# PISKES: Pragmatic Internet-Scale Key-Establishment System

Benjamin Rothenberger*, Dominik Roos†, Markus Legner*, and Adrian Perrig*†
*ETH Zurich, Switzerland †Anapaya Systems
{benjamin.rothenberger,markus.legner,adrian.perrig}@inf.ethz.ch,roos@anapaya.net

## ABSTRACT

Denial-of-service attacks have become increasingly prevalent in the Internet. In many cases they are enabled or facilitated by the lack of source authentication—it is often easy for an attacker to spoof its own IP address and thus launch reflection attacks or evade detection. There have been attempts in the past to resolve this issue through filtering or cryptography-based techniques; however, there is still no sufficiently strong system in place today—all proposals either provide weak security guarantees, are not efficient enough, or lack incentives for deployment. In this paper we present PISKES, a pragmatic Internet-scale key-establishment system enabling first-packet authentication. Through the PISKES infrastructure, any host can locally obtain a symmetric key to enable a remote service to perform source-address authentication. The remote service can itself locally derive the same key with efficient cryptographic operations.

PISKES thus enables packet authentication for a wide variety of systems including high-throughput applications like DNS. We have implemented a prototype system that enables a DNS server to verify the source of every received packet within 85 ns, which is over 220 times faster than a system based on asymmetric cryptography. PISKES has been developed for the SCION secure Internet architecture but is also applicable to today's Internet. With its strong source-authentication properties and highly efficient operation it has the potential to finally bring network-layer authentication to the Internet.

## CCS CONCEPTS

• **Security and privacy** → **Source authentication**; **Denial-of-service attacks**; *Key management*; Hash functions and message authentication codes.

## KEYWORDS

Source Authentication, Denial-of-Service Prevention, Key Derivation, Key Distribution

**ACM Reference Format:**
Benjamin Rothenberger*, Dominik Roos†, Markus Legner*, and Adrian Perrig*†. 2020. PISKES: Pragmatic Internet-Scale Key-Establishment System. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security (ASIA CCS'20), October 5–9, 2020, Taipei, Taiwan.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3320269.3384743


*ASIA CCS '20, October 5–9, 2020, Taipei, Taiwan*

ACM ISBN 978-1-4503-6750-9/20/10…$15.00
https://doi.org/10.1145/3320269.3384743

## 1 INTRODUCTION

Many of the widespread problems in today's Internet stem from the lack of authentication; in particular, denial-of-service (DoS) attacks often use reflection and amplification techniques enabled by connectionless protocols like DNS or NTP and the possibility of source-address spoofing [36, 53, 67]. This issue has been tackled in the past through several different approaches: source-address filtering at the network edge [24], cookie-based challenge–response systems [14, 22], or client certificates and asymmetric cryptography [32, 62]. Unfortunately, these systems provide relatively weak guarantees (filtering and cookies) or introduce a substantial overhead and open up additional DoS vulnerabilities (asymmetric cryptography). We describe several widely known systems and their downsides in §3.1 and provide an extensive overview in §8. We conclude that to obtain strong guarantees with high efficiency on a per-packet basis, an authentication system based on *symmetric* cryptography is required. However, this approach introduces its own challenges: in order to authenticate already the first packet received from a host, we cannot rely on in-band protocols to negotiate keys; on the other hand, storing symmetric keys for all potential senders on each server is infeasible in an Internet-scale system.

Recently, a system to dynamically (re-)create keys for any host, DRKey, has been proposed in the context of path-validation system OPT [41] and the SCION Internet architecture [58]. In this work, we extend the ideas from DRKey to build PISKES, a pragmatic Internet-scale key-establishment system, which enables packet authentication for various services both in SCION and today's Internet. The core property we strive for is to enable a service to rapidly derive a symmetric key to perform network-address authentication for an arbitrary source host. This will enable services such as DNS or NTP to instantly authenticate the first request originating from a client, thus providing a defense against reflection-based DoS attacks. In addition, the key can be used to authenticate the payload of the request and reply, which is particularly useful for DNS which by default does not include any authentication. Our PISKES prototype system enables the server to derive the symmetric key within two AES operations, which corresponds to 18 ns on a commodity server platform, and authenticate the first packet within 85 ns on commodity hardware. It is clear that such speeds cannot be achieved with protocols based on asymmetric cryptography that require multiple messages to be exchanged to establish a shared session key. For example, PISKES outperforms RSA 1024–based source authentication by a factor of more than 220, even under the assumption that the service already knows the client's public key. In addition to providing highly efficient network address verification, PISKES can also be used to authenticate Diffie–Hellman (DH) keys in a protocol such as TCPcrypt [8].

The main ideas behind PISKES are as follows. Autonomous systems (ASes)[1] can obtain certificates for their AS number and IP address range from a public-key infrastructure (PKI)—SCION's control-plane PKI [66] in a SCION deployment or the Resource Public Key Infrastructure (RPKI) [11] in today's Internet. PISKES uses such a PKI to bootstrap its own symmetric-key infrastructure. PISKES key servers are set up in all deploying ASes and contact each other on a regular basis to set up symmetric keys between pairs of ASes. These symmetric keys are then used as a root keys to efficiently derive a hierarchy of symmetric per-host and per-service keys. Thanks to a hardware implementation of the AES block cipher on most modern CPUs (Intel, AMD, ARM), such a key derivation is about four to seven times faster than a single DDR4 DRAM memory fetch [28, 50]. Our approach ensures rapid key derivation on the server side, whereas a slower key fetch is required by the client to a local key server. This one-sidedness makes the source authentication for the receiving side as efficient as possible and ensures that PISKES does not introduce new DoS attack vectors. PISKES is incrementally deployable and provides immediate benefits to deploying entities.

A fundamental tradeoff in PISKES is the additional trust requirements of end hosts in their local AS: as the key server is able to derive the end-to-end symmetric key, this key cannot be used directly to achieve secrecy between two end hosts. However, the PISKES keys can be used for authentication that the source host indeed belongs to the claimed AS, which suffices to resolve DoS attacks as we demonstrate in this paper.

The main contributions of this work are the following:

- We develop a key-establishment infrastructure using a key hierarchy and dynamic key derivation as a scalable and highly efficient approach for global key establishment.
- We describe how PISKES can be integrated into the SCION architecture and deployed in today's Internet.
- We formally verify the security guarantees of the proposed key-establishment protocol using the Tamarin prover tool.
- We describe how first-packet authentication can be achieved based on this key hierarchy and provide a prototype implementation of PISKES that demonstrates its efficiency.

# 2 PROBLEM DEFINITION

Our goal in this work is to design a system for highly efficient global first-packet authentication. We emphasize that we aim at providing packet authentication at the network layer based on the network address (i.e., the IP adress in the current Internet or the combination of AS number and local address in SCION), and not based on a higher-level identity such as a domain name or webserver identity. This section describes the security properties we strive to achieve and the adversary model we consider.

## 2.1 PISKES Design Goals

*2.1.1 Functional and Security Properties.* We aim to design a system that allows servers to authenticate the source and optionally the payload of packets they receive. In particular, already the first packet received from a particular source should be authenticated, without relying on prior communication between the two parties (e.g., through handshake messages). To achieve the authentication properties, the cryptographic keys exchanged with the key-establishment protocol must remain secret, and entity authentication of participating parties must be enforced.

*2.1.2 Efficiency & Scalability.* A system that enables applications requiring high bandwidth and low latency must facilitate high-speed processing. To that end, it must use efficient operations to keep the processing and communication overhead for entities minimal. A simple calculation underscores the importance of efficiency: Consider a system that is required to authenticate packets with a bandwidth of 10 Gbit/s. If we assume minimal-size (64 B, 84 B including preamble and inter-frame gap) packets, this corresponds to processing 14.9 million packets per second or one packet every 67.2 ns. Even considering pipelining and parallelism, this implies that a single packet needs to be authenticated within several hundreds of nanoseconds, ruling out asymmetric cryptography, which is orders of magnitude slower [23].

In addition to the processing efficiency, the communication overhead and required state introduced by the system must remain small for a global-scale deployment. In particular, it is infeasible to require a server to store symmetric keys for all several billion other hosts in the Internet.

*2.1.3 Deployability.* The system should be deployable on both the current Internet and future Internet architectures such as SCION. Deployment of the system should add minimal complexity to the existing Internet infrastructure. Traffic from legacy entities should not be affected by the deployment of the system. Early adopters should instantly benefit from deployment.

## 2.2 Adversary Model

We consider an adversary that can deviate from the protocol and attempt to violate its security goals. As a basis, we assume the *Dolev–Yao* model [20], where the adversary can reside at arbitrary locations within the network. He can passively eavesdrop on messages and also actively tamper with the communication by injecting, dropping, delaying, or altering packets. However, the adversary has no efficient way of breaking cryptographic primitives such as signatures, pseudorandom functions (PRFs), and message authentication codes (MACs). Furthermore, we assume that there exists a secure channel between end hosts and a key server within the same AS (see §7.3.5 for further details). Assuming the mentioned capabilities, the goal of the adversary is to obtain cryptographic keys of other parties to forge authenticated messages.

By compromising an entity, the adversary learns all cryptographic keys and settings stored. He can also control how the entity behaves, including fabrication, replay, and modification of packets. We consider compromises of both *end hosts* and *network nodes* (including key servers).

# 3 BACKGROUND

In this section, we describe several previous attempts at providing authentication in the Internet (§3.1) and proposals for global symmetric-key distribution (§3.2) and discuss why all of them are insufficient to address the problems introduced in the previous

[1] We use the term AS for an independently administered network domain, e.g., an Internet service provider, a content distribution network, or a university network.

section. Here we focus on several representative and well-known systems—an exhaustive overview of related work is provided in §8.

## 3.1 Authentication Systems

*3.1.1 Source Address Validation.* Source address validation (SAV), also known as best current practice (BCP) 38 [24], is not an authentication system in the strict sense but is still often considered a solution to source-address spoofing in the Internet. With SAV, ASes monitor traffic originating from their own hosts and filter out packets with a source address outside their own address space. However, due to incentive misalignments,[2] the adoption of SAV has been slow and a recent study found that many ASes still do not employ it in their networks [46]. Furthermore, it is impossible to determine from the outside if a particular AS employs SAV or if a particular packet originated from an AS that employs SAV as it does not carry any proof of authenticity. For an external service it is therefore impossible to filter traffic based on whether it originated from an AS employing SAV. Even with a full deployment of SAV in the Internet, on-path adversaries would still be able to spoof the source of packets and SAV thus provides very weak security properties. There exists a wide range of other filtering techniques with similarly limited properties [4, 21, 34, 43, 56].

*3.1.2 Cookies.* Several protocols, including TLS [63], IKEv2 [38], and DNS [22] define a cookie mechanism to provide a weak form of source authentication. The basic mechanism for these systems is similar: Upon receiving a request, the server replies to the sender with a cookie that encodes the request parameters without allocating state or processing the request. Only after receiving this cookie back from the source, the request is processed. Compared to SAV, cookies have the advantage that they can be enforced by services without relying on Internet service providers (ISPs) to perform filtering. However, cookies introduce additional latency of one round-trip time (RTT) and are still susceptible to spoofed packets by on-path adversaries.

*3.1.3 Client Certificates.* Strong authentication properties can be achieved through asymmetric cryptography and client certificates. These are supported, for example, by TLS [63] and DTLS [64]. However, authentication using client certificates requires expensive asymmetric cryptography in violation of our efficiency requirements (§2.1.2). Furthermore, these systems cannot authenticate the first packet and are vulnerable to signature-flooding attacks.

## 3.2 Key-Distribution Systems

*3.2.1 Passport.* Passport [44] provides mechanisms to establish shared keys between any pair of ASes based on a DH key exchange piggybacked on BGP messages. It relies on a secure routing system to ensure the authenticity of the shared keys, which can subsequently be used to authenticate the source of packets at the network layer. For our purposes (see §2), Passport by itself is inadequate for several reasons: (i) it only enables authentication at the AS level, (ii) it requires authenticating systems to keep a store of symmetric keys for all ASes (currently approximately 68 000 [6]), (iii) it has no mechanism to delegate keys to certain services. Other systems, such as Kerberos [54], are reviewed in §8.

[2]The costs of deploying SAV are paid by an AS itself while its benefits are experienced by the rest of the Internet.

**Table 1: Notation used in this paper.**

| | |
|---|---|
| $\|$ | bitstring concatenation |
| $A, B$ | autonomous systems (ASes) identified by AS number (ASN) |
| $H_A, H_B$ | end hosts identified by IP address |
| $KS_A, KS_B$ | key servers located in a specific AS |
| $SV_A$ | AS $A$'s local secret value |
| $SV_A^p$ | AS $A$'s local secret value for protocol $p$ |
| $\tilde{K}_\bullet^p$ | symmetric key derived (indirectly) from $SV^p$ |
| $K_{A\to B}$ | symmetric key between ASes $A$ and $B$, derived from $SV_A$ |
| $K_{A\to B}^p$ | symmetric key between ASes $A$ and $B$ for protocol $p$ |
| $K_{A\to B:H_B}^p$ | symmetric key between AS $A$ and end host $H_B$ in AS $B$ for protocol $p$ |
| $K_{A:H_A\to B:H_B}^p$ | symmetric key between end host $H_A$ in AS $A$ and end host $H_B$ in AS $B$ for protocol $p$ |
| $\mathsf{H}(\cdot)$ | non-cryptographic hash operation |
| $\mathsf{MAC}_K(\cdot)$ | message authentication code using key $K$ |
| $\mathsf{PRF}_K(\cdot)$ | pseudorandom function using key $K$ |
| $\{X\}_{PK_A}$ | public-key encryption of $X$ using AS $A$'s public key |
| $\{X\}_{PK_A^-}$ | public-key signature over $X$ using AS $A$'s private key |

*3.2.2 DRKey.* Dynamically Recreatable Keys (DRKeys) have been proposed to efficiently derive and distribute symmetric shared keys between routers and end hosts in the context of Origin and Path Trace (OPT) [41], a system providing path validation. The system has later been generalized and embedded in the SCION Internet architecture [58]. DRKey's fundamental idea is that each AS $A$ can efficiently derive a key hierarchy starting from a secret value $SV_A$, providing keys shared with other ASes, $K_{A\to B}$, and end hosts, $K_{A\to B:H_B}$. By periodically exchanging the keys $K_{A\to B}$ between ASes, from which host-level keys can be derived, DRKey enables an efficient global distribution of symmetric keys.

DRKey fulfills most of our requirements to a key-distribution system and thus provides the basis of PISKES. However, PISKES refines and extends the existing DRKey system [58] in several significant ways: (i) PISKES modifies DRKey to make it applicable to the current Internet in addition to SCION; (ii) it adds efficient mechanisms to delegate specific keys to services in an AS; (iii) it specifies many of its important practical aspects in further detail; and (iv) it fixes recently discovered vulnerabilities of DRKey's key-exchange mechanisms due to an inadequate application of signatures [33].

# 4 KEY DERIVATION AND DISTRIBUTION

In this section, we present the key-derivation and -distribution mechanisms used for PISKES. This is based on the DRKey system [58], but we significantly extend it with additional delegation mechanisms and other optimizations, see also §3.2.2. Furthermore, we also formally model and verify security properties of this key-distribution system, see §7.1.

We first provide a high-level overview to convey an intuition of the operation of our system. Figure 1 shows the basic use case of PISKES, where a host $H_A$ in AS $A$ desires to communicate with a server $S_B$ in AS $B$, and $S_B$ wants to authenticate the network

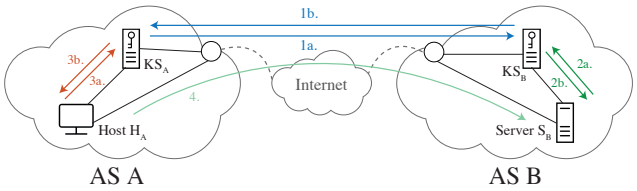


**Figure 1: Basic topology and key-establishment procedure for communication between an end host $H_A$ in AS $A$ and a server $S_B$ in AS $B$. ASes $A$ and $B$ have deployed the key servers $KS_A$ and $KS_B$, respectively.**

address of $H_A$ using a symmetric key. Table 1 lists the notation used in the remainder of this paper.

ASes that want to benefit from PISKES need to set up one or multiple key servers. We assume that ASes $A$ and $B$ have deployed $KS_A$ and $KS_B$, respectively. Each AS randomly selects a local secret value, $SV_A$ and $SV_B$, which is only shared with trustworthy entities (in particular, the key servers) in the *same* AS, it is never shared outside the AS. Since hosts are typically not trustworthy, they would not obtain the local secret value; however, in some cases a trusted router or server could obtain it as we are going to assume in this overview for simplicity—later we describe how to delegate symmetric values to less trusted services. The secret value will serve as the root of a symmetric-key hierarchy, as illustrated in Figure 2, where keys of a level are derived from keys of the preceding level.[3] In PISKES, the keys are derived using an efficient PRF [37, p. 331].

The key derivation used by $KS_B$ in our example is $K_{B\to A} = \mathsf{PRF}_{SV_B}(A)$. Thanks to the key-secrecy property of a secure PRF, $K_{B\to A}$ can be shared with another entity without disclosing $SV_B$. Note that the arrow notation indicates the secret value used to derive the key. Thus, $K_{B\to *}$ would typically be used if AS $B$ is on the performance-critical side, where $*$ denotes the set of remote ASes. To continue with our example depicted in Figure 1, $KS_A$ will prefetch keys $K_{*\to A}$ from key servers in other ASes, including $K_{B\to A}$ from $KS_B$. In Figure 1, this action is illustrated with the arrows labeled 1a and 1b. Also, we assume that server $S_B$ is trusted and can thus obtain the secret value $SV_B$, indicated with the arrows labeled 2a and 2b. When $H_A$ desires to authenticate to $S_B$, it contacts its local key server $KS_A$ and requests key $K_{B:S_B\to A:H_A}$, which $KS_A$ can locally derive from $K_{B\to A}$ (arrows 3a and 3b). $H_A$ can now directly use this symmetric key for authenticating to $S_B$ (arrow 4).

The important property of PISKES is that $S_B$ can rapidly derive $K_{B:S_B\to A:H_A}$, by using $SV_B$ and performing two PRF operations. This illustrates an important design aspect of PISKES: The one-wayness of the key-derivation function allows a key server to delegate key derivation to specific entities. Thus, the key-derivation process exhibits an asymmetry, meaning that the delegated entity $S_B$ can directly derive a required key, whereas host $H_A$ is required to fetch the corresponding key from its local key server. As opposed to other systems that rely on a dedicated server for key generation and distribution (such as Kerberos [54]), this delegation mechanism allows entities to directly obtain a symmetric key without communication to the key server.

[3] We emphasize that this key hierarchy is different from a PKI key hierarchy: in PISKES, higher-level keys are *derived* from a lower-level key, whereas in PKI hierarchies private keys sign public-key certificates of the next entity in the certificate chain.

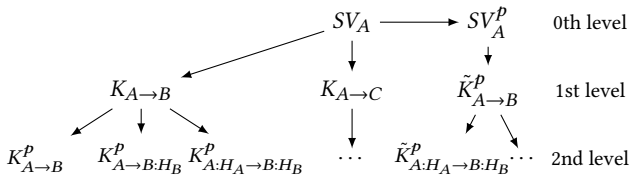


**Figure 2: PISKES multi-level key hierarchy derived by AS $A$. The notation $K_1 \to K_2$ indicates that $K_2$ is derived from $K_1$ through a pseudorandom function.**

## 4.1 Assumptions

The design of PISKES is based on the following assumptions:

- There exists an AS-level PKI, that authenticates the public key of an asymmetric key pair $(PK_E, PK_E^{-1})$ for each participating AS $E$ and certifies its network resources; we rely on the SCION control-plane PKI [66] certifying AS numbers for a deployment in SCION and on RPKI [11] certifying AS numbers and IP address ranges for a deployment in today's Internet.
- To verify the expiration time of keys and messages, PISKES relies on synchronization among ASes with a precision on the order of several seconds. This can be achieved using a secure time-synchronization protocol such as Roughtime [27].
- We assume that there exists an authentication mechanism for end hosts within an AS. This is needed for access control and is further discussed in §7.3.5.

## 4.2 Key Hierarchy

The PISKES key-establishment framework uses a key hierarchy consisting of three levels, illustrated in Figure 2.

*4.2.1 0th-Level (AS-internal).* On the zeroth level of the hierarchy, each AS $A$ randomly generates a local *AS-specific secret value* $SV_A$. The secret value represents the per-AS basis of the key hierarchy and is renewed frequently (e.g., daily). In addition, an AS can generate *protocol-specific* secret values

$$SV_A^p := \mathsf{PRF}_{SV_A}(\text{“}p\text{”}) \tag{1}$$

for an arbitrary protocol $p$, where “$p$” is its ASCII encoding. The purpose of these values is that they can be shared with specific services, such as DNS servers, that cannot be trusted with $SV_A$ but should still be able to efficiently derive additional keys. As this construction introduces additional communication and storage overhead, only widely used protocols such as DNS or NTP would have their own secret values.

*4.2.2 1st-Level (AS-to-AS).* By using key derivation, an AS $A$ can derive different symmetric keys using a PRF (e.g., based on AES) from the single local secret value $SV_A$ or a protocol-specific secret value $SV_A^p$. These derived keys, which are shared between AS $A$ and a second AS $B$, form the first level of the key hierarchy and are called *first-level* keys:

$$K_{A\to B} = \mathsf{PRF}_{SV_A}(B)\,. \tag{2}$$

The input to the PRF is the (globally unique) AS number of AS $B$. If a protocol-specific secret value $SV_A^p$ exists, protocol-specific

**Table 2: Second-level keys and their corresponding requests to the local key server (EH = end host).**

| Desc. | Key | Request Format |
|---|---|---|
| AS → AS | $K^{p}_{A\to B} = \mathrm{PRF}_{A\to B}(\text{“}p\text{”})$ | $\{0, \textit{req.ID}, p, A, B\}$ |
| AS → EH | $K^{p}_{A\to B:H_B} = \mathrm{PRF}_{A\to B}(\text{“}p\text{”} \parallel H_B)$ | $\{1, \textit{req.ID}, p, A, H_B\}$ |
| EH → EH | $K^{p}_{A:H_A\to B:H_B} = \mathrm{PRF}_{A\to B}(\text{“}p\text{”} \parallel H_A \parallel H_B)$ | $\{2, \textit{req.ID}, p, H_A, H_B\}$ |

first-level keys can be derived as

$$\tilde{K}^{p}_{A\to B} = \mathrm{PRF}_{SV^{p}_{A}}(B)\,. \tag{3}$$

The general first-level keys and those derived from protocol-specific secret values are periodically exchanged between key servers of different ASes, see §4.3.1.

Note that, instead of deriving first-level keys from a common secret value, it is also possible to use the keys defined by Passport. Protocol-specific first-level keys could then be derived from the standard first-level keys. However, this approach would require all entities that want to use PISKES to store and frequently update a key for each AS instead of being able to derive them from a single secret value. In the remainder of the paper, we therefore focus on the DRKey-inspired key hierarchy with explicit key-exchange messages.

*4.2.3 2nd-Level (AS-to-host, host-to-host).* Using the symmetric keys of the first level of the hierarchy, *second-level keys* are derived to provide symmetric keys to hosts within the same AS. Second-level keys can be established between a pair of AS infrastructure nodes (such as border routers or servers), end hosts, or a combination of both. For example, a key between end hosts $H_A$ in AS $A$ and $H_B$ in AS $B$ is derived as

$$K^{p}_{A:H_A\to B:H_B} = \mathrm{PRF}_{K_{A\to B}}(\text{“}p\text{”} \parallel H_A \parallel H_B)\,, \tag{4}$$

where $H_A$ and $H_B$ represent IP addresses of end hosts. For other second-level keys (e.g., between an AS infrastructure node and an end host), the derivation process is slightly adapted, see Table 2. If a protocol-specific first-level key exists, second-level keys can be derived as

$$\tilde{K}^{p}_{A:H_A\to B:H_B} = \mathrm{PRF}_{\tilde{K}^{p}_{A\to B}}(H_A \parallel H_B)\,. \tag{5}$$

## 4.3 Key Establishment

*4.3.1 First-Level Key Establishment.* Key exchange is offloaded to the key servers deployed in each AS. The key servers are not only responsible for first-level key establishment, they also derive second-level keys and provide them to hosts within the same AS.

To exchange a first-level key, the key servers of corresponding ASes perform the key exchange protocol. The key exchange is initialized by key server $KS_B$ by sending the request

$$req = A \parallel B \parallel \textit{val\_time} \parallel \tau \parallel [p], \tag{6a}$$

$$KS_B \to KS_A\colon \quad req \parallel \{req\}_{PK^{-}_{B}}, \tag{6b}$$

where $\tau$ denotes a timestamp of the current time and $\textit{val\_time}$ specifies a point in time at which the requested key is valid. If an optional protocol $p$ is supplied, the protocol-specific first-level key $\tilde{K}^{p}_{A\to B}$ is requested, otherwise the general key $K_{A\to B}$. The requested key may not be valid at the time of request, either because it already expired or because it will become valid in the future. For example, prefetching future keys allows for seamless transition to the new key. The request token is signed with $B$'s private key to prove authenticity of the request.[4]

Upon receiving the initial request, $KS_A$ checks the signature for authenticity and the timestamp for expiration. If the request has not yet expired, the key server $KS_A$ will reply with an encrypted and signed first-level key derived from the local secret value $SV_A$ or, if an optional protocol $p$ was supplied in the request, $SV^{p}_{A}$:

$$key = \begin{cases} \mathrm{PRF}_{SV^{p}_{A}}(B)\,, & \text{if request included } p \\ \mathrm{PRF}_{SV_A}(B)\,, & \text{otherwise} \end{cases} \tag{7a}$$

$$repl = \{A \parallel key\}_{PK_B} \parallel \textit{exp\_time} \parallel \tau \tag{7b}$$

$$KS_A \to KS_B\colon \quad repl \parallel \{repl\}_{PK^{-}_{A}} \tag{7c}$$

Once the requesting key server $KS_B$ has received the key, it shares it among other local key servers to ensure a consistent view. Each key server can now respond to queries by entities within the same AS requesting second-level keys. Alternatively, the proposed first-level key exchange protocol could also make use of TLS 1.3 with client certificates to secure the key exchange [63].

All first-level keys for other ASes are prefetched such that second-level keys can be derived without delay. However, on-demand key exchange between ASes is also possible. For example, in case a key server is missing a first-level key that is required for the derivation of a second-level key, the key server initiates a key exchange. ASes that contain a large number of end hosts benefit from prefetching most first-level keys, as they are likely to communicate with a large set of destination ASes. In today's Internet there exist around 68 000 active ASes [6]. Thus, setting up symmetric keys between all entities requires minor effort and state, see §7.2 for further details. To avoid explicit revocation, the shared keys are *short-lived* and new keys are established frequently (e.g., daily). Subsequent key exchanges to establish a new first-level key can use the current key as a first line of defense to avoid signature-flooding attacks.

*4.3.2 Second-Level Key Establishment.* End hosts request a second-level key from their local key server with the following request format (Table 2 provides specific examples):

$$\{\textit{type}, \textit{requestID}, \textit{protocol}, \textit{source}, \textit{destination}\}. \tag{8}$$

An end host $H_A$ in AS $A$ uses this format for issuing the following request to its local key server $KS_A$:

$$H_A \to KS_A\colon \quad \textit{format} \parallel \textit{val\_time} \parallel \tau. \tag{9}$$

We assume that this request and the reply are sent over a secure channel. We do not specify the details here as they may differ for different ASes, see §7.3.5. Similar to the first-level key exchange, $\textit{val\_time}$ specifies a point in time at which the requested key is valid. The key server only replies with a key to requests with a valid timestamp and only if the querying host is authorized to use the key. An authorized host must either be an end point of the communication that is authenticated using the second-level key or authorized separately by the AS.

[4] Instead of requesting first-level keys individually for all protocols as described here, it is also possible to return all available first-level keys to a single general key request.

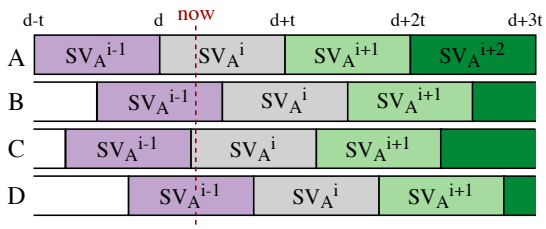


**Figure 3: Validity periods of secret values based on the destination.**

## 4.4 Key-Server Discovery

When a key server wants to contact another key server in a remote AS, it needs to know the IP address of the remote server.

In the SCION architecture, the conecept of service addresses can be used to anycast to a key server in a specific AS.

For the regular Internet, we make use of RPKI [11], which connects Internet resource information to a trust anchor. It is typically used as a trusted mapping from allocated IP prefixes to ASes authorized to originate them in BGP [47]. As the deployment numbers of RPKI are growing [55], we envision to extend the RPKI certificate with the IP address of the key server (e.g., by encoding it into the common name field or creating a separate extension [17]).

Using this mechanism, each AS publishes a list of IP addresses (or an IP anycast address) that is publicly accessible and shared by all key servers. The routing infrastructure will direct the packets to the topologically nearest key server. This mapping from an AS identifier to an IP address is verifiable through RPKI to prevent unauthorized announcements of key servers.

In case RPKI has not been fully deployed, key-server discovery could also work using a DNS entry that maps a domain to IP addresses of key servers.

## 4.5 Key Expiration

Shared symmetric keys are short-lived (i.e., up to one day lifetime) to avoid the additional complication of explicit key revocation. However, letting all keys expire at the same time would lead to peaks in key requests. Such peaks can be avoided by spreading out key expiration, which in turn will lead to spreading out the fetching requests. To this end, we introduce the deterministic mapping $\mathit{offset} : (\mathsf{A}, \mathsf{B}) \mapsto [0,\ t)$ that uniformly maps the AS identifiers of the source in AS $A$ and the destination in AS $B$ to a range between 0 and the maximum lifetime $t$ of $SV_A$. This mapping is computed using a (non-cryptographic) hash function:

$$\mathit{offset}(A, B) = \mathsf{H}(\mathsf{A} \parallel \mathsf{B}) \bmod t. \tag{10}$$

The offset is then used to determine the validity period of a key by determining the secret value $SV_A^j$ that is used to derive $K_{A\to B}$ at the current sequence $j$ as follows:

$$\left[\mathit{start}(SV_A^j) + \mathit{offset}(A, B),\ \ \mathit{start}(SV_A^{j+1}) + \mathit{offset}(A, B)\right). \tag{11}$$

Figure 3 illustrates the process of selecting the correct secret value of $A$. In the first row the sequence of secret values is depicted without any offset. The following three rows depict the intervals based on different destination ASes ($B$, $C$, and $D$). At the current time (indicated by the vertical red line), $A$ uses $SV_A^i$ to derive $K_{A\to C}$. On the other hand, $K_{A\to B}$ and $K_{A\to D}$ are derived using $SV_A^{i-1}$.

# 5 PISKES PACKET AUTHENTICATION

Using the symmetric keys described in the previous section, PISKES enables source and packet authentication for a wide variety of applications. In this section, we first describe the general authentication mechanism and then apply it to two different use cases: DNS requests and general host-to-host communication.

## 5.1 Authentication Mechanism

To send packets with PISKES to $S_B$ in AS $B$, the source $H_A$ in AS $A$ first requests the second-level key $K^p_{B:S_B\to A:H_A}$ from its local key server. For a packet $pkt$, the source then calculates an authentication tag

$$tag = \mathsf{MAC}_{K^p_{B:S_B\to A:H_A}}\left(\overline{pkt}\right), \tag{12}$$

where $\overline{pkt}$ is an immutable part of the packet, which can include parts of the layer-3 and layer-4 headers and optionally the layer-4 payload. It is important to only include immutable fields as the verification would otherwise fail at the destination, see also the following discussion in §5.1.1. The destination uses $A$'s AS number and $H_A$'s IP address to derive or request the shared key and recalculate the authentication tag.

To authenticate packets, the destination must be able to determine the key in use. In SCION, the AS number is part of the normal packet header and can thus be used directly to derive the first-level key. The additional (AS-local) host address then identifies the required second-level key. In a SCION deployment, PISKES thus simply adds the authentication tag to the original packet:

$$pkt^{\text{SCION}}_{\text{PISKES}} = pkt \parallel tag. \tag{13}$$

In the current Internet, only the IP address of the source is part of the IP header, but not the AS number. In addition to an authentication tag (16 B), PISKES therefore also adds the 4 B AS number [70] to the packet if used in today's Internet:

$$pkt^{\text{IP}}_{\text{PISKES}} = pkt \parallel A \parallel tag. \tag{14}$$

*5.1.1 Network Address Translation.* By breaking end-to-end connectivity at the network layer, network address translation (NAT) poses an obstacle to PISKES: if NAT is performed between source and destination, the destination may see a different source address than the source host. To enable successful source-address verification in this case, the local key-server always provides the second-level key based on the host's *public* IP address to the host (also in the case it employs carrier-grade NAT). In addition, it is important that the input to the MAC calculation does *not* include the source of the packet as this might be changed along the way. With this mechanism, the destination host uses the translated source IP address to derive the symmetric key, which coincides with the key used by the source to create the authentication tag.

*5.1.2 Misbehaving ASes.* By default, PISKES does not check if the source's IP address is really owned by the corresponding AS. The remote AS can therefore in principle derive and distribute host-level keys for IP addresses it does not own. To address this for ASes that are not fully trusted, the destination can sample authenticated packets and verify the IP-address ownership with RPKI. Misbehaving ASes are thus detectable and can be punished, e.g., by blacklisting. Note that ASes can only hurt their own customers by misbehaving,

not other ASes. In a SCION deployment, this issue does not occur as host addresses are only valid within an AS and can be assigned independently by each AS without requiring global coordination.

*5.1.3 Replay Attacks.* In order to rule out replay attacks, the destination can deploy a duplicate-detection system (e.g., based on Bloom filters) in addition to authenticating the packets. If the authenticated part of the packets, $\overline{pkt}$, does not provide sufficient freshness, PISKES can additionally add a timestamp or sequence number ($ts$) to the packet and the MAC calculation:

$$tag' = \mathsf{MAC}_{K^p_{B:S_B \to A:H_A}}\left(\overline{pkt} \parallel ts\right). \tag{15}$$

## 5.2 High-Speed DNS Authentication

DNS is among the most exploited protocols for traffic amplification in DoS attacks [53], as it is widely used and offers amplification factors up to 58 [67]. This section shows how PISKES can be used to provide source authentication in the context of DNS as a service that requires high-speed verification without storing per-client state. Existing solutions (e.g., DNS Cookies or DNS-over-TLS) do not provide the same properties (for details see §8).

**On-the-fly Key Derivation.** DNS servers typically require high throughput and have a large number of clients. Thus, high-speed verification of source authenticity with little required state is essential. To enable on-the-fly key derivation for a DNS server inside AS $B$, we make use of a DNS-specific secret value $SV_B^{\text{DNS}}$ and a first-level key $\tilde{K}^{\text{DNS}}_{B \to A}$ derived from it:

$$\tilde{K}^{\text{DNS}}_{B \to A} = \mathsf{PRF}_{SV_B^{\text{DNS}}}(A). \tag{16}$$

This first-level key is then exchanged with other ASes along with other first-level keys as discussed in §4.3.1. By sharing the protocol-specific secret value $SV_B^{\text{DNS}}$ with a DNS server $H_{\text{DNS}}$ located in AS $B$, the key server $KS_B$ delegates $H_{\text{DNS}}$ to locally derive second-level keys without contacting the key server and without storing any per-AS keys. For a DNS query received from some end host $H_A$, $H_{\text{DNS}}$ performs the first-level derivation as specified above and uses the resulting key $\tilde{K}^{\text{DNS}}_{B \to A}$ to derive the second-level key,

$$\tilde{K}^{\text{DNS}}_{B:H_{\text{DNS}} \to A:H_A} = \mathsf{PRF}_{\tilde{K}^{\text{DNS}}_{B \to A}}(H_{\text{DNS}} \parallel H_A). \tag{17}$$

The derivation operation from an end host perspective is the same as for any destination, but the end host's key server will use the stored protocol-specific first-level key to derive the final key. In addition to authenticating the source of the request, this key can be used to authenticate the response as well, thus significantly increasing the security of the DNS lookup.

**EDNS(0) Authentication Option.** Since modifying the DNS request and response format possibly breaks existing DNS resolvers and authoritative servers, EDNS(0) [18] provides a backwards-compatible extension mechanism for DNS. EDNS' deployment is already essential for some applications of DNS, e.g., for enabling UDP transport for DNS messages larger than 512 B, or for DNSSEC to signal the request to include DNSSEC data in the response.

We use a custom extension for PISKES authentication that includes the total query/response length, the source AS number (which is necessary in today's Internet), a timestamp, and the per-packet MAC. The per-packet MAC for DNS queries and responses is computed over the DNS header and all fields contained in the extension. Thus, the input size for the MAC computation is fixed. Using the PISKES EDNS(0) option, packet authentication for every DNS packet introduces 28 B of header overhead.

## 5.3 End-Entity Authentication

As a second application of PISKES, we show how the origin of packets between end hosts in the Internet can be authenticated. Authenticating data-plane packets requires highly efficient mechanisms to avoid opportunities for DoS attacks. We again make use of second-level keys established using PISKES to calculate per-packet MACs.

**Key Exchange for End-Entity Authentication.** For end-entity (EE) authentication, protocol-specific secret values provide little benefit, so we use the standard key derivation: The corresponding second-level key for communication between end host $H_A$ in AS $A$ and end host $H_B$ in AS $B$, is derived by the key server as

$$K^{\text{EE}}_{A:H_A \to B:H_B} = \mathsf{PRF}_{K_{A \to B}}(\text{“EE”} \parallel H_A \parallel H_B), \tag{18}$$

where $H_A$ and $H_B$ represent the IP addresses of the end hosts.

In this case, both end hosts need to contact their local key server to fetch the second-level key. Since the first-level key is derived from the zeroth-level key of AS $A$, the key server $KS_A$ can directly derive the final second-level key. Provided that the key server $KS_B$ in AS $B$ has set up the first-level key with AS $A$, it can directly derive the final key based on the stored $K_{A \to B}$ as well.

**Packet Authentication & Verification.** The symmetric key can be used to compute a MAC over the packet header or the entire packet (including payload), as described in §5.1. In order to add the MAC field to the packet, there exist multiple approaches, such as a custom header, a custom trailer, an IPv6 extension [13], or a TCP option [69]. The MAC value is added at the end host, but our system could also be implemented as a middlebox service (see §7.3.1).

Upon reaching its destination, the packet's MAC value is verified by the receiving end host using the second-level key. If the MAC is correct, the receiver is assured that the packet originated from the source AS. If the destination host trusts the source AS and its own AS, the destination host can also trust the source addresses.

# 6 IMPLEMENTATION AND EVALUATION

**Methodology.** To demonstrate the scalability of our approach, we implement a prototype of PISKES on a commodity server platform and evaluate its performance through a series of experiments. To analyze the performance and scalability of our system, we measure the throughput and latency introduced by each operation, and use micro-benchmarks and profiling for bottleneck detection. All reported results are obtained through averaging $10^5$ measurements. The highlights of the evaluation are the following:

- A single processor core, dedicated to perform key derivation, is able to saturate line-rate packet forwarding of a 10 Gbps link. The implementation is able to process around 14.7 million second-level key requests per second. Locally deriving a second-level key (2 AES operations) takes less than 18 ns.
- Packet authentication with second-level key derivation on-the-fly (e.g., on a DNS server) can be performed within 85 ns.

The commodity server was able to process 12.7 million authenticated packets per second using a single core.
- For a complete DNS resolution, PISKES only introduces an overhead of 3–5 %—significantly less than DNS-over-TLS or DNS-over-HTTPS.

## 6.1 Implementation

We perform our experiments on commodity servers equipped with an Intel Xeon E5-2680 CPU (20 MB L3 cache), 32 GB DDR3 RAM, and a 10 GbE network interface card (NIC). To evaluate PISKES in a minimal setting, we dedicate only two cores of the CPU to perform all required processing: one core processes incoming and outgoing packets, and the other core performs the key-derivation operations. To generate traffic, we utilize a dedicated traffic generator that is connected back-to-back with the server. The server receives the traffic, processes it, and sends it back to the traffic generator for measurement. All requests and responses are transported using IPv4 on the network layer and UDP on the transport layer.

The key-server prototype leverages Intel DPDK [60] for packet processing. As a PRF for key derivation as well as for MACs, PISKES utilizes AES-CMAC that we implemented in assembly using Intel's AES-NI instructions [28]. For asymmetric cryptography, we use cryptographic primitives based on Curve25519 [7], which offer high performance, small public-keys (32 B) and small signatures (64 B). The signatures are generated using the Ed25519 SUPERCOP REF10 implementation [59]. This implementation has been benchmarked on many available architectures and allows easily reproducible and comparable results. To measure the microbenchmarks, we utilize DPDK's time reference functions that internally fetches the RDTSC counter. The MAC comparison during packet authentication is implemented using SSE2 instructions.

For the evaluation of PISKES with DNS, we implemented a DNS stub resolver and server in Golang using a common DNS library [25]. Each component is run on a separate commodity server and we evaluate latency using microbenchmarks.

## 6.2 Performance Evaluation

*6.2.1 AES-CMAC Computation.* For the AES-CMAC computation, we compare the performance (in CPU cycles) of our implementation in assembly to the implementation in libcrypto of OpenSSL [61]. As the input gets padded to a multiple of the block size, we only evaluate full block sizes. Figure 4 shows that our implementation outperforms libcrypto by a constant offset (although libcrypto also uses AES-NI). For small input sizes, our custom assembly implementation roughly doubles the speed.

*6.2.2 AS-Level Key Exchanges.* For the AS-level key exchange, a key server that receives a key-exchange request checks the exchange's validity, then derives a first-level key, and finally sends the encrypted and signed key back to the sender. Thus, we evaluate how many first-level key exchanges the key server prototype can handle in the default experiment setup.

As a responder to key requests, the prototype implementation is able to process 7900 packets per second using a single processing core (see Table 3); verifying a single first-level request takes 125 µs. As an initiator of key exchanges, our prototype is able to process 21 600 packets per second; issuing a key exchange request takes 46.3 µs. As AS-level key exchanges can also be processed in parallel and only a single exchange needs to occur between each pair of ASes per day, even a key server that is implemented on commodity hardware can thus easily scale to Internet-wide deployment.

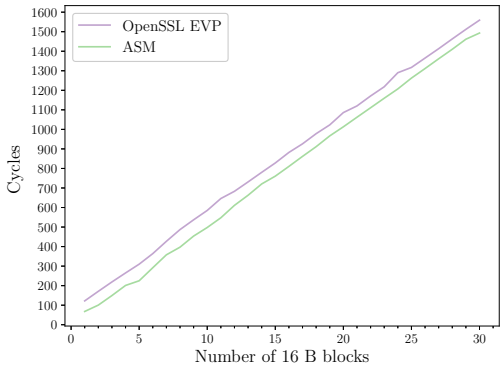


**Figure 4: AES-CMAC computation performance for different input sizes.**

**Table 3: AS-level key-exchange performance of a PISKES key server using multiple processing cores.**

| # cores | 1 | 2 | 4 | 6 | 8 |
|---|---|---|---|---|---|
| Packets per second | 7900 | 15 000 | 28 000 | 39 000 | 50 600 |

*6.2.3 Performance of Local Key Derivation.* A key server not only needs to process first-level key exchanges, but also answer second-level requests to clients located within the same AS. Second-level key derivation consists of a single dynamic key derivation. We evaluate the performance of local key derivation using our default experiment setup. As a baseline for the experiments, we use packet forwarding performance without any packet authentication.

Depending on the address type and second-level key type, the size of a request issued to the local key server varies. We assume that an IPv4 header measures 20 B, an IPv6 header 40 B, and a UDP header 8 B. Additionally, we use 1 B type input to the PRF to determine the key type. Thus, the smallest possible request size is 58 B (containing 30 B of payload), which fits into the smallest possible Ethernet frame size (64 B). Given a fixed-bandwidth link, the smallest request size results in the highest packet rate, which produces the maximum load on the key server. To estimate the impact of packet processing, we also evaluate the upper bound for second-level request size, resulting in a frame size of 117 B (48 B header + 69 B payload). In this case all involved hosts are addressed using IPv6 addresses. As we are using a block cipher (AES) as a PRF, the input size to the function must be a multiple of the cipher's block size. If the input size exceeds the block size, multiple block cipher operations are necessary to derive a key. Because we prepend the protocol identifier for second-level key derivation, the input size is further increased. For example, if communicating end hosts are both addressed using IPv6 addresses, a key derivation requires 3 block-cipher operations. The measurements account for this factor.

As Figure 5 shows, the key-derivation operation adds a negligible factor compared to the baseline for both minimal and maximal frame sizes. The prototype is able to process 14.8 million 64 B and

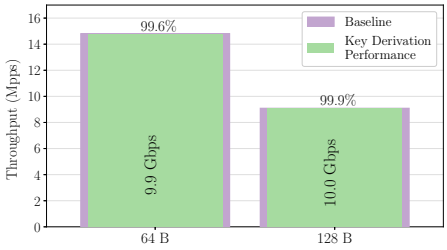


**Figure 5: Key derivation performance of our key server prototype connected using a 10GbE link.**

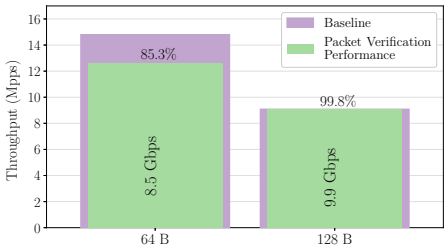


**Figure 6: Packet verification performance for an end host connected using a 10GbE link.**

9.08 million 117 B packets. Micro-benchmarking the key-derivation operation showed that each key derivation takes around 8.40 ns (64 B), or 17.5 ns (117 B).

*6.2.4 End-to-End Experiments.* As end-to-end experiments we evaluate the packet-verification performance at the receiving end host. The receiver first derives from a corresponding (cached) first-level key and then recalculates and compares the MAC; we assume that only the header is authenticated. In Figure 6, the performance of PISKES is compared to the base case without key derivation and authentication. While our implementation introduces some overhead for small packets (64 B), it is able to saturate line rate for packets of frame size of 128 B and larger. Processing a single packet takes 78.0 ns for a 64 B packet and 83.6 ns for a 128 B packet.

*6.2.5 PISKES with DNS.* To evaluate PISKES as a mechanism to provide source authentication to DNS, we compare the latency overhead introduced by PISKES to legacy DNS for three different transport modes: DNS over UDP [49], DNS over TCP [19], and DNS over TLS (DoT) [32]. For each transport mode, we evaluate the overhead of adding the PISKES EDNS(0) extension (including MAC computation), and then measure the impact of key derivation. We distinguish between only MAC verification without key derivation (i.e., the server has the second-level key cached), additional second-level key derivation (i.e., the server has the corresponding first-level key), and both first- and second-level key derivation (i.e., the server derives the key from the protocol-specific secret value $SV^{\text{DNS}}$). In our experiments, we measure the processing time (using microbenchmarks) on the DNS server and exclude any network time. Additionally, the measurements for TCP and TLS do not include the handshake in the beginning of the protocol.

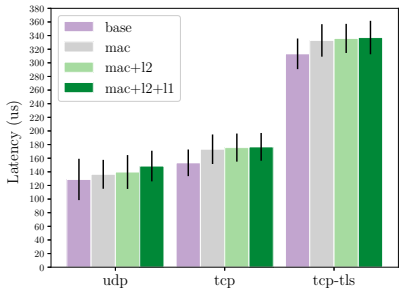


**Figure 7: Latency of DNS requests between client and server for DNS over UDP, TCP, and TLS. "base" refers to DNS without PISKES, "mac" denotes the overhead of adding the EDNS extension and MAC computation, "mac+l2" and "mac+l2+l1" additionally performs key derivation of the second-level respectively first-level key.**

Figure 7 depicts the results of our experiments. PISKES introduces only marginal overhead (3–5 %) compared to a baseline of legacy DNS, which is significantly lower than the overhead due to TLS. The significantly higher baseline for all evaluated transports compared to the previous experiments (on the order of hundreds of microseconds) can be attributed to the overhead of the implementation in Golang, which includes the Linux network stack, and is in line with other evaluations of DNS [10].

A recent evaluation of the DNS over HTTPS (DoH) ecosystem by Bottger et al. [10] shows that DoH increases the DNS resolution latency even more compared to DNS via UDP or TLS. Additionally, the authors show that delayed queries have a significant impact on TLS and DoH with HTTP/1.1 due to the serialization of the connection (head-of-line-blocking), whereas DNS via UDP remains unaffected. This shows that a system based on UDP as a transport layer is better suited for DNS.

# 7 ANALYSIS AND DISCUSSION

## 7.1 Formal Security Analysis

To show how PISKES prevents attacks that undermine source authenticity, we have formally modeled the key-exchange system and verified its security properties using the *Tamarin prover* [48]. Tamarin is a state-of-the-art security protocol-verification tool that supports both falsification and unbounded verification of symbolic models. Using the Tamarin prover tool, we show (i) the secrecy of key-exchange messages assuming that an agents' private key is not compromised; (ii) entity authentication as defined in Lowe's hierarchy of authentication specifications [45]; (iii) key uniqueness; and (iv) key agreement between hosts. For entity authentication, we show non-injective agreement using timestamps as time indicators and injective agreement using nonces. All models and generated proofs can be found at https://github.com/benrothen/piskes-verification.

We have shown that the adversary has no way of obtaining a first-level key or second-level key, except by compromising an AS or host involved in the key exchange. Given that second-level keys are host-specific, the adversary is effectively prevented from issuing a packet that evades our source-authentication system. Assuming

the adversary has compromised one or multiple hosts at arbitrary locations in the network, he is still only able to issue valid packets from these compromised locations. Even if the adversary controls a large number of end hosts, an authentic source IP address greatly assists in identifying misbehaving hosts. In the worst case where an attacker compromises a key server, he is able to impersonate all hosts located within the AS. However, end hosts located outside the corresponding AS are not affected. This engages an AS operator's interests to eliminate network-node attackers.

## 7.2 Overhead

*7.2.1 Memory Overhead.* PISKES introduces storage overhead by requiring key servers and end hosts to locally store keys. We analyze the two cases separately.

**Key Server.** PISKES requires each key server to maintain per-AS key information, which is renewed every 24 hours. The per-AS key information includes multiple first-level keys, as PISKES allows to request keys that were valid at a past point in time or will be valid in the future. Considering that the current number of ASes in today's Internet is around 68 000 [6], the storage overhead for key servers is on the order of a few megabytes, which would fit into the L3 cache of a commodity system. For example, if an AS stores first-level keys for three consecutive days, the memory overhead would result in 3 MB. This overhead has to be multiplied by the number of protocol-specific first-level keys; however, as only few protocols are expected to rely on these protocol-specific keys, the overhead will not increase beyond several tens of megabytes. Assuming five different first-level keys, we obtain a total storage overhead of 15 MB.

**End Entities.** End entities should cache second-level keys that have been obtained through requests to the local key server. Assuming a million different destinations, it would need 16 MB of memory to cache second-level keys. If memory is a limiting factor, fewer second-level keys could be cached, but fetched from the key server on demand.

*7.2.2 Communication Overhead.* PISKES introduces communication overhead at the AS level—with (i) the daily exchange of first-level keys and (ii) the second-level key exchanges between the end-entities and the key server—and at the end-entity level—by (iii) adding additional header fields, including an authentication tag, to the packet.

**AS Level.** The first-level key exchange between key servers in each AS is performed daily with a payload of less than 115 B. Thus, this introduces negligible communication overhead, even if multiple protocol-specific first-level keys are exchanged.

To estimate the number of second-level key exchanges, we analyze the number of distinct flows observed on a tier-1 backbone link [12], as for each distinct pair of end-points at least one second-level key exchange is required. Even though traffic observations on a backbone link might not lead to an exact result, it provides an estimate upper bound (worst-case scenario). During the one hour trace, we observed 23 300 000 distinct flows. Each second-level key exchange creates 201 B overhead, which results in a total overhead of 4.46 GB. Compared to the total size of the trace (185 GB), PISKES introduces less than 2.50 % communication overhead. Given that the second-level key establishment is performed within an AS, the resulting communication overhead is purely local.

**End-Entity Level.** End-entity authentication using PISKES requires adding an authentication tag and AS number (20 B) to the packet header. Findings of prior research suggest that the average packet size is around 736 B [35, 51]. On average, PISKES thus introduces a communication overhead of only 2.6 %. In case of applying PISKES to DNS with generally smaller request packets (< 100 B) [1], the additional 28 B correspond to an overhead of ∼30 %, which is still significantly less than the overhead of (D)TLS or HTTPS [10].

*7.2.3 Scalability.* The total number of pairwise shared keys scales quadratically with the number of ASes, resulting in a potentially high overhead given a large number of ASes. In practice, however, an AS only needs to proactively exchange keys with other ASes that it will likely communicate with (which can be determined based on past key requests). If a shared key is needed for an unexpected AS, the key exchange can occur on demand which will result in a higher latency for that request. We anticipate that such on-demand key exchanges will be rare, as the majority of interactions will likely be predictable.

However, even if the total number of ASes increased to one million (15 times as many as today), the storage overhead on key servers would only increase to ∼200 MB (assuming five different first-level keys per AS). The communication overhead of exchanging these five different first-level keys for all $10^{12}$ AS combinations, which requires two ∼100 B packets each,[5] amounts to a total daily communication overhead of $\sim 10^{12} \cdot 10 \cdot 100\,\text{B} = 10^{15}\,\text{B} = 1\,\text{PB}$. This is only 0.025 % of the total daily Internet traffic of 4 EB in 2017 [15].

## 7.3 Deployment

*7.3.1 Incremental Deployment.* PISKES allows incremental deployment as key servers could be gradually deployed in each AS. Already in the incremental deployment phase, PISKES prevents the addresses of upgraded ASes from being spoofed at other upgraded destination ASes. Thus, early adopters can immediately profit from PISKES's security properties.

Authenticating a packet requires packet modification either at the end host, or at a network appliance such as a middlebox or border router. Adding the MAC at the end host does not increase the request size en-route. However, updating end hosts might not be possible in the short-term. In this case, PISKES can be implemented on a middlebox that computes a per-packet MAC and modifies applicable bypassing packets. Also packet verification at the destination AS can be performed by a middlebox.

If a destination does not understand PISKES traffic, it would likely fail to process this traffic. In this case, the sending host contacts its local key server and asks if the destination AS supports PISKES. The key server might have previously derived second-level keys for an end host in the corresponding AS or might forward the query to a key server in the destination AS. We envision that if an AS supports PISKES, then it deploys a middlebox that performs the PISKES operations in case the end host does not support it. This will prevent sending authenticated traffic to a destination host that

[5] Further optimizations like sending all protocol-specific keys in one packet and combining them with the key request of the opposite direction could reduce this overhead.

does not support PISKES. In the worst case, the end host could fall back to legacy traffic.

In case of operational failures (e.g., a single key server fails), the end entity will try to contact another key server in the same AS. In the worst case, where all key servers fail, the system could fall back to the current system with unauthenticated traffic.

*7.3.2 Deployment Incentives.* As PISKES can be deployed on commodity hardware and integrates well with the current Internet infrastructure, the deployment obstacle for PISKES is low. In contrast to filtering-based systems, PISKES traffic can be recognized outside of ASes that have deployed PISKES and can thus be prioritized by servers. Thus, relatively few companies deploying PISKES to authenticate packets at their services—e.g., popular open DNS resolvers of Google or Cloudflare—provide strong incentives for ISPs to deploy PISKES and provide its services to their customers.

*7.3.3 Key-Server Latency.* The initial connection setup depends on the latency of the connection between the client and the key server. We envision that PISKES's key servers are positioned in an AS at a similar location as local DNS resolvers. As even public resolvers have an average query latency of less than 20 ms [57] traversing the Internet, we expect the latency of a local key derivation to be in the order of a few ms. Furthermore, in most cases locally fetching a key results in a lower latency than a full round-trip between client and server that is required by other protocols. For ASes that are geographically dispersed, multiple key servers can be deployed (e.g., co-located with an access router or per point-of-presence).

*7.3.4 PISKES and Trust in ASes.* The keys provided by PISKES do not provide full end-to-end authenticity or secrecy properties: The source and destination ASes are able to derive the keys and could thus perform an active attack. However, this is limited to these two ASes—active attacks by intermediate ASes are not possible. PISKES thus always enables AS-level source authentication and host-level source authentication under the additional assumption of an honest source AS.

*7.3.5 Authentication within an AS.* To achieve secrecy as well as end-host authentication for communication between end hosts and key servers, an AS needs an intra-domain end-host and/or user-authentication system. We discuss different authentication mechanisms based on the operational environment.

**Authentication using TLS.** In order to securely exchange second-level PISKES keys between end hosts and key server, the end host can establish a secure TLS channel to the key server. The identity of the communicating parties is authenticated using public-key cryptography for both the key server and the end host. Thus, the key server can uniquely identify the end host and verify its legitimacy to obtain a second-level key.

**Deployment in ISPs.** If the corresponding AS is an ISP, we assume that they can identify their customers (e.g., terminal connection number or router that has been deployed by the ISP). In this case, only an attacker that is present at the customers local network can gain access to learn keys.

**Company / University.** For ASes that are under the control of companies or universities, login credentials or other local authentication mechanisms can be used to identify the user. This gives companies the ability to run their own web servers and have full control over their key material.

**Mobile Devices.** For mobile devices such as smart phones that are connected to the Internet through a mobile telecommunication network, clients could be authenticated by the telecom provider, for example using the International Mobile Equipment Identity (IMEI).

*7.3.6 Network Mobility.* Network mobility allows entities to move from one AS to another while maintaining communication sessions. In PISKES, key derivations are based on the current location of the entity in the Internet. Therefore, as soon as an entity moves to another AS, it needs to contact the key server of the new AS and fetch new second-level keys. Because local key derivation is fast and the latency of obtaining a key is small, the overhead is minimal.

## 7.4 Further Use Cases of PISKES

*7.4.1 Encrypted SNI using PISKES.* The TLS Server Name Indication (SNI) extension [9] allows servers to host multiple domains on the same set of IP addresses. During the initial TLS handshake, clients specify which domain they want to connect to using an SNI extension that contains the hostname. The server returns the corresponding TLS certificate and applies the connection configuration for the specified domain. However, the SNI is transmitted in cleartext and thus leaks the site visited by a client to an on-path eavesdropper—a significant privacy risk.

To thwart this risk, the IETF is currently standardizing a mechanism for encrypting the SNI extension (eSNI) [65]. For key distribution, the server publishes its public key as a DNS record. Before connecting to a server, the client fetches the server's key via DNS. The SNI is then encrypted using a symmetric key established based on a DH key exchange. This introduces significant communication and computation overhead, as the DH key exchange is performed solely for SNI encryption and the client's public DH key is transmitted in the eSNI extension, requiring 256 B.

Additionally, the requirement of using DNS for key distribution also introduces privacy risks. An on-path eavesdropper can observe DNS requests to accurately determine which domains are visited by a client. This can be prevented using DNS over TLS (DoT) [32], which adds communication and processing overhead introduced by TLS. Furthermore, the keys distributed using DNS have no authentication or provenance information. This allows an attacker to inject DNS replies or perform DNS cache poisoning to tamper with the server's public key. DNSSEC [29] forms a defense mechanism, but suffers from deployment and security drawbacks [29]

**SNI Encryption.** PISKES can be used for encryption of the first packet being sent from client $H_A$ towards a server $S_B$ using the key

$$K^{\text{esni}}_{A:H_A \to B:H_B} = \text{PRF}_{K_{A\to B}}(\text{“eSNI”} \parallel S_A \parallel H_B)\,. \tag{19}$$

The SNI extension can be encrypted and authenticated using this key (e.g., using AES-GCM), providing secrecy against on-path attackers (excluding the source and destination AS). $H_A$ replaces the SNI extension in the TLS handshake with the encrypted SNI, introducing 20 B overhead (see §5.1). Upon receiving a TLS ClientHello, the server $S_B$ authenticates and decrypts the eSNI and extracts the requested domain.

This approach entails several advantages: (i) PISKES significantly reduces the processing and communication overhead for involved

entities compared to a DH-based key establishment and (ii) avoiding DNS as a mechanism for key distribution also avoids the privacy risks of DNS.

*7.4.2 Secure Control Message Protocol (SCMP).* Security of a control protocol is essential for the security of higher-level protocols. The Internet Control Message Protocol (ICMP) does not provide any form of authentication and enables attacks on lower layer protocols using maliciously generated packets [26]. Providing security for ICMP in today's Internet is challenging, as control packets are often created and processed by routers. Previously, it has been seen that packet authentication by routers is too expensive and might provide an additional attack vector for DoS attacks [39]. For example, a naive approach of adding digital signatures to authenticate ICMP messages would create a processing bottleneck at routers. Alternatively, ICMP could be authenticated using IPsec, which has high computational overhead and requires an expensive key setup between end hosts [16]. To provide scalability to high amounts of traffic, the authentication generation process needs to be highly efficient.

We propose the Secure Control Message Protocol (SCMP), an analogous protocol to ICMP in the current Internet but *with* source authentication. SCMP packets are protected and verified by border routers using a shared symmetric key on the AS level.

**Authentication and Keys.** SCMP's authentication uses PISKES's second-level keys to compute a MAC of the entire packet (not only its payload) and appends the MAC to the data field of the SCMP packet. This approach offers scalability and efficient packet processing. SCMP messages can be created by routers, end hosts, or other AS components. This entails different types of SCMP authentication key depending on the combination of source and recipient type of the SCMP packet.

Due to the high efficiency required for SCMP on routers, every router needs to be able to locally derive the necessary keys. We therefore propose to use a protocol-specific secret value $SV_A^{\text{scmp}}$, which is shared with all routers in an AS $A$ and can be used to derive second-level keys in two steps:

$$\tilde{K}^{\text{scmp}}_{A\to B} = \mathsf{PRF}_{SV_A^{\text{scmp}}}(B)\,, \tag{20a}$$

$$\tilde{K}^{\text{scmp}}_{A\to B:H_B} = \mathsf{PRF}_{\tilde{K}^{\text{scmp}}_{A\to B}}(H_B)\,. \tag{20b}$$

To verify an SCMP message, the receiving end host $H_B$ has to request this key from the local key server. If host $H_B$ wants to reply to the received SCMP messages, the host must reuse this key to allow efficient verification on the border router (without a fetching an additional key). Alternatively, an AS could also perform verification on behalf of the end host.

In addition to routers, SCMP messages can also be exchanged between two end hosts. In this case, they and are authenticated using $\tilde{K}^{\text{scmp}}_{A:H_A\to B:H_B}$, that is defined as

$$\tilde{K}^{\text{scmp}}_{A:H_A\to B:H_B} = \mathsf{PRF}_{\tilde{K}^{\text{scmp}}_{A\to B}}(H_A \parallel H_B) \tag{21}$$

Both end hosts $H_A$ and $H_B$ need to request this key from their local key server in order to authenticate a packet.

# 8 RELATED WORK

In this section, we extend the background of §3 and discuss additional previous research on source authentication, key distribution, and DoS-defense systems with a particular focus on DNS.

**Source Authentication.** In the past, the issue of source-address spoofing has often been tackled through filtering-based techniques (see §3.1.1), which provide weak guarantees at partial deployment or in the face of malicious ASes. Source authentication at the network layer can be achieved through VPNs like IPsec [40]; however, this generally relies on an expensive IKE handshake [38], which in turn relies on pre-shared symmetric keys or asymmetric cryptography for authentication. Passport [44] uses keys exchanged via BGP announcements to authenticate packets at the AS level, see §3.2. Cryptography-based source authentication has also been studied in the context of future Internet architectures—but generally with large overhead: ICING [52], integrated in the Nebula architecture [3], provides source authentication to on-path routers. It requires each router on a path to store and look up keys shared with other routers, resulting in a large memory overhead. ICING introduces 42 B header overhead per verifying router. OPT [41], designed for the SCION architecture [58], provides similar source-authentication properties with a slightly reduced overhead and more efficient key-establishment mechanism based on DRKeys. Still, it introduces significant communication overhead of 16 B per hop. The Accountable Internet Protocol (AIP) [2] proposes a different path to authentication based on self-certifying names.

**Symmetric-Key Distribution.** Similar to PISKES, Kerberos [54] uses a dedicated server for key generation and distribution, and builds on symmetric cryptography to provide mutual authentication. However, it is designed and suited for local area networks, not for deployment across the Internet. Kerberos' key-distribution server has no mechanism to provide key generation delegation to other servers. This delegation allows servers to derive keys on-the-fly for high-speed verification. More closely related to PISKES are the global key-distribution systems Passport [44] and DRKey [41, 58] that have been described in §3.2.

**Public Key Infrastructures.** Several systems have been proposed to create or improve global control-plane PKIs, including RPKI [11], DANE [31], and IPA [42]. As we have discussed in §2.1.2, asymmetric cryptography is not efficient enough to provide source authentication for data-plane packets. However, these PKIs can serve as anchoring points for the PISKES key distribution.

**General DoS Defense.** There exists a vast literature of systems tackling various types of DoS attacks [73]. In addition to the already discussed filtering- and source-authentication-based systems, there exist several proposals based on network capabilities, including SIFF [71], TVA [72], CAT [14], and SIBRA [5]. These systems generally operate within the network, in contrast to PISKES, which only relies on the end hosts and thus provides stronger properties in a partial deployment. Furthermore, these systems are orthogonal to PISKES in that they often provide stronger properties but themselves rely on an additional source-authentication system like PISKES. None of the capability-based techniques allow the destination to authenticate the first packet.

**Defense Mechanisms for DNS.** Orthogonal to general DoS defense mechanism, systems attempting to mitigate reflection attacks

targeting DNS have been proposed. For example, DNS Cookies [22] are an EDNS option that provide a weak form of authentication of DNS requests and responses by detecting and ignoring off-path spoofed responses. DNS over TLS [32], DNS over Datagram TLS (DTLS) [62], and DNS over HTTPS [30] provide confidential communication between DNS entities and can also provide source authentication when used in combination with client certificates. However, all three systems inherit communication and processing overhead introduced by (D)TLS negotiation (e.g., certificate transfer) [10, 63, 64], and require per-host state (without deployment of [68]). Additionally, the systems cannot authenticate the first packet that is sent by a client and, if no client certificates are used, they cannot defend against on-path attackers during the handshake.

## 9 CONCLUSION

An infrastructure that enables network address authentication is hard to come by—despite its vital importance we do not have a working system today. In this paper, we have presented PISKES, an efficient system enabling source authentication based on highly efficient symmetric cryptography and dynamic key derivation. Through its efficient key hierarchy, services can already authenticate the first packet received from any other host without relying on a handshake to set up keying material. Our prototype implementations have shown that packets can indeed be authenticated in less than 100 ns, and both communication and computation overhead of PISKES are small. PISKES is an important building block for DoS defense: on the one hand it can be used to prevent reflection and amplification attacks that are currently possible by abusing UDP-based services such as NTP and DNS; on the other hand, end hosts can use source authentication to rate-limit or drop traffic from specific sources and thus filter out unwanted traffic. Since PISKES offers unique and decisive advantages that previous systems lack–e.g., also provides early adopters with immediate benefits—and it can be set up using commodity hardware, it may finally bring packet authentication to the Internet.

## 10 ACKNOWLEDGMENTS

We gratefully acknowledge support from ETH Zürich and from the Zürich Information Security and Privacy Center (ZISC). We thank our colleagues from ETH Zürich who provided insight and expertise that supported and improved our research, especially Dennis Jackson, Tae-Ho Lee, Ralf Sasse, Paweł Szałachowski, and Brian Trammell. Lastly, we thank the anonymous reviewers for their useful feedback.

## REFERENCES

[1] M. Anagnostopoulos, G. Kambourakis, P. Kopanos, G. Louloudakis, and S. Gritzalis. 2013. DNS amplification attack revisited. *Computers & Security* 39 (2013).
[2] D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker. 2008. Accountable internet protocol (AIP). In *ACM SIGCOMM Computer Communication Review*, Vol. 38. ACM.
[3] T. Anderson, K. Birman, R. Broberg, M. Caesar, D. Comer, C. Cotton, M. J. Freedman, A. Haeberlen, Z. G. Ives, A. Krishnamurthy, et al. 2013. The NEBULA future internet architecture. In *The Future Internet Assembly*. Springer.
[4] F. Baker and P. Savola. 2004. *Ingress Filtering for Multihomed Networks.* RFC 3704.
[5] C. Basescu, R. M. Reischuk, P. Szalachowski, A. Perrig, Y. Zhang, H.-C. Hsiao, A. Kubota, and J. Urakawa. 2016. SIBRA: Scalable Internet Bandwidth Reservation Architecture. In *Proceedings of the Symposium on Network and Distributed System Security (NDSS)*.
[6] T. Bates. 2020. CIDR Report. https://www.cidr-report.org/as2.0/.
[7] D. J. Bernstein. 2006. Curve25519: new Diffie-Hellman speed records. In *International Workshop on Public Key Cryptography*. Springer.
[8] A. Bittau, D. Giffin, M. Handley, D. Mazieres, Q. Slack, and E. Smith. 2019. *Cryptographic Protection of TCP Streams (tcpcrypt).* RFC 8548.
[9] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, and T. Wright. 2003. *Transport Layer Security (TLS) Extensions.* RFC 3546.
[10] T. Böttger, F. Cuadrado, G. Antichi, E. L. a. Fernandes, G. Tyson, I. Castro, and S. Uhlig. 2019. An Empirical Study of the Cost of DNS-over-HTTPS. In *Proceedings of the Internet Measurement Conference*. ACM.
[11] R. Bush. 2014. *Origin Validation Operation Based on the Resource Public Key Infrastructure (RPKI).* RFC 7115.
[12] CAIDA. 2016. CAIDA Passive Monitor: equinix-chicago. https://www.caida.org/data/monitors/passive-equinix-chicago.xml.
[13] B. Carpenter and S. Jiang. 2013. *Transmission and Processing of IPv6 Extension Headers.* RFC 7045.
[14] M. Casado, A. Akella, P. Cao, N. Provos, and S. Shenker. 2006. Cookies Along Trust-Boundaries (CAT): Accurate and Deployable Flood Protection. In *SRUTI*.
[15] Cisco. 2018. Cisco Predicts More IP Traffic in the Next Five Years Than in the History of the Internet. https://newsroom.cisco.com/press-release-content?type=webcontent&articleId=1955935
[16] A. Conta, S. Deering, and M. Gupta. 2006. *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification.* RFC 4443.
[17] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. 2008. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.* RFC 5280.
[18] J. Damas, M. Graff, and P. Vixie. 2013. *Extension Mechanisms for DNS (EDNS(0)).* RFC 6891.
[19] J. Dickinson, S. Dickinson, R. Bellis, A. Mankin, and D. Wessels. 2016. *DNS Transport over TCP - Implementation Requirements.* RFC 7766.
[20] D. Dolev and A. Yao. 1983. On the security of public key protocols. *IEEE Transactions on information theory* 29, 2 (1983).
[21] Z. Duan, X. Yuan, and J. Chandrashekar. 2006. Constructing Inter-Domain Packet Filters to Control IP Spoofing Based on BGP Updates. In *INFOCOM*.
[22] D. Eastlake 3rd and M. Andrews. 2016. *Domain Name System (DNS) Cookies.* RFC 7873.
[23] ECRYPT. 2019. eBATS: ECRYPT Benchmarking of Asymmetric Systems. https://bench.cr.yp.to/results-dh.html.
[24] P. Ferguson and D. Senie. 2000. *Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing.* RFC 2827.
[25] M. Gieben. 2019. DNS library in Go. https://github.com/miekg/dns.
[26] F. Gont. 2010. *ICMP Attacks against TCP.* RFC 5927.
[27] Google. 2016. Roughtime. https://roughtime.googlesource.com/.
[28] S. Gueron. 2010. Intel AES New Instructions Set. *Intel Corporation* (2010).
[29] A. Herzberg and H. Shulman. 2013. DNSSEC: Security and availability challenges. In *IEEE Conference on Communications and Network Security (CNS)*.
[30] P. Hoffman and P. McManus. 2018. *DNS Queries over HTTPS (DoH).* RFC 8484.
[31] P. Hoffman and J. Schlyter. 2012. *The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA.* RFC 6698.
[32] Z. Hu, L. Zhu, J. Heidemann, A. Mankin, D. Wessels, and P. Hoffman. 2016. *Specification for DNS over Transport Layer Security (TLS).* RFC 7858.
[33] D. Jackson, C. Cremers, K. Cohn-Gordon, and R. Sasse. 2019. Seems Legit: Automated Analysis of Subtle Attacks on Protocols That Use Signatures. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. ACM.
[34] C. Jin, H. Wang, and K. G. Shin. 2003. Hop-count filtering: an effective defense against spoofed DDoS traffic. In *Proceedings of the ACM conference on computer and communications security*. ACM.
[35] W. John and S. Tafvelin. 2007. Analysis of Internet Backbone Traffic and Header Anomalies Observed. In *Proceedings of the ACM SIGCOMM Conference on Internet Measurement*. ACM.
[36] M. Jonker, A. King, J. Krupp, C. Rossow, A. Sperotto, and A. Dainotti. 2017. Millions of Targets Under Attack: A Macroscopic Characterization of the DoS Ecosystem. In *Proceedings of the Internet Measurement Conference*. ACM.
[37] J. Katz, A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone. 1996. *Handbook of applied cryptography.* CRC press.
[38] C. Kaufman, P. Hoffman, Y. Nir, P. Eronen, and T. Kivinen. 2014. *Internet Key Exchange Protocol Version 2 (IKEv2).* RFC 7296.
[39] S. Kent and R. Atkinson. 1998. *Security Architecture for the Internet Protocol.* RFC 2401.
[40] S. Kent and K. Seo. 2005. *Security Architecture for the IP.* RFC 4301.
[41] T. H.-J. Kim, C. Basescu, L. Jia, S. B. Lee, Y.-C. Hu, and A. Perrig. 2014. Lightweight source authentication and path validation. In *ACM SIGCOMM Computer Communication Review*, Vol. 44. ACM.
[42] A. Li, X. Liu, and X. Yang. 2011. Bootstrapping accountability in the Internet we have. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[43] J. Li, J. Mirkovic, M. Wang, P. Reiher, and L. Zhang. 2002. SAVE: Source address validity enforcement protocol. In *Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies*, Vol. 3. IEEE.

[44] X. Liu, A. Li, and X. Yang. 2008. Passport: Secure and Adoptable Source Authentication. *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[45] G. Lowe. 1997. A hierarchy of authentication specifications. In *Computer security foundations workshop, 1997. Proceedings., 10th*. IEEE.

[46] M. Luckie, R. Beverly, R. Koga, K. Keys, J. A. Kroll, and k. claffy. 2019. Network Hygiene, Incentives, and Regulation: Deployment of Source Address Validation in the Internet. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. ACM.

[47] C. Lynn, S. Kent, and K. Seo. 2004. *X.509 Extensions for IP Addresses and AS Identifiers*. RFC 3779.

[48] S. Meier, B. Schmidt, C. Cremers, and D. Basin. 2013. The TAMARIN prover for the symbolic analysis of security protocols. In *International Conference on Computer Aided Verification*. Springer.

[49] P. Mockapetris. 1987. *Domain names - concepts and facilities*. STD 13.

[50] J. Mukundan, H. Hunter, K.-h. Kim, J. Stuecheli, and J. F. Martínez. 2013. Understanding and mitigating refresh overheads in high-density DDR4 DRAM systems. *ACM SIGARCH Computer Architecture News* 41, 3 (2013).

[51] D. Murray and T. Koziniec. 2012. The state of enterprise network traffic in 2012. In *Communications (APCC), 2012 18th Asia-Pacific Conference on*. IEEE.

[52] J. Naous, M. Walfish, A. Nicolosi, D. Mazières, M. Miller, and A. Seehra. 2011. Verifying and enforcing network paths with ICING. In *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies*. ACM.

[53] NETSCOUT. 2019. 14th Annual Worldwide Infrastructure Security Report. https://www.netscout.com/sites/default/files/2019-03/SECR_005_EN-1901-WISR.pdf.

[54] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. 2005. *The Kerberos Network Authentication Service (V5)*. RFC 4120.

[55] NIST. 2019. RPKI Monitor. https://rpki-monitor.antd.nist.gov.

[56] K. Park and H. Lee. 2001. On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law internets. In *ACM SIGCOMM computer communication review*, Vol. 31. ACM.

[57] PerfOps. 2018. DNS Performance Analytics and Comparison. https://www.dnsperf.com/.

[58] A. Perrig, P. Szalachowski, R. M. Reischuk, and L. Chuat. 2017. *SCION: A Secure Internet Architecture*. Springer Verlag.

[59] O. Peters. 2017. Ed25519 Supercop ref10 implementation. https://github.com/orlp/ed25519.

[60] D. Project. 2017. Data Plane Development Kit. https://dpdk.org.

[61] T. O. Project. 2018. OpenSSL libcrypto. https://github.com/openssl/openssl.

[62] T. Reddy, D. Wing, and P. Patil. 2017. *DNS over Datagram Transport Layer Security (DTLS)*. RFC 8094.

[63] E. Rescorla. 2018. *The TLS Protocol Version 1.3*. RFC 8446.

[64] E. Rescorla and N. Modadugu. 2012. *Datagram Transport Layer Security Version 1.2*. RFC 6347.

[65] E. Rescorla, K. Oku, N. Sullivan, and C. A. Wood. 2019. *Encrypted Server Name Indication for TLS 1.3*. Internet-Draft draft-ietf-tls-esni-05. https://datatracker.ietf.org/doc/html/draft-ietf-tls-esni-05

[66] D. Roos, L. Chuat, and A. Perrig. 2019. SCION Control-Plane PKI. https://github.com/scionproto/scion/blob/master/doc/ControlPlanePKI.md

[67] C. Rossow. 2014. Amplification Hell: Revisiting Network Protocols for DDoS Abuse. In *Proceedings of the Symposium on Network and Distributed System Security (NDSS)*.

[68] J. Salowey, H. Zhou, P. Eronen, and H. Tschofenig. 2008. *Transport Layer Security (TLS) Session Resumption without Server-Side State*. RFC 5077.

[69] J. Touch, A. Mankin, and R. Bonica. 2010. *The TCP Authentication Option*. RFC 5925.

[70] Q. Vohra and E. Chen. 2012. *BGP Support for Four-Octet Autonomous System (AS) Number Space*. RFC 6793.

[71] A. Yaar, A. Perrig, and D. Song. 2004. SIFF: A stateless Internet flow filter to mitigate DDoS flooding attacks. In *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on*. IEEE.

[72] X. Yang, D. Wetherall, and T. Anderson. 2008. TVA: a DoS-limiting network architecture. *IEEE/ACM Transactions on Networking (ToN)* 16, 6 (2008).

[73] S. T. Zargar, J. Joshi, and D. Tipper. 2013. A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks. *IEEE communications surveys & tutorials* 15, 4 (2013).