin{aligned}\epsilon' &= \epsilon \cdot \Pr[\bigwedge_{j=1}^{q_s} \neg abort_S] \cdot \Pr[\neg abort_{out}] \\ &\geq \epsilon \cdot (\varepsilon^{q_s}) \cdot (1-\varepsilon) \\ &\geq \epsilon \cdot \varepsilon^{q_s}(1-\varepsilon).\end{aligned}$$

The variable ε is finally determined in order to optimize the probability described above. Here, let $f(\varepsilon)$ be a function $\varepsilon^{q_s}(1-\varepsilon)$. Then, $f(\varepsilon)$ is maximized at $\varepsilon_{opt} := \frac{q_s}{q_s+1}$ according to the derived function. That is, the following inequation can be obtained for the function $f(\varepsilon)$:

$$\begin{aligned}f(\varepsilon_{opt}) &= \left(\frac{q_s}{q_s+1}\right)^a \left(1 - \frac{q_s}{q_s+1}\right) \\ &= \left(1+\frac{1}{q_s}\right)^{-q_s} \left(\frac{1}{q_s+1}\right) \geq e^{-1}\left(\frac{1}{q_s+1}\right),\end{aligned}$$

where e is the base of the natural logarithm. The success probability ϵ' is then bounded as follows:

$$\epsilon' \geq \epsilon \cdot \left(\frac{1}{e(q_s+1)}\right).$$

The probability is polynomially bounded. □

5. Design of APVAS

In this section, we present *AS path validation based on aggregate signatures (APVAS)* as a new path validation protocol for BGPsec. As described in Sect. 1, APVAS is based on bimodal aggregate signatures. We first describe how bimodal aggregate signatures are deployed for path validation as a protocol specification. Then, we describe the prototype implementation on the BIRD Internet routing daemon (BIRD), a BGP software.

5.1 Protocol Specification

Figure 4 shows an example of a route advertisement on APVAS. Each intermediate AS takes information from the received update message and then verifies the routing information with **Algorithm** 5. Then, for the received routing information and aggregate signatures, an AS generates a new aggregate signature with **Algorithm** 3 and sends it to the neighbor ASes.

`Signature Segment Format` defined in the BGPsec protocol contains `Subject Key Identifier (SKI)` with size of 20 bytes, `Signature Length` with size of 2 bytes, and `Signature` with a variable length as described in Sect. 3. In contrast, to contain only a single signature in APVAS, `SKI`

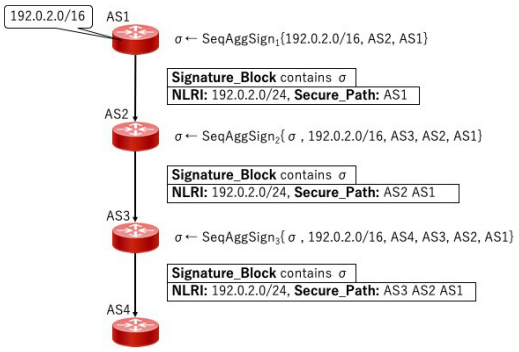

Fig. 4 Route advertisement on APVAS: The figure shows the use of each algorithm as the specification of APVAS.

Table 2 `Signature_Block` format on APVAS.

Signature_Block Length (2 octets)
Algorithm Suite Identifier (1 octet)
Signature Length (2 octets)
Signature (variable)
Subject Key Identifiers (SKIs) (variable)

is defined as `SKI Segment` with size of 20 bytes. The new `Signature_Block Format` of APVAS is shown in Table 2.

The proposed scheme, i.e., **Algorithms** 1–5, is utilized as the signature in APVAS as described above. We note that a plaintext m corresponds to a received update message. For data storing shown in Table 3, the remaining string except for `Signature Segment Format` is utilized as m. Likewise, for verification of update messages, the intermediate values of computation, i.e., outputs of bilinear maps, are utilized instead of a hashed value for the previous signers to generate signatures because the update messages do not include the hashed values. Consequently, the size of update messages is reduced. Moreover, after the convergence of paths, each intermediate AS can aggregate the individual signatures for each destination into a single short signature with **Algorithm** 4. The data size to be stored can then be reduced. This single short signature is stored in the router and deletes individual signatures while `BGPsec_PATH`s of all NLRIs remain stored in the router. Therefore, when APVAS advertises a route to peers after route convergence due to future topology changes, route refresh defined in RFC2918, graceful restart defined in RFC4724, etc., the `BGPsec_PATH`s of the NLRIs can be advertised to each peer without particular problem. Each signature for each NLRI can be reconstructed by using APVAS+ [15] with only one signature stored in the router, the public key of the ASes in each `BGPsec_PATH`, and the plaintext received from update messages and stored in the router. However, reconstructing signatures requires computational cost. We consider the computational cost of APVAS in Sect. 7.3. Although **Algorithm** 4 aggregates only two signatures, any number of signatures can be aggregated by iterating it.

We also describe the operation of parameters for deploying APVAS in the real world since APVAS needs to use pairing parameters and a type of hash function as global parameters to be agreed upon in advance. These parameters should be agreed upon globally in RFC or the IANA registry, and this kind of agreement about pairing parameters is in the progress on IETF[†]. Also, each AS generates a pair of secret and public keys individually. While the AS stores the secret key in a local environment as confidential information, the public key is registered in RPKI While the AS stores the secret key in a local environment as confidential information, the public key is registered in RPKI. The above methods for distributing the parameters to each organization are common with the conventional BGPsec, and thus they are considered to be practical.

Table 3 Sequence of octets to be hashed on APVAS.

e
Target AS Number
Signature Length (2 octets)
Signature (variable)
SKI :N-1
Secure_PATH Segment :N
. . .
SKI :1
Secure_PATH Segment :2
Secure_PATH Segment :1
Algorithm Suite Identifier
AFI
SAFI
NLRI

5.2 Prototype Implementation

We implement a prototype of APVAS in the C language. More specifically, APVAS is implemented by extending the BGPsec-enabled BIRD [48], which is an implementation of BGPsec on BIRD, with a pairing library TEPLA[††]. We call this implementation *prototype1* to distinguish it from another prototype used in other experiments. BIRD is a daemon software that utilizes a computer as a BGP router, and new functions written in the C language can be introduced to it. A router configuration and a network topology are specified by editing a configuration file. We have released our prototype implementation on GitHub[†††].

Source codes related to BGP are in directories under `proto/bgp/`, and the codes to be improved are as follows. First, we improve encoding and decoding of update messages by modifying the `encode_bgpsec_attr()` function and the `decode_bgpsec_attr()` function in `attrs.c`, respectively. We also improve signature generation and verification by replacing the `bgpsec_sign_data_with_key()` function and the `bgpsec_verify_signature_with_key()` function in `validate.c`, respectively, with an implementation of the bimodal aggregate signatures with TEPLA. The parameters utilized in pairing computation are shown in Table 4.

[†]https://datatracker.ietf.org/doc/draft-irtf-cfrg-pairing-friendly-curves/
[††]TEPLA: http://www.cipher.risk.tsukuba.ac.jp/tepla/index.html
[†††]https://github.com/fseclab-osaka/apvas

Table 4 Version of libraries and their utilized parameters.

TEPLA ver.	2.0
Pairing	ECBN254a
Finite Field	bn254_fpa

Table 5 Environment for experiments.

OS	Ubuntu16.04 LTS
CPU (two)	Intel Xeon Gold 6140
Memory	96 GB
Docker ver.	2.2.0.3
BIRD ver.	1.6.0
BIRD BGPsec ver.	0.9

6. Experiments

This section shows experimental evaluations of APVAS. To this end, we created prototype1, an implementation of APVAS. We then conduct experiments on a virtual network with a simple topology to evaluate the memory consumption of routers running APVAS and compare it with the results of BGPsec. Furthermore, we conduct experiments on the full route to confirm whether APVAS can be used in the real world. The primary purpose of our experiments is to understand the performance, i.e., in terms of memory consumption of routers, of APVAS on actual devices on a real-world scale. We also evaluate the computational cost as another metric of the performance of APVAS.

6.1 Experimental Settings

We describe the experimental settings below.

(1) Evaluation of Memory Consumption for `AS_PATH` Length

While BIRD can handle many kinds of network topologies, a linear network is the simplest network topology for understanding the relationship between the number of paths and the average `AS_PATH` length. In such a topology, AS routers are connected linearly. Moreover, the number of paths advertised to the network is the total number of paths statically advertised by each AS, and the average `AS_PATH` length is an average of distances between ASes which advertise the paths. In the experiments described below, we focus on a linear network to evaluate APVAS.

Twenty virtual routers are configured under the machine environment shown in Table 5, and private AS numbers from 65001 to 65021 are assigned to these routers, respectively. Each router is assigned with a static IP from 192.168.10.10, 192.168.10.20, 192.168.10.30, ..., 192.168.10.200, 192.168.10.210, and these routers connect to configure the network. For instance, AS65001 advertises routing information and then the other ASes receive it. More specifically, a router with an AS hop count of 1 from AS65001 is AS65002, a router with 2 hop is AS65003, and similarly, a router with 20 hop is AS65021. The above manner enables us to experiment with up to 20 routers from a router of the route origin. Since the upper bound of the number of NLRIs that BGPsec-enabled BIRD based prototype and its extended prototype of APVAS stably advertise is approximately 250 routes, i.e., prefixes. However, it often becomes unstable around 250 routes, so we adopt truly stable 200 routes as the number of NLRIs advertised from route origin, i.e., AS65001 in our experiments. We evaluate the memory consumption of AS65002 to AS65021 based on the environment described above by getting the total memory consumption by using the `show memory` command of BIRD. We then measure the computational time from when AS65001 starts to advertise 200 routes until AS65021 receives all the routes and the operation of the router's CPU converges.

(2) Evaluation of Memory Consumption with Full Routes

We want to confirm if APVAS can be used in the real world. To do so, we need to conduct additional experiments with 800,000 routes, which are equivalent to the full route information on a global scale according to the RIPE RIS[†] database. In such a situation, the memory consumption of a BGP router running APVAS is evaluated.

The prototype1 presented in the previous section is based on the BGPsec-enabled BIRD. We note that the BGPsec-enabled BIRD does not work stably when about 250 or more routes are advertised due to the bug, where a daemon of routers goes down when more than 250 routes are advertised. Although we tried to fix the bug, it could not be fixed. We thus cannot use prototype1 in experiments on the full routes. Therefore, we implemented *prototype2*, a new version prototype that can be used in the experiments on the full routes. The prototype2 of APVAS is implemented by extending BIRD [††], which works stably even when the full routes are advertised. Specifically, we edited `packets.c` and `bgp.h` under `proto/bgp`. The mechanisms for the APVAS specification are embedded in the `bgp_encode_prefixes()` function and the `bgp_do_rx_update()` function.

The experimental setting is the same as in the previous experiments, but the number of routers is 8 and the scales of routes to be advertised are 100,000, 200,000, ..., 800,000. For the performance evaluation of APVAS on a real-world scale, these experiments evaluate the amount of memory consumed by the entire virtual router running APVAS. Thus, the evaluation uses the `docker stats` command to measure the memory consumption of each docker container.

Note that the process of obtaining a pair of secret and pubic keys is outside the scope of the experiments. In the real-world setting, the resource public key infrastructure (RPKI) [4] although we do not take into account it. According to Chung et al. [49], RPKI is a hierarchical Public Key Infrastructure that binds Internet number resources (INRs) such as autonomous system numbers (ASNs) and IP

[†] https://www.ripe.net/analyse/internet-measurements/routing-information-service-ris

[††] https://bird.network.cz/

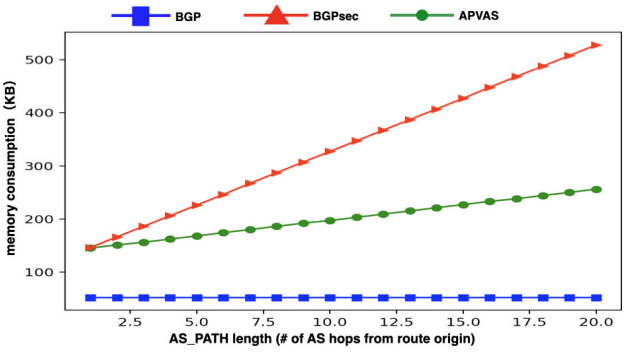

Fig. 5 The memory requirement for path validation of APVAS, BGP, and BGPsec.

addresses to public keys via certificates. Certificate holders can utilize the corresponding secret keys to make attestations about these INRs, most importantly, route origin authorization (ROA) objects. In general, secret keys on RPKI are also utilized for generating signatures for BGPsec. RPKI is "ready for the big screen" according to Chung et al. [49] and there is also an extension for certifying de facto ownership [50]. APVAS can still deploy RPKI because key management is unchanged and common in the conventional BGPsec. Likewise, routing information for advertisement is randomly generated in advance.

6.2 Results of Memory Consumption for AS_PATH Length

The memory consumption of the routers running APVAS, including the results under the same setting for BGP routers and the conventional BGPsec routers, is shown in Fig. 5, where each AS receives 200 routes. This figure shows only the memory requirement related to routing tables and BGPsec_PATH attributes, although BIRD includes the routing tables, BGPsec_PATH attributes, Route Origin Authorization (ROA) table, and the protocol information. The results show that APVAS successfully reduced the memory requirement compared to the conventional BGPsec because the memory requirements of other information in the current specification [6], e.g., ROA tables and the protocol itself, are stable in our experiments.

The results show that the memory consumption of routers running APVAS for all instances is smaller than that of routers running the conventional BGPsec. The memory consumption of routers running APVAS becomes more prominent at the average AS_PATH length one because of the data size of the bimodal aggregate signatures. While the conventional BGPsec utilizes ECDSA with a bit length of 384 bits per signature, the bimodal aggregate signatures have 512 bits.

As the evaluation of the computational cost, the route advertisement in BGP and BGPsec converged in a few seconds, while the route advertisement in APVAS took about five minutes until the convergence. The computational cost includes the route advertisement and signature generation/verification on BGPsec and APVAS, and thus the bimodal aggregate signatures for APVAS have higher computational cost than the ECDSA for BGPsec. Table 6 shows the detailed computational cost of APVAS.

Table 6 Evaluation on the computational cost of APVAS.

Length	Start [sec]	Convergence [sec]	Time [sec]
1	0.2	39	38.8
2	0.2	54.2	54
3	0.8	69.6	68.8
4	1	83.8	82.8
5	1.6	97.8	96.2
6	2	111.4	109.4
7	2.8	124.6	121.8
8	3.4	138.6	135.2
9	4	150.4	146.4
10	5	163.6	158.6
11	5.8	177	171.2
12	6.8	189.8	183
13	7.8	202.8	195
14	8.8	216	207.2
15	9.8	229	219.2
16	11.2	243	231.8
17	12.4	257.2	244.8
18	14	271.4	257.4
19	15.4	284.6	269.2
20	17	299.4	282.4

The "Length" column shows the length of AS_PATH where each router is running APVAS. The "Start [sec]" column shows the elapsed time when a router advertises full routes starts to advertise the first route. The "Convergence [sec]" column shows the elapsed time where each router receives the last route and validates it. Finally, the "Time [sec]" column shows the elapsed time in total, i.e., from the "Start [sec]" column to the "Convergence [sec]" column.

6.3 Results of Memory Consumption with Full Routes

We show the experimental results of the memory consumption of the entire routers running APVAS and BGP when the full routes are advertised. Figure 6 shows the measurement results when 100,000, 200,000,. . . , 800,000 routes are advertised, respectively. The results imply that the memory consumption of routers is stable regardless of the route length, even when the large-scale routes are advertised. For advertisement of 800,000 routes, i.e., full route, the router running APVAS consumes about 8.6 GB at AS_PATH length four. Moreover, by considering a conventional high-end router, APVAS is operable up to 200,000 routes.

Finally, the experimental results of the computational cost for APVAS are shown in Table 7. According to the table, APVAS takes about nine hours on a router with AS_PATH length 7 to advertise and verify the full route information.

7. Discussion

In this section, we discuss the experimental results in the previous section and the dependency of APVAS for the AS_PATH length. We then evaluate the computational cost of signatures for each router.

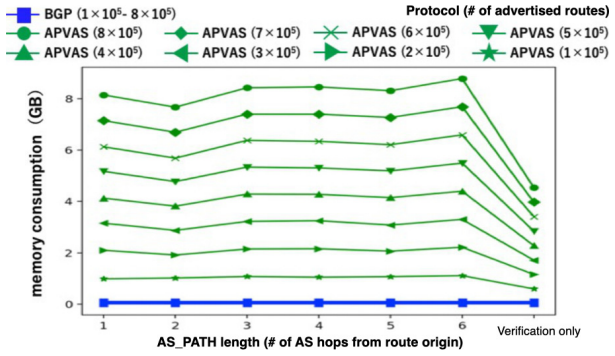

Fig. 6 The memory consumption of the routers running BGP and APVAS when each number of routes was advertised: The horizontal axis shows the length of `AS_PATH` of each router, i.e., the number of AS hops from route origin, i.e., AS65001, and the vertical axis shows the memory consumption of each router. The scale of the graph is overlarge for BGP, and thus all patterns overlap as close to zero as possible. The last router expressed as "Verification only" with `AS_PATH` length of 7 only verifies the signature related to the received route and need not generate a new signature to guarantee the validity of its routing information because of the non-existence of backward peer. Therefore the last router does not need the temporary memory required to generate the signature and needs only the memory related to the signature verification.

Table 7 The computational cost of APVAS.

	rt1	rt2	rt3	rt4	rt5	rt6	rt7	rt8
100K	0:24	0:25	0:34	0:45	0:48	0:56	1:05	1:05
200K	1:09	1:14	1:15	1:52	1:52	1:58	2:27	2:27
300K	2:17	2:19	2:22	2:51	2:55	3:01	3:34	3:34
400K	2:44	2:45	2:54	3:37	3:44	3:59	4:39	4:39
500K	3:20	3:23	3:29	4:22	4:33	4:56	5:44	5:45
600K	3:46	3:48	4:01	5:07	5:21	5:51	6:50	6:51
700K	4:28	4:31	4:43	5:50	6:11	6:47	7:56	7:57
800K	4:52	4:56	5:17	6:33	7:03	7:43	9:02	9:03

Each column indicates the number of routers running APVAS. For instance, a router of `rt1` advertises paths and other routers following `rt1` receives and verifies them. The data in the table indicates the elapsed time from an advertisement of the first route by `rt1`, and the unit means the value of [hour: minute]. Likewise, the rt1 column shows the elapsed time for advertising each number of routes, and other columns, i.e., rt2 to rt8, show the elapsed time until each router receives and verifies them.

7.1 Consideration Based on Experimental Results

According to the empirical study of Wang et al. [51], the average length of `AS_PATH` on the Internet is about 3.9, and the longest path is about 20. When the average length of `AS_PATH` is four, the memory requirement of `BGPsec_PATH` attributes by APVAS became 20% compared to the conventional BGPsec. The bulk of memory consumption for routers is related to the memory requirement of `BGPsec_PATH` attributes [7], and thus the results above confirm that APVAS can reduce the memory requirement by 20%. By taking the longest `AS_PATH`, i.e., 20, into account, in proportion to the length of `AS_PATH`, the difference in the memory consumption between routers running APVAS and those running the conventional BGPsec becomes large. The reason is that the memory requirement of APVAS is stable by virtue of the aggregate signatures via **Algorithm** 4 even if different `BGPsec_PATH` attributes appear. As shown in Table 1, the memory consumption of routers running BGPsec is expected to increase in the future, and then APVAS will have a more attractive advantage as a practical solution than the conventional BGPsec by the stability of APVAS.

We now discuss the memory consumption, including the cache of routers. The memory consumption of the entire routers, i.e., the result of `docker stats`, is 3.47 MB for BGPsec and 4.07 MB for APVAS in the setting of 200 route advertisements and `AS_PATH` length 4. The reason is that parings for bimodal aggregate signatures consume much memory as the initial setting compared to ECDSA of BGPsec. Paring does not affect the memory consumption of the routing table. However, the memory consumption for each router in this experiment is measured with the `docker stats` command, and then the pairing computation has a significant effect when the number of route advertisements is small. However, as shown in Fig. 5, APVAS has a 20% lower memory requirement for the path validation than BGPsec, indicating that APVAS outperforms BGPsec as the number of advertised routes increases, i.e., the full routes.

We now discuss the feasibility of APVAS from the memory consumption. For advertising 800,000 routes in Sect. 6, a router running APVAS consumes about 8 GB of memory. Meanwhile, APVAS can further reduce the memory consumption of routers potentially if our prototype implementation is optimized. In particular, as shown in Fig. 6, the last router in the linear network, i.e., at `AS_PATH` 7, has a connection in one direction only, so it only receives routes advertised from the forward and has no backward peer to send the routes it has received. That means the router with `AS_PATH` length of 7 only verifies the signature related to the received route and need not generate a new signature to guarantee the validity of its routing information because of the non-existence of backward peer. Therefore the last router does not need the temporary memory required to generate the signature and needs only the memory related to the signature verification on the Docker container. The memory consumption of the router was about 4.4 GB, which is half of the total memory consumption. It means that about 4 GB was consumed as a temporal memory for the signature verification. Consequently, the memory consumption of routers running APVAS can become 50% lower than the current implementation.

7.2 Dependency between Memory Consumption of Router and `AS_PATH` Length

We discuss dependencies between memory consumption and the length of `AS_PATH` for APVAS. The size of the BGP update message is 4,096 bytes, and multiple routes are transmitted within this range. In APVAS, the number of signatures is independent of the length `AS_PATH` by aggregating the signatures, but the number of SKIs sent in the update message depends on the length of `AS_PATH`. Therefore, the number of routes transmitted in a single update message varies in each AS. Thus, the memory consumption for each router depends

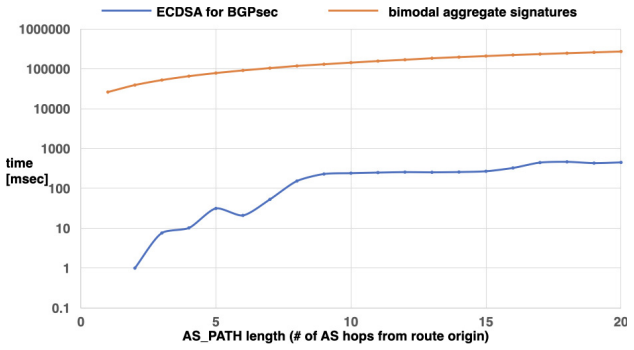

Fig. 7 The computational cost for BGPsec and APVAS.

a little on the AS_PATH length due to a temporary memory related to the update message.

7.3 Computational Cost of Signature Schemes

We evaluate the computational cost for the bimodal aggregate signatures in APVAS compared to ECDSA used in the conventional BGPsec. In particular, we measured the running time of the `decode_bgpsec_attr()` function that verifies signatures in the BGPsec-enabled Bird Routing Daemon and that of the prototype1 in the same experiment setting as the first experiment in Sect. 6.2. On evaluation of the running time of the `decode_bgpsec_attr()` function itself, only the signature verification are measured. In other words, the running time of other processes, e.g., the route advertisement, can be ignored. We measure the running time five times and compute the average. Figure 7 shows results in the computational cost for verification of ECDSA in BGPsec and **Algorithm 5** in APVAS. The computational cost of APVAS is about 3,670 times the computational time of ECDSA in BGPsec. The sizeable computational cost is due to the high computational cost of pairings used by bimodal aggregate signatures.

We then discuss the feasibility of APVAS. The above expensive cost is an important issue that needs to be addressed, and we plan to reduce the computational cost in the future. However, considering a realistic scenario, it is expected that APVAS is feasible in the real world because of the following two reasons.

First, regarding the route advertisement in APVAS takes five minutes as described in Sect. 6.2, it is limited to when the length of AS_PATH is 20, which is the maximum length. Indeed, the number of such lengths of AS_PATH is quite few [51]. As described in Sect. 7.1, the average length of AS_PATH is about 3.9, and then the time for the convergence in APVAS is 83.8 seconds, according to Table 6. Compared to a few seconds in BGPsec, the computational time of APVAS is at most 30 times BGPSec. Moreover, 99% of ASes can be reached when the length of AS_PATH is seven in the real world [52]. In that case, the time for the convergence is 124.6 seconds, according to Table 6, which is about 40 times BGPsec. Namely, APVAS can perform route advertisements and their convergence with a computational cost of 30 to 40 times BGPsec.

Second, as for the convergence with full routes taking nine hours as described in Sect. 6.3, we believe this is unlikely to happen. The advertisement of full routes is mainly considered when new AS providers appear. In practice, however, they often receive routes from an Internet Exchange Point (IXP), which operates and manages a physical infrastructure supporting public and private Internet interconnection in their region. When we checked the routing information of JPIX (https://www.jpix.ad.jp/en/) as an IXP of our region at 09:21 on 17 August 2022, there were about 3,500 routes. APVAS can process their route advertisements in 145 seconds. Since even the conventional BGP may take up to thirty minutes for convergence [53] due to forwarding loops [54], the performance of APVAS is comparable in a realistic situation. Likewise, RIPE†, which has the most significant number of routes, has about 100,000 routes at 09:15 on 17 August 2022. They can be processed in about one hour. Considering we may need that thirty minutes for the convergence of the conventional BGP as described above, we believe that APVAS is reasonably practical because it can process a region's paths in one hour with security guarantees.

Table 8 Comparison of evaluation terms with existing works.

Protocol	Memory Consumption	Prototype Implementation	Signature Aggregation
BGPsec [6]	✓	✓	
ABA [13]			✓
APAT [14]			✓
APVAS	✓	✓	✓

Each column represents that the paper of the protocol discusses the corresponding term.

7.4 Comparison with Other BGPsec-Based Protocols

We discuss evaluation terms of APVAS in a qualitative way in comparison with the existing protocols [13], [14] based on aggregate signatures [11]. The results are shown in Table 8. APVAS is the first work that evaluates the memory consumption based on the first prototype implementation. To the best of our knowledge, a prototype implementation of software routers has been presented for only the conventional BGPsec. However, the conventional BGPsec does not provide the signature aggregation because it is based on ECDSA. Thus, APVAS is the only aggregate signature-based work that evaluates the memory consumption via a prototype.

8. Conclusion

In this paper, we proposed APVAS, a path validation method that deploys novel bimodal aggregate signatures to reduce the memory consumption of routers running BGPsec. We implemented a prototype of APVAS via BIRD and measured the memory consumption of actual routers. Our experimental results confirm that APVAS can reduce the memory requirement of path validation by 20% compared to conventional BGPsec. We also confirm the effectiveness of APVAS

†https://www.ripe.net/

in the real world by using 800,000 routes, which are equivalent to the full route information on a global scale.

Furthermore, we discovered a new problem where the throughput of routers is downgraded by the computational cost of bimodal aggregate signatures, which is heavier than that of ECDSA. Thus, further studies on reducing the memory consumption and the computational cost, which takes misconfiguration such that a large amount of routing information is advertised into account, will need to be undertaken.

9. Code Availability

Our implementation of APVAS is publicly available via GitHub (https://github.com/fseclab-osaka/apvas) for reproducibility and as reference for future works.

Acknowledgments

This research was in part conducted under a contract of "Research and development on new generation cryptography for secure wireless communication services" among "Research and Development for Expansion of Radio Wave Resources (JPJ000254)", which was supported by the Ministry of Internal Affairs and Communications, Japan, and "Innovation Platform for Society 5.0" from Japan Ministry of Education, Culture, Sports, Science, and Technology (Code: S004541) and by JSPS KAKENHI Grant Number 22H03591. We would also like to appreciate the anonymous reviewers for their valuable comments.

References

[1] O. Junjie, N. Yanai, T. Takemura, M. Okada, S. Okamura, and J.P. Cruz, "APVAS: Reducing memory size of AS_PATH validation by using aggregate signatures," CoRR, vol.abs/2008.13346, 2020.
[2] Y. Rekhter, S. Hares, and T. Li, "A border gateway Protocol 4 (BGP-4)," RFC 4271, 2006.
[3] P. Vervier, O. Thonnard, and M. Dacier, "Mind your blocks: On the stealthiness of malicious BGP hijacks," Proc. NDSS 2015, pp.1–15, Internet Society, 2015.
[4] M. Lepinski and S. Kent, "An infrastructure to support secure Internet routing," Request for Comments, RFC 6480, 2012.
[5] G. Huston and G.G. Michaelson, "Validation of route origination using the resource certificate public key infrastructure (PKI) and route origin authorizations (ROAs)," RFC 6483, 2012.
[6] M. Lepinski and K. Sriram, "BGPsec protocol specification," RFC 8205, 2017.
[7] K. Sriram, "RIB size estimation for BGPSEC," 2011. https://www.nist.gov/document-7096
[8] M. Apostolaki, A. Zohar, and L. Vanbever, "Hijacking bitcoin: Routing attacks on cryptocurrencies," Proc. IEEE S&P 2017, pp.375–392, 2017.
[9] P. Ekparinya, V. Gramoli, and G. Jourjon, "The attack of the clones against proof-of-authority," Proc. NDSS 2020, pp.1–14, Internet Society, 2020.
[10] H. Birge-Lee, L. Wang, J. Rexford, and P. Mittal, "Sico: Surgical interception attacks by manipulating bgp communities," Proc. CCS 2019, pp.431–448, ACM, 2019.
[11] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and verifiably encrypted signatures from bilinear maps," Proc. EUROCRYPT 2003, LNCS, vol.2656, pp.416–432, Springer, 2003.
[12] A. Lysyanskaya, S. Micali, L. Reyzin, and H. Shacham, "Sequential aggregate signatures from trapdoor permutations," Proc. EUROCRYPT 2004, LNCS, vol.3027, pp.74–90, Springer, 2004.
[13] M. Zhao, S.W. Smith, and D.M. Nicol, "Aggregated path authentication for efficient BGP security," Proc. CCS 2005, pp.128–138, ACM, 2005.
[14] K. Tanaka, N. Yanai, M. Okada, T. Nishide, and E. Okamoto, "APAT: An application of aggregate signatures to BGPSEC," Fast Abstract in DSN 2016, 2016.
[15] T. Takemura, N. Yanai, N. Umeda, M. Okada, S. Okamura, and J.P. Cruz, "APVAS+: A practical extension of BGPsec with low memory requirement," Proc. ICC 2021, pp.1–8, IEEE, 2021.
[16] S. Kent, C. Lynn, and K. Seo, "Secure border gateway protocol (S-BGP)," IEEE J. Sel. Areas Commun., vol.18, no.4, pp.582–592, 2000.
[17] R.R. Sambasivan, D. Tran-Lam, A. Akella, and P. Steenkiste, "Bootstrapping evolvability for inter-domain routing with d-bgp," Proc. SIGCOMM 2017, pp.474–487, ACM, 2017.
[18] S. Pouryousef, L. Gao, and A. Venkataramani, "Towards logically centralized interdomain routing," Proc. NSDI 2020, pp.739–757, USENIX Association, 2020.
[19] N. Umeda, N. Yanai, T. Takemura, M. Okada, J.P. Cruz, and S. Okamura, "SQUAB: A virtualized infrastructure for experiments on BGP and its extensions," Proc. AINA 2021, LNNS, vol.225, pp.600–613, Springer, 2021.
[20] M. Brandt and H. Shulman, "Optimized BGP simulator for evaluation of internet hijacks," Proc. IEEE INFOCOM WKSHPS 2021, pp.1–2, IEEE, 2021.
[21] K. Sriram and D.C. Montgomery, "Resilient interdomain traffic exchange: BGP security and DDos mitigation," NIST Report, 2019.
[22] J.M. Smith, K. Birkeland, T. McDaniel, and M. Schuchard, "Withdrawing the BGP re-routing curtain: Understanding the security impact of bgp poisoning through real-world measurements," Proc. NDSS 2020, pp.1–18, Internet Society, 2020.
[23] R. Morillo, J. Furuness, C. Morris, J. Breslin, A. Herzberg, and B. Wang, "ROV++: Improved deployable defense against BGP hijacking," Proc. NDSS 2021, Internet Society, 2021.
[24] R. Lychev, S. Goldberg, and M. Schapira, "BGP security in partial deployment: Is the juice worth the squeeze?," SIGCOMM Computer Communication Review, vol.43, no.4, p.171–182, 2013.
[25] Y. Yang, X. Shi, Q. Ma, Y. Li, X. Yin, and Z. Wang, "Path stability in partially deployed secure bgp routing," Computer Networks, vol.206, p.108762, 2022.
[26] S. Goldberg, "Why is it taking so long to secure internet routing?," Queue, vol.12, no.8, p.20–33, 2014.
[27] R. Lychev, M. Schapira, and S. Goldberg, "Rethinking security for internet routing," Commun. ACM, vol.59, no.10, pp.48–57, 2016.
[28] P. Sermpezis, V. Kotronis, P. Gigis, X. Dimitropoulos, D. Cicalese, A. King, and A. Dainotti, "ARTEMIS: Neutralizing BGP hijacking within a minute," IEEE/ACM Trans. Netw., vol.26, no.6, pp.2471–2486, 2018.
[29] C. Gentry and Z. Ramzan, "Identity-based aggregate signatures," Proc. PKC 2006, LNCS, vol.3958, pp.257–273, Springer, 2006.
[30] S. Hohenberger, A. Sahai, and B. Waters, "Full domain hash from (leveled) multilinear maps and identity-based aggregate signatures," Proc. CRYPTO 2013, LNCS, vol.8042, pp.494–512, Springer, 2013.
[31] S. Hohenberger, V. Koppula, and B. Waters, "Universal signature aggregators," Proc. EUROCRYPT 2015, LNCS, vol.9057, pp.3–34, Springer, 2015.
[32] B. Liang, H. Li, and J. Chang, "The generic transformation from standard signatures to identity-based aggregate signatures," Proc. ISC 2015, LNCS, vol.9290, pp.21–41, Springer, 2015.
[33] J.H. Ahn, M. Green, and S. Hohenberger, "Synchronized aggregate signatures: New definitions, constructions and applications," Proc. CCS 2010, pp.473–484, ACM, 2010.
[34] S. Hohenberger and B. Waters, "Synchronized aggregate signatures from the RSA assumption," Proc. EUROCRYPT 2018, LNCS,

vol.10821, pp.197–229, Springer, 2018.
[35] K. Takemure, Y. Sakai, B. Santoso, G. Hanaoka, and K. Ohta, "Achieving pairing-free aggregate signatures using pre-communication between signers," IEICE Trans Fundamentals, vol.E104-A, no.9, pp.1188–1205, Sept. 2021.
[36] Y. Yao, Z. Li, and H. Guo, "A unified framework of identity-based sequential aggregate signatures from 2-level hibe schemes," Information Sciences, vol.516, pp.505–514, 2020.
[37] C. Gentry, A. O'Neill, and L. Reyzin, "A unified framework for trapdoor-permutation-based sequential aggregate signatures," Proc. PKC 2018, LNCS, vol.10770, pp.34–57, Springer, 2018.
[38] A. Boldyreva, C. Gentry, A. O'Neill, and D. Yum, "Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing (extended abstract)," 2010.
[39] M. Fischlin, A. Lehmann, and D. Schröder, "History-free sequential aggregate signatures," Proc. SCN 2012, LNCS, vol.7485, pp.113–130, Springer, 2012.
[40] K. Brogle, S. Goldberg, and L. Reyzin, "Sequential aggregate signatures with lazy verification from trapdoor permutations - (extended abstract)," Proc. ASIACRYPT 2012, LNCS, vol.7658, pp.644–662, Springer, 2012.
[41] N. Yanai, M. Mambo, K. Tanaka, T. Nishide, and E. Okamoto, "Another look at aggregate signatures: Their capability and security on network graphs," Proc. INTRUST 2015, LNCS, vol.9565, pp.32–48, Springer, 2015.
[42] G. Hartung, B. Kaidel, A. Koch, J. Koch, and A. Rupp, "Fault-tolerant aggregate signatures," Proc. PKC 2016, LNCS, vol.9614, pp.331–356, Springer, 2016.
[43] R. Ishii, K. Yamashita, Y. Sakai, T. Matsuda, T. Teruya, G. Hanaoka, K. Matsuura, and T. Matsumoto, "Aggregate signature with traceability of devices dynamically generating invalid signatures," Proc. of ACNSW, LNCS, vol.12809, pp.378–396, Springer, 2021.
[44] J.S. Coron and D. Naccache, "Boneh et al.'s k-element aggregate extraction assumption is equivalent to the diffie-hellman assumption," Proc. ASIACRYPT 2003, LNCS, vol.2894, pp.392–397, Springer, 2003.
[45] A. Boldyreva, "Threshold signatures, multisignatures and blind signatures based on the Gap-Diffie-Hellman-group signature scheme," Proc. PKC 2003, LNCS, vol.2567, pp.31–46, Springer, 2003.
[46] S. Lu, R. Ostrovsky, A. Sahai, H. Shacham, and B. Waters, "Sequential aggregate signatures and multisignatures without random oracle," Proc. EUROCRYPT 2006, LNCS, vol.4004, pp.465–485, Springer, 2006.
[47] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," Proc. CCS 1993, pp.62–73, ACM, 1993.
[48] "Bird bgpsec," http://www.securerouting.net/tools/bird/
[49] T. Chung, E. Aben, T. Bruijnzeels, B. Chandrasekaran, D. Choffnes, D. Levin, B.M. Maggs, A. Mislove, R.V. Rijswijk-Deij, J. Rula, and N. Sullivan, "RPKI is coming of age: A longitudinal study of RPKI deployment and invalid route origins," Proc. IMC 2019, pp.406–419, ACM, 2019.
[50] T. Hlavacek, I. Cunha, Y. Gilad, A. Herzberg, E. Katz-Bassett, M. Schapira, and H. Shulman, "DISCO: Sidestepping RPKI's deployment barriers," Proc. NDSS 2020, Internet Society, 2020.
[51] C. Wang, Z. Li, X. Huang, and P. Zhang, "Inferring the average as path length of the internet," Proc. IC-NIDC, pp.391–395, IEEE, 2016.
[52] M. Okada, Y. Katsuno, A. Kanaoka, and E. Okamoto, "32-bit as number based IP traceback," Proc. IMIS 2011, pp.628–633, IEEE, 2011.
[53] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian, "Delayed internet routing convergence," Proc. SIGCOMM 2000, pp.175–187, ACM, 2000.
[54] R.B. da Silva and E. Souza Mota, "A survey on approaches to reduce BGP interdomain routing convergence delay on the internet," IEEE Commun. Surveys Tuts., vol.19, no.4, pp.2949–2984, 2017.

Ouyang Junjie received B.Eng. degree in Engineering Science from Osaka University, Japan, in 2017, and M.S.Eng. from Osaka University, Japan, in 2019. His research interests include information security.

Naoto Yanai received the B.Eng. degree from The National Institution of Academic Degrees and University Evaluation, Japan, in 2009, the M.S. Eng. from the Graduate School of Systems and Information Engineering, the University of Tsukuba, Japan, in 2011, and the Dr.E. degree from the Graduate School of Systems and Information Engineering, the University of Tsukuba, Japan, in 2014. He had been an assistant professor at Osaka University, Japan, from 2014 to 2021 and is an associate professor at Osaka University. His research area is information security.

Tatsuya Takemura received the B.Eng. degree in Engineering Science from Osaka University, Japan, in 2019. He has recently joined the M.S. course in the Graduate School of Information Science and Technology in Osaka University, Japan. His research interests include machine learning and network security.

Masayuki Okada is a full professor at The University of Nagasaki. He experienced a BGP operation of an academic network since 2000. He joined JPNIC in 2004 and is responsible for the development and operation of the IP resource management system related to routing and JPIRR research, as well as the use of IRR. He received his Ph.D. degree in 2012 in Computer Science from University of Tsukuba.

Shingo Okamura received his B.E., M.E., and Ph.D. degrees in information science and technology from Osaka University in 2000, 2002, and 2005, respectively. Since 2005, he has worked for Osaka University. In 2008, he joined National Institute of Technology, Nara College. Currently, he is an associate professor at the college. His research interests include cryptographic protocols and cyber security. He is a member of IEICE, IPSJ, IEEJ, ACM, IEEE, and IACR.

Jason Paul Cruz received his B.S. degree in Electronics and Communications Engineering and M.S. degree in Electronics Engineering from the Ateneo de Manila University, Quezon City, Philippines, in 2009 and 2011, respectively, and his Ph.D. degree in Engineering from the Graduate School of Information Science, Nara Institute of Science and Technology, Nara, Japan in 2017. He is currently a Specially Appointed Assistant Professor at Osaka University, Osaka, Japan. His current research interests include role-based access control, blockchain technology, hash functions and algorithms, privacy-preserving cryptography, and Android programming.