# SPV: Secure Path Vector Routing for Securing BGP*


Yih-Chun Hu
UC Berkeley
yihchun@cs.cmu.edu

Adrian Perrig
Carnegie Mellon University
perrig@cmu.edu

Marvin Sirbu
Carnegie Mellon University
sirbu@cmu.edu



## ABSTRACT

As our economy and critical infrastructure increasingly relies on the Internet, the insecurity of the underlying border gateway routing protocol (BGP) stands out as the Achilles heel. Recent misconfigurations and attacks have demonstrated the brittleness of BGP. Securing BGP has become a priority.

In this paper, we focus on a viable deployment path to secure BGP. We analyze security requirements, and consider tradeoffs of mechanisms that achieve the requirements. In particular, we study how to secure BGP update messages against attacks. We design an efficient cryptographic mechanism that relies only on symmetric cryptographic primitives to guard an ASPATH from alteration, and propose the *Secure Path Vector* (SPV) protocol. In contrast to the previously proposed S-BGP protocol, SPV is around 22 times faster. With the current effort to secure BGP, we anticipate that SPV will contribute several alternative mechanisms to secure BGP, especially for the case of incremental deployments.



**Categories and Subject Descriptors:** C.2 [**Computer-Communications Networks**]: Security and protection; C.2.2 [**Network Protocols**]: Routing Protocols

**General Terms:** Security, Performance

**Keywords:** Interdomain routing, security, routing, Border Gateway Protocol, BGP


## 1 INTRODUCTION

Critical business and governmental functions increasingly rely on the Internet. Even though the Border Gateway Routing Protocol (BGP) is central for Internet packet routing, it was designed for a trusted environment and provides relatively minimal security against an attacker [46]. Recent studies and examples show that even routine misconfigurations severely disrupt Internet routing [34, 41]. The need to secure BGP has become increasingly pressing, so Kent et al. proposed the Secure BGP routing protocol (S-BGP) [27], and the IETF has established the rpsec working group [51] to detail requirements for a secure routing protocol.

There are several impediments to the deployment of BGP security, many of which are inherent to any Internet deployment that needs wide-spread adoption. A viable scheme must provide incremental benefits even when not all routers participate. Changes to existing router code should be minimized, and unchanged (legacy) routers must not break when enhanced messages are received from participating routers. Secured updates must fit within the length limit of BGP Update messages. Some form of certificate hierarchy is necessary for authenticating public keys of participants, but the use and management of private keys should be minimized for operational simplicity.

Computational efficiency in authenticating Update messages is important because core Internet routers receive a high volume of such messages, and they may arrive in bursts. Efficient verification during these bursts is of utmost importance, since such bursts generally occur when the routing topology has changed, and data packets will be dropped or misrouted until routing reconverges. Moreover, providing security against routing update replay attacks may require periodic Update messages rather than only event driven messages, making efficient authentication even more desirable.

In this paper, we make several contributions to securing BGP. We propose a new protocol, SPV, which replaces much of the computationally expensive asymmetric cryptography, as used in S-BGP, with a far less costly signature scheme based on symmetric cryptography. Moreover, the proposed cryptographic mechanisms can be cheaply implemented and easily parallelized in special purpose hardware, thus providing additional speedup. To the best of our knowledge, this is the first proposal to use efficient symmetric key cryptography to prevent an attacker from modifying and truncating the ASPATH. While still requiring some use of asymmetric cryptography and certificate hierarchies, the proposed scheme removes the need for keeping private keys on routers, which simplifies overall key management (we only store short-lived one-time private keys on routers). Using BGP traces, we demonstrate that SPV is much more efficient than previous approaches.

SPV also includes improved mechanisms for securing against ASPATH tampering as compared to S-BGP as currently proposed; we show how these same improvements could also be incorporated into S-BGP. The focus of SPV is on protecting BGP Update messages; we do not focus on the integrity of BGP policy definitions. However, SPV will protect against certain types of misconfiguration, such as a BGP speaker configured with the wrong ASN.

Finally, we show how SPV provides improved security against malicious ASPATH modifications, relative to S-BGP, when only some routers implement a secure BGP protocol; in fact our mecha-


*This research was supported in part by the Center for Computer and Communications Security at Carnegie Mellon under grant DAAD19-02-1-0389 from the Army Research Office, the National Science Foundation under grant CAREER CNS-0347807, the U.S. Department of Homeland Security (DHS) and the National Science Foundation (NSF) under grant ANI-0335241, and by gifts from Cisco, Intel, and Matsushita Electric Works Ltd. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of ARO, NSF, Carnegie Mellon University, UC Berkeley, Cisco, Intel, Matsushita Electric Works Ltd., or the U.S. Government or any of its agencies.




*SIGCOMM'04,* Aug. 30–Sept. 3, 2004, Portland, Oregon, USA.

nisms provide attack and misconfiguration protections even if only two ISPs deploy them. SPV provides an increased level of security, which should make it easier to convince system administrators of the value of deployment. Such tangible properties may help motivate ISPs to deploy these mechanisms, since an administrator can determine which recent routing problems the secure protocol could have prevented.

**Outline** Our paper is organized as follows. Section 2 introduces our assumptions and attacker model, and Section 3 discusses prior research that is closely related. We describe SPV in Section 4 and evaluate it in Section 5. Finally, we discuss other related work in Section 6 and conclude in Section 7.

## 2 BGP SECURITY THREATS

Several researchers have studied BGP vulnerabilities [2, 11, 28, 51]. Based on this prior research, we establish classes of BGP vulnerabilities, and we discuss the security properties that SPV provides in Section 5.1.

We consider *active attackers* that actively inject malicious traffic into the network. We consider a strong attacker model, where the attacker compromises routers in the network to perform denial-of-service (DoS) or falsification attacks. This is equivalent to a malicious insider who can control the routers and will thus know all cryptographic keys of the infrastructure. Even in this case, we want to prevent the attacker from falsifying external information, so that it can only affect internal prefixes.

We distinguish two main attack classes: falsification and denial-of-service (DoS). Our categorization is based on the routing protocol's reaction to the attack; in particular, because routing underlies almost all services, falsification attacks can result in denial of a specific service, but will in most cases not result in denial of the routing service itself.

### 2.1 Falsification Attacks

We loosely define a **falsification attack** as a bogus BGP protocol message that differs from a message that a correctly configured router would send.[1] The term "falsification" is also used by Barbir, Murphy, and Yang [2]. Due to space limitations, we do not discuss falsifications of components of an OPEN, NOTIFICATION, or KEEPALIVE message; these messages could be secured by using IPsec [26, 25] and appropriate certificates. In the subsequent discussion, we focus on the falsification of the UPDATE message.

We now consider the components of a BGP UPDATE message (withdrawn routes, path attributes, and network layer reachability information (NLRI)), and discuss the impact of a falsification of the contained information.

An attacker can falsify information in the list of **withdrawn routes**. For example, an attacker can fail to withdraw a route when the attacker no longer has a working route, or send a withdrawal for a working route. The latter may not be malicious, depending on the policy of that AS; for example, if two peers may change their transit agreement, then each of their neighbors may see withdrawals for working routes.

An attacker can **falsify the NLRI**, i.e., the IP prefixes that the UPDATE message and path attributes pertain to. Within the NLRI, an attacker can originate a route to a prefix with which it is not affiliated, and can try to force other routers to prefer that route by advertising a longer prefix for that route. Since longer (more specific) prefixes are preferred, the attacker would thus be creating a blackhole.

[1] Different BGP implementations may produce different messages; we consider a message to be correct if it adheres to the BGP protocol specification and is produced by a router with a "correct" configuration.

The *blackhole attack* is a general attack that relies on falsification. In a blackhole attack, a malicious AS injects malicious routing information to attract traffic that would otherwise not flow through it, thus gaining control of a path. The blackhole attack is very powerful, since an attacker in this position can deny routing to certain addresses, eavesdrop on all traffic to a particular destination, or use its routing position to perform man-in-the-middle attacks. An example of a blackhole is the AS 7007 incident, where, due to a misconfiguration, an AS announced short routes to many destinations, causing global connectivity problems for two hours [41]. We distinguish blackhole and grayhole attacks. In a *grayhole*, the attacker selectively drops traffic flowing through it, without injecting any malicious routing information into the network. Grayhole attacks cannot easily be prevented by a secure routing protocol (since no falsified routing information was injected), whereas blackhole attacks can be prevented.

We now discuss **falsification of the path attributes**. As a simple defense against falsification, the origin and local pref attributes received from an eBGP session can be set to a default value, and the MED and next hop attributes are ignored unless otherwise agreed upon by both peers. Due to lack of space, we also do not further discuss the atomic aggregate and aggregator attributes, though these are automatically secured by SPV, as described in Section 4.5.

A severe attack is **AS path** or **AS set falsification**, since falsifying the AS path has serious consequences. For example, many policy decisions are made based on the AS path, and the length of the AS path is (after the local pref value) the second consideration in selecting a path towards a destination. As a result, changing the AS path can cause the attacker's route to be preferred.

An attacker can create a more preferable route by **shortening the AS path**. In this paper, we refer to this attack as the **truncation attack**, and it can be used to form a blackhole. An attacker can also **modify the AS path** by altering ASNs. In this paper, we refer to this attack as the **modification attack**. This may cause the next AS to prefer a route through the attacker, and the attacker may be able to create a blackhole.

An attacker with control of the preferred route can select between the grayhole attack (i.e., selective dropping) and a number of similar attacks. For example, an attacker can make a neighboring AS discard a prefix by falsely prepending the neighboring ASN to that prefix in the UPDATE to trigger loop detection when that prefix reaches the AS [17]. (However, to recover from partitions of an AS, some ASes do not drop UPDATEs that would result in a loop [54].) A somewhat weaker attack is to prepend random ASNs to the AS path for the purpose of making the path longer. In general, this should discourage traffic from using this route; in fact, this preference for shorter AS paths is useful for traffic engineering (through the use of AS path prepending). An attacker can also attempt to use BGP's flap damping to delay a router from picking an alternate route to some destination. This achieves the same result as the grayhole attack (packets are dropped in the network), though it is somewhat stealthier. All these attacks, however, require that the attacker control the preferred path. As a result, we consider the grayhole attack to be more powerful, and ignore these attacks in the rest of the paper.

A **wormhole** or **tunnel** attack is a specific mechanism that can be used to perform blackhole attack, where multiple colluding BGP routers exchange BGP UPDATE traffic over a tunneled connection, such as PPP over TCP [24]. These tunneled BGP UPDATEs allow a router to claim better paths to a destination than actually physically exist. The tunneling attack is devastating even against secure BGP routing protocols; for example, in S-BGP, two colluding malicious routers can forward UPDATE traffic to each other, signing route attestations for each other. SPV is also unable to prevent these attacks.

**Unauthorized propagation or AS path announcements.** An attacker can propagate a route that it should not. For example, service providers generally do not want their routes to be readvertised by their customers; otherwise, a multihomed customer may begin advertising transit capability between two large providers. However, that customer is unlikely to have sufficient resources to provide such a service, which would result in either severely degraded performance or a blackhole. In addition, a router should only announce prefixes and AS paths that it actively uses to route towards that prefix. A malicious router may announce prefixes and AS paths it is not actively using, and greatly increasing the rate of BGP UPDATEs.

### 2.2 Denial-of-Service (DoS) Attacks

If we consider routing as a service, falsification attacks can also result in DoS. However, here we discuss DoS attacks on the BGP speaker, and on the TCP connection of the BGP session.

The classic DoS attack is a resource exhaustion attack. An attacker may be able to paralyze a router by **exhausting its computation resources**. Several approaches may be used to perform this attack. An attacker may be able to trick a router into performing resource-intensive operations, such as public-key certificate verifications or signature generations, which require on the order of milliseconds each. Recently, researchers discovered algorithmic complexity attacks, where an attacker fabricates inputs to evoke the worst-case running time of an algorithm, slowing the device down to a crawl [12]. As a result, a router may not be able to recompute its routing table, process BGP UPDATE messages, or even keep the BGP session alive. If a BGP session is torn down, each BGP peer will withdraw any routing table entries learned from the other BGP peer.

Another attack is to **exhaust the bandwidth** of a network link to starve off the TCP connection used for the BGP session. If the BGP peers are directly connected, they may allocate bandwidth resources for the BGP session; however, some BGP peers are not directly connected (such as peers connecting over an exchange point); in such cases, an attacker can flood intermediate links. A flooding prevention mechanism such as SIFF could be used in this case [58].

Finally, an attacker can use **low-layer protocol attacks** to directly attack the BGP session. The attacker may mount a PHY layer attack to disable communication links. Since BGP relies on the TCP protocol, it inherits all TCP vulnerabilities [4].[2] For example, due to a lack of origin authentication in the TCP protocol, an attacker can inject malicious TCP packets, spoofing the IP source address of the other TCP end point, an attack which we call **TCP poisoning**. In this attack, the attacker guesses the TCP sequence numbers and injects bogus TCP reset (RST) packets to cause the victim to close its TCP connection. Using IPsec will prevent this attack.

## 3 CLOSELY RELATED WORK

In this section, we discuss prior work that is closely related to SPV. We discuss the remaining related work in Section 6.

### 3.1 Hop-by-Hop Authentication

In hop-by-hop authentication, ISPs use authentication to prevent attacks against the eBGP TCP session (e.g., malicious message injection by an outsider). However, falsification of the AS path cannot be addressed by hop-by-hop authentication. However, hop-by-hop authentication is an important start, and several approaches based entirely on hop-by-hop authentication have been proposed in the literature. Kumar and Crowcroft discuss security requirements and propose hop-by-hop encryption and authentication to secure routing UPDATEs [31]. Smith and Garcia-Luna-Aceves [52] discuss weaknesses of the BGP protocol and propose some basic countermeasures, which use digital signatures to provide hop-by-hop authentication. TCP-MD5 authentication has also been proposed to provide authentication between two BGP speakers [21]. IPsec [25, 26] can also authenticate the link-level communication between peering routers. The TTL value can be used to verify that a packet really originates from a neighboring router [43].

### 3.2 Securing BGP Updates

Kent et al. propose S-BGP, the seminal work that protects the entire BGP UPDATE message [27]. S-BGP assumes two parallel certificate hierarchies: an address space PKI, and an AS ownership and router PKI. Both certificate hierarchies have ICANN as their certificate root. The address space certificate assigns ownership over an IP prefix to an entity, the address space hierarchy parallels the existing IP address allocation system (address issuer signs address owner's certificate). The AS ownership certificate assigns ownership over an ASN to an entity. An AS uses its AS ownership to sign certificates for routers.

The main goal of S-BGP is to protect the ASPATH from modification and truncation, and to prevent unauthorized advertisements of an IP prefix. To prevent unauthorized prefix advertisements, S-BGP uses *address attestations*, where the owner of an IP prefix signs a delegation message allowing its first-hop AS to advertise the prefix. Subsequently, each AS signs a *route attestation* for the AS path up to and including that AS, and a *delegation*, allowing the next AS the right to propagate that advertisement to its peers. Hence, a S-BGP UPDATE message starts with an address attestation which proves that the originator has permission to advertise the route. The originator signs a delegation which allows the next AS to propagate the route, and each AS in turn signs a delegation which allows the following AS to propagate the route.

Each delegation in the delegation chain ensures two properties: first, the next ASN cannot be modified without the previous ASN's private key, and second, the AS path cannot be propagated without the permission of the final AS on the AS path. As a result, an attacker cannot introduce itself onto the AS path unless it receives a delegation allowing it to propagate that prefix. It also cannot remove previous ASNs from the AS path, because to do so would require the private key of the AS before the removed AS. S-BGP requires several digital signatures in each UPDATE, and as a result has a high CPU overhead for verifying UPDATE messages. SPV improves on S-BGP through the use of more efficient symmetric cryptography.

### 3.3 Anomaly Detection

A number of approaches attempt to determine whether or not a given UPDATE is likely to be valid. SPV is a complementary approach, designed to prevent modification and truncation of ASPATHs. Though a combination of detection and prevention is promising, it is beyond the scope of this paper, but represents an interesting direction for future work. We now review some prior work in anomaly detection.

Secure Origin BGP (soBGP) is an effort to secure BGP [57]. The approach of soBGP is for each router to keep a database of network topology (AS connectivity information), BGP policy information, as well as trusted ISP certificates. The routers use this database to assess the authenticity of UPDATEs, mainly by detecting that a given UPDATE is impossible. Goodell et al. propose a separate protocol to secure BGP route updates, without changing BGP [19]. Their approach is similar in nature to soBGP, as it adds a separate mechanism to authenticate BGP UPDATEs.

[2] Even though TCP is technically at a higher layer of the protocol stack we include this in the low-layer attack section, as BGP relies on TCP.

Cheung and Levitt [10] and Bradley et al. [7] propose intrusion detection techniques for detecting and identifying routers that send bogus routing UPDATE messages.

Subramanian et al. present new mechanisms to detect invalid UPDATE messages in their Listen and Whisper work [55]. They suggest adding a small amount of cryptographic information to UPDATEs to enable an AS to detect ASPATH truncation (one of their techniques is also based on efficient symmetric cryptographic primitives). They also propose to monitor TCP flows to detect paths that become unavailable.

Kruegel et al. present mechanisms to detect malicious updates, based on AS topology information [29].

# 4 SECURING BGP

In this section, we describe our extensions to secure BGP, which we call secure path vector (SPV). Our goal is to achieve ASPATH integrity through purely symmetric functions. One of the main motivations for this direction is to remove the need for routers to perform computationally expensive public-key cryptographic operations and to store asymmetric private keys. Private keys often have long lifetimes, and their compromise represents a significant security breach. As a result, managing private keys is a challenging task. In SPV, routers need only store the short-lived private keys for one-time signatures; these short-term keys can be generated offline. This approach greatly improves security, as the prefix private key could be stored on an off-line workstation.

By ASPATH integrity, we mean that a malicious AS or misconfigured router cannot shorten the ASPATH (which we call the truncation attack) or change autonomous system numbers (ASNs) in the ASPATH. Preventing the shortening of the ASPATH prevents blackhole attacks that are due to truncated paths, and ensuring integrity of ASNs in the path prevents ASPATH modification attacks. To achieve this integrity, we develop an ASPATH protector, a backwards-compatible cryptographic mechanism which can be added as a path attribute.

To prevent the attack in which an attacker replays old UPDATE messages to advertise routes which do not currently exist, we assume that time is divided into epochs of fixed length, after which all routes must be readvertised. Each UPDATE message is valid for one epoch following the epoch in which it was announced. Epochs are further discussed in Section 4.5.

In SPV, we use four different kinds of public/private keys (these keys are described in more detail later in this section):

- A *single-ASN public key* authenticates the signature of one AS in the ASPATH. The corresponding *single-ASN private key* is used to derive the one-time signature and the single-ASN public key.

- An *epoch public key* authenticates one ASPATH protector, which consists of a sequence of single-ASN one-time signatures. The epoch public key is the root of a hash tree over multiple single-ASN public keys (we describe hash trees in Section 4.2.2).

- A *multi-epoch public key* authenticates multiple epoch public keys. This value is the root of a hash tree that is constructed over multiple epoch public keys.

- A *prefix public/private key* is used to authenticate messages from a given prefix. This is a standard public/private key pair (for example using the RSA algorithm), which follows the same structure as the address PKI structure of S-BGP [27]. The main purpose in SPV for the prefix public key is to authenticate multi-epoch public keys, producing the *multi-epoch public key certificate*. The prefix key pair prevents an attacker from advertising a prefix which it does not own.

  In the same way that S-BGP uses address attestations [27], we assume that ICANN will issue certificates for prefix public keys to designate address ownership.[3]

SPV secures BGP UPDATE messages as follows. A node advertising a prefix must have the *prefix private key* associated with that block in order to generate a valid SPV UPDATE message. Using a combination of one-time signatures, hash trees, and one-way chains, we design a novel construction to protect ASPATHs against truncation and modification attacks. We observed that authentication of the AS forwarding UPDATEs is not necessary to secure ASPATHs, and causes problems also for incremental deployment. We propose an ASPATH protector that prevents a malicious AS from truncating or maliciously modifying the ASPATH, without authenticating the AS that updated the ASPATH and forwarded the UPDATE message.

In our approach, the owner of a prefix creates a sequence of one-time signatures, where each one-time signature is used to secure one ASN in the ASPATH. The address attestation is used to authenticate the public keys of the one-time signature, and the address owner passes all the private keys in the UPDATE to the next AS. Each AS that forwards the UPDATE uses up one one-time signature to sign itself into the ASPATH and removes the private key for that signature.

When a subsequent AS receives an UPDATE message, it can verify the integrity of the ASPATH by verifying all the one-time signatures. A malicious AS cannot truncate the ASPATH because it cannot recreate the private key of the removed ASNs, and it cannot replace a previous ASN with its own ASN (except with very small probability, as we analyze in a later section).

### 4.1 Efficient Prefix Ownership Certificates

To ensure that a prefix actually belongs to the AS which is originating an update for it, we use certificates to build attestations of prefix ownership. These certificates are equivalent to the address space PKI structure of S-BGP [27]. ICANN assigns IP address space to registries, which in turn delegate smaller blocks to service providers. Service providers often delegate these blocks to their customers. At each step in the delegation, the recipient of the address block generates an asymmetric *prefix private key* to represent the block; we call the corresponding public key a *prefix public key*. The address issuer uses its prefix private key to sign the prefix public key of the delegated block, together with a list of prefixes which are delegated to the new key, forming the *prefix public key certificate*, or simply *prefix certificate*. The network that owns a prefix thus has a certificate signed by the issuer of the prefix authorizing the prefix public key to authenticate messages to originate from that prefix.

Disseminating the prefix public keys is a challenge. A promising approach is to leverage identity based cryptography (IBC) [6]. In IBC any value can serve as the public key—the name IBC was chosen because an arbitrary string such as a name, identity, or a prefix can be used as the public key. Based on the public key, the certification authority can then compute the corresponding private key. In IBC, no public-key certificates are necessary, because the name is the public key. Assuming ICANN as the globally trusted authority of an identity based PKI, ICANN could issue IBC private keys using the prefix as the public key, which would remove the requirement for certificates, which in turn would solve the certificate distribution problem. ICANN could authenticate the owner through the prefix public key certificates we describe above. However, IBC

[3]However, in contrast to S-BGP, which requires additional certificates to authenticate ASNs, address attestations are the only requirements for computationally expensive public-key cryptography in SPV. We use these address attestations to bootstrap the one-time public keys that protect the ASPATH.

has several drawbacks: key revocation is an issue since we cannot revoke identities or prefixes (usually addressed by short-lived keys by appending expiration times to the public keys along with frequent private-key reissuing); and ICANN will know the private key for every prefix. For the purposes of this paper, we assume either a certificate distribution mechanism for the prefix public keys, or the use of IBC.

Instead of signing UPDATE messages, we use an ASPATH protector, which is built entirely from efficient symmetric primitives. An ASPATH protector can be authenticated using a single value, called the *epoch public key*. Because the ASPATH protector changes periodically, an AS builds a hash tree (described in more detail in Section 4.2.2) over each of these authenticators for a small set of ASPATH protectors. We call the root of this hash tree the *multiple-epoch public key*, because all the ASPATH protectors for several epochs can be verified with that public key. The AS signs the multiple-epoch public key with the prefix private key, producing the *multi-epoch public key certificate*.

The multi-epoch public key certificate can be distributed in a number of ways. For example, routers can use a separate protocol to flood certificates through the network. Alternatively, certificates could be erasure-encoded and pieces flooded within the UPDATE messages themselves, for example using a Digital Fountain based approach [8].

### 4.2 Cryptographic Mechanisms

In this section we review the basic cryptographic mechanisms that we use in this work. We review one-way hash chains, tree-authenticated values (also known as Merkle hash trees [38]), and one-time signatures. For a reference text on cryptographic terminology and constructions see [37].

In this paper we make use of highly efficient, symmetric cryptographic functions, such as one-way hash functions (e.g., MD5 [48]), and block ciphers (e.g., AES [14]). Asymmetric cryptographic primitives, such as RSA signatures [49], are computationally expensive: RSA signature verification is about three orders of magnitude slower than one symmetric operation (block cipher or hash function operation), and signature generation is about four orders of magnitude slower. When implemented in hardware, the speed difference is even larger. For example, when using a Xilinx Virtex FPGA with a "-6" speed grade, a 1024-bit RSA accelerator [15] can perform 54,610 modular multiplications (or 18,203 RSA verifications) per second in 5458 slices[4] (679% of the performance of a 1GHz Pentium III), whereas an unpipelined implementation of AES [56] using 460 slices can perform 5 million hashes per second on a slower FPGA (137% of the performance of a 1GHz Pentium III). Since these implementations are trivially parallelizable, in the same size FPGA, symmetric cryptographic primitives provide 1625% of the performance of a 1GHz Pentium III. For efficiency, we base our techniques on symmetric primitives. We need two operations: a one-way function, and a pseudo-random number generator.

For the one-way function $H[x]$, we could use a cryptographic hash function such as MD5 [48]. For the security of our mechanisms, we require that the hash function provides second pre-image collision resistance, which means that given a random value $x$, it is computationally infeasible to find $x' \neq x$ such that $H[x] = H[x']$. For efficiency reasons we use a hash function constructed from a block cipher. We use the AES [14] block cipher in the Matyas, Meyer, and Oseas hash construction [36]: $o = H[i] = \mathsf{AES}_K(i) \oplus i$. Black, Rogaway, and Shrimpton also show the security of this construction [5]. For the key $K$, we use a publicly known key $K$, note that it is intractable to derive the input $i$ even if given output $o$ and key $K$, thus giving us the one-way property. This construction is standardized in ISO/IEC 10118-2 and is particularly efficient if we hash an input value that is of the same size as the block cipher.

To generate a sequence of pseudo-random numbers, we use pseudo-random functions (PRF) [18]. A PRF takes two arguments, $\mathcal{X}$ is the key and $i$ is the input value, and produces an output value $o = \mathcal{F}_{\mathcal{X}}(i)$ that is indistinguishable from a random value as long as the key $\mathcal{X}$ is secret. We will use the PRF to derive a sequence of random values, such that given a sequence of output values, assuming the key $\mathcal{X}$ is secret, it is intractable to find other unpublished output values, or to derive the key $\mathcal{X}$. We also use the AES block cipher as our PRF, so $F_{\mathcal{X}}(i) = \mathsf{AES}_{\mathcal{X}}(i)$.[5]

For our cryptographic primitives, we aim for a security level of 80 bits, requiring an attacker to perform on the order of $2^{80}$ cryptographic operations to break. This level of security is higher than a 1024 bit RSA key, which requires roughly $2^{72}$ operations to break [33]. Choosing 80 bits will provide ample protection even against a determined attacker until about year 2015 (choosing 96 bits is expected to be secure until year 2035) [33].

#### 4.2.1 One-Way Hash Chains

*One-way hash chains*, or simply *one-way chains*, are a frequently used cryptographic primitive in the design of secure protocols. We create a one-way chain by selecting the final value at random, and repeatedly apply a one-way hash function $H$ to derive previous values. For example, to create a chain, we select $v_0$ at random, derive $v_1 = H[v_0]$, $v_2 = H[v_1]$, etc.

The required one-way property of the hash function $H$ makes it computationally intractable for an attacker to derive value $v_i$ knowing value $v_{i+1}$. More generally, given value $v_i$ of a one-way chain, an adversary cannot find an earlier value $v_j$ such that $H^{i-j}[v_j]$ equals $v_i$. (The notation $H^x[y]$ here means that we apply the hash function $H$ $x$ times on the input $y$, e.g., $H^2[y] = H[H[y]]$.) Even when value $v_i$ is released, a hash function that is *second pre-image collision resistant* prevents an adversary from finding $v_i' \neq v_i$ such that $H[v_i'] = v_{i+1}$.

The main property of values of a one-way chain is that once a receiver trusts that a value $v_i$ is authentic, it can derive all following values of the one-way chain by repeatedly computing the one-way function $H$, and it can authenticate that an earlier value $v_j$ also belongs to the one-way chain, by checking that $H^{i-j}[v_j]$ equals $v_i$. Moreover, an adversary cannot derive later values of the one-way chain, unless the creator of the chain already published them.

#### 4.2.2 Hash Trees

*Hash Trees* (also known as Merkle hash trees [38]) are an efficient mechanism which reduces the problem of authenticating a sequence of values $v_0, v_1, \ldots, v_{w-1}$ to authenticating a single value $r_0$. In SPV, we use hash trees for three purposes: to authenticate the values of the single-ASN private key, to authenticate several single-ASN public keys, and to authenticate the epoch public keys.

To construct the hash tree, we place the values $v_0, \ldots, v_{w-1}$ at the leaf nodes of a binary tree, as Figure 1 shows. (For simplicity we assume a balanced binary tree, so $w$ is a power of two.) We first blind all the $v_i$ values with a one-way hash function $H$ to prevent disclosing neighboring values in the authentication information (as we describe below), so $v_i' = H[v_i]$. We construct a hash tree over the values $v_0', \ldots, v_{w-1}'$ as follows. Each internal node of the binary tree is derived from its two child nodes. The derivation of a parent node $m_p$ from its left and right child nodes $m_l$ and $m_r$ is

[4] A slice is used to measure the size of an implementation in an FPGA.

[5] A block cipher is a pseudo-random permutation (PRP), and using a PRP to implement a PRF is secure as long as we use it fewer than $2^{\ell/2}$ times for each key [3], where $\ell$ is the block size in bits. With 128-bit AES, we can use this PRP up to $2^{64}$ times; we satisfy this requirement since we only use the PRP only up to $2^8$ times with any key.

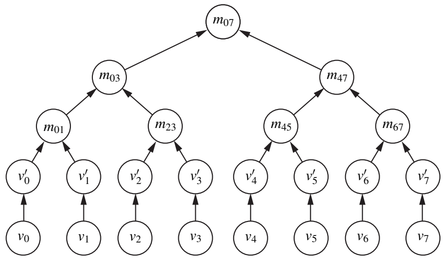


**Figure 1: Tree authenticated values**

$m_p = H[m_l \mid\mid m_r]$, where $||$ denotes concatenation. [6] We compute the levels of the tree recursively from the leaf nodes to the root node. Figure 1 shows this construction over the eight values $v_0, v_1, \dots, v_7$, e.g., $m_{01} = H[v'_0 \mid\mid v'_1]$, and $m_{03} = H[m_{01} \mid\mid m_{23}]$.

The root value of the tree enables authentication of all leaf nodes. To authenticate a value $v_i$ the sender discloses $i$, $v_i$, and all the sibling nodes of the nodes on the path from $v_i$ to the root node. The receiver can then use these nodes to recompute the values on the path up to the root, and if the recomputed root value matches the known root value, value $v_i$ is guaranteed to be authentic. For example, to authenticate value $v_2$ in Figure 1, the values $v'_3, m_{01}, m_{47}$ are required for recomputing the root. A receiver with the authentic root value $m_{07}$ can verify the equality:

$$m_{07} = H\left[ H\left[\, m_{01} \mid\mid H\left[\, H[\, v_2 \,] \mid\mid v'_3 \,\right]\right] \;\mid\mid m_{47} \right]$$

If the verification is successful, the receiver knows that $v_2$ is authentic. The extra $v'_0, v'_1, \dots, v'_7$ in Figure 1 are added to the tree to avoid disclosing (in this example) the value $v_3$ for the authentication of $v_2$.

### *4.2.3* One-Time Signatures

One-time signatures are based on efficient one-way functions[7] and are used to replace more expensive asymmetric signatures in cases where performance is critical. The drawback of one-time signatures is that if we sign multiple messages, the security degrades. The first instantiation of a one-time signature was by Lamport [32], subsequently researchers further developed these ideas [16, 39, 47, 50]. Most of these one-time signatures work by constructing a graph using one-way functions, where the verification value serves as the public key, and some randomly chosen values serve as the private key. For example, we could use the hash tree method we present above as a one-time signature to sign $t$ bits: we select $2^t$ random values $v_i$ (for example using a PRF $\mathcal{F}$ as outlined above: $v_i = \mathcal{F}_{\mathcal{X}}(i)$), and build a hash tree over those values. (In this section, we will use $t = 80$ as an example, since it corresponds to the $2^{80}$ security level we explained earlier). The root of the hash tree would become the public key, and all the $2^t$ values at the leaves would be the private key (or simply the random key $\mathcal{X}$ could also serve as the private key, as all other values are derived from it).

[6]The initial hash also prevents a shifting attack that is introduced through the concatenation of the arguments. Consider the case where we have two leaf nodes, $v_1$ and $v_2$. If we compute $H[v_1||v_2]$, an attacker could "shift" least significant bits from $v_1$ to become the most significant bits of $v_2$, thus breaking the authentication property. Because the initial hashing step produces hashes of equal size, this attack is not possible.

[7]A *trapdoor* is defined as a small amount of information that allows us to invert a one-way function. One-time signatures use one-way functions without trapdoors, because the only way to invert these functions is to construct a table with all inputs and outputs, which is not a small amount of information. Traditional signatures, such as RSA, are also one-way functions, but these functions have a trapdoor. For example in RSA, encryption with the public key is a one-way function, and the private key is the trapdoor information that allows us to invert the function.

This would allow us to sign message $M$ as follows. In the context of SPV, the message $M$ is an ASPATH suffix, e.g., $\langle A, B, C, D \rangle$. We compute a cryptographic hash of the message and keep the $t$ least significant bits: $[H[M]]_t = h$. We then disclose the value $v_h$ of the private key, along with the tree values necessary to recalculate the root value for verification (as we describe above in the hash tree section). Two approaches exist to attack this signature: the attacker inverts the one-way function to derive the private key, or the attacker finds another message with the same hash value. Assuming a secure one-way function with an output length of 80 bits, it is computationally intractable for an attacker to invert the one-way function. Hence, only the second attack is an option, which is to find a second message $M'$ such that the $t$-bit hash value is equal to that of message $M$: $[H[M]]_t = [H[M']]_t$. The probability that another message $M'$ has the same $t$-bit hash value as message $M$ is $1/2^t$, so an attacker would need to try $2^{t-1}$ messages on average until it could find another message with the same hash value $h$. This simple example illustrates a one-time signature, but it has a high computation and communication overhead compared to the provided security.

To improve on this scheme, we can disclose multiple leaf nodes to encode a signature. Assuming we have $n = 2^t$ leaf nodes, and we disclose $m$ leaf nodes for each signature, we could encode $\lfloor \log_2 \binom{n}{m} \rfloor$ bits—this encoding was proposed by Reyzin and Reyzin in their HORS signature [47]. In this encoding, we map the output of the hash function to one of the $\binom{n}{m}$ combinations.

We explain the HORS signature in more detail based on an example. Consider that we want to sign message $M$. In this simplified example, we use the parameters $n = 8$ and $m = 2$. To create the private key, the signer selects a random key $\mathcal{X}$, and derives the 8 values of the private key with the PRF $\mathcal{F}$ as follows: $v_i = \mathcal{F}_{\mathcal{X}}(i)$ for $1 \le i \le 8$. Next, the signer computes a hash tree over these values, as Figure 1 shows. The root value of the hash tree ($m_{07}$ in this case), serves as the public key. We assume that the verifier only knows the public key. To sign message $M$, the signer computes the hash $h_M = H[M]$ and derives the values to disclose from that by selecting two three-bit sequences from the hash. In this example, $H[M] = 011110\dots$ in binary notation, thus the signature will consist of the private key values $v_3$ and $v_6$. The signature will consist of values $v_3, v_6, v'_2, v'_7, m_{01}, m_{45}$. To verify the signature, the verifier recomputes the hash of the message as well as the root of the hash tree, and ensures that the recomputed root of the hash tree matches the public key $m_{07}$: based on $H[M]$ the verifier knows that the disclosed values of the private key are $v_3$ and $v_6$, and it verifies that $m_{07}$ equals $H[H[m_{01}||H[v'_2||H[v_3]]]||H[m_{45}||H[H[v_6]||v'7]]]$. If an attacker would want to forge this signature, it would have to invert the one-way function (which we assume is computationally impossible), or find another message that would also disclose values $v_3$ and $v_6$, which is easy in this case: $1/\binom{8}{2} = 1/28$.

We propose to use the HORS encoding in conjunction with a hash tree to achieve fast verification and a small size of the public key (the public key is simply the root node of the hash tree). This approach has a much better security/cost tradeoff than the simple scheme described above. Unfortunately, this approach also reduces security as more signatures are made: since each signature discloses $m$ values from the private key, an attacker with $r$ signatures has approximately $mr$ values. The probability that the attacker has any given value is at most $\frac{mr}{t}$, so the probability that an attacker can sign any given value is at most $\left(\frac{mr}{t}\right)^m$. In Section 5.1 we present a more sophisticated security analysis.

## 4.3 Basic ASPATH Protector

In this section, we present a new cryptographic mechanism, which we call an *ASPATH protector*. As we show later, this protector has some security weaknesses which we fix with a more sophisticated

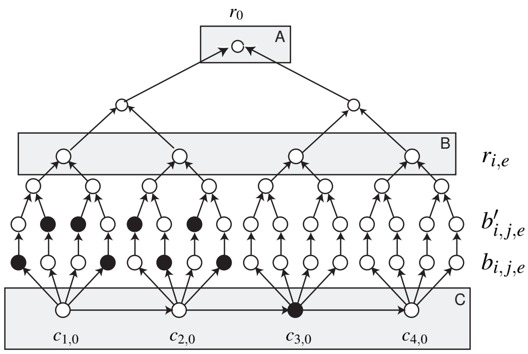


**Figure 2: This figure shows a diagram of one ASPATH protector capable of securing an ASPATH of length four. The shaded box labeled "A" highlights the epoch public key, box "B" highlights the four single-ASN public keys, and box "C" highlights the four single-ASN private keys that form a one-way hash chain. The black circles represent the ASPATH protector values that an AS sends with an UPDATE for one of its own prefixes to a peer; the first single-ASN signature (spanned by single-ASN private key $c_{1,0}$) signs its own ASN, the second single-ASN signature (spanned by $c_{2,0}$) signs the BGP peer's ASN, and the router reveals $c_{3,0}$ to allow further UPDATE propagation.**

structure in Section 4.4. The main purpose of the ASPATH protector is to secure ASPATHs from the truncation and modification attack.

**Properties.** Our efficient ASPATH protector achieves the following two properties. First, an attacker cannot claim a shorter route to a prefix than the length of the shortest route it has heard since the AS originating that prefix last advanced its epoch. Second, an attacker cannot modify the ASNs which have already been inserted into the ASPATH.

**ASPATH Protector Construction.** We now describe a cryptographic mechanism that enforces these properties. The intuition behind our scheme is to use a one-time signature scheme to sign each suffix of the ASPATH (that is, we sign the entire ASPATH at each AS traversed by an UPDATE) to make that suffix of the ASPATH immutable to later ASes that forward the UPDATE. We leverage one-time signatures to achieve ASPATH integrity, hash trees to enable authentication and verification of the one-time signatures, and one-way hash chains to reduce the size of the ASPATH protector. We describe these cryptographic mechanisms in more detail in Section 4.2.

To generate the ASPATH protectors, the AS selects a random key $\mathcal{X}$, and generates the seed values that span the individual ASPATH protectors for epoch $e$ using a PRF $\mathcal{F}$: $c_{1,e} = \mathcal{F}_{\mathcal{X}}(e)$.[8] The advantage of this construction is that the AS only needs to store the secret key $\mathcal{X}$, and can reconstruct any of the seed values. Alternatively, the AS could select all seed values at random, but would then have to store them all.

Each HORS one-time signature structure can be derived from one value, which we call the *single-ASN private key*. To secure an ASPATH with $\ell$ ASNs, we need $\ell$ one-time signatures. Thus, we randomly select $\ell$ single-ASN private keys for each epoch $e$: $c_{1,e},\ldots,c_{\ell,e}$. Each value $c_{i,e}$ is used to derive the $n$ nodes of the HORS one-time signature, from which $m$ values will be disclosed in a signature.[9] For each one-time signature, we derive $n$ values $b_{i,1,e},\ldots,b_{i,n,e}$ using a PRF $\mathcal{F}$ and the private key $c_{i,e}$ as the key to the PRF: $b_{i,j,e} = \mathcal{F}_{c_{i,e}}(j)$. To enable authentication of these values, we first blind these keys with a one-way function $b'_{(i,j,e)} = H[b_{(i,j,e)}]$,[10] and then we construct a hash tree over them. Each hash tree computed over each one-time signature has a root value $r_{i,e}$; we use $r_{i,e}$ to denote the root value that serves as *single-ASN public key* for the one-time signature with the single-ASN private key $c_{i,e}$. We then build a hash tree over the $r_{i,e}$ values, and use $r_e$ to denote the root value of that tree. We call $r_e$ an *epoch public key*, since it can be used to verify all the one-time signatures in the ASPATH protector within one epoch.

Each ASPATH protector would need to carry all the single-ASN private keys to enable subsequent ASes to sign in their ASN into the protector. To lower the communication overhead, we use a one-way chain to link the single-ASN private keys—this approach has the advantage that all subsequent single-ASN private keys can be derived from a previous single-ASN private key. Hence, value $c_{1,e}$ serves as the single-ASN private key for the first ASN, and as the seed value for the one-way hash chain which spans the subsequent single-ASN private keys $c_{2,e},c_{3,e},\ldots,c_{\ell,e}$ where $c_{i+1,e} = H[c_{i,e}]$.

Figure 2 shows one ASPATH protector, securing an ASPATH with up to four ASNs. The arrow from $c_{1,0}$ to $c_{2,0}$ in Figure 2 depicts the one-way function application: $c_{2,0} = H[c_{1,0}]$.

**ASPATH Protector Use and Verification.** In each epoch the owner of a prefix uses one ASPATH protector to announce its prefix in an UPDATE message. In the basic scheme, each one-time signature is used to sign one ASN into the ASPATH protector. As the ASPATH is passed in the UPDATE message, each AS uses the next single-ASN private key to sign its ASN into the protector, removing the current single-ASN private key $c_{i,e}$ by following the one-way chain one step and passing $c_{i+1,e} = H[c_{i,e}]$ to the next AS. Based on the multi-epoch public key, an AS can verify an ASPATH protector.

For example, consider three ASes with the following connectivity $A \leftrightarrow B \leftrightarrow C$. AS $A$ signs $H[A]$ with the HORS signature spanned by value $c_{1,0}$, and sends to $B$ the signature along with $c_{2,0}$.

We assume that $B$ has an multi-epoch public key to verify $r_0$, the root of the hash tree. $B$ can verify all the one-time signatures in the ASPATH protector by recomputing all one-time signatures and verifying that the final root value matches $r_0$. If successful, the AS knows that all one-time signatures are correct, if not successful, at least one of the values must be false.[11] More concretely, $B$ first verifies the correctness of the one-time signature, and checks that $H[A]$ is correctly encoded in the first HORS one-time signature. Based on $A$'s one-time signature, $B$ can compute value $r_{1,0}$, and based on value $c_{2,0}$ it computes $r_{2,0}$. $B$ then follows the one-way hash chain and computes $c_{3,0} = H[c_{2,0}],\ldots,c_{\ell,0}$ and from those single-ASN private keys it can compute $r_{3,0},\ldots,r_{\ell,0}$. Based on all these values, $B$ next computes the root of the hash tree, and verifies if it matches value $r_0$. If it matches, it knows that all signatures are correct.

$B$ then signs $H[\langle A,B\rangle]$ with the one-time signature spanned by $c_{2,0}$, and sends on $A$'s and $B$'s signature along with $c_{3,0}$ to $C$. To allow further propagation of the UPDATE, $B$ includes $c_{3,0}$, thus allowing $C$ to advertise the route after appending its identifier.

[8] Section 4.2 explains pseudo-random functions in more detail, and justifies why all our values are 80 bits long.

[9] In Section 5, we discuss our choice for $n$ and $m$ and the resulting security margin.

[10] The $b'$ values are used to prevent disclosing a neighboring $b$ value when publishing the $b$ values in a signature; Section 4.2.2 presents more details on this.

[11] Unfortunately, it is not possible to determine which signature is false; however, because the previous SPV speaker should have verified the signatures before sending them out, we can infer that the previous SPV speaker must be malicious. This reasoning does not apply, however, if the UPDATE is received from a legacy AS.

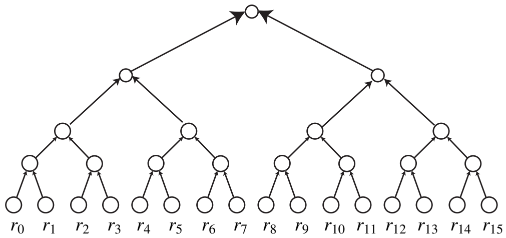


**Figure 3: This figure shows a hash tree constructed over the epoch public keys (root nodes of 16 ASPATH protectors). The root node of this tree is the multi-epoch public key, which is signed with the prefix public key to form the multi-epoch certificate.**

Figure 3 shows the hash tree that enables authentication of multiple ASPATH protectors. The root value can be used to authenticate the root value of each ASPATH protector, The internal nodes of the tree needed to authenticate values within a given ASPATH protector are included in the UPDATE.

By construction, our ASPATH protector secures the ASPATH against modifications, in particular against route shortening. One of the main advantages is the efficiency of authentication generation and verification, as we discuss in Section 5. In addition, the one-way hash function design enables several new properties not achieved by previously proposed mechanisms. Consider an AS that does not use SPV, but transparently forwards SPV values as part of the BGP UPDATE, unaltered. If the next AS implements SPV, it can construct the ASPATH protector *as if the previous AS implemented SPV*, as no AS-specific private key is involved in updating the ASPATH protector. This property is important for incremental deployment.

### 4.4 Advanced ASPATH Protector

The basic ASPATH protector has several drawbacks that we remedy in this section. We discuss each drawback in turn and describe our countermeasure. The collection of all the countermeasures make up the advanced ASPATH protector.

**Repeatable and Predictable Fraud.** As we discuss in Section *4.2.3*, one way to forge a one-time signature is to claim that a different message $M'$ was signed that has an identical hash as the legitimate message $M$ (so $H[M] = H[M']$), or that the encoding of $H[M']$ allows attacker to forge a valid signature given the disclosed values of multiple one-time signatures. For example, if $C$ receives the UPDATE from $B$ with ASPATH $\langle A,B\rangle$, it can replace $B$ with $C$ if $H[\langle A,B\rangle] = H[\langle A,C\rangle]$. Similarly, if $C$ receives two ASPATHs $\langle A,B\rangle$ and $\langle A,F\rangle$ for the same prefix, it learns more values of the second one-time signature and may be able to encode $H[\langle A,C\rangle]$. In both cases, if the condition is satisfied $C$ will always be able to forge a bogus ASPATH for all updates. As a countermeasure, we add the epoch number $e$ to each hash operation, so instead of computing $H[\langle A,B\rangle]$ we compute $H[e \mid\mid \langle A,B\rangle]$. This approach makes the probability of forgery independent between epochs, so an attacker cannot deterministically cheat. Thus, if an attacker can cheat with a low probability $p$, the probability that it can cheat each time in $x$ successive epochs is $p^x$.

**Single Malicious AS Fraud.** In the basic ASPATH protector, an attacker can easily forge a false ASPATH. Since the one-time signature is not binding the AS to sign its own ASN, a malicious AS can sign an arbitrary ASN into the ASPATH protector before sending it on to the next AS. However, this attack is easily prevented: the receiving AS needs to verify that the sending AS is the last ASN in the ASPATH.

Even with this rule, a malicious AS can still forge a false ASPATH as follows. Consider that malicious AS $C$ has two legitimate neighbor ASes $B$ and $D$ that both implement SPV. When $C$ receives a prefix from $B$ with the ASPATH $\langle A,B\rangle$, $C$ can send the forged ASPATH $\langle A,B,M,C\rangle$ to $D$. In the basic ASPATH protector, $C$ can simply use two one-time signatures to sign in $M$ and $C$.

To prevent this attack, we require that the previous AS already signs in the next AS into the ASPATH protector. (This approach is similar to the route attestations in S-BGP, where an AS delegates the right to forward an UPDATE to the following AS.) In the case we describe above, when $B$ forwards an UPDATE with ASPATH $\langle A,B\rangle$ to $C$, $B$ will sign $H[e \mid\mid \langle A,B,C\rangle]$ into the one-time signature. Moreover, when $C$ forwards the UPDATE, $D$ will verify that $C$'s signature encodes $H[e \mid\mid \langle A,B,C,D\rangle]$. It is now clear that $C$ cannot cheat any more, as it would need to create an ASPATH such as $\langle A,B,C,M,C\rangle$ which $D$ could discard because it contains a loop.

Because each AS now signs the next ASN into an single-ASN signature, we can optimize the efficiency of AS path prepending. In particular, we specify that a node that wishes to include its ASN more than once does not consume additional single-ASN signatures for the additional ASNs, but rather includes them when it signs in the next hop. For example, if $B$ wishes to use AS path prepending in the above example, then it signs $H[e \mid\mid \langle A,B,B,C\rangle]$ into the one-time signature.

**Multi-path Truncation Attack.** If a malicious AS receives multiple UPDATEs with ASPATHs of different length, it may be able to mount a *multi-path truncation attack*. For example, suppose a router has an authenticator for the ASPATH $\langle A,B,C,D\rangle$, and an authenticator for the ASPATH $\langle A,E,F,G,H,D\rangle$. From the first ASPATH, the router has the single-ASN private key for the ASPATH protector representing the fifth AS. The attacker can use that private key to modify the second ASPATH; for example, the attacker can change that ASPATH to $\langle A,E,F,G,D\rangle$, by using the single-ASN private keys from the first ASPATH to secure its own place, and the ASPATH public values from the second ASPATH to secure the path up to itself. However, the resulting truncated path will be one AS longer than if the malicious AS would simply forward the shorter ASPATH, but the longer (altered) ASPATH may get precedence due to routing policy. In this section, we discuss a novel technique for reducing the impact of an attacker who wishes to truncate longer ASPATHs based on information learned from shorter ASPATHs, which we call postmodification.

In our *postmodification* mechanism, we modify the signature such that the quality of a single-ASN signature degrades as the signature travels farther from the signer. In particular, we added an additional level of values into the ASPATH authenticator, to allow for degradation of the private values without converting them into public values. Previously, the hash tree was constructed over the $b'_{i,j,e}$ values, and now we use these $b'_{i,j,e}$ values as "semi-private" values for the purpose of postmodification and add another layer of values $b''_{i,j,e} = H[b'_{i,j,e}]$, trading off computation overhead with increased security.

We now describe the ASPATH postmodification technique in more detail, and then give a concrete example. As before, we reveal $m$ private values in each one-time signature. In addition, an AS degrades some private values of prior single-ASN signatures into semi-private values. We use $\mu_i$ to denote the number of private values that remain in the signature after $i$ hops. When a private value is removed, the corresponding semi-private value is added, such that each signature always has a total of $m$ values: $\mu_i$ private values and $m - \mu_i$ semi-private values. These $\mu_i$ values are monotonically decreasing: $m = \mu_0 \geq \mu_1 \geq \ldots \geq \mu_\infty = 0$.

We now give a concrete example of postmodification. We assume that $n = 4$, $m = 2$, $\mu_0 = 2$, $\mu_1 = 1$, $\mu_2 = 0$, and that we are in epoch $e = 0$. Consider four ASes with the connectivity

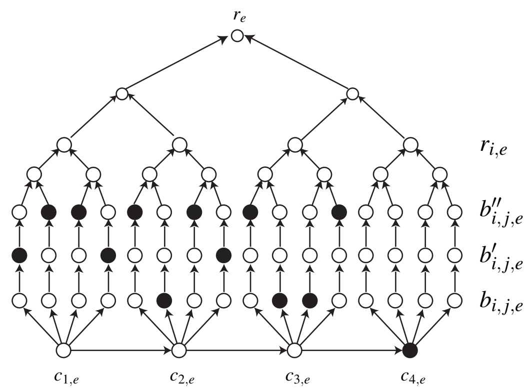


**Figure 4: This figure shows a diagram of one ASPATH Protector using the postmodification mechanism. The grey circles represent the values of the ASPATH Protector that an AS receives in an UPDATE: the origin AS signs the first two ASNs with the one-time signature spanned by $c_{1,e}$, the second AS signs the first three ASNs with the one-time signature spanned by $c_{2,e}$, etc. We describe the details on which values are disclosed in the text.**

$A \leftrightarrow B \leftrightarrow C \leftrightarrow D$. Figure 4 shows the ASPATH protector that AS $D$ receives from $C$. AS $A$ signs $h = H[0 \,||\, \langle A, B\rangle]$ with the HORS signature spanned by value $c_{1,0}$ (private values $b_{1,1,0}$ and $b_{1,4,0}$ were chosen, indicated by the black circles in the figure). These values are then sent along with $c_{2,0}$ to $B$ together with $b''_{1,2,0}$ and $b''_{1,3,0}$ (which are needed to authenticate those values).

$B$ computes the hash $h = H[0 \,||\, \langle A, B, C\rangle]$ which results in the selection of $b_{2,2,0}$ and $b_{2,4,0}$ for the private values. Since $\mu_0 - \mu_1 = 1$, $B$ needs to degrade one of the private values from the first authenticator as part of the postmodification. Again, based on the hash value $h$, $B$ deterministically selects one of the $b_{i,j,e}$ values, picks $b_{1,1,0}$ and thus forwards to $C$ $b'_{1,1,0} = H[b_{1,1,0}]$ along with the remaining private values of the ASPATH protector. Figure 4 shows the ASPATH authenticator that $D$ receives from $C$; where both private values from the first single-ASN signature have been converted to semi-private values ($\mu_2 = 0$ so $C$ degraded the value $b_{1,4,0}$), and $C$ also degraded the value $b_{2,4,0}$ from the second one-time signature.

This toy example should make it intuitively clear that downstream ASes receive smaller sets of the private values. As we evaluate in Section 5, such postmodification increases security by reducing the probability that an AS can perform the truncation attack.

### 4.5 Integration with BGP

To integrate our mechanisms with BGP, we include an optional, transitive path attribute that contains our authentication information for each hop in the UPDATE's ASPATH. We can use the *Partial* bit in the attribute flags to detect if there are any routers on the path that do not speak SPV, as a router that does not understand the SPV optional transitive attribute will set the Partial bit to indicate that it does not recognize the attribute.

**Aggregation** SPV supports aggregation in the same way as does S-BGP. In S-BGP, a BGP speaker aggregating two UPDATEs includes the authentication for both received UPDATEs and generates a new ASPATH for the aggregated prefix. In SPV, we cannot easily create a new ASPATH, so instead we choose one of the aggregated prefixes to carry the entire path. We then include the other prefix, except that we do not include the $c_{i,0}$ value from the bottom hash chain, thus preventing any node from readvertising the other prefix. For example, if AS $E$ receives two prefixes $x_1$ along path $\langle A, B, E\rangle$ and $x_2$ along path $\langle C, D, E\rangle$, it then chooses one of these prefixes, possibly $x_1$, to carry the entire path. It then prevents further propagation of $x_2$ by removing its bottom chain value $c_{i,0}$. When AS $E$ propagates the aggregated block to AS $F$, it sends $x_1$ with path $\langle A, B, E, F\rangle$ and $x_2$ with path $\langle C, D, E\rangle$. AS $F$ can see that the paths join at $E$, and that the aggregation is valid. Choosing a single branch in this way reduces the cost of authentication, without compromising security. As in Murphy's Internet-Draft [42], the aggregator field would be mandatory, and the common AS must match the ASN in the aggregator field.

SPV, like S-BGP, fully supports multihoming. As in BGP, a multihomed subscriber speaks BGP to its two or more providers. Each provider propagates these routes into the Internet, where each AS uses local policy to select a route.

**Securing Route Withdrawals** To secure route withdrawals, we rely on hop-by-hop authentication and policy. In particular, as in BGP and S-BGP, an SPV router $R$ should allow an AS $A$ to withdraw a route to prefix $C$ if and only if AS $A$ is a BGP peer and AS $A$ is $R$'s next-hop destination for packets sent to prefix $C$. That is, we only accept withdrawals from a valid next-hop router. We do not require that the AS prove that the withdrawn link is experiencing some physical problem because doing so is very difficult [27], and because it undermines our goals of allowing routers to keep their policy secret.

In S-BGP, route withdrawals are signed as part of the UPDATE message, which is significantly more expensive to verify than the hop-by-hop authentication used by SPV. For example, with 1024-bit RSA, a signature takes 401$\mu$s to verify, whereas a Message Authentication Code takes under 2$\mu$s.

**Expiration** Route announcements and withdrawals are often vulnerable to the *replay attack*, where a BGP speaker replays an ASPATH which it has previously heard. In particular, once a BGP speaker has legitimately heard authentication information for an ASPATH, it can replay that ASPATH and ASPATH protector, even after that route has been withdrawn. To reduce the impact of this attack, S-BGP provides replay protection through the use of a Timeout field, which is signed by the BGP speaker which originally advertised the prefix. Once this timeout expires, all S-BGP routers will withdraw this route; as a result, for seamless routing connectivity, each S-BGP speaker must readvertise each of its routes before the route's Timeout expires.

In SPV, we can prevent replays through the use of epochs: each epoch corresponds to a certain valid period, and each route is readvertised in each epoch. An epoch is an implicit Timeout: the advertised route times out after the epoch ends. The length of an epoch can be chosen in a way that provides higher security or lower overhead, but a minimum epoch length (for example, one day) should be enforced. Denial-of-Service attacks based on excessive epoch changing can then be mitigated, for example by reducing the priority of verifying new epochs in excess of three or four per day.

To prevent synchronized epoch changes from causing a flood of advertisements of different ASes, the boundaries between epochs should be chosen uniformly at random. One approach is to hash the prefix using a one-way hash function. The resulting number is taken to represent the fraction of an epoch to offset from some well-known time. For example, if the epoch is a day long, the prefix "128.32.0.0/16" hashes to `5fbe73c...`, which represents 8 hours, 58 minutes, and 33.58 seconds after the well-known time. If the well-known time is midnight UTC, then each of 128.32.0.0/16's epochs will begin at 08:58:33.58 UTC each day.

To avoid the need for time synchronization, a router accepts UPDATEs with an old epoch number for a fixed time after it first receives an authentic UPDATE with the new epoch number. This gives possibly better (or preferred) routes an opportunity to propagate through the network. In addition, if the fixed time is chosen to be sufficiently long (such as two hours), an attacker that does not control all the routes to a prefix is unlikely to be able to prevent the other advertisements from reaching other nodes for that long.

Both S-BGP and SPV require a timeout or epoch to prevent replay of old UPDATE messages. In general, since SPV is much more efficient than S-BGP for ASPATH protector generation and verification (as we evaluate in Section 5), SPV can operate with a shorter epoch (or Timeout) than can S-BGP. As a result, SPV can more readily defend against replay attacks. SPV also includes mechanisms that allow for correct operation even when BGP speakers are not time synchronized.

# 5 EVALUATION

We evaluate the security and efficiency of the SPV secure path vector routing protocol, and we contrast it with S-BGP. We first discuss the security of SPV, then we analyze the performance benefits.

Unless otherwise noted, performance results are driven by data from the Oregon route server [40] for the days of January 24–25, 2003, assuming connections to Level3 through AS 3356 and Cable and Wireless through AS 3561. These ASes were chosen to represent a modestly multihomed customer; the same performance and security levels should apply to any similarly connected customer. The specific period chosen includes the substantial BGP UPDATE traffic that resulted from the SQL Slammer worm. We chose that time to best reflect the security and performance of the routing protocol under heavy load.

## 5.1 Security Evaluation

To evaluate the security of our approach, we first examine how SPV prevents several attacks, such as falsely aggregating or deaggregating and falsely originating a route to a prefix. We then use statistical analysis to determine the effectiveness of our ASPATH protector and our truncation prevention schemes. Finally, we compare the security of SPV to that provided by S-BGP.

### *5.1.1* SPV Security against Attacks

**Security against Signature Forgery.** For the two most serious attacks that we want to protect, the attacker needs to alter one-time signatures: ASPATH modification (in which an attacker alters the ASNs in the ASPATH to cause a downstream BGP AS to choose a route it would not have otherwise chosen) and truncation (in which an attacker shortens the ASPATH to attract traffic).

In both cases the attacker needs to forge at least two one-time signatures. Consider a malicious AS $M$ receives the ASPATH $\langle A,B,C,M\rangle$. As we describe in Section 4.4, the first one-time signature encodes $H[e \mid\mid \langle A,B\rangle]$, the second one encodes $H[e \mid\mid \langle A,B,C\rangle]$, and the third one encodes $H[e \mid\mid \langle A,B,C,M\rangle]$. By inspection, it is clear that if $M$ wants to alter the ASPATH to $\langle A,B,F,M\rangle$, this change would affect two one-signatures that it would need to alter. Similarly, if $M$ would attempt to truncate the ASPATH and forward ASPATH $\langle A,B,M,G\rangle$ to AS $G$, it would also need to alter two one-time signatures. One exception to this two signature forgery requirement is that $M$ can replace itself with an arbitrary ASN and add itself at the end of the resulting path; for example, it could advertise the ASPATH $\langle A,B,C,H,M\rangle$. However, because no ASN is removed, we do not see this as more powerful than a grayhole attack, for reasons we discussed in Section 2.

We now analytically compute the security against signature forgery, and use these results to derive the parameters $n$ (number of private values per one-time signature) and $m$ (number of private values disclosed per one-time signature). Since the security of one-time signatures diminishes if the same private key is used for multiple signatures, we consider the case where the attacker learns $r$ signatures from the same private key. Given $r$ signatures, the expected number of private values that are disclosed is $s = n\cdot(1-(1-\frac{m}{n})^r)$.

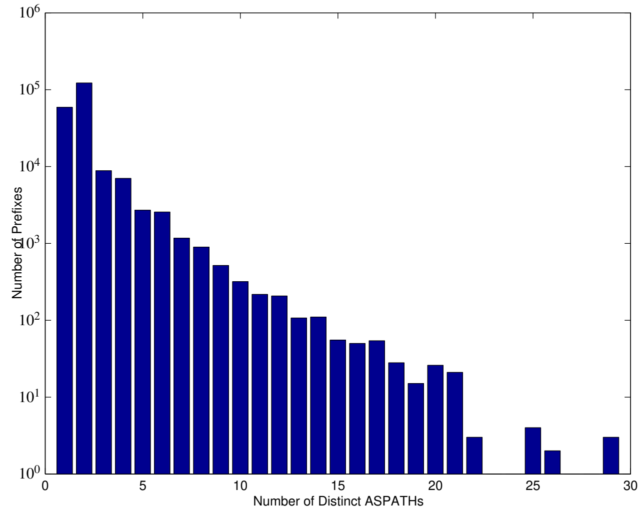


**Figure 5: Number of distinct routes to each prefix.**

With these parameters, the probability that an arbitrary message can be signed given the $s$ disclosed private values is $\binom{s}{m}/\binom{n}{m}$.

We aim for a forgery probability around $p = 2^{-11}$ to forge one digital signature. Since the attacker has to change two one-time signatures, and the event that the attacker can forge the signature are independent, the probability that an attacker can forge is at most $p^2 = 2^{-22}$. This may appear like a high probability, as an attacker could forge a signature after only $1/p^2 = 2^{22}$ tries. However, an attacker cannot try that many different signatures. In the case of truncation it would have to insert its own ASN as the last hop, since the following AS will ensure that. In the case of ASPATH modification, the attacker can try at most $2^{16}$ different ASNs; however, a neighbor might validate ASNs, in which case the attacker must choose from the $2^{14}$ active ASNs. In addition, if the neighbor knows (through prior configuration) which ASNs the attacker may be connected through, it can ensure that the ASN preceding the attacker is a neighbor of the attacker. Hence, the attacker's message space is so constrained that a $2^{-22}$ forgery probability may be acceptable. (If the size of an UPDATE message were not constrained to 4096 bytes, we could achieve much better security.)

Figure 5 shows a histogram of the number of distinct ASPATHs for each prefix received by our hypothetical AS connected to Level3 and Cable and Wireless on January 24, 2003. Each distinct ASPATH along which a prefix is received during a single epoch gives an attacker additional private and semi-private values. In our analysis, we conservatively do not make a distinction between private and semi-private values. We found that our AS had at most 15 routes to 99.9% of prefixes, so we chose parameters that gave relatively high security at 15 routes: when $n = 256$ and $m = 6$, $p = 1.2792\times 2^{-11}$ for an attacker with 15 distinct routes. The weighted average of $p$ across all prefixes is $1.8069\times 2^{-19}$. Unfortunately, because of limitations on the size of an UPDATE message, we can only carry 14 single-ASN signatures; that is, we can only authenticate 15 distinct ASNs. There are two ways to cope with this limitation. The first is to authenticate only the first 15 distinct ASNs. The second is to observe that future single-ASN signatures depend on previous ASNs, so keeping only the last 14 single-ASN signatures does not reduce security unless an attacker is willing to insert 14 bogus ASNs into the ASPATH.

**Aggregation.** Our aggregation mechanism is secure against policy violation because the AS performing the aggregation must include paths up itself, and must possess the $c_{i,e}$ value for adding itself to each ASPATH. If it has the $c_{i,e}$ values for each prefix, then it could continue authenticating both routes.

**Multiple Origin AS.** An attacker can attempt to advertise for a prefix with which it has no affiliation. In particular, the attacker will not have a certificate for that prefix (Section 4.1), so no legitimate SPV router will accept that advertisement.

#### 5.1.2 Resilience to Multi-Path Truncation

We evaluated the postmodification scheme for resilience to multi-path truncation, based on simulations on updates and routing tables obtained from Oregon RouteViews [40] for the day of January 24, 2003. We consider two hypothetical ASes. The first hypothetical AS is connected to Level3 and Cable and Wireless (as in the earlier evaluation); the second to three randomly selected ASes (234 (Blackrose Society), 293 (Energy Sciences Network), and 8297 (Teleglobe America)).

For a given prefix, an AS receives UPDATEs with several ASPATHs $\rho_1, \rho_2, \ldots, \rho_i$. When the AS wishes to shorten some $\rho_F$ by $h$ hops, it needs $\mu_{\ell_F - j} - \mu_{\ell_F + h - j}$ extra private values at each position $j$ from 1 to $\ell_F - h$, where $\ell_i$ is the length of the path $\rho_i$. At each of these positions, a different path $\rho_i$ will contribute some fraction of useful private values $\nu_{i,j}$. For example, if $j > \ell_i$, then $\nu_{i,j} = 1$ since all private values are released. When the $j$-hop prefix of $\rho_F$ and $\rho_i$ differ, $\nu_{i,j} = (\mu_{\ell_i - j})/t$, since the signatures will be uncorrelated. Otherwise, suppose $\rho_F$ and $\rho_i$ are identical until hop $H \geq j$. Then the same private values were included in the single ASN signature, and the same private values were discarded for the first $H - j$ hops, so $\nu_{i,j} = \mu_{\ell_i - j}/\mu_{H-j}$. Given all the ASPATHs that the attacker has, we compute the fraction of values it has at any hop $j$ as $1 - \prod_i 1 - \nu_{i,j}$, so the total probability of successful attack is $\prod_{j=1}^{\ell_F - h} (1 - \prod_i 1 - \nu_{i,j})^{\mu_{\ell_F - j} - \mu_{\ell_F + h - j}}$. [12]

When choosing $\mu_i$, we know that $\mu_0 = m$ and $\mu_\infty = 0$. The problem is choosing the rate at which $\mu$ declines. When $\mu_i = 0$, then if an attacker has a route $i$ hops shorter than the one being truncated, the attacker has all of the private values it needs. As a result, having $\mu_i$ decline slowly provides better security against an attacker with a much shorter route. However, because $\mu_i - \mu_{i+1}$ is smaller, an attacker needs to produce fewer private values to truncate a route, so an attacker with many distinct routes is more likely to succeed. Based on simulations with a small number of schemes, we chose $\mu_0 = 6, \mu_1 = \mu_2 = 5, \mu_3 = 4, \mu_4 = \mu_5 = 3, \mu_6 = 2, \mu_7 = \mu_8 = 1, \mu_9 = 0$.

For the first AS (connected to Level3 and Cable and Wireless), the attacker can only truncate an arbitrary ASPATH by one hop with probability $1.5599 \times 10^{-3}$. When a path of interest is less than 3 hops longer than the shortest path, the probability that the attacker can truncate a single hop is at most $1.3273 \times 10^{-7}$. When the shortest path is less than 6 hops shorter, the attacker can truncate with probability at most 0.4796%. If the shortest path is less than 10 hops shorter, the attacker truncates with probability at most 3.5900%, and the attacker always succeeds if the shortest path is more than 10 hops shorter.

In the other AS (connected to three random ASes), the attacker is also generally unable to truncate arbitrary ASPATHs. For example, when the shortest path is less than 6 hops shorter, the probability of successful truncation is at most $8.1894 \times 10^{-5}$, and when the shortest path is less than 8 hops shorter, the probability of successful truncation is at most 0.6978%. Between 8 and 10 hops shorter, the attacker succeeds at most 8.0707% of the time. Again, the attacker always succeeds if the shortest path is more than 10 hops shorter.

In general, ASPATHs with 6 extra unique ASNs are unlikely to be preferred routes. Furthermore, an attacker cannot generally select an arbitrary ASPATH to truncate, since it has a limited probability of success, and when it is able to succeed, such success is not repeatable in the next epoch. Finally, these evaluations show the probability of successfully truncating the last ASN before the attacker. Intuitively, these probabilities would be squared for truncating two ASNs, since twice as many extra private values would need to be included at each hop.

#### 5.1.3 Comparison to S-BGP

S-BGP is designed to provide security to a prefix only when the originating AS deploys S-BGP, and only to routers within a group of contiguous deployment that reaches the origin AS. SPV, on the other hand, attempts to provide security to any prefix when the originating AS deploys SPV. In this section, we explore the security achieved by SPV and S-BGP, and suggest how asymmetric cryptographic primitives could provide better properties than either, though at substantial cost.

When the origin AS doesn't deploy a secure routing protocol, S-BGP speakers can still sign attestations, ensuring that an S-BGP AS cannot be falsely added to the ASPATH. SPV does not achieve any properties in this case.

When an origin AS doesn't deploy secure routing, but all of its peers do, both S-BGP and SPV can, with the permission of the origin AS, secure the origin's prefix. In SPV, a single entity computes the private keys, and signs each peer's ASN into every UPDATE that would be protected by that private key. A peer is then unable to spoof being another peer. In S-BGP, threshold cryptography could be used, wherein peers together generate a key for the non-deploying AS, and use a separate protocol to sign UPDATEs for each other.

Within a contiguous group of deploying ASes, S-BGP ensures that each AS on the ASPATH has been transited by the UPDATE, and that ASNs cannot be dropped from the ASPATH. In SPV, an attacker controlling two ASes can insert bogus ASNs between its two ASNs. In addition, as an AS receives several UPDATEs from a single prefix within the same epoch, it can with increasing probability truncate the longer paths (but generally not the shorter ones) and insert itself into the path. Section 5.1.2 analyzes the effectiveness of multi-path truncation prevention.

An UPDATE that has traversed a non-deploying AS loses some amount of security under both schemes. In both S-BGP and SPV, the contiguous deploying prefix can be verified by any deploying router. In S-BGP, any speaker can remove any AS that follows a non-SBGP AS, but any subsequent S-BGP ASes must also be removed. In addition, any speaker can add a non-deploying AS after any other non-deploying AS, but any subsequent S-BGP ASes must be removed. In SPV, only the first SPV AS can add or remove an AS following a non-SPV AS; subsequent ASes cannot modify the ASPATH, except to the extent that an attacker can receive sufficient UPDATEs to compromise the security of the truncation prevention scheme.

To provide even stronger incremental deployment properties using asymmetric cryptography, we modify S-BGP such that each time a speaker sends an UPDATE for its own prefix, it also generates a public-private key pair for that UPDATE. Figure 6 illustrates this new protocol. For example, when AS $A$ originates an UPDATE for its own prefix to AS $B$, it generates a public-private key pair. We use $k_1^+$ to denote the public key and $k_1^-$ to denote the private key in this pair. It signs the UPDATE as in S-BGP, but also includes the public value $k_1^+$ in the signed fields. It also sends the private key $k_1^-$ to the next router; in this case, AS $B$. Each AS propagating

[12] This analysis conservatively ignores correlations that reduce the attacker's ability to attack, which occur when two ASPATHs are correlated, but are not correlated to the route being attacked.

$$
\begin{aligned}
A \rightarrow B: \quad & \langle \{\text{ASPATH}\, A, B; k_1^+\}_{K_A^-}, k_1^- \rangle \\
B \rightarrow C: \quad & \langle \{\text{ASPATH}\, A, B; k_1^+\}_{K_A^-}, \\
& \{\{\text{ASPATH}\, A, B, C; k_2^+\}_{K_B^-}\}_{k_1^-}, k_2^- \rangle \\
C \rightarrow X: \quad & \langle \{\text{ASPATH}\, A, B; k_1^+\}_{K_A^-}, \\
& \{\{\text{ASPATH}\, A, B, C; k_2^+\}_{K_B^-}\}_{k_1^-}, \\
& \{\{\text{ASPATH}\, A, B, C, X; k_3^+\}_{K_C^-}\}_{k_2^-}, k_3^- \rangle \\
X \rightarrow D: \quad & \langle \{\text{ASPATH}\, A, B; k_1^+\}_{K_A^-}, \\
& \{\{\text{ASPATH}\, A, B, C; k_2^+\}_{K_B^-}\}_{k_1^-}, \\
& \{\{\text{ASPATH}\, A, B, C, X; k_3^+\}_{K_C^-}\}_{k_2^-}, k_3^- \rangle \\
D \rightarrow M: \quad & \langle \{\text{ASPATH}\, A, B; k_1^+\}_{K_A^-}, \\
& \{\{\text{ASPATH}\, A, B, C; k_2^+\}_{K_B^-}\}_{k_1^-}, \\
& \{\{\text{ASPATH}\, A, B, C, X; k_3^+\}_{K_C^-}\}_{k_2^-}, \\
& \{\{\text{ASPATH}\, A, B, C, X, D, M; k_4^+\}_{K_D^-}\}_{k_3^-}, k_4^- \rangle
\end{aligned}
$$

**Figure 6: An example run of the stronger asymmetric protocol designed for incremental deployment described in Section 5.1.3. ASes $A, B, C, D$ are legitimate, deploying ASes, AS $X$ is a non-deploying AS, and AS $M$ is a malicious AS. If AS $M$ wants to remove AS $D$ from the ASPATH, it needs $k_3^-$ to sign the required certificate, in which case a non-deploying path exists from $X$ to $M$.**

that prefix generates a new public-private key pair, signs the entire UPDATE using both its private key and the private key it got from the previous AS, and passes only the new private key to the next AS (not the private key it got from the previous AS). For example, when $B$ propagates the UPDATE, it generates $k_2^+, k_2^-$, and signs the UPDATE and $k_2^+$ using both $K_B^-$ and $k_1^-$. This construct prevents an attacker from removing an ASN once the UPDATE has traversed a legitimate deploying router, unless the attacker receives the same UPDATE over a different path that does not contain a legitimate deploying router.

### 5.2 Performance Evaluation

In this section, we discuss the performance of SPV and contrast it with S-BGP. S-BGP can be used with any digital signature algorithm, we chose to compare with RSA 1024-bit modulus because RSA is the fastest algorithm for sequentially generating and verifying a signature. RSA verification is by far the fastest for all digital signature schemes, it is over 10 times faster than DSA for example. Since a signature is generated once and verified several times at each hop, RSA has a big advantage over other signature algorithms. While S-BGP can also cache verified signatures, the same trick can be applied to SPV and thus lower SPV overhead as well.

The security of 1024-bit RSA signatures requires on the order of $2^{72}$ operations to break [33], so on a first glance, the comparison appears unjust— why not compare with a version of RSA that also has a lower security margin? The reason is because RSA with a lower security margin requires a *one-time* effort to break, for example a 512-bit RSA modulus requires approximately $2^{50}$ operations to break [13]. In contrast, as long as the attacker cannot invert the hash function (which requires on the order of $2^{80}$ operations), an attacker can spoof an SPV update only with very low probability, and it cannot spoof that update during a different epoch without doing more computation.

**Computational Overhead** Figure 7 shows the cumulative distribution function of CPU usage required in each second in both our

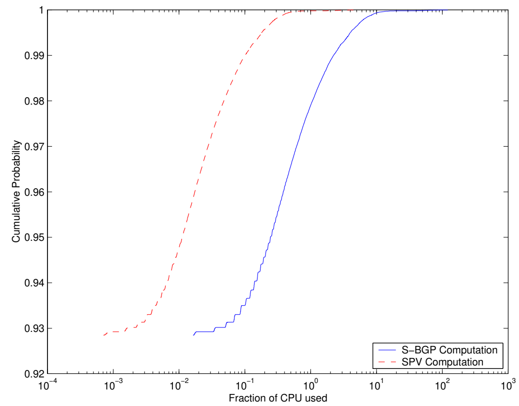


**Figure 7: CPU Utilization**

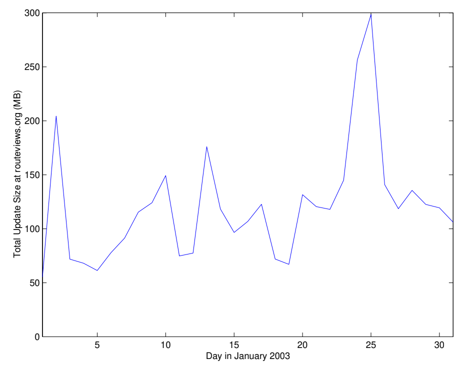


**Figure 8: BGP UPDATE traffic during an attack. Late January 24th, 2003, the "SQL Slammer" worm was unleashed. Most of the damage was seen on January 25th.**

protocol SPV and S-BGP. We built our hash function based on the Rijndael block cipher [14] with a 128-bit key and a 128-bit block size in the Matyas, Meyer, and Oseas construction [36, 5]. We evaluate S-BGP based on a 1024-bit RSA operation, as we discuss above. In evaluating both protocols we assume that the certificate signatures are cached. The CPU usage is based on our timings of a Pentium III running at 1GHz. In both SPV and S-BGP, CPU usage for verifying two streams is generally negligible; however, more connected parts of the network, the overhead of verification may be much higher. In such areas of the network, SPV may be more suitable because it is a factor of 22 faster.

The higher performance of SPV when implemented in hardware makes it more suitable for deployment in the core of the Internet. When an AS connects to many peers, the UPDATEs received over one second often take BGP over 100 seconds to process in software. When the Internet experiences attacks such as worms, the number of UPDATEs may increase significantly. Figure 8 shows the effects of the "SQL Slammer" worm on BGP routing traffic. Furthermore, as the Internet continues to grow, the number of prefixes will rise, thereby increasing the load caused by UPDATEs. Finally, SPV and S-BGP both support UPDATE expiration to limit the impact of replay attacks. To support this, the network will send a larger number of UPDATE packets. Processing this UPDATE load with S-BGP using general purpose CPUs is impractical. Because SPV is more efficient, it can authenticate routing for more sparsely connected networks. In more densely connected networks, hardware acceleration is required for both SPV and S-BGP, and a SPV accelerator will be much cheaper and faster than an S-BGP accelerator.

**Network Overhead** We evaluated the overhead of SPV relative to S-BGP. On the two days of our simulation, SPV incurred a factor of 2.731 more overhead than S-BGP.

## 6 Related Work

In this section, we review related work that was not previously described in Section 3. A number of secure distance vector [1, 30, 35, 53] and link state protocols [9, 20, 30, 45, 59] have been proposed, but we will focus our discussion on secure path vector routing protocols, since BGP is a path vector protocol.

In the wake of recent Internet attacks, many researchers focus on securing Internet protocols. As BGP is one of the fundamental Internet protocols, it is clear that securing BGP is a priority. The IETF has chartered the RPSEC working group to establish the security requirements for routing protocols [51]. In response to the formation of this working group, a number of researchers have drafted documents that outline BGP security requirements and BGP security vulnerabilities: Barbir, Murphy, and Yang describe attacks against BGP [2], as do Nordstrom and Dovrolis [43], and Convery et al. present a tree of possible attacks against BGP [11].

Pei, Zhang, and Massey present a framework for building resilient Internet routing protocols [44]. They present an analysis of the tradeoffs of various techniques.

In our previous work, we developed several new cryptographic mechanisms for securing ad hoc network routing protocols [22, 23], but these techniques were not applicable to BGP.

Although anecdotal evidence suggests that BGP attacks occur on a daily basis, so far misconfigurations account for the majority of route disruptions. Mahajan et al. find that each day, 200-1200 prefixes (0.2-1.0% of the BGP table size) are erroneous due to a misconfiguration each day, and that almost 3 in 4 of all new prefix advertisements are results of misconfiguration [34]. Secure routing protocols should also filter out updates due to misconfigurations, which will provide significant benefits.

## 7 Conclusions

Using purely symmetric cryptographic primitives to secure BGP is appealing for many reasons: software implementations enjoy at least a 20-fold speedup over digital signatures, and a hardware implementation would provide an additional 2.4-fold speedup.

In this paper we develop the SPV protocol, a protocol leveraging symmetric-key cryptography for securing against the truncation and modification attacks. SPV is configurable to allow tradeoffs between security and CPU usage.

SPV introduces three novel concepts to the design space of secure routing protocols: first, it includes private keys within the UPDATEs themselves; second, it does not authenticate the AS that inserts itself onto the path, but relies instead on hop-by-hop authentication to check this property; and finally, it provides security not by requiring overwhelming computational complexity but instead by limiting the number of options an attacker has for modifying critical routing information.

SPV is much faster than S-BGP, so SPV would perform better in periods of high BGP traffic. SPV can also use a shorter epoch time (or timeout) than can S-BGP, because it can more quickly authenticate new routes. In addition, when replay attacks are considered a threat, SPV allows for shorter timeouts than does S-BGP, and therefore can more effectively secure against replay attacks. SPV is an attractive alternative in the design space of secure interdomain routing protocols.

## 8 Acknowledgments

We gratefully acknowledge support, feedback, and fruitful discussions with Markus Adhiwiyogo, Prachi Gupta, Jorjeta Jetcheva, Steve Kent, Dave Maltz, David McGrew, Sandra Murphy, Srinivasan Seshan, Damon Smith, Dawn Song, Ion Stoica, Lakshminarayanan Subramanian, Gene Tsudik, Nick Weaver, Brian Weis, S. Felix Wu, Jibin Zhan, and Hui Zhang. We would especially like to thank Jennifer Rexford for her excellent suggestions on how to improve the paper, and the anonymous reviewers for their insightful comments.

## References

[1] F. Baker and R. Atkinson. RIP-2 MD5 Authentication. Internet Request for Comment RFC 2082, Internet Engineering Task Force, January 1997.

[2] A. Barbir, S. Murphy, and Y. Yang. Generic Threats to Routing Protocols. Internet-Draft draft-ietf-rpsec-routing-threats-06, April 2004.

[3] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A Concrete Security Treatment of Symmetric Encryption: Analysis of the DES Modes of Operation. In *Proceedings of the 38th Symposium on Foundations of Computer Science (FOCS)*, 1997.

[4] S. Bellovin. Security Problems in the TCP/IP Protocol Suite. *Computer Communication Review*, 19(2):32–48, April 1989.

[5] John Black, Phillip Rogaway, and Tom Shrimpton. Black-Box Analysis of the Block-Cipher-Based Hash-Functions Constructions from PGV. In *Advances in Cryptology (CRYPTO 2002)*, 2002.

[6] D. Boneh and M. Franklin. Identity-Based Encryption from the Weil Pairing. In *Advances in Cryptology — CRYPTO '2001*, pages 213–229, 2001.

[7] K. A. Bradley, S. Cheung, N. Puketza, B. Mukherjee, and R. A. Olsson. Detecting Disruptive Routers: A Distributed Network Monitoring Approach. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 115–124, May 1998.

[8] J. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A Digital Fountain Approach to Reliable Distribution of Bulk Data. In *Proceedings of ACM SIGCOMM '98*, pages 56–67, 1998.

[9] S. Cheung. An Efficient Message Authentication Scheme for Link State Routing. In *13th Annual Computer Security Applications Conference*, pages 90–98, 1997.

[10] S. Cheung and K. Levitt. Protecting Routing Infrastructures from Denial of Service Using Cooperative Intrusion Detection. In *The 1997 New Security Paradigms Workshop*, pages 94–106, September 1998.

[11] S. Convery, D. Cook, and M. Franz. An Attack Tree for the Border Gateway Protocol. Internet-Draft draft-ietf-rpsec-bgpattack-00, February 2004.

[12] S. Crosby and D. Wallach. Denial of Service via Algorithmic Complexity Attacks. In *Proceedings of the 11th USENIX Security Symposium*, pages 29–44, August 2003.

[13] Security of E-commerce threatened by 512-bit number factorization. `http://www.cwi.nl/~kik/persb-UK.html`, August 1999. CWI press release.

[14] J. Daemen and V. Rijmen. AES Proposal: Rijndael, March 1999.

[15] A. Daly and W. Marnane. Efficient architectures for implementing montgomery modular multiplication and RSA modular exponentiation on reconfigurable logic. In *Tenth ACM International Symposium on Field-Programmable Gate Arrays*, pages 40–49, February 2002.

[16] S. Even, O. Goldreich, and S. Micali. On-line/off-line digital signatures. In *Advances in Cryptology — CRYPTO '89*, pages 263–277, 1990.

[17] N. Feamster and H. Balakrishnan. Verifying the Correctness of Wide-Area Internet Routing. Technical Report MIT-LCS-TR-948, MIT, May 2004.

[18] O. Goldreich, S. Goldwasser, and S. Micali. How to Construct Random Functions. *Journal of the ACM*, 33(4):792–807, October 1986.

[19] G. Goodell, W. Aiello, T. Griffin, J. Ioannidis, P. McDaniel, and A. Rubin. Working around BGP: An Incremental Approach to Improving Security and Accuracy in Interdomain Routing. In *Proceedings of NDSS 2003*, February 2003.

[20] R. Hauser, A. Przygienda, and G. Tsudik. Reducing the Cost of Security in Link State Routing. In *Proceedings of NDSS '97*, pages 93–99, February 1997.

[21] A. Heffernan. Protection of BGP Sessions via the TCP MD5 Signature Option. RFC 2385, August 1998.

[22] Y.-C. Hu, D. B. Johnson, and A. Perrig. SEAD: Secure Efficient Distance Vector Routing for Mobile Wireless Ad Hoc Networks. *Ad Hoc Networks*, 1(1):175–192, 2003.

[23] Y.-C. Hu, A. Perrig, and D. B. Johnson. Efficient Security Mechanisms for Routing Protocols. In *Proceedings of NDSS 2003*, February 2003.

[24] Y.-C. Hu, A. Perrig, and D. B. Johnson. Packet Leashes: A Defense against Wormhole Attacks in Wireless Ad Hoc Networks. In *Proceedings of the Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2003)*, April 2003.

[25] S. Kent and R. Atkinson. IP Encapsulating Security Payload (ESP). Internet Request for Comment RFC 2406, Internet Engineering Task Force, November 1998.

[26] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. Internet Request for Comment RFC 2401, Internet Engineering Task Force, November 1998.

[27] S. Kent, C. Lynn, J. Mikkelson, and K. Seo. Secure Border Gateway Protocol (S-BGP) — Real World Performance and Deployment Issues. In *Proceedings of NDSS 2000*, pages 103–116, February 2000.

[28] S. Kent, C. Lynn, and K. Seo. Secure Border Gateway Protocol (S-BGP). *IEEE Journal on Selected Areas in Communications*, 18(4):582–592, April 2000.

[29] C. Kruegel, D. Mutz, W. Robertson, and F. Valeur. Topology-based Detection of Anomalous BGP Messages. In *Proceedings of the Symposium on Recent Advances in Intrusion Detection (RAID)*, September 2003.

[30] B. Kumar. Integration of Security in Network Routing Protocols. *SIGSAC Review*, 11(2):18–25, 1993.

[31] B. Kumar and J. Crowcroft. Integrating security in inter domain routing protocols. *Computer Communication Review*, 23(5):36–51, October 1993. Dept. of Comput. Sci., Univ. Coll. London, UK.

[32] L. Lamport. Constructing Digital Signatures from a One-Way Function. Technical Report SRI-CSL-98, SRI International Computer Science Laboratory, October 1979.

[33] A. Lenstra and E. Verheul. Selecting Cryptographic Key Sizes. *Journal of Cryptology*, 14(4):255–293, 2001.

[34] R. Mahajan, D. Wetherall, and T. Anderson. Understanding BGP Misconfiguration. In *ACM Sigcomm 2002*, August 2002.

[35] G. Malkin. RIP Version 2. Internet Request for Comment RFC 2453, Internet Engineering Task Force, November 1998.

[36] S. Matyas, C. Meyer, and J. Oseas. Generating Strong One-Way Functions with Cryptographic Algorithm. *IBM Technical Disclosure Bulletin*, 27:5658–5659, 1985.

[37] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press Series on Discrete Mathematics and its Applications. CRC Press, 1997.

[38] R. Merkle. Protocols for Public Key Cryptosystems. In *1980 IEEE Symposium on Security and Privacy*, 1980.

[39] R. Merkle. A Digital Signature Based on a Conventional Encryption Function. In *Advances in Cryptology — CRYPTO '87*, pages 369–378, 1988.

[40] D. Meyer. Route Views Project Page. `http://www.routeviews.org`.

[41] S. A. Misel. Wow, AS7007! NANOG mail archives, `http://www.merit. edu/mail.archives/nanog/ 1997-04/msg00340.html`, 1997.

[42] S. Murphy. BGP Security Protections. Internet-Draft draft-murphy-bgp-protect-01, October 2002.

[43] O. Nordström and C. Dovrolis. Beware of BGP Attacks. *ACM Computer Communications Review*, 34(2):1–8, April 2004.

[44] D. Pei, D. Massey, and L. Zhang. A Framework for Resilient Internet Routing Protocols. *IEEE Network*, 18(2):5–12, April 2004.

[45] R. Perlman. *Interconnections: Bridges and Routers*. Addison-Wesley, 1992.

[46] Y. Rekhter and T. Li. A Border Gateway Protocol 4 (BGP-4). RFC 1771, March 1995.

[47] L. Reyzin and N. Reyzin. Better than Biba: Short One-Time Signatures with Fast Signing and Verifying. In *Information Security and Privacy — 7th Australasian Conference (ACSIP 2002)*, July 2002.

[48] R. L. Rivest. The MD5 Message-Digest Algorithm. RFC 1321, April 1992.

[49] R. L. Rivest, A. Shamir, and L. M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[50] P. Rohatgi. A Compact and Fast Hybrid Signature Scheme for Multicast Packet Authentication. In *Proceedings of ACM CCS '99*, pages 93–100. ACM Press, November 1999.

[51] Routing Protocol Security Requirements (rpsec). IETF working group, `http://www.ietf.org/html.charters/rpsec-charter.html`, 2004.

[52] B. R. Smith and J.J. Garcia-Luna-Aceves. Securing the Border Gateway Routing Protocol. In *Global Internet'96*, pages 81–85, November 1996.

[53] B. R. Smith, S. Murthy, and J.J. Garcia-Luna-Aceves. Securing Distance Vector Routing Protocols. In *Proceedings of NDSS '97*, pages 85–92, February 1997.

[54] J. W. Stewart. *BGP4: inter-domain routing in the Internet*. Addison-Wesley, 1999.

[55] L. Subramanian, V. Roth, I. Stoica, S. Shenker, and R. Katz. Listen and Whisper: Security Mechanisms for BGP. In *Proceedings of First Symposium on Networked Systems Design and Implementation (NSDI 2004)*, March 2004.

[56] N. Weaver. *The SFRA: A Fixed Frequency FPGA Architecture*. Ph.D. thesis, University of California, Berkeley, 2003.

[57] R. White. Deployment Considerations for Secure Origin BGP (soBGP), draft-white-sobgp-bgp-deployment-01.txt. Draft, Internet Engineering Task Force, June 2003. Available at `http://www.watersprings.org/pub/id/draft-white-sobgp-bgp-deployment-01.txt`.

[58] A. Yaar, A. Perrig, and D. Song. SIFF: A Stateless Internet Flow Filter to Mitigate DDoS Flooding Attacks. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2004.

[59] K. Zhang. Efficient Protocols for Signing Routing Messages. In *Proceedings of NDSS '98*, March 1998.