Integrating BGPsec Validation with the RTRlib and Software Routers

Colin Sames

Grundprojekt

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Colin Sames

Integrating BGPsec Validation with the RTRlib and Software Routers

Integrating BGPsec Validation with the RTRlib and Software Routers eingereicht im Rahmen des Grundprojekt

im Studiengang Master of Science Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Thomas C. Schmidt

Eingereicht am: July 11, 2018

Contents

1 Introduction

The Internet is not a single network where everyone is directly connected with each other, but instead is a large network of smaller networks. These networks are called *Autonomous Systems (ASes)*. All devices that are connected to the Internet are provided with an IP address. This IP address is part of an IP address range, called *IP prefix*. Such prefixes are maintained by ASes, where each unique prefix is maintained by one AS only. If an Internet user connects to a certain Internet service, the data traffic between client and server is forwarded from AS to AS until it reaches its destination. To forward data traffic, an AS needs to know where the data traffic should be forwarded to next. To achieve this, dedicated routers within an AS implement the *Border Gateway Protocol (BGP)* [1]. With this protocol, routers share updated and withdrawn IP prefixes with other autonomous systems. Since two ASes are not necessarily directly connected to each other, they depend on intermediate ASes to forward the announcements to ASes they otherwise would not be able to reach. Such announcements contain information about *which AS maintains a certain IP prefix (origin)*, and *how to reach it (AS path)*. Altering any of these information, be it origin or AS path, may lead to wrongly forwarded traffic. Although altering the information is not always of malicious intent, it still causes undesirable effects. The results are unreachable Internet services, online fraud or espionage.

BGP itself provides no security mechanisms to protect the contents of announcements. Therefore, external protection mechanisms have been designed. Today, the deployed countermeasures to route manipulation only cover the prefix and its origin but not the AS path. Verification of the ownership is realized by the *Resource Public Key Infrastructure (RPKI)* [2]. It provides a way for ASes to verify that the prefix within an announcement indeed does belong to the AS that originates the announcement. The client part of this RPKI feature called *route origin validation* is implemented by the RTRlib [3][1].

With the origin being protected, the AS path is still vulnerable to manipulation. The solution to secure the AS path is BGPsec [4], an extension to BGP which adds cryptographic validation to the path attribute. The contents of an announcement that are to be protected are hashed and then signed. If any of the protected information is changed, the hash changes accordingly

[1] http://rtrlib.realmv6.org/

and validating the signature fails. Each AS that receives a BGPsec announcement can validate that the AS path information was not altered by any AS along the forwarding chain. If route origin validation and AS path validation are combined, route manipulation can be prevented.

This work introduces a concept to integrate BGPsec AS path validation into RTRlib, to extend its current RPKI features. Additionally, considerations are made regarding offering this functionality to software routers such as the FRRouting suite (FRR)[2], which handles routing interior and exterior routing of an AS. The result is an API that is designed to be lightweight and intuitive to use.

Currently, there are only few BGPsec implementations that cover AS path validation [5]. While such implementations offer the whole set of features that are specified by the RFC, our BGPsec implementation for RTRlib only provides the features that are necessary to validate and sign BGPsec AS paths. The library also aims to stay independent from any routing suites, so no operational features of BGPsec should be provided to prevent specialization with a certain software.

The remainder of this report is structured as follows. A brief introduction to BGP as well as its attack surface is given in Section 2. In Section 3, the current methods for securing BGP contents are presented. Further, a concept for extending RTRlib with BGPsec functionality is presented in Section 4. This work concludes in Section 5.

[2]`https://frrouting.org/`

2 The Border Gateway Protocol

Networks across the Internet that use BGP are called *Autonomous Systems (AS)*, since they autonomously manage their assigned Internet resources, such as IP address space, also called *IP prefix*. These autonomous systems exchange routing information with each other via BGP. BGP in its current state was standardized by the *Internet Engineering Task Force (IETF)*[1] in RFC 4271 [1]. In this section, a brief summary of the most important aspects of BGP are presented. Additionally, the attack vector of BGP is examined.

An AS consists of one or more networks that are governed by a single institution and controls routers that are dedicated to running BGP. An AS operates the IP prefixes that are assigned to it by itself. The assignment of IP prefixes originates from the *Internet Assigned Numbers Authority (IANA)*[2]. This institution passes IP prefixes down to one of five *Regional Internet Registries (RIRs)* [6] and from there they are distributed to ASes all over the world. Each RIR covers a certain region on the globe, e.g., the European region falls under the responsibility of RIPE NCC[3]. In addition to IP prefixes, each AS receives a unique *Autonomous System Number (ASN)* [7, 8]. This number is used to identify an AS anywhere on the Internet.

Although BGP is used to exchange routing information, it also resembles availability of routers, reachability of ASes and the willingness to forwarding traffic [9]. Two ASes that are connected to each other hold a certain relationship. These kind of relationships are classified by the *Gao-Rexford* model [10]. This model suggests that two directly connected ASes have a *provider-customer* or a *peer-to-peer* relationship [11–13]. In the first case, an AS (the customer) pays another AS (the provider) for transit service, so that the customer is enabled connectivity to the rest of the Internet. In the latter case, two ASes are treating each other as equals and exchange traffic, usually without money involved. This model holds true for most of the time, but since internal policies are still up to the provider, some ASes might stray from this rule.

[1]https://www.ietf.org/
[2]https://www.iana.org/
[3]https://www.ripe.net/

2.1 Protocol Description

Each BGP message consists of a header and a message content. There are four types of BGP messages: *open*, *update*, *keepalive* and *notification* [1, 12]. The open message contains information about configuration and capabilities of the router. The update message contains routing information such as announcements or withdrawals of ownership of IP prefixes. Keepalive messages are used to maintain a connection with a peering router. The notification message informs a peering router about any errors that occur during a session.

Before proceeding with the protocol procedure, the update message is described in more detail. The update message contains the propagated routing information such as the *prefix(es)*[4] and the *AS path*. The announced prefix covers the IP prefix of the *origin AS*. The AS path is a sequence of ASNs that indicates, along which path the update message was forwarded through the Internet. This way an AS can tell, how to reach the announced prefix. The origin AS, i.e., the holder of the announced IP prefix, can always be found at the rightmost position of the AS path. In literature, the AS path is often written as *(65522, 65511)*, with AS 65511 being the origin AS. The AS path length is the amount of elements in the path, which is 2 in this case. Figure 2.1 shows, how an update message looks like.

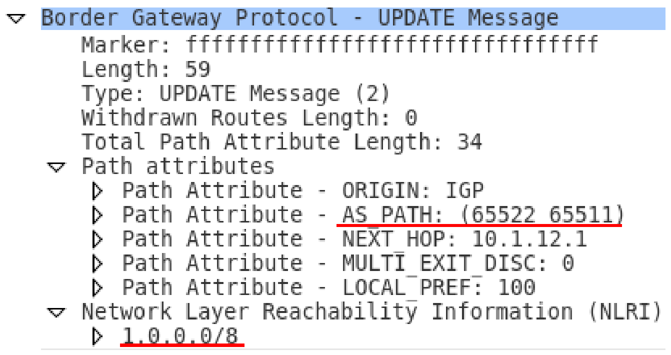

```
▽ Border Gateway Protocol - UPDATE Message
    Marker: ffffffffffffffffffffffffffffffff
    Length: 59
    Type: UPDATE Message (2)
    Withdrawn Routes Length: 0
    Total Path Attribute Length: 34
  ▽ Path attributes
    ▷ Path Attribute - ORIGIN: IGP
    ▷ Path Attribute - AS_PATH: (65522 65511)
    ▷ Path Attribute - NEXT_HOP: 10.1.12.1
    ▷ Path Attribute - MULTI_EXIT_DISC: 0
    ▷ Path Attribute - LOCAL_PREF: 100
  ▽ Network Layer Reachability Information (NLRI)
    ▷ 1.0.0.0/8
```

Figure 2.1: A BGP update message. The AS path and the prefix information are underlined.[5]

After speaker and receiver have established a TCP connection, they both send an open message. In case both routers agree with the contents of the open message, a keepalive is sent. If at any time during the session a router detects an error or incompatibility, it may send a notification message with the occurred error and then close the connection. As long as no errors occur, both routers may send update messages which contain the routing information.

[4]We will further only use the singular form *prefix* and imply that the usage of multiple prefixes is also possible
[5]Screenshot taken from https://www.cloudshark.org/captures/0216380f8421

To terminate the session, a router may send a notification message with an appropriate error code and then close the TCP connection [1, 12]. As an example, Figure 2.2 shows the message sequence of two routers.

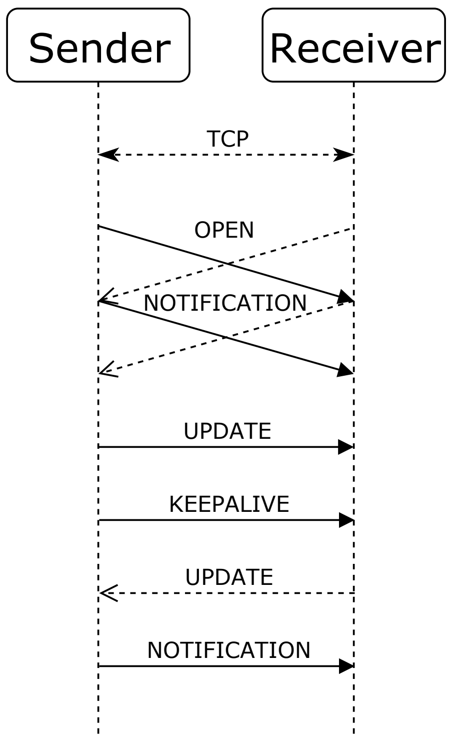


Figure 2.2: After a TCP connection is established, both routers send an open message. If they agree with the configurations, they send a keepalive message. They now send updates as they please. To keep the connection up, they send keepalive messages. Before closing the connection, a notification message is sent.

When a speaker receives an update message, various conditions are checked before the contained new route is added to the routers routing table [12]. At first, policy filters are applied to the route. If the route is not filtered, the router will check, whether or not the received route to the announced prefix is better than the currently held route. Therefore, multiple conditions such as lowest AS path length are evaluated. If the new route is considered better than the old one, the old route is replaced.

Before a router forwards or sends the BGP update, it has to prepend its own ASN to the AS path within the message. The own ASN is always prepended to the leftmost position of the AS path, as shown in Figure 2.3.

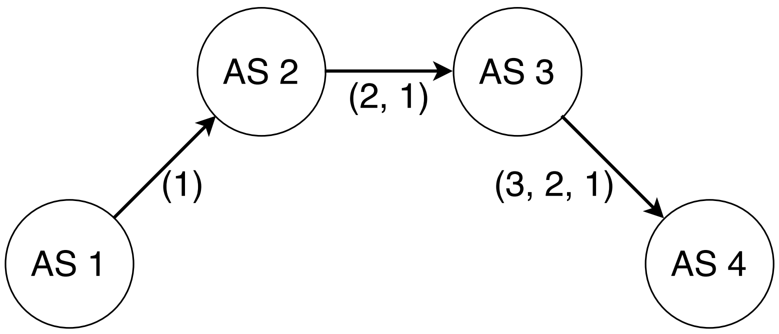


Figure 2.3: AS 1 sends a BGP update to AS 2. AS 2 prepends its own AS number to the AS path and forwards it to AS 3.

2.2 BGP Attack Surface

The Border Gateway Protocol is, by design, not robust to attacks. It is very prone to misconfiguration and the transported information can be manipulated without much effort. This section covers the most common attacks that can be launched against BGP.

2.2.1 BGP Update Manipulation

To manipulate Internet routing, three common techniques exist [14]. Depending on the technique, a new BGP update needs to be forged and propagated, or a received update is manipulated and then forwarded to other ASes. In the following examples, *AS X* is an AS that tries to *hijack* the traffic of *AS 1*. Hijacking the traffic of AS 1 means that AS X tries to route the traffic of AS 1 via its own AS.

The first way of manipulation is to claim ownership of an IP prefix, even though this prefix belongs to another AS. Therefore, AS X has to propagate an update that announces the IP prefix of AS 1 with AS X as origin AS. Without external mechanisms, other routers have no way of telling that the announced prefix (controlled by AS 1) does not actually belong to the originator of the update (AS X). Because of this, a forged announcement might be accepted by a receiving router.

The second way to manipulate Internet routing is by announcing a prefix that is more specific than the prefix of another AS. As an example, the prefix *1.2.3.0/25* is more specific than the prefix *1.2.3.0/24*. BGP routers are configured to accept announcements which contain IP prefixes that are more specific than the prefixes that are currently held within their routing table. Assume that AS 1 controls the prefix *1.2.3.0/24* and the routing table of AS 2 contains that prefix. AS X now tries to hijack the traffic of AS 1 by forging an update that contains the prefix *1.2.3.0/25* and AS X as originator. If AS 2 now receives the bogus announcement of AS X, it determines that the announced prefix is more specific than the current entry. AS 2 overwrites the entry of AS 1 with the new one, even though the propagated prefix does not belong to AS X.

Routers are also configured to prefer shorter routes over longer routes. Thus, the AS path is target to manipulation as well [13]. This leads to the third technique. If the AS path within an announcement is artificially elongated, it might be rejected by a router that holds an entry with a shorter path to the announced prefix. An AS path may also be manipulated to suggest routing via the own AS, or making it more attractive to other routers by shortening it. Assume that AS 5 contains the entry with the AS path (4,3,2,1) for the prefix of AS 1. AS X can try to manipulate the forwarded traffic to AS 1 by claiming that it knows a shorter path to the victim. Therefore, AS X forges an update that contains the wrong but shorter AS path (X,1). If AS 5 receives such an update, the route selection algorithm determines that the route via AS X is shorter and replaces the current path with the new path. AS 5 now forwards the traffic destined for AS 1 via AS X.

Any of these methods may be used with malicious intentions, but are also often cause of misconfigured routers. Because of this, directed attacks and innocent misconfigurations are hard to distinguish. Nevertheless, it is possible to cause trouble for other ASes by, e.g., making them unreachable [15, 16]. The next section presents the effects of BGP update manipulation.

2.2.2 Traffic Abuse

When an AS successfully gains control over the prefix of another AS, there are three malicious ways to deal with the hijacked traffic [17].

The first one is *Blackholing*. The attacking AS simply drops all hijacked traffic, instead of forwarding it. As a result, the AS whose prefix has been hijacked is not reachable for potentially a lot of Internet users. This is a Denial of Service (DoS) [18] attack on the victim AS.

The second option of treating the hijacked traffic is *Interception*. When intercepting traffic, the hijacking AS proceeds to forward the traffic to the original destination. This way, the attack is hard to discover since there are hardly any anomalies for the victim to discover. Such an attack must be carefully planned though. The hijacking AS still requires an intact path to the victim AS, otherwise the forwarded traffic would not reach the target. If maintaining a path to the victim is accomplished, the traffic is reaching its target, although it is routed via a different path. This attack can be used to eavesdrop on the traffic of Internet users.

The last of the three options is *Impersonating*. The attacker responds to the traffic that was hijacked. An Internet user that is responded to by the attacker might not notice any difference.

The received responses appear to come from the Internet service within the AS where the requests were initially sent to. This way, an attacker can, e.g., mimic a website and trick users to reveal sensitive personal information such as credit card numbers [14].

Often, the goal of an attacker is to misuse the hijacked traffic by sending spam emails. The email protocol SMTP [19] requires a TCP connection, thus the spammer requires the control over an IP prefix, if only for a short amount of time. A soon as the attacker obtains control, the emails are sent. After sending all emails, the spammer can forget about the hijacked prefix since his goal has been achieved. At some point, the victim AS will regain control over the prefix, yet the emails are already sent.

2.2.3 Real world examples

There are numerous examples of BGP attacks all over the world. It is impossible to create a complete list of all attacks as some go unnoticed or only last for a few minutes [20].

There are some infamous BGP attacks that have been made public. One very well-known attack is the *YouTube* hijack from 2008 [21]. On the 24th of February, the AS *Pakistan Telekom* claimed ownership of an IP prefix belonging to the video hosting service. The announcement spread across the Internet and *YouTube* was left unaccessible for about two hours.

Another, more recent case happened on April 26 in 2017. The AS *Rostelecom* announced 50 prefixes of other autonomous systems [22]. Among the victims were ASes like *Visa* and *MasterCard*. The whole incident lasted only a couple of minutes and it is unclear whether or not it was done on purpose.

More well-known incident are the *AS 7007 incident* in 1997 [23], allegedly caused by misconfiguration and *Googles May 2005 Outage* [24].

3 Protection Methods for BGP

The two most important information within a BGP update are the origin AS of a prefix as well as the AS path. Both are not protected by BGP and must therefore be secured with external mechanisms. Such mechanisms exist for both the origin AS and the AS path and are described in this section.

3.1 Route Origin Protection using RPKI

The *Resource Public Key Infrastructure (RPKI)* [2] is dedicated to protecting the origin AS information. RPKI holds cryptographically singed objects that contain three information: a prefix, a maximum prefix length and the AS that owns the prefix, the origin AS. These objects are called *Route Origin Authorizations (ROAs)* [25]. They are cryptographically verifiable and are stored in public repositories. To prevent the effort of validation for the routers, ROAs are pre-validated and then stored in *cache servers*. Communication between router and cache server is established by the *RPKI to Router (RTR)* protocol [26].

ROAs are cryptographic certificates that are created by the corresponding RIR of an AS. Each RIR holds its own root certificate. Figure 3.1 shows the hierarchy after which certificates are handed out. A root certificate is used to sign certificates of ASes within their region. These certificates are the ROAs that hold information about the assigned prefixes, length and origin AS. Note that there are per se no invalid ROAs. All ROAs that are held by the RPKI cache servers are validated by the corresponding RIR.

When a router receives a BGP update, it can validate the contained origin AS and prefix. This is done by comparing the prefix-length-AS triplet within the update with the triplet contained in the ROA. There are three outcomes to this validation [16].

1. The announcement is *valid* if it is covered by at least one ROA
2. The announcement is *invalid* if it is announced by an unauthorized AS (i.e., the AS does not match the ROA entry for the announced prefix) or is more specific (the announced prefix is more specific than the prefix of a matching ROA)
3. The ROA is *not found* and the status therefore unknown

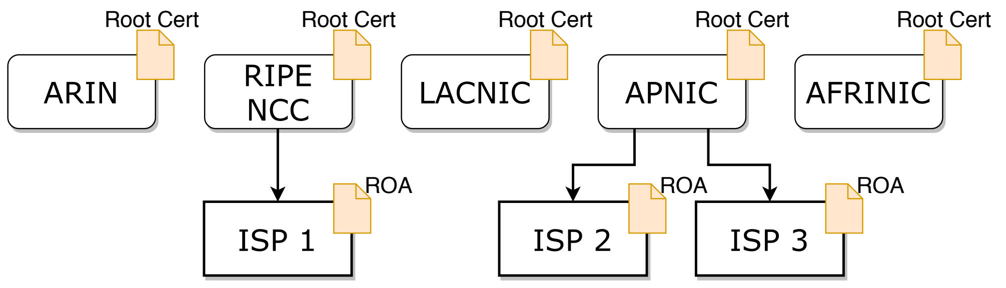


Figure 3.1: The architecture after which certificates are passed down. This structure is similar to the resource allocation hierarchy of IP prefixes.

Based on this validation, the router chooses how to proceed with the BGP update. It is not mandatory to accept a *valid* update, nor is it mandatory to deny an *invalid* update. This is completely up to the ASes routing policy [2, 16, 27]. The process of validation is illustrated in Figure 3.2.

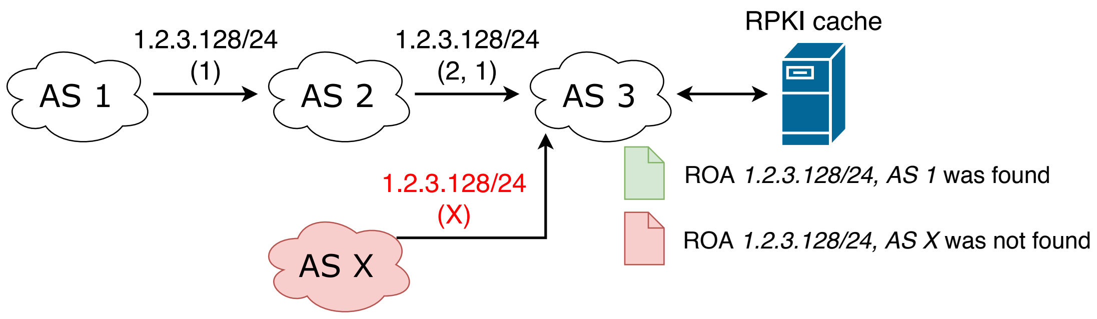


Figure 3.2: AS 1 propagates its routing information. AS X tries to claim the prefix of AS 1 for himself. AS 3 now receives two updates and validates both by requesting their ROA entries from the RPKI. Since there is only a ROA for *(1.2.3.128/24, AS 1)*, the bogus update from AS X must be invalid.

While RPKI helps protecting the origin AS, the AS path is still vulnerable. The following section will introduce a way of securing this attribute.

3.2 BGPsec

BGPsec is designed as an extension for BGP that provides AS path protection. The protection is achieved by cryptographically signing each element of the AS path. The protocol was developed by the IETF, the RFC 8205 [4] was published in 2017. Along with this RFC, additional documents were published that cover, e.g., algorithms, key and signature formats [28] or BGPsec router certificates [29].

The main feature of BGPsec is the replacement of the *AS_PATH* with the *BGPsec_PATH* attribute. While the AS_PATH holds the AS path information of a BGP update, the BGPsec_PATH of a BGPsec update additionally includes cryptographic signatures for each AS path element. BGPsec utilizes public-private-key cryptography [30] to sign and validate the AS path of update messages. The private key is stored and kept secret on the router, while the public key is stored inside the RPKI. BGPsec capable routers are able to validate, whether the AS path of a received update message has been tempered with. The private key is used to sign the contents of a BGPsec update message. Therefore, all information that need to be protected are hashed and the resulting digest is then signed with the private key. The resulting signature is appended to the BGPsec_PATH of the BGPsec update, which can then be send or forwarded to peering routers.

There are two very important cryptographic properties of a signature. One is *authentication*. Authentication proves ownership of the private key. If a signature is created with a certain private key, it can only be successfully validated by the public key that belongs to that private key. Validating a signature with anything other than the appropriate public key will fail. If a signature is successfully validated, it is certain that this signature was created by the one person or device that holds the private key.

A second property of signing is *integrity*. This means that anyone is prevented from tempering with the signed contents. For validating a signature, the original message, in this case the hashed content, is required. If a malicious AS changes any of the protected information, the calculated digest would consequently change as well. The validation of a signature with the wrong digest leads to an invalid result.

Before two routers are able to exchange BGPsec updates, they first need to negotiate BGPsec support. If one of the two participants does not speak BGPsec (or chooses not to), the session may continue, yet only BGP messages are exchanged between them for the rest of the session.

Should both routers agree on BGPsec support, the sending router can begin building the BGPsec update message. It constructs the BGPsec_PATH and inserts its own ASN, the prefix, a *Subject Key Identifier (SKI)* and other information into it. The SKI is used to later identify the public key of a router. A complete list of all information can be found in Section 4.3.2. These information are hashed, signed and then appended to the BGPsec_PATH attribute. The update is then send to the target peer, as shown in Figure 3.3. The detailed contents of the BGPsec_PATH attribute can be seen in Figure 3.4.

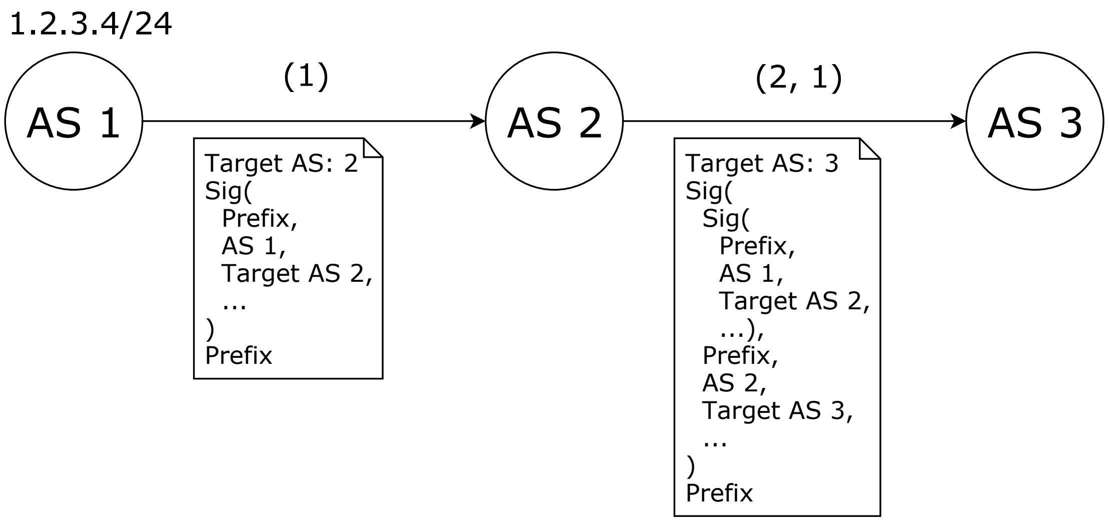


Figure 3.3: AS 1 creates a BGPsec update. Contained is the signature over the prefix, its ASN and other information. AS 2 adds itself to the AS path, updates the target AS field and creates a signature over all previous signatures and its own information.

As soon as the peering router receives the update message, it starts the validation process. Validation begins with the last appended signature. The router takes the SKI that was last appended to the update message and fetches the appropriate public key from the RPKI. The router then calculates the hash with the required information. Using the resulting digest and the public key, the last appended signature can be validated. If the signature is valid, the router continues by repeating the whole procedure with the second to last signature. The router proceeds until every signature inside the update message was validated this way. If a signature is not successfully validated, the process ends prematurely. Should all signatures be *valid*, the update itself is. If there is at least one invalid signature, the update is considered *not valid*. The process of AS path validation is illustrated in Figure 3.5.

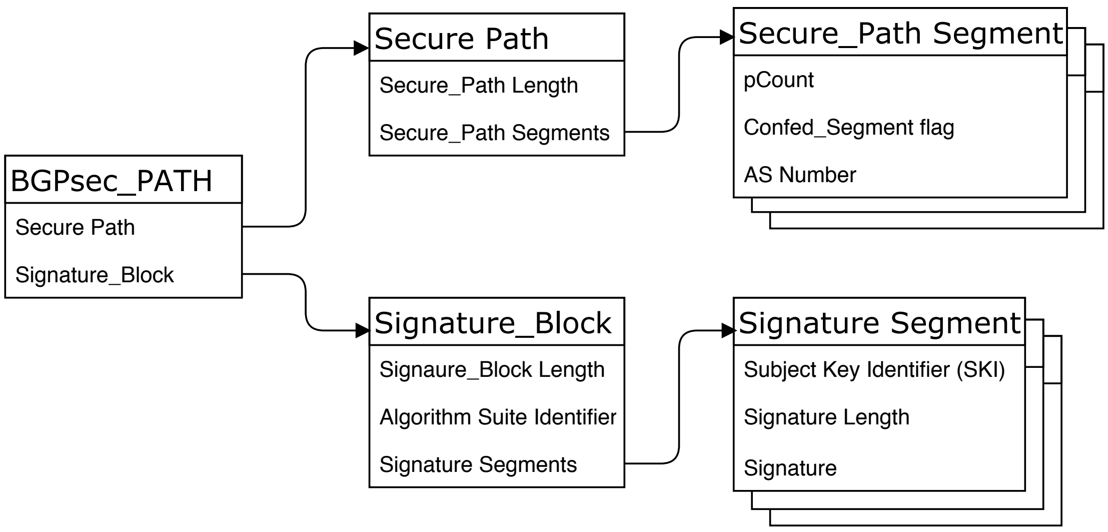


Figure 3.4: Structure of the BGPsec_PATH attribute in an UML like fashion. Both the Secure Path as well as the Signature_Block may contain multiple Secure_Path Segments and Signature Segments.

3.3 Existing Implementations

One library that implements the router part of RPKI features is the RTRlib[1]. It provides two core features:

- Implementation of the client part of the RTR protocol version 1
- Route Origin Validation

The RTR protocol allows for communication between router and cache server. The library initializes and maintains such a connection. If a connection is established, RTRlib extracts the entries of prevalidated ROAs from the cache server and stores them locally. The entries are synchronized with the cache server to keep them up to date. Fetching router keys is a feature of RTR version 1 [31]. When the router receives an update, it can pass the prefix-length-AS triplet to the RTRlib, which will perform route origin validation and then return the result. RTRlib is a C library that provides an API for these functionalities, so any application can include it and make use of RTR as well as route origin validation.

Some tools that are built upon RTRlib are a web browser extension (see Figure 3.6) that allows the user to view the validity status on the currently visited website [32], *rpki-read*, a

[1] https://github.com/rtrlib/rtrlib

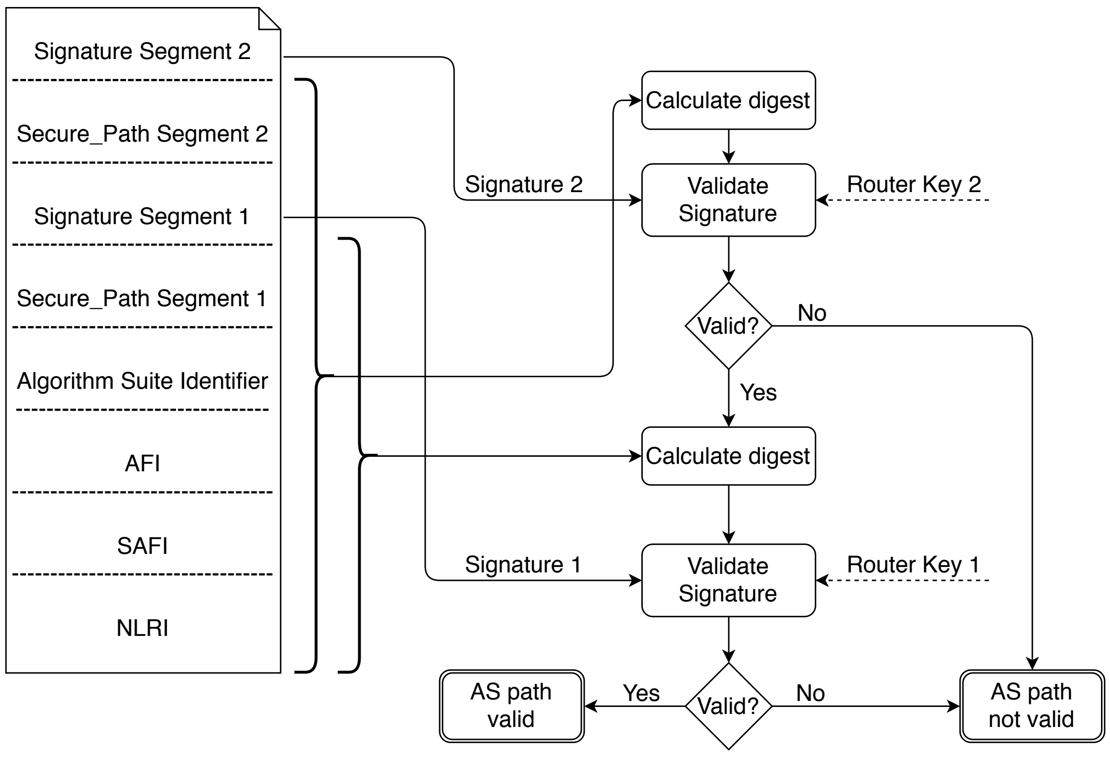


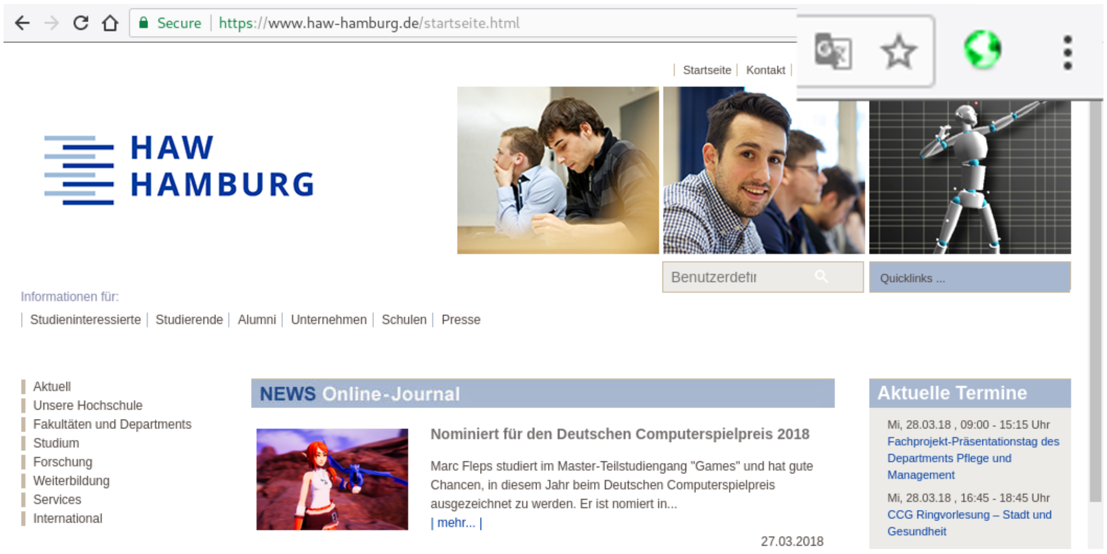
Figure 3.5: On the left is the BGPsec_PATH attribute with the AS path (2, 1). On the right is the AS path validation process of the receiving AS 3. The process is slightly simplified for the sake of illustration.

real-time RPKI dashboard, or *REST BGP Validator (RBV)*, a RESTful interface that offers RPKI validation over a web interface. The code is available in the RTRlib GitHub repository[2].

Besides RTRlib, other software exists that implements the RTR protocol, such as the *RPKI Validator*[3], which is developed by RIPE NCC. Another implementation is provided with the *RPKI Toolkit*, developed by Dragon Research Labs[4].

BGP is implemented by routing suites such as FRRouting (FRR) or BIRD[5]. FRR, a fork of Quagga[6], is a routing suite that supports an array of routing protocols for inter- and intra-domain routing. AS operators use FRR to configure routers to handle routing for their AS and apply policies to filter routes. FRR also makes use of RTRlib to implement RPKI functionality.

[2]https://github.com/rtrlib
[3]https://github.com/RIPE-NCC/rpki-validator-3
[4]https://github.com/dragonresearch/rpki.net
[5]http://bird.network.cz/
[6]https://www.quagga.net/



Nominiert für den Deutschen Computerspielpreis 2018

Marc Fleps studiert im Master-Teilstudiengang "Games" und hat gute Chancen, in diesem Jahr beim Deutschen Computerspielpreis ausgezeichnet zu werden. Er ist nomiert in...

| mehr... |

27.03.2018

Mi, 28.03.18 , 09:00 - 15:15 Uhr
Fachprojekt-Präsentationstag des Departments Pflege und Management

Mi, 28.03.18 , 16:45 - 18:45 Uhr
CCG Ringvorlesung – Stadt und Gesundheit

Figure 3.6: The earth-shaped symbol at the top right signals that the website is covered by a ROA. Green means valid, yellow means not found and red means invalid.

BIRD is similar to FRR. It also implements routing protocols and includes RTRlib to offer route origin validation support.

3.4 Deployment

The deployment of RPKI throughout the Internet is still underway. In 2015, research yields that there is a slow but steady increase of ROA coverage for IPv4 prefixes [33]. While the coverage was at about 2% in 2012, the number grew to about 5% in 2014. Later research shows that the ROA coverage in 2015 increased to about 6% [34]. The continuous increase of covered unique IPv4 prefix/origin pairs is observed by the NIST RPKI Monitor [35], so that meanwhile almost 10% of all announced IPv4 prefixes are covered by a ROA (2nd of May 2018) [36]. Yet, those ASes who do use RPKI do not necessarily filter invalid route announcements [37]. This may be due to internal routing policies.

As for BGPsec deployment, one problem is transitioning from BGP to BGPsec [38]. Routers will have to run performance critical actions, such as cryptographic validations. These actions must be executed with every BGPsec update message that arrives at the router, resulting in high overhead. Presumably, hardware upgrades must occur.

Another concern is broad deployment across the Internet. BGPsec works in a way that update messages only keep their cryptographic properties, as long as all routers along the path speak BGPsec. One non-BGPsec router is enough to break the chain of forwarding a protected AS path. ASes might hesitate to transition to BGPsec, while no one else does. Goldberg speaks of a “chicken-and-egg problem” [38] for AS operators.

4 BGPsec Implementation Concept for RTRlib and Software Routers

BGPsec AS path validation is not yet part of RTRlib, so AS paths cannot be validated there. The goal is to add this validation feature to RTRlib. While the operational part of BGPsec is covered by a software router such as FRR, validation functionality is handled by RTRlib. The procedure of AS path validation should mimic the procedure of origin validation in RTRlib (see Figure 4.1).

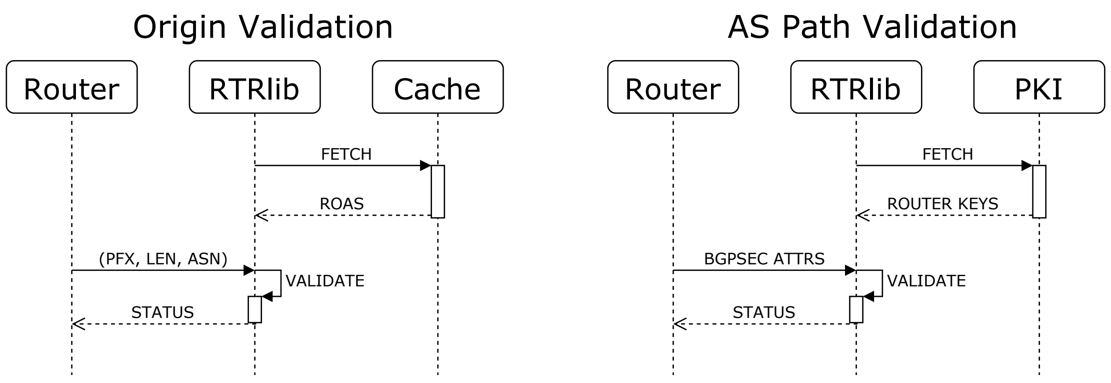


Figure 4.1: Left: the sequence of events when performing origin AS validation. Right: the sequence when validating an AS path. Both sequences are strongly simplified.

RTRlib aims to stay independent from any routing suite. The RPKI features of the library should be available to as many applications as possible, helping the deployment of RPKI even further. To make matters more easy to explain at some parts, FRR is chosen as an example for a software routing suite, but any other routing suite would serve just as well. There are a few reasons, why RTRlib and FRR were chosen for a BGPsec AS path validation implementation. First, RTRlib implements RTR version 1, which supports fetching and storing router keys. This is an essential feature for AS path validation. Second, because RTRlib is open source, free and widely acknowledged, it is likely to be picked up by anyone interested in using RPKI functionalities. A third, and more subjective reason is that the author is familiar with RTRlib as well as FRR.

There are two scopes to the implementation. One scope covers the operational functionalities such as negotiating BGPsec support, parsing and sending updates, etc. This is attributed to the routing suite. The other scope is the validation and signing process. These tasks will be carried out by RTRlib. In detail, there are four steps to successfully validate a BGPsec update.

1. Negotiate BGPsec support with a peering router
2. Parse, prepare and pass the data from the update to RTRlib
3. Validate the data and return the result
4. Sign the AS path and generate a BGPsec update.

Steps 1 and 2 need to be executed by the software router and are therefore implemented there. Step 3 is handled by RTRlib. The last step takes place in both RTRlib and the router. Sections 4.2 to 4.5 will cover the four steps in order.

4.1 Conceptual Considerations

Currently, RTRlib implements route origin validation. This functionality can be used by any application that includes the library without implementing any validation features itself. AS path validation aims for the same goal. Therefore, all necessary validation steps need to take place within RTRlib. The software router should not need to do any validation and only prepares the data in order to pass it to RTRlib. This way, a clear line can be drawn between what has to be done on side of the router, and what falls under the responsibility of RTRlib. An additional benefit to this approach is that the routing software does not need to include any libraries that are required for cryptographic operations itself. Instead it lets RTRlib include necessary libraries. Having a routing suite like FRR include some libraries that are already included by RTRlib just to perform AS path validation would be redundant.

Since AS path validation support for the routing suite depends on RTRlib, this feature should be checked at compile time. If the RTRlib version does not support AS path validation in general, BGPsec support for the router must be omitted. Otherwise, the router can make use of this feature.

Another consideration regards RPKI validation. RFC 8207 suggests, that BGPsec speakers should, in addition to AS path validation, also perform route origin validation [39]. A valid AS path does not mean that the announced prefix belongs to the origin AS. If validation of the origin AS is performed, it should be done before AS path validation, for the simple reason of performance. If the origin validation already concludes that the announcement is invalid, there is no need to do a performance critical AS path validation on top of that. For this concept, route origin validation is implicitly handled and therefore not explicitly depicted in any graphics.

With this settled, the aforementioned four steps are now described.

4.2 Step 1: Negotiation

Before BGPsec updates can be exchanged, a router must ensure that a peering router is able to speak BGPsec. BGPsec support is negotiated via the capability field of a BGP open message. A speaker may send an open message where the capability field contains a capability with code 7, which indicates BGPsec support. A payload is appended to the capability, containing BGPsec specific settings.

Current BGP implementations will need to extend capability parsing for code 7 and its payload. The payload contains information about whether or not a router wants to *send* or *receive* BGPsec updates and which *Address Family Identifier (AFI)* [40] it supports. It is either IPv4 or IPv6. As an example, a router may choose that it can *send* BGPsec updates for both IPv4 and IPv6. It propagates these information to a peering router by adding two capabilities to the open message.

Version = 0, Direction = 1, AFI = 1

Version = 0, Direction = 1, AFI = 2

The first capability tells the peering router that the sending router is capable to send BGPsec updates (direction set to 1) with IPv4 prefixes (AFI set to 1). The second capability tells the same about IPv6 prefixes (AFI set to 2).

The peering routers open message contains two capabilities as well.

Version = 0, Direction = 0, AFI = 1

Version = 0, Direction = 0, AFI = 2

This time, the direction is set to 0, which indicates that the router can *receive* BGPsec updates. Again, both AFI information must be sent via separate capabilities. If any router chooses not to support a certain AFI, the respective capability is omitted. Should both routers not find a common denominator, such as that a sending router only sends IPv6 prefixes, yet the peering router only wants to receive IPv4 prefixes, BGPsec updates cannot be sent and instead BGP must be used.

Figure 4.2 illustrates the whole process of capability negotiation.

Next, a received BGPsec update needs to be parsed and the included data needs to be prepared before it is passed to RTRlib.

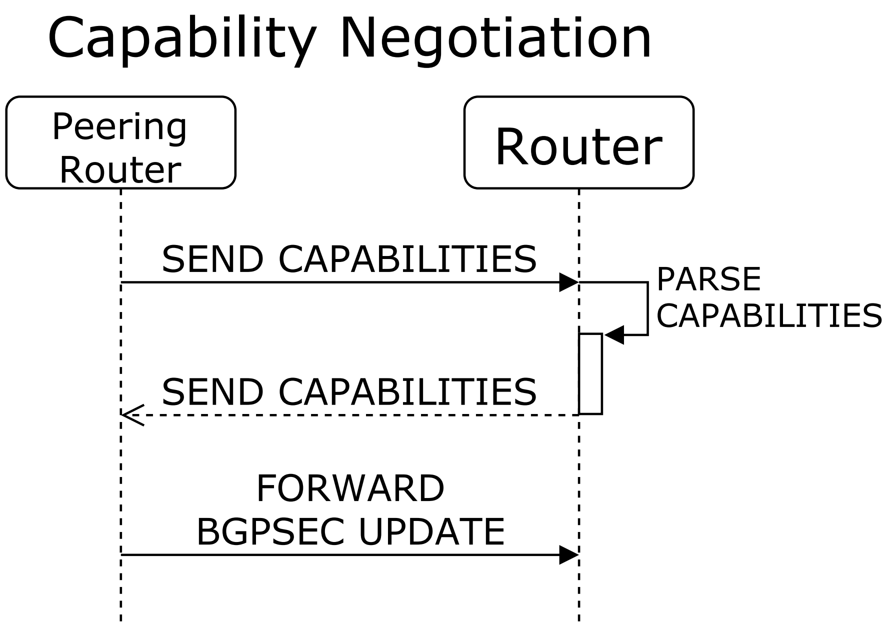


Figure 4.2: Before a router receives BGPsec updates, the capabilities must be exchanged via the open message. After a successful negotiation, BGPsec updates can be exchanged.

4.3 Step 2: Data Preparation

As soon as a BGPsec update arrives at the router, the router must perform an array of checks to make sure that the update is of integrity. The RFC specifies conditions that need to be met before further processing an update. A few examples are that there must be one Signature Segment for each Secure_Path Segment, or that the BGPsec update must not contain the BGP specific AS_PATH attribute. Because such checks are not part of the validation process, they fall under the responsibility of the software router.

An extra step that the routing suite needs to perform is checking, whether RTRlib supports the algorithms that were used to perform hashing and signing for the received update.

4.3.1 Algorithm Suite Identification

A BGPsec update contains an *Algorithm Suite Identifier*. This identifier tells an application, which hashing and signing algorithms were used. Any hashing or signing operations must be performed with the same algorithms that were used by the origin AS. Before any validation process is triggered, the software router needs to make sure that RTRlib supports the propagated algorithm suite. If RTRlib is not capable of performing the validation, it would be unnecessary to prepare the data for it. Thus, RTRlib must be able to expose all of its supported algorithm suites to an application. So in case RTRlib does not support a certain algorithm suite, FRR or any other software is able to check this. Such a check can be made by invoking one of two RTRlib functions.

Listing 4.1: Function headers of the algorithm support functions. The first one takes the algorithm suite identifier and returns 0 or 1, if supported or not. The second function returns a pointer to the supported suites and the amount of algorithm suites.

```
1  int bgpsec_check_algorithm_suite(int alg_suite);
2
3  int bgpsec_get_algorithm_suites(char *suites);
```

Listing 4.1 shows the header of the function that takes the algorithm suite identifier as parameter and returns 0 if the suite is supported and 1, if not. It might be more suitable for a routing suite like FRR to initially get all supported routing suites at once. To satisfy this need, RTRlib offers the function in line 3. It takes a pointer and sets its address to a char pointer that holds the identifier of all supported algorithm suites. The return value is the amount of suites the library supports. The routing suite can then check, if an announced algorithm suite identifier is stored at one of the addresses the pointer points to, instead of calling a function each time an update arrives.

An advantage of having these functions instead of statically linking the supported algorithm suites is that RTRlib could be exchanged with a new version that includes more suites, without having to rebuild the routing suite e.g., FRR. Restarting the suite is sufficient to adapt to the newly supported algorithm suites.

If the router receives a BGPsec update, it extracts the algorithm suite identifier and asks RTRlib if the algorithm is supported. If so, the software router can initiate the validation process. If not, the router must downgrade the BGPsec update to a normal BGP update because it has no way of validating its contents.

4.3.2 Required Parameters

Besides the algorithm suite identifier, there are various information required for AS path validation. Except for the target AS, they are all contained within the BGPsec update. Below is a list of all necessary parameters.

- Target AS
- All Signature Segments
  - SKI
  - Signature Length
  - Signature
- All Secure_Path Segments
  - pCount
  - Flags
  - AS Number
- Algorithm Suite Identifier
- AFI
- SAFI
- NLRI

Before the information are hashed, they have to be ordered in a manner which is specified by the RFC. For signing as well as validating, the order is identical. When the data is arranged, all the information (except for the Signature Segment appended last) are hashed with the algorithm specified by the algorithm suite identifier. The resulting digest is the *message* that was signed by the AS from where the update came last. Both sender and receiver now have the same message, which is the precondition for asymmetric signature validation.

For RTRlib to be able to calculate this message, all these information must be made available to the library. Therefore, the routing suite must extract the information from the update and store them in RTRlib data structures. Aligning the data as a byte stream beforehand is inconvenient because this way the data is hard to distinguish on the side of RTRlib and would have to be parsed again. As soon as all the data is extracted and stored in RTRlib structures, it is passed to RTRlib. A proposal for these structures are presented in Listing 4.2.

Listing 4.2: Strucures for storing the BGPsec update data and the function for passing the data.

```
 1  struct signature_seg {
 2          uint8_t ski[];
 3          uint16_t sig_len;
 4          uint8_t signature[];
 5  };
 6
 7  struct secure_path_seg {
 8          uint8_t pcount;
 9          uint8_t flags;
10          uint32_t asn;
11  };
12
13  struct bgpsec_data {
14          uint16_t target_as;
15          uint8_t alg_suite_id;
16          uint16_t afi;
17          uint8_t safi;
18          uint8_t nlri[];
19          uint16_t nlri_len;
20  };
21
22  int bgpsec_validate_as_path(...);
```

The structs in line 1 and 7 resemble the Signature Segment and the Secure_Path Segment as specified by the RFC (compare Figure 3.4). Storing the length of the SKI in the Signature Segment is not necessary, since it has a static size of 20 bytes. The length of the signature depends on the algorithm used to create it, hence the signature length field is mandatory to support future algorithm suites. The struct in line 13 contains the rest of the information, such as the AFI or the prefix. All these information are later passed to the AS path validation function of RTRlib. Line 22 shows the header of the function that receives the data. It is described in more detail in the next section. Figure 4.3 summarizes the process of data preparation.

The validation process itself in covered in the next section.

4.4 Step 3: Validation

To perform AS path validation, BGPsec relies on two different algorithms [28]. With the current algorithm suite, the information from Section 4.3.2 must be hashed with the *SHA-256*

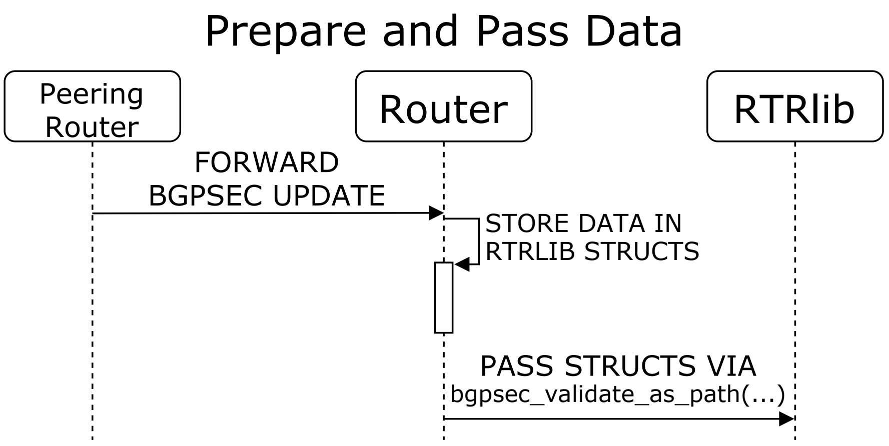

Prepare and Pass Data
Peering Router
Router
RTRlib
FORWARD BGPSEC UPDATE
STORE DATA IN RTRLIB STRUCTS
PASS STRUCTS VIA bgpsec_validate_as_path(...)

Figure 4.3: A peering router forwards a BGPsec update. The receiving router extracts the data and stores them in RTRlib structures. These structures are then passed to RTRlib.

algorithm [41]. The obtained digest is then signed using the *Elliptic Curve Digital Signature Algorithm (ECDSA)* with the *P-256* curve [42].

As soon as RTRlib receives the parameters, the validation process can begin. Algorithm 1 shows how the procedure could look like.

Algorithm 1 Validation Loop

```
1:  procedure BGPSEC_VALIDATE_AS_PATH(BGPSEC_DATA)
2:      $update\_status$
3:      for i < SKI_count do
4:          $router\_keys[i] \leftarrow get\_key(bgpsec\_data.ski[i])$
5:      for i < AS_hops do
6:          $digest \leftarrow calc\_digest(bgpsec\_data)$
7:          $sig \leftarrow bgpsec\_data.sig[i]$
8:          $update\_status \leftarrow validate\_signature(sig, digest, router\_keys[i])$
9:          if $update\_status$ = false then
10:             break
11:     if $update\_status$ = true then
12:         break
```

The `update_status` in line 2 is the current status of the update at all times during the procedure. In line 3 and 4, an iteration over all SKIs takes place to get the appropriate router keys, which are then stored in the `router_keys` array. The keys are acquired beforehand so RTRlib can make sure that all router keys are present. In case a router key is missing, the AS path cannot be fully validated and thus, continuing the validation procedure would be a waste of processing power. In line 6, the for-loop that handles AS path validation is started. The loop is repeated for every AS hop, i.e., every time the BGPsec update was forwarded. Within the loop, the digest for the currently processed signature gets calculated in line 7. Therefore, the whole BGPsec update data is required. Next, in line 8, the signature that is about to be validated is stored in `sig`. This variable is passed together with the digest and the appropriate router key to the `validate_signature` function in line 9 and the result is returned. If the result is false in line 10, the signature was not valid and the loop breaks. At this point, the whole update is considered not valid. If the result is true, the for-loop continues. If the loop finishes and the result is true, the BGPsec update is considered valid. With this, the validation procedure on side of RTRlib is finished and the result is returned to the routing suite.

As for route origin validation, RTRlib aims to provide a single function that has to be called by the routing suite in order to receive the validation result of the AS path. The header of the function is listed in Listing 4.3.

Listing 4.3: The function header of the AS path validation function.

```
int bgpsec_validate_as_path(struct bgpsec_data *data,
                            struct signature_seg *sig_segs,
                            struct secure_path_seg *sec_paths,
                            const unsigned int as_hops);
```

The Signature Segments and Secure_Path Segments are stored at memory locations that are accessible through pointers, one segment for each AS hop, therefore the hop count must be passed as well. The segments must be ordered from last to first segment. This means that the signature that is appended last to the update is at the first position of the memory. The signature that was created by the origin AS is at the last position. This ordering also applies to the Secure_Path segments. If desired in the future, operators may be able specify the sorting order by passing a flag to the function. To prevent breaking the API with this, a wrapper function that contains a default sorting option might be used for the validation function. To determine the outcome of the function, it returns an integer with a status code. Positive values signal a completed validation (0 = update is valid, 1 = update is not valid). Negative values indicate that an error occurred and there is no validation result. Depending on the error, a different negative value is returned.

Figure 4.4 shows a schematic diagram of validating an AS path.

The next step describes signing process and the creation of a BGPsec update.

4.5 Step 4: Generating the Signature

As soon as the routing suite receives the result of the AS path validation, it needs to decide how to proceed from there on:

The AS path is valid: the routing suite begins to initiate the signing procedure. After that, the router is ready to append itself to the BGPsec_PATH and forward the update to a BGPsec speaking peer.

The AS path is not valid: the routing suite should ignore the update because at least one signature was not correctly validated. This indicates that the update has been tempered with. Accepting or forwarding it would bear the risk of falling for AS path manipulation.

Signing the AS path is similar to validating it. The routing suite invokes an RTRlib function and passes all required information to it, including the routers private key. The digest that is signed is calculated the same way as it is for validation, except this time, the information of

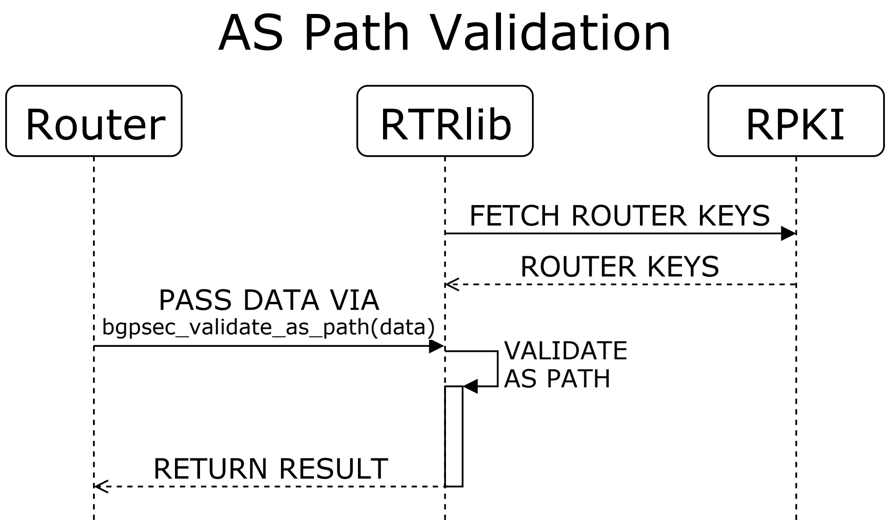


Figure 4.4: At first, RTRlib fetches all router keys from the RPKI since they are later required for the validation. When the software router passes the validation data to RTRlib, the library triggers the validation process and returns the result back to the router.

the signing AS are included. Signing is done using the routers private key. RTRlib obtains a signature and passes it back to the routing suite.

To make matters intuitive for the operator, the signing function call is similar to the call for validating a signature, as shown in Listing 4.4.

Listing 4.4: The function header of the signing function.

```
int bgpsec_create_signature(struct bgpsec_data *data,
                            struct signature_seg *sig_segs,
                            struct secure_path_seg *sec_paths,
                            const unsigned int as_hops,
                            char *priv_key,
                            char *new_signature);
```

For this function, the contents of `sig_segs` and `as_hops` are left unchanged while `data` and `sec_paths` need to be adjusted. The target AS field of the `data` structure must now hold the ASN of the AS where the update is forwarded to. Further, an additional Secure_Path Segment must be added to `sec_paths`. This segment contains the pCount, flags and ASN of the current AS, since they are required for hashing.

Two new parameters to this function are `priv_key` in line 5 and `new_signature` in line 6. `priv_key` holds the private key of the router that is used to sign the digest. If the function successfully creates a signature, it stores that signature in `new_signature` and returns its length. A negative return value indicates that an error occurred during the process.

Since forwarding an existing update is only one of two cases, there also needs to exist a way for generating a signature upon originating a BGPsec update. Since the process of first creating a BGPsec update and forwarding an existing one is very similar, an alternative function to `bgpsec_create_signature` in Listing 4.4 is not necessary. It is possible to pass NULL for `sig_segs` and have `sec_paths` only contain the segment for the own AS. Setting `as_hops` to 0 indicates that there was no prior forwarding and the data alignment is adjusted accordingly.

The routing suite may now (re)assemble the BGPsec update and send or forward it to the AS that was specified by the `target_as` field within the `data` struct. Note that a signature has to be created for each AS the router wants to forward it to. This is because the target AS is part of the hashing. Changing this value leads to a different digest.

At this point, the router has finished the BGPsec routine. Figure 4.5 summarizes the process of signing.

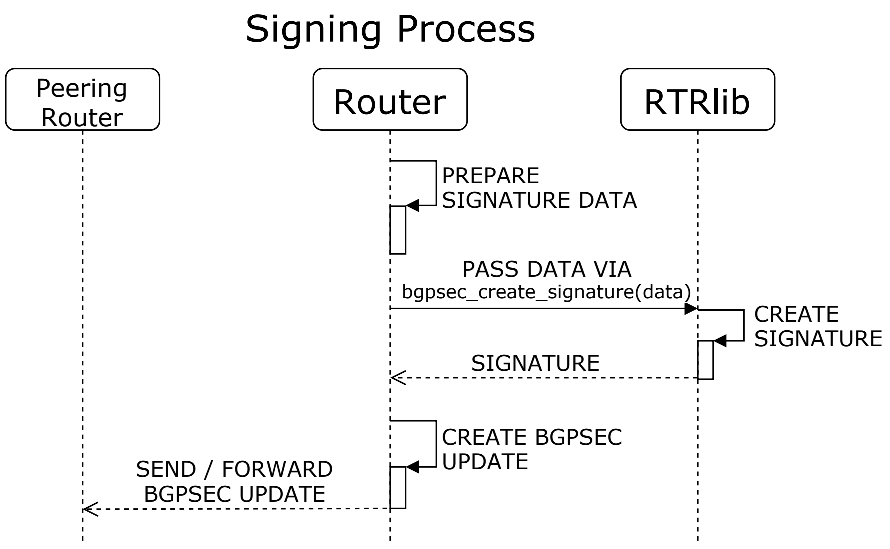


Figure 4.5: The router prepares the data that is necessary for the generation of a signature and passes it to RTRlib. The library creates and returns the signature. The process for originating a BGPsec update and forwarding an existing one is identical in this diagram.

4.6 Workflow

To give an overview of the whole procedure of processing a BGPsec update, i.e., going through all four steps, the graphics from the previous chapters are combined in Figure 4.6.

The graphic shows that the single responsibility principle is achieved. Any cryptographic operations, i.e., validating and signing, are handled by RTRlib. The router is responsible for everything that resolves around the validation process, i.e., communication with peering routers and (dis)assembling the BGPsec update. There also is no communication between router and RPKI directly, as RPKI features are encapsulated by RTRlib.

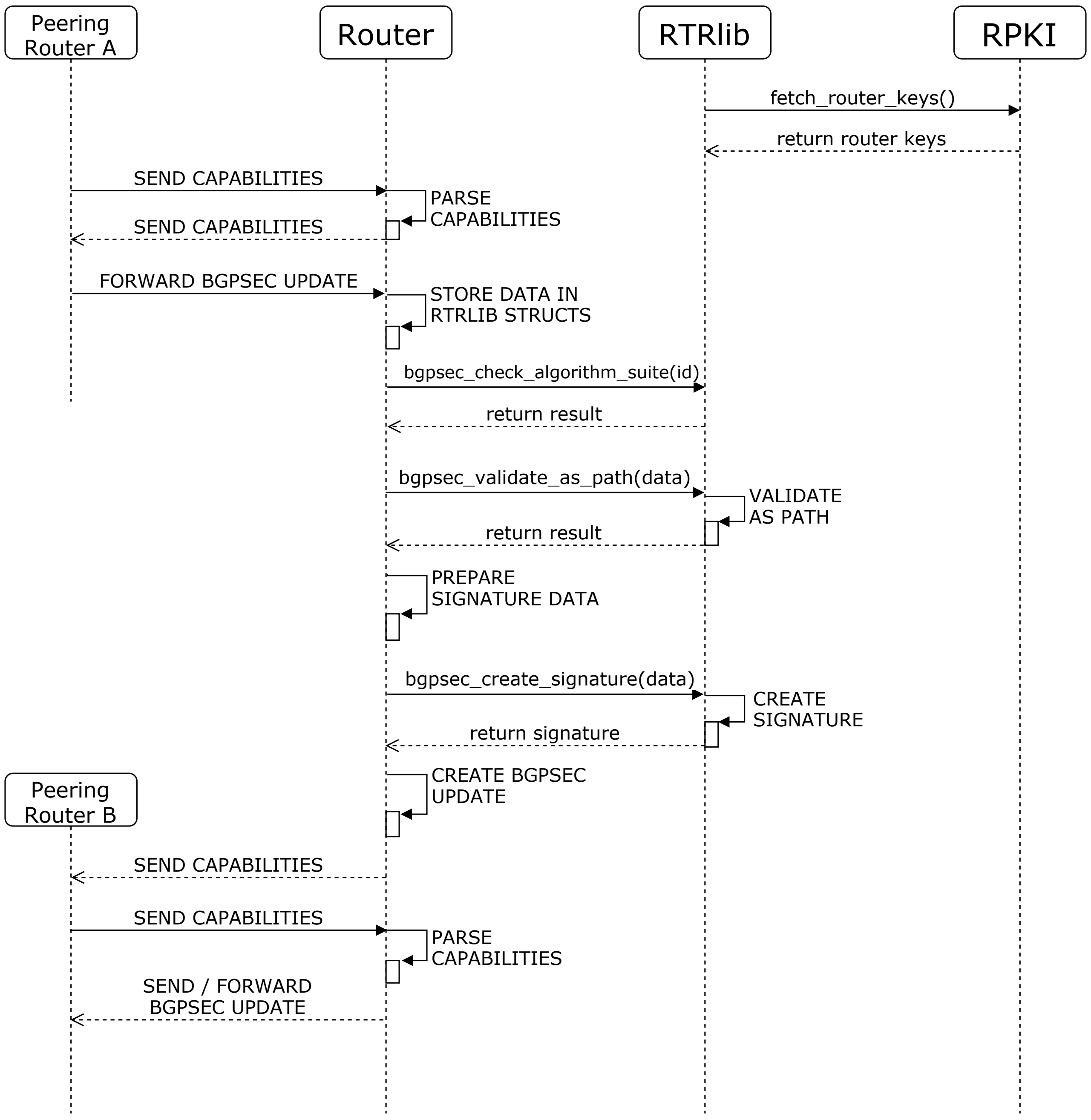


Figure 4.6: The whole workflow of processing a BGPsec update from receiving it from peer A, to forwarding it to peer B.

5 Conclusion

Offering BGPsec support to routing suites like FRR via the RTRlib will be a step by step process. The next step is implementing the core validation features for BGPsec AS path validation. This is done on side of RTRlib as shown in the previous section. One goal of the implementation is to be easily adaptable to new features and additions. Since the BGPsec RFC was released not long ago, changes in the next years might occur, especially concerning cryptographic validation suites.

The step after the implementation will be setting up a testing environment, to ensure the correctness of the implementation. The validation results of the RTRlib should be compared to other BGPsec implementations such as the BGP-SRx, developed by NIST [5], or the BGPsec implementation of BIRD. Additionally, the performance of RTRlib AS path validation should be measured and compared to other implementations. Performance of the validation is crucial, since it has high priority for operators [38].

This work tries to push forward the BGPsec availability, as securing Internet routing is as relevant as always and long overdue. Although deployment will take a long time, as RPKI has shown, it will eventually happen. The earlier implementations like this one can be tested throughout, the better it is for operators, who aim for smooth transitions from BGP to BGPsec.

Bibliography

[1] Y. Rekhter, T. Li, and S. Hares, “A Border Gateway Protocol 4 (BGP-4),” IETF, RFC 4271, January 2006.

[2] M. Lepinski and S. Kent, “An Infrastructure to Support Secure Internet Routing,” IETF, RFC 6480, February 2012.

[3] M. Wählisch, F. Holler, T. C. Schmidt, and J. H. Schiller, “RTRlib: An Open-Source Library in C for RPKI-based Prefix Origin Validation,” in *Proc. of USENIX Security Workshop CSET'13*. Berkeley, CA, USA: USENIX Assoc., 2013. [Online]. Available: https://www.usenix.org/conference/cset13/rtrlib-open-source-library-c-rpki-based-prefix-origin-validation

[4] M. Lepinski and K. Sriram, “BGPsec Protocol Specification,” IETF, RFC 8205, September 2017.

[5] NIST, “BGP Secure Routing Extension (BGP-SRx) Prototype,” 9 2017. [Online]. Available: https://www.nist.gov/services-resources/software/bgp-secure-routing-extension-bgp-srx-prototype (Accessed 16-05-2018).

[6] IANA, “Number Resources.” [Online]. Available: https://www.iana.org/numbers (Accessed 02-01-2018).

[7] IANA, “Autonomous System (AS) Numbers.” [Online]. Available: https://www.iana.org/assignments/as-numbers/as-numbers.xhtml (Accessed 20-02-2018).

[8] G. Huston, “Autonomous System (AS) Number Reservation for Documentation Use,” IETF, RFC 5398, December 2008.

[9] L. Gao, “On Inferring Autonomous System Relationships in the Internet,” *IEEE/ACM Trans. Netw.*, vol. 9, no. 6, pp. 733–745, 2001.

[10] L. Gao and J. Rexford, “Stable internet routing without global coordination,” *IEEE/ACM Transactions on Networking*, vol. 9, no. 6, pp. 681–692, Dec 2001.

[11] E. Chen and T. Bates, "An Application of the BGP Community Attribute in Multi-home Routing," IETF, RFC 1998, August 1996.

[12] I. van Beijnum, *"BGP"*. O'Reilly Media, Inc., 2002.

[13] J. Karlin, S. Forrest, and J. Rexford, "Autonomous Security for Autonomous Systems," *Computer Networks*, vol. 52, no. 15, pp. 2908–2923, 2008.

[14] K. Butler, T. Farley, P. McDaniel, and J. Rexford, "A Survey of BGP Security Issues and Solutions," *Proc. of the IEEE*, vol. 98, no. 1, pp. 100–122, January 2010.

[15] G. Huston, M. Rossi, and G. Armitage, "Securing BGP - A literature survey," *IEEE Communications Surveys & Tutorials*, vol. 13, no. 2, pp. 199–222, 2011.

[16] M. Wählisch, O. Maennel, and T. C. Schmidt, "Towards Detecting BGP Route Hijacking using the RPKI," in *Proc. of ACM SIGCOMM, Poster Session.* New York: ACM, August 2012, pp. 103–104. [Online]. Available: http://conferences.sigcomm.org/sigcomm/2012/paper/sigcomm/p103.pdf

[17] C. Zheng, L. Ji, D. Pei, J. Wang, and P. Francis, "A Light-Weight Distributed Scheme for Detecting IP Prefix Hijacks in Real-Time," in *Proc. of SIGCOMM '07.* New York, NY, USA: ACM, 2007, pp. 277–288.

[18] M. Handley and E. Rescorla, "Internet Denial-of-Service Considerations," IETF, RFC 4732, December 2006.

[19] J. Klensin, "Simple Mail Transfer Protocol," IETF, RFC 5321, October 2008.

[20] BGPmon, "Chinese ISP hijacks the Internet." [Online]. Available: https://bgpmon.net/chinese-isp-hijacked-10-of-the-internet/ (Accessed 12-03-2018).

[21] "YouTube Hijacking: A RIPE NCC RIS case study," http://www.ripe.net/internet-coordination/news/industry-developments/youtube-hijacking-a-ripe-ncc-ris-case-study, March 2008, retrieved 2013-08-16.

[22] A. Toonk, "BGPstream and The Curious Case of AS12388," 4 2017. [Online]. Available: https://bgpmon.net/bgpstream-and-the-curious-case-of-as12389 (Accessed 07-01-2018).

[23] A. Chadd, "The "AS7007 Incident"," 8 2006. [Online]. Available: http://lists.ucc.gu.uwa.edu.au/pipermail/lore/2006-August/000040.html (Accessed 06-01-2018).

[24] T. Wan and P. C. Van Oorschot, “Analysis of BGP prefix origins during Google’s May 2005 outage,” in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International.* IEEE, 2006, pp. 8–pp.

[25] M. Lepinski, S. Kent, and D. Kong, “A Profile for Route Origin Authorizations (ROAs),” IETF, RFC 6482, February 2012.

[26] R. Bush and R. Austein, “The Resource Public Key Infrastructure (RPKI) to Router Protocol,” IETF, RFC 6810, January 2013.

[27] R. NCC, “BGP Origin Validation,” 9 2016. [Online]. Available: http://www.ripe.net/manage-ips-and-asns/resource-management/certification/bgp-origin-validation (Accessed 12-01-2018).

[28] S. Turner and O. Borchert, “BGPsec Algorithms, Key Formats, and Signature Formats,” IETF, RFC 8208, September 2017.

[29] M. Reynolds, S. Turner, and S. Kent, “A Profile for BGPsec Router Certificates, Certificate Revocation Lists, and Certification Requests,” IETF, RFC 8209, September 2017.

[30] IEEE, “IEEE Standard Specifications for Public-Key Cryptography,” *IEEE Std 1363-2000*, pp. 1–228, Aug 2000.

[31] R. Bush and R. Austein, “The Resource Public Key Infrastructure (RPKI) to Router Protocol, Version 1,” IETF, RFC 8210, September 2017.

[32] M. Wählisch and T. C. Schmidt, “See How ISPs Care: An RPKI Validation Extension for Web Browsers,” *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 5, pp. 115–116, 2015.

[33] D. Iamartino, C. Pelsser, and R. Bush, “Measuring BGP route origin registration validation,” in *Proc. of PAM*, ser. LNCS. Berlin: Springer, 2015, pp. 28–40.

[34] M. Wählisch, R. Schmidt, T. C. Schmidt, O. Maennel, S. Uhlig, and G. Tyson, “RiPKI: The Tragic Story of RPKI Deployment in the Web Ecosystem,” in *Proc. of 14th ACM Workshop on Hot Topics in Networks (HotNets).* New York: ACM, Nov. 2015, pp. 11:1–11:7. [Online]. Available: http://dx.doi.org/10.1145/2834050.2834102

[35] NIST, “Global Prefix/Origin Validation using RPKI.” [Online]. Available: https://rpki-monitor.antd.nist.gov/ (Accessed 20-03-2018).

[36] R. READ, “RPKI READ Realtime Dashboard.” [Online]. Available: https://rpki-read.realmv6.org/ (Accessed 02-05-2018).

[37] A. Reuter, R. Bush, I. Cunha, E. Katz-Bassett, T. C. Schmidt, and M. Wählisch, “Towards a Rigorous Methodology for Measuring Adoption of RPKI Route Validation and Filtering,” *ACM Sigcomm Computer Communication Review*, vol. 48, no. 1, pp. 19–27, January 2018, selected Best of CCR for SIGCOMM 2018. [Online]. Available: https://ccronline.sigcomm.org/2018/towards-a-rigorous-methodology-for-measuring-adoption-of-rpki-route-validation-and-filtering/

[38] S. Goldberg, “Why is It Taking So Long to Secure Internet Routing?” *Commun. ACM*, vol. 57, no. 10, pp. 56–63, Sep. 2014.

[39] R. Bush, “BGPsec Operational Considerations,” IETF, RFC 8207, September 2017.

[40] IANA, “Address Family Numbers.” [Online]. Available: https://www.iana.org/assignments/address-family-numbers/address-family-numbers.xhtml (Accessed 26-02-2018).

[41] National Institute of Standards and Technology, “FIPS 180–3, Secure Hash Standard, Federal Information Processing Standard (FIPS), Publication 180-3,” http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf, Department of Commerce, Gaithersburg, MD, US, Tech. Rep., October 2008.

[42] NIST, “Digital Signature Standard,” Federal Information Processing Standards 186–4, July 2013.