Tobias Klenze

Formal Development of Secure Data Plane Protocols

Diss. ETH No. 27649

DISS. ETH NO. 27649

FORMAL DEVELOPMENT OF SECURE DATA PLANE PROTOCOLS,

A dissertation submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH
(Dr. sc. ETH Zurich)

presented by

TOBIAS KLENZE
M. Sc. in Computer Science, TU Munich

born on 15 March 1990
citizen of Germany

accepted on the recommendation of

Prof. Dr. David Basin, examiner
Prof. Dr. Adrian Perrig, co-examiner
Dr. Christoph Sprenger, co-examiner
Prof. Dr. Véronique Cortier, co-examiner

2021

DOI: 10.3929/ethz-b-000506662

Abstract

Today's Internet is built on decades-old networking protocols that lack scalability, reliability and security. In response, the networking community has developed *path-aware* Internet architectures that solve these issues while simultaneously empowering end hosts. In these architectures autonomous systems authorize forwarding paths in accordance with their routing policies, and protect paths using cryptographic authenticators. For each packet, the sending end host selects an authorized paths and embeds it and its authenticators in the header. This allows routers to efficiently determine how to forward the packet. The central security property of the data plane, i.e., of forwarding, is that packets can only travel along authorized paths. This property, that we call *path authorization*, protects the routing policies of autonomous from malicious senders. Further security properties are *source authentication*, which allows routers and the destination to authenticate the sender, and *path validation*, which allows the sender and the destination to verify that the packet traversed the path embedded in the header.

Existing schemes that achieve these properties require long authenticators to be embedded in each packet's header. We propose EPIC, a family of protocols that achieve the same guarantees with substantially reduced overhead.

The fundamental role of packet forwarding in the Internet's ecosystem and the complexity of the authentication mechanisms employed call for a formal analysis of EPIC, and of related protocols. We develop a parameterized verification framework for data plane protocols in Isabelle/HOL. We first formulate an abstract model without an attacker for which we prove path authorization. We then refine this model by introducing a Dolev–Yao attacker and by protecting authorized paths using (generic) cryptographic validation fields. This model is parametrized by the path authorization mechanism and assumes five simple verification conditions that allow us to prove the refinement of the abstract model. We propose two novel attacker models and two sets of assumptions on the underlying routing protocol that allow us to classify protocols and distinguish them by how strong their guarantees are. We validate our framework by instantiating it with several concrete protocols and proving that they each satisfy the verification conditions (and hence path authorization). Invariants needed for the security proof are proven in the parametrized model; instances do not need to show invariants. Our framework thus supports low-effort security proofs for data plane protocols. We extend our framework in multiple ways, to allow a wide range of protocols and different attackers to be modeled. Our verification results hold for arbitrary network topologies and sets of authorized paths, a generality that state-of-the-art automated protocol verifiers cannot currently provide.

Zusammenfassung

Das heutige Internet basiert auf Jahrzehnte alten Netzwerkprotokollen, denen es an Skalierbarkeit, Zuverlässigkeit und Sicherheit mangelt. Die Netzwerk-Gemeinschaft hat daraufhin Internet-Architekturen geschaffen, die *path-aware* sind, und damit diese Probleme lösen und gleichzeitig Endhosts mehr Gewicht verschaffen. In diesen Architekturen können Autonome Systeme Forwarding-Pfade gemäss ihrer Routing-Regeln autorisieren und durch kryptografische Authentifikatoren schützen. Der sendende Endhost selektiert für jedes Paket einen dieser autorisierten Pfade und bettet ihn zusammen mit den Authentifikatoren in den Header ein. Dies erlaubt es Routern effizient zu bestimmen, wie das Paket weiterzuleiten ist. Die zentrale Sicherheitseigenschaft der Data Plane d.h. des Forwardings ist, dass sich Pakete nur entlang autorisierter Pfade bewegen. Diese Eigenschaft, welche wir *Pfadautorisierung* nennen, schützt die Routing-Regeln von Autonomen System vor angreifenden Sendern. Weitere Sicherheitseigenschaften sind *Quellenauthentifizierung*, welche es Routern und dem Empfänger ermöglichen den Sender zu authentifizieren, und *Pfadvalidierung*, welche es dem Sender und dem Empfänger ermöglichen zu verifizieren, dass der Pfad den das Paket zurückgelegt hat, dem im Header eingebetteten Pfad entspricht.

Bestehende Protokolle die diese Eigenschaften erfüllen setzen voraus, dass lange Authentifikatoren in die Header jedes einzelnen Paketes eingebettet werden. Wir schlagen EPIC vor, eine Familie an Protokollen, welche die gleichen Garantien unter deutlich weniger Aufwand erreichen.

Die elementare Rolle von Paket-Forwarding im Internet und die Komplexität der Authentifizierungs-Mechanismen erfordern eine formale Analyse von EPIC und von ähnlichen Protkollen. Wir entwickeln ein parametrisches Verifikations-Framework für Data-Plane-Protokolle in Isabelle/HOL. Wir entwickeln zuerst ein abstraktes Modell ohne Angreifer, für welches wir Pfadautorisierung zeigen. Wir verfeinern dann dieses Modell indem wir einen Dolev–Yao-Angreifer hinzufügen, und indem wir autorisierte Pfade durch (generische) kryptografische Validierungsfelder schützen. Dieses Modell ist über den Pfadautorisierungs-Mechanismus parametrisiert und nimmt fünf einfache Verifikationsbedingungen an, mit deren Hilfe wir die Verfeinerung des abstrakten Modells zeigen können. Wir schlagen zwei neue Angreifermodelle und zwei Arten von Annahmen über das zugrundeliegende Routing-Protokoll vor, welche es uns ermöglichen Protokolle zu klassifizieren und zu bestimmen wie stark ihre Garantien sind. Wir validieren unser Framework indem wir es mit mehreren konkreten Protokollen instanziieren und zeigen, dass sie jeweils die Verifikationsbedingungen (und damit Pfadautorisierung) erfüllen. Invarianten, welche für den Sicherheitsbeweis notwendig sind, werden im parametrischen Modell bewiesen. Instanzen müssen keine Invarianten zeigen. Unser Framework erlaubt daher die Sicherheit von Data-Plane-Protokollen ohne grosse Mühe zu beweisen. Wir erweitern unser Framework auf mehrere Arten um viele verschiedene Protokolle und Angreifer modellieren zu können. Unsere Verifikationsresultate sind für beliebige Netzwerktopologien und Mengen an autorisierten Pfaden gültig. Diese Generalität kann von derzeitigen automatisierten Protokoll-Verifikations-Tools nicht geleistet werden.

Acknowledgements

First, I would like to thank my advisor, David Basin, for providing me with the opportunity to pursue a PhD in his group, for always offering his assistance, and for his steady guidance through the past five years. My appreciation goes to Adrian Perrig for his support in my projects and his contagious enthusiasm about transforming the Internet. I am thankful to Véronique Cortier for taking the time to carefully review this thesis.

I was very fortunate to work with Giacomo Giuliari, Markus Legner and Zuzana Frankovska, who were all very talented students. In particular, it was a great pleasure to continue to work with Markus Legner, with whom I had many fruitful discussions and who provided helpful feedback on this thesis. My gratitude also goes to Benjamin Rothenberger, Kenny Paterson, Marc Wyss, Sofia Giampietro, Samuel Hitz, Sergio Monroy and Yih-Chun Hu for their insights, contributions, and help with manuscripts.

My deepest gratitude and admiration goes to Christoph Sprenger, not only because his exceptional grasp of formal methods and his support throughout my PhD were instrumental to the success of our work, but also because his diligence, integrity, mentorship and humble dedication to science were profoundly inspiring to me.

Lastly, I would like to thank the people around me for supporting me in the years of my PhD and before. Time flew by, thanks to my colleagues and friends Reza, Karel and Flo, with whom I frequently had inspiring discussions and wonderful adventures. My heartfelt thanks goes to my parents, who encouraged me to embark on this path and always supported me alongside it. Most importantly, my deepest appreciation goes to Fangwen, for her love, for making me a better person, and for all the good moments that we shared in these years.

Contents

Notation

A_i	AS corresponding to the ith hop on the path; H_{S} and H_{D} are located in A_1 and A_ℓ, respectively
C_i	cryptographic result used for authenticating and updating the ith hop fields
H_{S}, H_{D}	source and destination hosts of a packet
HI	hi_i is hop information of AS i consisting of ts_{exp}, ingress interface, and egress interface
HVF	hop validation field, a field that protects a hop field cryptographically. In EPIC, this is defined as $hvf_i = \langle V_i, S_i \rangle$.
K_i	secret symmetric key of A_i
K_i^{S}	host key shared between A_i, A_1, and H_{S}, which can be efficiently calculated by A_i
$\mathscr{I}$	set of interfaces
K_{SD}	key shared between A_1, H_{S}, A_ℓ, and H_{D}
ℓ	AS-level path length
l_{val}, l_{seg}	length in bytes of V_i, S_i
l_{PRF}	block size in bytes of $\mathsf{PRF}(\cdot)$ and $\mathsf{MAC}(\cdot)$
$\mathsf{MAC}_K(\cdot)$	message authentication code using key K
$\mathbb{N}$	set of natural numbers
$\mathscr{N}$	set of nodes
P, $p = \lvert P \rvert$	packet payload and payload size
PO	packet origin consisting of $\textsc{Src}$, TS_{path}, and ts_{pkt}
$\mathsf{PRF}_K(\cdot)$	pseudorandom function using key K
$S_i^{(l)}$	segment identifier in protocol level l allowing ASes to chain hops to paths

$\sigma_j^{(l)}$	hop authenticator in level l authorizing the jth hop as calculated by A_j during path exploration
Src	$(A_1,\ H_S)$; source AS and host address
TS_{path}	path timestamp created during path exploration
ts_{exp}	expiration time of a hop field relative to TS_{path}
ts_{pkt}	packet creation time relative to TS_{path}
$V_{i;j}^{(l)}$	validator in protocol level l corresponding to the ith hop after processing by A_j; when its value stays constant, we omit j.
V_{SD}	destination validation field

Mathematical Notation

$A \times B$, A^*	cartesian product, finite sequences
$\mathcal{P}(A)$, $A_\perp$	powerset, option type (sum of A and $\{\perp\}$)
$A \to B$	total function
$A \rightharpoonup B$	partial function
$dom(f)$, $ran(f)$	function domain and range
$(\!\| x \in A, y \in B \|\!)$	set of records
$(\!\| x = a, y = b \|\!)$	concrete record
$x(r)$, $r(\!\| x := v \|\!)$	record field x access, and update
$f(x := v)$	function update
$\langle\rangle$, $x \# xs$, $\langle a, b, c\rangle$	empty, cons, concrete sequence
$xs \leq ys$, $x \in xs$	sequence prefix, sequence membership
$hd(xs)$, $tl(xs)$	list head and tail if cons, else $\perp$
$xs \cdot ys$, $rev(xs)$	concatenation of string or sequence, sequence reversal
$xs[\![i{:}j]\!]$	substring from position i (incl.) to position j (excl.) of sequence xs

1 Introduction

While the advent of the Internet is undoubtedly one of the most significant developments in human history, its quick popularization has created substantial technical challenges and raised profound societal questions. These issues have reached the mainstream, and concerns about disinformation and abuse on the Internet are frequent topics in popular media. Less attention has been paid to questions of the governance, security and reliability of the underlying infrastructure of the Internet, in particular the network layer. Since all Internet applications rely on this infrastructure, these questions also deserve our attention.

Researchers have in recent years suggested that an answer should come in the form of a reliable, open, decentralized and secure future Internet architecture. This thesis works towards that goal.

We begin with a short history of the Internet and how its organic growth from a small, trusted network into a global system with adversarial actors allowed the problems to develop that the Internet is facing today. These problems motivate the idea for new architectures that are secure from the ground up. At the same time, the future Internet should shift control from the network to end hosts. A central question is how protocols can achieve security despite this shift in control. Our focus is on architectures that do so using cryptographic authenticators in each packet. We show the limitations of existing proposals for a secure future Internet architecture and set the stage for our two main contributions: First, we propose *EPIC*, a protocol suite aimed at providing secure forwarding at substantially lower overhead than existing protocols. Second, we formally verify security properties of EPIC and a number of other protocols. We do so in a symbolic model using an interactive theorem prover and following a foundational approach. Together, these contributions bring us closer to a formally verified, secure and efficient Internet.

1.1 The early Internet

Some of the first ideas of an Internet came from J.C.R. Licklider, who, in 1962, dreamed of a “Galactic Network”. Independently from Licklider, Leonard Kleinrock proposed the use of packet-switching over circuit-switching in networking. Lastly, Robert Taylor, working for ARPA (today known as DARPA) sought to connect his organization's high performance computers across the United States.

The intersection of the work of these three researchers [72], who provided the vision for the Internet, the technical groundwork to realize it, and the concrete motivation to deploy such a network, resulted in the creation of ARPANET in 1969, a precursor to the Internet [65]. Initially connecting four different research sites, ARPANET later connected 19 such nodes. Eventually networks of different kinds were connected in an “inter-networking” effort, giving rise to name “Internet”.

ARPANET showed the feasibility of many new concepts that proved to be important for the Internet, such as packet-switching, decentralization and the interoperability of networks of different architectures. The usage of protocol layers showed to be instrumental for the Internet, when Vinton Cerf and Robert Kahn proposed the Internet Protocol (IP) as well as the Transmission Control Protocol (TCP) [28] in 1974. The User Datagram Protocol (UDP) was added in 1980 as an alternative networking layer.

1.1.1 Security

While the designers of the early Internet were concerned with its security, a number of factors lead to the decision not to include cryptographic protection into its design.

Network operators often knew, and trusted each other and thought they could exclude troublemakers and adversaries from the Internet [99].

It was believed that adversaries would primarily be sophisticated, state-sponsored attackers. Profit-seeking criminals were not seen as a major threat [30]. The main concern of the US military and intelligence community with respect to these adversaries was the confidentiality of communication, instead of availability. To achieve confidentiality, the end-to-end encryption of critical applications was more suitable, rather than securing the network itself. The US intelligence community also preferred cryptographic protections to be limited in scope. The concern was that the uncontrolled and widespread use of cryptography could be detrimental to their goals – a position that continues to clash with the arguments by security researchers to this date.

Lastly, specialized hardware was required for cryptography, since implementations in software were too slow. Hence, securing individual applications also seemed more feasible than securing the network in a comprehensive way.

1.1.2 Growing pains

At the core of the Internet is a routing protocol that allows connected entities to exchange and propagate routing information and thus to discover paths throughout the Internet. By 1989 it became clear that the existing protocols were unable to cope with the massive increase in routes. When the Border Gateway Protocol (BGP) was designed in that year to deal with these growing pains of the Internet, it was intended as a quick fix that was simple and fast to deploy.

By that time, it was also clear that security on the Internet played a major role. The widespread effect of the previous year's Morris worm and other attacks were still in recent memory. But the rapid increase in new routes required a fast solution, and BGP was only ever supposed to be a temporary solution. The designers of BGP sketched out the design of the future routing protocol on a few napkins, giving BGP the nickname "three napkin protocol" [100], and decided against including cryptographic features to secure the discovery and exchange of routes.

Once BGP was rolled out, secure protocols had no chance of deployment. Companies that were increasingly commercializing the Internet and driving its growth had little interest in security, arguing that the additional complexity would hinder deployment, and that customers would be unwilling to pay extra for security [100].

1.2 Today's Internet

Since the deployment of BGP, the Internet has grown into a global network of ca. 70,000 independently managed networks [14], called *autonomous systems* (ASes), which are run by entities such as Internet service providers (ISPs), content providers, or public institutions. Despite being intended as a temporary solution, BGP is still the main routing protocol between autonomous systems. This protocol did surprisingly well given the Internet's rapid growth since its introduction. However, the availability, performance and security concerns associated with the BGP ecosystem are becoming increasingly clear. We outline them here.

1.2.1 Scalability and reliability limitations

Networking in today's Internet is plagued by numerous performance and security problems. Forwarding uses longest-prefix matching on large routing tables, which scales poorly and requires expensive hardware support. Networking using BGP relies on the convergence of the distributed state. Changes to the network topology trigger routing updates that can lead to outages lasting tens of minutes [60], and in some topologies, BGP does not converge to a stable state at all [51]. These concerns about BGP are not merely theoretical worst case analyses: BGP updates have been identified as the cause of up to half of all disruptions in experimental Voice-over-IP (VoIP) setups, and an even higher share of dropped calls [68].

1.2.2 Lack of security

The current Internet does not only have these efficiency and scalability limitations, but also lacks security at the networking level. For instance, due to the lack of authentication of packets' sources, adversaries can launch attacks with spoofed source addresses. While the widespread use of ingress filtering, as proposed in BCP 38 [45], could prevent such attacks, as long as such measures are voluntary, attackers can use networks without filtering to spoof their origin. Even if a measure against source spoofing was universally deployed, it would not solve the independent problem of BGP hijacking attacks, in which ASes make malicious BGP announcements, thereby illegitimately attracting traffic of IP prefix ranges. Without secure routing, all of the ca. 70,000 ASes in the Internet must be trusted not to carry out prefix hijacking attacks [10].

In response to these well-known and longstanding security problems, the networking community has been working to augment BGP with additional security mechanisms. Protocols such as BGPSec [74], S-BGP [62], soBGP [104], psBGP [101], PGBGP [58] and BGP origin validation [23] add security mechanism to the existing BGP infrastructure. Unfortunately, these additions are insufficient to solve the Internet's problems [31, 75, 91], or introduce new problems such as high overhead [49, 86] and kill switches [91]. In short, they trade off security with performance and they fail to address the reliability problems of BGP's convergence-based approach. As the networking layer already suffers from scalability and efficiency limitations, solutions that amend BGP at the cost of performance are unlikely to be deployed in the future.

1.2.3 Lack of path transparency and control

Another limitation of the current Internet is the lack of control that end hosts have on the networking layer. Concretely, a host that sends a packet has little or no influence on the packet's path. While hosts can sometimes select the first hop (through multi-homing) [34], the path beyond it is outside of their control. Each AS decides on its own where traffic towards a certain destination should be routed to. This decision is usually driven by commercial interests, such as avoiding higher-cost links, rather than path metrics that are relevant to the endpoints, such as latency or bandwidth. Hence, cost-minimizing practices such as hot potato routing are used by many ASes. Even if path metrics relevant to users were prioritized, selecting a single path means balancing conflicting properties such as low latency and high bandwidth. All packets from a given source would traverse the same path to a given destination instead of each application using the path that best fulfills its requirements.

The lack of path control also leads to many other problems, such as compliance, when data is not allowed to leave a particular jurisdiction; privacy leaks, when BGP hijacking attacks are used to de-anonymize users [98]; or re-routing attacks being used to obtain fake certificates [20].

Even more limiting, the current Internet not only rules out path control by end hosts, but it also lacks a way for end hosts to reliably verify the *actual* path a packet took on its way to the recipient. While applications such as `traceroute` enable network probing, the obtained information does not necessarily reflect the actual network topology, as the tracing packets can be treated differently by on-path routers. Furthermore, in an adversarial environment, the results cannot be trusted with, due to the lack of authentication in existing network probing techniques [3, 8].

1.3 A new Internet

Over the past 15 years, different architectures for a new Internet have been proposed, many of which give transparency and choices to end hosts [5, 19, 50, 83, 88, 89, 106, 108]. Like most modern networking protocols, they are composed of two parts: (i) the low-bandwidth *control plane*, in which neighboring nodes exchange topology and path information, and (ii) the high-bandwidth *data plane*, in which data packets containing user's payloads are forwarded across the network along the paths discovered in the control plane.

In *path-aware* networking architectures, forwarding paths, which are created in the control plane, are transparent to end hosts in the data plane. Many architectures allow end hosts to not only *see* which paths are used for forwarding, but also allow them to actively *select* paths. We focus on these architectures and use the term path-aware networking to mean protocols that shift at least some control from routers and network infrastructure to hosts. While path-awareness enables end hosts to choose paths that suit their applications' needs – such as low latency, or high bandwidth – full control, meaning that they can create *arbitrary* forwarding paths, is not desirable. If arbitrary paths could be constructed (*source routing*), then malicious end hosts could create paths that disrupt forwarding or are uneconomical for the on-path ASes.

The path-aware Internet architectures that we consider allow each Internet AS to restrict which paths end hosts are permitted to use. Typically, this is done via a path policy of each AS that determines if a particular path is permitted. The enforcement of these policies can be split between the control and data plane: the control plane authorizes paths consistent with each AS' policy, and the data plane ensures that packets are only forwarded along authorized paths. We call the latter property *path authorization*.

Path authorization can be realized either by storing allowed paths on each border router [50, 106], or by cryptographically securing the publicly distributed path information and verifying it during forwarding [5, 19, 83, 88–90]. Stateful solutions scale poorly to the inter-domain context, can suffer from inconsistencies across distributed nodes, and are less efficient than cryptographic solutions [64]. We thus focus on systems that use cryptographic authenticators for each (AS-level) hop in the header of packets. In these architectures, data packets themselves carry the path on which they are to be forwarded, with a pointer indicating the packet's position on the path (*packet-carried forwarding state*). Hence, border routers do not need to keep inter-AS routing tables or perform expensive lookup operations and are essentially stateless.

1.4 Existing path-aware networking architectures

In existing systems, fixing the length of per-hop authenticators poses a dilemma. Sufficiently long authenticators cause a high communication overhead as they are embedded in each packet. On the other hand, short and efficient authenticators are insecure: An attacker can conduct an online brute-force attack, i.e., send packets with fabricated authenticators between two hosts under his control until a packet is successfully forwarded. Once an authenticator is successfully brute-forced, the attacker can send an unlimited number of packets along the unauthorized path. So far, there is no solution that is both efficient and secure.

Parallel to the development of next-generation Internet architectures, recognition grew that additional security properties are desirable [22, 24, 25, 27, 64, 83, 105]. End hosts and routers need to authenticate the source of packets as well as other parts of packets' headers and their contents (*source and packet authentication*). Furthermore, the source and destination may need to be able to reconstruct and validate a packet's actual path (*path validation*).

A prime application of source authentication is defending against denial-of-service (DoS) attacks, in which network links or end hosts are flooded with excessive amounts of traffic. These attacks are often enabled by the attacker's ability to spoof her own address. Source authentication at network routers protects both the network and the destination by filtering unauthentic packets early and before they reach any bottleneck links. In addition, more sophisticated DoS-defense mechanisms such as bandwidth-reservation systems fundamentally depend on efficient source authentication mechanisms [13].

On the other hand, path validation protects the path choices made by the source of packets; if messages need to follow a specific path due to, e.g., compliance reasons, end hosts should be able to check whether their path directive is actually obeyed by on-path routers. Also, in a path-aware Internet, end hosts may be able to choose between several paths of different properties and costs. If using and paying for a more expensive path (e.g., through a satellite network), end hosts have a legitimate interest in obtaining proof that this path was actually traversed. Otherwise, malicious ASes could re-route the packet along a cheaper path and avoid the high-cost path.

While solutions exist that provide source authentication and path validation, they come with significant communication and computation overhead: ICING [83] and OPT [64, 88] have an overhead of hundreds of bytes per packet for realistic path lengths. A recent proposal, PPV [105], reduces the overhead and reaches practically feasible efficiency, but only verifies individual links on the path probabilistically and only enables source authentication for the destination. However, as described above, filtering out packets before they reach bottleneck links is important in defending against DoS attacks; hence, for this application it is preferable for *every packet* to be checked at *every hop*.

1.5 EPIC protocols

In this thesis, we propose EPIC, a family of cryptographic data plane protocols for path-aware Internet architectures. These protocols have increasingly strong security properties, including path authorization, source authentication, and path validation. EPIC is an acronym that stands for *Every Packet Is Checked*.

The key insight of our protocols is how they escape the dilemma between low communication overhead and security: On the one hand, we use relatively short per-hop authentication fields in EPIC to limit communication overhead. On the other hand, we ensure that an attacker with the ability to forge a few of these fields by sending a large number of packets cannot cause significant damage. We achieve this in two ways: First, by binding the authenticators used on a path to a specific packet and by preventing the same packet from being forwarded twice, EPIC ensures that a forged authenticator does not allow an attacker to send more than a single packet. It thus prevents volumetric DoS attacks based on unauthorized paths. Second, EPIC uses a longer authentication field for the destination which is unforgeable with high probability for even strong attackers, such that the very few packets that were able to deceive intermediate routers are detectable at the destination.

EPIC also uses the same field that proves the authorization of a path to an AS to provide source authentication. Moreover, after an AS has validated this field, it embeds a proof of its traversal in the same field, providing path validation to the source and destination. Using a single per-hop field for different properties decreases the communication overhead compared to existing approaches that use different per-hop fields for each property.

As a result of using short per-hop authenticators and utilizing the same authenticator field for different security mechanisms, EPIC has substantially lower communication overhead, and scales better with the path length than state-of-the-art protocols like ICING [83] and OPT [64].

1.6 Formal verification

The complexity of data plane protocols such as EPIC and their central role in a new Internet architecture calls for their formal verification. This will yield strong guarantees of their correctness and security. There are several payoffs from this effort. First, it enables the early detection of protocol flaws and vulnerabilities, avoiding critical exploits and high costs for corrections after deployment has begun. This is especially true for the data plane since it will be implemented in high-performance software or hardware routers, which are difficult to update after their deployment. Second, a formal proof increases confidence in the architecture's security, thereby fostering its adoption. Third, in a given AS, the number of border routers can be orders of magnitude higher than the number of control plane servers, making changes that require their replacement very costly.

We focus our verification effort on path authorization. Data plane protocols exhibit characteristics that make the verification of this property particularly challenging. First, we want to verify this security property over arbitrary network topologies and authorized paths therein, as determined by the control plane. Second, the formalism must be expressive enough to describe (i) path authorization, which is a non-local property that involves all ASes on a path, and (ii) assumptions on a control plane adversary's capabilities to shorten, modify, and extend certain authorized paths. Third, the number of participants and the message sizes in a protocol run depend on the (unbounded) length of the path embedded in a given packet. We anticipate that state-of-the-art automated security protocol verifiers such as Tamarin [82] and ProVerif [21] could only be used for bounded verification of path authorization, instead of verification under arbitrary sets of authorized paths.

For this reason, we employ a higher-order logic theorem prover, Isabelle/HOL [85], which allows us to model and verify data plane protocols in their full generality. As research in novel path-aware Internet architectures has led to several interesting candidate (families of) data plane protocols, we would like to verify these without the need to restart the specification and verification effort from scratch for each protocol or variant. Specifications and proofs should thus share as much structure as possible. To achieve this, we propose a parametrized framework in Isabelle/HOL for the verification of data plane protocols. Figure 1.1 gives an overview of our framework.

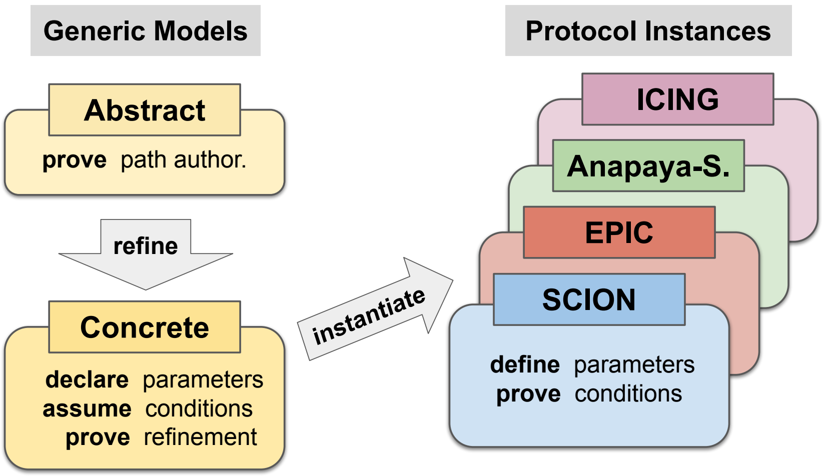

Figure 1.1: Overview of our verification framework and of protocol instance models. Refinement and instantiation preserve properties.

We first develop a simple abstract model of a packet forwarding protocol without an attacker, for which we formulate and easily prove path authorization. We then refine this model into a more concrete one, where we introduce a Dolev–Yao adversary and (generic) cryptographic authenticators, called *hop validation fields* (*HVF*), that protect each AS-level hop along a forwarding path. A key insight is that the main difference between path authorization mechanisms is how the *HVF* is computed. This allows us to define a single skeleton protocol model, which we can instantiate to a wide range of actual protocols. We achieve this by parametrizing the concrete model by (i) a cryptographic hop validation field check that must be performed by each AS locally to determine the authorization of the forwarding path, (ii), a predicate that allows one to incorporate additional assumptions on the set of authorized paths, (iii) a function that extracts an entire forwarding path from a hop validation field, and (iv) a set of (cryptographic) terms added to the attacker's knowledge.

We identify five simple *verification conditions* on these parameters that suffice to prove that the concrete model refines the abstract one and therefore inherits the path authorization property. These conditions require the *HVF* to be unforgeable and to contain the forwarding path.

Our development is also parametrized by an arbitrary network topology and a set of authorized paths constructed by the control plane. Our security proofs hold for all network topologies and control planes that satisfy some realistic assumptions.

To define a concrete data plane protocol in our framework, we instantiate the parameters (i)-(iv) and discharge the five verification conditions. We do so for the data plane of SCION [88], several protocols belonging to the EPIC family, Anapaya-SCION [4], and ICING [84], as well as for variants of these protocols. The instantiations and the associated proofs of the verification conditions are substantially shorter, simpler, and more manageable than redoing a full specification and security proof for each protocol. In particular, discharging the conditions does not involve reasoning about state transitions (unlike, e.g., proving an invariant).

We extend our framework in various ways to allow for (i) a wider range of protocols to be modeled, (ii) more accurate modeling of protocols and (iii) different attacker models. For instance, we add additional fields as parameters, extend our term algebra with an abstraction of exclusive-or (XOR), support certain header updates by on-path routers, and introduce additional mechanisms that allow instances to define an oracle-based strong attacker model.

We classify path authorization mechanisms depending on how paths are authorized in the control plane: in *undirected* protocols, each on-path AS authorizes the path in its entirety. In *directed* protocols, each such AS only authorizes the path in one direction. Our verification framework supports both classes of protocols. All protocols we have studied, except for ICING, are directed.

Since path-aware Internet architectures have not yet been widely deployed, they are not well-known outside of the networking community. Hence, there is little existing verification work. The most closely related works are the verification of a weaker AS-local form of path authorization [29] and of different security properties [107] for such architectures. Both of these works mechanize their proofs in Coq using a non-foundational approach, i.e., relying on an axiomatization or external tools. We further discuss related work in §4.9.

1.7 Contributions

The contributions of this thesis can be split up into the EPIC work, which proposes concrete new protocols, and the verification work, which formally verifies security properties for a whole class of protocols.

1.7.1 EPIC protocols

In the work *EPIC: Every Packet Is Checked in the Data Plane of a Path-Aware Internet* presented in the 29th USENIX Security Symposium by Markus Legner, Tobias Klenze, Marc Wyss, Christoph Sprenger, and Adrian Perrig [71], our main contributions are:

- We propose EPIC, a series of protocols that use unique authenticators for each packet to resolve the security–efficiency dilemma in the data plane of path-aware Internet architectures.
- We propose a new attacker model that combines a Dolev–Yao [37] adversary with a cryptographic oracle. This allows us to express EPIC's resilience against even powerful attackers. EPIC achieves all desirable security goals in this stronger attacker model.
- We show that EPIC has a communication overhead that is 3–5 times smaller compared to the state-of-the-art solutions OPT and ICING for realistic path lengths.
- Using Intel's Data Plane Development Kit (DPDK) [38], we implement EPIC and show that our router implementation running on commodity hardware can saturate a 40 Gbps link while using only four processing cores.

Markus Legner is the main author of the EPIC protocols. The protocols were implemented by Marc Wyss and the performance analysis was performed by Markus Legner and Marc Wyss. My contributions were primarily in the security analysis and in proposing a novel attacker model.

1.7.2 Formal verification

In the work *Formal Verification of Secure Forwarding Protocols* in the 2021 IEEE 34rd Computer Security Foundations Symposium by Tobias Klenze, Christoph Sprenger, and David Basin [66], our main contributions are:

- We develop a generic framework for verifying security properties for a general class of data plane protocols for arbitrary network topologies. This framework has four protocol parameters that are required to satisfy five simple verification conditions.
- The five conditions provide insight into the common structure underlying data plane protocols of path-aware Internet architectures.
- We instantiate our framework with nine different variants of realistic data plane protocols proposed in the literature and prove that they satisfy path authorization by establishing the parametrized model's verification conditions.
- All of our definitions and results are formalized in Isabelle/HOL following a foundational approach, which only relies on the axioms of higher-order logic and thus provides strong soundness guarantees.

This thesis presents three additional features of the framework and adds a protocol instance. These contributions are not part of the work cited above.

- We extend the verification framework by (i) mutable per-packet fields that are updated by each on-path node, (ii) a parameter for additional protocol-specific control plane assumptions, and (iii) a novel abstraction of XOR for authentication protocols.
- Using these extensions, we instantiate our framework with a proposed successor to the authentication mechanism used in SCION [4], which we call Anapaya-SCION.

1.8 Overview of the thesis

Chapter 2 gives background on path-aware Internet architectures, and informally describes desirable data plane security properties. In Chapter 3, we propose the EPIC protocol family, analyze its security informally, and compare its performance performance to other protocols. We abstract from EPIC and propose a parametrized framework for the verification of a wide range of different data plane protocols in Chapter 4. We discuss related work at the end of Chapters 3 and 4. We conclude in Chapter 5.

All results in the formal verification chapter of this thesis are formalized in Isabelle/HOL and are available in the supplementary material.

2 Preliminaries

Path-aware Internet architectures are complex systems composed of multiple components and protocols. We present the data plane requirements and in particular, the security goals of individual actors and desirable properties, in §2.1. We then describe path-aware future Internet architectures in more detail, in particular the control plane mechanisms that are outside the scope of our work, but that are relevant to it because they interact with the data plane. This background is given in §2.2.

2.1 Problem definition

We target the problem of securing the inter-domain data plane of path-aware Internet architectures.

Each autonomous system (AS) has centralized control within its own network, which simplifies managing and securing the intra-AS communication, e.g., through software-defined networking [18]. By contrast, networking between ASes requires coordination between separate entities without central control. Our work focuses on securing this inter-AS communication. We exclude the equally important, but orthogonal problem of securing intra-AS networking. Thus, in line with previous work [81], we abstract from the internal networks of ASes and consider all security properties at the level of ASes (or the end hosts that connect to them); in particular, throughout the remainder of this thesis, "hop" stands for "AS-level hop".

We also only focus on securing the data plane. We assume that the control plane is secure and constructs paths according to the ASes' policies and the participants of our data plane protocols obtain the required symmetric keys and path information via secure control-plane channels. While securing key distribution and other control-plane functionality is itself a challenging task, it is orthogonal to the challenges for the data plane: as we argue below, in the control plane, asymmetric cryptography can be used to provide strong security guarantees, whereas in the data plane only symmetric cryptography

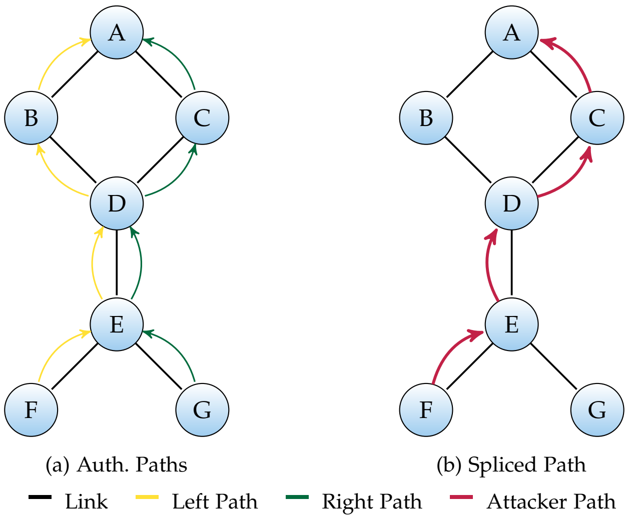

FIGURE 2.1: If *path authorization* holds, a malicious sender at node F cannot splice the two authorized paths in (a) to create the unauthorized forwarding path in (b).

is sufficiently efficient. In practice, future Internet architectures propose the use of public-key infrastructures to secure control-plane operations, such as the Resource Public Key Infrastructure (RPKI) of today's Internet [73].

EPIC, and other protocols protect the interests of both end hosts and honest ASes. We present the security goals and properties of both types of agents below. But first, we give some more detail on our network model.

2.1.1 Network properties

We model ASes as *nodes* in a graph. Figure 2.1 gives an example of a (tiny) Internet topology to which we will return throughout this thesis. The internal structure of an AS is shown in Figure 2.2. ASes are interconnected at *border routers*, which sit at the edge of the internal network of each AS. When a packet is received by a border router via an inter-AS link, there are two cases: If the packet's destination is inside the AS, the border router forwards it through the internal network to the end host. Otherwise, the border router determines the next inter-AS link and forwards it through the internal network to the border router adjacent to that link (cf. Figure 2.2).

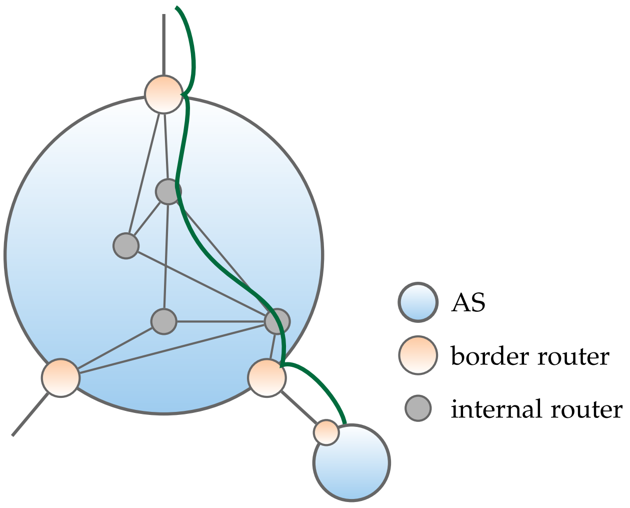

FIGURE 2.2: Internals of AS E of Figure 2.1 with one path shown. The internal path between border routers is decided by the AS.

As mentioned, the path-aware Internet architectures that we examine provide end hosts with *path control*, which is the ability to choose from a set of authorized forwarding paths for each destination. End hosts select their desired forwarding path at the granularity of inter-AS links, and embed the path alongside authenticators in each data packet. This *packet-carried forwarding state* removes the need for border routers to keep state for inter-AS forwarding. Path control empowers end hosts to make path choices that are suitable for their applications' needs. For instance, Voice-over-IP (VoIP) requires little bandwidth but low latency, whereas data synchronization requires high bandwidth, but latency is less important. These applications can thus use different paths. Moreover, *multipath routing* allows multiple paths between the same source-destination pair to be used simultaneously, even by the same application.

2.1.2 Security requirements for end hosts

We consider two fundamental security properties for end hosts: *path validation* and *packet authentication*.

Source and packet authentication

Packet authentication provides proof of a packet's origin and content to the destination, preventing source spoofing by the sender or packet modification by routers that are possible in today's Internet. Since packet authentication includes the authentication of the source address, we sometimes also use the term *source authentication*.

Path validation

While path control allows sources to select a forwarding path, it is by itself insufficient to protect the security interests of end hosts as it does not provide any guarantees that the sender's directives are actually obeyed. *Path validation* enables the destination of a packet to verify that the actually traversed path of the packet matches the path intended by the sender. The destination can send this proof back to the sender, allowing the source to also verify that its intended path was indeed traversed.

Key establishment and trust assumptions

Packet authentication can only be realized if there are keys shared between the source and the destination. Furthermore, path validation requires additional keys between end hosts and on-path ASes.

Both properties can be realized by using public key cryptography, or distributing symmetric keys between each pair of end hosts and each pair of end host and AS. Using public key cryptography would either involve a handshake to establish symmetric keys prior communication, an approach that typically is only employed at a higher level in the protocol stack (such as in TLS), or it would involve asymmetric cryptographic operations *per packet*, which incurs a prohibitive computational overhead on end hosts and routers (cf. §2.1.5). Alternatively, symmetric keys could be distributed prior to communication, but setting these up between any pair of end hosts would have a prohibitive overhead. These naïve solutions are unsuitable to realize authentication and path validation properties on the network layer.

Dynamic key derivation techniques such as PISKES [92], DRKey (proposed in OPT [64]) or non-interactive Diffie–Hellman (used in ICING [83]) are suitable alternatives. They scale significantly better, albeit at the cost of additional trust assumptions or communication overhead.

DRKey / PISKES, the mechanism used by EPIC protocols, requires source and destination hosts to trust both of their ASes due to the key-distribution mechanisms. Due to this additional trust assumption, network-level authentication does not replace the security offered by higher-layer protocols such as TLS. At the same time, higher-level authentication is not a replacement for network-layer authentication: network-layer schemes can be used for packet filtering that sets in *prior* to stateful TCP and TLS handshakes, and is thus highly efficient.

We postpone the discussion of the concrete trust assumptions of EPIC to §3.2, and only note here that similar assumptions are also part of in other network-level authentication schemes that provide similar properties [64, 88].

2.1.3 Security requirements for ASes

For ASes, we consider four security properties: *path authorization*, *detectability*, *source authentication* and *path validation*.

Path authorization

The end host's power of choosing paths is balanced against the interest of network providers, who express their routing policies in the control plane by selectively authorizing paths. We describe these control plane processes in §2.1.4, but crucially, each AS is driven by its own economic interests, which gives rise to path policies that collectively define a set of authorized paths.

Path authorization is the data plane property that all data packets traverse the Internet only along authorized paths. Path authorization protects ASes from malicious senders who could try to forge paths that are advantageous to themselves (e. g., by using costly paths that they have not paid for and violate the ASes' path policies), detrimental to ASes (e. g., uneconomical *valley* paths [46]), or disrupt forwarding entirely (e. g., through loops).

Note that path authorization is not just a local property; in particular, it is not enough for each hop to authorize its local routing information (*prev* and *next* interfaces), since that would still allow for attacks such as forwarding loops. If a strictly hierarchical structure with defined provider–customer and peering relationships is assumed, forwarding loops can be prevented with local checks [47]. However, the same is not true of path policies in general as our example in Figure 2.1a illustrates. In this example, two paths leading to the destination node A are authorized: the left path F–E–D–B–A, and the right path G–E–D–C–A. Node F is only authorized to use the left path and is forbidden to send packets to A via C. Path authorization implies that an attacker at F cannot craft a packet that traverses the path given in Figure 2.1b. Each AS only checking the authorization of its local forwarding information cannot guarantee this.

We distinguish between two types of path authorization: In *undirected* protocols, an AS on a path authorizes the full path. In *directed* protocols, an AS only authorizes the partial path consisting of its own hop and all *subsequent* hops in forwarding direction. Hops that are added *before* the AS do not need to be authorized by it. For instance, in Figure 2.1a, AS D could decide which of the partial paths D-B-A and D-C-A to allow, but once authorization is granted, extensions authorized by E and E's children are also implicitly authorized by D. While in our formalization both types of protocols achieve the same path authorization property, they do so under different control plane assumptions.

Stateful routing protocols such as BGP allow ASes to inspect the (partial) route before determining whether to add it to their own, local routing table. In the data plane, each router simply follows the forwarding directive in its table. Since forwarding is a local process on a router and not a security protocol, a data plane attacker cannot influence it. Hence, the current Internet technically achieves path authorization. However, the lack of sufficient control plane security in the BGP ecosystem renders this guarantee ineffective, since overall security is achieved only when both the control and the data plane are secure. Attacks are possible in the control plane, and a malicious upstream AS could trick an AS into adding entries to their routing table that will result in forwarding paths that violate their routing policy. Only the local routing information can be protected with certainty.

Detectability

Since the entire forwarding path is contained in each packet, a malicious source cannot hide her presence on the path. *Detectability* states that the actual path that a packet traverses is contained in (i.e., a prefix or suffix of) the path embedded in the packet's header.

This property is sometimes called *weak detectability*, since it does not prevent source spoofing. Rather, it ensures that *if* a source is spoofed, then the attacker must be one of the nodes on the packet's forwarding path, thus ensuring basic accountability for data packets.

Source authentication

Source authentication is a stronger property than detectability. It ensures that routers can validate the origin of each packet, thus ruling out source spoofing attacks. This is an important property, since in many DoS attacks on the current Internet, the attacker spoofs the origin of attack traffic. While some protocols (e.g., IPSec) enable source authentication, they typically only filter traffic at the destination. Dropping malicious traffic early is not only more efficient than destination filtering, it also protects against DoS attacks that target the networking infrastructure itself [57, 97], rather than an end host: source authentication by routers ensures that traffic is filtered before any bottleneck links are reached. Furthermore, sophisticated DoS-defense schemes such as bandwidth-reservation systems [13] rely on source authentication to prevent adversaries from using up reserved bandwidth of honest sources.

Path validation

Path validation is a property that has been proposed not only for the sending and receiving end hosts (cf. §2.1.2), but also for on-path ASes. This allows them to check that the path chosen by the sender was indeed successfully traversed up to their own AS on the path.

Path validation for on-path ASes prevents path-switching attacks, in which an intermediate router replaces the source-selected path by a different path. For instance, if the source has paid for an expensive low-latency path, an intermediate on-path AS could swap out that path by one that is cheaper. However, this attack can also be prevented in different ways, for instance by letting each on-path AS authenticate the packet's header, including its path.

Another use case for path validation is that an AS may offer certain services, such as virus-scanning or intrusion-detection on each packet that traverses its network. Path validation allows checking that such an AS was indeed traversed. However, it does not guarantee that the offered security check was indeed executed on the packet.

While path validation for routers is a property that has been provided by previous work [64, 83], it remains unclear if there are practically important use cases. It seems that path validation by the destination or source and packet authentication by either the destination or on-path ASes is sufficient for the proposed use cases and to defend against the most important known attacks. These properties are much easier to achieve than path validation for routers.

Trust assumptions

Similarly to the setting of security for end hosts, security for ASes requires special trust assumptions. We discuss these in §3.2.

2.1.4 Other security properties

We outline some desirable properties the data plane cannot guarantee and that are hence out of scope.

Control plane security

The control plane is responsible for authentically and efficiently discovering and distributing paths (see §2.2.1) and ensuring that they do not contain loops and fulfill the policies of ASes. However, a secure control plane cannot substitute a secure data plane: the data plane needs to provide path authorization, i.e., enforce the decisions that ASes make in the control plane for data traffic.

Geofencing

A desirable security property is that a specific AS or set of ASes is *avoided*, i. e., not traversed. Laws or compliance rules might impose rules on the geographical area which packets are restricted to (*geofencing*). For instance, communication between a bank and the central bank in the same country should be forwarded only domestically, and not leave the country.

Unfortunately, neither path validation not any other property can guarantee this in the presence of an on-path attacker. Such an attacker can always forward a packet honestly while also sending a copy of the packet to an off-path AS.

2.1.5 Efficiency requirements

The need to keep ever-growing forwarding tables on routers of the current Internet requires expensive hardware and fundamentally limits its scalability. It is therefore essential that a future Internet minimizes router state.

The data plane must also have low communication and computation overhead and minimize additional latency during setup and processing. A simple calculation underscores this: Consider 400 Gbps links, which are currently being deployed in the Internet, and 500 B packets. To saturate the link, a router needs to process one packet every 10 ns. Even taking into account pipelining and parallelism, packet processing in the data plane must proceed within hundreds of nanoseconds—ruling out any asymmetric cryptography, which requires several microseconds for a single operation [40]. In contrast, block ciphers with hardware acceleration such as AES can be computed within tens of nanoseconds and are suitable to use in the data plane [26, 52]. The validation of the MAC is substantially faster, and scales better, than looking up authorized paths in a table on each router [88].

2.1.6 Formal verification requirement

As outlined in the introduction, the data plane plays a critical role in the Internet ecosystem. Testing and ad-hoc inspection of a protocol cannot reliably be used to systematically uncover all bugs, or to prove its correctness and security. A formal analysis can achieve this goal under reasonable assumptions. Hence, formal verification is also a requirement in future Internet architectures, and data plane protocols specifically.

2.2 Background and definitions

To provide the necessary context for constructing and verifying data plane protocols, we sketch out an abstract path-aware control plane, in particular the path-exploration and -registration mechanisms. This description is loosely based on SCION's control plane [88] but abstracts from many low-level details. We postpone the discussion of how EPIC can be integrated into real architectures to §3.5.2.

The background on the formal methods and the additional notation that we use in our verification framework is introduced in §4.2 in the verification chapter.

The table on page page xiii summarizes the notation and acronyms used throughout the thesis.

2.2.1 Path exploration and registration

To discover paths between any pair of ASes, each AS periodically initiates path exploration by sending *beacons* to their neighboring ASes. We illustrate

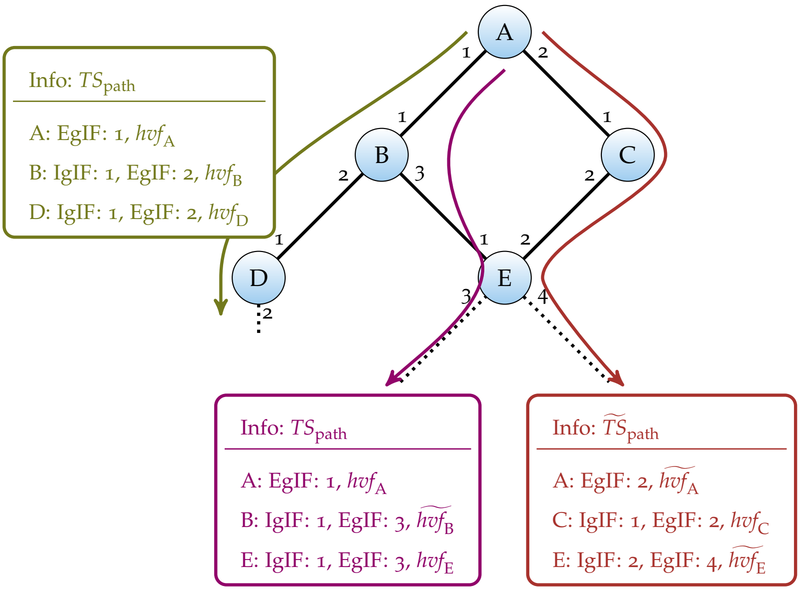
A
B
C
D
E
1
2
1
1
2
3
2
1
1
2
2
3
4
Info: TS_{path}
A: EgIF: 1, hvf_A
B: IgIF: 1, EgIF: 2, hvf_B
D: IgIF: 1, EgIF: 2, hvf_D
Info: TS_{path}
A: EgIF: 1, hvf_A
B: IgIF: 1, EgIF: 3, $\widetilde{hvf}_B$
E: IgIF: 1, EgIF: 3, hvf_E
Info: $\widetilde{TS}_{\text{path}}$
A: EgIF: 2, $\widetilde{hvf}_A$
C: IgIF: 1, EgIF: 2, hvf_C
E: IgIF: 2, EgIF: 4, $\widetilde{hvf}_E$

FIGURE 2.3: Paths are explored by periodic beacons. Here, we show beacons originating at AS A that are subsequently extended and forwarded by other ASes. The numbers in the figure indicate AS-specific interface identifiers; the tilde symbol above a variable name distinguishes different values of these fields. Here, E chooses to disseminate the beacons received from B and C selectively.

this process in Figure 2.3. An AS can decide which paths to authorize by selectively forwarding the authenticated beacons to neighbors and registering them at public *path servers*.

Paths in the Internet can be viewed at different granularities, e.g., at the level of ASes or individual routers. We consider paths at an AS-level granularity, but include the ingress and egress *interface IDs* of each (AS-level) hop that connect the AS to its neighbors. Each AS is free to assign these identifiers to its external connections without coordination with other ASes. Hence, they do not need to expose any information about their internal network structure. The interface IDs along a path are recorded in the control plane and later used for packet forwarding in the data plane.

The beacons that are used to discover paths are each initialized with an absolute timestamp TS_{path}. An AS that spawns a new beacon or receives a beacon and decides to disseminate it adds its own *hop entry* before registering the beacon with a path server or sending it to a neighboring AS. This hop entry consists of three parts: a *hop information* field *HI* that is used in the data plane as a forwarding directive, a *hop validation field HVF*, which protects the path (including the hop's own *HI* field) in the data plane, and a signature over the entire beacon used in the control plane to authenticate the beacon.

The hop information field *HI* consists of an expiration time ts_{exp} relative to the beacon's timestamp; the ingress interface, through which the beacon was received; and the egress interface, through which it is forwarded. The first and last hop information fields on a path only contain a single interface, as there is no predecessor and successor AS, respectively.

In EPIC and in other path-aware networking protocols, the hop validation field is a cryptographic value such as a *message authentication code* (MAC), a cryptographic hash, or a tuple of such values. It allows the routers to verify the authorization of the path during forwarding. The hop information *HI* of an AS together with the authenticator *HVF* that protects it is called *hop field*.

Lastly, the signature that each AS adds to the beacon protects the beacon's authenticity in the control plane. Since the hop authenticator, not the signature, is used for authenticating the path during forwarding, this signature is removed when beacons are turned into data plane paths by end hosts. Thus, each hop entry is turned into a hop field.

The forwarding information consisting of a sequence of such hop fields (the *path*) is fixed by the sending end host and remains static. A moving path pointer indicates the current hop field. In the data plane, the border routers of the AS check this MAC upon receiving a data packet. If it is valid, then the path was authorized in the control plane, and the border router forwards the packet. Before the packet is sent out to the next AS, the path pointer is incremented.

ASes can make decisions during path exploration about which paths to authorize based on their own economic interests. To that end, ASes can inspect the complete upstream path and only forward beacons that do not contain loops and are consistent with their path policies to their customers.

Hop validation fields

We present different ways in which a *HVF* can serve to protect forwarding information and thus provide path authorization in the following chapter. Here, we show a simple mechanism that is similar to how SCION protects path information.

In SCION, each AS A has a key K_A that is shared among its border routers and the control plane servers. As a *HVF*, SCION uses a *hop authenticator* σ. It is defined as a MAC over the *HI* field of A, which we call hi_A, and the MAC of the next hop B:

$$\sigma_A = \mathsf{MAC}_{K_A}(\langle hi_A, \sigma_B \rangle)\,. \tag{2.1}$$

Crucially, the MAC is created not only over the local routing information, but also over the next MAC. This chains the MACs and protects the entire subsequent path. During forwarding, each border router checks the validity of its own MAC embedded in the packet header.

2.2.2 Path construction and forwarding

To simplify, we assume that packets are always forwarded in opposite direction of beaconing. Protocols such as SCION allow paths to be reversed in order to allow for forwarding in either direction, but this is not an essential feature, as beaconing can occur bidirectionally as well.

In the data plane, we name the interfaces used for forwarding *prev*, for the interface over which the packet is received, and *next*, for the interface over which the packet is to be sent out. Since we consider forwarding to only occurs in *opposite* direction to beaconing, *prev* is the egress interface on the beacon, and *next* is the ingress interface of the beacon.

To create a forwarding path, an end host H_S queries its local path server (located in the same AS) for beacons from the intended destination AS A_ℓ to his own AS A_1. H_S selects a beacon from those offered by the path server, and verifies its signatures. By removing the signatures from the beacon, the beacon is turned into a path that can be directly embedded into the packet. A data plane packet thus contains the entire forwarding state. For a path from A_1 to A_ℓ it has the format

$$\begin{aligned} \textsc{Packet} &= \langle \textsc{Path},\ \textsc{ValHd},\ P \rangle, && (2.2a) \\ \textsc{Path} &= \langle TS_{\text{path}},\ \textsc{Src},\ \textsc{Dest},\ HI_1,\ \cdots,\ HI_\ell \rangle, && (2.2b) \\ \textsc{ValHd} &= \langle ts_{\text{pkt}},\ HVF_1,\ \cdots,\ HVF_\ell,\ V_{\text{SD}} \rangle, && (2.2c) \end{aligned}$$

where P denotes the packet's payload, $\textsc{Src} = \langle A_1, H_S \rangle$ denotes the source, and $\textsc{Dest} = \langle A_\ell, H_D \rangle$ denotes the destination. $\textsc{ValHd}$ contains fields necessary for verifying the packet. The timestamp ts_{pkt} indicates the time at which the packet is sent relative to TS_{path} and is used to provide freshness. The *destination validation field* V_{SD} allows the destination to authenticate the packet and validate its path. Per AS-level hop there is a hop validation field hvf_i which is used to achieve various security properties.

The protocols EPIC Level 1–3 define $hvf_i = \langle V_i, S_i \rangle$, i. e., as a tuple of two values. In these protocols, the *segment identifier* S_i is a cryptographic code based on σ_i used for path authorization. It can be created from σ_i and uniquely identifies the portion of the path in between the beacon initiator A_ℓ and A_i. The *validator* V_i is a cryptographic tag that is filled in by the source host and allows the intermediate router to validate the packet. Hence, the hop authenticator σ is not embedded directly in the packet, but it is used to compute both V_i and S_i.

We describe the EPIC protocols in terms of S_i and V_i (in §3.1). Since the other protocols that we formally verify do not have these two fields, we use a generic field HVF in our formal models.

We define the *packet origin* as the triple of source, path timestamp, and packet timestamp,

$$PO = (\textsc{Src}, TS_{\text{path}}, ts_{\text{pkt}}). \qquad (2.3)$$

The length of the timestamp field ts_{pkt}, i.e., the precision of the timestamp in EPIC Level 1–3 is chosen such that it is different for each packet sent by a source. Hence, the packet origin of each packet is unique in these protocols.

As the forwarding information for each AS-level hop is included in the packet header, intermediate routers at the border of an AS can follow the forwarding directive embedded in the packet (after cryptographically validating it). Hence, they do not need to keep forwarding tables for inter-AS forwarding. In case of a link failure, an end host can be notified and immediately switch to a backup path without needing to wait for the network to reconverge.

2.2.3 Notation

This section presents the notation used in the next chapter, in which we propose and analyze the EPIC protocols. The notation used in the chapter presenting the formal models is different in some aspects, since these models abstract from EPIC. We introduce that notation in §4.2.1.

Terminology

We use the term *path* in different ways. Depending on the context, it may be a path in a graph, annotated with interface identifiers at each hop, or it may be a sequence of hop fields (a data plane path), or a sequence of hop entries (a control plane path).

In SCION, paths can be reversed and concatenated. To distinguish the paths produced by the control plane from the paths that result from their combination, the former are called *segments* in SCION. In this thesis, we largely avoid this distinction, since neither the EPIC protocols nor our formal framework incorporate segment combinations. Hence, *segments* are called *paths* in this work.

Cryptographic notation

We explain some of the notation introduced in the table on page page xiii. We denote the application of a pseudorandom function (PRF) and the computation of a message authentication code (MAC) with key K as $\mathsf{PRF}_K(\cdot)$ and $\mathsf{MAC}_K(\cdot)$, respectively. We write l_{val} and l_{seg} for the lengths in bytes of the validators and the segment identifiers, respectively. The block size of PRFs and MACs in bytes is denoted by l_{PRF}, where $l_{\mathrm{PRF}} = 16$ for AES. In some protocols, validators are updated by intermediate routers; in this case, we write $V_{i;j}$ for the validator corresponding to A_i after processing by A_j and use $V_{i;0}$ for their initial values. We use superscripts to distinguish the different EPIC protocols, named L0–L3, e. g., $V_i^{(0)}, \dots V_i^{(3)}$. Concatenation of (binary) strings is denoted by , , and $X[\![i{:}j]\!]$ is the substring from byte i (inclusive) to byte j (exclusive) of X.

2.2.4 Global symmetric-key distribution

Some of the protocols that we propose require the source host to create for each packet authenticators, which either the destination or intermediate routers verify. While asymmetric cryptography scales well in the number of networking entities, the computation overhead of a per-packet usage is prohibitive as shown in §2.1.2. On the other hand, the standard use of symmetric cryptography would require routers to store a key for each packet source, which is infeasible on core routers in the Internet. In order to be able to use symmetric cryptography without per-host state on intermediate routers, we leverage the dynamically-recreatable-key (DRKey) / PISKES system [64, 92], which we will summarize in this section.

With DRKey, one party, e. g., a router in an AS A, can derive symmetric keys by simply applying PRFs to deterministic inputs, while the other party has to fetch keys from a key server (over a secure control-plane channel). DRKey defines AS-level keys shared between ASes A and B:

$$K_{A\to B} = \mathsf{PRF}_{K_A}(B)\,. \tag{2.4}$$

Here, K_A is a secret key of the AS A, which is shared between all its (border) routers and key servers but with no external entities, and B is a unique and public identifier of AS B. The arrow in the derived key indicates the asymmetry between A and B: AS A is able to quickly derive the keys on the fly using symmetric cryptography, while AS B needs to fetch the key $K_{A\to B}$ by an explicit request to A's key server, protected by asymmetric cryptography. DRKeys are valid for time periods on the order of one day, such that these key requests happen relatively infrequently.

Given an AS-level key, host-level keys can be derived by another application of a PRF:

$$K_{A\to B:H_B} = \mathsf{PRF}_{K_{A\to B}}(H_B) \tag{2.5a}$$
$$K_{A:H_A\to B:H_B} = \mathsf{PRF}_{K_{A\to B}}(H_A,\ H_B)\,. \tag{2.5b}$$

An end host H_B in AS B can query the key servers of B in order to obtain the keys (2.5a) or (2.5b), which can be calculated by AS B from the AS-level key (2.4). These keys are shared between all entities in the subscripts, e. g., $K_{A_\ell:H_\mathrm{D}\to A_1:H_\mathrm{S}}$ is shared among A_ℓ, H_D, A_1, and H_S. Therefore, when authenticating sources using DRKey, no end-host-to-end-host guarantees are obtained: A malicious AS A_1 could claim that a packet originating from H_S came from a different host H_S' in A_1. The destination host H_D in AS A_ℓ can only authenticate the source host under the assumption that A_1 and

A_ℓ are honest. As discussed above, these are common restrictions in order to accommodate the efficiency requirements of high-speed routers. As we discuss in §3.3.5, using DRKeys introduces little communication overhead and negligible additional latency.

Other AS-level key-establishment systems could be used for exchanging AS-level symmetric keys. For example, Passport establishes symmetric keys $K_{A \leftrightarrow B}$ between any pair of ASes by means of a Diffie–Hellman key exchange on top of BGP announcements [76]. These keys can be used in place of $K_{A \rightarrow B}$ in Equation (2.5) but require also to input the AS identifier in order to distinguish $K_{A:H \leftrightarrow B}$ from $K_{A \leftrightarrow B:H}$. Furthermore, as they cannot be recreated on the fly at border routers, a router would need to cache a symmetric key to every other AS.

Irrespective of the system used to exchange AS-level keys, the communication between end hosts and key servers relies on secure control-plane channels in order to prevent malicious entities from impersonating key servers or discovering keys. As we explained above, this is an orthogonal problem to securing the data plane, and thus outside the scope of this work.

3

EPIC: Every Packet is Checked

We now present our EPIC protocol family. This chapter is structured as follows: §3.1 describes the EPIC Level 0-3 protocols. We analyze their security properties in §3.2 and their performance and overhead in §3.3. §3.4 presents an extension of EPIC Level 3 that achieves path validation for routers. We discuss our results in §3.5, and related work in §3.6.

3.1 EPIC protocols

In this section, we develop three protocol levels 1–3 of EPIC with increasingly strong security properties. We present the protocols in a step-by-step development, thus explaining for each security property the mechanism and prerequisites to achieve it. All protocols use the packet format given in §2.2.2. As a starting point, we begin by describing a simple protocol (referred to as "EPIC Level 0") that represents the approach taken in the SCION data plane (with minor simplifications) [88]. Its primary security property is path authorization.

3.1.1 Level 0: Path authorization

EPIC Level 0 achieves path authorization using static MACs that are calculated during path exploration and directly serve as validators for forwarding. During the path-exploration process, an AS A_i calculates the hop authenticator $\sigma_i^{(0)}$ as a MAC over the beacon's timestamp, the hop information, and the previous (in beaconing direction) hop authenticator ($\sigma_j^{(0)}$), truncated to l_{val} bytes:

$$\sigma_i^{(0)} = \mathsf{MAC}_{K_i}\left(TS_{\text{path}},\ hi_i,\ \sigma_j^{(0)}\right) [\![0{:}l_{\text{val}}]\!]. \tag{3.1}$$

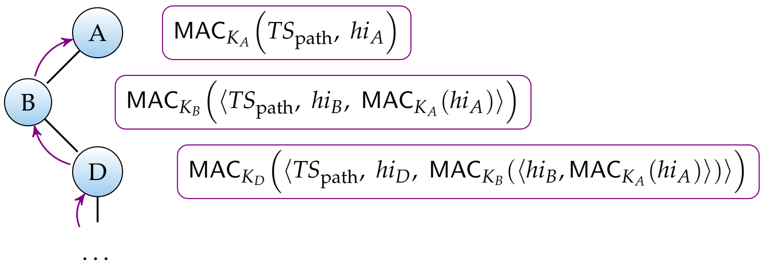
A
B
D
$\mathsf{MAC}_{K_A}\left(TS_{\text{path}}, hi_A\right)$
$\mathsf{MAC}_{K_B}\left(\langle TS_{\text{path}}, hi_B, \mathsf{MAC}_{K_A}(hi_A)\rangle\right)$
$\mathsf{MAC}_{K_D}\left(\langle TS_{\text{path}}, hi_D, \mathsf{MAC}_{K_B}(\langle hi_B, \mathsf{MAC}_{K_A}(hi_A)\rangle)\rangle\right)$
...

FIGURE 3.1: EPIC Level 0 hop authenticator that contain nested MACs. The fields hi_i contain the local forwarding data of AS i.

For the AS initiating the beacon, there is no previous AS A_j and hence no hop authenticator, so $\sigma_j^{(0)}$ is not included. Figure 3.1 illustrates the nested MACs.

This hop authenticator directly serves as the hop validation field in the data plane, $hvf_i^{(0)} = V_i^{(0)} = \sigma_i^{(0)}$; segment identifiers and additional header fields ts_{pkt} and V_{SD} as defined in Equation (2.2) are therefore unused in EPIC Level 0. The procedure to create and forward packets is the following:

SOURCE H_S obtains a path, including all hop authenticators, from the path server in its AS. It constructs the packet according to Equation (2.2) by copying the path timestamp and the hop information and hop authenticator for each hop.

TRANSIT At every AS A_i, the border router first checks that the packet was received through the correct interface according to hi_i and that the hop field is not expired. Then the router recalculates $V_i^{(0)} = \sigma_i^{(0)}$ according to Equation (3.1) and checks that it coincides with the validator in the packet header. If the packet passes both checks, the router forwards it to the next hop specified in hi_i, otherwise it drops the packet. The only state required on AS border routers is the AS' secret key K_i and intra-AS forwarding information.

The construction presented here ensures that end hosts and ASes can only send packets on paths that are authorized by *all* on-path ASes. Chaining hops by including the hop authenticator of the previous hop in the MAC calculation defined in Equation (3.1) guarantees that *complete upstream paths* are authorized and hosts cannot combine individual hops arbitrarily.

3.1.2 Level 1: Improved path authorization

EPIC Level 0 suffers from a dilemma between secure hop fields and acceptable communication overhead: Assuming short hop authenticators with $l_{\text{val}} = 3$ (the default length of hop authenticators in SCION [88]), these fields are susceptible to online brute-force attacks. In certain topologies, an attacker has to send at most $2^{24} \approx 1.6 \cdot 10^7$ probe packets to find a correct MAC of an unauthorized hop, which takes under 10 seconds on a gigabit link. Afterwards, the attacker can use the unauthorized hop field to send arbitrary traffic until the eventual expiration of the path. MACs can be made longer and thus harder to forge, but only at the expense of increased communication overhead, see §3.3.3. The fundamental problem is that the static validators can be directly reused to send additional packets.

With EPIC Level 1 we resolve this dilemma by replacing these static hop authenticators by *per-packet* validators that cannot be reused for additional packets. During path exploration, an AS A_i calculates its hop authenticator σ_i as follows:

$$\sigma_i^{(1)} = \mathsf{MAC}_{K_i}\left(TS_{\text{path}},\ hi_i,\ S_j^{(1)}\right). \tag{3.2}$$

Here, $S_j^{(1)}$ is the segment identifier of the previous hop AS A_j during the path exploration, which is obtained by simply truncating the hop authenticator. For all i, we define:

$$S_i^{(1)} = \sigma_i^{(1)}\,[\![0{:}l_{\text{seg}}]\!]. \tag{3.3}$$

The hop authenticator is then subsequently used by the source host to calculate the per-packet validators:

$$V_i^{(1)} = \mathsf{MAC}_{\sigma_i^{(1)}}\left(ts_{\text{pkt}},\ \textsc{Src}\right)[\![0{:}l_{\text{val}}]\!]. \tag{3.4}$$

As the hop authenticators are not part of the packet header to limit communication overhead, the additional segment identifiers are required for chaining hops. They allow ASes to derive the hop authenticators on the fly. The aim of EPIC Level 1 is improving path authorization; the field V_{SD} is thus not used. An attacker trying to forge an unauthorized path needs to find at least one validator that fulfills Equation (3.4) without access to σ_i by sending a large number of probing packets. However, in contrast to EPIC Level 0, this validator cannot be used to send additional packets, which carry different packet timestamps.

Even though each validator is only valid for a specific packet origin, an attacker could launch a DoS attack by replaying packets for which he knows the validators or simply reusing the packet timestamp. From L1 onward, we employ a *replay-suppression system* in border routers or inside an AS' network to prevent this [70]. This system tracks and uniquely identifies packets based on the packet origin *PO*, i.e., source, path timestamp, and the packet timestamp, see Equation (2.3). In order for the packet origin to serve as a unique packet identifier, the packet timestamp must be sufficiently long, see §3.5.4 for a more detailed discussion. The replay-suppression system uses Bloom filters to identify duplicates but discards old packets in order to make this process viable in high-bandwidth networking applications, see §3.3.5. Note that packets are processed by the replay-suppression system *after* being authenticated in order to prevent an attacker from poisoning the system with unauthentic packets.

The procedure to create and forward packets is slightly more complicated than for EPIC Level 0:

SOURCE H_S obtains the desired path including all hop authenticators from its path server. H_S calculates the packet timestamp ts_{pkt} and adds it to the header. The host then calculates the segment identifiers according to Equation (3.3) and validators according to Equation (3.4) and constructs all hop fields consisting of hi_i, $S_i^{(1)}$, and $V_i^{(1)}$.

TRANSIT An AS checks the interfaces and expiration in the same way as in EPIC Level 0. It first recalculates the hop authenticator as in Equation (3.2) using the previous hop's segment identifier (in construction direction) and then recalculates its own segment identifier according to Equation (3.3) and the validator as in Equation (3.4). If interfaces, segment identifier, and validator are all correct, and the timestamp is current, the AS forwards the packet, otherwise it drops it.

3.1.3 Level 2: Authentication

We now extend the previous protocol by a mechanism to allow intermediate routers to authenticate the source of a packet and the destination to authenticate the entire packet. The hop authenticators σ_i, segment identifiers S_i, and the additional header field ts_{pkt} are unchanged. We define the *host keys*

$$K_i^S = K_{A_i \to A_1 : H_S} \tag{3.5a}$$

for every on-path AS A_i and an additional key

$$K_{\mathrm{SD}} = K_{A_\ell:H_\mathrm{D} \to A_1:H_\mathrm{S}} \tag{3.5b}$$

shared between source and destination. These keys are based on the derivation defined in Equation (2.5) and can be used to provide path authorization and source authentication in a single validator:

$$V_i^{(2)} = \mathsf{MAC}_{K_i^\mathrm{S}}(ts_{\mathrm{pkt}},\ \textsc{Src},\ \sigma_i)\,[\![0{:}l_{\mathrm{val}}]\!]. \tag{3.6}$$

The destination host H_D can authenticate the source of the packet and verify that neither the path (as defined in Equation (2.2b)) nor the payload was modified through the additional destination validation field

$$V_{\mathrm{SD}}^{(2)} = \mathsf{MAC}_{K_{\mathrm{SD}}}(ts_{\mathrm{pkt}},\ \textsc{Path},\ P)\,. \tag{3.7}$$

The procedure to create and forward packets is as follows:

SOURCE In addition to EPIC Level 1, the source H_S fetches all necessary host keys from the local key server and subsequently calculates the validators according to Equation (3.6) as well as V_{SD} according to Equation (3.7).

TRANSIT In addition to the checks in EPIC Level 1, every AS needs to recalculate the host key K_i^S according to Equations (2.4), (2.5a), and (3.5a) and then check if the validator in the packet header satisfies Equation (3.6). As all keys can be locally calculated, no key fetching or per-host state is necessary.

DESTINATION H_D obtains the key K_{SD} from its local key server and validates $V_{\mathrm{SD}}^{(2)}$ according to Equation (3.7).

3.1.4 Level 3: End-host path validation

EPIC Level 3 further extends the security properties of EPIC Level 2 by enabling the source and destination of a packet to perform path validation. To that end, on-path ASes *overwrite* their validators with proofs to the source and destination that they have processed the packet. Upon receiving the packet, the destination can directly validate the path based on the destination validation field and enables path validation for the source by replying with a confirmation message. We define

$$C_i = \mathsf{MAC}_{K_i^\mathrm{S}}(ts_{\mathrm{pkt}},\ \textsc{Src},\ \sigma_i)\,, \tag{3.8}$$

which is equal to Equation (3.6) without truncation. This cryptographic result has a length of l_{PRF} bytes, which is generally longer than the validators that we propose in this work. This allows us to split the result into multiple separate pieces, which are uncorrelated as we assume the MAC to be a PRF; in particular, under the assumption $l_{\text{PRF}} \geq 2 \cdot l_{\text{val}}$, we can define

$$C_i^{[1]} = C_i[\![0{:}l_{\text{val}}]\!], \quad C_i^{[2]} = C_i[\![l_{\text{val}}{:}2l_{\text{val}}]\!]. \tag{3.9}$$

The source then performs the same setup as for EPIC Level 2, setting each validator to $V_{i;0}^{(3)} = C_i^{[1]}$ (which equals $V_i^{(2)}$). The router in AS A_i calculates the C_i defined in Equation (3.8) and checks that the validator is correct. Finally, it updates the validator with $V_{i;i}^{(3)} = C_i^{[2]}$. Without requiring any additional cryptographic computation, the router thus leaves a confirmation for H_{S} that it successfully validated and forwarded the packet (assuming that A_1 is honest), since only A_i, H_{S}, and A_1 can compute $C_i^{[2]}$. We allow H_{D} to also validate this confirmation (under the further assumption that H_{S} and A_ℓ are honest) by including the correct final values $V_{i;\ell}^{(3)}$ in the destination validation field:

$$V_{\text{SD}}^{(3)} = \mathsf{MAC}_{K_{\text{SD}}}\left(ts_{\text{pkt}}, \textsc{Path}, V_{1;\ell}^{(3)}, \cdots, V_{\ell;\ell}^{(3)}, P\right). \tag{3.10}$$

Note that, as each validator is only updated once, we have $V_{i;\ell}^{(3)} = V_{i;i}^{(3)}$. In order to allow H_{S} to validate the path, H_{D} needs to send a confirmation message containing the timestamps of the original message together with the updated values $V_{i;\ell}^{(3)}$. To prevent circular confirmations, such a message must be sent to H_{S} as an EPIC Level 2 (or lower) packet (cf. §4.8). To validate the path, H_{S} stores the expected validators upon sending a packet. When it receives a reply by the destination that contains the values $V_{i;\ell}^{(3)}$ that the destination received, it validates them against the stored values. If no correct confirmation is received after some timeout, the source can conclude that the original packet has been lost or redirected.

Both source and destination host are free in their reaction to failed path validation: The destination can choose to ignore it and rely on the source to take appropriate action (soft fail) or reject the corresponding packets (hard fail). The source can switch paths on a short timescale and, in case of frequent failures, switch its Internet provider. Note that fault localization in general is a very complex problem and cannot be achieved through EPIC alone in an adversarial environment [12].

```
1:  procedure INITIALIZATION BY $H_S$
Require: SRC, DEST, $TS_{\text{path}}$, $K_{SD}$, $P$, $\forall i \in \{1,\dots,\ell\}$: $hi_i$, $\sigma_i$, $K_i^S$
2:      construct $\boxed{\text{PATH}}$ according to Equation (2.2b)
3:      $\boxed{ts_{\text{pkt}}} \leftarrow$ (current time) $- TS_{\text{path}}$
4:      for all $i \in \{1,\dots,\ell\}$ do
5:          $\boxed{S_i} \leftarrow \sigma_i[\![0{:}l_{\text{seg}}]\!]$          ▷ segment identifier (Equation (3.3))
6:          $C_i \leftarrow \mathsf{MAC}_{K_i^S}\left(\boxed{ts_{\text{pkt}}}, \boxed{\text{SRC}}, \sigma_i\right)$
7:          $C_i^{[1]} \leftarrow C_i[\![0{:}l_{\text{val}}]\!]$; $C_i^{[2]} \leftarrow C_i[\![l_{\text{val}}{:}2l_{\text{val}}]\!]$
8:          $\boxed{V_i} \leftarrow C_i^{[1]}$          ▷ initial value of validator
9:          $V_{i;\ell} \leftarrow C_i^{[2]}$          ▷ final value of validator
10:     $\boxed{V_{SD}} \leftarrow \mathsf{MAC}_{K_{SD}}\left(\boxed{ts_{\text{pkt}}}, \boxed{\text{PATH}}, V_{1;\ell}, \dots, V_{\ell;\ell}, P\right)$
11:     send $\boxed{\text{PACKET}}$ according to Equation (2.2)
12:     store $V_{i;\ell}$ for all $i$ under key $(TS_{\text{path}}, ts_{\text{pkt}})$ for validation

13: procedure VALIDATION AT $H_S$
Require: $K_{SD}$
14:     receive EPIC Level 2 packet with payload $\boxed{TS_{\text{path}}}$, $\boxed{ts_{\text{pkt}}}$, and $\boxed{V_1} \dots \boxed{V_\ell}$
15:     if EPIC Level 2 verification failed then
16:         return "validation failed"
17:     if $\langle \boxed{TS_{\text{path}}}, \boxed{ts_{\text{pkt}}} \rangle$ is not a valid key in store then
18:         return "validation failed"
19:     retrieve $V_{i;\ell}$ for all $i$ under key $\langle \boxed{TS_{\text{path}}}, \boxed{ts_{\text{pkt}}} \rangle$
20:     for all $i \in \{1,\dots,\ell\}$ do
21:         if $\boxed{V_i} \neq V_{i;\ell}$ then
22:             return "validation failed"
23:     return "validation succeeded"
```

ALGORITHM 1: Initialization and path validation at H_S in EPIC Level 3. The second procedure is executed upon receiving a reply packet that contains the path validation proof for the source. Packet contents such as header fields are denoted by $\boxed{\text{FIELD}}$ and $\leftarrow$ is an initialization or assignment. For readability, some superscripts omitted.

```
1: procedure Forwarding by AS $A_i$
Require: $K_i$
2:     if packet is not received through the interface $\boxed{prev}$ in $\boxed{hi_i}$ then
3:         drop packet
4:     if $\boxed{ts_{\text{exp}}}$ in $\boxed{hi_i}$ is expired then
5:         drop packet                                   ▷ hop field is expired
6:     if (current time) $- \boxed{TS_{\text{path}}} - \boxed{ts_{\text{pkt}}} \notin [-\epsilon, L+\epsilon]$ then
7:         drop packet        ▷ packet timestamp invalid (lifetime $L$, clock skew $\epsilon$)
8:     $(A_1 : H_S) \leftarrow \boxed{\text{Src}}$
9:     $K_{A_i \to A_1} \leftarrow \mathsf{PRF}_{K_i}(A_1)$          ▷ derive AS-level DRKey (Equation (2.4))
10:    $K_i^S \leftarrow \mathsf{PRF}_{K_{A_i \to A_1}}(H_S)$       ▷ derive host-level DRKey (Equation (3.5))
11:    $\sigma_i \leftarrow \mathsf{MAC}_{K_i}\left(\boxed{TS_{\text{path}}}, \boxed{hi_i}, \boxed{S_j}\right)$   ▷ hop authenticator (Equation (3.2))
12:    if $\boxed{S_i} \neq \sigma_i[\![0{:}l_{\text{seg}}]\!]$ then        ▷ check segment identifier (Equation (3.3))
13:        drop packet
14:    $C_i \leftarrow \mathsf{MAC}_{K_i^S}\left(\boxed{ts_{\text{pkt}}}, \boxed{\text{Src}}, \sigma_i\right)$
15:    $C_i^{[1]} \leftarrow C_i[\![0{:}l_{\text{val}}]\!]$; $C_i^{[2]} \leftarrow C_i[\![l_{\text{val}}{:}2l_{\text{val}}]\!]$
16:    if $\boxed{V_i} \neq C_i^{[1]}$ then                          ▷ authenticate packet
17:        drop packet
18:    $\boxed{V_i} \leftarrow C_i^{[2]}$                            ▷ update validator
19:    forward packet according to $\boxed{hi_i}$
```

ALGORITHM 2: Packet validation and updates at intermediate routers in EPIC Level 3. Syntax as in Algorithm 1.

```
1: procedure VALIDATION AND REPLY AT $H_D$
Require: $K_{SD}$
2:     ${V_{SD}}' \leftarrow \mathsf{MAC}_{K_{SD}}\left(\boxed{ts_{pkt}}, \boxed{\textsc{Path}}, \boxed{V_1}, \ldots, \boxed{V_\ell}, P\right)$
3:     if $\boxed{V_{SD}} \neq {V_{SD}}'$ then
4:         return "validation failed"
5:     if (current time) - $\boxed{TS_{path}}$ - $\boxed{ts_{pkt}}$ $\notin [-\epsilon, L+\epsilon]$ then
6:         return "validation failed"          ▷ timestamp expired or in the future
7:     send EPIC Level 2 packet to $H_S$ with payload
8:         $\langle TS_{path}, ts_{pkt}, V_1, \ldots, V_\ell \rangle$
9:     return "validation succeeded"
```

ALGORITHM 3: Packet and path validation at H_D in EPIC Level 3. Syntax as in Algorithm 1.

The algorithms for initialization, validation, and update in EPIC Level 3 are shown in Algorithms 1–3. These algorithms do not include the replay-suppression system, which we assume is an external system in each AS that inspects the packet origin of all authenticated packets and eliminates any duplicates. Algorithms 2 and 3 both enforce the validity of the absolute timestamp $TS_{path} + ts_{pkt}$: the packet must neither exceed a fixed lifetime L nor must this timestamp lie in the future. These checks take into account a maximum clock skew of ϵ.

3.2 Security analysis

In this section we define the security properties in turn and compare our protocols with ICING [83], OPT [64], and PPV [105]. An overview is shown in Table 3.1.

We propose different variants of a Dolev-Yao adversary, but we do not give any proofs in this chapter. The formal models and proofs of path authorization are presented in Chapter 4.

3.2.1 Basic and strong attacker models

BASIC-ATTACKER MODEL A Dolev–Yao adversary can typically observe, drop, inject, replay, or alter packets anywhere in the network [37]. However, if an attacker can re-route packets arbitrarily, it becomes impossible to ensure that packets follow authorized paths. We therefore consider all

	who	add. honesty assums.	packets
P1: path authorization	A_i	–	non-oracle
P2: freshness	A_i, H_D	–	all
P3: packet authentication	H_D	H_S, A_1, A_ℓ	all
P4: source authentication	A_i	H_S, A_1	non-oracle
P5: path validation	H_S	A_1	all
P6: path validation	H_D	H_S, A_1, A_ℓ	all

	L0 (BA)	L1 (SA)	L2 (SA)	L3 (SA)	ICING (BA)	OPT (BA)	PPV (BA)
P1: path author.	✓	✓	✓	✓	✓	✓	✗
P2: freshness	✗	✓	✓	✓	(✓)	(✓)	✗
P3: packet authen.	✗	✗	✓	✓	✓[2]	✓	✓
P4: source authen.	✗	✗	✓	✓	✓[2]	✓[1]	✗
P5: path validation	✗	✗	✗	✓	✓[2]	✓	✗
P6: path validation	✗	✗	✗	✓	✓[2,3]	✓	(✓)[3]

[1] A_i has to additionally assume the honesty of H_D.

[2] A_1 and A_ℓ do not need to be honest.

[3] A_1 and H_S do not need to be honest.

TABLE 3.1: The lower table lists ✓ satisfied, (✓) partially satisfied, and ✗ unsatisfied properties of our protocols EPIC Levels 0–3, as well as of ICING, OPT, and PPV. The upper table defines the scope and trust assumptions of the properties. The 2nd column list for whom the property holds. These entities are assumed to be honest. The 3rd column lists additional entities that have to be assumed to be honest for the property to hold. For protocols evaluated in the strong-attacker model (SA) rather than the basic-attacker model (BA), the 4th column indicates if the property holds only for packets that do not originate from the oracle, or for all packets.

packets that the attacker sends as new packets. An attacker can compromise one or multiple ASes, including their routers, end hosts, and cryptographic keys. This attacker can only send and receive packets at the compromised (and colluding) AS locations. It represents our *basic attacker*.

As is standard in Dolev–Yao models, our model assumes cryptography to be perfect. Consequently, the cryptographic primitives that the protocol is built on must be secure. In particular, this requires that cryptographic keys and authentication fields be sufficiently long to prevent an attacker from brute-forcing authentication fields. If short keys or fields are used, the model's assumptions are violated and the security guarantees no longer hold. For instance, in the case of EPIC Level 0, if a short hop authenticator was used, in certain topologies an attacker would only need to forge a single hop field to create an unauthorized path that could be used to send an arbitrary number of malicious packets that violate *path authorization*. Consequently, EPIC Level 0 must use long authentication fields to be secure.

STRONG-ATTACKER MODEL In contrast, our protocols EPIC Levels 1–3 are designed to decrease communication overhead by using short validators and segment identifiers. A malicious sender could therefore send large amounts of probing packets—and, with a small chance, guess the correct values for these fields in individual packets.

To reflect the attacker's ability to brute-force the validators and segment identifiers in the model, we propose a *strong-attacker* model, which weakens the assumption of perfect cryptography of the basic attacker. In particular, this model allows a malicious sender to obtain valid validators and segment identifiers of the validation header by querying an *oracle*.

We define for EPIC levels $l \in \{1, 2, 3\}$ $oracle(l)$ as the function that for given PO and HI fields produces valid validators V_i and segment identifiers S_i:

$$oracle(l)(PO, hi_1, \dots, hi_\ell) = (V_1^{(l)}, \dots, V_\ell^{(l)}, S_1^{(l)}, \dots, S_\ell^{(l)}). \tag{3.11}$$

The attacker can thus query the oracle and learn the V_i and S_i (but not σ_i or V_{SD}). As this allows him to trivially construct packets that violate the security properties for ASes, we restrict the security guarantees to packets whose origin PO was not part of an oracle query. Security under this model then means that, while the attacker may be able to forge individual packets (obtained from the oracle in the model), this does not help him to craft *different* packets that violate the guarantees.

Additionally, we need to argue in our security analysis that forging individual packets (as modeled by an oracle invocation) does not represent a serious risk for the security of the system: in the next subsection, we will show that the *likelihood* of success of such an attack is low in many practical cases and, even if it succeeds, its *impact* is severely limited. Consequently, the attacker's benefit from brute-forcing a packet is small compared to the computational costs involved.

Protocols that are secure under the basic-attacker model are not necessarily less secure than those under the strong-attacker model, but their implementations must ensure that authenticators are long enough to rule out any practical brute-force attacks. The length of the authenticators is crucial for the communication overhead, which we discuss later.

3.2.2 Low risk of forging individual packets

The strong-attacker model explicitly acknowledges the ability of an attacker to brute-force individual validators and segment identifiers in the EPIC Levels 1–3 through its oracle. However, in practice, the risk of such an attack is limited in four ways.

First, forging even a single packet (i. e., at least one validator) is expensive as it cannot be performed locally but only by sending packets. Second, a forged packet will be forwarded at most once. The validators are bound to the packet origin (source and timestamp). If the attacker brute-forces a validator and creates an unauthorized (but valid) path, she can violate path authorization or source authentication at routers, but only for a specific *PO*. Any packets with an outdated timestamp in their *PO* will be dropped immediately by routers, meaning that the attack can only happen in a short time frame. Furthermore, the replay-suppression system prevents more than one packet with the same *PO* from being forwarded. Third, in many practical attacks more than a single validator needs to be brute-forced and the attack becomes exponentially harder in the number of fields to be forged. The probability of forging n validators and segment identifiers for any packet is given by $2^{-8n(l_{\text{val}}+l_{\text{seg}})}$. Fourth, the security guarantees for end hosts are not affected, since they are based on the validation field V_{SD}, which is cryptographically strong.

Attacks that only allow a tiny number of packets to be falsely validated by ASes do not pose a grave threat to them. Their concerns regarding path authorization are primarily driven by economic interests, and it suffices if path-policy enforcement works for the vast majority of packets. On the other hand, the main application of source authentication at routers is DoS defense by filtering out unauthentic packets before they reach a bottleneck and enforcing bandwidth reservations through source attribution. For these in-network security applications a small number of forged packets that fool routers (but not the destination) have minimal consequences.

3.2.3 Path authorization

The following property protects the path policies of ASes:

P1 *Path authorization*: Packets traverse the network only along paths authorized by all honest on-path ASes.

This enforces the control-plane choices in the data plane and prevents path-splicing attacks: in these, a malicious source would combine hop fields from multiple authorized paths to create an unauthorized path. An *on-path attacker* can exchange the authorized path that the source picked by a different authorized path. Nevertheless, each portion of the path that the packet traverses along *honest* ASes is still authorized.

EPIC LEVEL 0 AND OPT EPIC Level 0 satisfies path authorization due to its chained hop authenticators: each authenticator contains in its MAC recursively all previous authenticators (in beaconing direction). Thus, the MAC binds the entire portion of the path from the authenticating AS to the end (in forwarding direction). Since the property is only achieved in the basic-attacker model, hop authenticators have to be long enough to prevent brute-force attacks. Otherwise, attackers could forge a path and not only use it to send a single packet, but use it for arbitrarily many packets until a hop field expires (based on ts_{exp}) or the ASes' long-term keys K_i are rotated. OPT also only satisfies P1 in the basic-attacker model since its mechanism is based on SCION / EPIC Level 0.

EPIC LEVELS 1–3 EPIC Level 1 and onward achieve property P1 in the strong-attacker model. These protocols create a validator for each hop, which is a MAC containing both the hop authenticator σ and the packet origin fields (SRC, TS_{path}, and ts_{pkt}).[1] The former ensures path authorization, similar to EPIC Level 0. The latter ensures that this property holds even under the strong attacker: an attacker who obtains a validator for a specific

1 TS_{path} is indirectly contained in the validator through the hop authenticator.

PO from the oracle cannot use it to create a validator that is valid for a different *PO*, as the validator is *bound* to its *PO*.

Both the segment identifier and the validator directly appear in the packet and are truncated for efficiency reasons. In contrast, the hop authenticator σ itself does *not* appear in the packet and thus does not need truncation as it can be recomputed on demand. The combination of long hop authenticators and short validators and segment identifiers minimizes risk; on one hand, a successful brute-forcing attack on a 16 B hop authenticator is practically infeasible; on the other hand, such an attack on a validator or segment identifier, which is possible by sending a large number of probing packets, has limited impact, as we have discussed in §3.2.2.

ICING AND PPV ICING achieves path authorization in the basic-attacker model through its *proofs of consent* (PoCs), which are used to calculate authenticators. PPV does not consider path authorization.

3.2.4 Freshness

In order to prevent DoS attacks by repeated packet resending, we require that each packet's origin (*PO*) is unique.

P2 *Freshness*: Packets are uniquely identifiable and cannot be replayed.

EPIC Levels 1–3 achieves freshness using a replay-suppression system where *PO*, i. e., the combination of source, path timestamp, and packet timestamp, serves as a unique packet identifier. With such a system in place, the attacker can send at most one unauthorized packet per forged validator, which is an enormous cost for a very limited return value.

EPIC Level 0 lacks unique packet identifiers required for replay suppression; ICING and OPT have limited support for replay suppression but do not discuss this in their work. PPV does not use sequence numbers or timestamps and instead uses a "PacketID" based on source, destination, and the hash of the payload. This is insufficient to uniquely identify packets or to enable an efficient replay-suppression system.

3.2.5 Packet and source authentication

Packet and source authentication are desirable properties at the network layer. We formulate authentication as non-injective agreement properties [77]. Together with property P2 and enforcement of the timestamp's validity, they yield strong recent-injective-agreement properties [77].

P3 *Packet authentication for H_D*: The destination H_D agrees with the source H_S on the packet origin, path, and payload unless H_S, its AS, or H_D's AS are corrupted.

P4 *Source authentication for routers*: On-path ASes agree with the source on the packet origin unless the source or its AS are corrupted.

EPIC EPIC Levels 0–1 do not provide any authentication. EPIC Levels 2–3 achieve P3 in the strong-attacker model by computing the destination validation field V_SD as the MAC under K_SD of the packet timestamp ts_pkt, the path (including TS_path), and the payload, see Equations (3.7) and (3.10). Since we assume that V_SD is unforgeable (it is not included in $oracle(l)$'s output), any source, path, payload, or timestamp modifications by an attacker can be detected by the destination.

EPIC Levels 2–3 achieve P4 since their validators are computed as the MAC under the host key K_i^S of the packet timestamp, the source, and the hop authenticator (which is calculated based on the path timestamp). The reasoning is similar to the one for property P3 above, with the difference that individual validators are forgeable by sufficiently strong attackers (included in $oracle(l)$'s output). The modification of part of the packet origin, i. e., the timestamps or the source, requires forging all honest ASes' validators on the path from the attacker to the destination. As a consequence, these routers may falsely authenticate the source of a packet, but, due to freshness (P2), this is limited to *individual packets*, see also §3.2.2.

OPT OPT authenticates the source and payload, but it achieves property P4 only in the basic-attacker model and only under the additional assumption that H_D is honest. This is due to the use of DRKeys of the form $K_{A_i \to A_1:H_\text{S},A_\ell:H_\text{D}}$, which are not only shared between H_S and the intermediate AS A_i, but also with H_D. This weakens the source-authentication property compared to EPIC as all validators could also have been created by A_ℓ or H_D. For example, if source authentication is used for bandwidth attribution, a malicious destination could slander the source by fabricating packets or sharing this key.

ICING AND PPV ICING achieves both authentication properties P3–P4 through its *proofs of provenance* (PoPs). PPV achieves property P3 through its "PacketID", which is calculated using a secret key shared between H_S and H_D. There is no mechanism in PPV for authentication to routers (P4).

HONESTY ASSUMPTIONS In all schemes discussed here except for ICING (which is not based on DRKey), an end host's use of a host key shared with its AS requires the host's trust in its AS. While this may appear like a strong assumption, a malicious source or destination AS would need to launch an active attack to circumvent the authentication mechanisms, which hosts can detect by comparing authenticators out of band. Hosts have contracts with their ASes and could have a legal remedy when misbehavior occurs. This is in stark contrast to today's Internet, where hijacks can be performed by an off-path adversary with no relationship to the affected hosts, and no common jurisdiction to settle disputes.

The alternatives to using DRKey in the data plane are using asymmetric cryptography or using symmetric cryptography with pairwise end-to-end keys, which both violate our efficiency requirements (see Sections 2.1.5 and 2.2.4).

3.2.6 Path validation

Path-validation properties ensure that the *actual* path corresponds to the sender's *intended* path. This is primarily interesting to the end points, for instance if there are compliance rules that mandate certain paths. It can be considered the dual property to path authorization: while path authorization protects the routing decisions of ASes from malicious end hosts, path validation protects the path choices of end hosts from malicious on-path ASes.

P5 *Path validation for H_S*: Upon receiving a reply from H_D, the source H_S can verify that the original packet traversed all honest ASes on the path intended by H_S.

P6 *Path validation for H_D*: H_D can verify that the packet traversed all honest ASes on the path from H_S to H_D intended by H_S.

Both P5 and P6 are achieved by EPIC Level 3 in the strong-attacker model through the destination validation field V_{SD} (for which the attacker's ability to forge validators is irrelevant).

ICING and OPT also satisfy path-validation properties P5 and P6. They additionally ensure that ASes are traversed in the correct order. PPV does not allow the source to validate the path (P5) and only probabilistically validates individual links at the destination (P6).

HONESTY ASSUMPTIONS For EPIC Level 3 and OPT, property P5 requires that the source assumes the honesty of its own AS, since they share the host key. Likewise, for property P6, the destination must assume the honesty of its own AS and also of the source and its AS, since all validation fields are computed by H_S. This assumption is not needed for ICING, which does not rely on DRKey and uses separate keys for the destination. PPV also uses a key which is not shared with the source to achieve property P6 and therefore does not need to assume the source to be honest.

3.3 Implementation and evaluation

In this section, we describe our prototype implementation and evaluate its performance. In addition, we analyze the communication overhead of EPIC, OPT, ICING, and PPV as well as of supporting systems. For this analysis, we assume the sizes for various fields in the EPIC header shown in Table 3.2.

3.3.1 Implementation and measurement setup

To show that EPIC is practically feasible, we implemented and evaluated EPIC Level 3 prototypes for the source, the routers, and the destination according to the algorithm specification in Algorithms 1–3 using Intel DPDK [38]. As other EPIC levels have a strict subset of processing steps, they would achieve strictly better performance.

In summary, the following evaluation shows that the system can be implemented efficiently even on commodity hardware, it is parallelizable and scales well to core links on the Internet, has significantly lower communication overhead compared to existing systems, requires virtually no state on routers, and limits additional control-plane overhead.

EPIC PACKET STRUCTURE In our prototype implementation, we follow the packet structure of Equation (2.2), using the field sizes specified in Table 3.2, and extend it with some auxiliary fields (a pointer to the current hop field, the total path length, a version number, and additional flags) and an Ethernet header.

field	content	#	size
TS_{path}	path timestamp	1	4
SRC	source AS and host	1	8
V_i	validator	ℓ	3
S_i	segment identifier	ℓ	2
ts_{pkt}	packet timestamp offset	1	8
V_{SD}	destination validation field	1	16

TABLE 3.2: Size in bytes and number of occurrences (#) of various header fields in a path of length ℓ.

CRYPTOGRAPHIC PRIMITIVES As we calculate many PRFs and MACs over short inputs and want to avoid the overhead due to subkey generation of CMAC [56], we use the AES-128 block cipher in CBC mode for both PRFs and MACs. As we calculate MACs over variable-length inputs, we prepend the input length and use zero padding such that the CBC-MAC indeed fulfills all properties of a PRF and a MAC [17]. Because EPIC and DRKey heavily rely on MAC and PRF calculations, we use Intel's AES-NI hardware instructions [93], available on all modern Intel CPUs, to reduce the computation time.

HVF STORE AT THE SOURCE The store of validators of sent packets at the source is implemented as a hash table as it enables insertion and retrieval of data using the 12-byte key $(TS_{\text{path}}, ts_{\text{pkt}})$ with average complexity $\mathcal{O}(1)$ and there exists a ready-to-use hash-table implementation in DPDK.

MEASUREMENT SETUP The prototypes are evaluated using a Spirent SPT-N4U test module, which serves as packet generator and bandwidth monitor, and a commodity machine with an 18-core Intel Xeon 2 GHz processor executing the component to be tested, i.e., the source or router. The two machines are connected with a 40 Gbps Ethernet link.

We evaluate the performance of the prototype as a function of the EPIC Level 3 payload. However, the size of the EPIC header depends on the AS-level path length and therefore contributes dynamically to the Ethernet packet content. To test the prototypes using the same EPIC Level 3 payload range, independent of the path length, we enable jumbo-frame support (Ethernet frames with more than 1500 B payload) on both machines.

The current average path length in the Internet is less than 4 AS-level hops [55, 79, 102, 105]. However, as we expect that number to increase due to the benefits of being an AS in a path-aware Internet, we consider path lengths of up to 16 AS-level hops in our evaluation (the current average number of router-level hops is 13).

3.3.2 Performance evaluation

In this section, we evaluate the performance of our implementation in terms of throughput (total traffic) and goodput (payload traffic). Note that we account for the full header overhead as described above when referencing the goodput.

SOURCE For the evaluation of the source we assume that it has already fetched the necessary hop authenticators and DRKeys, which corresponds to the situation of an existing connection. The throughput achieved by the source (using a single CPU core) is shown in Figure 3.2. For packets of $p \geq 500\,\mathrm{B}$ and path lengths of $\ell \leq 8$, the prototype implementation consistently achieves throughput above 2 Gbps. Figures 3.8 and 3.9 in §3.3.4 further illustrate the parallelizability of the implementation, which enables throughputs of tens of Gbps, and the linear increase of the processing time with both payload size and path length.

The processing at the source and destination is similar for ICING, OPT, and PPV; in all protocols either a MAC or hash is calculated over the packet's payload, which dominates the computational effort. In the future, these cryptographic computations could be offloaded to multiple dedicated hardware units in network-interface cards (NICs).

ROUTER Figure 3.3 shows the forwarding performance of an EPIC Level 3 router for a path of length $\ell = 8$. In these measurements, we assume no cached hop authenticators or DRKeys, they are always recalculated on the fly. For packets with a payload $p \geq 500\,\mathrm{B}$, the 40 Gbps link is saturated for all path lengths using only 4 cores; using 16 cores, the link is even saturated for small packets ($p = 100\,\mathrm{B}$). As the implementation is easily parallelizable (see Figure 3.7 in §3.3.4), it can be used even on 100 Gbps or 400 Gbps links by adding more processing cores or dedicated hardware. An important observation is that the processing time of the router is 445 –

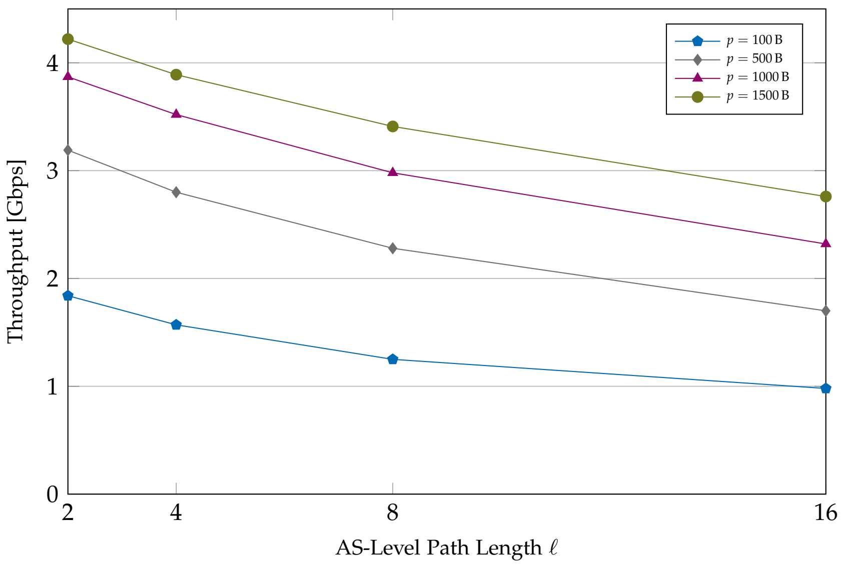

FIGURE 3.2: EPIC Level 3 packet throughput generated by the source on a single core for different payload sizes.

460 ns independent of both payload size and path length. The forwarding performance in terms of Mpps (million packets per second) is thus also independent of these parameters and amounts to approximately 2 Mpps per processing core. These results are further illustrated by Figures 3.5–3.7 in §3.3.4.

The processing on routers is similar for all levels of EPIC, OPT, and PPV, which all have a small constant number of cryptographic operations. In ICING, every router calculates both a hash and a MAC over the payload and in addition performs ℓ symmetric cryptographic operations (one for each router). In the software implementation provided by ICING's authors [83], each router has a processing time of $\sim$50 µs for $\ell = 10$, which is two orders of magnitude slower than EPIC. If keys are not cached, additional Diffie–Hellman computations are necessary, leading to processing times of ≥ 100 ms [40].

COMPARISON TO IP Comparing the performance of EPIC to IP is challenging due to the strong impact of routing-table sizes on software performance and hardware cost for IP. Highly optimized software switch implementations like DPDK vSwitch achieve throughputs of $\sim$11 Mpps

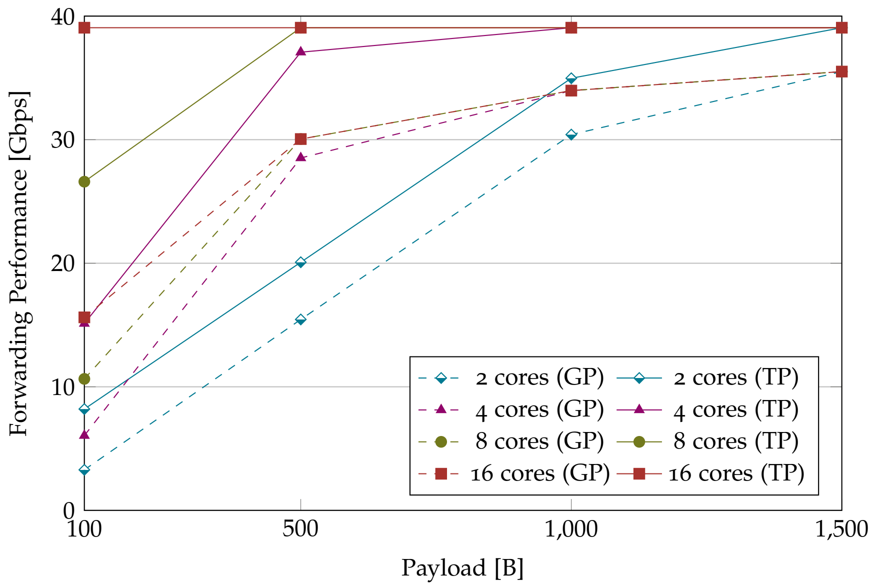

FIGURE 3.3: Throughput (TP) and goodput (GP) of a router plotted against the payload for 2, 4, 8, and 16 cores and $\ell = 8$.

	L0	L1	L2–L3	ICING	OPT	PPV
	3ℓ	$5\ell + 8$	$5\ell + 24$	$42\ell + 13$	$19\ell + 52$	64
for $\ell = 8$	24	48	64	349	204	64

TABLE 3.3: Communication overhead in bytes in EPIC, ICING, OPT, and PPV due to security-related fields.

on a single core (corresponding to a processing time of approximately 90 ns) [41]. However, these values are only valid for small routing tables when no memory accesses are necessary (as a single DRAM access takes ~70 ns). Our prototype implementation is approximately five times slower at ~2 Mpps, but the throughput is independent of the number of concurrent flows due to packet-carried forwarding state. Furthermore, the processing time could be further reduced through optimizations such as concurrent execution of cryptographic operations.

Hardware implementations, which are particularly relevant in a production deployment, compare even more favorably. IP routers require large amounts of expensive ternary content-addressable memory (TCAM) for longest-prefix matching. In contrast, EPIC requires very little additional hardware for its cryptographic operations. Naous et al. [83] have compared the gate count of FPGA implementations of ICING and IP routers and found comparable values (13.4 million vs. 8.7 million gates) even for very small amounts of TCAM in the IP router; in comparison, hardware implementations of AES are very efficient and only require 13,000 gates [2].

3.3.3 Communication overhead

In addition to processing overhead and performance, we also evaluate the communication overhead of EPIC and compare it to other systems. To allow for a meaningful comparison, we evaluate only the overhead owed to security here, since the normal routing headers (e.g., IPv4/v6, SCION) depend on the underlying networking architecture. Thus, we use *HD* to refer to the size of all security-related header fields (in EPIC, these are ts_{pkt}, V_{SD}, and S_i, V_i for all hops i). We define the goodput ratio as the ratio between goodput and throughput, or, equivalently, as the ratio of payload and total packet size, $GR = \frac{p}{p+HD}$. Table 3.3 shows the size of the security-related header for all considered systems, Figure 3.4 depicts the goodput ratio.

We find that the goodput ratio is high for all variants of EPIC. For $\ell = 8$, the security-related header is between 24 B for EPIC Level 0 and 64 B for EPIC Level 3, which corresponds to a goodput ratios 98 % and 94 %, respectively, for payloads of size $p = 1000\,\text{B}$. The goodput ratio of OPT is significantly worse with $GR \sim 83\,\%$ for the same values of ℓ and p, and does not scale as well as the overhead of EPIC with the length of the paths. For ICING, we find a five times larger overhead than EPIC Levels 2–3 and $GR \sim 74\,\%$ for these parameters. As PPV performs checks at only two routers along the path, its overhead is constant in the path length. Still, EPIC Levels 2–3 have a higher goodput ratio than PPV for path lengths up to $\ell = 8$.

In Table 3.3, authenticators for path authorization are 3 B for EPIC Level 0 and OPT (the default for SCION on which they are based). This is despite only achieving property P1 in the basic-attacker model, meaning that brute-force attacks are unmitigated and exploitable for practical attacks. To correct for this, the size of validators would need to be increased to a similar length of other brute-force-resistant fields like the destination validation field, i.e., 16 B. Considering these modifications, which are shown by "sec. L0" and "sec. OPT" in Figure 3.4, the goodput ratio is even more favorable for EPIC Levels 1–3, which significantly outperform both protocols.

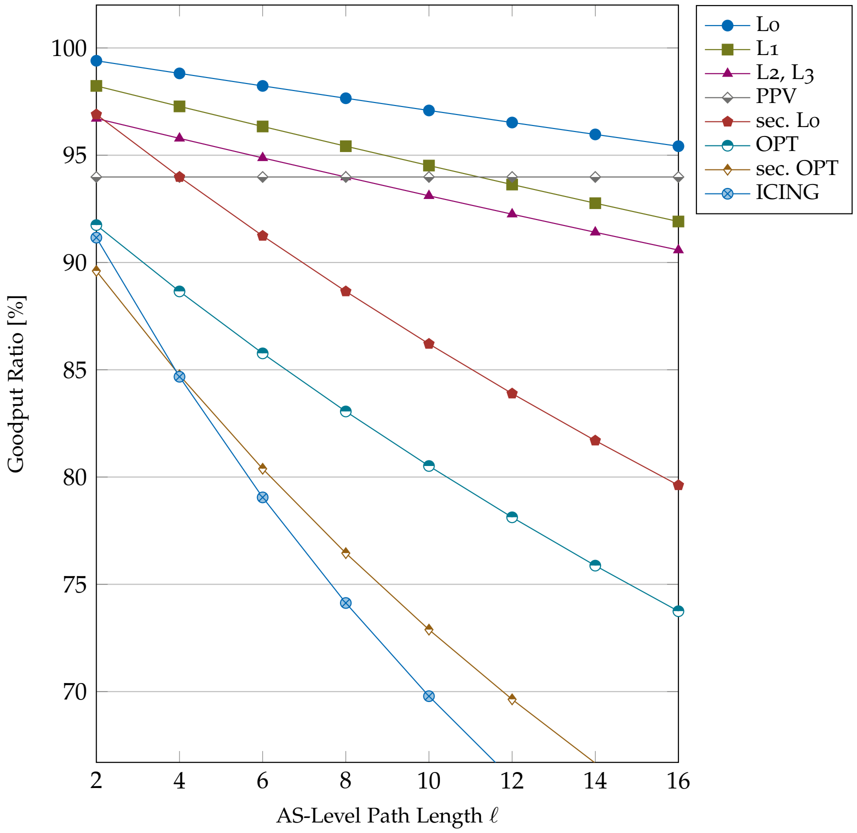

Figure 3.4: Goodput ratio of different protocols as a function of the AS-level path length ℓ for a 1000 B payload, calculated from Table 3.3. "sec. L0" and "sec. OPT" correspond to L0 and OPT with authenticators of 16 B that are required to rule out brute-force attacks. For $GR < 2/3$, the total packet size exceeds the maximum size for an Ethernet payload.

3.3.4 Additional evaluation results

PROCESSING-TIME ANALYSIS Figure 3.5 shows a fine-grained processing-time analysis of the router for EPIC Level 3, highlighting the overhead caused by cryptographic operations. The times include necessary copying and padding of the input to the AES block cipher.

Figure 3.6 shows the processing time of an EPIC Level 3 router for different path lengths and EPIC payload sizes. As expected, the processing time is independent of the path length and payload size and shows low deviation of only a few percent.

PARALLELIZABILITY As shown in Figure 3.7, the router implementation achieves almost perfectly linear speedup when parallelized over multiple CPU cores. As a consequence, the EPIC router can be easily scaled to larger network links by adding more processing cores or dedicated hardware. The source shows a similar linear speedup as a function of the number of cores, see Figure 3.8.

PROCESSING TIME AT THE SOURCE The processing time at the source for EPIC Level 3 is depicted in Figure 3.9. It increases linearly with both the AS-level path length (due to the validator for each hop) and in the EPIC payload (due to the destination validation field).

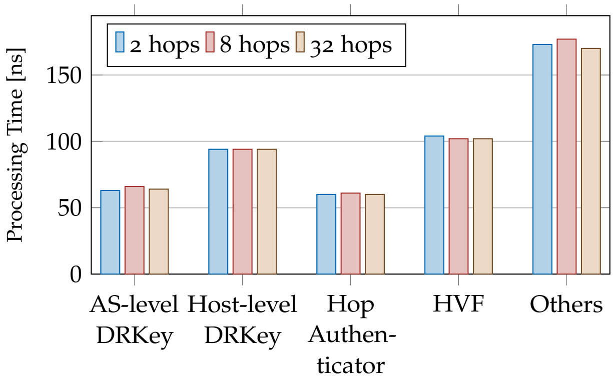

Figure 3.5: EPIC Level 3 router processing times for different sub-tasks. The category 'Others' aggregates all non-cryptographic operations, for example checking the expiration time, writing the updated hop-validation field, or increasing the hop pointer.

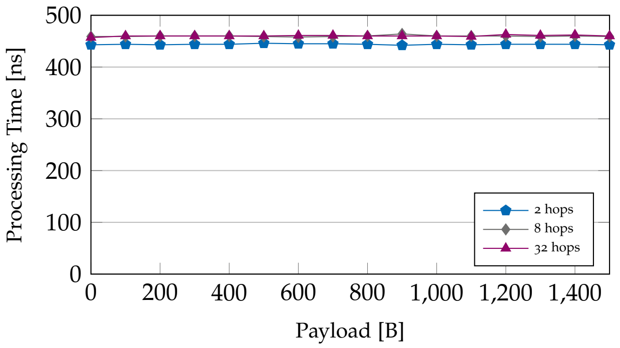

Figure 3.6: EPIC Level 3 router processing time as a function of the payload for $\ell \in \{2, 8, 32\}$.

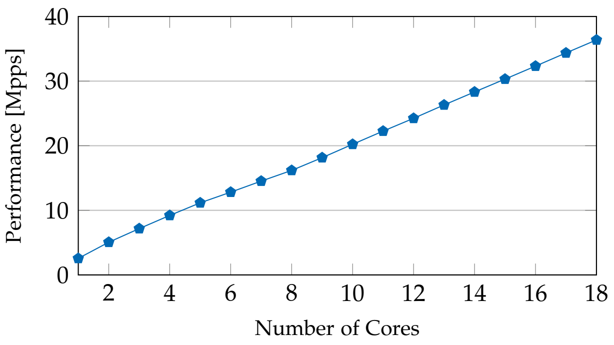

FIGURE 3.7: Forwarding performance of a router, as a function of the number of used cores measured for $p = 0$ and $\ell = 2$. As the packet-processing time is independent of p and ℓ as shown in Figures 3.5 and 3.6, this result is also valid for larger packets and longer paths.

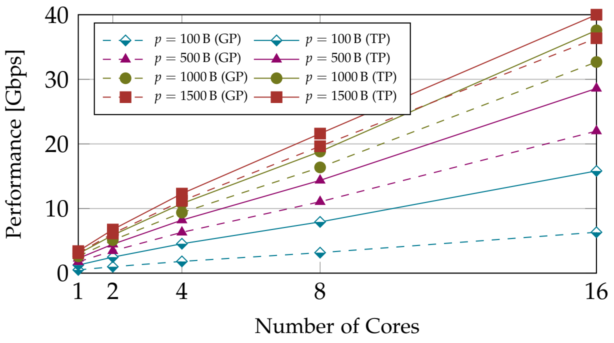

FIGURE 3.8: EPIC Level 3 packet-generation performance at the source, plotted against different number of cores and payload sizes, and for $\ell = 8$. The legend entries 'TP' and 'GP' denote the throughput and goodput, respectively.

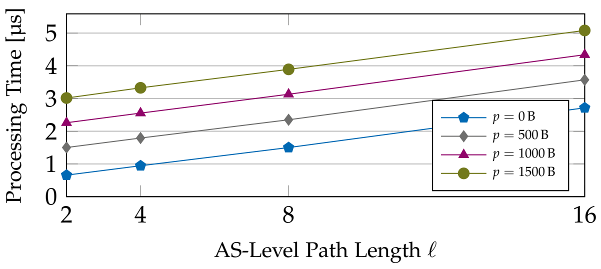

FIGURE 3.9: EPIC Level 3 packet-processing time at the source on a single core for different EPIC payload sizes and path lengths.

3.3.5 Other overhead

STATE AT ROUTERS In EPIC, routers can perform all cryptographic checks and updates with a single AS-specific secret value, there is no per-host or per-flow state required. This is equivalent to OPT and PPV, which both rely on DRKey, but a significant advantage compared to ICING, which requires per-flow state [64, 83]. In terms of routing information, border routers only need to store intra-AS information as packet headers contain the inter-AS forwarding information. This is a huge improvement over the current Internet, shared by all architectures based on packet-carried forwarding state.

REPLAY SUPPRESSION All EPIC levels except L0 depend on a replay-suppression system for freshness (P2), which has additional state and overhead. Since this task can be taken over by dedicated machines, we did not include it in the router measurements above. Prototypes that are entirely implemented in software have been deployed successfully on 10 Gbps links [70]. In turn, EPIC Levels 1–3 provide important properties for replay suppression: (i) the system can use the timestamp to discard packets that are expired, thus limiting the number of packets that need to be tracked in Bloom filters, and (ii) by authenticating all packet contents tracked by the replay-suppression system, attackers are prevented from modifying unauthenticated fields and replaying packets. If the replay-suppression system were not deployed, the packet timestamp could still be used to filter out expired packets, and an attacker could only replay packets in a very short time window due to the check in line 6 of Algorithm 2.

CONTROL-PLANE OVERHEAD In EPIC, end hosts have to request paths from the path server and, for EPIC Levels 2–3, host-level symmetric keys from the key server, before they can communicate with a new destination. We assume that the underlying path-aware Internet architecture minimizes latency by locally caching public paths, e.g., at path servers in SCION [88]. End hosts also cache paths themselves, such that only the initial packet to a new destination requires a path lookup. This caching strategy can also be applied to EPIC's hop authenticators and the host keys required in EPIC Levels 2–3. Concretely, AS-level keys can be set up between every pair of ASes ahead of time (either using PISKES / DRKey or Passport) such that local key servers can immediately respond to requests by end hosts. In the current Internet, storing 16 B keys for each AS only amounts to $\sim$1 MB [80]. Given that path and key information is available at local ASes, the additional latency incurred in EPIC is minimal: only the round-trip time between the source host and its own AS, and the destination host and its AS is added to the connection setup. End hosts can cache both paths and host keys, which eliminates additional latency for subsequent packets. Further optimization would be possible by combining DNS, path, and key requests, which would eliminate all additional latency for the initial packet compared to today's Internet.

3.4 Path validation for routers

As we argued in §2.1.3, path validation is primarily interesting to the end points. Despite this, ICING and OPT allow not only the source and destination to validate the path of a packet, but also enable intermediate routers to validate the portion of the path that has already been traversed. The authors of ICING and OPT provide little motivation to provide path validation for routers, and since we are not aware of any important use cases of this feature we have omitted it from our main protocols EPIC Levels 1–3.

However, for the sake of completeness we describe EPIC Level 4, which extends EPIC Level 3 by a security mechanism to also satisfy path validation for routers:

P7 *Path validation for routers*: Each router A_i can verify that the packet traversed all honest ASes from H_S to A_i on the path intended by H_S.

This protocol otherwise has the same security properties and communication overhead as EPIC Level 3.

	after processing by				
validator	H_S	A_1	A_2	A_3	A_4
$V_1^{(4)}$	$C_1^{[1]}$	$C_1^{[2]}$	$C_1^{[2]}$	$C_1^{[2]}$	$C_1^{[2]}$
$V_2^{(4)}$	$C_2^{[1]} \oplus C_1^{[3]}$	$C_2^{[1]}$	$C_2^{[2]}$	$C_2^{[2]}$	$C_2^{[2]}$
$V_3^{(4)}$	$C_3^{[1]} \oplus C_2^{[3]} \oplus C_1^{[3]}$	$C_3^{[1]} \oplus C_2^{[3]}$	$C_3^{[1]}$	$C_3^{[2]}$	$C_3^{[2]}$
$V_4^{(4)}$	$C_4^{[1]} \oplus C_3^{[3]} \oplus C_2^{[3]}$	$C_4^{[1]} \oplus C_3^{[3]} \oplus C_2^{[3]}$	$C_4^{[1]} \oplus C_3^{[3]}$	$C_4^{[1]}$	$C_4^{[2]}$

Table 3.4: Values of validators in EPIC Level 4 as a packet is forwarded from A_1 to A_4. Colors indicate α in $C_i^{[\alpha]}$.

In EPIC Level 4, the source of the packet *obfuscates* the validators of all ASes by XORing them with cryptographic results of previous ASes. Unless the previous ASes on the path reverse this obfuscation, the validator of an AS is invalid. As obfuscation values, we propose to use another piece $C_i^{[3]} = C_i[\![2l_{\text{val}}{:}3l_{\text{val}}]\!]$ of C_i defined in Equation (3.8) (assuming $l_{\text{PRF}} \geq 3 \cdot l_{\text{val}}$). The source of a packet now initializes the validator by

$$V_{i;0}^{(4)} = C_i^{[1]} \oplus C_{i-2^0}^{[3]} \oplus C_{i-2^1}^{[3]} \oplus \cdots \oplus C_{i-2^k}^{[3]} \tag{3.12}$$

and hence in an obfuscated way. The intermediate ASes update future validators by XORing them with $C_i^{[3]}$ and hence each remove a layer of obfuscation. Note that an obfuscation value $C_i^{[3]}$ is not used to obfuscate the validation fields $V_j^{(4)}$ *for all* subsequent j. As we show below, obfuscating the values of all subsequent ASes would enable colluding ASes to easily skip ASes on the path and deceive subsequent routers. Instead, we only use obfuscation at *exponentially growing* distances.

Table 3.4 presents the evolution of the validator values $V_i^{(4)}$ as the packet traverses four ASes. The source obfuscates the validators such that they will have the value $C_i^{[2]}$ upon reception by the destination if and only if all routers processed the packet successfully.

For EPIC Level 4, path validation for routers (P7) is achieved under the following honesty assumption in addition to those described in Table 3.1: on any contiguous part of the path of at least three hops there is a majority of honest ASes.

Hop-skipping attack

For property P7 in EPIC Level 4, colluding ASes may in some cases be able to deceive ASes on the future path to accept a packet, even if ASes on the past path were skipped, by analyzing validators and recovering the obfuscation values $C_i^{[3]}$ for skipped ASes i. An example with two skipped ASes is shown in Figure 3.10.

Note that in EPIC Level 4, if the validators V_i were obfuscated with $C_j^{[3]}$ for all $j < i$ (instead using exponential distances), two colluding ASes could always recover the obfuscation values of the ASes between them. Assume that there are colluding ASes i and j on the path and a number of honest ASes in between them. To skip the honest ASes, the malicious AS j after the skipped honest ASes would take the packet after it is processed by the first malicious AS i, compute it's own $C_j^{[1]}$ and XOR it to the $V_{j;i}^{(4)}$ field contained in the packet. The result is the XOR of the $C_k^{[3]}$ values for all skipped ASes k, i.e., $C_{i+1}^{[3]} \oplus \cdots \oplus C_{j-1}^{[3]}$. AS j then XORs this value onto the $V^{(4)}$ fields of all following hops. Thus, any two colluding ASes could create a wormhole that would be detectable by H_S and H_D but not by subsequent ASes.

Using exponential distances in obfuscation has an additional advantage: packet processing scales better with the path length, as fewer hop fields have to be updated by each router.

While the honesty assumption of a majority of honest ASes on any contiguous portion of the path of at least three hops seems safe, we have not proven the property under this (or any) honesty assumption. This honesty assumption is also rather strong, and it seems likely that there is a weaker assumption that accurately reflects the fact that EPIC Level 4 rules out attacks on path validation in many other scenarios. Finding a weaker honesty assumption and proving that it is sufficient to rule out attacks against EPIC's path validation is left for future work.

3.5 Discussion

3.5.1 Low communication overhead of EPIC

The benefit of EPIC's lower overhead compared to OPT comes in part from the fact that EPIC does not use separate fields for path authorization on the one hand, and for authentication and path validation on the other hand.

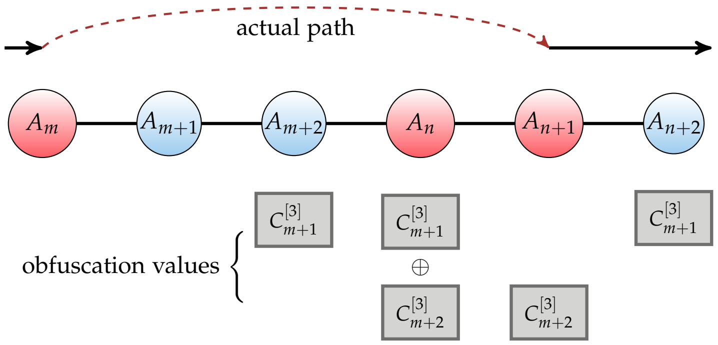

FIGURE 3.10: Example where colluding malicious ASes A_m, A_n, and A_{n+1} skip two intermediate ASes ($n = m + 3$) and deceive future ASes on the path to accept the diverted packet. The pattern of obfuscation values produced by A_{m+1} and A_{m+2} for following ASes is drawn below their nodes. From $V^{(4)}_{n;m}$ and $V^{(4)}_{n+1;m}$ that are embedded in the packet and the values $C^{[1]}_{n}$ and $C^{[1]}_{n+1}$ known to the attacker, the attacker can deduce the two values $C^{[3]}_{m+1} \oplus C^{[3]}_{m+2}$ and $C^{[3]}_{m+2}$. Hence, he can remove the obfuscation normally removed by A_{m+1} and A_{m+2} from all future validators. If only A_n but not A_{n+1} were controlled by the attacker, he would only be able to deduce $C^{[3]}_{m+1} \oplus C^{[3]}_{m+2}$ and therefore could not deceive A_{n+1}.

The larger contributor to a lower overhead is however the shorter length of validators in EPIC of 3 B, compared to the 16 B OPV fields in OPT. While a shorter authenticator translates to easier brute-force attacks (and thus seemingly weaker security), we have shown the practical usefulness of such attacks is severely limited by EPIC, as the attacker can only send a single packet that traverses an unauthorized path, and in EPIC Level 2-3 that packet will be discarded by the destination, see §3.2.2. EPIC is the first data plane protocol designed to limit the consequences of a successful brute-force attack to a single packet; previous protocols rely on long authenticators to prevent harmful attacks.

3.5.2 Deployment on path-aware architectures

Our data plane protocols are generic and applicable to a wide range of path-aware networking protocols. We now describe how EPIC fits into these architectures.

In SCION, the authenticators used in EPIC can be used directly instead of the built-in MACs that protect hop fields. However, a difference to EPIC is that in SCION only a subset of ASes called *cores* (typically, Tier-1 providers) initiate beacons. These beacons have limited reach and do not discover the entire Internet topology for scalability reasons. Thus, end hosts must combine paths from multiple beacons to obtain global end-to-end paths. SCION defines rules for combining multiple segments to rule out loops and uneconomical routes (such as *valley paths* [47] [88]) and allows paths to be used in either direction. While our presentation of EPIC abstracts from these aspects, we designed the protocols with path combinations and bidirectionality in mind. For combined paths, path authorization holds for each segment individually while path validation applies to the complete path.

Besides SCION, multiple other path-aware Internet architectures cryptographically protect forwarding directives in packet headers, including NEBULA [5, 83], PoMo [19], and Platypus [89, 90]. PoMo introduces an abstract "motivation" header that can be calculated in the same way as the validators of EPIC. NEBULA uses "proofs of consent" for path authorization and, with the ICING extension, achieves source authentication and path validation through its "proofs of provenance". EPIC can be used to replace these proofs to significantly reduce both computation and communication overhead while only slightly weakening security properties. The "bindings" in Platypus already implement a system similar to EPIC Level 1; they could, however, easily be augmented with source authentication and path validation with EPIC Levels 2–3.

3.5.3 Incremental deployment

The (incremental) deployment of a new Internet architecture is very challenging but is facilitated by the reuse of existing intra-domain infrastructure and protocols. An extensive discussion of (incremental) deployment is provided for the SCION architecture [88]. In turn, the incremental deployment of EPIC on an existing path-aware architecture—e.g., as a premium product for customers requiring stronger security properties such as the financial and healthcare sectors—is benefited precisely by their path awareness: EPIC only requires support by on-path ASes and can thus be supported on some paths without requiring global coordination. An upgraded end host can then favor these paths, providing benefits to early adopters.

3.5.4 Timestamps and time synchronization

The path timestamp TS_{path} encodes Unix time with second-level precision; both the expiration time of hop fields (ts_{exp}) and the packet timestamp introduced in EPIC (ts_{pkt}) are relative to TS_{path}. The length of the ts_{exp} field determines a maximum lifetime for hop fields. As a path expires when one of its hop fields expires, the packet timestamp offset ts_{pkt} only needs to cover the period between creation and expiration of a beacon. For instance, in SCION, this period is at most one day [88]. An 8 B field then corresponds to a granularity of $\sim$5 fs. This enables end hosts to send $2 \cdot 10^{14}$ packets with unique timestamps per second, which is sufficient for any practical application. We can consequently use the packet origin, i. e., the triple of source, path timestamp, and packet timestamp defined in Equation (2.3), to uniquely identify all packets in the network.

The timestamps serve multiple purposes in EPIC: they (i) allow routers to drop packets that are too old or use expired paths, (ii) uniquely identify packets, and (iii) ensure that the replay-suppression system only needs to track recent packets. For the first purpose, a coarse global time synchronization providing a precision of multiple seconds is sufficient. The second purpose does not require time synchronization at all, as packets are uniquely identified based on the packet origin, which also includes the source. The third purpose has been shown to work based on per-AS sequence numbers and therefore only requires relatively precise time synchronization *within* an AS [70]. The higher-order bits of the packet timestamp can serve as sequence numbers in this replay-suppression system.

3.5.5 Key distribution

The use of DRKeys in EPIC Levels 2–3 creates potential issues of circular dependencies: how is it possible to exchange DRKeys when they themselves are required for sending packets? In a steady state, this is unproblematic as ASes can proactively exchange new AS-level keys before the current keys expire using EPIC Levels 2–3 packets. For an initial key exchange, which only happens very infrequently, we propose to support EPIC Level 1 in addition, such that key requests can be sent over this lower-level protocol. Although EPIC Level 1 has lower security guarantees and may be susceptible to DoS attacks, these issues are mitigated by the fact that only a single request and response are needed for fetching a key. Even in a persistent, powerful DoS attack, such an exchange would succeed eventually.

3.5.6 Confirmation packets in EPIC Level 3

In EPIC Level 3, the confirmation message that allows the source to validate the path of its packets is sent as an EPIC Level 2 packet. This is necessary as each confirmation message would otherwise trigger yet another confirmation and consequently cause an infinite sequence of such confirmations. Using EPIC Level 2 means that the path of the original packet but not the confirmation message can be validated; all other security properties are retained (see also Table 3.1). Even in case a malicious on-path AS is able to modify the path of the confirmation message without being detected, this does not deteriorate the security properties of the original packet.

There are a number of possible optimizations for the confirmation message similar to acknowledgments in TCP: Instead of directly sending confirmation message for every received packet, the receiver can batch several confirmation messages and send them in a single packet. Confirmation messages can also be "piggy-backed" on normal data packets sent from the receiver to the source. Finally, instead of sending all validators, only a hash of them can be returned to the source and validated against the stored values.

3.5.7 Failure scenarios

As the EPIC protocols depend on several additional systems, a failure of any of these systems could potentially break connectivity. Most failure scenarios are comparable to similar issues in today's Internet: Failures of path or key servers are similar to failures of DNS servers today and can be prevented with similar techniques (e.g., replication, access control). Concerning potential misconfigurations, EPIC may actually increase the networks resilience as some concepts of new Internet architectures such as SCION's isolation domains ensure that the effects of misconfigurations are locally confined [88].

The most notable additional prerequisite of EPIC is time synchronization; it is possible that (i) a host, (ii) some router or server in an AS, or (iii) a complete AS is unsynchronized with the Internet. The first case can be handled by the host's AS replying with a corresponding control message triggering a re-synchronization. Cases (ii) and (iii) can be detected through increased packet-drop rates and can thus trigger a re-synchronization within the AS or with its neighbors. All cases may cause brief outages but can be resolved within a short time period (one second or less in most cases).

3.6 Related work

Over the past 15 years, much research was conducted on path-aware Internet architectures and routing schemes including Platypus [89, 90], PoMo [19], Pathlet Routing [50], NIRA [106], NEBULA [5], and SCION [88, 108]. Many of these systems recognized the need to find a balance of control between end hosts and ASes. This is why PoMo includes a "motivation" field containing a proof to routers that either the sender or receiver is a paying customer [19], NEBULA requires a "proof of consent" for the complete path of traversed ASes [5, 83], and SCION secures the authorization of its hop fields using MACs [88]. These solutions correspond to EPIC Level 0 in terms of the path authorization properties achieved. NIRA and Pathlet Routing obtain similar properties by restricting allowed paths (NIRA) and keeping state in routers (NIRA and Pathlet Routing) [50, 106]. Platypus uses a system similar to Level 1 presented in §3.1.2 where each network capability is secured by a "binding", but it does not address the issue of chaining multiple hops to paths [89, 90].

In addition, since PFRI (integrated into PoMo) discussed a high-level outline for a path-validation system via an "accountability" field in packets [27], multiple path-validation schemes have been proposed. ICING [83] is integrated into the NEBULA architecture and provides path validation using a validation field for each hop [83]. It uses aggregate MACs [59] in order to limit the bandwidth overhead but still requires each router to perform one symmetric cryptographic computation for *each* other router on the path (and, if keys are not cached, an additional *asymmetric* Diffie–Hellman computation), which makes it very expensive. Subsequent proposals try to reduce the complexity through different means: OPT reduces the required cryptographic computations to a constant number by sacrificing some guarantees for intermediate routers, yet it still has a high communication overhead [64, 88]. OSV tries to create a more efficient system by replacing cryptographic primitives by orthogonal sequences based on Hadamard matrices [24, 25]. Finally, PPV reduces both computation and communication overhead by only probabilistically validating a single link for each packet [105].

4 Formal Verification of Data Plane Protocols

Future Internet architectures offers more control to end users, higher efficiency and better security. While experience and testing can be used to evaluate the first two promises, they cannot be used to conclusively show the security of these systems. Testing does not cover all possible behaviors of a system, in particular of a distributed system with infinitely many behaviors. To show the security of such a system, one needs to find suitable abstractions, such that all behaviors can be reasoned about. Such techniques are provided by formal verification methods.

In this chapter, we formally verify secure data plane protocols. Since no such protocol is widely deployed yet on the Internet, and no likely winner can be determined at this point, we further abstract our formal proofs to not only capture the behaviors of a single protocol, but all behaviors of a class of protocols. This class of protocols includes EPIC Level 0–2 and a number of other data plane protocols.

We begin in §4.1 with the scope of this verification work. Section 4.2 introduces some formal notation and definitions. Section 4.3 presents an abstract model of a data plane protocol without an adversary, in which the security properties hold trivially. This model has environment parameters, for instance, to define the network topology and set of authorized paths. The concrete model in §4.4 introduces both an attacker and cryptographic checks to defend against the attacker. It is parametrized not only over the environment, but also introduces protocol parameters that abstract the security mechanisms used by protocols. As we show in §4.5, the concrete model refines the abstract model, and hence inherits the security properties. This refinement requires a set of assumptions on the environment parameters and a set of conditions on the protocol parameters.

We give concrete protocols in §4.6. These instance models define the protocol parameters and prove the associated conditions, thereby showing that they inherit the security properties.

Our actual formalization has a number of additional features, which for presentational reasons we do not include in the models presented in Sections 4.3–4.6. We introduce these as extensions of our framework in §4.7. Finally, we discuss our results in §4.8 and provide related work in §4.9.

4.1 Scope of verification

In this section, we describe the scope of our verification effort. We state the security properties that we verify and our attacker model. A formal account is given in Sections 4.3 and 4.4.

Our work applies to a wide range of future Internet architectures that follow a clean-slate path-aware approach [5, 19, 50, 83, 88, 89, 106, 108] and use cryptographic authenticators to establish path authorization.

4.1.1 Security properties that we verify

We verify two security properties: *path authorization* and *detectability* (cf. §2.1). They primarily protect ASes against malicious senders, but our attacker model (§4.4.3) also includes colluding on-path adversaries.

4.1.2 Security properties that we do not verify

Source and packet authentication allow border routers or the destination to authenticate the sender and packet. *Path validation* allows the destination to verify that the path contained in the packet was actually traversed. We do not verify these data plane properties, for reasons given in §4.8.5.

As in the previous chapter, intra-AS forwarding is out of scope, since each AS exercises control over its own network, and global coordination is not required for intra-AS security. We also do not specify or verify the control plane, as its properties are independent from those of the data plane. For instance, path authorization is independent of the property that a path authorized by the control plane is in accordance with the routing policies of all on-path ASes.

4.1.3 Verified data plane protocols

Our verification framework applies to both directed and undirected protocols. We analyze four data plane protocols, plus variants of these protocols, and prove path authorization and detectability:

- SCION [88], a complete future Internet architecture. We formalize its directed data plane.
- EPIC [71], a family of directed data plane protocols that provide three levels of security guarantees. We verify levels 1 and 2. EPIC levels 2 and 3 add authentication and path validation mechanisms, which we do not verify.
- Anapaya-SCION [4], a proposed successor to SCION, which uses mutable fields and XOR to accumulate authenticators.
- ICING [84], the undirected data plane protocol in the NEBULA Internet architecture [5]. It also provides path validation, which we do not verify.

We will formalize these protocols in §4.6 as instances of our parametrized framework.

4.2 Preliminaries

In this section, we provide background on event systems, refinement, and model parametrization. We also describe the differences in the packet format, notation and protocol features to the previous chapter. For instance, since our formalization abstracts from the control plane beaconing process, we refer to paths in forwarding direction, which is opposite to the beaconing direction used in SCION and EPIC.

Despite our use of Isabelle/HOL, we largely use standard mathematical notation and deliberately blur the distinction between types and sets.

4.2.1 Differences to previous chapter

The previous chapter presented the EPIC protocol suite. In order to specify these protocols as concretely as possible and to allow for a precise evaluation, we provided a detailed description that included low-level characteristics such as the length of individual header fields. Naturally, our formalization lacks such specificity and abstracts from several details.

For instance, the definition of packets (Equation (2.2), page 27) includes source and destination fields SRC, DEST, a payload field P, and the destination validation field V_{SD}. None of these fields are important for the path authorization and detectability security properties, hence they are not part of our models.

Furthermore, since our formal models are generic and apply to a wide range of different data plane protocols, we abstract from certain protocol-specific idiosyncrasies of EPIC. For instance, we have just a single field hvf_i for each AS i instead of a separate validator V_i and segment identifier S_i. When we instantiate our parametrized model with EPIC Level 1 in §4.6.2, we define hvf_i as a pair of V_i and S_i.

Lastly, the presentation of our formalization is simplified for ease of understanding. For instance, our formal model includes an authenticated per-packet field that can be used to include a timestamp (TS_{path} in the case of SCION and EPIC). We leave this field out of the definitions below. We also support a strong attacker model using an oracle, but leave out the required parameters and associated conditions. We introduce these features of our model in the section §4.7, which presents extensions of our simplified framework.

4.2.2 Event systems, invariants, and refinement

Event systems are labeled transition systems, where transitions are labeled with *events*. Formally, an event system is of the form $\mathcal{E} = (\mathbb{S}, s^0, E, \{\xrightarrow{e}\}_{e\in E})$, where $\mathbb{S}$ is a set of states, $s^0 \in \mathbb{S}$ is the initial state, E is a set of events, and $\xrightarrow{e} \subseteq \mathbb{S} \times \mathbb{S}$ is the transition relation corresponding to event e. As usual, we write $s \xrightarrow{e} s'$ for $(s, s') \in \xrightarrow{e}$. The set of states reachable from a state s, written $reach(\mathcal{E}, s)$, is inductively defined by $s \in reach(\mathcal{E}, s)$, and $s' \in reach(\mathcal{E}, s)$ and $s' \xrightarrow{e} s''$ implies $s'' \in reach(\mathcal{E}, s)$. A state property P is a subset of $\mathbb{S}$ (or, equivalently, a predicate on $\mathbb{S}$). A state property P is an *invariant* of $\mathcal{E}$, written $\mathcal{E} \models P$, if $reach(\mathcal{E}, s^0) \subseteq P$.

Given an abstract event system $\mathcal{E}_{\mathrm{a}} = (\mathbb{S}_{\mathrm{a}}, s^0_{\mathrm{a}}, E_{\mathrm{a}}, \{\xrightarrow{e}_{\mathrm{a}}\}_{e\in E_{\mathrm{a}}})$ and a concrete event system $\mathcal{E}_{\mathrm{c}} = (\mathbb{S}_{\mathrm{c}}, s^0_{\mathrm{c}}, E_{\mathrm{c}}, \{\xrightarrow{e}_{\mathrm{c}}\}_{e\in E_{\mathrm{c}}})$ we say that $\mathcal{E}_{\mathrm{c}}$ *refines* $\mathcal{E}_{\mathrm{a}}$ if there are refinement mappings $\pi_0 : \mathbb{S}_{\mathrm{c}} \to \mathbb{S}_{\mathrm{a}}$ on states and $\pi_1 : E_{\mathrm{c}} \to E_{\mathrm{a}}$ on events such that $\pi_0(s^0_{\mathrm{c}}) = s^0_{\mathrm{a}}$ and for all $s_{\mathrm{c}}, s'_{\mathrm{c}} \in \mathbb{S}_{\mathrm{c}}$ and $e_{\mathrm{c}} \in E$ such that $s_{\mathrm{c}} \xrightarrow{e_{\mathrm{c}}}_{\mathrm{c}} s'_{\mathrm{c}}$

we have $\pi_0(s_c) \xrightarrow{\pi_1(e_c)}_a \pi_0(s'_c)$. This is functional forward simulation [78] with event mappings added. Refinement preserves invariants from the abstract to the concrete model, i. e., $\mathcal{E}_a \models P$ implies that $\mathcal{E}_c \models \pi_0^{-1}(P)$, where $\pi_0^{-1}(P) = \{s \in \mathcal{S}_c \mid \pi_0(s) \in P\}$.

In our models, we often use parametrized events and states structured as records. We use the notation

$$e(\bar{x}) : \ g(\bar{x}, \bar{v}) \ \triangleright \ \bar{w} := \bar{u}(\bar{x}, \bar{v})$$

to specify such events, where $\bar{x}$ are the event's parameters (the bar representing a vector), $\bar{v}$ are the state record's fields, $g(\bar{x}, \bar{v})$ is the *guard* predicate defining the executability of the event, $\bar{w} \subseteq \bar{v}$ are the updated fields, and $\bar{u}$ are update functions (one for each variable in $\bar{w}$). This notation denotes the transition relation defined by $s \xrightarrow{e(\bar{x})} s'$ iff $g(\bar{x}, s(\bar{v}))$ holds, $s'(\bar{w}) = \bar{u}(\bar{x}, s(\bar{v}))$ and, for the state fields $\bar{z} = \bar{v} - \bar{w}$ that are not updated, $s'(\bar{z}) = s(\bar{z})$. We often use updates of parametrized channel fields holding sets of messages. For example, if m is an integer, the event $\mathbf{send}(A, B, m) : \ m > 0 \ \triangleright \ ch(A, B) \mathrel{+}= m$ adds packet m to the channel ch between A and B if m is positive, i. e., the intended update is $ch(A, B) := ch(A, B) \cup \{m\}$. Otherwise the state remains unmodified, in particular, $ch(A', B') := ch(A', B')$ for all $(A', B') \neq (A, B)$.

4.2.3 Parametrization

The generality of our models rests on their parametrization. A parametrized model may include assumptions on its parameters. An instance must define the parameters and prove the assumptions. For easy identification, we will highlight parameters in gray when they are first introduced. We also list all environment and protocol parameters in Table 4.1.

Parametrization is independent of refinement. For instance, a model can be parametrized and concrete at the same time (as is the case in our framework). In our Isabelle/HOL formalization, we implement parametrization using *locales* [11].

4.3 Abstract model

We define an event system that models the abstract data plane of a path-aware network architecture. This model includes neither cryptography nor an attacker. We prove that it satisfies path authorization and detectability.

To distinguish definitions of this abstract model from those of the concrete model that refines it (§4.4), we use subscripts 'a' and 'c', respectively.

4.3.1 Environment parameters

We model the Internet as a multigraph, where nodes represent ASes and edges represent the network links between them. More precisely, a *network topology* is a triple $(\mathscr{N}, \mathscr{I}, \mathit{target})$, where $\mathscr{N}$ is a set of nodes, $\mathscr{I}$ is a set of interfaces, and *target* is an environment parameter to our model with the type

$$\mathit{target} : \mathscr{N} \times \mathscr{I} \rightharpoonup \mathscr{N} \times \mathscr{I}. \tag{4.1}$$

This formalizes that *target* is a partial bijective function that models links between ASes.[1] We say that an interface i is *valid* for a node A, if $(A, i) \in \mathit{dom}(\mathit{target})$, whereby $\mathit{target}(A, i) = (B, j)$ denotes the node B and interface j at the other end of the link. Our definition thus allows for multiple links between a given pair of nodes, with possibly different routing policies.

We often reason directly about paths in the network, rather than the network topology. These paths are defined in terms of both nodes and their interfaces. We define a *path* to be a finite sequence of *hop information fields* from the set

$$\mathrm{HI} = (\!|\, \mathit{id} \in \mathscr{N},\ \mathit{prev} \in \mathscr{I}_\perp,\ \mathit{next} \in \mathscr{I}_\perp \,|\!). \tag{4.2}$$

Each hop information field contains the local routing information of a node, i. e., its node identifier and the interfaces that identify the links to the previous and the next hop on the path. Both interfaces are defined as option types, indicated by the subscript $\perp$. When there is no previous or next hop, we assign $\perp$ to the respective interface. The hop fields that will be introduced in the concrete model below augment the hop information fields with a cryptographic hop validation field. Since our abstract model does not contain an adversary, such authenticators are not required here.

Our model's second environment parameter is

$$\mathit{auth}_\mathrm{a} \subseteq \mathrm{HI}^*, \tag{4.3}$$

the set of *authorized paths* along which packets are allowed to travel. Packets can also traverse just a part of an authorized path. To account for such

1 The bijectivity of the *target* mapping models unicast communication (in contrast to, e.g., broadcast), however this is not required as an assumption for our proofs.

partial paths, we define $\mathit{auth}_{\mathrm{a}}^{\rightleftarrows}$, the *fragment closure* of $\mathit{auth}_{\mathrm{a}}$, as the set of paths *his* such that there exist a $\mathit{his}' \in \mathit{auth}_{\mathrm{a}}$ and paths $\mathit{his}_1, \mathit{his}_2 \in \mathrm{HI}^*$ such that $\mathit{his}' = \mathit{his}_1 \cdot \mathit{his} \cdot \mathit{his}_2$.

Note that authorized paths cannot be assumed to be actual paths in the network multigraph, since attackers in the control plane can interfere with the path construction (see §4.5.1).

Our third parameter is the set of *compromised* nodes

$$\mathcal{N}_{attr} \subseteq \mathcal{N}. \tag{4.4}$$

All other nodes are called *honest*. This environment parameter only becomes relevant after introducting the adversary in the concrete model (§4.4.3), where the attacker has access to the keys of compromised nodes. We nevertheless introduce it here, since using the same environment parameters in all of our models simplifies our presentation.

The environment assumptions (ASM) expressed over these parameters are introduced for the refinement of the abstract to the concrete model (§4.5.1).

4.3.2 State

We model packet forwarding from a node's internal network to an inter-node link, and vice-versa, using two types of asynchronous channels: *internal* (one per node) and *external* (two per interface-node pair, one in each direction). We represent these channels as sets of packets $\mathrm{PKT}_{\mathrm{a}}$, defined below. We define the state as

$$\begin{aligned}\mathbb{S}_{\mathrm{a}} = (\!|\, & \mathit{int} \in \mathcal{N} \rightarrow \mathcal{P}(\mathrm{PKT}_{\mathrm{a}}), \\ & \mathit{ext} \in \mathcal{N} \times \mathcal{I} \times \mathcal{N} \times \mathcal{I} \rightarrow \mathcal{P}(\mathrm{PKT}_{\mathrm{a}}) \,|\!).\end{aligned}$$

The initial state s_{a}^0 is the state in which all channels are empty. We overload the set inclusion operator to apply to states: A packet m is in state s, $m \in s$, iff $m \in \mathit{ran}(\mathit{int}(s)) \cup \mathit{ran}(\mathit{ext}(s))$. For a valid interface i of A with $\mathit{target}(A,i) = (B,j)$, we define $\mathit{ext}^{\mathrm{send}}(A,i) = \mathit{ext}(A,i,B,j)$ and $\mathit{ext}^{\mathrm{recv}}(A,i) = \mathit{ext}(B,j,A,i)$.

In the following definition of packets, we abstract from the payload and only model the packet-carried forwarding state:

$$\mathrm{PKT}_{\mathrm{a}} = (\!|\, \mathit{past} \in \mathrm{HI}^*,\ \mathit{fut} \in \mathrm{HI}^*,\ \mathit{hist} \in \mathrm{HI}^* \,|\!).$$

Parameter	Introduced	Description	Used to express
target	Equation (4.1)	network topology	
$auth_{\mathrm{a}}$	Equation (4.3)	authorized paths	ASM
$\mathcal{N}_{attr}$	Equation (4.4)	attacker nodes	
ψ	Equation (4.8)	crypt. check of *HVF*	COND
auth-restrict	Equation (4.9)	restr. on auth. paths	
extract	Equation (4.11)	extracts path from *HVF*	
ik_0^+	Equation (4.12)	add. attr. knowledge	

TABLE 4.1: Environment (1–3) and protocol (4–7) parameters.

dispatch-int$_{\mathrm{a}}(A, m)$:

$$fut(m) \in auth_{\mathrm{a}}^{\rightleftarrows} \wedge$$
$$hist(m) = \langle\rangle$$
$$\triangleright \; int(A) \mathrel{+}= m$$

dispatch-ext$_{\mathrm{a}}(A, i, m)$:

$$fut(m) \in auth_{\mathrm{a}}^{\rightleftarrows} \wedge$$
$$hist(m) = \langle\rangle \wedge$$
$$(A, i) \in dom(target)$$
$$\triangleright \; ext^{\mathrm{send}}(A, i) \mathrel{+}= m.$$

dispatch-int$_{\mathrm{c}}(A, m)$:

$$m \in DY(ik) \wedge$$
$$hist(m) = \langle\rangle$$
$$\triangleright \; int(A) \mathrel{+}= m.$$

dispatch-ext$_{\mathrm{c}}(A, i, m)$:

$$m \in DY(ik) \wedge$$
$$hist(m) = \langle\rangle$$
$$(A, i) \in dom(target)$$
$$\triangleright \; ext^{\mathrm{send}}(A, i) \mathrel{+}= m.$$

FIGURE 4.1: Dispatching events of the abstract (left) and concrete (right) model, with differences highlighted.

$$
\begin{array}{ll}
\multicolumn{2}{l}{\mathbf{send}_{\mathrm{a}}(A, m, hi, i) :} \\
 & hi = hd(fut(m)) \land \\
 & hi \neq \bot \land \\
 & A = id(hi) \land \\
 & i = next(hi) \land \\
 & m \in int(A) \land \\
 & (A, i) \in dom(target) \\
\triangleright & ext^{\text{send}}(A, i) \mathrel{+}= fwd_{\mathrm{a}}(m).
\end{array}
$$

$$
\begin{array}{ll}
\multicolumn{2}{l}{\mathbf{recv}_{\mathrm{a}}(A, m, hi, i) :} \\
 & hi = hd(fut(m)) \land \\
 & hi \neq \bot \land \\
 & A = id(hi) \land \\
 & m \in ext^{\text{recv}}(A, i) \land \\
 & (A, i) \in dom(target) \\
\triangleright & int(A) \mathrel{+}= m.
\end{array}
$$

$$
\begin{array}{ll}
\multicolumn{2}{l}{\mathbf{deliver}_{\mathrm{a}}(A, m, hi) :} \\
 & fut(m) = \langle hi \rangle \land \\
 & A = id(hi) \land \\
 & m \in int(A) \\
\triangleright & int(A) \mathrel{+}= fwd_{\mathrm{a}}(m).
\end{array}
$$

$$
\begin{array}{ll}
\multicolumn{2}{l}{\mathbf{send}_{\mathrm{c}}(A, m, hf, i) :} \\
 & hf = hd(fut(m)) \land \\
 & hf \neq \bot \land \\
 & A = id(hf) \land \\
 & i = next(hf) \land \\
 & m \in int(A) \land \\
 & (A, i) \in dom(target) \land \\
 & \psi(hf, hd(tl(fut(m))), tok(m)) \\
\triangleright & ext^{\text{send}}(A, i) \mathrel{+}= fwd_{\mathrm{c}}(m).
\end{array}
$$

$$
\begin{array}{ll}
\multicolumn{2}{l}{\mathbf{recv}_{\mathrm{c}}(A, m, hf, i) :} \\
 & hf = hd(fut(m)) \land \\
 & hf \neq \bot \land \\
 & A = id(hf) \land \\
 & m \in ext^{\text{recv}}(A, i) \land \\
 & (A, i) \in dom(target) \land \\
 & i = prev(hi) \land \\
 & \psi(hf, hd(tl(fut(m))), tok(m)) \\
\triangleright & int(A) \mathrel{+}= m.
\end{array}
$$

$$
\begin{array}{ll}
\multicolumn{2}{l}{\mathbf{deliver}_{\mathrm{c}}(A, m, hf) :} \\
 & fut(m) = \langle hf \rangle \land \\
 & A = id(hf) \land \\
 & m \in int(A) \land \\
 & \psi(hf, \bot, tok(m)) \land \\
\triangleright & int(A) \mathrel{+}= fwd_{\mathrm{c}}(m).
\end{array}
$$

FIGURE 4.2: Other events of the abstract (left) and concrete (right) model, with guards added in the concrete model highlighted.

A packet consists of the desired future path *fut*, and the (presumed) traversed path *past*, stored in reverse direction. The full path is $rev(past(m)) \cdot fut(m)$. While this splitting of the path simplifies our proofs, the forwarding path could equivalently be defined as a single sequence with a moving pointer indicating the current position on the path. We call a packet *m authorized*, if $fut(m) \in auth_{\mathrm{a}}^{\rightleftarrows}$. Additionally, each packet records a path *hist*, also in the reverse direction. This path represents the packet's actual trajectory and is used to express security properties. This can be seen as an auxilliary *history variable* [1], meaning that it is not part of the protocol, but serves to specify and prove properties of protocol executions.

4.3.3 Events

The events of the abstract model are given on the left-hand side of Figures 4.1 and 4.2. The life cycle of a packet is captured by the following events: **dispatch-int**$_{\mathrm{a}}$ creates a new packet containing an authorized future path in the internal channel of a node. The packet is transferred with alternating **send**$_{\mathrm{a}}$ and **recv**$_{\mathrm{a}}$ events between internal and external channels, according to the forwarding path contained in the packet. Finally, the packet is delivered to the end host with an event **deliver**$_{\mathrm{a}}$. The events **dispatch-int**$_{\mathrm{a}}$ and **deliver**$_{\mathrm{a}}$ model the interaction with end hosts, whereas **send**$_{\mathrm{a}}$ and **recv**$_{\mathrm{a}}$ represent the border routers' packet forwarding actions. The additional **dispatch-ext**$_{\mathrm{a}}$ event creates and sends a packet directly to an *ext* channel. This event is not required for normal data plane operations, but serves to introduce a malicious sender at an inter-AS link in the refinement.

We now describe these events in more detail. The **dispatch-int**$_{\mathrm{a}}$ and **dispatch-ext**$_{\mathrm{a}}$ events create an authorized packet by setting its future path to (a fragment of) an authorized path and inserting it into an internal or external channel. The history is set to the empty sequence in both events, and the past path can be set arbitrarily to allow the refinement into attacker events, where the attacker may disguise the origin of the packet. The **send**$_{\mathrm{a}}$ and **recv**$_{\mathrm{a}}$ events both use the current hop information field, i. e., the hop information field at the head of the future path, to determine where the packet should be forwarded. Hence, they require a non-empty future path. The **recv**$_{\mathrm{a}}$ event transfers a packet from the external channel at (A, i) to A's internal channel. The **send**$_{\mathrm{a}}$ event takes a packet m from the internal channel and places the transformed packet $fwd_{\mathrm{a}}(m)$ on the external channel at (A, i). The partial function $fwd_{\mathrm{a}} : \mathrm{PKT_a} \rightharpoonup \mathrm{PKT_a}$ moves the current hop information field of m into the past path and adds it to the history. It is defined for m with $fut(m) \neq \langle\rangle$ by

$$fwd_{\mathrm{a}}(m) = (\!|past = hd(fut(m))\,\#\,past(m), fut = tl(fut(m)),\\ hist = hd(fut(m))\,\#\,hist(m)|\!).$$

We define the functions head $hd : \mathrm{HI}_{\perp}^{*} \to \mathrm{HI}_{\perp}$ and tail $tl : \mathrm{HI}_{\perp}^{*} \to \mathrm{HI}_{\perp}^{*}$ by $hd(x\,\#\,xs) = x$ and $tl(x\,\#\,xs) = xs$ and by mapping $\langle\rangle$ and $\perp$ to $\perp$ in both functions.

The **deliver**$_{\mathrm{a}}$ event models delivering a packet m containing a single hop information field in its future path to an end host. However, we do not explicitly model end hosts and their state. Hence, we simply add the packet $fwd_{\mathrm{a}}(m)$ to the internal channel of the AS and thereby push the last hop information field into the *past* and *hist* paths.

4.3.4 Properties

Path authorization states that packets can only traverse the network along authorized paths. This ensures that the data plane enforces the control plane's routing policies. Formally, for all packets m in a state s, $rev(hist(m)) \in auth_{\mathrm{a}}^{\rightleftarrows}$. Recall that the order of nodes is reversed in *hist*. We strengthen this to an inductive invariant by adding the future path:

$$\forall m \in s.\ rev(hist(m)) \cdot fut(m) \in auth_{\mathrm{a}}^{\rightleftarrows}. \tag{4.5}$$

The proof in this abstract model is straightforward. New packets are required to have an authorized future path and an empty history. For existing packets, $rev(hist(m)) \cdot fut(m)$ remains invariant during their forwarding. The *past* path is irrelevant for path authorization.

We furthermore formalize *detectability* and show that it is an invariant: all traversed hops are recorded on (i.e., a prefix of) the past path:

$$\forall m \in s.\ hist(m) \leq past(m). \tag{4.6}$$

This property is independent of $auth_{\mathrm{a}}$ and follows directly from the events' definitions. Our presentation will focus on path authorization, as it is the data plane's central security property.

4.4 Concrete model

We refine the abstract forwarding protocol into a concrete model. In this model, the packets' hop fields include (generic) cryptographic hop validation fields to secure the authorized paths against a Dolev–Yao attacker (§4.4.3). We present the concrete model's events in §4.4.4 and the refinement in §4.5.

The concrete model retains the environment parameters of the abstract model (§4.3.1), and adds four *protocol parameters*, which we introduce below. One of them is the cryptographic check that ASes apply to their hop validation fields, which allows us to abstract from the concrete cryptographic mechanism used. We focus on the setting for directed path authorization here, and defer the treatment of undirected path authorization to §4.7.3. We compare both classes of protocols in §4.8.1.

4.4.1 Cryptographic terms, hop fields, packets and states

We introduce an algebra $\mathbb{T}$ of cryptographic terms:

$$\mathbb{T} = \mathscr{N} \mid \mathscr{I}_\perp \mid \mathbb{N} \mid K_{\mathscr{N}} \mid \langle \mathbb{T},\ \mathbb{T},\ \ldots,\ \mathbb{T} \rangle \mid \mathsf{H}(\mathbb{T}).$$

Terms consist of node identifiers, interfaces, natural numbers (e. g., for timestamps), keys (one per node), as well as finite sequences, and cryptographic hashes of terms. We define message authentication codes (MACs) using hashing by $\mathsf{MAC}_k(m) = \mathsf{H}(\langle k, m \rangle)$. Our framework also supports encryption and signatures, which we do not use here.

Hop fields (HF), used in the concrete model, extend the hop information fields (HI), used in the abstract model, with a *hop validation field* (*HVF*). This is a cryptographic authenticator that authenticates the hop information:

$$\mathrm{HF} = (\!|\ id \in \mathscr{N},\ prev \in \mathscr{I}_\perp,\ next \in \mathscr{I}_\perp,\ HVF \in \mathbb{T}\ |\!). \tag{4.7}$$

In the concrete model, *path* refers to a sequences of HFs. We define the function *abstr-hf*: HF $\to$ HI projecting concrete hop fields to abstract hop information fields by dropping *HVF* and we lift it element-wise to paths. To keep our notation compact, we write $\overline{hf_A}$ and $\overline{hfs}$ to denote the application of *abstr-hf* to hop fields and sequences of hop fields.

We next define concrete packets as follows:

$$\mathrm{PKT_c} = (\!|\ tok \in \mathbb{T},\ past \in \mathrm{HF}^*,\ fut \in \mathrm{HF}^*,\ hist \in \mathrm{HI}^*\ |\!).$$

The past and future paths are sequences of concrete hop fields, while the history remains a sequence of HI fields. Concrete packets contain an additional *packet token field* (*tok*), which is used by instances for various purposes, for instance as a source-supplied unique packet identifier.

The concrete state space $\mathbb{S}_c$ has the same record structure as the abstract $\mathbb{S}_a$, but the channels now carry concrete packets:

$$\mathbb{S}_c = (\!|\, int \in \mathscr{N} \to \mathcal{P}(\mathrm{PKT_c}),\\ ext \in \mathscr{N} \times \mathscr{I} \times \mathscr{N} \times \mathscr{I} \to \mathcal{P}(\mathrm{PKT_c}) \,|\!).$$

The initial state s_c^0 is defined similarly to s_a^0 as the empty channel state.

Auxiliary functions

We define the overloaded function *terms* for hop fields (*terms* : HF $\to \mathcal{P}(\mathbb{T})$), paths (*terms* : HF* $\to \mathcal{P}(\mathbb{T})$) and packets (*terms* : PKT$_c$ $\to \mathcal{P}(\mathbb{T})$) as follows:

$$\begin{aligned} terms(hf) &= \{HVF(hf)\} \\ terms(hfs) &= \bigcup_{hf \in hfs} terms(hf) \\ terms(pkt) &= \{tok(pkt)\} \cup terms(past(pkt)) \cup terms(fut(pkt)). \end{aligned}$$

For a set T of terms and for a hop field, path, or packet x, we write $x \in T$ for $terms(x) \subseteq T$.

We define a function *hi-term* : HI $\to \mathbb{T}$ that maps hop information fields to terms as

$$hi\text{-}term(hi) = \langle id(hi),\ prev(hi),\ next(hi)\rangle.$$

We leave the conversion of HI fields to terms via this function implicit below.

4.4.2 Protocol parameters and authorized paths

We define four protocol parameters. The first is a *cryptographic validation check*

$$\psi : \mathrm{HF} \times \mathrm{HF}_\perp \times \mathbb{T} \to \mathbb{B}, \tag{4.8}$$

which each border router performs to check the validity of its hop field. This parameter abstracts the cryptographic structure of the hop validation field, which is only determined in concrete protocol instances. Here, $\psi(hf_A, hf_B, u)$ holds iff the hop field hf_A is valid given the next hop field hf_B (if any, and $\perp$ otherwise) and the packet's *tok* field u.

We also define a function Ψ : HF* $\times\ \mathbb{T} \to$ HF*, which we apply to the future path of a packet to obtain the longest prefix of *hfs* such that for every hop field hf_A on the path, and its successor hop field hf_B ($\perp$, if none exists) and the *tok* field u, $\psi(hf_A, hf_B, u)$ holds:

$$\begin{aligned} \Psi(hf_A \# hf_B \# hfs, u) &= hf_A \# \Psi(hf_B \# hfs, u) && \text{if } \psi(hf_A, hf_B, u) \\ \Psi(\langle hf_A \rangle, u) &= \langle hf_A \rangle && \text{if } \psi(hf_A, \perp, u) \\ \Psi(hfs, u) &= \langle\rangle && \text{otherwise.} \end{aligned}$$

We use this function in the mapping of future paths from the concrete model to the abstract model (§4.5.3) to truncate the path at the first invalid hop field. This does not reduce the system's possible behavior, since forwarding is performed by honest agents that do not forward packets along invalid hop fields. We call a path *hfs* or a packet *pkt* with $fut(pkt) = hfs$ *cryptographically valid (for u)* if $\Psi(hfs, u) = hfs$.

Instances restrict the set of concrete authorized paths to incorporate additional constraints that the respective control plane guarantees. The second parameter of our model is a predicate

$$auth\text{-}restrict : \mathrm{HF}^* \times \mathbb{T} \to \mathbb{B}, \tag{4.9}$$

which decides if a given concrete path and *tok* field satisfy these constraints. Note that we cannot simply incorporate these constraints into the local check $\psi(hf_A, hf_B, u)$ performed by routers. First, because these may be control plane assumptions rather than checks performed by routers. Second, they are required to limit the intruder knowledge in ways that ψ cannot. In the next section we define that the intruder learns a value u if there is an authorized paths *hfs* with $\Psi(hfs, u) = hfs$. Without restriction, this would allow the attacker to learn arbitrary terms, since $hfs = \langle\rangle$ is authorized and valid for any u value.

We define the set of concrete authorized paths, $auth_c \in \mathbb{T} \to \mathcal{P}(\mathrm{HF}^*)$, as the set of paths *hfs* that are cryptographically valid for a *tok* field u, satisfy the restriction and whose projection to HI^* is authorized:

$$auth_c(u) = \{hfs \mid \Psi(hfs, u) = hfs \wedge auth\text{-}restrict(hfs, u) \wedge \overline{hfs} \in auth_a\}. \tag{4.10}$$

We overload $auth_c$ and define the set $auth_c \subseteq \mathrm{HF}^*$ as the union of $auth_c(u)$ over all u. Similar to the abstract model, a concrete packet m is authorized if $fut(m)$ is a fragment of an authorized path, i.e., $fut(m) \in auth_c^{\rightleftarrows}$.

To achieve path authorization, protocols use the *HVF* to protect the future (abstract) path. The third protocol parameter is

$$extract : \mathbb{T} \to \mathrm{HI}^*, \tag{4.11}$$

which is intended to extract this path from a given *HVF*. For instance, in SCION, the hop validation field consists of a MAC over the hop's *id* and interfaces and, the next hop's *HVF*, allowing for a recursive extraction. This function is only required in proofs and not in the definition of the event system. Hence it may use features that would be infeasible to implement in the actual system, such as inverting hashes and MACs.

We lift *extract* to hop fields by $extract(hf) = extract(HVF(hf))$ and to paths by defining $extract(\langle\rangle) = \langle\rangle$ and $extract(hf\#hfs) = extract(hf)$. In §4.5.2, we will define a consistency condition (to be discharged by each instance model) that implies that *extract* coincides with $\overline{\cdot}$ on those paths that are both cryptographically valid and derivable by the attacker.

The fourth protocol parameter is a set of cryptographic terms

$$ik_0^+ \subseteq \mathbb{T}. \tag{4.12}$$

It allows protocol instances to give the attacker additional terms and is used in the definition of the intruder knowledge below.

ik_0^+ and *auth-restrict* are parameters that allow modeling the protocol-dependent intruder knowledge. While they are important in the instance models, they do not play an important role for the refinement proof in the concrete model.

4.4.3 Attacker model

In §3.2.1, we proposed a Dolev–Yao adversary who is restricted to sending and receiving packets at compromised locations. For path authorization, this attacker can be strengthened. We allow the attacker to eavesdrop on and inject new packets in all *int* and *ext* channels, but has access only to the keys of compromised nodes. We first define the attacker's message derivation capabilities, which are used in the attacker events introduced in §4.4.4.

As usual, we model the attacker's knowledge as a set of terms and her message derivation capabilities as a closure operator $DY : \mathcal{P}(\mathbb{T}) \to \mathcal{P}(\mathbb{T})$ on sets of terms. Our formalization of DY is based on Paulson [87] and defines $DY(H) = DY^{\uparrow}(DY^{\downarrow}(H))$ for a set of terms H as the composition of two closure operators defined by the rules in Figure 4.3. The decomposition closure $DY^{\downarrow}(H)$ closes H under the projection of sequences to their elements and the composition closure $DY^{\uparrow}(H)$ includes all public terms (i. e., node identifiers, interfaces, and numbers) and closes H under the construction of sequences and hashes.

$$\frac{t \in H}{t \in DY^{\downarrow}(H)} \qquad \frac{\langle t_1, \ldots, t_n \rangle \in DY^{\downarrow}(H)}{t_i \in DY^{\downarrow}(H)}\ 1 \leq i \leq n$$

$$\frac{t \in H}{t \in DY^{\uparrow}(H)} \qquad \frac{t \in \mathscr{N} \cup \mathscr{I}_{\perp} \cup \mathbb{N}}{t \in DY^{\uparrow}(H)}$$

$$\frac{t \in DY^{\uparrow}(H)}{\mathsf{H}(t) \in DY^{\uparrow}(H)} \qquad \frac{t_1 \in DY^{\uparrow}(H) \ \cdots \ t_n \in DY^{\uparrow}(H)}{\langle t_1, \ldots, t_n \rangle \in DY^{\uparrow}(H)}$$

FIGURE 4.3: Rules for Dolev–Yao message decomposition ($DY^{\downarrow}$) and composition ($DY^{\uparrow}$).

We define the intruder knowledge in a state $s \in \mathbb{S}_c$ as the Dolev–Yao closure (DY) of the set of terms $ik(s)$, defined by

$$ik_0 = \bigcup \{terms(x) \cup \{u\} \mid x \in auth_c(u)\} \cup \{K_i \mid i \in \mathscr{N}_{attr}\}, \qquad (4.13)$$

$$ik(s) = ik_0 \cup ik_0^+ \cup \bigcup_{m \in s} terms(m). \qquad (4.14)$$

The set $ik(s)$ is the union of the initial intruder knowledge ik_0, additional terms ik_0^+, and all terms in packets of state s. The set ik_0 consists of authorized paths (the *HVF* of their hop fields and the *tok* field with which they are valid) and compromised nodes' keys.

4.4.4 Events

Each event of the abstract model is refined into a similar event of the concrete model (Figures 4.1 and 4.2, right, where differences between the models are highlighted in yellow). In the events' guards we omit the state and just write *ik*. The concrete model retains the packet life-cycle of the abstract model (§4.3.3). The **dispatch-int**$_c$ and **dispatch-ext**$_c$ events can send arbitrary attacker-derivable packets, instead of only authorized packets as in the abstract model. To defend against the attacker, we introduce interface and cryptographic checks in **send**$_c$, **recv**$_c$, and **deliver**$_c$. We now discuss the concrete model's events in more detail.

Attacker events

The two attacker events **dispatch-int**$_c$ and **dispatch-ext**$_c$ model that the attacker is active: she has the capability to send a packet on any AS' internal or external channel, regardless of whether the AS is honest or compromised. In both events, the packet m created by the attacker may contain arbitrary past and future paths, but its hop validation fields and packet token field must be derivable from the intruder knowledge, i. e., $terms(m) \subseteq DY(ik(s))$. Note that the event **dispatch-int**$_c$ still covers honest senders, as the attacker knows all authorized paths.

Similar to their abstract counterparts, both events set the history *hist* to $\langle\rangle$. The motivation for this is to exclude attacks where an attacker modifies a packet's forwarding path en-route, since these attacks are unavoidable in the presence of a sufficiently strong on-path adversary. For example, consider Figure 2.1a (page 16) and suppose that the attacker has access to D's external channels. Then D may receive a packet arriving on the left path from F, exchange its forwarding path by the right path, and forward the modified packet to C. This would (trivially) violate path authorization. By resetting the history, we effectively consider all packets sent by the attacker as new ones. An additional reason for this modeling choice is that an on-path attacker can not only re-route packets, but also modify their contents arbitrarily. This makes it generally impossible to correlate packets sent by the attacker with those the attacker has previously received. Consequently, path authorization must hold separately for the packets before and after the replacement of the forwarding path by the attacker.

Note that in the **dispatch-ext**$_c$ event, the attacker's hop information field is not recorded in the history. This is because the attacker could modify her own hop field in arbitrary ways, and even omit it entirely. The attacker node is still identifiable in the history via the *target* function because the interface identifier *prev* of the next hop points to the link between it and the source AS of the packet.

Honest events

The honest events are **send**$_c$, **recv**$_c$, and **deliver**$_c$. To secure the protocol against the attacker introduced in this model, border routers now perform two validation checks. First, upon receiving a packet from another node, **recv**$_c$ includes the guard $i = prev(hf)$ to check that interface i over which the packet is received matches the interface *prev* of the packet's current hop field *hf*. Second, all honest events check the cryptographic *hop validation field* that is added to hop fields in this refinement using the check $\psi(hf, hf', u)$, where hf, hf', and u are the packet's current hop field, next hop field, and *tok* field, respectively. This check ensures that the hop field (and indeed the whole or partial path) is authorized.

The events **send**$_{\mathrm{c}}$ and **deliver**$_{\mathrm{c}}$ use the function fwd_{c} to forward a packet:

$$fwd_{\mathrm{c}}(m) = (\!|tok = tok(m),\, past = hd(fut(m))\,\#\,past(m),\, fut = tl(fut(m)),$$
$$hist = \overline{hd(fut(m))}\,\#\,hist(m)|\!).$$

This function is defined similarly to fwd_{a}, but the *tok* field is not modified and the hop field being moved from the future to the past path is converted from HF to HI using $\overline{\cdot}$ before it is added to $hist(m)$.

Constant intruder knowledge

In reachable states s, all packets $m \in s$ are derivable from the static intruder knowledge $ik_0 \cup ik_0^+$. Hence, the attacker does not learn any new messages during the protocol execution. Under the Dolev–Yao closure, we can hence drop the state-dependent part $\bigcup_{m \in s} terms(m)$ of the intruder knowledge and use $DY(ik_0 \cup ik_0^+)$ instead of $DY(ik(s))$. We show the following lemma as an invariant.

Lemma 1: Constant intruder knowledge

For all reachable states s, $DY(ik(s)) = DY(ik_0 \cup ik_0^+)$.

Proof. Invariant proof. By definitions of events. □

4.5 Refinement

We prove the following refinement theorem for the abstract event system $\mathcal{E}_{\mathrm{a}} = (\mathbb{S}_{\mathrm{a}}, s^0_{\mathrm{a}}, E_{\mathrm{a}}, \{\xrightarrow{e}_{\mathrm{a}}\}_{e \in E_{\mathrm{a}}})$ and the concrete event system $\mathcal{E}_{\mathrm{c}} = (\mathbb{S}_{\mathrm{c}}, s^0_{\mathrm{c}}, E_{\mathrm{c}}, \{\xrightarrow{e}_{\mathrm{c}}\}_{e \in E_{\mathrm{c}}})$, where for $i \in \{\mathrm{a},\ \mathrm{c}\}$, $\mathbb{S}_{\mathrm{i}}$ and s^0_{i} are as defined in the previous sections,

$$E_i = \{\textbf{dispatch-int}_{\mathrm{i}},\ \textbf{dispatch-ext}_{\mathrm{i}},\ \textbf{send}_{\mathrm{i}},\ \textbf{recv}_{\mathrm{i}},\ \textbf{deliver}_{\mathrm{i}}\},$$

and $\xrightarrow{e}_i$ for all $e \in E_i$ as in Figures 4.1 and 4.2. The assumptions (ASM), and conditions (COND), as well as the refinement mappings π_0 and π_1 will be defined in this section.

Theorem 1: Concrete model refines abstract model

$\mathcal{E}_\mathrm{c}$ refines $\mathcal{E}_\mathrm{a}$ under the refinement mappings $\pi_0 \colon \mathbb{S}_\mathrm{c} \to \mathbb{S}_\mathrm{a}$ on states and $\pi_1 \colon E_\mathrm{c} \to E_\mathrm{a}$ on events, assuming ASM and COND.

The refinement proof rests on several global *assumptions* (ASM) about the control plane and on a set of *conditions* (COND) about the authentication mechanism used. To establish that a concrete protocol satisfies the path authorization and detectability properties, it suffices to define the protocol's authentication mechanism by instantiating the protocol parameters and discharge the associated conditions, which we do for several protocols in §4.6.

In this section, we first introduce the assumptions and conditions. Afterwards we define the state and event refinement mappings. Finally, we show the interesting cases in the refinement of the attacker event, which is the crux of Theorem 1. The full refinement proof, formalized in Isabelle/HOL, is accessible in the supplementary material.

4.5.1 Control plane assumptions

We define environment assumptions about the authorized paths $auth_\mathrm{a}$ constructed by the control plane. There are two types of assumptions. First, there are two assumptions about the correct functioning of the control plane, which is independent of the data plane. Second, additional assumptions close the set of authorized paths in order to exclude trivial attacks on the routing policies of colluding ASes. These provide upper and lower bounds on the set $auth_\mathrm{a}$, respectively.

Correctness assumptions

The first control plane assumption is that authorized paths are *terminated*: the first hop information field's *prev* is $\bot$ and the last hop information field's *next* is $\bot$, except for when the respective hop information field belongs to an attacker. Second, we assume that authorized paths are *interface-valid*:

interfaces of adjacent hop information fields on a path point to the same link, except for when both hop fields belong to attacker nodes. This exception accounts for unavoidable out-of-band communication by adversaries, so-called *wormholes* [54].

To formalize interface validity, we introduce the interface validity predicate

$$\phi : \mathrm{HI} \times \mathrm{HI}_{\perp} \rightarrow \mathbb{B}.$$

In the following, we let hi_A (respectively hi_B) denote a hop information field for which $id(hi_A) = A$ (resp. $id(hi_B) = B$). We also use this shorthand for hop fields hf_A. The parameters of ϕ are the current hop information field hi_B and a preceding hop information field hi_A. If there is no previous hop field, no interface must be checked.

$$\begin{aligned} \phi(hi_B, \perp) &= \mathsf{true} \\ \phi(hi_B, hi_A) &= (target(A, next(hi_A)) = (B, prev(hi_B))) \\ &\quad \vee (A \in \mathcal{N}_{attr} \wedge B \in \mathcal{N}_{attr}) \end{aligned}$$

We define a function $\Phi : \mathrm{HI}^* \times \mathrm{HI}_{\perp} \rightarrow \mathrm{HI}^*$ as follows. For the longest prefix of a sequence of hop information fields *his* and the initial previous hop information field hi_{prev}, $\Phi(his, hi_{prev})$ returns the longest prefix of *his* such for all fields hi_B on *his* and their respective predecessor hi_A on *his*, $\phi(hi_B, hi_A)$ holds. For the first hop information field on *his*, ϕ must hold with hi_{prev} as the predecessor. We write $\Phi(his)$ as a shorthand for $\Phi(his, \perp)$.

Formally, we define

$$\begin{aligned} \Phi(\langle\rangle, hi_{prev}) &= \langle\rangle \\ \Phi(hi\#his, hi_{prev}) &= hi\#\Phi(his, hi) && \text{if } \phi(hi, hi_{prev}) \\ \Phi(hi\#his, hi_{prev}) &= \langle\rangle && \text{otherwise.} \end{aligned}$$

ASM 1 and ASM 2 formalize the correctness of the control plane.

ASM 1: Terminated
A hop information field hi with $id(hi) \notin \mathcal{N}_{attr}$ on $hi\#his \in auth_{\mathrm{a}}$ (resp. on $his \cdot \langle hi \rangle \in auth_{\mathrm{a}}$) has $prev(hi) = \perp$ (resp. $next(hi) = \perp$).

ASM 2: Interfaces valid
All paths $his \in auth_{\mathrm{a}}$ are interface-valid, namely $\Phi(his) = his$.

Closure assumptions

We have chosen a strong attacker model, in which both end hosts sending packets and ASes (including on-path ASes) can be compromised. As a consequence, we need to make assumptions on the attacker's behavior in the control plane. Concretely, we assume that if all *honest* on-path ASes consent to the authorization of a path (and create corresponding hop validation fields), then the compromised on-path ASes consents as well. This is necessary because in the data plane the attacker can craft a corresponding forwarding path using the honest nodes' hop validation fields. It is also acceptable to regard such paths as being authorized, since all honest agents consented to their authorization. In the extreme case in which all ASes are compromised, all paths must be assumed to be authorized as well.

Formally, we assume that the set of authorized paths is closed under certain path modifications. In this section, we only present path modifications possible in directed protocols. The closure assumptions for undirected protocols are given in §4.7.3.

For example, assume that ASes E and F in Figure 2.1 (page 16) are compromised. Since E is on the right path G–E–D–C–A, she can take the suffix E–D–C–A, change her own hop field (and issue a new *HVF* using the compromised key) such that *prev* points to F, and prepend a new hop field for F to obtain the path F–E–D–C–A. None of these changes require consent from the other on-path nodes, since each AS only decides on the authorization of the partial paths with its respective AS at the beginning and path extensions are implicitly authorized. Hence, this does not impact any honest AS' policy. We accept such path modifications as authorized in ASM 3–ASM 6.

ASM 3: Empty & Single
$\langle\rangle \in \mathit{auth}_{\mathrm{a}}$ and $\langle hi \rangle \in \mathit{auth}_{\mathrm{a}}$ for all $id(hi) \in \mathcal{N}_{attr}$.

ASM 4: Prepend
If the first hop information field belongs to the attacker, she can prepend another attacker hop information field. Formally, if $hi_B \# his \in \mathit{auth}_{\mathrm{a}}$, $B \in \mathcal{N}_{attr}$, and $A \in \mathcal{N}_{attr}$ then $hi_A \# hi_B \# his \in \mathit{auth}_{\mathrm{a}}$.

ASM 5: Suffix
The attacker can take a path's *suffix* if her hop information field is the suffix' head. Formally, if $his' \cdot hi_A \# his \in \mathit{auth}_{\mathrm{a}}$ and $A \in \mathcal{N}_{attr}$ then $hi_A \# his \in \mathit{auth}_{\mathrm{a}}$.

ASM 6: Modify
If the first HI field belongs to the attacker, she can modify its *prev*.

Formally, if $hi_A \# his \in auth_a$, $next(hi'_A) = next(hi_A)$, and $A \in \mathcal{N}_{attr}$ then $hi'_A \# his \in auth_a$. Note that $id(hi'_A) = id(hi_A) = A$.

These are not merely assumptions of our protocol model but are inherent to the path authorization mechanism of directed protocols. Undirected protocols (such as ICING) require that the entire path is authorized by each on-path AS. As we show in §4.7.3, ASM 3–ASM 6 can then be replaced by weaker assumptions. Note also that the assumptions are only required since we assume a very strong attacker model in which the end host attacker colludes with on-path ASes. This is in stark contrast to BGP, which does not achieve security even when there are only off-path attackers. When all compromised ASes are off-path, then in our model of SCION, the above closure assumptions are not needed.

With the assumption that the empty path is authorized (ASM 3), the importance of *auth-restrict* in the definition of $auth_c$ becomes apparent. $\Psi(\langle\rangle, u) = \langle\rangle$ holds for all values of u. Hence, without adding a restriction, $\langle\rangle \in auth_c(u)$ would hold for all u, giving the attacker *all* terms, including keys, via ik_0 (cf. Equation (4.13)).

4.5.2 Conditions on authentication mechanisms

We define five conditions that relate the protocol parameters ψ, *auth-restrict*, *extract*, and ik_0^+ introduced in Equations (4.8), (4.9), (4.11), and (4.12) with each other and with the environment parameters $\mathcal{N}_{attr}$ and $auth_a$ (via $auth_c$). These conditions are used in the refinement proof in §4.5.4. We will have to prove these conditions for any instance of the concrete model.

COND 1 and COND 2 together require that the attacker cannot derive valid hop fields for honest nodes that are not already contained in $auth_c$. They also constrain the parameter ik_0^+, such that instances cannot provide the attacker with terms that allow her to create valid but unauthorized hop fields.

COND 1: Attacker knowledge derivation:
$hf \in DY(ik_0 \cup ik_0^+)$, $\psi(hf, hf', u)$, and $id(hf) \notin \mathcal{N}_{attr}$ imply $hf \in DY^{\downarrow}(ik_0 \cup ik_0^+)$.

COND 2: Attacker knowledge decomposition:
$hf \in DY^{\downarrow}(ik_0 \cup ik_0^+)$ and $\psi(hf, hf', u)$ imply $\exists hfs \in auth_c.\ hf \in hfs$.

Note that all valid hop fields that belong to attacker-controlled nodes and are derivable using her keys are already contained in the set of authorized paths by ASM 3–ASM 6, hence COND 1 does not need to cover these hop fields.

COND 3 and COND 4 relate $\Psi(hfs, u)$, the longest cryptographically valid prefix of *hfs*, to $extract(hfs)$, which extracts the subsequent path from the first hop field in *hfs*. In particular, on a cryptographically valid path they coincide (modulo the projection to abstract paths, $\overline{\cdot}$). For instance, consider the SCION path given in Figure 3.1. hvf_D is

$$\mathsf{MAC}_{K_D}(\langle hi_D, \mathsf{MAC}_{K_B}(\langle hi_B, \mathsf{MAC}_{K_A}(hi_A)\rangle)\rangle)\ .$$

In this instance *extract* would be defined to extract the forwarding data from the nested MACs, i.e., $extract(hvf_D) = \langle hi_D, hi_B, hi_A\rangle$. This is exactly the HI-level path in Figure 3.1 of D and the following ASes.

COND 3: Path prefix of extract: $\overline{\Psi(hfs, u)} \leq extract(hfs)$.

COND 4: Extract prefix of path:
If $\Psi(hfs, u) = hfs$ and $auth\text{-}restrict(hfs, u)$, then $extract(hfs) \leq \overline{hfs}$.

Finally, COND 5 requires the *HVF* to protect the *tok* field. This ensures that a hop field that is valid for a certain value for *tok* cannot be used to forward a packet with a different value.

COND 5: *tok* protected:
$\psi(hf, hf', u)$ and $\psi(hf, hf'', u')$ imply $u' = u$.

4.5.3 Refinement mappings

We define the refinement mapping $\pi_0 \colon \mathbb{S}_c \to \mathbb{S}_a$ on states as the element-wise mapping of the *int* and *ext* channels under a function that maps concrete packets to abstract packets. We define $to_a : \mathrm{PKT_c} \times \mathrm{HI}_\perp \to \mathrm{PKT_a}$ by

$$\begin{aligned} to_a(m, hi_{prev}) = (\!|\ & past = \overline{past(m)}, \\ & fut = \Phi(\overline{\Psi(fut(m), tok(m))},\ hi_{prev}), \\ & hist = hist(m)\ |\!). \end{aligned}$$

Because of the interface and cryptographic checks that we introduce in the concrete model, no forwarding occurs on invalid hop fields, and they *may* be safely truncated. Since the abstract model does not have such checks, the abstraction function *must* truncate them in order to establish a refinement relation.

For *int* channels, we map all concrete packets m to abstract packets $to_{\mathrm{a}}(m, \perp)$. We set $hi_{prev} = \perp$ since the interface of the first hop information field does not need to be checked against a preceding hop information field. For packets in *ext* channels (A, i, B, j), we need to check that the first hop information field is interface-valid with the channel that carries the packet (i.e., $id = B$, $prev = j$). We do so by giving the parameter $hi_{prev} = (\!|\, id = A,\ prev = \perp,\ next = i \,|\!)$ in the to_{a} mapping of each packet. Hence, if the first hop information field does not have the correct interface and *id*, then the packet's future path is mapped to $\langle\rangle$.

The refinement mapping $\pi_1 \colon E_{\mathrm{c}} \to E_{\mathrm{a}}$ maps each event on the right side of Figures 4.1 and 4.2 to the corresponding event on the left side, where packet and hop field parameters are transformed using to_{a} and $\overline{\cdot}$, e. g. $\mathbf{send}_{\mathrm{c}}(A, m, hf, i)$ is mapped to $\mathbf{send}_{\mathrm{a}}(A, to_{\mathrm{a}}(m), \overline{hf}, i)$.

4.5.4 Refinement proof

In the refinement proof, we first show that the concrete initial state maps to the abstract initial state under π_0, i.e., $\pi_0(s^0_{\mathrm{c}}) = s^0_{\mathrm{a}}$. This holds trivially, since in both event systems the initial state does not contain any packets.

We then show that each concrete event e can be simulated by its abstract counterpart $\pi_1(e)$. This is straightforward for the honest events, since the concrete model only adds guards and the concrete guards imply the validity of the first hop field (ensuring that *fut* is not mapped to $\langle\rangle$ under to_{a}). The state updates of these events preserve the refinement relation. The difficult cases are the attacker events. In particular, we must show that the concrete dispatch events' guards imply their abstract counterparts. This is formalized as Theorem 2 below, stating that the attacker can only derive paths that, restricted to their valid prefix, are authorized.

To improve readability, we will often omit the parameter *tok* from ψ, *auth-restrict*, and Ψ.

Lemmas

We first prove two lemmas that are helpful for the attacker refinement proof below. The first lemma states that the extraction of a cryptographically valid path is the path itself (modulo $\overline{\cdot}$).

Lemma 2: Extract is path for valid paths

If $\Psi(hfs) = hfs$ and $auth\text{-}restrict(hfs)$, then $extract(hfs) = \overline{hfs}$.

Proof. By COND 3 and COND 4. □

The second lemma asserts that the valid prefix of any extension of an attacker-extractable hop field is authorized.

Lemma 3: Valid prefix of extension of derivable *hf* is authorized

Suppose $hf \in DY^{\downarrow}(ik_0 \cup ik_0^+)$ for some hop field hf. Then $\overline{\Psi(hf \# hfs)} \in auth_a^{\rightleftarrows}$ for all paths hfs.

Proof. If hf is invalid, i. e., $\neg\psi(hf, hd(hfs))$, then $\Psi(hf \# hfs) = \langle\rangle$ and the conclusion holds trivially.

Otherwise, we can apply COND 2 and obtain hfs', hfs_0, hfs_1, and hfs_2 such that $hfs' \in auth_c$, $hfs' = hfs_0 \cdot hfs_1$, and $hfs_1 = hf \# hfs_2$. Since $hfs' \in auth_c$, $\Psi(hfs') = hfs'$ and thus also $\Psi(hfs_1) = hfs_1$. Then we can apply Lemma 2 and obtain $extract(hfs_1) = extract(hf) = \overline{hfs_1}$. Since hfs_1 is a suffix of the authorized path hfs', we have $extract(hf) \in auth_a^{\rightleftarrows}$.

Finally, from COND 3, we have $\overline{\Psi(hf \# hfs)} \leq extract(hf)$. Since $auth_a^{\rightleftarrows}$ is closed under prefixing, $\overline{\Psi(hf \# hfs)} \in auth_a^{\rightleftarrows}$. □

Attacker refinement proof

Theorem 2: Attacker Refinement

If a packet $m \in DY(ik_0 \cup ik_0^+)$ then $\Phi(\overline{\Psi(fut(m))},\ hi_{prev}) \in auth_a^{\rightleftarrows}$.

Proof. We prove this theorem by induction over $hfs = fut(m)$. Here, we only sketch the proof and focus on the interesting cases in which at least two hop fields are left, i. e., $hfs = hf_A \# hf_B \# hfs'$, and both have valid hop validation fields and interfaces. Recall that the subscript identifies the node; i. e., we use hf_A to denote a hop field for which $id(hf_A) = A$ holds.

- $A \notin \mathcal{N}_{attr}$: If the attacker can derive hf_A without K_A, then by COND 1 $\overline{hf_A}$ must already be in $DY^{\downarrow}(ik_0 \cup ik_0^{+})$. Then by Lemma 3, we have $\overline{\Psi(hfs)} \in auth_{\mathrm{a}}^{\rightleftarrows}$ and by fragment closure also $\Phi(\overline{\Psi(hfs)}, hi_{prev}) \in auth_{\mathrm{a}}^{\rightleftarrows}$ as required.

- $A \in \mathcal{N}_{attr}$ and $B \notin \mathcal{N}_{attr}$: This is the most difficult case. By COND 1, $hf_B \in DY^{\downarrow}(ik_0 \cup ik_0^{+})$, and by COND 2, we obtain hfs_{gen} such that $hfs_{gen} \in auth_{\mathrm{c}}$ and $hf_B \in hfs_{gen}$. Paths in $auth_{\mathrm{a}}$ and, by extension, paths in $auth_{\mathrm{c}}$ are terminated (ASM 1). However, by the case assumption, $\overline{hf_B}$ is interface-valid with $\overline{hf_A}$ as the preceding AS and thus cannot be terminated. Hence, there must be a hf' preceding hf_B on hfs_{gen}. As $hfs_{gen} \in auth_{\mathrm{c}}$, there exist hfs_{pre} and hfs_{post} such that

$$hfs_{gen} = hfs_{pre} \cdot hf' \# hf_B \# hfs_{post} \in auth_{\mathrm{c}}.$$

  Since $hf' \in DY^{\downarrow}(ik_0 \cup ik_0^{+})$, we can apply Lemma 3 and have

$$\overline{\Psi(hf' \# hf_B \# hfs')} \in auth_{\mathrm{a}}^{\rightleftarrows}.$$

  Also, as hop fields in $auth_{\mathrm{c}}$ are valid, $\psi(hf', hf_B)$ Hence for some his'_{pre} and his'_{post}

$$his'_{pre} \cdot \overline{(hf' \# \Psi(hf_B \# hfs'))} \cdot his'_{post} \in auth_{\mathrm{a}}.$$

  Finally, we use the assumptions on authorized paths to show that the attacker can remove the hop information fields his'_{pre} preceding $\overline{hf'}$ (ASM 5) and swap out $\overline{hf'}$ for $\overline{hf_A}$ (ASM 6). To apply these assumptions, we must show that $id(\overline{hf'}) \in \mathcal{N}_{attr}$ and, respectively, that $\overline{hf'}$ and $\overline{hf_A}$ have the same id and $next$. $\overline{hf_B}$ is interface-valid with the predecessor $\overline{hf_A}$ (by the case assumption) and with the predecessor $\overline{hf'}$ (by the assumption on the interface-validity of authorized paths, ASM 2). Thus $\overline{hf_A}$ and $\overline{hf'}$ must have the same AS identifier $id(\overline{hf'}) = A \in \mathcal{N}_{attr}$ and interface $next(\overline{hf'}) = next(\overline{hf_A})$. Hence, we have $\overline{\Psi(hfs)} \in auth_{\mathrm{a}}^{\rightleftarrows}$ and by fragment closure also $\Phi(\overline{\Psi(hfs)}, hi_{prev}) \in auth_{\mathrm{a}}^{\rightleftarrows}$.

- $A \in \mathcal{N}_{attr}$ and $B \in \mathcal{N}_{attr}$: This case uses the suffix and prepend assumptions on authorized paths of §4.5.1. By the induction hypothesis

$$\Phi(\overline{\Psi(hf_B \# hfs')},\ hi_{prev}) \in auth_{\mathrm{a}}^{\rightleftarrows}.$$

  By case assumption of the validity of hf_B, there is a his_{pre}, his_{post} such that

$$his_{pre} \cdot \overline{hf_B} \# \Phi(\overline{\Psi(hfs')},\ \overline{hf_B}) \cdot his_{post} \in auth_{\mathrm{a}}.$$

By ASM 5, we can take the suffix:

$$\overline{hf_B} \# \Phi(\overline{\Psi(hfs')},\ \overline{hf_B}) \cdot his_{post} \in auth_a.$$

Finally, ASM 4 allows prepending $\overline{hf_A}$ to this authorized path.

COND 5 is required to show that *hfs* and hfs_{gen} are valid for the same *tok* in the second case above (however, we have elided packet token fields from the presentation). ASM 3 is needed in cases we have not shown, e.g., when $A \in \mathcal{N}_{attr}$ and hf_B is invalid. □

4.6 Instances

We now instantiate the concrete parametrized model to several protocols from the literature and variants thereof. To do so, we instantiate the model's protocol parameters and prove the associated conditions (§4.5.2). Since refinement and instantiation preserve properties, the path authorization and detectability security properties proven for the abstract model also hold for these instance models.

4.6.1 SCION

SCION embeds a MAC in each hop validation field. In this instance, hvf_A for each hop A with a next hop B is the MAC computed over the abstract hop information field of A (containing *prev*, *next* and *id*), the abstract hop field of B, and the hvf_B field, using the symmetric key K_A shared by all border routers in the AS A:

$$hvf_A = \mathsf{MAC}_{K_A}(\langle hi_A, hi_B, hvf_B \rangle)\,. \tag{4.15}$$

The last AS A has no successor, so $hvf_A = \mathsf{MAC}_{K_A}(hi_A)$. Our model includes the timestamp TS_{path} in the MAC computation as well, but we defer its presentation to §4.7.6.

We instantiate ψ with the check of Equation (4.15) and set $ik_0^+ = \emptyset$. In SCION, the *tok* field is not used. We simply set $\textit{auth-restrict}(hfs, u) = (u = 0)$ (where 0 is a term containing the natural number "0") to ensure that this field does not leak any new terms that the intruder cannot otherwise derive.

In this and all following instances, we only define *extract* for valid patterns; all other patterns are mapped to $\langle\rangle$.

$$extract(\mathsf{MAC}_{K_A}(hi_A)) = hi_A$$
$$extract(\mathsf{MAC}_{K_A}(\langle hi_A, hi_B, hvf_B\rangle)) = hi_A \# extract(hvf_B)$$

To show that this model of SCION inherits the security properties proven in the parametrized models, we prove the parametrized model's conditions COND 1–COND 5 (§4.5.2). First, we observe that the intruder knowledge only contains keys and MACs, which cannot be decomposed, hence $DY^{\downarrow}(ik_0 \cup ik_0^{+}) = ik_0$. With this simplification in place, we easily prove COND 1, COND 2, and COND 5 by unfolding the definitions of ik_0, $auth_c$, and ψ. We establish COND 3 and COND 4 by routine inductions over *hfs*.

VARIANTS By dropping hi_B from Equation (4.15) we obtain the variant described in §2.2.1 where we instantiate ψ with the check Equation (2.1). The proof of the conditions is almost identical.

4.6.2 EPIC

EPIC Level 1 uses a *hop authenticator* σ (which is a static authenticator almost identical to Equation (2.1)) to compute the segment identifier S, which is static, and V_A, which changes with each packet. Packets contain a *packet origin*, which is a triple of the source address, path timestamp, and packet timestamp offset. The time given by the timestamp and the offset is precise enough to guarantee that the packet origin of each packet is unique.

We define the hop validation field hvf_A as follows:

$$hvf_A = \langle S_A, V_A\rangle. \tag{4.16}$$

The values σ_A, S_A and V_A are computed as follows.

$$\sigma_A = \mathsf{MAC}_{K_A}(\langle hi_A, S_B\rangle) \tag{4.17}$$
$$V_A = \mathsf{MAC}_{\sigma_A}(tok)\,. \tag{4.18}$$
$$S_A = \mathsf{H}(\sigma_A). \tag{4.19}$$

If A has no successor, then $\sigma_A = \mathsf{MAC}_{K_A}(hi_A)$. In the EPIC protocol specification from the previous chapter, we set S_A to be the value of σ_A shortened to a few bytes. Here, we model truncation as a hash function. We discuss this and other differences between protocols and their models in §4.8.2.

To check the validity of the *HVF*, a border router A re-computes its own σ_A (using its key K_A and S_B from the successor hop field) and then re-computes hvf_A from σ_A and the *tok* included in the packet.

FORMALIZATION We defined *tok* as a natural number, i.e., a public value, which represents the packet origin. Making use of an extension presented in the next section that parametrizes its type, we do not need to use *auth-restrict* to restrict this field. We instantiate the predicate ψ with the conjunction of the four equations given above. We define *extract* such that it first extracts the hop authenticator, and then the path. Patterns not covered below map to $\langle\rangle$.

$$\begin{aligned}
&\mathit{extract}(\langle S_A, \mathsf{MAC}_{\sigma_A}(\mathit{tok})\rangle) = \mathit{extract}'(\sigma_A)\\
&\mathit{extract}'(\mathsf{MAC}_{K_A}(\mathit{hi}_A)) = \mathit{hi}_A\\
&\mathit{extract}'(\mathsf{MAC}_{K_A}(\langle \mathit{hi}_A, \mathsf{H}(\sigma_B)\rangle)) = \mathit{hi}_A \,\#\, \mathit{extract}'(\sigma_B)
\end{aligned}$$

According to our definition of the intruder knowledge given in Equation (4.13), the attacker knows the *HVF* values of all authorized paths. We define ik_0^+ such that the attacker additionally knows all hop authenticators of authorized paths, since these are public in EPIC protocols.

We show that EPIC is an instance of our concrete parametrized model, and thus inherits the security properties proven in the abstract model. The proof is closely related to that of the SCION instance, but requires additional case distinctions since ik_0^+ provides the attacker with more ways to derive terms, i. e., from hop authenticators. We also need to prove a lemma stating that if a hop authenticator from ik_0^+ is used to create a *HVF* of a valid hop field with some *tok* value, then that hop field is contained in an authorized path.

VARIANTS As presented in the previous chapter of this thesis, the hop validation field and the segment identifier are shortened to a few bytes in EPIC, and can, with considerable but realistic effort, be brute-forced by an attacker. This allows the attacker to send packets that have valid authenticators, but are unauthorized. While the shortening of the hop fields is not reflected in the above model, it is reflected in the strong attacker model via the attacker's brute-force capabilities. We will present this strong attacker model as an extension of our framework in §4.7.2.

In addition to verifying EPIC level 1 in the standard attacker model, we verify it also in the strong attacker model. We verify level 2, also in

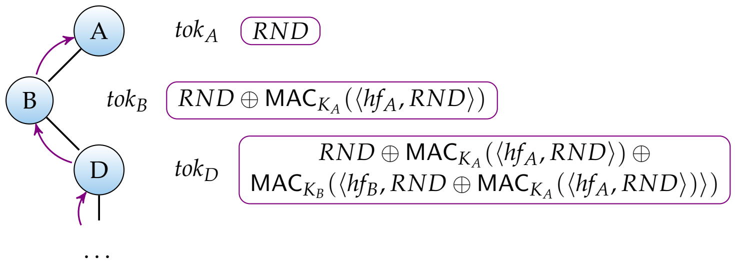
A
B
D
...
tok_A
RND
tok_B
$RND \oplus \mathsf{MAC}_{K_A}(\langle hf_A, RND\rangle)$
tok_D
$RND \oplus \mathsf{MAC}_{K_A}(\langle hf_A, RND\rangle) \oplus \mathsf{MAC}_{K_B}(\langle hf_B, RND \oplus \mathsf{MAC}_{K_A}(\langle hf_A, RND\rangle)\rangle)$

FIGURE 4.4: Values that the mutable *tok* field has in Anapaya-SCION as a packet traverses the network from D to A. It is updated by the receiving router according to Equation (4.20). The *tok* displayed next to each AS *i* is used to compute the *HVF* of AS *i* according to Equation (4.21).

the strong attacker model. We do not model or verify the separate replay suppression system.

4.6.3 Anapaya-SCION

The SCION protocol as presented in §4.6.1 was first published in 2011 [108]. In 2019, the company Anapaya that commercializes the SCION architecture, proposed to change the way in which MACs are nested [4]. In this variant, which we call Anapaya-SCION, the *HVF* of each hop field is a MAC over the local routing information *hi* and the *tok* field. The *tok* field is updated by routers during the forwarding of a packet, and combines the *HVFs* of all subsequent hops with exclusive-or (XOR), thereby including the upstream path similar to the MAC chaining of SCION and EPIC.

The control plane creates forwarding paths such that for each hop on the path, the *tok* value embedded in its *HVF* is the XOR of the *HVFs* of all previous hops in beaconing direction and of a random initialization value *RND*. Since paths are reversed on the data plane, there, the *HVFs* of all *following* hops *in forwarding direction* are included in the *tok* field. During forwarding along authorized segments, the updates of the *tok* field by routers successively remove (by the cancellation property of XOR) the *HVFs* that were added during beaconing until finally, only *RND* remains on *tok*.

We will use the topology and *tok* fields given in Figure 4.4 as a running example.

MOTIVATION In the original SCION protocol, there are a number of different cases for switching between different segments. In our models, these cases are not visible, since we abstract from segment switching. However, when implementing the SCION router, engineers found that fields in multiple variable memory locations needed to be fetched depending on the type of segment switching occurring. In particular, not only the location of the current hop field, but also the locations of other hop fields used to compute the *HVF* are variable. Implementations needed to either (i) first fetch the fields that allow making the required case distinction and then only load the memory locations required for the case at hand or (ii) already from the beginning load all the fields that are required in *any* of the cases. Neither option is efficient, in particular in hardware implementations.

This motivated creating the Anapaya-SCION protocol, in which only the location of the current hop field is variable, and the locations of all other fields needed for forwarding are fixed in the packet header. This simplifies the protocol by reducing the number of case distinctions and allows routers to process packets more quickly.

While the implementation of Anapaya-SCION may indeed be simpler, formally proving its security is more difficult than for the other instances and requires the use of several extensions presented below.

PROTOCOL DESCRIPTION The packet token field is mutable. When an AS receives a packet from an inter-AS channel, it updates its *tok* field using the following function, which takes the current *tok* field u and the current hop field:

$$\mathit{upd\text{-}tok}(u, \mathit{hf}) = u \oplus \mathit{HVF}(\mathit{hf}), \tag{4.20}$$

where $\oplus$ is exclusive-or. The receiving router first updates the *tok* field and then proceeds normally by checking the interface and *HVF*. The *tok* field update only needs to be performed once per AS. Hence, the sending router (which forwards an intra-AS packet to an inter-AS channel) does not need to perform the update before computing the *HVF*. The *HVF* is computed as:

$$\mathit{hvf}_A = \mathsf{MAC}_{K_A}(\langle \mathit{hi}_A, \mathit{tok}\rangle)\,. \tag{4.21}$$

EXAMPLE Assume that, given the topology in Figure 4.4, an end host in AS D sends a packet to an end host in AS A. The packet is initialized by the end host with tok_D as the *tok* field. The sending router at D checks the hvf_D

based on the initial *tok* value. It then pushes the current hop field into the past path and forwards the packet to the inter-AS channel between D and B. The receiving router at AS B first checks the interface over which the packet was received. It then updates the *tok* field. To do so, it XORs the current tok_D field with the (unvalidated) hvf_B embedded in the packet, yielding tok_B. Only after updating the *tok* field can the router check the validity of the hvf_B field, by recomputing the MAC based on tok_B. The forwarding between AS B and AS A proceeds similarly. Note that the initial value RND, which is random, does not need to be checked by AS A.

Formalization

REQUIRED EXTENSIONS In order to formalize this instance, we need to extend the framework presented in the previous section in three ways. First, by adding an abstract XOR operator, second, by parametrizing the type of *tok*, and third, by allowing *tok* fields to be updated en-route.

Since the focus here is on the Anapaya-SCION protocol, we only briefly introduce these extensions, and defer their full description to Sections 4.7.1, 4.7.4, and 4.7.5.

- We extend our framework with an abstraction of XOR by adding a constructor XOR for finite sets of terms (of type $\mathcal{P}_{\mathit{fin}}(\mathbb{T})$) to the datatype $\mathbb{T}$. XOR H represents the XORing of all elements of H. We define the operator $\oplus : \mathcal{P}_{\mathit{fin}}(\mathbb{T}) \times \mathcal{P}_{\mathit{fin}}(\mathbb{T}) \to \mathcal{P}_{\mathit{fin}}(\mathbb{T})$, for combining finite sets of terms with XOR, as the symmetric difference of these sets and the identity element $\{\}$ as the empty set. Our XOR abstraction captures the algebraic properties of XOR while simplifying the interface with the attacker. The XOR extension requires changes to the Dolev–Yao model formalization, but no other changes in the concrete model.

- We parametrize the type of *tok*. Instead of $\mathbb{T}$ it is of the abstract type *'TOK*. This requires an additional protocol parameter to extract the intruder knowledge from a given *tok* field, but requires no extra proof effort.

- We allow for updatable packet token fields by adding the update function *upd-tok* : $'TOK \times \mathrm{HF} \to 'TOK$ as a new parameter to the concrete model. This requires adding two simple conditions. Definitions and proofs of the concrete model must be changed to account for the mutable field.

With these extensions in place, the formalization of the update function *upd-tok* as in Equation (4.20) is straightforward. We instantiate the type of *'TOK* with $\mathcal{P}_{fin}(\mathbb{T})$, i.e., finite sets of terms, since *tok* accumulates terms using XOR. We define the hvf_A follows:

$$hvf_A = \mathsf{MAC}_{K_A}\left(\langle \overline{hf_A}, u\rangle\right).$$

Note that the next hop field hf_B is not required in this definition.

RESTRICTION As in the other protocols, we need to ensure that the *tok* field of authorized paths does not leak arbitrary terms to the attacker. According to the protocol specification, the *tok* field is initialized on the control plane by some random value *RND*. The randomness in the initialization is not required to achieve the security properties, and it is unclear why the protocol designers chose to include it.

In our model, we verify a simplified version in which we assume that *tok* fields are initialized with the identity element $\{\}$. We define *auth-restrict* as follows:

$$\begin{aligned} \textit{auth-restrict}(hfs \cdot \langle hf_Z\rangle, u) &= \psi(hf_Z, \bot, \{\}) \\ \textit{auth-restrict}(\langle\rangle, u) &= (u = \{\}) \end{aligned}$$

The first case is that of a non-empty path. We require that the last hop field on the path is valid for the *tok* field with the identity element. This implies that for valid paths $\Psi(hfs, u) = hfs$, u does not contain any terms besides the *HVFs* of subsequent hop fields. For an empty path, u itself has to be the identity element. This ensures that no unintended terms are leaked to the attacker via the set of authorized paths.

EXTRACT The *extract* function for Anapaya-SCION is more difficult to define than in the other instances. Using the definite description operator THE and the *is-next* predicate defined below, it is defined as follows:

$$\begin{aligned} \textit{extract}(\mathsf{MAC}_{K_A}(\langle hi_A, u\rangle)) = \; & hi_A \,\#\, (\text{if } \exists hf_B.\ \textit{is-next}(hf_B, u) \\ & \quad \text{then } \textit{extract}(\textit{HVF}(\textsc{the}\ hf_B.\ \textit{is-next}(hf_B, u))) \\ & \quad \text{else } \langle\rangle). \end{aligned}$$

As in the other models, *extract* first takes the current *HVF*, and if it is of the correct form (as defined in Equation (4.21)), the HI embedded in the MAC of the *HVF* is extracted. If there is a next hop field hf_B, then *extract* recursively calls itself on that next hop's *HVF* embedded in the MAC. If there is no such element, the *extract* function does not recurse. In contrast to SCION, the next hop's *HVF* is not *directly* embedded in the MAC of the current *HVF*. Instead, the *tok* field is embedded, which accumulates via XOR all *HVFs* remaining on the path. The extract function needs to identify and pick out the *next* hop's *HVF* among this set.

The difficulty in formally defining *extract* is determining the next hop field. To this end, we define the predicate $\mathit{is\text{-}next} : \mathrm{HF} \times {'}\mathit{TOK} \to \mathbb{B}$ below, where $\mathit{is\text{-}next}(\mathit{hf}_B, u)$ holds if hf_B is the next hop field on the *tok* field u. As we show, there is at most one hop field that satisfies this predicate for a given u. Hence, if this hop field exists, we can use the definite description operator THE to refer to it.

We observe that on paths that are valid and satisfy *auth-restrict*, if the *tok* field is not empty, then it contains exactly one element hvf_B that is a MAC created using the *all other elements* of the *tok* set, i.e., $\mathit{hvf}_B = \mathsf{MAC}_{K_B}(\mathit{hi}_B, \mathit{tok} \setminus \{\mathit{hvf}_B\}))$ (where $\setminus$ is set difference). Let hf_B be the hop field such that $\mathit{HVF}(\mathit{hf}_B) = \mathit{hvf}_B$ and $\overline{\mathit{hf}_B} = \mathit{hi}_B$. This hop field satisfies the following predicate, i.e., $\mathit{is\text{-}next}(\mathit{hf}_B, \mathit{tok})$ holds.

$$\mathit{is\text{-}next}(X, u) = (\mathit{HVF}(X) = \mathsf{MAC}_{K_{\mathit{id}(X)}}\left(\overline{X}, \mathit{upd\text{-}tok}(u, X)\right)).$$

The predicate can only hold if $\mathit{HVF}(X) \in u$, in which case $\mathit{upd\text{-}tok}(u, X)$ is equivalent to $u \setminus \{\mathit{HVF}(X)\}$. To see this, assume, $\mathit{HVF}(X) \notin u$. Then $\mathit{HVF}(X)$ contains itself under a constructor,

$$\mathit{HVF}(X) = \mathsf{MAC}_{K_{\mathit{id}(X)}}\left(\overline{X}, u \cup \{\mathit{HVF}(X)\}\right),$$

leading to a contradiction with $\mathit{is\text{-}next}(X, u)$ if we consider the size of the term.

EXAMPLE Consider in Figure 4.4 that we extract the path from D's hop validation field, which is defined as

$$\mathit{hvf}_D = \mathsf{MAC}_{K_D}(\langle \mathit{hi}_D, \mathit{tok}_D \rangle)\,.$$

Given tok_D, a possible value X such that $\mathit{is\text{-}next}(X, \mathit{tok}_D)$ holds is the hop field hf_B, where $\mathit{HVF}(\mathit{hf}_B) = \mathit{hvf}_B$, $\overline{\mathit{hf}_B} = \mathit{hi}_B$ and $\mathit{hvf}_B = \mathsf{MAC}_{K_B}(\langle \mathit{hi}_B, \mathit{tok}_B \rangle)$, since $\mathit{upd\text{-}tok}(\mathit{tok}_D, \mathit{hvf}_B) = \mathit{tok}_B$. This is not just some value that satisfies *is-next* given tok_D, but it is the *only* value to do so. Hence, $\mathit{extract}(\mathit{hvf}_D) = \mathit{hi}_D \,\#\, \mathit{extract}(\mathit{hvf}_B)$. If we repeat these steps, we obtain $\mathit{extract}(\mathit{hvf}_D) = \langle \mathit{hi}_D, \mathit{hi}_B, \mathit{hi}_A \rangle$, as expected.

PROOFS We first show that if a term t is contained in a *tok* field u, and $auth_c(u)$ is not empty (i.e., there exists a path with u as the *tok* field), then t is a *HVF* of an authorized path, and hence already contained in the intruder knowledge. We thus do not need to consider *tok* fields in the initial intruder knowledge. With this simplification proven, the use of XOR does not complicate the proofs in the instance model.

Yet, proving the conditions is more difficult in Anapaya-SCION than in other protocol instances, in particular COND 3 and COND 4, due to the complex definition of *extract*.

4.6.4 ICING

Among the protocols we study, ICING [84] provides the strongest security properties, albeit at the cost of the highest overhead [71]. In particular, ICING allows ASes to authorize the entire path and is thus an instance of the undirected setting. This requires a separate concrete model, whose parameters, assumptions, and conditions slightly differ from those presented above. We discuss this model in §4.7.3.

ICING uses *proofs of consent* (PoCs) to achieve path authorization. These are created by applying a pseudorandom function (PRF) using a *tag key* on the entire forwarding path. The tag key for each AS is derived from its master key K_A, and the local hop field hf_A.

FORMALIZATION In our symbolic model, PRFs and MACs are modeled identically, we thus use MACs in our definitions. We use these PoCs as hop validation fields:

$$hvf_A = \mathsf{MAC}_{\langle K_A,\, hf_A \rangle}(rev(past(m)) \cdot fut(m))\,. \tag{4.22}$$

We define $ik_0^+ = \emptyset$. The function *extract* requires extracting the entire path (past and future) in the undirected setting:

$$extract(\mathsf{MAC}_{\langle K_A,\, hf_A \rangle}(hfs)) = hfs. \tag{4.23}$$

VARIANTS We verified two other versions of the protocol. In the first one, the hop validation field consists of ICING's *path authenticator*, which includes an expiration timestamp and a path hash besides the PoC. These additional details are not essential for achieving path authorization and detectability but reduce the gap between the model and proposed protocol. We define ik_0^+ to consist of all authorized PoCs, since the attacker cannot extract them directly from the packets in this version.

Extension	**SCION**	**EPIC**	**Anap.**	**ICING**
Type parametrization	✓	✓	✓	✓
Strong attacker model	✗	(✓)	✗	✗
Undirected setting	✗	✗	✗	✓
Exclusive-or	✗	✗	✓	✗
Mutable *tok* fields	✗	✗	✓	✗
Additional auth. fields	✓	✓	✓	✓

Figure 4.5: Extensions used by protocol models. (✓) means that the extension is used by some (but not all) variants.

The second version is a further simplified variant of ICING compared to the one presented first, which omits the hop field in the key input of the MAC computation. Proving the concrete model's conditions is straightforward for all three versions. However, the simplest one requires the additional assumption that an AS cannot have two different hop fields on the same path, since otherwise they would have the same MAC, despite having different local forwarding information.

4.7 Extensions

We now describe a number of features of our formalization that we elided to simplify the presentation. Figure 4.5 lists which extensions are used in the individual protocol models presented in the previous section.

4.7.1 Type parametrization in parametrized models

While we have presented all definitions of the concrete model with specific types for simplicity, our formalization uses type parameters for some fields. This allows for greater flexibility in the modeling of protocols.

When the type of a field is only determined by the instances, defining the intruder knowledge in the parametrized model requires an additional protocol parameter. This parameter defines what terms an attacker can learn from analyzing the field.

For instance, the type of *tok* is the abstract type *'TOK*. We add a parameter which captures what an attacker can learn from analyzing a *tok* field:

$$\mathit{analz\text{-}tok} : \mathit{'TOK} \to \mathcal{P}(\mathbb{T}). \tag{4.24}$$

We change the definition of *terms* for packets to use $\mathit{analz\text{-}tok}(\mathit{tok}(\mathit{pkt}))$ instead of $\{\mathit{tok}(\mathit{pkt})\}$.

Instances

In Anapaya-SCION, we instantiate the type of *tok* to finite sets of $\mathbb{T}$ and define *analz-tok* as the identity function. In SCION, *'TOK* is $\mathbb{T}$ and $\mathit{analz\text{-}tok}(t) = \{t\}$. We note that for all valid hop fields in EPIC (and in SCION), *tok* is a natural number, which the attacker can already derive. Hence, we simplify the model when formalizing the EPIC protocols and set *'TOK* to $\mathbb{N}$ and $\mathit{analz\text{-}tok}(t) = \{\}$.

4.7.2 Strong attacker model

In §3.2.1, we proposed a strong attacker model for EPIC protocols, which reflects the fact that an adversary can with some effort brute-force correct *HVF* fields for individual *tok* values. In Equation (3.11) we defined $\mathit{oracle}(l)$ for each EPIC protocol level $l \in \{1, 2, 3\}$. Given the *PO* of a packet and its *hi* fields, this oracle returns valid validators V_i and segment identifiers S_i.

We model the oracle slightly differently in our formalization, in order to abstract from EPIC and allow for a more generic oracle. We add a predicate

$$\mathscr{O} : \mathit{'TOK} \to \mathbb{B} \tag{4.25}$$

as an additional environment parameter of the concrete parametrized model, which is true for all *tok* values for which the attacker queried the oracle. In instances that make use of the strong attacker model (i.e., EPIC), the ik_0^+ set is defined to additionally contain all valid *HVF* fields of (possibly unauthorized) paths which are created over *tok* values such that $\mathscr{O}(\mathit{tok})$ holds.

While this strictly strengthens the attacker, her events must be restricted to rule out trivial attacks, in which the attacker sends a valid but unauthorized packet with a *tok* value for which she queried the oracle. We add the guard $\neg\mathscr{O}(\mathit{tok}(m))$ to the $\textbf{dispatch-int}_\text{c}$ and $\textbf{dispatch-ext}_\text{c}$ events to prevent the attacker from sending packets whose *HVF* fields are directly obtained from the oracle. We also add $\neg\mathcal{O}(tok(m))$ to the premises of COND 1 and COND 2.

The instance proofs for the EPIC protocols are similar to the proof in the basic attacker model. However, they must additionally account for the attacker obtaining valid hop fields from the oracle.

4.7.3 Undirected authorization schemes

For brevity, we have focused on directed authorization schemes, where each AS only controls the authorization of the traversal of subsequent ASes (in forwarding direction) and the traversal of previous ASes is outside of its control. This setting allows the attacker to legitimately extend and change her own path in the control plane without consent by subsequent ASes, and hence requires the control plane assumptions ASM 3–ASM 6.

We have a separate parametrized model for the undirected authorization scheme, where the entire path must be approved by all on-path ASes. The control plane assumptions can be relaxed, and ASM 3–ASM 6 are replaced by the following weaker assumption stating that an attacker can create authorized paths consisting entirely of compromised nodes.

ASM 7: Fully compromised paths
$his \in auth_{\mathrm{a}}$ if $id(hi) \in \mathcal{N}_{attr}$ for all $hi \in his$.

In this model, the cryptographic check parameter has the entire path (including the past path) as an argument: $\psi : \mathrm{HF} \times \mathrm{HF}^* \times \mathbb{T} \to \mathbb{B}$. The parameter *extract* retains its type, but returns the entire path (including past path) instead of just the future path. Since each hop validation field contains the entire path, COND 3 and COND 4 are replaced by COND 6, which states that for a valid *hf*, *extract* returns the entire path. Formally,

COND 6: Undirected extract:
$\psi(hf, hfs, tok)$ implies $extract(hf) = \overline{hfs}$.

Formalization and proofs

In the undirected setting, the entire path is embedded in each *HVF* and cannot be modified unless it is completely under the attacker's control. Induction is neither required to show the refinement of the dispatch events in the concrete model nor to show the conditions in the ICING instance model. Hence, proofs are substantially easier than in the directed setting.

We again utilize parametrization to avoid redundancy and duplicated proof efforts in our models. Rather than having one concrete model for the directed setting and another concrete model for the undirected setting, we use an intermediate model which generalizes the definitions in the directed and undirected models. The concrete model's event system definition, invariant proof and refinement proof all belong to this common intermediate model, which interfaces with the directed and undirected concrete models via a number of parameters and conditions. In particular, it interfaces with these models by assuming Theorem 2. The proof of this theorem is done separately, since it differs between the directed and undirected setting.

4.7.4 Exclusive-or abstraction

We extend our framework with an abstraction of exclusive-or (XOR), which is used in the Anapaya-SCION instance.

The XOR operator $\oplus$ can be characterized by the following equations for associativity (A), commutativity (C), the identity element 0 (I) and self-inverseness (S).

$$(x \oplus y) \oplus z = x \oplus (y \oplus z) \qquad (A) \qquad (4.26)$$
$$x \oplus y = y \oplus x \qquad (C) \qquad (4.27)$$
$$x \oplus 0 = x \qquad (I) \qquad (4.28)$$
$$x \oplus x = 0. \qquad (S) \qquad (4.29)$$

This operator is particularly difficult to support in symbolic protocol analysis.

Existing modeling approaches

Most automated protocol verifiers that support equational theories cannot straightforwardly implement XOR via the above equations, as they do not form a subterm-convergent equational theory. Simply speaking, this means that XOR cannot be characterized by a set of equations that each simplify a given term by replacing it with a subterm or a constant. This is in contrast to, for instance, symmetric encryption, which can be modeled by a subterm-convergent equational theory described by the single equation $dec(k, enc(k, m)) = m$. While there are some security protocol verifiers that support XOR and other non-subterm-convergent theories, verifying protocols that make use of this constructor remains difficult.

A generic approach to incorporating a large class of equational theories into verifiers is provided by Escobar et al. [43], who propose an approach that they call *folding variant narrowing*. While this technique is powerful and applicable to a wide range of equational theories, modeling and reasoning about equality modulo axioms as required by their approach requires considerable effort in protocol verification by interactive theorem proving. An alternative way of modeling XOR is to use *term normalization* to ensure that *any* two terms that are equal under the equational theory are equivalent under normalization. With this approach, if we normalize all terms under consideration, then equality does not require equational theories.

For a binary $\oplus$ operator, such a normalization could be solved by (i) removing parentheses from terms being XORed and putting them into a sequence, (ii) linearly ordering the sequence and (iii) applying (I) and (S) exhaustively as simplification rules. For instance, assuming that x, y and z are non-XOR terms and that $x < y < z$, the term $(z \oplus x) \oplus (y \oplus z)$ would be successively transformed to (i) $\text{XOR}\langle z, x, y, z\rangle$, (ii) $\text{XOR}\langle x, y, z, z\rangle$, (iii) $\text{XOR}\langle x, y, 0\rangle$ and finally $\text{XOR}\langle x, y\rangle$. Since terms are enumerable, one can define a linear order on them. The approach of modeling XOR as a binary operator and defining normalization is taken by Schaller et al. [94] in their Isabelle/HOL formalization.

Our XOR model

We introduce a new term constructor

$$\text{XOR}\ (\mathcal{P}_{\mathit{fin}}(\mathbb{T}))$$

to the definition of $\mathbb{T}$. This constructor takes a finite set of $\mathbb{T}$ and represents XORing all elements of that set. In particular, $\{\}$ represents the identity 0. Finite sets are unordered and hence do not require a linear order. Moreover, each element occurs at most once, which models the self-inverse property (S).

We define a binary exclusive-or function $\oplus : \mathcal{P}_{\mathit{fin}}(\mathbb{T}) \times \mathcal{P}_{\mathit{fin}}(\mathbb{T}) \to \mathcal{P}_{\mathit{fin}}(\mathbb{T})$ as the symmetric difference of two sets:

$$X \oplus Y = (X \cup Y) \setminus (X \cap Y). \tag{4.30}$$

If a term is contained in both X and Y, it is not in $X \oplus Y$ according to this definition. Our definition of XOR satisfies the four properties given above. We extend the rules for term derivation below to incorporate the new XOR constructor.

Normal terms

The use of finite sets greatly helps to reduce the ways in which a value can be represented by different terms. For instance, $\{x\} \oplus \{y\}$ and $\{y\} \oplus \{x\}$ are not different terms that represent the same value, they are identical terms (as are $\{x\} \oplus \{x\}$ and $\{\}$). However, the above abstraction does not ensure that each value is represented by *at most* one term. For instance, XOR $\{x, \text{XOR}\ \{y,z\}\}$ and XOR $\{x,y,z\}$ represent the same value. To rule this out, we define a predicate *normal*, which prohibits directly nested XOR constructors.

$$\begin{aligned}
\mathit{normal}(\mathsf{H}(x)) &= \mathit{normal}(x) \\
\mathit{normal}(\langle x_0, \ldots, x_n \rangle) &= \forall t \in \{x_0 \ldots, x_n\}.\ \mathit{normal}(t) \\
\mathit{normal}(\text{XOR}\ \{x_0, \ldots, x_n\}) &= \forall t \in \{x_0 \ldots, x_n\}.\ \mathit{normal}(t) \land \forall Y.\ t \neq \text{XOR}\ Y \\
\mathit{normal}(x) &= \mathsf{true} \qquad \text{(for all other x).}
\end{aligned}$$

For instance, if x, y and z are normal, non-XOR terms, then the term XOR $\{x, \text{XOR}\ \{y,z\}\}$ is not normal, but XOR $\{x, \mathsf{H}(\text{XOR}\ \{y,z\})\}$ is.

Rather than defining a normalization function that turns an arbitrary term into a normal term, we define our event system such that all terms in reachable states are already normal, i.e., do not contain directly nested XOR constructors. We show this below.

Model limitations

Two different normal terms represent distinct values. However, there is a notable exception to this: XOR-terms over single (non-XOR) elements represent the same values as the elements themselves, despite being distinct terms. Concretely, x and XOR $\{x\}$ are different terms and both normal (assuming that x is non-XOR and normal), even though in an implementation they may be represented by the same bitstring. This is similar to how the terms XOR $\{\}$, $\langle\rangle$ and the term containing the natural number 0 may all have the same bitstring representation, even though these are different terms in the model. XOR terms are modeled with their own constructor and hence are different from non-XOR terms in the model.

Successful verification in our model does not rule out type-flaw attacks, including attacks that exploit x and XOR $\{x\}$ having the same bitstring representation. A possible solution to this limitation is to require implementations to tag each field with its type [53], including an "XOR type". This would cause x and XOR $\{x\}$ to be represented by different bitstrings, and ensure that each bitstring corresponds to at most one value.

$$\frac{\text{XOR }\{t_1,\ldots,t_n\} \in DY^{\downarrow}(H)}{t_i \in DY^{\downarrow}(H)}\ 1 \leq i \leq n$$

$$\frac{t_1 \in DY^{\uparrow}(H)\ \cdots\ t_n \in DY^{\uparrow}(H)}{\text{XOR }\{t_1,\ldots,t_n\} \in DY^{\uparrow}(H)}\ \forall t \in \{t_0 \ldots, t_n\}.\ \forall Y.\ t \neq \text{XOR } Y$$

Figure 4.6: Rules added to the Dolev–Yao message decomposition ($DY^{\downarrow}$) and composition ($DY^{\uparrow}$) presented in Figure 4.3. The composition rules out directly nested XOR constructors (c.f. §4.7.4).

Attacker capability overapproximation

Broadly speaking, XOR is utilized in security protocols for two different purposes. First, XOR is used to achieve secrecy. Plaintext values can be masked using XOR with values unknown to the attacker. The prototypical example for this use is the One-Time Pad encryption scheme, in which a plaintext message is XORed with a key (of equal length) to obtain the ciphertext. Second, XOR is used as a compression function. As opposed to one-way compression functions, XOR is trivial to invert if one of the inputs is known. Nevertheless, XOR is used in many protocols because it is simple, implementations in hardware are highly efficient and one-wayness is sometimes not required.

We observe that none of the properties of interest in data plane protocols are secrecy properties. Since XOR is used only as a compression function, we can overapproximate the attacker capabilities when analyzing terms containing XOR. We do this by adding the derivation rules in Figure 4.6 to the Dolev–Yao closure. We simply model that the attacker learns both terms x and y when analyzing a term $x \oplus y$, or, in terms of our formalization, the attacker learns all $t \in Y$ when analyzing XOR Y. This behavior is captured by the first derivation rule of the above figure, which covers term decomposition. The second rule allows the attacker to construct new XOR terms. Its side condition ensures that this rule (like other construction rules) preserves term normality.

As an alternative to the overapproximation that we propose, XOR could be modeled such that a term x can be decomposed from $x \oplus y$ if and only if y is known. Hence, the set of terms derivable from an XOR-term would depend on other terms known. Our overapproximation of the attacker derivation capabilities greatly simplifies reasoning about the intruder knowledge and the derivation of terms, since the decomposition of XOR terms is defined without reference to other terms.

Protocol instances

We only use the XOR abstraction in the Anapaya-SCION instance. We could use this extension to formalize the use of XOR in EPIC Level 3 and in ICING. In both protocols, the use of XOR is unrelated to path authorization and would additionally require updatable hop fields, which are currently not supported in our framework. Hence, we do not verify EPIC Level 3 and only verify a simplified version of ICING without XOR.

To show that all terms contained in an instance model are normal, first we observe that all messages in a state are contained in the intruder knowledge $ik(s)$. By the constant intruder knowledge invariant proven in the concrete model, Lemma 1, we can reduce the normality of terms in $ik(s)$ to the normality of terms in $DY(ik_0 \cup ik_0^+)$. We prove that all terms derivable from the initial intruder knowledge $ik_0 \cup ik_0^+$ are normal by showing that (i) all terms in the initial intruder knowledge are normal, and (ii) that normality is closed under term derivation. The former has to be shown separately for each instance (i.e., for Anapaya-SCION), whereas the latter is proven in general in the concrete model.

4.7.5 Mutable packet token fields

This extension is used to model protocols in which routers receiving a packet from an inter-AS channel update the *tok* field. We extend our framework by an update parameter

$$\textit{upd-tok} : \text{'TOK} \times \text{HF} \rightarrow \text{'TOK}. \tag{4.31}$$

This function is called to update a *tok* field u given a hop field *hf*, resulting in the new *tok* field $\textit{upd-tok}(u, \textit{hf})$.

We introduce a function $\textit{upd-pkt} : \text{PKT}_c \rightarrow \text{PKT}_c$, which applies *upd-tok* to update a packet's *tok* field using the first hop field of the future path. It is defined as

$$\begin{aligned} &\textit{upd-pkt}(m) = m(\!|\, \textit{tok} := \textit{upd-tok}(\textit{tok}(m), \textit{hf}) \,|\!) && \text{if } \textit{fut}(m) = \textit{hf} \# \textit{hfs} \\ &\textit{upd-pkt}(m) = m && \text{otherwise.} \end{aligned}$$

The $\mathbf{recv}_c$ event's guard is changed as follows: instead of requiring that ψ holds on (fields of) m, we require the check to hold on $\mathit{upd\text{-}pkt}(m)$. Furthermore, the update of the event is changed and instead of m, $\mathit{upd\text{-}pkt}(m)$ is added to $\mathit{int}(A)$. Hence, the receiving router updates the *tok* field *prior* to processing, whereas the sending router pushes the first hop field of the future path into the past path *post* processing.

This extension requires a number of changes to the framework, such as modifications of the definition of Ψ to account for the *tok* update. Three additional conditions are needed. First, to ensure that the update function does not reveal anything that the attacker could not already derive, second, that *auth-restrict* is closed under updates and third, that $\mathcal{O}$ is closed under updates.

The refinement proof presented in §4.5 remains structurally the same, but has a number of modifications to keep track of changes in the *tok* field.

Lifting updates to paths

Our proofs often involve reasoning about path fragments. For instance, the suffix $\mathit{hf} \# \mathit{hfs}_{\mathit{post}}$ of an authorized path $\mathit{hfs}_{\mathit{full}} = \mathit{hfs}_{\mathit{pre}} \cdot \mathit{hf} \# \mathit{hfs}_{\mathit{post}} \in \mathit{auth}_c(u)$ is valid, i.e., $\Psi(\mathit{hf} \# \mathit{hfs}_{\mathit{post}}, u') = \mathit{hf} \# \mathit{hfs}_{\mathit{post}}$ for some u'. Since the packet's *tok* field changes as the packet is forwarded, $\mathit{hf} \# \mathit{hfs}_{\mathit{post}}$ and $\mathit{hfs}_{\mathit{full}}$ are potentially valid for different values $u' \neq u$. In order to obtain the correct *tok* value u' for $\mathit{hf} \# \mathit{hfs}_{\mathit{post}}$ given the *tok* value u for the full path $\mathit{hfs}_{\mathit{full}}$, we define a function that lifts *upd-tok* to paths. We can use this function to apply the *tok* field update given the preceding path $\mathit{hfs}_{\mathit{pre}}$.

Unfortunately, obtaining u' given the original u valid for $\mathit{hfs}_{\mathit{full}}$ does not simply involve applying *upd-tok* iteratively for each hop field on $\mathit{hfs}_{\mathit{pre}}$. When we reason about hop field validity (and valid prefixes of hop field sequences), we assume that the update function has already been applied to the current head of the path. Hence, assuming that $\mathit{hfs}_{\mathit{pre}} = \mathit{hf}' \# \mathit{hfs}'$, we need to update *tok* for each hop field on $\mathit{hfs}' \cdot \langle \mathit{hf} \rangle$, i.e., we drop the first hop field of $\mathit{hfs}_{\mathit{pre}}$ and add the next hop field after $\mathit{hfs}_{\mathit{pre}}$ in the sequence of hop fields for which an update has to be performed. This "shifted" reasoning requires additional case distinctions and adds some complexity to our definitions and proofs.

Instances

Anapaya-SCION updates the *tok* field using XOR, as defined in Equation (4.20). The other protocol instances do not update the *tok* field and hence define the *upd-tok* to return a given *tok* field unmodified.

Instances need to discharge the three additional conditions, which is straightforward for Anapaya-SCION (and trivial for the other instances).

4.7.6 Additional authenticated fields

To allow for more accurate modeling of protocols, our formalization includes additional per-hop and per-packet fields, which are included in $auth_\mathrm{a}$ and must thus be included in the authentication mechanisms defined by instances.

For instance, in SCION, packet headers include an expiration time that is fixed in the control plane and is included in the MAC computation of the hop validation field. Consequently, paths have a limited lifespan and must be replaced on a regular basis. To model this, our formalization includes an *authenticated info field* associated with each packet. In all of our instance models, this field contains a natural number representing the time at which the path expires.[2]

ICING allows ASes to include arbitrary forwarding information that is authenticated in the control plane in an opaque string called *tag*. Our formalization defines abstract and concrete hop fields in an extensible way, such that additional data can be added in instances. This can be used to model additional forwarding data that must be protected, such as ICING's *tag*.

These extra fields enable a more realistic modeling of existing protocols and make it more likely that future protocols can be modeled. For instance, if a new protocol includes flags indicating a path's priority, or introduces access control fields to allow only some user classes to use certain paths, no changes to the abstract and concrete models are required to create an instance model that proves the protocol's security.

2 In SCION and EPIC protocols, there is a per-path absolute timestamp TS_path and each hop information field contains a relative offset ts_exp, such that the hop field expires at time $TS_\mathrm{path} + ts_\mathrm{exp}$. Since the property that we verify does not take expiration of paths into account and routers in our model do not check for timestamp expiration, we simplify and only model an absolute expiration timestamp being authenticated.

Formalization

This extension requires changes throughout our framework. Here, we describe some of the modifications in the abstract model. The concrete model requires similar changes.

We add per-packet authenticated info fields to PKT_a and per-packet additional authenticated hop information to the definition of HI. We use type parameters *'AINFO* and *'AAHI* for these fields. To distinguish definitions of this extensions from those presented above we add "ext" to their name.

$$\begin{aligned}\text{HI-ext} &= (\!|\, id \in \mathcal{N},\ prev \in \mathcal{I}_\perp,\ next \in \mathcal{I}_\perp,\ aahi \in \text{'}AAHI \,|\!)\\ \text{PKT-ext}_\text{a} &= (\!|\, ainfo \in \text{'}AINFO,\ past \in \text{HI-ext}^*,\ fut \in \text{HI-ext}^*,\ hist \in \text{HI-ext}^* \,|\!).\end{aligned}$$

The type of authorized paths is as follows:

$$\textit{auth-ext}_\text{a} \subseteq \text{'}AINFO \times \text{HI-ext}^*. \tag{4.32}$$

Adding these fields requires changes to the definitions, parameters, assumptions, conditions and lemmas of both the models presented in the previous sections and in the above extensions. Moreover, we define an additional parameter (similar to §4.7.1) that can be set by instances to extend the intruder knowledge by the values of both kinds of fields. While the extension requires numerous changes, the use of these additional fields does not add significant complexity and is not essential to the insights provided by our proofs. Hence, we have elided their presentation above.

4.8 Discussion

4.8.1 Undirected vs. directed protocols

We briefly compare undirected and directed protocols. As shown in §4.7.3, undirected protocols achieve path authorization under weaker assumptions. While this sounds desirable, the undirected protocol also have disadvantages.

In the data plane, existing undirected protocols require each hop to incorporate the entire path into the hop validity check. This incurs a processing overhead linear in the path length. In contrast, the directed protocols that we study only need to check a constant number of fields.

In the control plane, the ways in which paths are authorized in undirected architectures have two drawbacks. First, the beacons creating paths in undirected protocols must complete a round-trip: the first leg to discover the path and the second leg to authorize it. In contrast, directed protocols can achieve both in a single leg, where forwarding along a path is in the opposite direction of path construction. Second, the control plane must mediate between conflicting path policies by ASes. If there is no path that satisfies the constraints by all on-path ASes, then no forwarding can occur. In directed protocols it is simpler to exclude this possibility, for instance by mandating that each AS disseminates at least one beacon from a given AS to each of its neighbors.

In summary, there is a trade-off between these protocol classes that depends highly on the control plane and overall architecture.

4.8.2 Differences between models and real protocols

We now discuss how the protocols from the literature relate to our formalization of them.

Our models abstract from real protocols in ways that are typical for protocol verification. Real protocols operate on bitstrings rather than typed terms, and they contain fields that we do not model, since they are unimportant to the path authorization security mechanism. For instance, real protocols include version numbers and other header fields that are not present in our model. Our model also allows more behaviors than the actual protocol, for instance, by allowing honest senders to include a non-empty *past* path. For the purpose of verifying our properties, which are all safety properties, such overapproximations are sound. Our events accurately model the checks that border routers perform. The $\mathbf{send}_c$ and $\mathbf{recv}_c$ events correspond to the actions of real routers (modulo the queuing of incoming packets). In particular, a router receiving a packet from another AS will check the validity of the interface, and both the receiving and the sending routers will validate the hop validation field as specified in our instance models. While our events accurately model the checks that border routers perform, the interaction with end hosts is simplified, as we do not differentiate end hosts within an AS and do not model intra-AS topologies.

In contrast to our models, the protocols do not include the AS identifier (*id*) in hop fields. Nevertheless, a *HVF* uniquely identifies the AS for which the hop field is valid, since the key of the AS is used in the MAC computation. Hence, it would be possible to refine the instance models to models in which hop fields do not explicitly contain the identifier. Alternatively, one could change the parametrized model to remove the AS identifier. This would likely require additional conditions.

Other than these abstractions, our instance models differ from the actual protocols in the following ways:

SCION

SCION is a complex architecture that includes many features that are necessary for Internet-scale operation. Beacons do not directly establish paths between any pair of ASes, but only between large ASes (e. g., Tier 1 providers) and their customer ASes and between the large-scale ASes themselves. Such partial paths (called *segments*) can then be reversed and concatenated (as discussed in §2.2.2) by end hosts to connect distant ASes. While path authorization only holds for each segment individually, segments cannot be combined arbitrarily: there are rules that ensure that the economic interests of ASes are respected, similar to those of the Gao–Rexford model [47]. Nevertheless these rules only provide *local* properties and not global properties (such as path authorization, which is expressed over entire paths). SCION also allows peering links between ASes, which similarly to segment combinations, are only authorized locally. For these reasons, we do not include segment combinations or peering links in our framework.

EPIC

EPIC trims hop authenticators σ to a short length to reduce space overhead. We model the trimming of σ by a hash function. Similar to hashing, trimming makes it difficult to recover the original value. The trimming enables brute-force attacks, which we model by the oracle discussed in §4.7.2.

Anapaya-SCION

As mentioned above, our formalization does not include the initialization value RND for the tok field.

ICING

We leave out ICING's *proofs of provenance*. They are cryptographic authenticators used for path validation and are unrelated to path authorization. In the original specification, they are combined with the PoCs using XOR.

4.8.3 Formalization details and statistics

Our formalization in Isabelle/HOL closely follows the models and proofs described in this paper, modulo differences in notation and changes required for the extensions, as discussed above. Most of the proof burden is handled in the abstract and concrete parametrized models. In particular, the crux of the proof, Theorem 2, is part of the concrete model. A substantial portion of the instance models is boilerplate definitions and proofs that only vary slightly between the instances. Figure 4.7 gives an overview of the different parts of our framework and the lines of Isabelle/HOL code associated with them.

4.8.4 Consistency of environment assumptions and executability of event system

All instance models are still parametrized by the environment parameters, i.e., the underlying Internet topology defined by *target*, the set of authorized paths $auth_{\mathsf{a}}$, the set of compromised nodes $\mathcal{N}_{attr} \subseteq \mathcal{N}$ and the oracle predicate $\mathcal{O}$. We instantiate these parameters with the topology and authorized paths given in Figure 2.1, $\mathcal{N}_{attr} = \{F\}$, and an $\mathcal{O}$ function that always produces false. We discharge the associated assumptions ASM 1–ASM 6 from §4.5.1 in this example model to show their consistency. Furthermore, we show the executability of the instantiated event system for the EPIC level 1 protocol in the strong attacker model, showing that it is indeed possible to send a packet from a source to a destination, i. e., the model's events can be executed in the correct order.

4.8.5 Unverified data plane security properties

Source and packet authentication

These properties allow for the identification of a packet's origin and in some cases its header and content by ASes or the destination. The challenge in designing protocols that provide these properties is that they require keys between the source and the authenticating entity. As mentioned in the introduction, naïve solutions, such as using public key cryptography per packet, or distributing symmetric keys between each pair of entities are prohibitively inefficient. Hence, protocols often use dynamic key derivation techniques such as DRKey [64, 92] (proposed in OPT [64] and used in

Formalization of framework	**LoC**
Infrastructure (Dolev-Yao, Event System, etc.)	3802
Abstract Model & Network Model	699
Concrete Model (w/o Theorem 2)	852
Theorem 2 for directed setting	638
Theorem 2 for undirected setting	276
Total	**6267**

Formalization of instances	**LoC**
SCION	323
SCION simplified	319
EPIC Level 1 Basic Attacker	397
EPIC Level 1 Strong Attacker	439
EPIC Level 2 Strong Attacker	466
Anapaya-SCION	568
ICING	330
ICING simplified	258
ICING further simplified	267
Executability proof for EPIC Level 1 SA	425
Total	**3792**

FIGURE 4.7: Overview of formalization in Isabelle/HOL.

EPIC [71]) or non-interactive Diffie–Hellman (used in ICING [84]). These scale better, but come with additional trust assumptions or communication overhead. With shared keys in place, authentication by a router or the destination can be easily implemented, modeled, and verified as a single-message two-party protocol. In contrast to network-wide properties like path authorization and detectability, the verification of source and packet authentication does not require any of the special features listed in the introduction. In particular, the set of authorized paths is irrelevant for this property, the number of protocol participants is fixed, and a protocol run does not depend on the length of the path. This makes it feasible to use automated tools such as Tamarin and ProVerif, in which protocol analysis is simpler than in Isabelle/HOL. For these reasons, we exclude source and packet authentication from our verification framework.

Path validation

This property proves to subsequent ASes and the destination that all previous hops on the path embedded in the packet were indeed traversed. While path validation is provided by some architectures [22], including EPIC Level 3-4, there are several reasons why it is less critical than the properties presented above. First, path validation only establishes a lower bound on the set of ASes that have been traversed and does not stop on-path attackers from sending copies of packets to ASes that are not part of the sender's intended path. Second, if there is at most one on-path attacker, then the much simpler packet authentication property is sufficient to imply path validation for the destination. We outline this in §5.1.1. Third, path validation protects honest end hosts against malicious on-path *ASes* that re-route their packets. ASes are legal entities that have business relationships and contracts in defined jurisdictions and could suffer legal consequences when misbehavior is detected. In contrast, the properties presented above defend against malicious *sources* (in some cases, with colluding ASes). Malicious end hosts are a ubiquitous threat to Internet security and legal rulings are often not enforceable.

For these reasons, we do not verify path validation in this work.

4.9 Related work

As mentioned in the introduction, there exists relatively little work on the verification of path authorization for packet forwarding in path-aware internet architectures. We review those works here as well as other research on the verification of secure routing (i. e., path construction) protocols.

4.9.1 Data plane protocols for path-aware architectures

Over the past two decades, several other path-aware architectures have been developed [19, 50, 89, 106]. Several of these use forwarding tables or other state on routers (instead of cryptographic authenticators) to achieve path authorization [50, 106], which does not fit into our framework. Others are not specified in sufficient detail to allow for formal verification [19] or only achieve local properties without considering full path authorization over multiple hops [89]. Finally, some data plane protocols [22], including OPT [64], focus only on source authentication and path validation, neither of which we verify.

4.9.2 Verification of secure data plane protocols

Chen et al. [29] define SANDLog, a Prolog-style declarative language for specifying both data and control plane protocols. They also present an invariant proof rule for SANDLog programs and a verification condition generator, which targets Coq. They verify route authenticity of S-BGP and both route authenticity (in the control plane) and data path authenticity (in the data plane) of SCION. Hence, their coverage of SCION is more comprehensive than ours. However, their data plane property is weaker than our path authorization, as it only guarantees that each traversed hop appears on some authorized path, but does not relate successively traversed hops.

Zhang et al. [107] prove source authentication and path validation properties of the OPT packet forwarding protocols [64]. These properties differ from those that we formally prove in this thesis. They use LS^2, a logic for reasoning about secure systems, in combination with axioms from Protocol Composition Logic (PCL) [35]. They directly embed their logic's axioms and prove the protocols' properties in Coq. As PCL does not have a formal semantics (cf. [33]), the soundness of their approach is questionable. In contrast, we use a foundational approach that only relies on the axioms of higher-order logic and on definitions.

4.9.3 Verification of secure routing protocols

Cortier et al. [7] propose a process calculus for modeling routing protocols, including a model of the network topology and a localized Dolev-Yao adversary. They propose two constraint-based NP decision procedures for analyzing routing protocols for a bounded number of sessions. The first one analyzes a protocol for any network topology, i.e., it decides whether there exists a network topology for which there is an attack on the protocol. The second procedure analyzes a protocol for a given network topology. They also define a logic to express properties such as loop-freedom and route validity. They analyze two ad-hoc routing protocols from the literature. This work is extended to protocols with recursive tests in [6].

Cortier et al. [32] prove a reduction result showing that for proving path validity it is sufficient to consider just five topologies of four nodes. Path validity is similar to our ASM 2 but omitting interfaces. They then analyze two ad-hoc routing protocols using ProVerif.

4.9.4 Verification of network configurations

A different line of research is devoted to the verification of network configurations. Earlier work focused purely on the data plane [61, 63, 67] while more recent work also takes the control plane into account [15, 16, 44, 48, 103]. Verified properties include reachability, isolation, way-pointing, and loop freedom. These works are restricted to a setting with a fixed, concrete network topology and they do not consider security properties.

4.9.5 Parametrization in security protocol verification

Parametrization is a common abstraction technique. It has been used in security protocol verification to achieve the verification of more than one protocol. For instance, Lallemand et al. [69] use parametrization for an abstract realization of channels with different security properties, also in the context of refinement. Schaller et al. [94] employ parametrization to verify physical properties of a number of different wireless protocols.

4.9.6 Exclusive-or in security protocol verification

Automated security protocol verifiers, such as Maude-NPA [42], AKISS [9], and Tamarin [39] have incorporated support for exclusive-or (XOR). Yet it is widely recognized that XOR is a challenging feature to support in security protocol verification. Some verification works abstract from XOR, if the protocol's security properties do not depend on it [36]. This is the approach that we follow for the ICING protocol instance (§4.8.2), which uses additional authenticators that are used to achieve properties that are unrelated to path authorization. However, it cannot be applied to Anapaya-SCION, since Anapaya-SCION's use of XOR is central for achieving its security properties.

Schaller et al. [94, 95] formalize XOR for a Dolev–Yao model in Isabelle/HOL. Their formalization models XOR as a binary operator, and uses normalization to reduce equality in the theory of XOR to syntactic equality on terms.

Escobar et al. [43] provide a generic approach for a wide range of non-subterm-convergent equational theories. Given an equational theory that satisfies the *finite variant property*, they divide it up into a set of oriented equations that can be used as rewrite rules that are safe to perform (for instance, because they are subterm convergent) and into a set of "axioms" that are not. Equality of terms is decided by narrowing along the safe rewrite rules, and comparing the resulting terms modulo axioms. In the case of XOR, which they introduce as an example instance of their generic approach, (A) and (C) are axioms, and (I) and (S) are rewrite rules. Their approach requires carefully defining the set of equations in order to make them *coherent* with the set of axioms. In the case of XOR, this is achieved by adding $x \oplus x \oplus y = y$ to the set of equations. Their approach can be used not only to model XOR, but also to decide the unification problem for the equational theories that fullfil the finite variant property.

5

Discussion and Conclusion

5.1 Discussion

5.1.1 Data plane properties

We distinguish between (i) the path intended by the sender, (ii) the path embedded in the packet header when it reaches the destination and (iii) the actual path that the packet traversed. Figure 5.1 gives an overview of these different kinds of paths from the destination end host's perspective, and how properties relate them.

The equivalence between paths (i)–(iii) is achieved when the destination is provided with packet authentication and path validation. Note that packet authentication alone is sufficient to ensure their equivalence in case there is at most one on-path attacker. By having the destination authenticate a packet (including the embedded path), any change of the embedded path by an on-path attacker will be detected by the destination even without a mechanism that provides path validation. The attacker can only avoid detection by reinstating the original path at a second compromised on-path AS. Only the path in between two attacker nodes can diverge from the sender's intended path. Hence, path validation becomes only relevant for the destination if there are at least two colluding on-path attackers.

5.1.2 Assumed security of control plane

Our formal models and proofs depend on the correctness and the security of the control plane, which we formalized in as six assumptions (for directed protocols) in §4.5.1. Underlying our definition of authorized paths in the concrete model there is an additional assumption: all authorized paths have correct hop validation fields.

This assumption could be violated if the attacker controls beacon servers in the control plane. Consequently, there is a class of potential attacks on

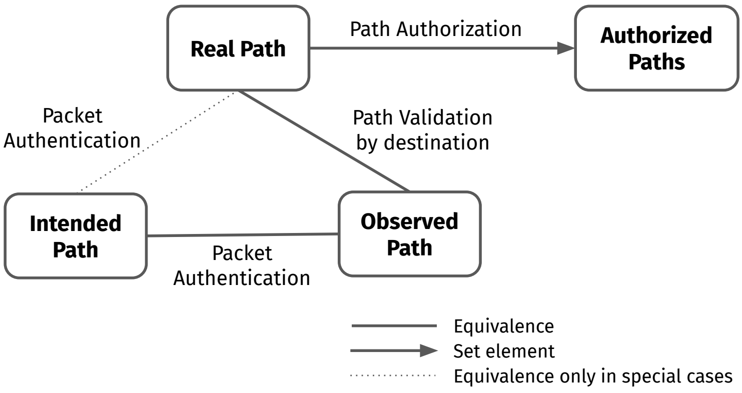

FIGURE 5.1: Relationship between different types of paths. We consider a scenario, where a packet has reached its destination, which is assumed to be honest and to have validated the packet successfully. The dotted property edge is only sufficient to guarantee equivalence when there is at most one on-path attacker.

data plane properties that include both a control- and data plane attacker. We have identified one attack in this class that applies to SCION, which we present below. Similar attacks are most likely also possible in EPIC protocols and in Anapaya-SCION. In undirected protocols, such as ICING, this particular attack does not seem to be feasible, since each hop field's MAC contains the entire path directly, rather than via a nesting of MACs.

We discuss below a number of ways in which this attack can be mitigated.

The beacon MAC switching attack

This attack uses a combination of a control- and data plane attacker. Consider for instance, that in Figure 2.1 on page 16, the beacon server of node D is compromised. Additionally, a malicious end host in node F wants to send data plane packets along the red path. All border routers are assumed to be honest in this scenario.

Normally (without an adversary), the beacon server at AS D would create two hop information fields and two hop validation fields, one of each for the left and the right beacon. We mark the values for the right beacon with a tilde.

$$\begin{aligned} hvf_D &= \mathsf{MAC}_{K_D}(\langle hi_D, hvf_B \rangle) \\ \widetilde{hvf_D} &= \mathsf{MAC}_{K_D}\left(\langle \widetilde{hi_D}, \widetilde{hvf_C} \rangle\right) \end{aligned}$$

In this attack, the control plane attacker at the beacon server of node D processes the right beacon coming from AS C as usual and amends it with its own hop entry (i.e. hop field, and signature). However, for the left beacon, instead of including the appropriate MAC hvf_D alongside hi_D (with interfaces leading to B and E), the compromised beacon server at AS D can include *any* MAC in its hop entry. In particular, it can include $\widetilde{hvf_D}$, i.e., the MAC for the hop field that it adds to the right beacon (i.e., with interfaces leading to C and E and C's MAC as predecessor). The resulting hop field consisting of hi_D and $\widetilde{hvf_D}$ is invalid. However, the signature created by the compromised beacon server is a correct signature over this forged hop field.

The beacon servers at ASes E and F only check the validity of the signature of the hop entry of AS D; they cannot detect that the MAC $\widetilde{hvf_D}$ that AS D included in the beacon is invalid for the hi_D field given in the hop entry. Since the signature is valid, ASes E and F extend the beacon, believing they are extending a beacon corresponding to the left path.

The resulting beacon will have valid signatures, although it is not valid for forwarding. If a packet is sent along the path, MAC validation will fail at the border router of D, which is assumed to be honest. However, a malicious source at AS F can use the resulting beacon to splice its path together with the path given by the right beacon, resulting in the red forwarding path. To do so, the sender creates the unauthorized but valid path consisting of the hop fields hf_F, hf_E, $\widetilde{hf_D}$, $\widetilde{hf_C}$ and $\widetilde{hf_A}$.

Proposed solutions

The border routers sending out packets on an inter-AS link could have special handling for beacons and check that the hop field embedded in the most recent AS entry on the beacon is valid. This defense works unless the border router is also compromised.

However, this solution may not always be desirable. For instance, it could be that two ASes exchange beacons only via a special control plane link, and not over the links over which forwarding occurs. Such a control plane design would be incompatible with our proposed solution. Furthermore, requiring border routers to apply special rules to beacons makes their specification and implementation more complex, and adds potential sources for errors.

Alternatively, each AS receiving a beacon could attempt to send a packet alongside it, before registering it. The above attack produces beacons that do not correspond to valid paths. If the beacon server at AS E sends a packet to D with the beacon's path embedded, and the honest border router at D returns an error, E can discard the beacon.

Fundamentally, the lack of a secure link between the MAC and the signature allows this attack to occur. MACs are opaque to entities that do not have a key, hence they cannot detect if the MAC was created over the correct value. For instance, an AS that extends a given beacon cannot verify that a MAC already embedded in the beacon is correct for the corresponding hop field that is also embedded in the beacon. Crucially, the AS cannot exclude the possibility that this MAC is valid for another hop field that belong to a different forwarding path. The signature does not help, since the compromised beacon server could include a correct signature over the incorrect hop field.

SCION already mandates that ASes regularly rotate their symmetric keys. The protocol could state that they must reveal expired keys. With the keys revealed, any entity can check *a posteriori* the validity of the hop fields signed by ASes. While this does not prevent attacks, it makes attacks on publicly registered beacons visible after a certain amount of time.

Discussion

This attack assumes a compromised beacon server and end host. We note that the attacker can achieve the same consequences as in this combined control- and data plane attack by launching a pure data plane attack that is both trivial to execute and cannot be avoided in the protocol design. In the pure data plane attack, the attacker compromises a router instead of a beacon server, in addition to the sending end host. Then, the compromised router can simply exchange the packet's forwarding path en route. Hence, the combined control- and data plane attack that we have discovered is only practically important in the presence of an attacker that is strong enough to compromise a beacon server, but not capable of compromising a router.

However, our attack is also of theoretical interest, since it raises the question of whether there are other, unknown combined control- and data plane attacks of higher severity. Our formal models, which assume the control plane's security, cannot answer this question. Further research is needed to study this class of attacks and to define properties that express the security of entire networking systems.

5.1.3 Other attacks discovered

We accompanied the development of the SCION Internet architecture as it matured, and turned from an academic project into one with a real-life deployment. In the early prototyping phase, we discovered several attacks. With the exception of the combined control and data plane attack outlined above, all attacks are related to segment switching aspects in the prototype router, for instance missing checks. The most severe attack allowed malicious sources to create arbitrary forwarding paths. This could be used to create forwarding loops (although the number of iterations in the loop is bound by the maximum header size). Due to the relative simplicity of the attacks that we found, we did not implement proof-of-concepts. However, we confirmed the presence of these attacks by studying prototype implementations of the border router and by obtaining feedback from the SCION development team. Fortunately, all bugs found turned out to be easily fixable and were fixed by the SCION development team in the protocol description and implementations.

The attacks relating to segment switching were found early on during the effort to understand the protocol in order to model it precisely. In this early phase of the protocol's development, the specification was lacking detail and the only resource giving a precise specification was the reference prototype implementation, in which some of these bugs were discovered. Due to the lack of a sufficiently precise independent specification at the time, it remains unclear if these should be classified as protocol bugs or as implementation bugs.

As shown by our formal proofs, the cryptographic mechanisms employed in path-aware protocols are secure in principle. However, close attention has to be paid to how these mechanisms are used in entire Internet architectures.

5.2 Conclusion

Several path-aware Internet architectures proposed in recent years promise to improve the security and efficiency of the Internet by providing path control to end hosts. However, this shift of control requires mechanisms to protect the routing policies of ASes from malicious end hosts on the one hand, and raises the challenge of verifying that the path directives were followed by ASes on the other hand. Previous systems for both path authorization and path validation faced a dilemma between security and efficiency in terms of communication overhead.

The highly efficient EPIC protocols proposed in this thesis resolve this dilemma, and furthermore enable all on-path routers and the destination to authenticate the source of a packet. Thus, by ensuring that the source and path of every packet is checked efficiently at the network layer, EPIC enables a wide range of additional in-network security systems like packet filtering for DoS-defense systems and provides a secure foundation for the data plane of a future Internet.

Proposing new protocols is necessary, but not enough. We also need to verify that the claimed security properties actually hold. The verification of future Internet architectures, and in particular path authorization, is a challenging problem, since (i) automated protocol verification tools lack the expressiveness to reason about arbitrary sets of authorized paths and (ii) the relevant protocols are likely to undergo changes before their eventual standardization and widespread deployment. General guarantees for evolving protocols require general specifications and proofs that abstract from the idiosyncrasies of particular protocol instances. Our parametrized framework provides a solution to these challenges. It substantially reduces the per-protocol specification and verification work compared to restarting verification from scratch for each protocol. For each instance, one must only define the parameters and prove the static conditions to establish path authorization and detectability. Our abstractions are general enough to cover a whole class of protocols proposed in the literature.

5.3 Future work

As future work, the framework could be extended to handle SCION's down-segments as well as segment combinations and peering links. Furthermore, security properties such as packet authentication and path validation could be included in the verification framework. Since not all protocols achieve these properties, an interesting question is how they could be incorporated in a modular fashion without requiring separate parametrized models with duplicated proof effort.

As our next step, we plan to verify the Anapaya-SCION router implementation and to use the Igloo methodology [96] to soundly link protocol and code verification. This will require introducing protocol aspects that our models abstract from, such as peering links, segment switching, and buffering of packets.

More generally, it would be interesting to investigate the existence of a reduction result for path authorization and other data plane security properties in the vein of [32].

Bibliography

[1] Martın Abadi and Leslie Lamport. "The Existence of Refinement Mappings". In: *Theor. Comput. Sci.* 82.2 (1991).

[2] Abdallah Alma'aitah and Zine-Eddine Abid. "Transistor level optimization of sub-pipelined AES design in CMOS 65nm". In: *Proceedings of the International Conference on Microelectronics (ICM)*. IEEE. 2011.

[3] Lisa Amini, Anees Shaikh, and Henning Schulzrinne. "Issues with inferring Internet topological attributes". In: *Computer Communications* 27.6 (2004).

[4] Anapaya Systems. *SCION Header Specification*. `https://scion.docs.anapaya.net/en/latest/protocols/scion-header.html`. 2021.

[5] Tom Anderson, Ken Birman, Robert Broberg, Matthew Caesar, Douglas Comer, Chase Cotton, Michael J. Freedman, Andreas Haeberlen, Zachary G. Ives, Arvind Krishnamurthy, William Lehr, Boon Thau Loo, David Mazières, Antonio Nicolosi, Jonathan M. Smith, Ion Stoica, Robbert van Renesse, Michael Walfish, Hakim Weatherspoon, and Christopher S. Yoo. "The NEBULA Future Internet Architecture". In: *The Future Internet*. Springer, 2013.

[6] Mathilde Arnaud, Véronique Cortier, and Stéphanie Delaune. "Deciding Security for Protocols with Recursive Tests". In: *Automated Deduction - CADE-23 - 23rd International Conference on Automated Deduction, Wroclaw, Poland, July 31 - August 5, 2011. Proceedings*. Ed. by Nikolaj Bjørner and Viorica Sofronie-Stokkermans. Vol. 6803. Lecture Notes in Computer Science. Springer, 2011, 49.

[7] Mathilde Arnaud, Véronique Cortier, and Stéphanie Delaune. "Modeling and verifying ad hoc routing protocols". In: *Inf. Comput.* 238 (2014), 30.

[8] Brice Augustin, Xavier Cuvellier, Benjamin Orgogozo, Fabien Viger, Timur Friedman, Matthieu Latapy, Clémence Magnien, and Renata Teixeira. "Avoiding Traceroute Anomalies with Paris Traceroute". In:

Proceedings of the ACM SIGCOMM conference on Internet measurement. 2006.

[9] D. Baelde, S. Delaune, I. Gazeau, and S. Kremer. “Symbolic Verification of Privacy-Type Properties for Security Protocols with XOR”. In: *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*. 2017, 234.

[10] Hitesh Ballani, Paul Francis, and Xinyang Zhang. “A study of prefix hijacking and interception in the Internet”. In: *ACM SIGCOMM Computer Communication Review* 37.4 (2007).

[11] Clemens Ballarin. “Locales: A Module System for Mathematical Theories”. In: *J. Autom. Reason.* 52.2 (2014), 123.

[12] Cristina Basescu, Yue-Hsun Lin, Haoming Zhang, and Adrian Perrig. “High-Speed Inter-Domain Fault Localization”. In: *Proceedings of the IEEE Symposium on Security and Privacy*. 2016.

[13] Cristina Basescu, Raphael M. Reischuk, Pawel Szalachowski, Adrian Perrig, Yao Zhang, Hsu-Chun Hsiao, Ayumu Kubota, and Jumpei Urakawa. “SIBRA: Scalable Internet Bandwidth Reservation Architecture”. In: *Proceedings of the Symposium on Network and Distributed System Security (NDSS)*. 2016.

[14] Tony Bates, Philip Smith, and Geoff Huston. *CIDR Report*. `https://www.cidr-report.org/as2.0/`. 2020.

[15] Ryan Beckett, Aarti Gupta, Ratul Mahajan, and David Walker. “A General Approach to Network Configuration Verification”. In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication, SIGCOMM 2017, Los Angeles, CA, USA, August 21-25, 2017*. ACM, 2017, 155.

[16] Ryan Beckett, Aarti Gupta, Ratul Mahajan, and David Walker. “Abstract interpretation of distributed network control planes”. In: *Proc. ACM Program. Lang.* 4.POPL (2020), 42:1.

[17] Mihir Bellare, Joe Kilian, and Phillip Rogaway. “The security of the cipher block chaining message authentication code”. In: *Journal of Computer and System Sciences* 61.3 (2000).

[18] Kamal Benzekki, Abdeslam El Fergougui, and Abdelbaki Elbelrhiti Elalaoui. “Software-defined networking (SDN): a survey”. In: *Security and Communication Networks* 9.18 (2016).

[19] Bobby Bhattacharjee, Ken Calvert, Jim Griffioen, Neil Spring, and James P. G. Sterbenz. "Postmodern internetwork architecture". In: *NSF Nets FIND Initiative* (2006).

[20] Henry Birge-Lee, Yixin Sun, Anne Edmundson, Jennifer Rexford, and Prateek Mittal. "Bamboozling Certificate Authorities with BGP". In: *Proceedings of the USENIX Security Symposium*. 2018.

[21] Bruno Blanchet. "An Efficient Cryptographic Protocol Verifier Based on Prolog Rules". In: *14th IEEE Computer Security Foundations Workshop (CSFW-14 2001), 11-13 June 2001, Cape Breton, Nova Scotia, Canada*. IEEE Computer Society, 2001, 82.

[22] Kai Bu, Avery Laird, Yutian Yang, Linfeng Cheng, Jiaqing Luo, Yingjiu Li, and Kui Ren. "Unveiling the Mystery of Internet Packet Forwarding: A Survey of Network Path Validation". In: *ACM Comput. Surv.* 53.5 (2020).

[23] R. Bush. *Origin Validation Operation Based on the Resource Public Key Infrastructure (RPKI)*. RFC 7115. 2014.

[24] Hao Cai and Tilman Wolf. "Source Authentication and Path Validation in Networks Using Orthogonal Sequences". In: *Proceedings of the International Conference on Computer Communication and Networks (ICCCN)*. 2016.

[25] Hao Cai and Tilman Wolf. "Source authentication and path validation with orthogonal network capabilities". In: *Proceedings of the IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 2015.

[26] CALOMEL. *AES-NI SSL Performance: a study of AES-NI acceleration using LibreSSL, OpenSSL*. `https://calomel.org/aesni_ssl_performance.html`. 2018.

[27] Kenneth L. Calvert, James Griffioen, and Leonid Poutievski. "Separating routing and forwarding: A clean-slate network layer design". In: *Proceedings of the International Conference on Broadband Communications, Networks and Systems (BROADNETS)*. 2007.

[28] V. Cerf and R. Kahn. "A Protocol for Packet Network Intercommunication". In: *IEEE Transactions on Communications* 22.5 (1974), 637.

[29] Chen Chen, Limin Jia, Hao Xu, Cheng Luo, Wenchao Zhou, and Boon Thau Loo. “A Program Logic for Verifying Secure Routing Protocols”. In: *Logical Methods in Computer Science* Volume 11, Issue 4 (2015).

[30] David D. Clark. *Designing an Internet*. 1st. The MIT Press, 2018.

[31] Danny Cooper, Ethan Heilman, Kyle Brogle, Leonid Reyzin, and Sharon Goldberg. “On the risk of misbehaving RPKI authorities”. In: *Proceedings of the ACM Workshop on Hot Topics in Networks (HotNets)*. 2013, 1.

[32] Véronique Cortier, Jan Degrieck, and Stéphanie Delaune. “Analysing Routing Protocols: Four Nodes Topologies Are Sufficient”. In: *Principles of Security and Trust - First International Conference, POST 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012, Proceedings*. Ed. by Pierpaolo Degano and Joshua D. Guttman. Vol. 7215. Lecture Notes in Computer Science. Springer, 2012, 30.

[33] Cas Cremers. “On the Protocol Composition Logic PCL”. In: *Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security*. Association for Computing Machinery, 2008, 66.

[34] DARPA. *Internet Protocol*. RFC 791. 1981.

[35] Anupam Datta, Ante Derek, John C. Mitchell, and Arnab Roy. “Protocol Composition Logic (PCL)”. In: *Electr. Notes Theor. Comput. Sci.* 172 (2007), 311.

[36] Alexandre Debant and Stéphanie Delaune. “Symbolic Verification of Distance Bounding Protocols”. In: *Principles of Security and Trust*. Ed. by Flemming Nielson and David Sands. Cham: Springer International Publishing, 2019, 149.

[37] Danny Dolev and Andrew Chi-Chih Yao. “On the security of public key protocols”. In: *IEEE Trans. Information Theory* 29.2 (1983).

[38] DPDK Project. *Data Plane Development Kit*. `https://dpdk.org`.

[39] J. Dreier, L. Hirschi, S. Radomirovic, and R. Sasse. “Automated Unbounded Verification of Stateful Cryptographic Protocols with Exclusive OR”. In: *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*. 2018, 359.

[40] ECRYPT. *eBATS: ECRYPT Benchmarking of Asymmetric Systems*. `https://bench.cr.yp.to/results-dh.html`. 2019.

[41] Paul Emmerich, Daniel Raumer, Florian Wohlfart, and Georg Carle. "Assessing soft-and hardware bottlenecks in PC-based packet forwarding systems". In: *ICN* (2015).

[42] Santiago Escobar, Catherine Meadows, and José Meseguer. "Maude-NPA: Cryptographic Protocol Analysis Modulo Equational Properties". In: *Foundations of Security Analysis and Design V: FOSAD 2007/2008/2009 Tutorial Lectures*. Ed. by Alessandro Aldini, Gilles Barthe, and Roberto Gorrieri. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, 1.

[43] Santiago Escobar, Ralf Sasse, and José Meseguer. "Folding variant narrowing and optimal variant termination". In: *The Journal of Logic and Algebraic Programming* 81.7 (2012). Rewriting Logic and its Applications, 898.

[44] Seyed Kaveh Fayaz, Tushar Sharma, Ari Fogel, Ratul Mahajan, Todd D. Millstein, Vyas Sekar, and George Varghese. "Efficient Network Reachability Analysis Using a Succinct Control Plane Representation". In: *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016*. Ed. by Kimberly Keeton and Timothy Roscoe. USENIX Association, 2016, 217.

[45] P. Ferguson. *Best Current Practice 38: Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing*. `https://tools.ietf.org/html/bcp38`. 2000.

[46] Lixin Gao. "On inferring autonomous system relationships in the Internet". In: *IEEE/ACM Transactions on Networking* 9.6 (2001), 733.

[47] Lixin Gao and Jennifer Rexford. "Stable Internet routing without global coordination". In: *IEEE/ACM Transactions on Networking* (2001).

[48] Aaron Gember-Jacobson, Raajay Viswanathan, Aditya Akella, and Ratul Mahajan. "Fast Control Plane Analysis Using an Abstract Representation". In: *Proceedings of the ACM SIGCOMM 2016 Conference, Florianopolis, Brazil, August 22-26, 2016*. Ed. by Marinho P. Barcellos, Jon Crowcroft, Amin Vahdat, and Sachin Katti. ACM, 2016, 300.

[49] Yossi Gilad, Avichai Cohen, Amir Herzberg, Michael Schapira, and Haya Shulman. "Are We There Yet? On RPKI's Deployment and Security." In: *NDSS*. 2017.

[50] P. Brighten Godfrey, Igor Ganichev, Scott Shenker, and Ion Stoica. "Pathlet Routing". In: *Proceedings of ACM SIGCOMM*. 2009.

[51] Timothy G. Griffin and Gordon Wilfong. "An Analysis of BGP Convergence Properties". In: *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. SIGCOMM '99. Cambridge, Massachusetts, USA: ACM, 1999, 277.

[52] Shay Gueron. "Intel® advanced encryption standard (AES) new instructions set". In: *Intel Corporation* (2010).

[53] J. Heather, G. Lowe, and S. Schneider. "How to prevent type flaw attacks on security protocols". In: *Proceedings 13th IEEE Computer Security Foundations Workshop. CSFW-13*. 2000, 255.

[54] Yih-Chun Hu, Adrian Perrig, and David B Johnson. "Wormhole attacks in wireless networks". In: *IEEE journal on selected areas in communications* 24.2 (2006), 370.

[55] Bradley Huffaker, Marina Fomenkov, Daniel J. Plummer, David Moore, and Kimberly Claffy. "Distance metrics in the Internet". In: *Proceedings of the IEEE International Telecommunications Symposium (ITS)*. 2002.

[56] Tetsu Iwata, Junhyuk Song, Jicheol Lee, and Radha Poovendran. *The AES-CMAC Algorithm*. RFC 4493. 2006.

[57] Min Suk Kang, Soo Bum Lee, and Virgil D. Gligor. "The Crossfire Attack". In: *IEEE Symposium on Security and Privacy*. 2013.

[58] Josh Karlin, Stephanie Forrest, and Jennifer Rexford. "Pretty good BGP: Improving BGP by cautiously adopting routes". In: *Proceedings of the IEEE International Conference on Network Protocols*. IEEE. 2006.

[59] Jonathan Katz and Andrew Y. Lindell. "Aggregate Message Authentication Codes". In: *Topics in Cryptology – CT-RSA*. Springer, 2008.

[60] Ethan Katz-Bassett, Colin Scott, David R. Choffnes, Ítalo Cunha, Vytautas Valancius, Nick Feamster, Harsha V. Madhyastha, Thomas E. Anderson, and Arvind Krishnamurthy. "LIFEGUARD: practical repair of persistent route failures". In: *SIGCOMM 2012*. Ed. by Lars Eggert, Jörg Ott, Venkata N. Padmanabhan, and George Varghese. ACM, 2012, 395.

[61] Peyman Kazemian, George Varghese, and Nick McKeown. "Header Space Analysis: Static Checking for Networks". In: *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012*. Ed. by Steven D. Gribble and Dina Katabi. USENIX Association, 2012, 113.

[62] S. Kent, C. Lynn, and K. Seo. "Secure Border Gateway Protocol (S-BGP)". In: *IEEE Journal on Selected Areas in Communications* 18.4 (2000).

[63] Ahmed Khurshid, Xuan Zou, Wenxuan Zhou, Matthew Caesar, and Philip Brighten Godfrey. "VeriFlow: Verifying Network-Wide Invariants in Real Time". In: *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2013, Lombard, IL, USA, April 2-5, 2013*. 2013, 15.

[64] Tiffany Hyun-Jin Kim, Cristina Basescu, Limin Jia, Soo Bum Lee, Yih-Chun Hu, and Adrian Perrig. "Lightweight Source Authentication and Path Validation". In: *Proceedings of the 2014 ACM Conference on SIGCOMM*. Association for Computing Machinery, 2014, 271.

[65] L. Kleinrock. "An early history of the internet [History of Communications]". In: *IEEE Communications Magazine* 48.8 (2010), 26.

[66] Tobias Klenze, Christoph Sprenger, and David Basin. "Formal Verification of Secure Forwarding Protocols". In: *2021 IEEE 34rd Computer Security Foundations Symposium (CSF)*. IEEE, 2021.

[67] Dexter Kozen. "NetKAT – A Formal System for the Verification of Networks". In: *Programming Languages and Systems - 12th Asian Symposium, APLAS 2014, Singapore, November 17-19, 2014, Proceedings*. Ed. by Jacques Garrigue. Vol. 8858. Lecture Notes in Computer Science. Springer, 2014, 1.

[68] Nate Kushman, Srikanth Kandula, and Dina Katabi. "Can You Hear Me Now?! It Must Be BGP". In: *SIGCOMM Comput. Commun. Rev.* 37.2 (2007), 75.

[69] Joseph Lallemand, David Basin, and Christoph Sprenger. "Refining Authenticated Key Agreement with Strong Adversaries". In: *2017 IEEE European Symposium on Security and Privacy (EuroS P)*. 2017, 92.

[70] Taeho Lee, Christos Pappas, Adrian Perrig, Virgil Gligor, and Yih-Chun Hu. "The Case for In-Network Replay Suppression". In: *Proceedings of the ACM Asia Conference on Computer and Communications Security (ASIACCS)*. 2017.

[71] Markus Legner, Tobias Klenze, Marc Wyss, Christoph Sprenger, and Adrian Perrig. “EPIC: Every Packet Is Checked in the Data Plane of a Path-Aware Internet”. In: *29th USENIX Security Symposium (USENIX Security)*. USENIX Association, 2020, 541.

[72] Barry M Leiner, Vinton G Cerf, David D Clark, Robert E Kahn, Leonard Kleinrock, Daniel C Lynch, Jon Postel, Larry G Roberts, and Stephen Wolff. “A brief history of the Internet”. In: *ACM SIGCOMM Computer Communication Review* 39.5 (2009), 22.

[73] M. Lepinski and S. Kent. *An Infrastructure to Support Secure Internet Routing*. RFC 6480. 2012.

[74] Matthew Lepinski and Kotikalapudi Sriram. *BGPsec Protocol Specification*. RFC 8205. 2017.

[75] Qi Li, Yih-Chun Hu, and Xinwen Zhang. “Even rockets cannot make pigs fly sustainably: Can BGP be secured with BGPsec?” In: *Proceedings of the NDSS Workshop on Security of Emerging Networking Technologies (SENT)*. Internet Society. 2014.

[76] Xin Liu, Ang Li, and Xiaowei Yang. “Passport: Secure and Adoptable Source Authentication”. In: 2008.

[77] Gavin Lowe. “A Hierarchy of Authentication Specification”. In: *10th Computer Security Foundations Workshop (CSFW ’97), June 10-12, 1997, Rockport, Massachusetts, USA*. IEEE Computer Society, 1997.

[78] Nancy A. Lynch and Frits W. Vaandrager. “Forward and Backward Simulations: I. Untimed Systems”. In: *Inf. Comput.* 121.2 (1995).

[79] Damien Magoni and Jean-Jacques Pansiot. “Internet topology modeler based on map sampling”. In: *Proceedings of the International Symposium on Computers and Communications (ISCC)*. 2002.

[80] Patrick Maigron. *Autonomous System Number statistics*. `https://www-public.imtbs-tsp.eu/~maigron/RIR_Stats/RIR_Delegations/World/ASN-ByNb.html`.

[81] James McCauley, Yotam Harchol, Aurojit Panda, Barath Raghavan, and Scott Shenker. “Enabling a Permanent Revolution in Internet Architecture”. In: *Proceedings of ACM SIGCOMM*. Beijing, China, 2019.

[82] Simon Meier, Benedikt Schmidt, Cas Cremers, and David A. Basin. “The TAMARIN Prover for the Symbolic Analysis of Security Protocols”. In: *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*. Ed. by Natasha Sharygina and Helmut Veith. Vol. 8044. Lecture Notes in Computer Science. Springer, 2013, 696.

[83] Jad Naous, Michael Walfish, Antonio Nicolosi, David Mazieres, Michael Miller, and Arun Seehra. “Verifying and enforcing network paths with ICING”. In: *Proceedings of the ACM International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*. 2011.

[84] Jad Naous, Michael Walfish, Antonio Nicolosi, David Mazières, Michael Miller, and Arun Seehra. “Verifying and enforcing network paths with ICING”. In: *Proceedings of the 2011 Conference on Emerging Networking Experiments and Technologies, Co-NEXT '11, Tokyo, Japan, December 6-9, 2011*. 2011, 30.

[85] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*. Vol. 2283. Springer, 2002.

[86] NIST. *RPKI Monitor*. `https://rpki-monitor.antd.nist.gov`. 2020.

[87] Larry Paulson. “The inductive approach to verifying cryptographic protocols”. In: *J. Computer Security* 6 (1998).

[88] Adrian Perrig, Pawel Szalachowski, Raphael M. Reischuk, and Laurent Chuat. *SCION: A Secure Internet Architecture*. Springer, 2017.

[89] Barath Raghavan and Alex C. Snoeren. “A system for authenticated policy-compliant routing”. In: *ACM SIGCOMM Computer Communication Review* 34.4 (2004).

[90] Barath Raghavan, Patric Verkaik, and Alex C. Snoeren. “Secure and Policy-Compliant Source Routing”. In: *IEEE/ACM Transactions on Networking* 17.3 (2009).

[91] Benjamin Rothenberger, Daniele E. Asoni, David Barrera, and Adrian Perrig. “Internet Kill Switches Demystified”. In: *Proceedings of the European Workshop on Systems Security (EuroSec)*. 2017.

[92] Benjamin Rothenberger, Dominik Roos, Markus Legner, and Adrian Perrig. “PISKES: Pragmatic Internet-Scale Key-Establishment System”. In: *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*. ASIA CCS '20. Taipei, Taiwan: Association for Computing Machinery, 2020, 73.

[93] Jeffrey Rott. “Intel advanced encryption standard instructions (AES-NI)”. In: *Technical Report, Intel* (2010).

[94] P. Schaller, B. Schmidt, D. Basin, and S. Capkun. “Modeling and Verifying Physical Properties of Security Protocols for Wireless Networks”. In: *2009 22nd IEEE Computer Security Foundations Symposium*. 2009, 109.

[95] Benedikt Schmidt, Patrick Schaller, and David Basin. “Impossibility results for secret establishment”. In: *2010 23rd IEEE Computer Security Foundations Symposium*. IEEE. 2010, 261.

[96] Christoph Sprenger, Tobias Klenze, Marco Eilers, Felix A. Wolf, Peter Müller, Martin Clochard, and David Basin. “Igloo: Soundly Linking Compositional Refinement and Separation Logic for Distributed System Verification”. In: *Proc. ACM Program. Lang.* 4. OOPSLA (2020).

[97] Ahren Studer and Adrian Perrig. “The Coremelt Attack”. In: *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*. 2009.

[98] Yixin Sun, Anne Edmundson, Laurent Vanbever, Oscar Li, Jennifer Rexford, Mung Chiang, and Prateek Mittal. “RAPTOR: Routing Attacks on Privacy in Tor”. In: *Proceedings of USENIX Security Symposium*. 2015.

[99] Craig Timberg. *Net of insecurity. Part I: A flaw in the design*. `https://www.washingtonpost.com/sf/business/2015/05/30/net-of-insecurity-part-1/`. 2015.

[100] Craig Timberg. *Net of insecurity. Part II: The long life of a quick 'fix'*. `https://www.washingtonpost.com/sf/business/2015/05/31/net-of-insecurity-part-2/`. 2015.

[101] Tao Wan, Evangelos Kranakis, and Paul C van Oorschot. “Pretty Secure BGP, psBGP.” In: *NDSS*. 2005.

[102] Cun Wang, Zhengmin Li, Xiaohong Huang, and Pei Zhang. “Inferring the average AS path length of the Internet”. In: *Proceedings of the IEEE International Conference on Network Infrastructure and Digital Content (IC-NIDC)*. 2016.

[103] Konstantin Weitz, Doug Woos, Emina Torlak, Michael D. Ernst, Arvind Krishnamurthy, and Zachary Tatlock. “Scalable Verification of Border Gateway Protocol Configurations with an SMT Solver”. In: *Proc. ACM Program. Lang.* OOPSLA 2016. Amsterdam, Netherlands: Association for Computing Machinery, 2016, 765.

[104] Russ White. “Securing BGP through secure origin BGP (soBGP)”. In: *Business Communications Review* 33.5 (2003), 47.

[105] Bo Wu, Ke Xu, Qi Li, Zhuotao Liu, Yih-Chun Hu, Martin J. Reed, Meng Shen, and Fan Yang. “Enabling Efficient Source and Path Verification via Probabilistic Packet Marking”. In: *Proceedings of the IEEE/ACM International Symposium on Quality of Service (IWQoS)*. 2018.

[106] Xiaowei Yang, David Clark, and Arthur W. Berger. “NIRA: A New Inter-Domain Routing Architecture”. In: *IEEE/ACM Transactions on Networking* (2007).

[107] Fuyuan Zhang, Limin Jia, Cristina Basescu, Tiffany Hyun-Jin Kim, Yih-Chun Hu, and Adrian Perrig. “Mechanized Network Origin and Path Authenticity Proofs”. In: *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2014.

[108] Xin Zhang, Hsu-Chun Hsiao, Geoffrey Hasker, Haowen Chan, Adrian Perrig, and David Andersen. “SCION: Scalability, Control, and Isolation On Next-Generation Networks”. In: *Proceedings of the IEEE Symposium on Security and Privacy*. 2011.