Lightweight Source Authentication and Path Validation

Tiffany Hyun-Jin Kim
CyLab, CMU
hyunjin@cmu.edu

Cristina Basescu
ETH Zürich
cba@inf.eth.ch

Limin Jia
CyLab, CMU
liminjia@cmu.edu

Soo Bum Lee
Qualcomm
soobuml@qti.qualcomm.com

Yih-Chun Hu
UIUC
yihchun@uiuc.edu

Adrian Perrig
ETH Zürich
perrig@inf.eth.ch

ABSTRACT

In-network source authentication and path validation are fundamental primitives to construct higher-level security mechanisms such as DDoS mitigation, path compliance, packet attribution, or protection against flow redirection. Unfortunately, currently proposed solutions either fall short of addressing important security concerns or require a substantial amount of router overhead. In this paper, we propose lightweight, scalable, and secure protocols for shared key setup, source authentication, and path validation. Our prototype implementation demonstrates the efficiency and scalability of the protocols, especially for software-based implementations.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: Security and protection; C.2.1 [**Network Architecture and Design**]: Circuit-switching networks, Packet-switching networks

Keywords

Source Authentication, Path Validation, Retroactive Key Setup

1. INTRODUCTION

Source authentication and path validation are useful primitives to help mitigate various network-based attacks, such as DDoS, address spoofing, and flow redirection attacks [7]. Path validation, in particular, provides a way to enforce path compliance according to the policies of ISPs, enterprises, and datacenters. Endhosts and ISPs desire to validate service level agreement compliance regarding data delivery in the network: Did the packet truly originate from the claimed client? Did the client select a path that complies with the service provider's policy? Did the packet indeed travel through the path selected by the client?

Unfortunately, the current Internet provides almost no means for source authentication and path validation by routers or endhosts, opening up numerous attack surfaces. For example, a malicious ISP may forward a packet on an inferior path while claiming to its client that it forwarded the packet on the premium path. Alternatively, a malicious router may inject packets with a spoofed source address to incriminate a victim source node into having sent an excessive number of packets. A malicious router may simply alter the contents of received packets as well. The inability to detect such attacks near the point of deviation wastes downstream resources. Furthermore, an adversary may exploit the routing protocol to divert traffic to traverse a point of eavesdropping it controls—a serious issue in particular for sensitive information.

End-to-end encryption and authentication mechanisms, such as TLS, do not solve any of the above issues, since they are agnostic to which path the packet takes. A stronger approach is needed, which enables routers and destinations to perform source authentication and path validation. As we discuss in the related work, existing solutions either require extensive overhead, or only partially address fundamental problems, affecting both feasibility and practicality in the existing network. For example, ICING [26] addresses both source authentication and path validation, but it requires each intermediate router on a path to store and look up keys shared with other routers; ICING requires 42 bytes per verifying router in the packet header. Furthermore, ICING requires each router to calculate a Message Authentication Code (MACs) for *all other routers on the path*. In contrast, our protocol does not require any per-client state on routers; it requires only 16 bytes per hop (which can be reduced to 2 bytes for a lower level of security), and only a single MAC and a single PRF computation per router *irrespectively of the path length*. Moreover, one of our protocol instantiations prevents against coward attacks [20], where an adversary only attacks when it knows that the attack will not be detected. Our protocol, however, offers reduced security in the case of a malicious sender colluding with a malicious router on the path, which we describe in detail in the related work section. Since in the common case, sender and receiver trust each other, the performance gain of $O(1)$ MAC operation per router instead of $O(n)$ is worth the tradeoff.

Contributions. In this paper, we present *Dynamically Recreatable Key* (DRKey) protocols that enable routers to (re-)create symmetric keys shared with the endhosts on the fly. The stateless operation of DRKey on routers prevents state exhaustion DoS attacks and simplifies router architecture. We further enrich DRKey with a new notion called *retroactive key setup* that provides the following desirable properties: (1) in contrast to previous protocols, source and destination can start the communication without needing to wait for the expensive key setup to complete, providing *efficiency*; (2) if misbehavior is suspected, endhosts set up keys retroactively to verify previous packets, *defending against coward attacks*.

Based on the dynamically (re-)creatable keys through DRKey protocols, we present *Origin and Path Trace* (OPT)—-lightweight, scalable, and secure protocols for source authentication and path validation. We introduce an extension called *Retroactive-PathTrace* that supports the destination to perform path validation with retroac-

SIGCOMM'14, August 17–22, 2014, Chicago, IL, USA.

http://dx.doi.org/10.1145/2619239.2626323.

tive key setup and to detect coward attackers with small, constant overhead in the packet header. Our OPT protocols enable implementation on SW routers with minimal performance impact.

2. PROBLEM DEFINITION

2.1 Desired Security Properties

Source authentication and data authentication. The destination and each intermediate router should be able to determine whether the packet indeed originated from the claimed source and whether the packet content has not been altered en route. In this paper, source authentication includes data authentication.

Path validation. The source, intermediate routers, and the destination should be able to validate that the packet indeed traversed the path known to (or selected by) the source. Successful path validation ensures that the packet traversed each honest router on the path in the correct order. Unfortunately, no scheme can provide any guarantees for malicious routers: if malicious router R_m publishes its secret keys, another malicious router $R_{m'}$ could perform cryptographic operations on a packet without traversing R_m.

2.2 Elided Security Properties

No packet delivery guarantee. Routers generally have the freedom to decide whether or not to forward packets. Hence, it is not the purpose of path validation to guarantee that packets will be delivered to the specified destination.

No detection of packet siphoning. Misbehaving router R_m on the source-selected path can siphon packets and send them over a separate channel to a remote entity. Since R_m still forwards the packet to R_{m+1}, this attack is not detected. We consider R_m as obeying the protocol as long as it performs protocol-compliant operations with the packet.

No locating of packet altering and dropping routers. Locating routers that alter or drop packets is the goal of fault-localization mechanisms—another challenging problem especially in inter-domain settings [39]. Since path validation is a simpler problem, the goal is to achieve a more efficient protocol than heavy-weight fault localization.

2.3 Adversary Model

We consider a computationally-bounded network attacker that deviates from the protocol and violates its security goals as we describe next.

Packet alteration. A malicious router alters any part of the packet, such as source address, header information, or payload data.

Packet injection. A malicious router fabricates a packet and sends it towards a destination of its choice. A packet replay attack is a special case of packet injection.

Path deviation. A malicious router may perform *path deviation attacks*, which cause packets to be forwarded along a path other than the path previously selected by the source. We subdivide this attack as follows:

- **Path detour:** Malicious router R_m causes a packet to deviate from the intended forwarding path, but later the packet returns to the correct downstream router R_{m+1} to resume traversal of *all* routers on the intended path.
- **Router skipping:** A malicious router redirects the packet and skips other router(s) on the path. Thus, some routers on the intended path does not forward the packet.
- **Out-of-order traversal:** An adversary causes path deviations such that routers on the intended path are not traversed in the right order.

Coward attack. An adversary launches a coward attack [20] only when the adversary believes that the attack cannot be detected. For example, an attacker diverts traffic only when the protocol is inactive (e.g., required keys for validation have not been established).

Denial-of-Service (DoS). As part of DoS attacks, we consider memory and computation exhaustion attacks on routers performing source authentication and path validation.

Collusion. Protocol participants may collude to carry out any of the attacks listed above. For example, two or more intermediate routers may collude to claim the use of an expensive path for monetary profit, or the source may collude with an intermediate router to spoof authenticators for its downstream routers if the destination prefers/trusts skipped routers. Also, both the source and the destination could collude with some intermediate routers to frame another router on the path by not forwarding packets to it.

In Section 6, we explore potential attacks against our protocols that violate the desired properties and discuss how OPT defends against these attacks.

3. OPT DESIGN OVERVIEW

We consider a setting in which source S sends a packet to destination D along a sequence of routers R_i. We refer to S, D, and R_i's as *tracing entities*. At a very high level, the main insights for achieving source authentication and path validation without requiring routers to maintain per-source or per-path-length state are as follows: (1) In the packet header, source S includes $H(P)$, which is the hash of the packet payload to help receiving entities identify the packet while avoiding expensive hash computation at each router; (2) On demand, each router R_i generates key K_i using a symmetric cryptographic operation, and requires only router's local secret SV_{R_i} and a special value called SESSIONID in the packet header as inputs. Consequently, generating deterministic keys is stateless and faster than storing or retrieving secrets. (3) Each router performs source authentication using a MAC computed over $H(P)$; (4) Each router R_i extends a special authentication field called PVF by performing a MAC operation. Hence, path validation is achieved through a chain consisting of nested MACs.

3.1 Assumptions

For the communication properties of the network, we assume that the source knows the path that the packet will traverse at the AS- or router-level granularity to enable path validation, and that the source knows which entities in that path desire to perform the validations. This information can stem from (1) the BGP protocol where the source can learn the AS path that the packet is expected to traverse, (2) Pathlet routing [11] or SCION [40] where the source can specify the path in the packet header, or (3) i3 [33] or Platypus [29] where the source can define a sequence of servers to traverse. Alternatively, an ISP may provide the premium path information to clients as an extra service (e.g., transatlantic cable for financial transactions [28]), in which case a client can validate that the traversed path is indeed the premium path paid for.

For the cryptographic key setup, the source and the destination need to be able to authenticate the router's cryptographic materials (i.e., validate a signature that binds an entity to some cryptographic materials). In the case of AS-level tracing, the AS needs to be authenticated, and such authentication can be achieved through RPKI [4], which is already operational. RPKI provides a PKI that enables authentication of AS certificates, each of which binds an AS number to its public key, given the correct RPKI public root key. In case of router-level tracing, we assume that each AS in turn creates certificates for each router using the AS's private key—enabling the tracing entity to verify via the AS certificate using RPKI.[1]

[1] Alternatively, OPT can authenticate entities based on mechanisms

Table 1: Notation.

(PK_E, PK_E^{-1})	Entity E's public-private key pair
$Cert_{PK_E}$	Entity E's public-key Certificate
$\hat{K}_{SD}$	Long-term symmetric key between S and D, which is valid over multiple sessions
K_E	Symmetric key among S, D, and entity E for a single session
$K_{E_1E_2}$	Symmetric key between entities E_1 and E_2 for a single session
$K_{E_1E_2\sigma}$	Symmetric key for E_1 and E_2 in session σ for E_1-initiated packets
SV_E	Entity E's local secret value
P	Network packet payload
$(PK_\sigma, PK_\sigma^{-1})$	Public-private key pair of session σ
$PATH_\sigma$	Session σ's path information
T_σ	Time when S initiates session σ
SESSIONID	Hash of session σ's public key, path, session initiation time
$AUTH_\sigma$	Authenticated and encrypted SESSIONID and private key for session σ
$SignKEY_{PK_E^{-1},\sigma}$	Signature on a symmetric key for session σ using entity E's private key
$EncKEY_{K,\sigma}$	Encryption of a symmetric key for session σ using key K
$KEYS_\sigma$	Set of shared keys between routers and S for session σ
DATAHASH	Hash of the packet's payload
PVF	Field enabling D to verify the path
PVF^S	Field enabling R_i and D to verify the path
PVF^D	Field enabling D to confirm the actual path
OV_i	Field enabling R_i to validate the packet sender
OPV_i	Field enabling R_i to verify both the packet sender and path
$Sign_{PK_E^{-1}}(\cdot)$	Signature using entity E's private key
$CheckSig_{PK_E^{-1}}(\cdot)$	Signature verification using entity E's private key
$Enc_K(\cdot), Dec_K(\cdot)$	Encryption, decryption using key K
$AuthEnc_K(\cdot)$	Authenticated encryption using key K
$AuthDec_K(\cdot)$	Authenticated decryption using key K
$F_K(\cdot)$	Pseudo-random function using key K
$MAC_K(\cdot)$	Message Authentication Code using key K
$H(\cdot)$	Cryptographic hash operation

We also assume that the source and destination entities that perform the tracing of the intermediate routers can establish a secret key between each other. In case the tracing entities are AS infrastructure hosts such as edge routers, firewalls, or a middlebox at a service provider, either RPKI can be used as described above or an administrator can set up trusted public keys between entities that need path verification. If endhosts perform tracing, then a shared key can be set up through SSL or TLS if one of the endhosts is a HTTPS server, through IPsec or SSH. The public keys can be verified through a regular PKI, administrator-based set up of trusted keys, TOFU (Trust On First Use) in SSH, TOFU with Perspectives [35], RPKI with domain-certified host keys, self-certifying IDs as public keys [1,23,24,34], or self-validation using an anonymous service [10]. We assume that one of these approaches is used to set up symmetric key $\hat{K}_{SD}$ between source S and destination D.

3.2 Main Insights

One of OPT's crucial insights is our approach to avoid storing per-flow state on routers; unlike prior approaches that require each router to maintain a secret key for each flow, our design enables routers to derive the secret keys on the fly using only local secrets stored at the routers and an efficient pseudo-random function. Thus, we avoid storing all the keys.

More precisely, OPT runs in *sessions*. In each session σ, source S sends packets to destination D on path $PATH_\sigma$. S and D leverage long-term symmetric key $\hat{K}_{SD}$ to set up keys with each router in $PATH_\sigma$. DRKey, the details of which is explained in Section 4, is instrumental in achieving OPT's efficiency properties. In particular, the source prepares and inserts a special field in the packet header called SESSIONID such that intermediate routers R_i on $PATH_\sigma$ dynamically compute the shared symmetric key with S and D (R_i only needs to look up its local secret SV_{R_i} for computation).

that use self-certifying IDs as public keys [1,23,24,34] as assumed in ICING. However, such mechanisms have issues with key revocations. Hence, we prefer to use RPKI.

Our key establishment mechanism does not require any per-flow state on the routers. Consequently, OPT is robust against DoS attacks based on state exhaustion. Moreover, computing pseudo-random function (PRF) F is faster than performing a cache access; for instance, a key derivation using AESni takes 32 cycles, whereas a L3 cache read operation requires approximately 40 cycles (on Intel "Sandy-Bridge"-based Xeon architecture).

OPT includes the hash of the packet payload $H(P)$ in the header, which enables an important optimization: routers can either parallelize the computations of MAC and the hash of the packet, or probabilistically validate $H(P)$.

3.3 OPT Protocol Overview

DRKey for path selection and key setup. When source S initiates session σ at time T_σ, S selects path $PATH_\sigma$ to destination D, generates asymmetric public/private key pair $(PK_\sigma, PK_\sigma^{-1})$, and creates a session identifier, where SESSIONID = $H(PK_\sigma \| PATH_\sigma \| T_\sigma)$. After preparing some values that support source authentication and path validation for other entities on $PATH_\sigma$, S forwards the OPT packet to its downstream router on $PATH_\sigma$. If T_σ is recent (i.e., within some predefined interval by the AS), each intermediate router R_i sets up shared symmetric key K_i using its local secret SV_{R_i} and SESSIONID. Detailed DRKey protocols are explained in Section 4.

Generation of verification fields. S uses the path information to pre-compute verification fields, one for each router R_i on $PATH_\sigma$, and a special field called PVF such that routers can perform source authentication and path validation.

Verification and update by intermediate routers. Upon receiving a packet, R_i first regenerates the shared symmetric key K_i and recomputes the verification field based on PVF. When the computed value matches what is present in the packet header for R_i, it successfully authenticates the source and the content of the packet, and validates the traversed path. R_i then updates PVF, by applying a MAC operation using K_i to the field. This process helps downstream routers and the destination to validate that each router on the path has indeed seen the packet.

Verification by destination. The destination finally recomputes the verification fields using all the symmetric keys shared with other entities on the path. Successful verification indicates source and packet content authentication as well as path validation.

4. DYNAMICALLY RECREATABLE KEYS

This section introduces the DRKey protocols that enable routers to set up shared keys with source S and destination D. Section 4.1 describes the case when both S and D trust each other. Section 4.2 relaxes this assumption and describes the case when S and D do not trust each other. Section 4.3 describes how S and D retroactively set up shared keys with intermediate routers to enable path validation of *prior* packets.

4.1 DRKey for Benign Source and Destination

When both S and D trust each other, each entity on the source-selected path needs to pre-establish only one symmetric key that is shared with both S and D. Figure 1 shows the key setup steps and the associated cryptographic operations.

S creates a fresh public/private key pair $(PK_\sigma, PK_\sigma^{-1})$ for each session such that routers encrypt session symmetric key K_i's. Since S and D trust each other, they share private key PK_σ^{-1}, the encrypted and authenticated value of which is sent to D. The public key is used to derive the SESSIONID as follows: SESSIONID = $H(PK_\sigma \| PATH_\sigma \| T_\sigma)$, where $PATH_\sigma$ is the source-selected path and T_σ is when S initiates session σ. Note that T_σ prevents replay attacks since routers can drop expired packets based on loose time synchronization.

Initialization by Source S

0. Assume long-term symmetric key $\hat{K}_{SD}$ shared with D
   (Optional) Assume public/private key pair (PK_S, PK_S^{-1}), and $Cert_{PK_S}$
1. Initiate new session σ and pick random session key $(PK_\sigma, PK_\sigma^{-1})$
2. Obtain $PATH_\sigma = \langle S, R_1, R_2, D \rangle$
3. Measure current time T_σ
4. Compute $\text{SESSIONID} = H(PK_\sigma \| PATH_\sigma \| T_\sigma)$
   (Optional) Authenticate session: $Sign_{PK_S^{-1}}(\text{SESSIONID})$
5. Compute $K_{SD\sigma} = F_{\hat{K}_{SD}}(S \| D \| \text{SESSIONID})$
   $K_{DS\sigma} = F_{\hat{K}_{SD}}(D \| S \| \text{SESSIONID})$
6. Compute $AUTH_\sigma = AuthEnc_{K_{SD\sigma}}(\text{SESSIONID} \| PK_\sigma^{-1})$

$S \rightarrow R_1$

7. Forward $\{PK_\sigma, PATH_\sigma, T_\sigma, AUTH_\sigma,$ (optional) $Sign_{PK_S^{-1}}(\text{SESSIONID})\}$

Pairwise Key Derivation by R_1

8. Compute $K_1 = F_{SV_{R_1}}(\text{SESSIONID})$
9. Encrypt K_1: $EncKEY_{R_1,\sigma} = Enc_{PK_\sigma}(K_1)$
   Sign: $SignKEY_{R_1,\sigma} = Sign_{PK_{R_1}^{-1}}(K_1 \| PK_\sigma)$

$R_1 \rightarrow R_2$

10. Forward $\{PK_\sigma, PATH_\sigma, T_\sigma, AUTH_\sigma,$ (optional) $Sign_{PK_S^{-1}}(\text{SESSIONID}),$
    $EncKEY_{R_1,\sigma}, SignKEY_{R_1,\sigma}\}$

Pairwise Key Derivation by R_2

11. Computes $K_2 = F_{SV_{R_2}}(\text{SESSIONID})$
12. Encrypt K_2: $EncKEY_{R_2,\sigma} = Enc_{PK_\sigma}(K_2)$
    Sign: $SignKEY_{R_2,\sigma} = Sign_{PK_{R_2}^{-1}}(K_2 \| PK_\sigma)$

$R_2 \rightarrow D$

13. Forward $\{PK_\sigma, PATH_\sigma, T_\sigma, AUTH_\sigma,$ (optional) $Sign_{PK_S}(\text{SESSIONID}),$
    $EncKEY_{R_1,\sigma}, SignKEY_{R_1,\sigma}, EncKEY_{R_2,\sigma}, SignKEY_{R_2,\sigma}\}$

Key Retrieval by Destination D

14. Already has $\hat{K}_{SD}$, which is the long-term shared symmetric key with S
15. Check that D is the last entity on $PATH_\sigma$
16. Compute $\text{SESSIONID} = H(PK_\sigma \| PATH_\sigma \| T_\sigma)$ and check the integrity
17. Compute $K_{SD\sigma} = F_{\hat{K}_{SD}}(S \| D \| \text{SESSIONID})$
    $K_{DS\sigma} = F_{\hat{K}_{SD}}(D \| S \| \text{SESSIONID})$
18. Decrypt $AUTH_\sigma$ and authenticate PK_σ^{-1}: $AuthDec_{K_{SD\sigma}}(AUTH_\sigma)$
19. Decrypt K_1 and K_2 and check their signatures:
    $Dec_{PK_\sigma^{-1}}(EncKEY_{R_1,\sigma}), CheckSig_{PK_{R_1}}(SignKEY_{R_1,\sigma})$
    $Dec_{PK_\sigma^{-1}}(EncKEY_{R_2,\sigma}), CheckSig_{PK_{R_2}}(SignKEY_{R_2,\sigma})$
    K_1 and K_2 become shared symmetric keys between each router and D
20. Compute $K_D = F_{SV_D}(\text{SESSIONID})$

$D \rightarrow S$

21. Forward authenticated and encrypted shared keys:
    $KEYS_\sigma = AuthEnc_{K_{DS\sigma}}(K_1 \| K_2 \| K_D \| AUTH_\sigma)$

Key Retrieval by Source S

22. Decrypt and authenticate the keys received from D: $AuthDec_{K_{DS\sigma}}(KEYS_\sigma)$
    K_1, K_2 and K_D become shared keys between S and R_1, R_2, and D

Figure 1: A session key setup for path $S \rightarrow R_1 \rightarrow R_2 \rightarrow D$.

Each intermediate router R_i generates shared key K_i using an efficient PRF keyed with a secret SV_i only known to R_i. The PRF takes SESSIONID as an input. For high efficiency, we compute our PRF from a pseudo-random permutation using AES. The overhead of the key setup is negligible to affect the on-going traffic (see Section 8). Resulting key K_i is encrypted with public key PK_σ, and digitally signed to enable verification that (encrypted) K_i indeed originates from the correct router (as any entity could have performed the public-key encryption on an arbitrary key).

To prevent reflection attacks (i.e., replaying message in the opposite order of communication), communication between S and D uses different symmetric keys for each direction: $K_{SD\sigma}$ and $K_{DS\sigma}$ for S-initiated and D-initiated packets, respectively.

The optional operations in Figure 1 are used only if the router also needs to authenticate S, in which case S also signs the SESSIONID, and certificates needed for routers to verify S's public key are included in the message.

4.2 Extended-DRKey for Distrusting Source and Destination

When we assume that the source S and the destination D trust each other and that they share the same public-private key pair for the session, then each intermediate router needs to set up only one shared key with both S and D. However, S and D may not necessarily trust each other, or they may collude. To strengthen the security guarantees under such circumstances, we introduce *Extended-DRKey* , which requires each intermediate router to set up two keys, K_{Si} and K_{Di}, where K_{Si} is the shared symmetric key between S and R_i and K_{Di} is the shared symmetric key between R_i and D. Figure 2 describes the Extended-DRKey protocol and its cryptographic operations.

Unlike DRKey in Section 4.1, Extended-DRKey does not require $AUTH_\sigma$ since R_i shares different keys with S and D. For creating shared keys (i.e., K_{Si} and K_{Di}), R_i and D use distinct local secrets to encode the directionality of the keys.

Path Agreement and Key Setup Initialization by Source S

1. Initiate new session $\sigma$and pick random session key $(PK_{S\sigma}, PK_{S\sigma}^{-1})$
   R_i uses this key to authenticate S's packets
2. Obtain $PATH_\sigma = \langle S, R_1, R_2, D \rangle$
3. Measure current time T_σ
4. Compute $\text{SESSIONID}_S = H(PK_{S\sigma} \| PATH_\sigma \| T_\sigma)$
   (Optional) Authenticate session: $Sign_{PK_S^{-1}}(\text{SESSIONID}_S)$

$S \rightarrow D$

5. Forward $\{PK_{S\sigma}, PATH_\sigma, T_\sigma, Sign_{PK_S^{-1}}(PK_{S\sigma} \| PATH_\sigma \| T_\sigma)\}$

Path Agreement and Key Setup Initialization by Destination D

6. Pick random session key $(PK_{D\sigma}, PK_{D\sigma}^{-1})$ that R_i uses with D
7. Compute $\text{SESSIONID}_D = H(PK_{D\sigma} \| PATH_\sigma \| T_\sigma)$
   (Optional) Authenticate session: $Sign_{PK_D^{-1}}(\text{SESSIONID}_D)$

$D \rightarrow S$

8. Forward $\{PK_{D\sigma}, PATH_\sigma, T_\sigma, Sign_{PK_D^{-1}}(PK_{S\sigma} \| PK_{D\sigma} \| PATH_\sigma),$
   $\text{SESSIONID}_D,$ (optional) $Sign_{PK_D^{-1}}(\text{SESSIONID}_D)\}$

Initialization by S

$S \rightarrow R_1$

9. Forward $\{PK_{S\sigma}, PK_{D\sigma}, PATH_\sigma, T_\sigma,$
   (optional) $Sign_{PK_S^{-1}}(\text{SESSIONID}_S), Sign_{PK_D^{-1}}(\text{SESSIONID}_D)\}$

Pairwise Key Derivation by R_1

10. Compute $K_{S1} = F_{SV_{R_1S}}(\text{SESSIONID}_S))$
    $K_{D1} = F_{SV_{R_1D}}(\text{SESSIONID}_D))$
11. Encrypt K_{S1}: $EncKEY_{R_1S,\sigma} = Enc_{PK_{S\sigma}}(K_{S1})$
    K_{S2}: $EncKEY_{R_1D,\sigma} = Enc_{PK_{D\sigma}}(K_{D1})$
12. Sign: $SignKEY_{R_1S,\sigma} = Sign_{PK_{R_1}^{-1}}(K_{S1} \| PK_{S\sigma} \| S)$
    $SignKEY_{R_1D,\sigma} = Sign_{PK_{R_1}^{-1}}(K_{D1} \| PK_{D\sigma} \| D)$

$R_1 \rightarrow R_2$

13. Forward $\{PK_{S\sigma}, PK_{D\sigma}, PATH_\sigma, T_\sigma,$
    (optional) $Sign_{PK_S^{-1}}(\text{SESSIONID}_S), Sign_{PK_D^{-1}}(\text{SESSIONID}_D),$
    $EncKEY_{R_1S,\sigma}, SignKEY_{R_1S,\sigma}, EncKEY_{R_1D,\sigma}, SignKEY_{R_1D,\sigma}\}$

Pairwise Key Derivation by R_2

14. Compute $K_{S2} = F_{SV_{R_2S}}(\text{SESSIONID}_S))$
    $K_{D2} = F_{SV_{R_2D}}(\text{SESSIONID}_D))$
15. Encrypt K_{S2}: $EncKEY_{R_2S,\sigma} = Enc_{PK_{S\sigma}}(K_{S2})$
    K_{S2}: $EncKEY_{R_2D,\sigma} = Enc_{PK_{D\sigma}}(K_{D2})$
16. Sign: $SignKEY_{R_2S,\sigma} = Sign_{PK_{R_2}^{-1}}(K_{S2} \| PK_{S\sigma} \| S)$
    $SignKEY_{R_2D,\sigma} = Sign_{PK_{R_2}^{-1}}(K_{D2} \| PK_{D\sigma} \| D)$

$R_2 \rightarrow D$

17. Forward $\{PK_{S\sigma}, PK_{D\sigma}, PATH_\sigma, T_\sigma,$
    (optional) $Sign_{PK_S^{-1}}(\text{SESSIONID}_S), Sign_{PK_D^{-1}}(\text{SESSIONID}_D),$
    $EncKEY_{R_1S,\sigma}, SignKEY_{R_1S,\sigma}, EncKEY_{R_1D,\sigma}, SignKEY_{R_1D,\sigma},$
    $EncKEY_{R_2S,\sigma}, SignKEY_{R_2S,\sigma}, EncKEY_{R_2D,\sigma}, SignKEY_{R_2D,\sigma}\}$

Key Retrieval by D

18. Check that D is the last entry in $PATH_\sigma$
19. Decrypt K_{D1} and K_{D2} and check their signatures
    $Dec_{PK_{D\sigma}^{-1}}(EncKEY_{R_1D,\sigma}), CheckSig_{PK_{R_1}}(SignKEY_{R_1S,\sigma})$
    $Dec_{PK_{D\sigma}^{-1}}(EncKEY_{R_2D,\sigma}), CheckSig_{PK_{R_2}}(SignKEY_{R_2S,\sigma})$
    K_{D1} and K_{D2} become shared symmetric key between each router and D
20. Compute $K_D = F_{SV_D}(\text{SESSIONID}_S))$
21. Encrypt K_D: $EncKEY_{D,\sigma} = Enc_{PK_{S\sigma}}(K_D)$
22. Sign: $SignKEY_{D,\sigma} = Sign_{PK_D^{-1}}(K_D \| PK_{S\sigma} \| S)$

$D \rightarrow S$

23. Forward $\{EncKEY_{R_1S,\sigma}, SignKEY_{R_1S,\sigma}, EncKEY_{R_2S,\sigma}, SignKEY_{R_2S,\sigma},$
    $EncKEY_{D,\sigma}, SignKEY_{D,\sigma}\}$

Key Retrieval by Source S

24. Decrypt and authenticate keys received from D
    K_{S1}, K_{S2} and K_D become shared keys between S and R_1, R_2, and D.

Figure 2: A modified session key setup when S and D do not trust each other. The path is: $S \rightarrow R_1 \rightarrow R_2 \rightarrow D$.

4.3 Retroactive-DRKey

The key setup protocols as presented in Figures 1 and 2 run *once* before the session starts. However, the key setup process incurs the following extra latency and computational overhead: (1) Key setup itself requires an extra round trip between the source and the destination; and (2) Key setup requires each intermediate router to encrypt and sign its shared key. Furthermore, running an independent key setup protocol *a priori* allows routers to launch a coward attack, since the key setup protocol warns possibly misbehaving routers to start behaving correctly and to avoid detection. Consequently, achieving path validation without the apparent key setup process is desirable.

We introduce Retroactive-DRKey that enables entities to set up shared keys at any time *after* the first packet in a session reaches the destination. Note that Retroactive-OPT is invoked only if the source or the destination wishes to perform source authentication or path validation (i.e., there could be sessions during which the source or the destination performs *no* retroactive key setup).

To support such a feature, we still assume that source S and destination D establish a shared symmetric key $\hat{K}_{SD}$ in advance, and S derives a session key pair $(PK_\sigma, PK_\sigma^{-1})$ before the session starts. Unlike DRKey or Extended-DRKey, Retroactive-DRKey utilizes that S creates K_D—a shared symmetric key with D for the session (i.e. $K_D = F_{SV_S}(\text{SESSIONID})$)—and includes encrypted and authenticated $K_D$in $AUTH_\sigma$ in the data packets of the forwarding protocol header. In this way, S can already use K_D to compute some fields such that D can check. When a forwarding protocol is used with Retroactive-DRKey, the routers use some keys for OPT during a session, and only reveal them at a later time (Section 5.2.1).

Retroactive-DRKey is very similar to the key setup protocol in Figure 1. The only difference is that D does not derive K_D, because it is already included in each forwarded packet. Retroactive-DRKey runs *at most once* during or after a session ends. We observe that S can set T_σ to a future time when S may want to trigger source and path validation if S of D detect an anomaly.

5. OPT PROTOCOL DESCRIPTION

The DRKey protocols described in Section 4 and the techniques we introduce in this section span a protocol family of source authentication and path validation with varying assumptions and properties. Unfortunately, exploring the entire design space is out of scope for this paper, and we will present several protocol instantiations: (1) *OriginValidation* for source authentication (S and D trust each other), (2) *PathTrace* for path validation (S and D trust each other), and (3) *Origin and Path Trace (OPT)* for source authentication and path validation (S and D may not trust each other).

5.1 OriginValidation for Source Authentication

OriginValidation enables each intermediate router and the destination to perform source authentication using MACs computed over the hash of the packet. For efficient authentication, the source includes the following fields in the packet header:

- DATAHASH: Hash of the packet's payload $H(P)$;
- SESSIONID: Hash of the session public key, path, and session initiation time $H(PK_\sigma \| PATH_\sigma \| T_\sigma)$;
- OV_i: Origin Verification field. OV_i is a Message Authentication Code computed over DATAHASH using key K_i that R_i shares with S (i.e., $\text{OV}_i = MAC_{K_i}(H(P))$). Similarly, $\text{OV}_D = MAC_{K_D}(H(P))$. The source creates an OV field for each intermediate router and the destination.

OriginValidation provides efficient MAC verification using the DATAHASH field without requiring each intermediate router to compute the hash over the entire packet. Figure 3 represents the packet header, and only DATAHASH, SESSIONID and OV fields are needed for OriginValidation.

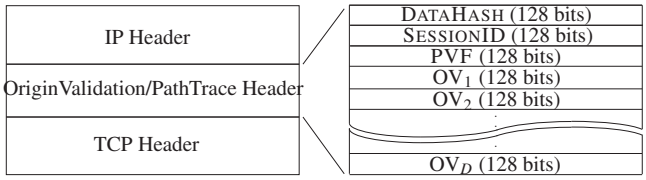


Figure 3: The packet header format for OriginValidation and PathTrace. DATAHASH, SESSIONID, and OVs help intermediate routers and the destination authenticate the source (OriginValidation), and DATAHASH, SESSIONID, and PVF help the destination validate the path (PathTrace).

When intermediate router R_1 receives a packet from the source S, R_1 computes the symmetric key (K_1) it shares with S using R_1's local secret and SESSIONID from the packet header. Then R_1 generates a MAC as follows: $\text{OV}'_1 = MAC_{K_1}(\text{DATAHASH})$. If OV'_1 is the same as OV_1 in the packet header, R_1 is assured that the packet indeed originated from the claimed source S, and forwards the packet to R_2. R_2 and D perform the same operations as R_1.

Although we present the protocol with OV fields of size 128, the size can be altered to reflect the desired level of security. In general, assuming a secure MAC function, the success probability of a forged n-bit MAC is 2^{-n}, which already results in a low rate at $n = 16$. Thus, for many applications, 2 byte long OV fields suffice, as a router would only let 1 in 65536 packets pass on average.

5.2 PathTrace

PathTrace is to help the source and destination validate that a received packet traversed the source-selected path. This main objective is achieved by Path Validation Field (PVF), which is a nested MAC that intermediate routers update in the packet header as they forward the packet. In Figure 3, only DATAHASH, SESSIONID, and PVF fields are used for PathTrace, thus, the packet overhead is irrespective of the path length. Next we describe how PathTrace supports the source and the destination to validate the path.

PathTrace for destination. To enable *only* the destination to validate the path, the source generates PVF_0—the initial PVF value which is a MAC of DATAHASH using the shared symmetric key between the source and the destination. Then the source initializes the PVF field in the header with PVF_0:

$$\text{PVF} \leftarrow \text{PVF}_0 = MAC_{K_D}(\text{DATAHASH}). \quad (1)$$

Any intermediate router R_i on the path generates PVF_i and updates the PVF field in the header as follows:

$$\text{PVF} \leftarrow \text{PVF}_i = MAC_{K_i}(\text{DATAHASH} \| \text{PVF}_{i-1}). \quad (2)$$

The shared symmetric key K_i is shared with *both* the source and the destination according to the key setup protocol in Section 4.1. Hence, upon receiving a packet, the destination first re-creates the nested MACs (here shown for a path of 2 routers):

$$\begin{aligned} \text{PVF}' &= MAC_{K_2}(\text{DATAHASH} \| \\ &\quad MAC_{K_1}(\text{DATAHASH} \| MAC_{K_D}(\text{DATAHASH}))). \end{aligned} \quad (3)$$

If PVF' is the same as PVF in the packet header, the destination is assured that the packet was indeed delivered on the source-selected path. Otherwise, the destination drops the packet.

PathTrace for source. To help the source authenticate that its packet is delivered to the intended destination using the source-selected path, the destination forwards the PVF from the received

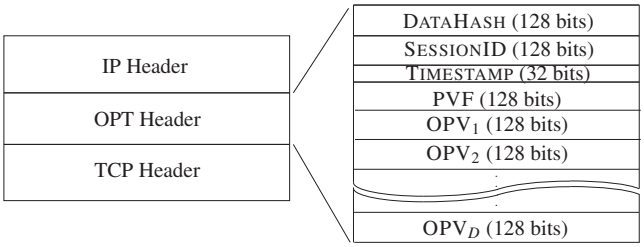


Figure 4: OPT header. The source S initializes all the fields. Intermediate routers only update the PVF field.

packet header back to the source as follows:

$$D \rightarrow S : Enc_{K_D}(\text{PVF} \| \text{DATAHASH}). \quad (4)$$

Upon receiving this information, the source first decrypts the message using K_D and then performs the validation by re-constructing the nested MACs using DATAHASH as shown in Eq. (3) and comparing it with PVF. A successful validation indicates that the packet was indeed delivered on the source-selected path.

5.2.1 Retroactive-PathTrace

Retroactive-PathTrace supports path validation without the apparent key setup process in advance. Instead, it utilizes Retroactive-DRKey that runs after the session ends. Unlike PathTrace, in Retroactive-PathTrace the source cannot pre-compute the OPV_i fields; hence no OPV fields can be used in the packet header. Instead, under the assumption that the source and the destination trust each other, Retroactive-PathTrace requires that the source creates K_D—the session-specific shared key with the destination, and the source additionally includes K_D and its authenticated encryption in the packet header. The source uses K_D to compute PVF_0 and the routers derive their shared key and update the PVF field accordingly.

Retroactive-PathTrace requires the destination to store per-packet information for later checking. However, the benefit of defending against coward attacks overcomes such a disadvantage. Namely, the destination stores for each packet the tuple (SESSIONID, DATAHASH,PVF). When the destination wants to validate the path, it requests the source to initiate Retroactive-DRKey such that intermediate routers reveal the key that was used for the received packets. Then the destination can check the PVF fields and detect coward attacks. The source can independently initiate the retroactive process as well.

5.3 OPT: Origin and Path Trace

In this section, we introduce OPT that combines OriginValidation and PathTrace such that all entities (including intermediate routers) on the path can perform both source authentication and path validation when they trust the source. We assume that all the routers in a session are loosely time synchronized (within a few milliseconds, e.g., using NTP).[2]

Figure 4 illustrates the OPT header format. In addition to DATAHASH, SESSIONID, and PVF, an OPT header includes the following additional fields to enable each intermediate router to perform path validation.

- TIMESTAMP: Time when S creates the OPT packet to mitigate timing-based attacks, such as replay attacks.
- OPV_i: Origin and Path Verification field. OPV_i is a MAC that enables all entities on the path to perform path validation.

[2] `http://www.ntp.org/`

Algorithm 1 OPT header initialization and validation pseudo code. An arrow represents the header field initialization.

function SOURCE INITIALIZATION
Require: K_i and K_D that R_i's and D share with S, respectively after running key setup protocol in Figure 1
  DATAHASH $\leftarrow H(P)$
  SESSIONID $\leftarrow H(PK_\sigma \| PATH_\sigma \| T_\sigma)$
  $\text{PVF} \leftarrow \text{PVF}_0 = MAC_{K_D}(\text{DATAHASH})$
  l = source-selected path length
  for each intermediate router R_i, where $1 \leq i < l$ **do**
    $\text{PVF}_i = MAC_{K_i}(\text{PVF}_{i-1})$
    $\text{OPV}_i \leftarrow MAC_{K_i}(\text{PVF}_{i-1} \| \text{DATAHASH} \| R_{i-1} \| \text{TIMESTAMP})$
  end for
  for destination D **do**
    $\text{OPV}_D \leftarrow MAC_{K_D}(\text{PVF}_{l-1} \| \text{DATAHASH} \| R_{l-1} \| \text{TIMESTAMP})$
  end for
  TIMESTAMP $\leftarrow$ current time
end function

function VALIDATION AND UPDATE BY R_i
  (Note PVF in OPT header = PVF_{i-1})
  Compute $\text{OPV}'_i = MAC_{K_i}(\text{PVF}_{i-1} \| \text{DATAHASH} \| R_{i-1} \| \text{TIMESTAMP})$
  if $\text{OPV}'_i == \text{OPV}_i$ **then**
    $\text{PVF} \leftarrow \text{PVF}_i = MAC_{K_i}(\text{PVF}_{i-1})$
    Forward the packet to R_{i+1}
  else
    Drop the packet
  end if
end function

function DESTINATION VALIDATION
  (Note PVF in OPT header = PVF_{l-1})
  l = source-selected path length
  Compute $\text{PVF}' = MAC_{K_{l-1}}(\ldots(MAC_{K_1}(MAC_{K_D}(\text{DATAHASH}))))$
  Compute $\text{OPV}'_D = MAC_{K_D}(\text{PVF}_{l-1} \| \text{DATAHASH} \| R_{l-1} \| \text{TIMESTAMP})$
  if ($\text{PVF}' == \text{PVF}$) && ($\text{OPV}'_D == \text{OPV}_D$) **then**
    Validation succeeds
    Prepare packet using Eq. (4) and forward to source
  else
    Drop the packet
  end if
end function

The SOURCE INITIALIZATION pseudo code in Algorithm 1 describes how the source initializes the OPT header fields. Each OPV field includes the following as inputs.

Previous PVF: Including PVF_{i-1} in the OPV_i computation supports the detection of a malicious intermediate router that forwards the packet to a benign router, which is not specified by the source but follows the protocol.

Previous router address: PVF by itself cannot support entities to detect the packet injection attack. Hence, we include the address of the previous router from which each entity receives the packet.

TIMESTAMP: This field mitigates authenticator cloning attacks. Consider an example where packet P_{crt} is expected to be sent along the source-selected path $PATH_{crt}$, the source previously sent packet P_{old} on $PATH_{old}$, and P_{crt} and P_{old} have the same payload. Consider router R_{bad} that is in both $PATH_{crt}$ and $PATH_{old}$ such that $PATH_{crt} = \{R_1, R_2, \ldots, R_{bad}, R_{bad+1}, \ldots, R_n\}$ and $PATH_{old} = \{R'_1, R'_2, \ldots, R_{bad}, R'_{bad+1}, \ldots, R_m\}$. In this scenario, R_{bad} can replace $\{R_{bad+1}, \ldots, R_n\}$ with $\{R'_{bad+1}, \ldots, R_m\}$ in $PATH_{crt}$ and all the corresponding fields in the P_{crt} header with those in P_{old}. Therefore, without TIMESTAMP, the destination cannot detect the misbehavior and ends up validating path $\{R_1, R_2, \ldots, R_{bad}, R'_{bad+1}, \ldots, R_m\}$ for P_{crt}. By setting the TIMESTAMP field when the source sends out a packet, authenticator cloning attacks are mitigated with loose time synchronization between the source and other entities on the path.

VALIDATION AND UPDATE BY R_i and DESTINATION VALIDATION functions in Algorithm 1 describe the OPT procedure that intermediate router R_i and the destination performs, respectively.

5.3.1 Distrusting source and destination

The previous protocols assumes that the source and the destination are honest and trust each other. We now relax this assumption and present an extension that handles distrusting entities. In OPT, the source can generate all PVFs by itself since it knows all K_i's. Consequently, a malicious source can collude with an intermediate router (e.g., R_2) and forward the packet on a path $S \rightarrow R_2 \rightarrow D$ without going through R_1 in Figure 1.

To prevent such an attack and address the problem of a distrusting source and destination, we use the key setup protocol in Section 4.2 such that intermediate routers generate two separate shared keys for the source and the destination. Unlike OPT, the Extended-OPT header requires two PVF fields: PVF^S that enables intermediate routers and the destination to validate the source, and PVF^D that enables the destination to confirm the *actual* path, even if the source is malicious and colludes with (at least) one intermediate router.

6. SECURITY ANALYSIS

We prove that OPT has origin authenticity and path validation properties when both the source and the destination are trusted. This property holds on any network configuration, including ones that have malicious routers. Extended-OPT offers stronger properties: the router's origin and path validation property assumes that only the source is honest; and the destination's path validation property does not assume the source is honest.

We describe how OPT and its variants defend against the adversary model described in Section 2.3. OPT's security properties have been formally verified using the Coq interactive theorem prover; details can be found in our technical report [17].

Packet alteration. Without the secret keys (K_D and K_i), a malicious router cannot compute valid PVF_i and OPV_i. Consequently, in OPT, a successful verification of PVF_{i-1} (PVF_n) based on OPV_i (OPV_D) implies that there can be no packet alteration attacks to router R_i (the destination), provided that the source and destination are trusted. A malicious destination can carry out the packet alteration attack, as it can generate all the necessary PVF and OPV fields. Extended-OPT provides similar protections for intermediary routers except that only the source needs to be trusted.

Packet injection attack. OPT routers can check that an incoming packet does come from an intended AS, as such information is included in OPV. Therefore, a malicious router A can only inject packet to a router B if A is B's neighbor and the link AB is on the intended path. We will revisit this attack when discussing collusion attacks.

In order to inject a packet with a valid header, an attacker can replay a previously seen packet, which is only possible when the same payload has been sent on that path before. Such replay attacks can be mitigated by including a timestamp in the packet. OPT is also vulnerable to packet injection when the destination colludes with the injecting router.

Path deviation attack. OPT ensures that a successful verification of PVF_{i-1} (PVF_n) against OPV_i (OPV_D) implies that the payload R_i (the destination) received has traversed all the honest routers in the source-intended path in the correct order, assuming that both source and destination are honest. OPT provides this guarantee because the attacker does not possess the secret keys required to compute PVF_i and OPV_i, and thus cannot generate valid PVF_i or OPV_i. As a result, malicious routers cannot mount router skipping or out-of-order traversal attacks.

This indicates that if a malicious router selects a path not intended by the source, an honest intermediary router will reject the packet. However, a malicious router can mount a path detour attack and send the payload to other routers that are not on the intended path.

In Extended-OPT, even if the destination is malicious, it cannot select the unauthorized path that drops or reorders honest routers. Extended-OPT also assures the destination that neither the malicious routers nor the malicious source can select a path that drops or reorders honest routers on the source-intended path.

Coward attack. Retroactive-OPT can mitigate coward attacks by requiring all forwarding routers to compute relevant PVF and OPV fields for probabilistic auditing. As a router cannot reliably guess when audits will happen, it does not know when to carry out an attack. We are unaware of any other path validation protocols that can defend against the coward attack, including ICING.

DoS attack. We consider attacks aiming to exhaust memory and computational resources on routers (Section 2.3). Each OPT router stores only a secret value (SV_{R_i}), regardless of the number of sources sending traffic and the number of flows transiting the router. For this reason, memory exhaustion attacks are not possible under OPT. OPT routers perform very few symmetric cryptographic operations per packet during forwarding, which run at line speed (Section 7). Therefore, OPT is more resilient to computation resource exhaustion attacks than existing schemes such as ICING that provide similar security guarantees.

Collusion. The path and source validation are conditioned upon whether a router is honest, i.e., correctly runs the protocol. When no two malicious routers are adjacent to each other on the intended path before R_i, malicious routers can redirect the packet to any routers as it chooses. However, all preceding links on the desired path are still traversed in the correct sequence for this packet to be accepted by R_i. Similarly, a malicious router could replace the path in the packet and trick its neighbor into forwarding the packet to a router outside the intended path. Again, this packet will be dropped when it reaches an honest router.

When there are multiple adjacent malicious routers on the intended path (R_{j1} to R_{jn}), a wormhole is present: an honest router down the path can only conclude that the packet has entered into the hole via R_{j1} and exited the hole from R_{jn}, but has no knowledge as to where the packet has been to in between these two points. In particular, when the source colludes with R_i, R_{i+1} can be tricked into accepting and forwarding any packet.

7. IMPLEMENTATION AND EVALUATION

We implemented OPT in Section 5.3 with DRKey as a user-level application that performs source authentication and path validation.

The cryptographic operations performed by a router during packet forwarding are purely symmetric cryptographic operations, which we instantiate with the AES block cipher. We list the cryptographic functions used in the implementation. To compute MACs, we used CBC-MAC based on AES, since it requires a single AES operation to authenticate a 128-bit value. For computing the PRF, we also use the same CBC-MAC. We implement AES using AESni, a new CPU instruction set provided by recent Intel and AMD CPUs to speed up AES operations. AESni is fast: According to Intel, executing an encryption using AES-128 in CBC mode takes 4.15 cycles per byte to encrypt a 1 KB buffer [13]. While this number bounds the processing latency per byte, router throughput can be increased, as we can process 4 blocks in parallel on a single core in AES-128 in CBC mode, resulting in 1.33 cycles per byte. We implemented authenticated encryption using Galois/Counter Mode (GCM) with AES.

We use SHA-3 for computing hashes on long strings, such as the hash of the payload DATAHASH. We truncate the hash from

Table 2: Per-session storage (σ), long-term storage (LT) related to the key setup, and communication overhead of DRKey's and ICING's key setup for a path of length n.

		Router	Source	Destination
Storage [#items]	DRKey (σ)	0	$n+2$	$n+2$
	ICING (σ)	2	$n+1$	$2*n+1$
	DRKey (LT)	1	1	2
	ICING (LT)	$\leq$ 400,000	0	0
Key setup [#packets]	DRKey	2		
	ICING	$4*n+4$		

a 256-bit value to a 128-bit value. For computing hashes on short strings, such as $H(PK_\sigma)$, we use the Merkle-Damgard construction with a Matyas-Meyer-Oseas AES-based compression function that makes use of the fast hardware AESni instructions. We choose a single-block-length compression function that outputs 128-bit hash values. We use 128-bit hash values to obtain a smaller OPT header size. Nevertheless, this decision does not pose security concerns: an adversary besides the source needs to perform a second-pre-image collision attack, which is still in the order of $O(2^{128})$ for SHA-3 and close to $O(2^{128})$ for Matyas-Meyer-Oseas.

For signatures, we use Ed25519 [6], providing high efficiency and security, and small signatures. For a security level of 2^{128} operations, Ed25519 signature generation and verification on a 3.4GHz Core i7 takes 20*us* and 60*us*, respectively. Public keys and signatures are only 32 and 64 bytes, respectively. Certificates can thus be as small as 128 bytes, enabling routers to add their certificate to the key setup message. As explained in Section 3, router certificates can be generated by an AS and signed with the AS's private key. The AS's public key can be obtained and verified through RPKI.

For encrypting routers' keys in DRKey, we use RSA-2048, which offers fast encryption. Although decryption is an order of magnitude slower, it is performed by the endhost which is less performance critical.

ICING implementation and configuration. We compare OPT with ICING, the code of which we obtained from their website[3]. To ensure the fairness of our comparison, we implemented ICING that also uses AESni. In ICING, the source obtains a proof of consent (PoC) from the consent server of each node on the path. A PoC certifies that the node consents to the full path. Furthermore, each node has an associated tag that describes a set of actions that the source can take for packets. The authors describe an optimization for computing the tag keys needed for the PoCs [25], which can diminish the number of required PRF rounds to 0. In our ICING implementation, we favor ICING and consider that computing these keys has no computational or memory lookup overhead.

Another important concept in ICING is the proofs of provenance (PoPs)—proofs to the nodes that the packet originates from the sender. Computing PoPs requires shared symmetric keys between each pair of ICING nodes on the path. These keys can be either derived on demand, using non-interactive Diffie-Hellman, or cached once computed. Since non-interactive Diffie-Hellman is expensive and impractical on the fast path, PoP keys are always retrieved from the cache in our ICING implementation.

7.1 DRKey Evaluation

DRKey enables the design of low-overhead protocols in terms of router resources, such as OPT. In OPT, we use DRKey for key setup, which is executed once per session. Thus, the key setup cost is amortized over an OPT session.

Table 2 provides an analysis of *storage overhead* at the source, destination, and intermediate routers for DRKey and ICING. We also compare the *communication overhead* for setting up keys.

Given a path of length n (excluding the source and the destination), DRKey requires the source and the destination to store, per session, $n+1$ symmetric keys and a public-private session key pair. The long-term storage of the source and the destination, which outlives multiple sessions, consists of their shared symmetric key and the destination's secret value. The OPT routers do not store any per-session information. Instead, the routers merely store a single local secret each.

In contrast, ICING's source and destination need to store $n+1$ symmetric keys. Each ICING router needs to store pairwise keys with every router in the Internet, which, according to the ICING authors, is within 400,000. The source also stores PoCs of all entities in the path.

Regarding the communication overhead of the key setup in DRKey, the source and the destination send one message each, resulting in 2 messages per session *regardless of the path length*. In ICING, however, the source contacts the PoC server of every node on the path (routers and destination), which leads to $2*(n+1)$ messages for a path of length n. The source also sets up pairwise shared keys with each entity on the path, requiring at least a round trip (2 messages) per entity, resulting in at least $2*(n+1)$ messages. We do not count the messages that are necessary to set up pairwise shared keys between ICING routers, because these keys are set up once between all entities in the Internet and then stored at each entity.

Table 3: Packet latency during DRKey processing.

Entity	Path length	Latency
Router	Irrelevant	381 μs
Source	2	621 μs
	4	609 μs
	8	628 μs
Destination	2	3820 μs
	4	5520 μs
	8	14814 μs

To prove the efficiency of DRKey, we evaluate the *per-packet computation overhead* at the source, destination and intermediate routers. Our experiment measures the latency of key setup packets while they transit the network entities. For the experiment, we use a traffic generator that initiates key setup operations and connects to a server that performs the key setup operations of the source, router, or destination. After the key setup, the server forwards the packets back to the traffic generator, which measures the receive rate.

Table 3 presents the latency of DRKey packets at the source, routers, and the destination. The results show that our cryptographic algorithm choices optimize router performance, which is in any case independent of path length.

For the source and the destination, a longer path increases the amount of computation. In the case of the source, this is hardly noticeable, because the source does not perform public-key cryptography operations that depend on the path length. In contrast, the destination performs per-hop public-key decryption using RSA-2048 to obtain the shared keys, which is expensive and considerably affects the latency. Nevertheless, the results satisfy our objectives: since the source and the destination have a significantly lower traffic throughput than routers, they can spend more cycles on performing key setup.

7.2 OPT Evaluation

We evaluate OPT with respect to the desired performance proper-

[3] `http://www.cs.utexas.edu/icing/`

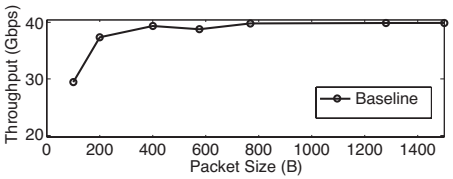


Figure 5: Throughput of I/O Engine. Since our experiments use I/O Engine for traffic delivery, I/O engine represents our baseline.

ties for source and path validation, namely *efficient forwarding* and *scalable state*, and the cost associated with meeting them. Specifically, we examine (1) OPT's overhead in terms of per-packet processing by measuring both the *throughput* (the bandwidth utilized by whole packets including the Ethernet header)[4] and *goodput* (the bandwidth used to transmit the payload of the packets, excluding the OPT protocol header); and (2) scalability with respect to the path length. For our evaluation, we set up a software router that validates the source and the path of the incoming packets before forwarding them. Our comparison is with ICING [26], which we discuss in more detail in Section 9 since both provide similar security guarantees. We experiment with OPT and ICING to perform source and path validations and we use PacketShader's I/O Engine [15] to send/receive packets to/from the NICs.

A central aspect of our work is the forwarding speed of a router. Since OPT does not keep any per-source or per-flow state, an OPT router's forwarding speed is not influenced by varying the number of sources sending packets or the number of flows transiting a router. In contrast, we analyze the impact of (1) cryptographic operations and (2) memory lookup of cryptographic keys, because the forwarding overhead of path validation protocols that use cryptography depends on these metrics.

Evaluation system. Our testbed consists of two routers A and B. Both are equipped with two Intel Ethernet Server Adapter X520-T2 NICs, and they both run Ubuntu Linux Kernel version 3.2.0-3. System A runs a traffic generator featuring 6 x 2GB DDR3 RAM and two Xeon L5640 2.26GHz (6 cores) processors. System B runs our software router code featuring 16 x 4GB DDR3 RAM and two Intel Xeon E5-2680 2.70GHz (8 cores) processors. The traffic generator generates traffic at a rate of 40*Gbps*, which is processed on the software router and sent back for measurement.

7.3 Experiment setup

We first describe how packets are forwarded from one router to the other. Then we describe the software router implementation for source authentication and path validation for OPT and ICING.

We use PacketShader's I/O Engine [15], a high-speed open source implementation to send/receive packets to/from the NICs. On the sending router, I/O Engine takes the packets generated by the user-level traffic generator and sends them to the NIC. When the packets arrive at the second router's NIC, I/O Engine takes the packets from the NIC and delivers them to the user-level application, where they are processed according to the protocol (OPT or ICING). The last step is to send these packets back to the first router. In the remainder of this section we consider the packet processing that takes place at the second router from the moment I/O Engine delivers the packets to the user-level application and until the packets are ready to be sent back.

Experiments. We measure the forwarding speed of the software router for OPT and ICING. Our experiments consider AS-level path validation scenarios, where path validation is performed at a single router (e.g., ingress router) within an AS. Since an AS is administered by a single authority and its internal routing topology is confidential, redundant path verifications are unlikely to happen inside an AS in practice. Consequently, we perform tests with a maximum path length of 10 hops (without counting the source). The minimum path length is 2 hops, corresponding to the case of a source, an intermediate hop, and a destination.

To measure the forwarding overhead at a router, which includes the memory overhead for storing keys and for retrieving them, we consider a network where each node has α neighbors. The parameter α is important only if the router performs work that depends on the number of neighbors. This is not the case for OPT, because the router merely computes the key it shares with the source to verify its OPV field, then computes the hash of the payload and finally updates the PVF field. These operations do not depend on the number of neighbors the router has nor on the path length.

However, in ICING the router has to look up the shared symmetric keys with each node on the path in a table that contains the keys of all the nodes the router had previously seen on a path. For a path of maximum length n, these nodes are located within n-hop distance away from the router. Our choice for the parameter $\alpha = 3$ and the maximum path length of 10 hops gives $\frac{(3^{11}-1)}{2} = 9841$ as a maximum key table size, which is within ICING's maximum key table size of $400,000$ [25].

The router receives packets at a line rate of 40*Gbps*. To quantify the throughput and goodput with respect to the overhead of OPT and ICING, we perform tests with payload sizes of 20*B*, 256*B*, 576*B*, 768*B*, and 1024*B*. We add to these values the OPT and ICING header overhead, respectively, whose size corresponds to the tested path length. The 20*B* of the smallest payload size correspond to the TCP header size to simulate TCP/ICING and TCP/OPT. We note that TCP/OPT includes the IP header since OPT runs over the IP network; whereas TCP/ICING does not include the IP header since ICING is designed to replace IP. Hence, the goodput computation favors ICING.

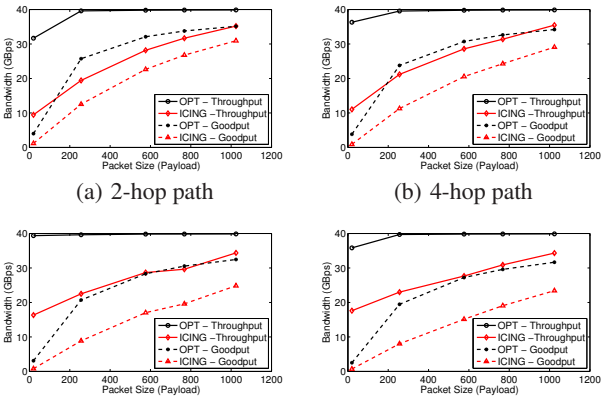
The biggest payload size is computed by subtracting from the MTU the header size of ICING for the longest tested path (10 hops), as ICING has a higher header overhead than OPT. More precisely, ICING has a per-hop overhead of 42*B* plus 24*B* for the source identifier and 13*B* of common fields, which gives a header length of 457*B* for a 10-hop path. The computation is $1500B - 457B = 1043B$, which explains our choice of 1024*B* of the maximum payload. In case of OPT, 52*B* common fields, 16*B* per-hop overhead and 40*B* TCP/IP header result in 252*B*. As a result, the maximum payload size is dictated by the size of the ICING header for the longest path considered.

Method. We generate traffic for 10 seconds at a rate of 40*Gbps*, which is forwarded to the router running the protocol for source authentication and path validation, and then forwarded back to the source. To measure the throughput, we employ I/O Engine's scripts, which operate as follows. These scripts read the RX and TX counter values of the NIC at the beginning of the experiment and then read the values again every second to compute the number of packets sent and received. Consequently, we obtain the throughput values every second, which we then average to compute the final throughput value. For goodput results we subtract the OPT or ICING header, respectively, from the packet size.

7.4 Forwarding Overhead

We evaluate the most computationally intensive protocol version of OPT described in Section 5.3. The evaluation results show that OPT outperforms ICING by a significant margin. Since the other versions of OPT feature smaller packet headers and less computa-

[4]We add 20B Ethernet overhead in computing the throughput.



(a) 2-hop path (b) 4-hop path

(c) 8-hop path (d) 10-hop path

Figure 6: Throughput and goodput (i.e., throughput obtained only for the payload part of the packet) of OPT and ICING for different packet sizes expressed in bytes and path lengths varying from 2 to 10 hops.

tional overhead, they would outperform ICING with an even larger margin.

Figure 6 depicts the results for throughput and goodput for OPT and ICING. We performed experiments for different path lengths and packet sizes described in Section 7.3, and the numbers we obtained show consistent results over all experiments, as explained in the next paragraphs.

A first observation is that throughput registers higher values than goodput, as expected, due to the OPT or ICING header overhead that adds to the packet size, yet is not accounted for when measuring goodput. We also notice that, for all path lengths, OPT's throughput is close to 40*Gbps* except for the smallest packet size (i.e., 20B). We note that OPT's throughput for small packets is mainly limited by I/O Engine's throughput as shown in Figure 5. As the path length grows, I/O Engine's bottleneck becomes released because of the reduced number of packet copies between the NIC and the user-level packet processing engine[5]; and as a consequence, OPT's throughput becomes close to 40*Gbps*. This result is consistent with Figure 5.

For each path length, the goodput of both OPT and ICING increase as the packet size increases, because the protocol header represents a smaller fraction of the total packet size as the payload size increases. Even though ICING's throughput also increases with the packet size, its value is much smaller than OPT's throughput. Given the choices for our ICING implementation, as explained earlier, this result is mainly due to the key table lookup for the PoP keys. Instead, OPT uses AESni operations to derive the keys shared with the source, on the fly. This choice in protocol design therefore proves to be decisive in obtaining a higher forwarding speed in OPT as much as 10*Gbps* in comparison to ICING.

7.5 Path Length Scalability

In order to analyze the protocols' scalability with respect to the path length, we depict in Figure 7 the ratio between the goodput and the throughput (named *goodput ratio*) for 256B and 1024B packets. We vary the path length from 2 to 10 hops.

The goodput ratios of 256B packets are lower those of than 1024B packets since small packets have higher header overhead (see previous section). Path length increase adds more overhead to the packet header, resulting in goodput degradation for both OPT and ICING.

[5]The increased header size reduces the number of packets needed to saturate the link bandwidth.

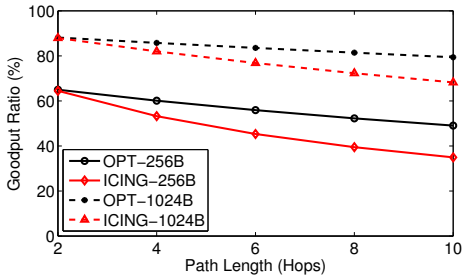


Figure 7: The goodput ratio (i.e., goodput/throughput) of OPT and ICING for small and large packets, in the context of path lengths varying from 2 to 10 hops.

The figure shows that OPT has better path length scalability than ICING since the goodput ratio of OPT decreases slower than that of ICING as the path length increases. Specifically, when the 2-hop path is used as a baseline, for 256B packets, OPT's average per-hop goodput degradation ratio is $\frac{(64.7-49.0)}{64.7 \cdot 8} = 3.03\%$, while ICING's is $\frac{(64.5-34.9)}{64.5 \cdot 8} = 5.74\%$; for 1024B packets, OPT's goodput degradation ratio per hop is $\frac{(88.1-79.3)}{88.1 \cdot 8} = 1.25\%$, while ICING's is $\frac{(87.9-68.2)}{87.9 \cdot 8} = 2.80\%$.

8. DISCUSSION

Key lifetime. The keys associated with a session σ are valid as long as (1) $PATH_\sigma$ between S and D in the session σ does not change, and (2) S or D do not terminate the session due to application-driven session lifetime requirement.

According to the first point, the maximum key lifetime is determined by route stability. Recent end-to-end route stability analyses reveal that most of network routes are stable from tens of minutes to days, even when ISPs apply traffic engineering techniques [9, 18, 22]. Although many routes are still short-lived, entities send packets over long-lived routes (longer than 6 hours) for 96% of the times [9]. In particular, considering the fact that the load balancing within an ISP causes most route variations (i.e., up to 82%), OPT running at AS-level uses more stable routes than router-level OPT. Furthermore, when routers perform per-flow or per-destination load balancing, paths used in OPT are not affected.

Yet an attacker could cause changes in network routes, as demonstrated by numerous incidents such as Man-in-the-Middle BGP route hijacking [8]. In this case, the network routes could flap as the attacker wishes. Yet, some future Internet architecture proposals (Nebula [2], Pathlets [12], SCION [40], XIA [14]) relieve this pain point by having packets carry forwarding information in the packet header, so that the source is always aware of the path and would set up a new session if the path changes.

We expect paths to be stable on the order of at least tens of minutes. Similarly, we expect that applications re-key sessions at the earliest at a granularity of tens of minutes. Thus, the key setup overhead represents a tiny fraction of the total computation and communication overhead of a long-lived high-bandwidth connection.

Efficient packet content authentication. As described in Section 5.3, each intermediate router uses the DATAHASH field in the OPT header when it verifies its OPV field. Such a verification does not authenticate the packet content since a malicious intermediate router could alter the data without changing DATAHASH. To mitigate such an attack, a router can verify the DATAHASH field in parallel while the packet is being scheduled for transmission.

As an alternative, probabilistic verification schemes [16] can be

applied such that every router decreases the verification probability if the DATAHASH verification succeeds. However, if a router detects a packet with a bogus hash value, the probability to run hash verification increases. Furthermore, as soon as a router receives multiple mismatching hash values, it immediately performs hash verification on all subsequent packets. As a result, the routers neighboring a malicious router would verify all hashes of packets incoming on interfaces arriving from the malicious router. With such a probabilistic verification approach, we can further improve the efficiency and practicality while providing data authentication.

OPT in the current Internet. OPT could be incrementally deployed in the current Internet. An AS could announce its OPT functionality within BGP update messages (as a transitive attribute) or as extension to RPKI certificates, enabling the selection and construction of end-to-end OPT paths at source ASes. Endhosts could obtain OPT path information from a local route server which collects BGP and RPKI information.

To carry OPT-based information in packets, the simplest approach would be an IPv6 extension header. In IPv4, spare IP header bits would need to be used to indicate the presence of an extra OPT header or trailer, but an extra header after the IP header may disrupt processing at legacy firewalls or other middleboxes. With the increasing support for IPv6, we prefer incremental deployment via the IPv6 extension header. Since the DRKey information is larger than the 256 bytes that fit into an IPv6 extension header, we propose to use a new protocol number for DRKey packets, which routers would check for and process in the slow path. Alternatively, ASes could specify addresses of DRKey servers that would handle DRKey packets to set up the keys for the routers and thus would share the secret keys K_{R_i} of routers. An endhost could then place a sequence of DRKey server addresses (similar to a loose source routing option in IPv4) into the DRKey packet, which would be sequentially processed and forwarded until the destination. This latter approach avoids routers from analyzing the IP protocol field.

9. RELATED WORK

The most closely related work for source authentication and path validation is **ICING** by Naous et al. [26]. In a nutshell, the source pre-computes a verifier MAC (V_i) for each intermediate router R_i using the respective shared secret key as well as the hash value of the path and the static content in the header. For each packet, R_i first reconstructs and XORs the MAC for the source and each upstream router, and verifies if the XORed MACs are equivalent to what is stored in V_i. Then R_i (1) computes a MAC for each downstream router on the path using the shared secret key, the hash of the path, and the data, and (2) XORs the MAC for each downstream router with each V_j ($j \geq i$).

ICING is more heavy-weight than OPT. ICING requires each R_i to derive a Diffie-Hellman (DH) key with each router R_j on the path, which requires routers to cache keys to avoid the heavy-weight DH computation during packet forwarding. For the case the keys are not cached any more, ICING suggests adding the 20-byte public key of each router into each packet, resulting in a high per-packet overhead. Also, ICING requires each node to insert a MAC for each subsequent verifier, requiring each node to compute $n+1$ MAC operations per packet. In contrast, OPT does not require routers to store keys shared with sources or other routers, nor perform a MAC computation for each router on the path. In terms of security, even ICING intermediate routers can detect colluding path deviation attacks mounted by the source and a malicious router to another intermediary router. In contrast, Extended-OPT supports the destination to detect such attacks. More specifically, the origin and path validation property of the routers still depends on the correct behavior of the source, but not on the destination. Consequently, Extended-OPT offers the same security guarantee as ICING for the destination. In other words, if the source is malicious, an illegitimate packet travels longer in Extended-OPT than ICING, but will be rejected by the destination. Beside the malicious source collusion attack, OPT and ICING provide the same kind of source and path authenticity properties for the destination. On the other hand, retroactive key setup OPT with path tracing (which only enables the destination to verify the path) can mitigate the coward attack, which ICING fails to mitigate.

Liu et al. propose **Passport** for intermediate routers to perform source authentication [21]. Passport proposes that BGP route advertisements carry Diffie-Hellman public keys, which enables any two ASes to compute a shared secret based on Diffie-Hellman key exchange. Using the respective shared secret key with each downstream AS, the source AS computes a MAC for each AS on the path, and inserts it in the Passport header. Each intermediate AS authenticates the source AS by recomputing the MAC using the shared key and confirming that it matches the MAC in the Passport header. Similar to Passport, **the accountability service** by Bender et al. [5] requires the source AS to embed an authenticator for subsequent ASes. In **SNAPP**, the sender sets up keys with routers through a key derivation mechanism that is similar to DRKey (although it cannot provide retroactive key setup) and uses these keys for embedded source authenticators. Passport and the accountability service provide weaker security guarantees than OPT, as they provide only source AS authentication, and fail to defend against source and data spoofing, as well as path deviation attacks. While SNAPP does prevent against source and data spoofing, it does not prevent path deviation attacks.

Other source and path validation schemes. In IP traceback, researchers proposed mechanisms that require routers carry per-packet state [19, 31], perform packet marking [30, 32], or active path interrogation [27]. Pi suggests a path identifier to detect source IP address spoofing [36]. Unfortunately, these schemes are susceptible to attacks listed in Section 2.3, because they were designed for a different purpose. Similarly, network capability mechanisms [3, 29, 37, 38] cannot provide source authentication or path validation as the capability can be easily copied and inserted by the last AS.

10. CONCLUSION

Despite the importance of network-based source authentication and path validation, these primitives have not been implemented so far, perhaps because of the lack of an efficient protocol that does not burden the router. This paper introduces (1) DRKeys as efficient and dynamically recreatable key setup protocols, and (2) OPT as an extremely lightweight, scalable, and secure protocol that provides source authentication and path validation. Compared with currently proposed solutions, OPT achieves performance improvements with minimal latency and computational overhead on routers regardless of the path length. Moreover, OPT does not require routers to maintain per-source or per-flow state, further improving its practicality. We also introduce a retroactive key setup process that protects against coward attacks, as routers cannot know in advance which paths are being monitored subsequently. We anticipate that OPT's security and performance properties will bring source authentication and path validation into the realm of practicality.

11. ACKNOWLEDGMENTS

We thank George Danezis, Yue-Hsun Lin, Raphael Reischuk, members of the XIA team, our shepherd Ratul Mahajan, and the anonymous reviewers for their insightful feedback and suggestions.

We gratefully acknowledge funding support for this research from CyLab at Carnegie Mellon, NSF under award CNS-1040801, European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement 617605, and a gift from KDDI.

12. REFERENCES

[1] D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker. Accountable Internet Protocol (AIP). In *Proceedings of ACM SIGCOMM*, 2008.

[2] T. Anderson, K. Birman, R. Broberg, M. Caesar, D. Comer, C. Cotton, M. Freedman, A. Haeberlen, Z. Ives, A. Krishnamurthy, W. Lehr, B. Loo, D. Mazières, A. Nicolosi, J. Smith, I. Stoica, R. van Renesse, M. Walfish, H. Weatherspoon, and C. Yoo. The nebula future internet architecture. In A. Galis and A. Gavras, editors, *The Future Internet*, volume 7858 of *Lecture Notes in Computer Science*, pages 16–26. Springer Berlin Heidelberg, 2013.

[3] T. Anderson, T. Roscoe, and D. Wetherall. Preventing Internet Denial-of-Service with Capabilities. In *Proceedings of Hotnets-II*, 2003.

[4] ARIN. Resource Public Key Infrastructure (RPKI). `https://www.arin.net/resources/rpki/`.

[5] A. Bender, N. Spring, D. Levin, and B. Bhattacharjee. Accountability as a Service. In *Proc. of USENIX SRUTI*, 2007.

[6] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang. High-speed high-security signatures. In *Proc. of CHES*, 2011.

[7] J. Cowie. The new threat: Targeted internet traffic misdirection. `http://www.renesys.com/2013/11/mitm-internet-hijacking/`, Nov. 2013.

[8] J. Cowie. The New Threat: Targeted Internet Traffic Misdirection. Popular Mechanics, `http://www.renesys.com/2013/11/mitm-internet-hijacking/`, Nov 2013.

[9] I. Cunha, R. Teixeira, and C. Diot. Measuring and characterizing end-to-end route dynamics in the presence of load balancing. In *Proc. of PAM'11*, 2011.

[10] Y. Gilad and A. Herzberg. Plug-and-Play IP Security: Anonymity Infrastructure Instead of PKI. In *Proceedings of ESORICS*, 2013.

[11] B. Godfrey, S. Shenker, and I. Stoica. Pathlet Routing. In *Proc. of SIGCOMM*, 2009.

[12] P. B. Godfrey, I. Ganichev, S. Shenker, and I. Stoica. Pathlet routing. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, 2009.

[13] S. Gueron. Intel Advanced Encryption Standard (AES) New Instructions Set, Mar. 2010. white paper 323641-001, Revision 3.

[14] D. Han, A. Anand, F. Dogar, B. Li, H. Lim, M. Machado, A. Mukundan, W. Wu, A. Akella, D. G. Andersen, J. W. Byers, S. Seshan, and P. Steenkiste. XIA: Efficient support for evolvable internetworking. In *Proceedings of USENIX Conference on Networked Systems Design and Implementation*, 2012.

[15] S. Han, K. Jang, K. Park, and S. Moon. PacketShader: a GPU-accelerated software router. *ACM SIGCOMM Computer Communication Review*, Aug. 2010.

[16] H.-C. Hsiao, A. Studer, C. Chen, A. Perrig, F. Bai, B. Bellur, and A. Iyer. Flooding-Resilient Broadcast Authentication for VANETs. In *Proc. of MobiCom*, 2011.

[17] L. Jia, C. Basescu, T. H.-J. Kim, A. Perrig, Y.-C. Hu, and F. Zhang. Mechanized network origin and path authenticity proofs. Technical Report CMU-CyLab-14-007, Carnegie Mellon University, 2014.

[18] M. S. Kang, S. B. Lee, and V. D. Gligor. The Crossfire Attack. In *Proc. of IEEE Security and Privacy*, 2013.

[19] J. Li, M. Sung, J. Xu, and L. Li. Large-Scale IP Traceback in High-Speed Internet: Practical Techniques and Theoretical Foundation. In *Proc. of IEEE Security and Privacy*, 2004.

[20] B. Liu, J. T. Chiang, J. J. Haas, and Y.-C. Hu. Coward Attacks in Vehicular Networks. *Mobile Computing and Communications Review*, 2010.

[21] X. Liu, A. Li, X. Yang, and D. Wetherall. Passport: Secure and Adoptable Source Authentication. In *Proc. of NSDI*, 2008.

[22] H. V. Madhyastha, E. Katz-Bassett, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iPlane Nano: Path Prediction for Peer-to-peer Applications. In *Proc. of NSDI*, 2009.

[23] D. Mazieres, M. Kaminsky, M. F. Kaashoek, and E. Witchel. Separating Key Mangement from File System Security. In *Proceedings of SOSP*, 1999.

[24] R. Moskowitz and P. Nikander. *Host Identity Protocol (HIP) Architecture*, May 2006. `http://tools.ietf.org/html/rfc4423`.

[25] J. Naous. Path-policy Compliant Networking and a Platform for Heterogeneous IAAS management. In *PhD thesis*, 2011.

[26] J. Naous, M. Walfish, A. Nicolosi, D. Mazieres, M. Miller, and A. Seehra. Verifying and Enforcing Network Paths with ICING. In *Proceedings of ACM CoNEXT*, 2011.

[27] V. N. Padmanabhan and D. R. Simon. Secure Traceroute to Detect Faulty or Malicious Routing. *ACM SIGCOMM Computer Communications Review*, January 2003.

[28] J. Pappalardo. New Transatlantic Cable Built to Shave 5 Miliseconds off Stock Trades. Popular Mechanics, `http://www.popularmechanics.com/technology/engineering/infrastructure/a-transatlantic-cable-to-shave-5-milliseconds-off-stock-trades`, Oct 2011.

[29] R. Raghavan and A. C. Snoeren. A system for authenticated policy-compliant routing. In *Proc. of ACM SIGCOMM*, 2004.

[30] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical Network Support for IP Traceback. In *Proc. of SIGCOMM*, 2000.

[31] A. C. Snoeren, C. Partridge, L. A. Galindo, C. E. Jones, F. Tchakounito, S. T. Kent, and W. T. Strayer. Hash-Based IP Traceback. In *Proceedings of ACM SIGCOMM*, 2001.

[32] D. X. Song and A. Perrig. Advanced and Authenticated Marking Schemes for IP Traceback. In *Proceedings of IEEE INFOCOM*, 2001.

[33] I. Stoica, D. Adkins, S. Zhaung, S. Shenker, and S. Surana. Internet indirection infrastructure. In *Proc. of SIGCOMM*, 2002.

[34] M. Walfish, J. Stribling, M. Krohn, H. Balakrishnan, R. Morris, and S. Shenker. Middleboxes No Longer Considered Harmful. In *Proceedings of OSDI*, 2004.

[35] D. Wendlandt, D. G. Andersen, and A. Perrig. Perspectives: Improving SSH-style host authentication with multi-path probing. In *Proceedings of USENIX Annual Technical Conference*, June 2008.

[36] A. Yaar, A. Perrig, and D. Song. Pi: A Path Identification Mechanism to Defend against DDoS Attacks. In *Proc. of IEEE Security and Privacy*, 2003.

[37] A. Yaar, A. Perrig, and D. Song. An Endhost Capability Mechanism to Mitigate DDoS Flooding Attacks. In *Proc. of the IEEE Security and Privacy*, May 2004.

[38] X. Yang, D. Wetherall, and T. Anderson. A DoS-limiting Network Architecture. In *Proc. of SIGCOMM*, 2005.

[39] X. Zhang. *Secure and Efficient Network Fault Localization*. PhD thesis, Carnegie Mellon University, 2012.

[40] X. Zhang, H.-C. Hsiao, G. Hasker, H. Chan, A. Perrig, and D. G. Andersen. SCION: Scalability, control, and isolation on next-generation networks. In *Proceedings of the IEEE Symposium on Security and Privacy (Oakland)*, May 2011.