IsaNet: A Framework for Verifying Secure Data Plane Protocols

Tobias Klenze [a], Christoph Sprenger [a,*] and David Basin [a]

[a] *Department of Computer Science, ETH Zurich, Switzerland*

E-mails: tobias.klenze@inf.ethz.ch, sprenger@inf.ethz.ch, basin@inf.ethz.ch

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 problems while simultaneously empowering end hosts to exert some control on their packets' route through the network. In these architectures, autonomous systems authorize forwarding paths in accordance with their routing policies, and protect these paths using cryptographic authenticators. For each packet, the sending end host selects an authorized path and embeds it and its authenticators in the packet 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, which we call *path authorization*, protects the routing policies of autonomous systems from malicious senders.

The fundamental role of packet forwarding in the Internet's ecosystem and the complexity of the authentication mechanisms employed call for a formal analysis. We develop IsaNet, 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. We propose novel attacker models and different sets of assumptions on the underlying routing protocol. We validate our framework by instantiating it with nine concrete protocol variants and prove that they each satisfy the verification conditions (and hence path authorization). The invariants needed for the security proof are proven in the parametrized model instead of the instance models. Our framework thus supports low-effort security proofs for data plane protocols. In contrast to what could be achieved with state-of-the-art automated protocol verifiers, our results hold for arbitrary network topologies and sets of authorized paths.

Keywords: security protocols, formal verification, future Internet, data plane

1. Introduction

The Internet is a global network of ca. 70,000 independently managed networks, called *autonomous systems* (ASes), which are run by entities such as Internet service providers (ISPs), content providers, or public institutions. Routing is based on the aging Border Gateway Protocol (BGP), a protocol that scales poorly and has no built-in security. In response to these well-known problems, the networking community has been working to augment BGP with security mechanisms [1–3], but the proposed solutions have proven to be insufficient, inefficient, or they introduce new problems [4–6].

In parallel to the proposed BGP enhancements, which trade off performance for security, many researchers have acknowledged the need for a clean-slate approach. The networking community has invested substantial effort into developing novel security protocols with the objective of building a new

*Corresponding author. E-mail: sprenger@inf.ethz.ch.

Internet architecture that is both more efficient and more secure. We focus here on *path-aware* architectures [7–14]. In contrast to the current Internet, these provide end hosts with some control over the paths along which they send their packets.

Networking architectures generally consist of a *control plane*, where routers exchange topology information and establish paths, and a *data plane* (also called *forwarding plane* or simply *forwarding*), where packets are forwarded along these paths. In the path-aware Internet architectures that we study, the control plane constructs forwarding paths as sequences of cryptographically authenticated forwarding directives, one for each AS on the path. The source selects such a forwarding path for each packet and includes it in the packet's header, a technique known as *packet-carried forwarding state*. Routers forward the packets according to the forwarding information of their AS, which they extract from the packet's path and validate by checking the associated cryptographic authenticator. Since each packet contains its own forwarding state, routers do not require routing tables unlike with current BGP-based routers.

Path-aware architectures allow end hosts to select paths, but the architectures must also ensure that the policies of ASes are followed. These policies rule out impractical or uneconomical paths. To protect ASes from malicious sources, the control plane should only construct and authenticate paths consistent with the policy of each AS, and the data plane should only forward packets along these paths. The latter property is called *path authorization* and it is the central security property of path-aware data planes. The cryptographic authenticators embedded in the forwarding paths ensure that malicious end hosts cannot tamper with the paths produced by the control plane to craft packets that are forwarded along unauthorized paths.

The complexity of data plane protocols and their central role in a new Internet architecture calls for their formal verification. This is needed for strong guarantees of their correctness and security. It also enables the early detection of protocol flaws and vulnerabilities, avoiding critical exploits and expensive corrections after deployment has begun. This is especially important for the data plane since it will be implemented in large numbers of high-performance software or hardware routers, which are difficult to update after their deployment. Furthermore, a formal proof increases confidence in the architecture's security, thereby fostering its adoption.

Data plane protocols exhibit several characteristics that make the verification of path authorization particularly challenging. First, we want to verify protocols 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 [15] and ProVerif [16] could only be used for bounded verification of path authorization, instead of verification under arbitrary sets of authorized paths.

We instead employ a higher-order logic theorem prover, Isabelle/HOL [17], 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 redo the specification and verification effort from scratch for each protocol or variant. Specifications and proofs should thus share as much structure as possible with each other. To achieve this, we propose a parametrized framework in Isabelle/HOL for the verification of data plane protocols. Since we model network protocols in Isabelle/HOL, we name our framework *IsaNet*. Figure 1 provides an overview of our framework.

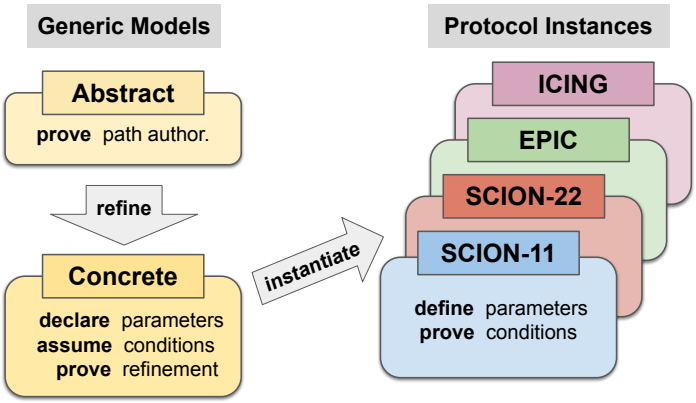


FIGURE 1: Overview of our models. Refinement and instantiation preserve properties.

We follow a refinement approach, in which we formally relate models at different levels of abstraction via simulation relations in order to develop protocols that are secure by construction. The resulting trace inclusion guarantee ensures that properties proven on the abstract models are preserved to more concrete models. We first develop a simple abstract event system model of a packet forwarding protocol. We omit the attacker in this model, which makes it easy to 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) validators* (HV), 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 HV 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 four protocol parameters. For example, we have a parameter representing the cryptographic validator check that must be performed by each AS locally to determine the authorization of the forwarding path. We identify five *verification conditions* (simply called *conditions* below) 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 HV to be unforgeable and to be computed over 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. We support both a standard and a strong attacker model.

To define a concrete data plane protocol in IsaNet, we instantiate our model's protocol parameters and to prove its security, we discharge the associated conditions. We do so for the data plane of the original SCION protocol [13, 18], which we will call *SCION-11* after its year of publication, and its substantially revised version [19, 20], which we will analogously call *SCION-22*. We also verify members of the EPIC protocol family [21], and ICING [22], as well as variants of the above protocols. The instantiations and associated proofs 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).

The results reported on in this paper revise and substantially extend those of [23]. The main extensions are the addition of the SCION-22 protocol instance (§7.3), an algebraic model of exclusive-or (§8.2), and packet header updates by on-path routers (§8.3).

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 other existing verification work. The most closely related works are the verification of a weaker AS-local form of path authorization [24] and of different security properties [25] 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 §10.

Contributions. Our main contributions are as follows. (i) We develop IsaNet, 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. (ii) The five conditions provide insight into the common structure underlying data plane protocols of path-aware Internet architectures. (iii) 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 conditions. (iv) We incorporate many features of proposed protocols, which allows us to verify a wide range of protocols. (v) 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. All results are available online [26].

2. Problem Domain and Overview

In this section, we provide background on secure data planes and give an overview of our framework, in particular the protocols and security properties that we verify.

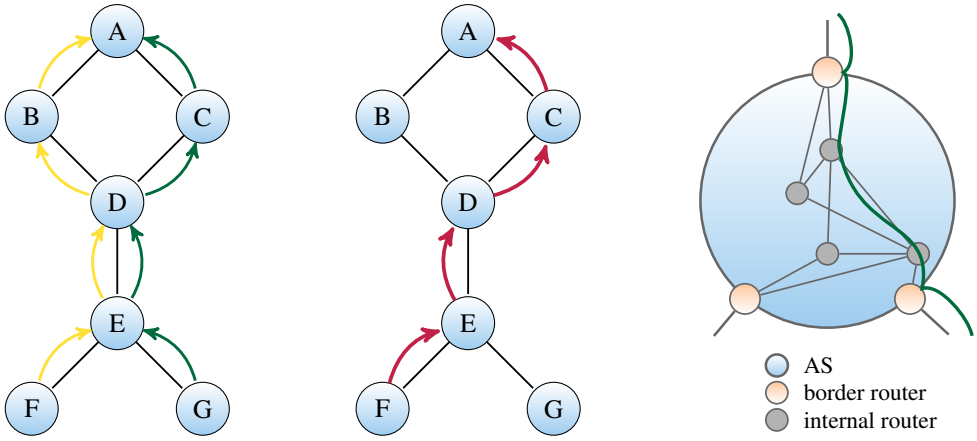
2.1. Motivation for future Internet architectures

Networking in today's Internet is plagued by numerous performance and security problems. Packet 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 [27], and in some topologies, BGP does not converge to a stable state at all [28].

The current Internet not only has these efficiency and scalability limitations, it also lacks security at the networking level. Due to the lack of authentication of packets' sources, adversaries can launch attacks with spoofed source addresses. BGP hijacking attacks, which involve malicious BGP announcements, allow attackers to illegitimately attract 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 [29].

Various protocols have been proposed that add security mechanisms to the existing BGP infrastructure. These include BGPSec [3], S-BGP [1], soBGP [30], psBGP [31], PGBGP [32], and BGP origin validation [2]. Unfortunately, these additions are insufficient to solve the Internet's problems [4–6], or they introduce new problems such as high overhead [33, 34] and kill switches [6]. 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.

Our work instead applies to data planes of a wide range of future Internet architectures that follow a clean-slate path-aware approach [7–14]. We first discuss these data planes generically, and present SCION-11 as an example in §2.5, where we also describe SCION's control plane.



(a) Auth. paths (yellow and green)

(b) Spliced path (red)

(c) Internals of AS E, one path shown.

FIGURE 2: 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). Internal paths in (c) are decided by the AS.

2.2. *Data planes of future Internet architectures*

Each AS in the Internet administers its internal network including its routing and forwarding mechanisms. We abstract from internal forwarding and consider the networking *between* ASes, which requires entities to agree on a common protocol. Since we view the Internet as a network of ASes, we also refer to ASes as *nodes*. See Figure 2 for an example of a (tiny) Internet topology. The internal structure of an AS is shown in Figure 2c. Nodes are interconnected at *border routers*, which sit at the edge of each node's network and perform both inter-AS and intra-AS forwarding.

The path-aware Internet architectures that we examine provide end hosts with *path control*. This means that they can 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 also empowers end hosts to make path choices that are suitable for their applications' needs. For instance, Voice-over-IP requires little bandwidth but low latency, whereas data synchronization requires high bandwidth, but latency is less critical. These applications can thus benefit from using paths with different properties. Moreover, *multipath routing* allows multiple paths between the same source-destination pair to be used simultaneously, even by the same application.

The end host's power to choose paths is balanced against the interest of ASes. Paths are discovered and authorized in the control plane (cf. §2.5), which must ensure that paths are only authorized if they satisfy the routing policies of ASes.

The forwarding paths embedded into packets consist of a *hop field* (HF) for each AS on the path, each of which consists of a *(hop) info field* (HI) and a *(hop) validator* (HV). The info field includes the AS identifier *id*, as well as the interfaces *prev*, on which the packet is received, and *next*, on which the packet is to be sent out. The AS internally forwards the packet between the border routers adjacent to these interfaces. The path is fixed by the sending end host and remains static. A pointer moving through the sequence of hop fields indicates the current hop field. The validator HV, created in the control plane,

proves the authorization of the routing information during forwarding. Upon receiving a data packet, a border router checks the HV of its hop field. If it is valid, then the path was authorized in the control plane, and the border router forwards the packet.

There are two kinds of path authorization mechanisms that differ in how paths are authorized in the control plane: in *undirected* protocols, each AS authorizes the path in its entirety. In *directed* protocols, each AS only authorizes the partial path consisting of its own hop and all subsequent hops in forwarding direction. For instance, in Figure 2a, 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. Undirected protocols requires weaker assumptions (cf. Sections 6.1 and 8.5), but have practical disadvantages compared to directed ones (cf. Appendix B).

In the protocols we study, path authorization still holds even if the malicious sender learns the keys of compromised on-path ASes. However, we must make assumptions to exclude trivial violations of path authorization.

2.3. Security properties that we verify

We verify two data plane security properties (formally defined in §4.4): *path authorization* and *detectability*. They protect ASes against malicious senders and compromised ASes.

Path authorization is the data plane property that all data packets traverse the Internet only along authorized paths. It protects ASes from malicious senders forging paths that are advantageous to themselves (e. g., by using costly paths that these senders have not paid for), detrimental to ASes (e. g., uneconomical *valley* paths [35]), or disrupt forwarding entirely (e. g., through loops).

Path authorization is not just a local property; in particular, it is insufficient for each validator HV to only authenticate the local hop info field HI, 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 only local checks [36]. However, path policies in general cannot be protected just with local checks, as our example in Figure 2a 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 in Figure 2b. Each AS checking only authorizing and checking the local hop information cannot guarantee this.

Detectability states that the actual path that a packet traverses is contained in the path embedded in the packet's header. Since the entire forwarding path is contained in each packet, a malicious source cannot hide its presence on the path. This property does not prevent source spoofing, but rather ensures that *if* a source is spoofed, then the attacker must be in one of the nodes on the packet's forwarding path, thus ensuring basic accountability for data packets.

2.4. 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 and defer their treatment to Appendix C.

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

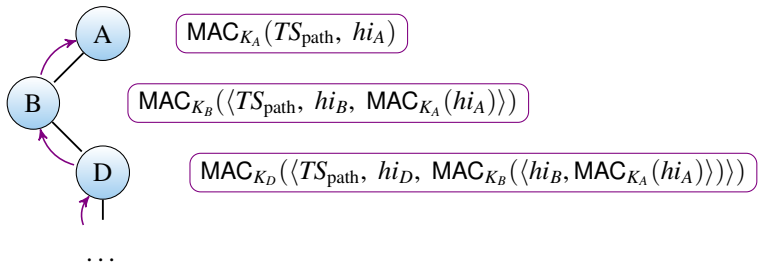


FIGURE 3: Validators of simplified SCION-11 that contain nested MACs. The fields hi_i contain the info fields.

of the property that a path authorized by the control plane is in accordance with the routing policies of all on-path ASes.

2.5. *SCION control and data plane*

We now describe a concrete protocol that implements the control and data plane and achieves path authorization. We use the (simplified) data plane protocol of the SCION architecture as an example.

Authorized paths are established on SCION's control plane using path-discovery *beacons*. Beacons are initialized by a subset of nodes and constructed in the opposite direction of forwarding. Each AS decides which of the beacons it has created or received should be extended by a hop field and propagated to a given neighbor. Since each AS implicitly authorizes the further dissemination by its neighbor, SCION is a directed protocol. Beacons contain two types of authenticators: signatures, which authenticate the beacons themselves in the control plane, and message authentication codes (MACs), which are used to achieve path authorization in the data plane. The signatures are stripped off the beacons before they are embedded into a data plane packet. In the data plane, which transports vastly more packets than the control plane, asymmetric cryptography is too slow to be employed for each packet. Consequently, SCION's data plane relies entirely on symmetric cryptography using a key K_A that is shared by all border routers of AS A.

In the SCION-11 variant that we present here, the validator hv_A is a MAC over A's local forwarding data hi_A, and the validator of the next hop B:

$$hv_A = \mathsf{MAC}_{K_A}(\langle hi_A, hv_B\rangle)\,. \tag{1}$$

Crucially, the MAC is created not only over the local info field, but also over the next MAC. As Figure 3 illustrates, this nests 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. The validation of the MAC is substantially faster, and scales better, than looking up authorized paths in a table on each router [13].

2.6. *Verified data plane protocols*

In §7, we instantiate our parametrized model with SCION-11 and the following protocols, proving that they satisfy path authorization and detectability.

- EPIC [21], 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 source and packet authentication as well as path validation mechanisms, which we do not verify.

$\mathbb{N}$, $\mathbb{B}$	natural numbers, booleans	$(\!\| x = a, y = b \|\!)$	concrete record
$A \times B$	cartesian product	$x(r)$, $r(\!\| x := v \|\!)$	record field x access, update
$\mathcal{P}(A)$, A^*	powerset, finite sequences	$f(x := v)$	function update
$A_\perp$	option set (sum of A and $\{\perp\}$)	$\langle\rangle$, $x \# xs$, $\langle a, b, c\rangle$	empty, cons, concrete seq.
$A \rightharpoonup B$, $A \to B$	partial and total function	$xs \leqslant ys$, $x \in xs$	seq. prefix, seq. membership
$dom(f)$, $ran(f)$	function domain and range	$hd(xs)$, $tl(xs)$	list head and tail if cons, else $\perp$
$(\!\| x \in A, y \in B \|\!)$	set of records	$xs \cdot ys$, $rev(xs)$	seq. concatenation, seq. reversal

TABLE 1: Summary of notation and definitions.

- SCION-22 [19, 20], which uses mutable fields and XOR to accumulate authenticators.
- ICING [22], the undirected data plane protocol in the NEBULA Internet architecture [12]. It also provides path validation, which we do not verify.

3. Preliminaries

In this section, we provide background on event systems, refinement, and model parametrization. We introduce relevant notation in Table 1. Despite our use of Isabelle/HOL, we largely use standard mathematical notation and deliberately blur the distinction between types and sets.

3.1. 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 the 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 $\pi_0(s_\mathrm{c}) \xrightarrow{\pi_1(e_\mathrm{c})}_\mathrm{a} \pi_0(s'_\mathrm{c})$. This is functional forward simulation [37]. Refinement preserves invariants from the abstract to the concrete model, namely, $\mathcal{E}_\mathrm{a} \models P$ implies that $\mathcal{E}_\mathrm{c} \models \pi_0^{-1}(P)$, where $\pi_0^{-1}(P) = \{s \in \mathbb{S}_\mathrm{c} \mid \pi_0(s) \in P\}$.

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

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

	Parameter	Introduced	Description	Used to express	
Environment Parameters used in abstr. & concr. model	tg	Equation (2)	network topology	ASM	COND
	$auth_{\mathrm{a}}$	Equation (4)	authorized paths		
	$\mathcal{N}_{attr}$	Equation (5)	attacker nodes		
Protocol Parameters used in concr. model	ψ	Equation (10)	cryptographic check of HV		
	auth-restrict	Equation (11)	restriction on authorized paths		
	extract	Equation (13)	extracts HI-path from HV		
	ik_0^+	Equation (14)	additional attacker knowledge		

TABLE 2: Our models' parameters. Used in §6 to express assumptions (ASM) and conditions (COND).

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 parameterized channel fields holding sets of messages. For example, the event $\mathbf{send}(A,B,m) : \ dest(m) = B \ \triangleright \ ch(A,B) \mathrel{+}= m$ adds the message m to the channel ch between A and B if m's destination is B, i. e., the intended update is $ch(A,B) := ch(A,B) \cup \{m\}$. There are no other changes to the state, in particular, $ch(A',B') := ch(A',B')$ for all $(A',B') \neq (A,B)$. If the guard $dest(m) = B$ is false, then the event cannot be executed.

3.2. Parametrization

Our models' generality rests on their parametrization. A parametrized model may make assumptions on its parameters. An instance must define the parameters and prove all assumptions. For easy identification, we will highlight parameters in gray when they are first introduced. 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* [38].

4. 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 in Theorem 1. The environment parameters introduced here and the concrete model's protocol parameters are summarized in Table 2.

To distinguish this abstract model, $\mathcal{E}_{\mathrm{a}}$, from the concrete model $\mathcal{E}_{\mathrm{c}}$ (§5) refining it (§6), we use the subscripts 'a' and 'c', respectively. Formally, we will define for $i \in \{\mathrm{a},\mathrm{c}\}$ the event systems $\mathcal{E}_i = (\mathbb{S}_{\mathrm{i}}, s_{\mathrm{i}}^0, E_i, \{\xrightarrow{e}_i\}_{e \in E_i})$. We will define $\mathbb{S}_{\mathrm{i}}$ and s_{i}^0 in the sections below, and Figure 4 will define both E_i, and $\xrightarrow{e}_i$ for all $e \in E_i$.

4.1. 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 $(\mathscr{N}, \mathscr{I}, tg)$, where $\mathscr{N}$ is a set of nodes, $\mathscr{I}$ is a set of interfaces, and tg (*target*) is an environment parameter to our model with the type

$$tg : \mathscr{N} \times \mathscr{I} \rightharpoonup \mathscr{N} \times \mathscr{I}, \tag{2}$$

which models links between ASes. We say that an interface i is *valid* for a node A, if $(A, i) \in dom(tg)$, whereby $tg(A, i) = (B, j)$ 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 about paths in the network, defined in terms of both nodes and their interfaces, rather than the network topology itself. We define a *path* to be a finite sequence of *hop info fields* (called *info fields* and abbreviated HI below) from the set

$$\mathrm{HI} = (\!|\, id \in \mathscr{N},\ prev \in \mathscr{I}_{\perp},\ next \in \mathscr{I}_{\perp} \,|\!). \tag{3}$$

Each info 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 info fields with a cryptographic validator. Since our abstract model does not contain an adversary, such authenticators are not required here.

Our model's second environment parameter is the set of *authorized paths*

$$auth_{\mathrm{a}} \subseteq \mathrm{HI}^{*}, \tag{4}$$

along which packets are authorized to travel. Packets can also traverse just a part of an authorized path. To account for these partial paths, we define $auth_{\mathrm{a}}^{\rightleftarrows}$, the *fragment closure* of $auth_{\mathrm{a}}$, as the set of paths *his* such that there exist a $his' \in auth_{\mathrm{a}}$ and paths $his_1, his_2 \in \mathrm{HI}^{*}$ such that $his' = his_1 \cdot his \cdot his_2$.

We will later introduce an assumption that neighboring hop fields in authorized paths must point to valid links in the network topology defined by the function tg. However, there is an exception for the interfaces of neighboring compromised ASes, since attackers in the control plane can arbitrarily set these interface fields and disseminate beacons out-of-band. Such *wormholes* [39] are unavoidable in routing protocols. We introduce our control plane assumptions in §6.1.

Our third parameter is the set of *compromised* nodes (also called *attacker* or *adversary* nodes)

$$\mathscr{N}_{attr} \subseteq \mathscr{N}. \tag{5}$$

All other nodes are called *honest*. This environment parameter only becomes relevant after introducing the adversary in the concrete model (§5.3), 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 (§6.1).

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

$fut(m) \in auth_{\mathrm{a}}^{\rightleftarrows} \wedge hist(m) = \langle\rangle$

▹ $int(A) \mathrel{+}= m$

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

$fut(m) \in auth_{\mathrm{a}}^{\rightleftarrows} \wedge hist(m) = \langle\rangle \wedge (A, i) \in dom(tg)$

▹ $ext^{\mathrm{send}}(A, i) \mathrel{+}= m.$

send$_{\mathrm{a}}(A, m, hi, i)$:

$hi = hd(fut(m)) \wedge hi \neq \bot \wedge A = id(hi) \wedge$

$i = next(hi) \wedge m \in int(A) \wedge (A, i) \in dom(tg)$

▹ $ext^{\mathrm{send}}(A, i) \mathrel{+}= fwd_{\mathrm{a}}(m).$

recv$_{\mathrm{a}}(A, m, hi, i)$:

$hi = hd(fut(m)) \wedge hi \neq \bot \wedge A = id(hi) \wedge$

$m \in ext^{\mathrm{recv}}(A, i) \wedge (A, i) \in dom(tg)$

▹ $int(A) \mathrel{+}= m.$

deliver$_{\mathrm{a}}(A, m, hi)$:

$fut(m) = \langle hi \rangle \wedge A = id(hi) \wedge m \in int(A)$

▹ $int(A) \mathrel{+}= fwd_{\mathrm{a}}(m).$

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

$m \in DY(ik) \wedge hist(m) = \langle\rangle$

▹ $int(A) \mathrel{+}= m.$

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

$m \in DY(ik) \wedge hist(m) = \langle\rangle \wedge (A, i) \in dom(tg)$

▹ $ext^{\mathrm{send}}(A, i) \mathrel{+}= m.$

send$_{\mathrm{c}}(A, m, hf, i)$:

$hf = hd(fut(m)) \wedge hf \neq \bot \wedge A = id(hf) \wedge$

$i = next(hf) \wedge m \in int(A) \wedge (A, i) \in dom(tg) \wedge$

$\psi(hf, hd(tl(fut(m))), tok(m))$

▹ $ext^{\mathrm{send}}(A, i) \mathrel{+}= fwd_{\mathrm{c}}(m).$

recv$_{\mathrm{c}}(A, m, hf, i)$:

$hf = hd(fut(m)) \wedge hf \neq \bot \wedge A = id(hf) \wedge$

$m \in ext^{\mathrm{recv}}(A, i) \wedge (A, i) \in dom(tg) \wedge$

$i = prev(hf) \wedge \psi(hf, hd(tl(fut(m))), tok(m))$

▹ $int(A) \mathrel{+}= m.$

deliver$_{\mathrm{c}}(A, m, hf)$:

$fut(m) = \langle hf \rangle \wedge A = id(hf) \wedge m \in int(A) \wedge$

$\psi(hf, \bot, tok(m)) \wedge$

▹ $int(A) \mathrel{+}= fwd_{\mathrm{c}}(m).$

FIGURE 4: Events of the abstract (left) and concrete (right) model, with differences highlighted.

4.2. State

We model packet forwarding from a node's internal network to an inter-node link, and vice-versa, using two types of channels: *internal channels* (one per node) and *external channels* (two per interface-node pair, one in each direction). We model these channels as sets of packets. We will define our forwarding events as reading from and placing new packets into channels, but without removing the existing packets. Hence, we model asynchronous communication with message reordering and replay.

We will define packets ($\mathrm{PKT}_{\mathrm{a}}$) below, and first define the state as

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

In the initial state s_a^0, all channels are empty. We overload the set membership operator to apply to states: A packet m is in a state s, written $m \in s$, iff $m \in ran(int(s)) \cup ran(ext(s))$. For a valid interface i of A with $tg(A,i) = (B,j)$, we define $ext^{\text{send}}(A,i) = ext(A,i,B,j)$ and $ext^{\text{recv}}(A,i) = ext(B,j,A,i)$.

We define packets without their payload, only consisting of the forwarding state and a history:

$$\text{PKT}_a = (\!|\, past \in \text{HI}^*,\ fut \in \text{HI}^*,\ hist \in \text{HI}^* \,|\!).$$

A packet consists of the desired future path *fut*, and the (presumed) traversed path *past*, stored in the 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. Additionally, each packet records a path *hist*, also in the reverse direction, which represents its actual trajectory. This can be seen as an auxiliary *history variable* [40], meaning that it is not part of the protocol, but serves to specify and prove properties of protocol executions.

4.3. Events

The events of the abstract model are given on the left-hand side of Figure 4. The life cycle of a packet is captured by the following events: **dispatch-int**$_a$ creates a new packet containing an authorized future path in the internal channel of a node. The packet is transferred with alternating **send**$_a$ and **recv**$_a$ 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**$_a$. The events **dispatch-int**$_a$ and **deliver**$_a$ model the interaction with end hosts, whereas **send**$_a$ and **recv**$_a$ represent the border routers' packet forwarding actions. The additional **dispatch-ext**$_a$ 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**$_a$ and **dispatch-ext**$_a$ events create a new packet with an authorized future path and insert 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 packet's origin. The **send**$_a$ and **recv**$_a$ events both use the current info 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**$_a$ event transfers a packet from the external channel at (A,i) to A's internal channel. The **send**$_a$ event takes a packet m from the internal channel and places the transformed packet $fwd_a(m)$ on the external channel at (A,i). The partial function $fwd_a : \text{PKT}_a \rightharpoonup \text{PKT}_a$ moves the current info field of m into the past path and adds it to the history.

$$fwd_a(m) = (\!|past = hd(fut(m)) \# past(m),\ fut = tl(fut(m)),\ hist = hd(fut(m)) \# hist(m)|\!).$$

This function is only defined when $fut(m) \neq \langle\rangle$. We define head $hd : \text{HI}^*_\perp \to \text{HI}_\perp$ and tail $tl : \text{HI}^*_\perp \to \text{HI}^*_\perp$ functions by $hd(x \# xs) = x$ and $tl(x \# xs) = xs$ and by mapping $\langle\rangle$ and $\perp$ to $\perp$ in both functions.

The **deliver**$_a$ event models delivering a packet m containing a single info field in its future path to an end host. Since we do not explicitly model end hosts and their state, we simply add the packet $fwd_a(m)$ to the internal channel of the AS and thereby push the last info field into the *past* and *hist* paths.

4.4. 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,

$$\textbf{path-auth} = \{s \in \mathbb{S}_{\mathrm{a}} \mid \forall m \in s.\ rev(hist(m)) \cdot fut(m) \in auth_{\mathrm{a}}^{\rightleftarrows}\}. \tag{6}$$

We formalize *detectability*: all traversed hops are recorded on (i.e., a prefix of) the past path,

$$\textbf{detectability} = \{s \in \mathbb{S}_{\mathrm{a}} \mid \forall m \in s.\ hist(m) \leqslant past(m)\}. \tag{7}$$

Theorem 1. *The abstract model satisfies path authorization and detectability, i.e.,* $\mathcal{E}_a \models$ **path-auth** *and* $\mathcal{E}_a \models$ **detectability**.

Proof. The proofs in this abstract model are straightforward. For path authorization, new packets are required to have an authorized future path and an empty history and for existing packets, $rev(hist(m)) \cdot fut(m)$ remains invariant during their forwarding. The *past* path is irrelevant for path authorization. Detectability is independent of $auth_{\mathrm{a}}$ and follows directly from the events' definitions. □

5. 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 (§5.3). We present the concrete model's events in §5.4 and the refinement in §6.

The concrete model retains the environment parameters of the abstract model (§4.1), and adds four *protocol parameters*, which we introduce below. One of them is the cryptographic check that ASes apply to their validators, which allows us to abstract from the concrete cryptographic mechanism used.

Our presentation will focus on path authorization, specifically in the directed setting. We defer the treatment of undirected path authorization to §8.5. We compare both classes of protocols in Appendix B.

5.1. 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 $\mathsf{MAC}_k(m) = \mathsf{H}(\langle k, m \rangle)$. IsaNet also supports encryption and signatures, which we do not use here.

Hop fields (HF), used in the concrete model, extend the (hop) info fields (HI), used in the abstract model, with a cryptographic *(hop) validator* (HV) that authenticates the info field.

$$\mathrm{HF} = (\!|\, id \in \mathscr{N},\ prev \in \mathscr{I}_{\perp},\ next \in \mathscr{I}_{\perp},\ \mathrm{HV} \in \mathbb{T} \,|\!). \tag{8}$$

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 info fields by dropping HV and we lift it element-wise to paths. To keep our notation succinct, we write $\overline{hf_A}$ and $\overline{hfs}$ to denote the application of *abstr-hf*.

We next define concrete packets as follows:

$$\mathrm{PKT_c} = (\!|\, tok \in \mathbb{T},\ past \in \mathrm{HF}^*,\ fut \in \mathrm{HF}^*,\ hist \in \mathrm{HI}^* \,|\!). \tag{9}$$

The past and future paths are sequences of 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 $\mathbb{S}_\mathrm{c}$ has the same record structure as the abstract $\mathbb{S}_\mathrm{a}$, but the channels now carry concrete packets. The initial state s_c^0 is defined similarly to s_a^0 as the empty channel state.

We define the overloaded function *terms* for hop fields as the projection to HV, for paths as the union of *terms* for all hop fields on the path, and for packets as $terms(pkt) = \{tok(pkt)\} \cup terms(past(pkt)) \cup terms(fut(pkt))$. For a set T of terms and for a path or packet x, we write $x \in T$ for $terms(x) \subseteq T$. We will leave the conversion of HI fields to terms implicit below.

5.2. Protocol parameters and authorized paths

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

$$\psi : \mathrm{HF} \times \mathrm{HF}_\perp \times \mathbb{T} \to \mathbb{B}, \tag{10}$$

which each border router performs to check the validity of its hop field. This parameter abstracts the cryptographic structure of the validator, 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 $\Psi : \mathrm{HF}^* \times \mathbb{T} \to \mathrm{HF}^*$, which we apply to the future path *hfs* 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 (§6.3) to truncate the path at the first invalid hop field. As demonstrated by our successful refinement, this does not reduce the system's possible behavior. This is because 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 can restrict the set of concrete authorized paths to incorporate assumptions on the control plane. The second parameter of our model is a predicate over a given concrete path and *tok* field.

$$\textit{auth-restrict} : \mathrm{HF}^* \times \mathbb{T} \to \mathbb{B}. \tag{11}$$

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:

$$auth_c(u) = \{hfs \mid \Psi(hfs, u) = hfs \wedge auth\text{-}restrict(hfs, u) \wedge \overline{hfs} \in auth_a\}. \tag{12}$$

We overload $auth_c$ and define the set $auth_c \subseteq \mathrm{HF}^*$ as the union of $auth_c(u)$ over all u.

Protocols use the HV to protect the future (abstract) path. The third protocol parameter is

$$extract : \mathbb{T} \to \mathrm{HI}^*, \tag{13}$$

which is intended to extract this path from a given HV. For instance, consider the SCION-11 path given in Figure 3. The validator consists of a MAC over the hop's info field and the next hop's HV, allowing for a recursive extraction. Concretely, *extract* would be defined such that $extract(hv_D) = \langle hi_D, hi_B, hi_A \rangle$. 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 $extract(hf) = extract(\mathrm{HV}(hf))$ and to paths by defining $extract(\langle\rangle) = \langle\rangle$ and $extract(hf \# hfs) = extract(hf)$. In §6.2, we will define a 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 additional cryptographic terms

$$ik_0^+ \subseteq \mathbb{T}, \tag{14}$$

given to the attacker in the definition of the intruder knowledge below. The parameters ik_0^+ and *auth-restrict* do not play an important role in the concrete model. We will return to them in the instances, where they allow modeling control plane assumptions and protocol-dependent intruder knowledge.

5.3. Attacker model

We model a Dolev–Yao adversary who can eavesdrop on and inject new packets in all *int* and *ext* channels, but only has access to the keys of compromised nodes. We first define the attacker's message derivation capabilities, which are used in the attacker events introduced in §5.4.

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 [41] 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 5. 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 (e. g., interfaces) and closes H under the construction of sequences and hashes.

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

$$ik_0 = \bigcup \{terms(x) \cup \{u\} \mid x \in auth_c(u)\} \cup \{K_i \mid i \in \mathcal{N}_{attr}\}, \tag{15}$$

$$ik(s) = ik_0 \cup ik_0^+ \cup \bigcup_{m \in s} terms(m). \tag{16}$$

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

$$\frac{t \in H}{t \in DY^{\uparrow}(H)} \qquad \frac{t \in \mathscr{N} \cup \mathscr{I}_{\perp} \cup \mathbb{N}}{t \in DY^{\uparrow}(H)} \qquad \frac{t \in DY^{\uparrow}(H)}{\mathsf{H}(t) \in DY^{\uparrow}(H)} \qquad \frac{t_1 \in DY^{\uparrow}(H) \ \cdots \ t_n \in DY^{\uparrow}(H)}{\langle t_1, \ldots, t_n \rangle \in DY^{\uparrow}(H)}$$

FIGURE 5: Rules for Dolev–Yao message decomposition ($DY^{\downarrow}$) and composition ($DY^{\uparrow}$).

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

5.4. Events

Each event of the abstract model is refined into a similar event of the concrete model (Figure 4, right). The concrete model retains the packet life-cycle of the abstract model (§4.3). The **dispatch-int**$_c$ and **dispatch-ext**$_c$ events can send arbitrary attacker-derivable packets, instead of just packets containing an authorized path like 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 events in more detail.

5.4.1. Attacker events

The two attacker events **dispatch-int**$_c$ and **dispatch-ext**$_c$ model that the attacker is active and can send a packet on an internal or external channel of any AS. This is 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. However, its validators and packet token field must be derivable from the intruder knowledge, i. e., $terms(m) \subseteq DY(ik(s))$. In the events' guards, we write ik and omit the state. 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 $\langle\rangle$. We do this to exclude attacks where the 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, suppose that the attacker in Figure 2a 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 info 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 *tg* function because the interface identifier *prev* of the next hop points to the link between it and the packet's source AS.

5.4.2. Honest events

To secure the protocol against the attacker introduced in this model, honest events now perform two validation checks. First, upon receiving a packet from another node, $\mathbf{recv}_\mathrm{c}$ includes the guard $i = \mathit{prev}(\mathit{hf})$ to check that the 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 validator using $\psi(\mathit{hf}, \mathit{hf}', u)$, 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 subsequent path are authorized. The events $\mathbf{send}_\mathrm{c}$ and $\mathbf{deliver}_\mathrm{c}$ use the function fwd_c to forward a packet, which is defined similarly to fwd_a. The tok field is not modified by fwd_c, and the current hop field is converted from HF to HI using $\overline{\cdot}$ before it is added to $\mathit{hist}(m)$.

5.4.3. Constant intruder knowledge

The dispatch events only add packets to the network that are derivable from the intruder knowledge. Furthermore, all other events only add packets that already exist in the network or can be easily derived from them using fwd_c. Hence we can prove inductively that all packets in the network are already derivable from the initial intruder knowledge, which is known to the attacker in the initial state.

More precisely, under the Dolev–Yao closure DY, the intruder knowledge $\mathit{ik}(s)$ for reachable states s and the initial intruder knowledge are identical. We prove this as an invariant:

Lemma 1. *For all reachable states s, $\mathit{DY}(\mathit{ik}(s)) = \mathit{DY}(\mathit{ik}_0 \cup \mathit{ik}_0^+)$.* □

Since the attacker does not learn any new messages during the protocol execution, we can drop the state-dependent part of the intruder knowledge under DY, which greatly simplifies reasoning about the intruder.

6. Refinement

We prove that the concrete event system refines the abstract one. Our proof rests on several global *assumptions* (ASM) about the control plane and on a set of *conditions* (COND) on the authentication mechanism used. These will be defined later in this section, as are the refinement mappings π_0 and π_1.

Theorem 2. *$\mathcal{E}_c$ refines $\mathcal{E}_a$ under the refinement mappings $\pi_0 : \mathbb{S}_c \to \mathbb{S}_a$ on states and $\pi_1 : E_c \to E_a$ on events, assuming ASM and COND.*

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 discharging the associated conditions, which we do for several protocols in §7.

We first introduce the assumptions and conditions and then define the refinement mappings. Finally, we show the interesting cases in the attacker events' refinement, which is the crux of Theorem 2's proof.

6.1. Control plane assumptions

We define two types of environment assumptions about the authorized paths auth_a constructed by the control plane. First, we assume the correct functioning of the control plane, which is independent of the data plane. Second, we make assumptions about the control plane attacker's behavior in order to exclude trivial attacks on the routing policies of colluding ASes. These provide upper and lower bounds on the set auth_a, respectively.

6.1.1. Correctness assumptions

We assume that authorized paths are *interface-valid*: interfaces of adjacent info fields on a path point to the same link, except when both hop fields belong to attacker nodes. With this exception, we account for unavoidable out-of-band communication by adversaries, so-called *wormholes* [39]. To formalize interface validity, we introduce the predicate $\phi : \mathrm{HI} \times \mathrm{HI}_\perp \to \mathbb{B}$, which checks the validity of two adjacent info fields. In the following, we let hi_A (respectively hi_B) denote an info field for which $id(hi_A) = A$ (respectively $id(hi_B) = B$). Given the current info field hi_B and a preceding info field hi_A, we define $\phi(hi_B, hi_A) = (tg(A, next(hi_A)) = (B, prev(hi_B))) \vee (A \in \mathscr{N}_{attr} \wedge B \in \mathscr{N}_{attr})$. If there is no previous hop field, no interface must be checked, i.e., $\phi(hi_B, \perp) = \mathsf{true}$.

We define a function $\Phi : \mathrm{HI}^* \times \mathrm{HI}_\perp \to \mathrm{HI}^*$ that returns the longest interface-valid prefix of a sequence *his*. Concretely, $\Phi(his, hi_{prev})$ 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 info 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

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

We furthermore assume that authorized paths are *terminated*: the first info field's *prev* is $\perp$ and the last info field's *next* is $\perp$, except for when the respective info field belongs to the attacker. ASM 1 and ASM 2 formalize the correctness of the control plane.

ASM 1: Interfaces valid: All paths $his \in auth_{\mathrm{a}}$ are interface-valid, namely $\Phi(his) = his$.

ASM 2: Terminated: An info field hi with $id(hi) \notin \mathscr{N}_{attr}$ on $hi \# his \in auth_{\mathrm{a}}$ (resp. on $his \cdot \langle hi \rangle \in auth_{\mathrm{a}}$) has $prev(hi) = \perp$ (resp. $next(hi) = \perp$).

6.1.2. Assumptions to rule out artificial attacks

Recall that path authorization protects ASes' routing policies from malicious end hosts. This property only holds for the policies of *honest* ASes. We must exclude artificial attacks, in which the sender violates the policies of *compromised* ASes.

The honest ASes' routing policies govern the dissemination of beacons, and hence the creation of authorized paths. In the directed setting, an honest AS A sending a beacon to a neighbor AS B implicitly consents that B and all subsequent ASes may extend the path at their discretion. This is in contrast to the undirected setting, which requires all ASes to explicitly agree on the entire path.

The *data planes* of directed protocols allow the attacker to perform the following actions, which at first glance seem to violate path authorization: a malicious AS on a path discovered by a beacon can arbitrarily extend the path with additional compromised ASes in the data plane by crafting additional validators using the compromised keys – even if the extended paths are not authorized in the control plane. However, the *control planes* of the directed setting allow *all* ASes to similarly disseminate beacons that they receive and the resulting extended paths can be considered as authorized by the mentioned implicit consent. Hence, the "malicious path extensions" in the data plane are artificial attacks that have the same outcome as legitimate control plane actions, and that furthermore do not violate the policy of any honest AS. We thus assume that the paths extended in this way are authorized as well.

For example, assume that ASes E and F in Figure 2 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 HV 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 other on-path nodes, since each AS implicitly authorizes path extensions by its customer ASes. Hence, this does not impact the policy of any honest AS.

We characterize the attacker's possible and permitted behavior via the following assumptions. First, taking into account the path reversal of the data plane, the extension of paths becomes: prepending a compromised AS to an authorized path starting with an attacker results in an authorized path (ASM 4). We furthermore consider authorized paths shortened by the attacker to be authorized in ASM 5, allow the attacker who is first on a path to change its info field in ASM 6, and assume that the empty path and singleton paths consisting of an attacker are authorized in ASM 3.

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

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

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

ASM 6: Modify: If the first HI field belongs to the attacker, she can modify its *prev*. Formally, if $hi_A \# his \in auth_{\mathrm{a}}$, $next(hi'_A) = next(hi_A)$, and $A \in \mathcal{N}_{attr}$ then $hi'_A \# his \in auth_{\mathrm{a}}$. 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 require that the entire path is authorized by each on-path AS. As we show in §8.5, ASM 3–ASM 6 can then be replaced by weaker assumptions.

Note that all these assumptions are only required since we assume a very strong attacker model where the end host attacker colludes with on-path ASes. We emphasize that 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 the above closure assumptions are not needed.

6.2. Conditions on authentication mechanisms

We define five conditions that relate the protocol parameters ψ, *auth-restrict*, *extract*, and ik_0^+ introduced in Equations (10), (11), (13), and (14) with each other and with the environment parameters $\mathcal{N}_{attr}$ and $auth_{\mathrm{a}}$ (via $auth_{\mathrm{c}}$). These conditions are used in the refinement proof in §6.4. We will need 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 $auth_{\mathrm{c}}$. 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 $\exists hfs \in auth_{\mathrm{c}}.\ hf \in hfs$.

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 cover them.

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*.

COND 3: Path prefix of extract: $\overline{\Psi(hfs, u)} \leqslant extract(hfs)$.
COND 4: Extract prefix of path: If $\Psi(hfs, u) = hfs$ and $auth\text{-}restrict(hfs, u)$, then $extract(hfs) \leqslant \overline{hfs}$.

Finally, COND 5 requires the HV 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 $u' = u$.

6.3. Refinement mappings

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

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

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 $to_a(m, \perp)$. For packets in *ext* channels (A, i, B, j), we must check that the first info 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 info field does not have the correct interface and *id*, then the packet's future path is mapped to $\langle\rangle$.

The refinement mapping $\pi_1 : E_c \to E_a$ maps each event on the right side of Figure 4 to the corresponding event on the left side, where parameters are transformed using to_a and $\overline{\cdot}$.

6.4. 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., $\pi_0(s_c^0) = s_a^0$. This holds trivially, since neither initial state contains any packets.

We then show that each concrete event e can be matched 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 3 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.

6.4.1. Lemmas

We prove two lemmas that are helpful for the attacker refinement proof below. The first lemma states that a valid path that satisfies *auth-restrict* coincides with its extraction (modulo the projection $\overline{\cdot}$). The second lemma asserts that the valid prefix of any extension of an attacker-extractable hop field is authorized.

Lemma 2. *If $\Psi(hfs) = hfs$ and $auth\text{-}restrict(hfs)$, then $extract(hfs) = \overline{hfs}$.*

Proof. By COND 3 and COND 4. □

Lemma 3. *Suppose $hf \in DY^{\downarrow}(ik_0 \cup ik_0^{+})$ for a 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 by ASM 3.

Otherwise, we can apply COND 2 and obtain hfs', hfs_0, hfs_1, and hfs_2 such that $hfs' \in auth_{\mathrm{c}}$, $hfs' = hfs_0 \cdot hfs_1$, and $hfs_1 = hf \# hfs_2$. Since $hfs' \in auth_{\mathrm{c}}$, $\Psi(hfs') = \overline{hfs'}$ 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 $extract(hf) \in auth_{\mathrm{a}}^{\rightleftarrows}$. Finally, from COND 3, we have $\overline{\Psi(hf \# hfs)} \leqslant extract(hf)$. Since $auth_{\mathrm{a}}^{\rightleftarrows}$ is closed under prefixing, $\overline{\Psi(hf \# hfs)} \in auth_{\mathrm{a}}^{\rightleftarrows}$. □

6.4.2. Attacker refinement proof

Theorem 3. *If a packet m is derivable by the attacker, i.e., $m \in DY(ik_0 \cup ik_0^{+})$, then the interface- and cryptographically-valid prefix of its future path is authorized, i.e., for all hi_{prev}, $\Phi(\overline{\Psi(fut(m))}, hi_{prev}) \in auth_a^{\rightleftarrows}$.*

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

- $A \notin \mathcal{N}_{attr}$: In this case, the attacker can derive hf_A without K_A. Then by COND 1 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 2). 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 therefore 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 info 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 1). 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 §6.1. By the induction hypothesis $\Phi(\overline{\Psi(hf_B \# hfs')},\ hi_{prev}) \in auth_{\mathrm{a}}^{\rightleftarrows}$. By the 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, the suffix without his_{pre} is also in $auth_{\mathrm{a}}$. 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). □

7. 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. Since refinement and instantiation preserve properties, the path authorization and detectability security properties proven for the abstract model also hold for these instance models.

7.1. SCION-11

In SCION-11, hv_A for a hop A without successor is $hv_A = \mathsf{MAC}_{K_A}(hi_A)$, where K_A is a key shared by all border routers in A. If there is a next hop B, then hv_A also includes its info and validator field of B,

$$hv_A = \mathsf{MAC}_{K_A}(\langle hi_A, hi_B, hv_B \rangle)\,. \tag{17}$$

We instantiate ψ with the check of Equation (17) and set $ik_0^+ = \emptyset$. In SCION-11, the tok field is not used. We simply set $\mathit{auth\text{-}restrict}(hfs, u) = (u = 0)$ (where 0 is a term representing the natural number "0") to ensure that this field does not leak any new terms that the intruder cannot otherwise derive. In all instances, we only define $extract$ for valid patterns and all other patterns are mapped to $\langle\rangle$.

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

To show that this model of SCION-11 inherits the security properties proven in the parametrized models, we prove the parametrized model's conditions. First, we observe that the intruder knowledge only contains keys and MACs, which cannot be decomposed, and 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, $auth_{\mathrm{c}}$, and ψ. We establish COND 3 and COND 4 by routine inductions over hfs.

Variants. By dropping hi_B from Equation (17) we obtain the variant described in §2.5 where we instantiate ψ with the check given by Equation (1). The proof of the conditions is almost identical.

7.2. EPIC

EPIC Level 1 uses a *hop authenticator* σ (which is a static authenticator almost identical to Equation (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 origin of each packet is unique.

We define the validator hv_A and the values σ_A, S_A and V_A as follows:

$$hv_A = \langle S_A, V_A \rangle, \quad \sigma_A = \mathsf{MAC}_{K_A}(\langle hi_A, S_B \rangle), \quad V_A = \mathsf{MAC}_{\sigma_A}(tok), \quad S_A = \mathsf{H}(\sigma_A).$$

If A has no successor, then $\sigma_A = \mathsf{MAC}_{K_A}(hi_A)$. In the EPIC protocol specification, S_A is defined to be the first few bytes of σ_A. Here, we model truncation as a hash function. We discuss this and other differences between protocols and their models in §9.2.

End hosts first obtain the public hop authenticators for a path. For each packet they compute the timestamp offset, embed it in the *tok* field and compute for each hop field the packet-specific HV. To check the validity of the HV, 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 hv_A from σ_A and the *tok* included in the packet.

Two mechanisms in EPIC limit the effects of brute-force attacks on the validator: First, the HV is bound to a packet's origin. Second, a replay-suppression system at each border router prevents multiple packets with the same packet origin from being forwarded. Consequently, a successful brute-force attack on the HV can result in at most a single packet being forwarded along an unauthorized path.

Only by brute-forcing the underlying static hop authenticator σ could the attacker dynamically create valid but unauthorized HVs for arbitrary *tok* fields and thus send an unlimited number of packets. However, σ is a long authenticator. Hence, the success probability of such a brute-force attack is negligible.

Formalization. We formalize EPIC similarly to SCION-11. However, we additionally account for the attacker's ability to brute-force individual validator fields by introducing a strong attacker model, which gives the attacker access to an oracle. We discuss this extension of our framework and its use in the EPIC formalization in §8.4.

We use another extension (introduced below) to define the type of *tok* as a natural number. This public value represents the packet origin. The function *auth-restrict* always returns true. 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 $\langle\rangle$.

$$extract(\langle S_A, \mathsf{MAC}_{\sigma_A}(tok) \rangle) = extract'(\sigma_A)$$
$$extract'(\mathsf{MAC}_{K_A}(hi_A)) = hi_A, \quad extract'(\mathsf{MAC}_{K_A}(\langle hi_A, \mathsf{H}(\sigma_B) \rangle)) = hi_A \# extract'(\sigma_B).$$

According to our definition of the intruder knowledge given in Equation (15), the attacker knows the HV 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 the 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 similar to that of the SCION-11 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 valid HV of a hop field, then that hop field is contained in an authorized path.

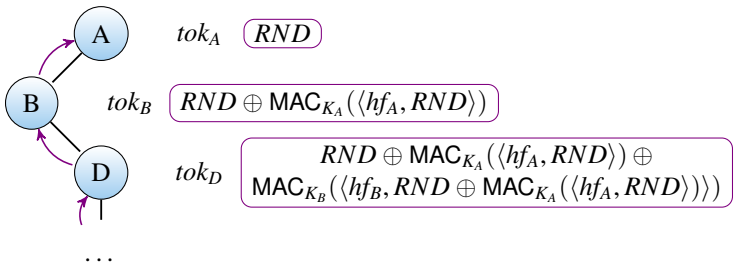


FIGURE 6: Values that the mutable *tok* field has in SCION-22 as a packet traverses the network from D to A. The *tok* field is updated by the receiving router according to Equation (18). The *tok* displayed next to each AS i is used to compute the HV of AS i according to Equation (19).

Variants. We verify EPIC level 1 and level 2 in the strong attacker model (see §8.4).

7.3. SCION-22

The SCION-22 protocol substantially revises the SCION packet header, and in particular changes how the forwarding path is used to compute the HV. The HV 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 packet forwarding and combines the HVs of all subsequent hops with exclusive-or (XOR), thereby including the upstream path similar to the MAC chaining of SCION-11 and EPIC.

7.3.1. Protocol description

The control plane creates forwarding paths such that for each hop on the path, the *tok* value embedded in its HV is the XOR of the HVs of all previous hops in the beaconing direction and of a random initialization value *RND*. Paths are reversed on the data plane, where the HVs of all *following* hops *in the 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 HVs that were added during beaconing until finally, only *RND* remains in *tok*.

We will use the topology and *tok* fields given in Figure 6 as a running example.

Motivation. In the original SCION-11 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 HV change. 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 start 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 SCION-22 protocol, where only the location of the current hop field changes, 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 SCION-22 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 details. 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 *hf*:

$$\mathit{upd\text{-}tok}(u, \mathit{hf}) = u \oplus \mathrm{HV}(\mathit{hf}), \tag{18}$$

where $\oplus$ is exclusive-or. The receiving router first updates the *tok* field and then proceeds normally by checking the interface and the HV. 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 HV. The HV is computed as:

$$\mathit{hv}_A = \mathsf{MAC}_{K_A}(\langle \mathit{hi}_A, \mathit{tok} \rangle)\,. \tag{19}$$

Example. Assume that, given the topology in Figure 6, 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 hv_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 by XORing the current tok_D field with the (unvalidated) hv_B embedded in the packet, yielding tok_B. Only after updating the *tok* field can the router check the validity of the hv_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.

7.3.2. Formalization

Required extensions. In order to formalize this instance, we needed to extend the framework presented in the previous section in three ways. First, we add an abstract XOR operator, second, we parametrize the type of *tok*, and third, we allow *tok* fields to be updated en-route.

Since our focus here is on the SCION-22 protocol, we only briefly introduce these extensions, and present further details in Sections 8.1–8.3.

- We extend IsaNet with a model of XOR by adding a constructor $\oplus$ for finite sets of terms (of type $\mathcal{P}_{\mathit{fin}}(\mathbb{T})$) to the datatype $\mathbb{T}$, where $\oplus H$ represents the XORing of all elements of H. We overload the operator $\oplus$ and define a binary version $\oplus : \mathcal{P}_{\mathit{fin}}(\mathbb{T}) \times \mathcal{P}_{\mathit{fin}}(\mathbb{T}) \to \mathcal{P}_{\mathit{fin}}(\mathbb{T})$, for combining two finite sets of terms with XOR, as the symmetric difference of these sets and the identity element $\{\}$ as the empty set. Our XOR model 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 $\mathit{upd\text{-}tok} : \mathit{'TOK} \times \mathrm{HF} \to \mathit{'TOK}$ as a new parameter to the concrete model. This requires adding two simple conditions. The 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 (18) is straightforward. We instantiate the type of *'TOK* with $\mathcal{P}_{\mathit{fin}}(\mathbb{T})$, i.e., finite sets of terms, since *tok* accumulates terms using XOR. We instantiate ψ with the check of Equation (19). 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, this *tok* field is initialized in 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 where we assume that *tok* fields are initialized with the identity element $\{\}$. We define *auth-restrict* as follows:

$$\begin{aligned} \mathit{auth\text{-}restrict}(\mathit{hfs} \cdot \langle \mathit{hf}_Z \rangle, u) &= \psi(\mathit{hf}_Z, \bot, \{\}) \\ \mathit{auth\text{-}restrict}(\langle\rangle, u) &= (u = \{\}) \end{aligned}$$

We require that the last hop field on the path is valid for the *tok* field with the identity element. This implies for paths *hfs* that are valid, i.e., where $\Psi(\mathit{hfs}, u) = \mathit{hfs}$ holds, that u does not contain any terms besides the HVs 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 SCION-22 is more difficult to define than in the other instances. It requires the use of the definite description operator THE, which returns a unique value satisfying a given condition. For instance, THE $x.\ x = 3 + 5$ is 8. We show the existence and uniqueness of the value separately. Using the THE operator and the *is-next* predicate defined below, we define *extract* as follows:

$$\begin{aligned} \mathit{extract}(\mathsf{MAC}_{K_A}(\langle \mathit{hi}_A, u \rangle)) = \ \mathit{hi}_A \ \# \ (\text{if } \exists \mathit{hf}_B.\ \mathit{is\text{-}next}(\mathit{hf}_B, u) \\ \text{then } \mathit{extract}(\mathrm{HV}(\text{THE } \mathit{hf}_B.\ \mathit{is\text{-}next}(\mathit{hf}_B, u))) \text{ else } \langle\rangle). \end{aligned}$$

As in the other models, *extract* first takes the current HV, and if it is of the form defined in Equation (19), the HI embedded in the HV's MAC is extracted. If there is a next hop field hf_B, then *extract* recursively calls itself on that next hop's HV embedded in the MAC. If there is no such element, the *extract* function does not recurse. In contrast to SCION-11, the next hop's HV is not *directly* embedded in the MAC of the current HV. Instead, the *tok* field is embedded, which accumulates via XOR all HVs remaining on the path. The extract function needs to identify and select the *next* hop's HV among this set. The difficulty in formally defining *extract* is determining the next hop field. Below, we define the predicate $\mathit{is\text{-}next} : \mathrm{HF} \times \mathit{'TOK} \to \mathbb{B}$, 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 hv_B that is a MAC created using *all other elements* of the *tok* set, i.e., $\mathit{hv}_B = \mathsf{MAC}_{K_B}(\mathit{hi}_B, \mathit{tok} \setminus \{\mathit{hv}_B\}))$ (where $\setminus$ is set difference). Let hf_B be the hop field such that $\mathrm{HV}(\mathit{hf}_B) = \mathit{hv}_B$ and $\overline{\mathit{hf}_B} = \mathit{hi}_B$. This hop field satisfies the predicate $\mathit{is\text{-}next}(\mathit{hf}_B, \mathit{tok})$, defined as

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

The predicate can only hold if $\mathrm{HV}(X) \in u$, in which case $\mathit{upd\text{-}tok}(u, X)$ is equivalent to $u \setminus \{\mathrm{HV}(X)\}$. To see this, assume, $\mathrm{HV}(X) \notin u$. Then $\mathrm{HV}(X)$ contains itself under a constructor, $\mathrm{HV}(X) = \mathsf{MAC}_{K_{id(X)}}(\overline{X}, u \cup \{\mathrm{HV}(X)\})$, leading to a contradiction with $\mathit{is\text{-}next}(X, u)$ if we consider the size of the term.

Example. Consider in Figure 6 that we extract the path from $hv_D = \mathsf{MAC}_{K_D}(\langle hi_D, tok_D \rangle)$. Given tok_D, a value X such that *is-next*(X, tok_D) holds is the hop field hf_B, where $\mathrm{HV}(hf_B) = hv_B$, $\overline{hf_B} = hi_B$ and $hv_B = \mathsf{MAC}_{K_B}(\langle hi_B, tok_B \rangle)$, since *upd-tok*$(tok_D, hv_B) = tok_B$. This is not just some value that satisfies *is-next* given tok_D, but it is the *only* value to do so. Hence, *extract*$(hv_D) = hi_D \,\#\, extract(hv_B)$. If we repeat these steps, we obtain *extract*$(hv_D) = \langle hi_D, hi_B, hi_A \rangle$, as expected.

Proofs. We first show that if a term t is contained in a *tok* field u, and $auth_{\mathrm{c}}(u)$ is not empty (i.e., there exists a path with u as the *tok* field), then t is a HV 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. After this simplification is proven, the use of XOR does not complicate the proofs in the instance model. Yet, proving the conditions is more difficult in SCION-22 than in other protocol instances, in particular COND 3 and COND 4, due to the complex definition of *extract*.

7.4. ICING

Among the protocols we study, ICING [22] provides the strongest security properties, albeit at the cost of the highest overhead [21]. 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 §8.5.

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 info field hi_A.

Formalization. In our symbolic model, PRFs and MACs are modeled identically and we thus formalize PoCs as validators using MACs. We define $ik_0^+ = \emptyset$. The function *extract* requires extracting the entire path (past and future) in the undirected setting. Formally,

$$hv_A = \mathsf{MAC}_{\langle K_A,\, hi_A \rangle}(rev(past(m)) \cdot fut(m))$$
$$extract(\mathsf{MAC}_{\langle K_A,\, hi_A \rangle}(hfs)) = hfs.$$

Variants. We verified three versions of the protocol. The first is presented above. In the second version, the validator 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.

The third version is a further simplified variant of ICING compared to the one presented first, which omits the info 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.

8. Extensions

We now describe a number of features of our formalization that we previously elided to simplify the presentation. Figure 8 in Appendix A lists which extensions are used in the individual protocol models.

8.1. 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 when modeling 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 that captures what an attacker can learn from analyzing a *tok* field:

$$\textit{analz-tok} : \textit{'TOK} \to \mathcal{P}(\mathbb{T}). \tag{20}$$

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

Instances. In SCION-22, the *tok* field contains the XOR of different terms. For reasons that will become clear below, we instantiate the type of *tok* to finite sets of $\mathbb{T}$ and define *analz-tok* as the identity function. In SCION-11, *'TOK* is $\mathbb{T}$ and $\textit{analz-tok}(t) = \{t\}$. We note that for all valid hop fields in EPIC (and in SCION-11), *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 $\textit{analz-tok}(t) = \{\}$.

8.2. Exclusive-or abstraction

We extend IsaNet with an abstraction of exclusive-or (XOR), which is used in the SCION-22 instance. The XOR operator $\oplus$ can be characterized by the following equations for associativity (A), commutativity (C), the identity element 0 (I) and self-inverse (S).

$$(x \oplus y) \oplus z = x \oplus (y \oplus z) \quad (A) \qquad x \oplus y = y \oplus x \quad (C)$$
$$x \oplus 0 = x \quad (I) \qquad x \oplus x = 0. \quad (S)$$

Existing modeling approaches. The XOR operator is difficult to support in symbolic protocol analysis. Most automated protocol verifiers that support equational theories cannot straightforwardly implement XOR using the above equations, as these equations do not form a subterm-convergent equational theory [42]. 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 $\textit{dec}(k, \textit{enc}(k, m)) = m$. While there are 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 *folding variant narrowing*. Automated protocol verifiers such as Maude-NPA [44] and Tamarin [45] follow this approach. 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, involves considerable effort in protocol verification using 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 identical 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) $\oplus\langle z, x, y, z\rangle$, (ii) $\oplus\langle x, y, z, z\rangle$, (iii) $\oplus\langle x, y, 0\rangle$ and finally $\oplus\langle x, y\rangle$. Since terms are enumerable, one can define a linear order on them. This approach of modeling XOR as a binary operator and defining normalization is taken by Schaller et al. [46] in their Isabelle/HOL formalization.

Our XOR model. In our framework, we follow a different approach. We introduce a term representation and an attacker model that simplify our reasoning about protocols using XOR. Instead of using term normalization, we directly model the canonical representations of XOR terms based on a new term constructor

$$\oplus : \mathcal{P}_{fin}(\mathbb{T}) \to \mathbb{T},$$

which we add to the definition of $\mathbb{T}$. Given a finite set X of terms from $\mathbb{T}$, the term $\oplus X$ represents the XORing all elements of X. In particular, the term $\oplus\{\}$ represents the identity 0. Provided that the terms in X are not themselves of the form $\oplus Y$ for some finite set Y, the properties of finite sets make $\oplus X$ a canonical normal form representation with respect to the four equations (A), (C), (I), and (S) of the theory of XOR. For example, we represent the term $t = (x \oplus (0 \oplus z)) \oplus (y \oplus x)$ in an algebra with binary XOR as $\oplus\{y, z\}$ in $\mathbb{T}$. The latter is a canonical representation of t's equivalence class, namely, it is unique with respect to (A) and (C) and reduced with respect to (I) and (S). To rule out nested XOR terms like $\oplus\{x, \oplus\{x, y\}\}$, we define a predicate *normal*, which holds if no directly nested $\oplus$ constructors occur.

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

Note that, for instance, $\oplus\{x, \mathsf{H}(\oplus\{y, z\})\}$ is normal, provided x, y and z are normal, non-$\oplus$ terms, since no *direct* nesting occurs.

In order to combine two given terms $\oplus X$ and $\oplus Y$ using XOR, we overload $\oplus$ and define a binary XOR function $\oplus : \mathcal{P}_{fin}(\mathbb{T}) \times \mathcal{P}_{fin}(\mathbb{T}) \to \mathcal{P}_{fin}(\mathbb{T})$ as the symmetric set difference:

$$X \oplus Y = (X \cup Y) \setminus (X \cap Y). \tag{21}$$

The XOR of $\oplus X$ and $\oplus Y$ is thus $\oplus(X \oplus Y)$, which satisfies the four properties (A), (C), (I), and (S). Note also that $\oplus(X \oplus Y)$ is normal whenever $\oplus X$ and $\oplus Y$ are.

Rather than defining a normalization function that turns an arbitrary term into a normal term, we define our event systems such that all terms in reachable states are already normal, i.e., they do not contain directly nested $\oplus$ constructors. We show this below for the SCION-22 instance.

$$\frac{X \subseteq_{fin} DY^{\uparrow}(H)}{\oplus X \in DY^{\uparrow}(H)} \; \forall Y.\ \oplus Y \notin X \qquad\qquad \frac{\oplus X \in DY^{\downarrow}(H)}{t \in DY^{\downarrow}(H)} \; t \in X$$

FIGURE 7: Rules added to the Dolev–Yao message decomposition ($DY^{\downarrow}$) and composition ($DY^{\uparrow}$) presented in Figure 5. The composition rule excludes directly nested $\oplus$ constructors.

Attacker models for XOR. We propose a novel overapproximation of the attacker capabilities that greatly simplifies reasoning about the composition and decomposition of XOR terms. This overapproximation is based on the observation that, 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 of this is the one-time pad encryption scheme, where a plaintext message is XORed with a fresh random 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. However, many authentication and integrity-protecting protocols either do not need one-wayness or even require an invertible function. XOR is also popular in protocols, as it is simple. Moreover, implementations in hardware are highly efficient. We now discuss attacker models for each of these two use cases.

In both cases, the attacker can compose messages using the rule on the left-hand side of Figure 7. This rule's side condition ensures that it preserves term normality (as do the existing rules from Figure 5). The difference is with the decomposition rules. The following rule reflects an attacker's standard capabilities to decompose XOR terms in our setting (cf. [45]):

$$\frac{\oplus X \in DY^{\downarrow}(H) \qquad \oplus Y \in DY(H)}{\oplus (X \oplus Y) \in DY^{\downarrow}(H)}. \tag{22}$$

Note that, for completeness, $\oplus Y$ can be derived using both decomposition and composition rules. This rule is suitable for the secrecy use case. However, the decomposition of a term $\oplus X$ using this rule introduces the difficulty that what we can derive from $\oplus X$ depends on other terms, namely those in Y. For instance, for $X = \{x, y\}$ and $Y = \{y\}$, we can derive $\oplus \{x\}$, while with $Y = \{y, z\}$ we get $\oplus \{x, z\}$.

In the case, where XOR is used as an invertible compression function only, we propose the simpler decomposition rule on the right-hand side of Figure 7. This rule overapproximates a realistic attacker's behavior by allowing them to learn any term $t \in X$ from a given term $\oplus X$. This substantially simplifies reasoning since the dependence on the derivability of a second XOR term is removed. This overapproximation is sound, as the derivation rule in Equation (22) is admissible[1] in the system with the rules in Figures 5 and 7. We can also show that the side condition of the composition rule is without loss of generality. Since our data plane protocols only use XOR as an invertible compression function, we only implemented the attacker model for this setting.

Protocol instances. We use our XOR formalization based on finite sets and our attacker overapproximation in the SCION-22 instance. We show that all terms occurring during the execution of the protocol

[1] A rule R is admissible in a derivation system if it does not add any deductive power, i. e., any proof using R can be converted into one not using R.

are normal. To see this, we observe that all messages in a state are contained in the intruder knowledge $ik(s)$. By Lemma 1, we can reduce the normality of terms in $ik(s)$ to the normality of terms in $DY(ik_0 \cup ik_0^+)$, which we prove in the instance model.

Discussion. Our finite set representation of XOR makes our formalization substantially more manageable, as explicit reasoning modulo equational theories is not required. Rather than a normalization function that re-orders terms using associativity and commutativity and applies identity and self-inverse simplification rules, we only require preventing directly nested $\oplus$ operators, which we do via the *normal* predicate. Note that our representation distinguishes the terms $\oplus\{t\}$ and t, which one would expect to map to the same bitstring in the implementation. However, collisions in general seem unavoidable for bitstring representations of XOR terms, e.g., the terms $1 \oplus 3$, $5 \oplus 7$, and 2 would commonly map to the same bitstring. Still, we must be careful to avoid protocol models that exhibit different observable behaviors based on a distinction between such terms, e. g., by sending two different messages, since such models would be unimplementable. Our model of SCION-22 does not have this problem. We will further discuss the general problem of message representation and implementation in the context of symbolic models in §9.3.

Our overapproximation of the attacker derivation capabilities also greatly simplifies reasoning about the intruder knowledge and the derivation of terms. Consequently, the use of XOR only adds a moderate amount of complexity in the SCION-22 instance. This is surprising since security protocol verification typically becomes much harder when XOR is used. While we have only used XOR in one instance, we believe that our XOR theory is applicable to a large class of security protocols, including several protocols studied in [47, 48].

Lastly, our XOR theory itself is very concise: all definitions and lemmas, including those related to the *normal* predicate, add less than 300 lines of code to the term algebra theory. In contrast, the XOR theory proposed by Schaller et al. [46] adds thousands of LoC.

8.3. 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 IsaNet by an update parameter

$$\textit{upd-tok} : \textit{'TOK} \times \text{HF} \to \textit{'TOK}. \tag{23}$$

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

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

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

The $\textbf{recv}_\text{c}$ event's guard is changed as follows: instead of requiring that ψ holds on (fields of) m, we require the check to hold on $\textit{upd-pkt}(m)$. Furthermore, the update of the event is changed and instead of m, $\textit{upd-pkt}(m)$ is added to $\textit{int}(A)$. 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 our framework, such as modifying the definition of Ψ to account for the *tok* update. Additional conditions are needed, in particular: (i) The update function does not reveal anything that the attacker could not already derive, and (ii) *auth-restrict* is closed under updates. The refinement proof presented in §6 requires 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 $hf \# hfs_{post}$ of an authorized path $hfs_{full} = hfs_{pre} \cdot hf \# hfs_{post} \in auth_c(u)$ is valid, i.e., $\Psi(hf \# hfs_{post}, u') = hf \# hfs_{post}$ for some u'. Since the packet's *tok* field changes as the packet is forwarded, $hf \# hfs_{post}$ and hfs_{full} are potentially valid for different values $u' \neq u$. In order to obtain the correct *tok* value u' for $hf \# hfs_{post}$ given the *tok* value u for the full path hfs_{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 hfs_{pre}. Unfortunately, obtaining u' given the original u valid for hfs_{full} does not simply involve applying *upd-tok* iteratively for each hop field on hfs_{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 $hfs_{pre} = hf' \# hfs'$, we need to update *tok* for each hop field on $hfs' \cdot \langle hf \rangle$, i.e., we drop the first hop field of hfs_{pre} and add the next hop field after hfs_{pre} in the sequence of hop fields for which an update must be performed. This "shifted" reasoning requires additional case distinctions and adds some complexity to our definitions and proofs.

Instances. SCION-22 updates the *tok* field using XOR, as defined in Equation (18). The other protocol instances do not update the *tok* field and hence define *upd-tok* to return a given *tok* field unmodified. Discharging the three additional conditions is straightforward for all instances.

8.4. Strong attacker model

Legner et al. [21] propose a strong attacker model for EPIC, which reflects that an adversary can, with some effort, brute-force correct *hv* fields for individual *tok* values. We model this attacker capability with an oracle. We add a predicate

$$\mathcal{O} : \text{'TOK} \to \mathbb{B} \tag{24}$$

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 HV fields of (possibly unauthorized) paths that are created over *tok* values such that $\mathcal{O}(tok)$ holds.

While this addition strictly strengthens the attacker, her events must be restricted to rule out trivial attacks where the attacker sends a packet with a *tok* value for which she queried the oracle. We add the guard $\neg\mathcal{O}(tok(m))$ to the **dispatch-int**$_c$ and **dispatch-ext**$_c$ events to prevent the attacker from sending packets whose HV 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.

8.5. 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 the 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: $his \in auth_{\text{a}}$ 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: $\psi : \text{HF} \times \text{HF}^* \times \mathbb{T} \to \mathbb{B}$. The parameter *extract* retains its type, but returns the entire path (including past path) instead of just the future path. Since each validator 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 HV 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 that 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 3. The proof of this theorem is done separately, since it differs between the directed and undirected setting.

8.6. Additional authenticated fields

To allow for more accurate modeling of protocols and to future-proof our framework, our formalization includes additional per-hop and per-packet fields, which are included in $auth_{\text{a}}$ and must thus be included in the authentication mechanisms defined by instances. We use the authenticated per-packet field to model SCION's path expiration time that is fixed in the control plane and is included in the MAC computation of the validator. We use the per-hop authenticated fields to model ICING's *tag*.

This extension requires changes to the definitions, parameters, assumptions, conditions and lemmas of both the models presented in the previous sections and in the above extensions. Nevertheless, the use of these additional fields does not add significant complexity and is not essential to the insights provided by our models and proofs. Hence, we have elided their presentation above.

9. Discussion

In this section, we discuss some additional aspects of our formalization (§9.1), we point out differences between our models and the actual protocol specifications or implementations (§9.2), and we discuss the representation of messages in our models and in the implementations (§9.3).

9.1. Formalization details

Presentation 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 parametrized models, not the instance models. In particular, this is true of the crux of the proof, Theorem 3. A substantial portion of the instance models is boilerplate definitions and proofs that only vary slightly between the instances. Table 3 in Appendix D gives an overview of the Isabelle/HOL code of IsaNet.

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 tg, the set of authorized paths $auth_{\text{a}}$, 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, $\mathcal{N}_{attr} = \{F\}$, and an $\mathcal{O}$ function that always produces false. We discharge the assumptions ASM 1–ASM 6 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.

9.2. Differences between models and real protocols

We discuss the abstractions that we have made in our protocol models, distinguishing generic abstractions used in the parametric model and instance-specific abstractions.

Generic abstractions. Our models abstract from real protocols in several ways, many of which are standard in protocol verification. First, we simplify the structure of messages. We omit additional message fields and checks that are either irrelevant to path authorization or unnecessary due to our more abstract message representation (e. g., the protocol version, an explicit path length, and the current hop field pointer). Moreover, unlike our models, the actual protocols do not include the AS identifier (*id*) in the hop fields. Its addition simplifies our proofs. It would however be possible to remove the identifier in a refinement step since the MAC key used in the HV uniquely identifies the AS for which the hop field is valid. Second, we abstract the actual protocol behavior. Our models receive, process, and forward packets in a single atomic step, whereas in implementations these will each correspond to individual steps. We are confident that an additional refinement step could relax the atomicity of these steps by introducing ingress and egress buffers along with a finer-grained step interleaving of their constituent operations. Moreover, in our models, routers can receive and send the same message not just once, but multiple times. This is also unproblematic as we do not consider liveness properties. Third, we abstract the environments in which the protocols run. We abstract from the intra-AS network topologies and the protocols they run, and we do not explicitly model end hosts. We also ignore the fact that real routers will concurrently execute multiple protocols, for example, SCION-22 alongside different levels of EPIC, as well as control message protocols. This may give rise to multi-protocol attacks, where different protocols interfere with each other [49]. We leave the proof of resistance against such attacks for future work.

Instance-specific abstractions. Our instance models additionally differ from the actual protocols in the following ways. In SCION, forwarding paths can be created by connecting multiple partial paths, called *segments*, according to certain rules. In our current formalization, we only consider single up-segment paths. In EPIC, hop authenticators σ are shortened to reduce space overhead. We model the σ's shortening using a hash function. Similar to hashing, shortening makes it difficult to recover the original value. It also enables brute-force attacks, which we model by the oracle discussed in §8.4. In SCION-22, our formalization does not include the initialization value *RND* for the *tok* field and ignores the fact that MACs are truncated from six to two bytes before XOR-ing. In ICING we leave out the *proofs of provenance* that are combined with PoCs using XOR. These are cryptographic authenticators used for path validation and are unrelated to path authorization.

9.3. Message representation and implementation

Our verification results hold in a symbolic (Dolev-Yao) attacker model where messages are represented as terms and cryptography is assumed to be perfect. Note that to simplify verification we use terms in combination with other Isabelle/HOL datatypes such as records and lists to model parts of network packets (see Equations (8) and (9)). We consider this as unproblematic, since these structures could be encoded as message terms in a further refinement, where pattern matching in the receive event would enforce the correct packet structure. Unlike our models, protocol implementations manipulate bitstring messages and use cryptographic libraries with non-perfect (computational) security guarantees. Hence, the symbolic model focuses on the security of the main protocol logic, but may miss attacks exploiting cryptographic weaknesses or problems related to the parsing of bitstring messages. The fundamental gap between symbolic models and implementations is a general problem of the symbolic approach. Below we discuss two possible approaches to address it.

Computational soundness. The strongest way to bridge this gap are computational soundness results (see [50] for an overview). Such results would allow us to deduce cryptographic security guarantees from our symbolic verification results. However, these results often require rather strong assumptions, which may be hard to fulfill in practice. For example, in [51, 52], the authors assume an injective mapping from terms to bitstrings. There are several reasons why the existence of such a mapping may be unrealistic. First, distinguishing different types of messages (e. g., SHA-256 hashes from AES-256 ciphertext) may require systematic tagging, which is often prohibited by implementation constraints. Second, unlike in the symbolic model, cryptographic hash functions are non-injective by definition. Moreover, for some primitives and computational models, computational soundness is unachievable. For example, Unruh [53] presents a general impossibility result for XOR.

Perfect cryptography view on bitstrings. A simpler solution to this problem, which essentially keeps up the perfect cryptography illusion on the bitstring level, is to define the cryptographic library over an abstract data type of messages and consider two different implementations where messages respectively are instantiated with terms (which is useful for testing) and bitstrings (for the actual implementation) [54, 55]. Apart from the cryptographic operations, the parsing of (non-cryptographic) message formats is a common source of implementation errors. Actual protocol message formats are often based on a combination of fixed-length fields, variable-length fields, and tags (see, e. g., the SCION-22 header format [20]). Whether provably sound and secure parsing is achievable for the protocols studied here deserves further investigation, possibly along the lines of [56, 57].

10. Related work

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

Data plane protocols for path-aware architectures. Over the past two decades, several other path-aware architectures have been developed [7–10]. Several of these use forwarding tables or other kinds of state on the routers (instead of cryptographic authenticators) to achieve path authorization [9, 10], which does not fit into our framework. Others are not specified in sufficient detail to allow for formal verification [8] or only achieve local properties without considering full path authorization over multiple hops [7]. Finally, some data plane protocols [58], including OPT [59], focus only on source authentication and path validation, neither of which we verify.

Verification of secure data plane protocols. Chen et al. [24] 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 the 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. It only guarantees that each traversed hop appears on some authorized path, but does not relate successively traversed hops.

Zhang et al. [25] prove source authentication and path validation properties of the OPT forwarding protocols [59]. These properties differ from those that we formally prove. They use LS^2, a logic for reasoning about secure systems, in combination with axioms from Protocol Composition Logic (PCL) [60]. They directly embed their logic's axioms and prove the protocols' properties in Coq. As PCL does not have a formal semantics (cf. [61]), 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.

Verification of secure routing protocols. Cortier et al. [62] 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 [63].

Cortier et al. [64] 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 1 but omitting interfaces. They then analyze two ad-hoc routing protocols using ProVerif.

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. [65] use parametrization for an abstract realization of channels with different security properties, also in the context of refinement. Schaller et al. [46] employ parametrization to verify physical properties of a number of different wireless protocols.

Exclusive-or in security protocol verification. Automated security protocol verifiers, such as Maude-NPA [44], AKISS [66], and Tamarin [45] have incorporated support for exclusive-or (XOR). Yet it is widely recognized that XOR is challenging to support in security protocol verification. Some verification works abstract from XOR, when the protocol's security properties do not depend on it [67]. This is the approach that we follow for the ICING protocol instance, which uses additional authenticators that achieve properties that are unrelated to path authorization. However, it cannot be applied to SCION-22, since SCION-22's use of XOR is central to achieving its security properties.

Schaller et al. [46, 68] 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 using 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 fulfill the finite variant property.

11. Conclusion

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 IsaNet 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 just define the parameters and prove the static conditions to establish path authorization and detectability. Our abstractions are general enough to cover a large class of protocols proposed in the literature.

The present work constitutes an important step towards our ultimate goal of verifying a high-performance implementation of the SCION-22 router written in Go. We are pursuing this goal using the Igloo methodology [69] to soundly link protocol and code verification. This methodology guarantees that the implementation will inherit the properties we have proven for our model. To achieve this goal, we are further refining our SCION-22 model, adding more detail such as packet buffering and segment switching, and we are extracting a program specification of the router's behavior from the refined model. This specification will be used for the verification of the router code.

In future work, 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. Furthermore, 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 [64], which would enable the fully automated verification of secure dataplane protocols.

Acknowledgments

We thank Sofia Giampietro and the anonymous reviewers for their insights, careful reading of the manuscript and helpful suggestions.

References

[1] S. Kent, C. Lynn and K. Seo, Secure Border Gateway Protocol (S-BGP), *IEEE Journal on Selected Areas in Communications* **18**(4) (2000).
[2] R. Bush, Origin Validation Operation Based on the Resource Public Key Infrastructure (RPKI), RFC, 7115, 2014. ISSN 2070-1721.
[3] M. Lepinski and K. Sriram, BGPsec Protocol Specification, RFC Editor, 2017. doi: 10.17487/RFC8205. https://rfc-editor.org/rfc/rfc8205.txt.
[4] D. Cooper, E. Heilman, K. Brogle, L. Reyzin and S. Goldberg, On the risk of misbehaving RPKI authorities, in: *Proceedings of the ACM Workshop on Hot Topics in Networks (HotNets)*, 2013, pp. 1–7. doi: 10.1145/2535771.2535787.
[5] Q. Li, Y.-C. Hu and X. 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. doi: 10.3929/ethz-a-010189168.
[6] B. Rothenberger, D.E. Asoni, D. Barrera and A. Perrig, Internet Kill Switches Demystified, in: *Proceedings of the European Workshop on Systems Security (EuroSec)*, 2017.
[7] B. Raghavan and A.C. Snoeren, A system for authenticated policy-compliant routing, *ACM SIGCOMM Computer Communication Review* **34**(4) (2004). doi: 10.1145/1030194.1015487.
[8] B. Bhattacharjee, K. Calvert, J. Griffioen, N. Spring and J.P.G. Sterbenz, Postmodern internetwork architecture, *NSF Nets FIND Initiative* (2006).
[9] X. Yang, D. Clark and A.W. Berger, NIRA: A New Inter-Domain Routing Architecture, *IEEE/ACM Transactions on Networking* (2007).
[10] P.B. Godfrey, I. Ganichev, S. Shenker and I. Stoica, Pathlet Routing, in: *Proceedings of ACM SIGCOMM*, 2009.
[11] X. Zhang, H.-C. Hsiao, G. Hasker, H. Chan, A. Perrig and D. Andersen, SCION: Scalability, Control, and Isolation On Next-Generation Networks, in: *Proceedings of the IEEE Symposium on Security and Privacy*, 2011.
[12] T. Anderson, K. Birman, R. Broberg, M. Caesar, D. Comer, C. Cotton, M.J. Freedman, A. Haeberlen, Z.G. Ives, A. Krishnamurthy, W. Lehr, B.T. Loo, D. Mazières, A. Nicolosi, J.M. Smith, I. Stoica, R. van Renesse, M. Walfish, H. Weatherspoon and C.S. Yoo, The NEBULA Future Internet Architecture, in: *The Future Internet*, Springer, 2013. doi: 10.1007/978-3-642-38082-2_2.
[13] A. Perrig, P. Szalachowski, R.M. Reischuk and L. Chuat, *SCION: A Secure Internet Architecture*, Springer, 2017. ISBN 978-3-319-67079-9. doi: 10.1007/978-3-319-67080-5.
[14] J. Naous, M. Walfish, A. Nicolosi, D. Mazieres, M. Miller and A. Seehra, Verifying and enforcing network paths with ICING, in: *Proceedings of the ACM International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, 2011.
[15] S. Meier, B. Schmidt, C. Cremers and D.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*, N. Sharygina and H. Veith, eds, Lecture Notes in Computer Science, Vol. 8044, Springer, 2013, pp. 696–701. doi: 10.1007/978-3-642-39799-8_48.
[16] B. 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, pp. 82–96. doi: 10.1109/CSFW.2001.930138.
[17] T. Nipkow, L.C. Paulson and M. Wenzel, *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, Vol. 2283, Springer, 2002. ISBN 3-540-43376-7. doi: 10.1007/3-540-45949-9.
[18] X. Zhang, H.-C. Hsiao, G. Hasker, H. Chan, A. Perrig and D.G. Andersen, SCION: Scalability, control, and isolation on next-generation networks, in: *2011 IEEE Symposium on Security and Privacy*, IEEE, 2011, pp. 212–227.

[19] L. Chuat, M. Legner, D. Basin, D. Hausheer, S. Hitz, P. Müller and A. Perrig, *The Complete Guide to SCION*, Springer, 2022. ISBN 978-3-031-05287-3. doi: 10.1007/978-3-031-05288-0.

[20] Anapaya Systems, SCION Header Specification, 2022, https://scion.docs.anapaya.net/en/latest/protocols/scion-header.html.

[21] M. Legner, T. Klenze, M. Wyss, C. Sprenger and A. 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, pp. 541–558. ISBN 978-1-939133-17-5. https://www.usenix.org/conference/usenixsecurity20/presentation/legner.

[22] J. Naous, M. Walfish, A. Nicolosi, D. Mazières, M. Miller and A. 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*, K. Cho and M. Crovella, eds, ACM, 2011, p. 30. ISBN 978-1-4503-1041-3. doi: 10.1145/2079296.2079326.

[23] T. Klenze, C. Sprenger and D. Basin, Formal Verification of Secure Forwarding Protocols, in: *2021 IEEE 34rd Computer Security Foundations Symposium (CSF)*, IEEE, 2021.

[24] C. Chen, L. Jia, H. Xu, C. Luo, W. Zhou and B.T. Loo, A Program Logic for Verifying Secure Routing Protocols, *Logical Methods in Computer Science* **Volume 11, Issue 4** (2015). doi: 10.2168/LMCS-11(4:19)2015. https://lmcs.episciences.org/1620.

[25] F. Zhang, L. Jia, C. Basescu, T.H. Kim, Y. Hu and A. Perrig, Mechanized Network Origin and Path Authenticity Proofs, in: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, G. Ahn, M. Yung and N. Li, eds, ACM, 2014, pp. 346–357. doi: 10.1145/2660267.2660349.

[26] T. Klenze and C. Sprenger, IsaNet: Formalization of a Verification Framework for Secure Data Plane Protocols, *Archive of Formal Proofs* (2022), https://isa-afp.org/entries/IsaNet.html, Formal proof development.

[27] E. Katz-Bassett, C. Scott, D.R. Choffnes, Í. Cunha, V. Valancius, N. Feamster, H.V. Madhyastha, T.E. Anderson and A. Krishnamurthy, LIFEGUARD: practical repair of persistent route failures, in: *SIGCOMM 2012*, L. Eggert, J. Ott, V.N. Padmanabhan and G. Varghese, eds, ACM, 2012, pp. 395–406. doi: 10.1145/2342356.2342435.

[28] T.G. Griffin and G. Wilfong, An Analysis of BGP Convergence Properties, in: *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '99, ACM, New York, NY, USA, 1999, pp. 277–288–. ISBN 1581131356. doi: 10.1145/316188.316231.

[29] H. Ballani, P. Francis and X. Zhang, A study of prefix hijacking and interception in the Internet, *ACM SIGCOMM Computer Communication Review* **37**(4) (2007). doi: 10.1145/1282427.1282411.

[30] R. White, Securing BGP through secure origin BGP (soBGP), *Business Communications Review* **33**(5) (2003), 47–47.

[31] T. Wan, E. Kranakis and P.C. van Oorschot, Pretty Secure BGP, psBGP, in: *NDSS*, 2005.

[32] J. Karlin, S. Forrest and J. Rexford, Pretty good BGP: Improving BGP by cautiously adopting routes, in: *Proceedings of the IEEE International Conference on Network Protocols*, IEEE, 2006.

[33] Y. Gilad, A. Cohen, A. Herzberg, M. Schapira and H. Shulman, Are We There Yet? On RPKI's Deployment and Security, in: *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*, The Internet Society, 2017. https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/are-we-there-yet-rpkis-deployment-and-security/.

[34] NIST, RPKI Monitor, 2020, https://rpki-monitor.antd.nist.gov.

[35] L. Gao, On inferring autonomous system relationships in the Internet, *IEEE/ACM Transactions on Networking* **9**(6) (2001), 733–745.

[36] L. Gao and J. Rexford, Stable Internet Routing without Global Coordination, *IEEE/ACM Trans. Netw.* **9**(6) (2001), 681—692. doi: 10.1109/90.974523.

[37] N.A. Lynch and F.W. Vaandrager, Forward and Backward Simulations: I. Untimed Systems, *Inf. Comput.* **121**(2) (1995). doi: 10.1006/inco.1995.1134.

[38] C. Ballarin, Locales: A Module System for Mathematical Theories, *J. Autom. Reason.* **52**(2) (2014), 123–153. doi: 10.1007/s10817-013-9284-7.

[39] Y.-C. Hu, A. Perrig and D.B. Johnson, Wormhole attacks in wireless networks, *IEEE journal on selected areas in communications* **24**(2) (2006), 370–380.

[40] M. Abadi and L. Lamport, The Existence of Refinement Mappings, *Theor. Comput. Sci.* **82**(2) (1991). doi: 10.1016/0304-3975(91)90224-P.

[41] L. Paulson, The inductive approach to verifying cryptographic protocols, *J. Computer Security* **6** (1998). http://www.cl.cam.ac.uk/users/lcp/papers/Auth/jcs.pdf.

[42] M. Abadi and V. Cortier, Deciding knowledge in security protocols under equational theories, *Theoretical Computer Science* **367**(1) (2006), 2–32, Automated Reasoning for Security Protocol Analysis. doi: https://doi.org/10.1016/j.tcs.2006.08.032. https://www.sciencedirect.com/science/article/pii/S030439750600572X.

[43] S. Escobar, R. Sasse and J. Meseguer, Folding variant narrowing and optimal variant termination, *The Journal of Logic and Algebraic Programming* **81**(7) (2012), 898–928, Rewriting Logic and its Applications. doi: https://doi.org/10.1016/j.jlap.2012.01.002. https://www.sciencedirect.com/science/article/pii/S1567832612000033.

[44] S. Escobar, C. Meadows and J. Meseguer, *Maude-NPA: Cryptographic Protocol Analysis Modulo Equational Properties*, in: *Foundations of Security Analysis and Design V: FOSAD 2007/2008/2009 Tutorial Lectures*, A. Aldini, G. Barthe and R. Gorrieri, eds, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 1–50. ISBN 978-3-642-03829-7. doi: 10.1007/978-3-642-03829-7_1.

[45] 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, pp. 359–373. ISSN 2374-8303. doi: 10.1109/CSF.2018.00033.

[46] 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, pp. 109–123. ISSN 2377-5459. doi: 10.1109/CSF.2009.6.

[47] T. van Deursen and S. Radomirovic, Attacks on RFID Protocols, *IACR Cryptology ePrint Archive* **2008** (2008), 310.

[48] J. Dreier, L. Hirschi, S. Radomirovic and R. Sasse, Verification of stateful cryptographic protocols with exclusive OR, *J. Comput. Secur.* **28**(1) (2020), 1–34. doi: 10.3233/JCS-191358.

[49] C. Cremers, Feasibility of Multi-Protocol Attacks, in: *Proceedings of the The First International Conference on Availability, Reliability and Security, ARES 2006, The International Dependability Conference - Bridging Theory and Practice, April 20-22 2006, Vienna University of Technology, Austria*, IEEE Computer Society, 2006, pp. 287–294. doi: 10.1109/ARES.2006.63.

[50] V. Cortier, S. Kremer and B. Warinschi, A Survey of Symbolic Methods in Computational Analysis of Cryptographic Systems, *J. Autom. Reasoning* **46**(3–4) (2011), 225–259. http://dx.doi.org/10.1007/s10817-010-9187-9.

[51] D. Micciancio and B. Warinschi, Soundness of Formal Encryption in the Presence of Active Adversaries, in: *Theory of Cryptography, First Theory of Cryptography Conference, TCC 2004, Cambridge, MA, USA, February 19-21, 2004, Proceedings*, M. Naor, ed., Lecture Notes in Computer Science, Vol. 2951, Springer, 2004, pp. 133–151. ISBN 3-540-21000-8. doi: 10.1007/978-3-540-24638-1_8.

[52] V. Cortier and B. Warinschi, Computationally Sound, Automated Proofs for Security Protocols, in: *Programming Languages and Systems, 14th European Symposium on Programming,ESOP 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, S. Sagiv, ed., Lecture Notes in Computer Science, Vol. 3444, Springer, 2005, pp. 157–171. ISBN 3-540-25435-8. doi: 10.1007/978-3-540-31987-0_12.

[53] D. Unruh, The impossibility of computationally sound XOR, *IACR Cryptol. ePrint Arch.* (2010), 389. http://eprint.iacr.org/2010/389.

[54] K. Bhargavan, C. Fournet, A.D. Gordon and S. Tse, Verified interoperable implementations of security protocols, *ACM Trans. Program. Lang. Syst.* **31**(1) (2008), 5:1–5:61. doi: 10.1145/1452044.1452049.

[55] K. Bhargavan, A. Bichhawat, Q.H. Do, P. Hosseyni, R. Küsters, G. Schmitz and T. Würtele, DY*: A Modular Symbolic Verification Framework for Executable Cryptographic Protocol Code, in: *IEEE European Symposium on Security and Privacy, EuroS&P 2021, Vienna, Austria, September 6-10, 2021*, IEEE, 2021, pp. 523–542. doi: 10.1109/EuroSP51992.2021.00042.

[56] S. Mödersheim and G. Katsoris, A Sound Abstraction of the Parsing Problem, in: *IEEE 27th Computer Security Foundations Symposium, CSF 2014, Vienna, Austria, 19-22 July, 2014*, IEEE Computer Society, 2014, pp. 259–273. doi: 10.1109/CSF.2014.26.

[57] T. Ramananandro, A. Delignat-Lavaud, C. Fournet, N. Swamy, T. Chajed, N. Kobeissi and J. Protzenko, EverParse: Verified Secure Zero-Copy Parsers for Authenticated Message Formats, in: *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*, N. Heninger and P. Traynor, eds, USENIX Association, 2019, pp. 1465–1482. https://www.usenix.org/conference/usenixsecurity19/presentation/delignat-lavaud.

[58] K. Bu, A. Laird, Y. Yang, L. Cheng, J. Luo, Y. Li and K. Ren, Unveiling the Mystery of Internet Packet Forwarding: A Survey of Network Path Validation, *ACM Comput. Surv.* **53**(5) (2020). doi: 10.1145/3409796.

[59] T.H.-J. Kim, C. Basescu, L. Jia, S.B. Lee, Y.-C. Hu and A. Perrig, Lightweight Source Authentication and Path Validation, in: *Proceedings of the 2014 ACM Conference on SIGCOMM*, Association for Computing Machinery, 2014, pp. 271–282. ISBN 9781450328364. doi: 10.1145/2619239.2626323.

[60] A. Datta, A. Derek, J.C. Mitchell and A. Roy, Protocol Composition Logic (PCL), *Electr. Notes Theor. Comput. Sci.* **172** (2007), 311–358.

[61] C. 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, pp. 66—76. ISBN 9781595939791. doi: 10.1145/1368310.1368324.

[62] M. Arnaud, V. Cortier and S. Delaune, Modeling and verifying ad hoc routing protocols, *Inf. Comput.* **238** (2014), 30–67. doi: 10.1016/j.ic.2014.07.004.

[63] M. Arnaud, V. Cortier and S. 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*, N. Bjørner and V. Sofronie-Stokkermans, eds, Lecture Notes in Computer Science, Vol. 6803, Springer, 2011, pp. 49–63. doi: 10.1007/978-3-642-22438-6_6.

[64] V. Cortier, J. Degrieck and S. 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*, P. Degano and J.D. Guttman, eds, Lecture Notes in Computer Science, Vol. 7215, Springer, 2012, pp. 30–50. ISBN 978-3-642-28640-7. doi: 10.1007/978-3-642-28641-4_3.

[65] J. Lallemand, D. Basin and C. Sprenger, Refining Authenticated Key Agreement with Strong Adversaries, in: *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2017, pp. 92–107. doi: 10.1109/EuroSP.2017.22.

[66] 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, pp. 234–248. ISSN 2374-8303. doi: 10.1109/CSF.2017.22.

[67] A. Debant and S. Delaune, Symbolic Verification of Distance Bounding Protocols, in: *Principles of Security and Trust*, F. Nielson and D. Sands, eds, Springer International Publishing, Cham, 2019, pp. 149–174. ISBN 978-3-030-17138-4.

[68] B. Schmidt, P. Schaller and D. Basin, Impossibility results for secret establishment, in: *2010 23rd IEEE Computer Security Foundations Symposium*, IEEE, 2010, pp. 261–273.

[69] C. Sprenger, T. Klenze, M. Eilers, F.A. Wolf, P. Müller, M. Clochard and D.A. Basin, Igloo: soundly linking compositional refinement and separation logic for distributed system verification, *Proc. ACM Program. Lang.* **4**(OOPSLA) (2020), 152:1–152:31. doi: 10.1145/3428220.

[70] B. Rothenberger, D. Roos, M. Legner and A. Perrig, PISKES: Pragmatic Internet-Scale Key-Establishment System, in: *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, ASIA CCS '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 73—86. ISBN 9781450367509. doi: 10.1145/3320269.3384743.

Extension	SCION-11	EPIC	SCION-22	ICING
Type parametrization	✓	✓	✓	✓
Exclusive-or	✗	✗	✓	✗
Mutable *tok* fields	✗	✗	✓	✗
Strong attacker model	✗	(✓)	✗	✗
Undirected setting	✗	✗	✗	✓
Additional auth. fields	✓	✓	✓	✓

FIGURE 8: Extensions used by protocol models. (✓) means that the extension is used by some (but not all) variants.

Appendix A. Extensions Table

Figure 8 gives an overview of the extensions used by protocol instances.

Appendix B. Undirected vs. directed protocols

While undirected protocols achieve path authorization under weaker assumptions (cf. §8.5), undirected protocol have several 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 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 of these 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 on the control plane and overall architecture.

Appendix C. Unverified data plane security properties

C.1. 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 shared between the source and the authenticating entity. 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 [59, 70]. 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.

C.2. Path validation

When path validation holds, then subsequent ASes and the destination have the guarantee that all previous hops on the path embedded in the packet were indeed traversed. While path validation is provided by some architectures [58], 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 it 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. 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.

Appendix D. Formalization Details and Statistics

Table 3 gives an overview of the different parts of our framework and the lines of Isabelle/HOL code associated with them.

Formalization of framework	**LoC**
Infrastructure (Dolev-Yao, Event System, etc.)	2709
Abstract Model & Network Model	694
Concrete Model (w/o Theorem 3)	855
Theorem 3 for directed setting	652
Theorem 3 for undirected setting	263
Total	**5173**
Formalization of instances	**LoC**
SCION-11	321
SCION-11 simplified	317
EPIC Level 1 Basic Attacker	395
EPIC Level 1 Strong Attacker	437
EPIC Level 2 Strong Attacker	464
SCION-22	567
ICING	330
ICING simplified	258
ICING further simplified	267
Executability proof for EPIC Level 1 SA	406
Total	**3762**

TABLE 3: Overview of Isabelle/HOL formalization.