Mechanized Network Origin and Path Authenticity Proofs

Fuyuan Zhang
CyLab, CMU
fuzh@andrew.cmu.edu

Limin Jia
ECE & INI, CMU
liminjia@cmu.edu

Cristina Basescu
ETH Zürich
cba@inf.ethz.ch

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

Yih-Chun Hu
UIUC
yihchun@uiuc.edu

Adrian Perrig
CyLab, CMU/ETH Zürich
aperrig@inf.ethz.ch

ABSTRACT

A secure routing infrastructure is vital for secure and reliable Internet services. Source authentication and path validation are two fundamental primitives for building a more secure and reliable Internet. Although several protocols have been proposed to implement these primitives, they have not been formally analyzed for their security guarantees. In this paper, we apply proof techniques for verifying cryptographic protocols (e.g., key exchange protocols) to analyzing network protocols. We encode LS^2, a program logic for reasoning about programs that execute in an adversarial environment, in Coq. We also encode protocol-specific data structures, predicates, and axioms. To analyze a source-routing protocol that uses chained MACs to provide origin and path validation, we construct Coq proofs to show that the protocol satisfies its desired properties. To the best of our knowledge, we are the first to formalize origin and path authenticity properties, and mechanize proofs that chained MACs can provide the desired authenticity properties.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: Security and protection; C.2.2 [**Network Protocols**]: Protocol verification

Keywords

Origin authenticity, path authenticity, secrecy, formal methods, mechanized proofs

1. INTRODUCTION

A secure routing infrastructure is central to providing secure and reliable Internet services. The current Internet routing infrastructure has many known security issues [38, 16]. Source authentication and path validation are two fundamental building blocks for security mechanisms that can mitigate some of these issues. Source authentication allows routers to validate whether a received packet indeed originated from the claimed source. Path validation allows routers to verify whether received packets indeed travelled on the source-selected path. Source authentication and path validation are key to implementing mechanisms such as path compliance, packet attribution, and protection against redirection of flows. For instance, path validation provides a way to enforce path compliance according to the policies of ISPs, enterprises, and datacenters. Without it, a malicious ISP can use an inferior path while claiming to its client that it forwarded the packet on the premium path, incurring greater charges.

Several protocols for source authentication and path validation have been proposed; however, their security guarantees have not been formally verified [31, 9, 39, 37]. So far, the state-of-the-art security analysis in this space is limited to enumerating attack scenarios and arguing how the proposed protocol can defend against those attacks. In this paper, we apply proof techniques for verifying cryptographic protocols (e.g., key exchange protocols) to the verification of source authentication and path validation protocols.

Analyzing network protocols is far more complex than analyzing cryptographic protocols, as the analysis needs to consider arbitrary network topologies. Furthermore, the protocol programs as well as the properties are recursive, often bound by the size of the network. If we were to apply model checking tools for analyzing cryptographic protocols naïvely, we would have to fix the size of the network topology and limit ourselves to verifying properties specific to a set of topologies. It is unclear how to use these model checking tools to prove a general property that holds for all topologies. In some special cases, small-world theorems have been attempted for simpler ad hoc wireless protocols [20], where only a finite number of topologies need to be checked to conclude that a property holds on all topologies. Unfortunately, to the best of our knowledge, general small-world theorems applicable to verifying path validation protocols do not exist. Therefore, model checking techniques cannot be immediately applied to proving properties of these protocols. As a first step towards understanding the formal aspects of these source authentication and path validation protocols, we manually construct proofs of general security properties that are independent of the network topology.

Several proof systems exist for reasoning about protocols [22, 23, 25, 40]. Paulson showed how to encode the network adversary model and the protocol, and prove properties of protocols in Isabelle/HOL [40]. PCL is a program logic that includes compositional reasoning principles for protocols [22]. Several protocols have been verified using PCL [41]. LS^2 generalizes PCL and provides a set of reasoning principles for deriving trace properties of programs that run concurrently with adversaries [23]. LS^2 can be used to reason about protocols, as well as general system designs.

Because of its generality, we use LS^2 as our reasoning logic. We encode LS^2 in Coq using shallow embedding, a technique frequently used in encoding one proof system in another (e.g., the encoding of Ynot [36]). LS^2-specific inference rules are isolated in a monad. Other generic inference rules such as first-order logic rules are not encoded; instead, Coq's logic is directly used when constructing LS^2 proofs. We instantiate LS^2 with protocol-specific data structures, predicates, and axioms, which are also encoded in Coq. Shallow embedding allows us to use Coq's libraries and tools and enables us to efficiently construct proofs.

We analyze the OPT protocols that use chained Message Authentication Codes (MACs) to provide source authentication and path validation for routers [29]. These protocols have two phases: key setup and packet forwarding. We prove the secrecy and authenticity of keys of the key setup phase, and origin and path authenticity properties of the forwarding phase. To the best of our knowledge, we are the first to formalize origin and path authenticity properties of packet forwarding provided by chained MACs, and construct machine-checkable proofs of these properties.

More concretely, this paper makes the following contributions:

- We encode LS^2 and basic constructs for reasoning about cryptographic protocols in Coq.
- We formalize and construct machine-checkable proofs of the secrecy and authenticity properties of the key setup protocol.
- We formalize and construct machine-checkable proofs of the origin and path authenticity properties of the forwarding protocol.

The rest of this paper is organized as follows. In Section 2, we review LS^2 and present the protocols that we verify. In Section 3, we show our Coq encoding of LS^2 and of protocol-specific reasoning constructs. Then, in Sections 4 and 5, we explain our formal definitions of security properties and discuss our proof techniques. We show attacks to alternative protocol designs identified during the proof process and discuss the connection between prevented attack scenarios and the verification results in Section 6. In the end, we discuss related work in Section 7.

The proof for the key setup protocol described in this paper has 1092 lines of Coq code, the proof for the forwarding protocol has 3788 lines of code, and the encoding of both protocols has 2657 lines of code. The basic definitions including the state monad and protocol-specific constructs are an additional 1566 lines of code. It took approximately two person months to complete the proofs. The Coq source files are available for download at the following URL: `http://www.ece.cmu.edu/ liminjia/net-verif`.

2. BACKGROUND

We briefly review the program logic we use for verification and explain the OPT source authentication and path validation protocol.

2.1 Reasoning Logics

LS^2 is a program logic for deriving properties of programs that run in parallel with other programs, including programs controlled by an adversary [23]. Subsequent work extends LS^2 and introduces a compositional reasoning framework for interface-confined adversaries [25]. A network consists of both trusted nodes that run the prescribed protocols and adversarial nodes that run arbitrary code. These adversaries are limited in their capabilities. We assume a variant of the Dolev-Yao network attacker model (explained in Section 2.2). Therefore, these two proof systems are well suited for reasoning about network protocols. We use the core reasoning principles formalized by Garg et al. [25] and refer to this proof system as LS^2 for the rest of this paper.

The semantics of LS^2 assumes interleaving executions of system components, some of which can be adversarial. Each component is called a *thread* and is assigned a unique thread ID. Each thread is represented as a tuple of the thread ID, the program that the thread is executing, and a local execution environment. A thread makes a transition step if its program makes a step, and the system makes a transition if one of the threads makes a transition. A *trace* is a sequence of transitions of the system. Each transition is associated with a time point, denoting the time when the transition takes effect. A system's behavior is modelled as the set of traces generated from the initial configuration.

The formal properties of a system are specified as properties of its traces, using first-order logic formulas. Predicates that specify actions of threads, such as sending messages, take a time point as an argument, indicating when that action happens. Using these time points, we can encode temporal logic formulas, and, therefore, specify safety properties (informally, nothing bad has happened so far). Many desired properties of protocols are safety properties.

Trusted principals are modelled as threads running programs corresponding to prescribed protocols, and adversaries are modelled as threads running arbitrary code. We explain how we encode adversary's capabilities in Section 3.2.

One key feature of LS^2 is that program assertions include both partial correctness and invariant assertions. A partial correctness assertion specifies properties of traces that contain the complete execution of that program. An invariant assertion specifies properties of traces that contain unfinished execution of that program. Invariant properties are crucial to proving security properties; safety properties should hold not only after the program terminates, but also throughout the execution of the program. Section 3 presents detailed inference rules for LS^2. For a full description of LS^2 and its soundness, please refer to Garg et al.'s work [23, 25].

LS^2 is a general logic that is meant to be instantiated with domain-specific definitions for reasoning about specific applications. To reason about protocols using cryptographic functions, LS^2 needs to be extended with relevant definitions and axioms. PCL [22] has definitions of many such data structures and axioms. For our proofs, we reuse and refine definitions from PCL.

2.2 Origin and Path Trace (OPT)

OPT is a protocol that supports source authentication and path validation [29]. We have verified both OPT and the extended OPT protocols proposed in [29]. In this paper, we focus on explaining the verification of the extended OPT, as it provides stronger security guarantees. The analysis of OPT is analogous, and we refer interested readers to our companion technical report [49] for these details. For the rest of this paper, we refer to the extended OPT protocol simply as OPT.

OPT enables all entities (e.g., routers, switches, middleboxes) on the path to authenticate the source and the content of the received packet, and to validate the path on which the packet traveled. OPT assumes that each router R in the network has a public-private key pair. A router R's key pair is denoted by $\mathsf{pk}(R)$, $\mathsf{pk}^{-1}(R)$. OPT has two phases: key setup and packet forwarding. During the key setup process, each intermediate router on the source-selected path sets up two symmetric keys to be shared with the source and the destination, respectively. The destination also sets up a symmetric key shared with the source. Once the key setup process is finished, the source, routers, and the destination use these keys to compute and verify MACs of forwarded packets.

2.2.1 Attacker Model

We use a variant of the Dolev-Yao attacker model. We consider attackers that can send and receive messages intended for them, compute cryptographic functions using the keys that they have, and decompose and construct messages based on the messages and keys they know. Unlike Dolev-Yao attackers, our attackers cannot intercept messages not meant for them, or inject messages into part of the network that they are not directly connected to. This is a reasonable model as our attackers represent malicious network nodes (e.g., ISPs).

2.2.2 Key Setup (DRKey Protocol)

The key setup protocol of the OPT protocol is called Dynamically Recreatable Key (DRKey) protocol. The pseudo code of the DRKey protocol is listed in Figure 1. We write S to denote the source node and D to denote the destination node. The source

```
$fverify(path, keysigs, pk, pk^{-1}, dir)$ =
  match ($path$, $keysigs$) with (), () ⇒ ret()
  | ($path'$, $rt$), ($keysigs'$, ($c$, $sig$)) ⇒
    $k$ = dec($c$, $pk^{-1}$);
    verify($sig$, ($k, pk, dir$), pk($rt$));
    $keys$ = $fverify(path', keysigs', pk, pk^{-1}, dir)$;
    ret($k, keys$)
  | _ ⇒ stuck

source() =
  ($D, pkS, pkS^{-1}, pkD, path$) = pickSession();
  send($S, D, pkS, pkD, path$, (), ());
  $keysigsS$ = recv;
  $keysS$ = $fverify((path, D), keysigsS, pkS, pkS^{-1}, upstream)$;
  acceptS(self, $D$, $path$, $keysS$, $pkS$)

dest() =
  ($S, D, pkS, pkD, path, keysigsS, keysigsD$) = recv;
  ($path', S', pkS', pkD^{-1}$) = lookupSession($pkD$, self);
  if $path' = path$ && $pkS = pkS'$ && $S = S'$
  then $keysD$ = $fverify(path, keysigsD, pkD,$
                          $pkD^{-1}, downstream)$;
       acceptD($S$, self, $path$, $keysD$, $pkD$);
       $kS$ = prf(SV$_S$(self), hash($pkS$));
       $ckS$ = enc($kS$, $pkS$);
       $sigS$ = sign(($kS, pkS, upstream$), pk$^{-1}$(self));
       send(($keysigsS$, ($ckS$, $sigS$)))

router() =
  ($S, D, pkS, pkD, path, keysigsS, keysigsD$) = recv;
  $kS$ = prf(SV$_S$(self), hash($pkS$));
  $ckS$ = enc($kS$, $pkS$);
  $sigS$ = sign(($kS, pkS, upstream$), pk$^{-1}$(self));
  $kD$ = prf(SV$_D$(self), hash($pkD$));
  $ckD$ = enc($kD$, $pkD$);
  $sigD$ = sign(($kD, pkD, downstream$), pk$^{-1}$(self));
  send($S$, $D$, $pkS$, $pkD$, $path$,
       ($keysigsS$, ($kS$,$sigS$)), ($keysigsD$, ($kD$, $sigD$)))
```

Figure 1: Key setup.

could choose different paths to send packets to the destination. To distinguish between these paths, the protocol uses the notion of sessions. For each session, the source chooses a path to forward packets along and generates a public-private key pair (pkS, pkS^{-1}); and the destination generates the pair (pkD, pkD^{-1}). pkS^{-1} and pkD^{-1} are kept secret. Subsequently, pkS and pkD are used to generate the symmetric keys between the source and each router, and between the destination and each router, respectively.

Each session is uniquely identified by the tuple (S, D, pkS, pkD, $path$). The source and the destination exchange pkS and pkD at the beginning of the key setup phase by exchanging signatures of relevant fields of the session tuple using their long-term key pairs. The source continues the key setup phase by sending the session tuple to the first router on the intended path. When the router receives the packet, it generates a symmetric key (kS) for the source and a symmetric key (kD) for the destination as follows.

$$\begin{aligned} kS &= \mathsf{prf}(\mathsf{SV}_S(\mathsf{self}), \mathsf{hash}(pkS)) \\ kD &= \mathsf{prf}(\mathsf{SV}_D(\mathsf{self}), \mathsf{hash}(pkD)) \end{aligned}$$

Here, "self" refers to the identity of the principal that runs this program. Each router R has two local secrets, $\mathsf{SV}_S(R)$ and $\mathsf{SV}_D(R)$. The former is used to compute the symmetric key shared with the source and the latter the key shared with the destination. We write prf to denote pseudo random functions. The advantage of generating keys in this manner is that routers do not need to store these keys, as they can be efficiently recomputed on the fly, assuming that pkS and pkD are contained in each packet header.

The router then encrypts the keys using the session public keys of the source and the destination, pkS and pkD, respectively, and signs the keys to attest to their authenticity. The router signs two triples: (kS, pkS, $upstream$) and (kD, pkD, $downstream$), where $downstream$ and $upstream$ indicate for whom the key is intended (i.e., $upstream$ for the source, and $downstream$ for the destination). The router appends the pair of the encrypted key and signature to the list of key and signature pairs already in the packet header and sends the packet to the next router.

When the destination receives a packet, it first checks that the session information (the source, the path, and source's session public key) contained in the packet is valid. Then, it decrypts the keys and verifies the signatures of the keys. If the verification succeeds, the destination accepts all the keys in the packet denoted $keysD$. The destination then generates a symmetric key for the source and sends to the source the pair of the encryption and signature of this key (kS, $sigS$), together with the encryption and signature pairs generated by routers. Similarly, the source only accepts the keys after successfully verifying all the signatures in $keysigsS$.

2.2.3 Forwarding

A code snippet of the forwarding protocol is presented in Figure 2. There are three main data structures: (1) PVFS (path validation field for the source), (2) OPV (origin and path validation field for the routers), and (3) PVFD (origin and path validation field for the destination). Each data structure is indexed with a natural number i corresponding to its position on the intended path. Given a path R_1, R_2,..., R_n, we write K_{Si} to denote the symmetric key that the source and router R_i have shared during key setup. We write K_D to denote the symmetric key that the source and the destination share. We write K_{Di} to denote the symmetric key that the destination and router R_i share. Validation fields are computed as follows.

$$\begin{array}{lcl} \mathrm{PVFS}_0 &=& \mathsf{mac}(K_D, (\mathsf{hash}(pkt))) \\ \mathrm{PVFS}_i &=& \mathsf{mac}(K_{Si}, (\mathrm{PVFS}_{i-1})) \\ \mathrm{OPV}_i &=& \mathsf{mac}(K_{Si}, (\mathrm{PVFS}_{i-1}, R_{i-1}, \mathsf{hash}(pkt))) \\ \mathrm{PVFD}_0 &=& \mathrm{PVFS}_0 \\ \mathrm{PVFD}_i &=& \mathsf{mac}(K_{Di}, (\mathrm{PVFD}_{i-1}, R_{i-1}, \mathsf{hash}(pkt))) \end{array}$$

The source S computes PVFS_0, PVFD_0, and all the OPV_i fields, and includes them in the packet header. The recursive function $compPvfOpvs$ (on the 3rd line) uses the keys shared between the source and the routers to compute the OPV_i and PVFS_i fields. S sends a packet that includes in its header: S, D, $path$, the hash of the source public key ($\mathsf{hash}(pkS)$), the hash of the destination public key ($\mathsf{hash}(pkD)$), PVFS_0, PVFD_0, the list of OPV_i for all routers on the path ($opvs$), and the payload of the packet pkt.

When router R_i receives a packet, it first recomputes the symmetric keys based on the key hash fields in the header as follows: K_{Si} = prf($\mathsf{SV}_S(R_i)$, kSh), and K_{Di} = prf($\mathsf{SV}_D(R_i)$, kDh). Then R_i computes its orgin path verification field opv: opv = mac(K_{Si}, ($pvfS$, R, hash(pkt))), where $pvfS$ is the path validation field in the header and R is the preceding router.

R_i only accepts the packet if opv is the same as the origin path validation field included in the header. Finally, R_i computes PVFS_i and PVFD_i using the $pvfS$ and $pvfD$ fields and sends out the packet with updated $pvfS'$ and $pvfD'$.

When the destination receives a packet, it searches its local state and finds the keys shared with the routers and the source, based

```
source() =
    (pkt, pkS, pkD, D, path, keys, kD) = pick;
    pvf0 = mac(kD, hash(pkt));
    (pvfS, opvs) = compPvfOpvs(path, keys, pkt, self, pvf0);
    send(self, D, path, hash(pkS), hash(pkD),
            pvf0, pvf0, opvs, pkt)

router() =
    R = pickNeighbor;
    (S, D, path, kSh, kDh, pvfS, pvfD, opvs, pkt) = recvFrom(R);
    fCkVrf(opvs, mac((prf($SV_S$(self), kSh)), (pvfS, R, hash(pkt)));
    acceptRt(pkt, pvfS, R, kSh);
    $pvfS'$ = mac((prf($SV_S$(self), kSh)), pvfS);
    $pvfD'$ = mac((prf($SV_D$(self), kDh)), (pvfD, R, hash(pkt)));
    send(S, D, path, kSh, kDh, $pvfS'$, $pvfD'$, opvs, pkt)

dest() =
    R = pickNeighbor
    (S, $D'$, $path'$, kSh, kDh, pvfS, pvfD, opvs, pkt) = recvFrom(R);
    keys = SessionlookUp(S, self, kSh, kDh, $path'$);
    ($pvfD'$, _) = fCkPvfD(pkt, mac(prf($SV_S$(self), kSh), hash(pkt)),
                               keys, $path'$, S);
    If self = $D'$ && $path'$ = ($path_1$,R) && pvfD = $pvfD'$
    then acceptDst(S, path, keys, pkt, pvfD, kSh, kDh,
                        (prf($SV_S$(self), kSh));
    else stuck
```

Figure 2: Code snippet for origin and path validation.

on the path, the source, and the hashes of the public keys in the packet. The destination computes PVFD_n based on the path, the hash of the payload, and the list of keys using a recursive function *fCkPvfD*. The destination accepts the packet only if the computed PVFD_n is the same as the one in the packet header.

3. ENCODING OF LS^2 AND PCL

We now present the encoding of LS^2 and definitions and axioms specific to analyzing protocols.

3.1 Shallow Embedding of LS^2

LS^2 inference rules derive invariant and partial correctness properties of a program that interacts with adversaries. There are two approaches to encode LS^2 in Coq. One is to define all constructs of LS^2 in Coq, and the other is via shallow-embedding, which uses much of Coq's logical reasoning capability directly and places LS^2-specific reasoning rules in a monad. The main advantage of the latter is that many of Coq's built-in libraries and automation techniques can be straightforwardly applied, and thus eases the process of constructing LS^2 proofs. One drawback is that the proof of the soundness of LS^2 cannot itself be encoded using this encoding scheme. However, this aspect is beyond the scope of this paper. The soundness of LS^2 has been proven manually by Garg et al. [23].

ST monad. LS^2 has four kinds of judgments: assertions about invariant properties of a program, assertions about partial correctness properties of a program, assertions about invariant properties of threads that remain idle (i.e., do not perform any actions such as sending, or signing messages), and first-order logic inference rules. Rules for the first three are encoded in Coq using a monad. For the fourth, we directly use Coq's first-order reasoning rules.

A state monad, denoted ST P Q, specifies properties of a program, where P is the invariant property and Q is the partial correctness property. Core definitions of the ST monad are shown in Figure 3. First we define the type tprop, which takes a thread ID and a beginning and an ending time point as arguments and returns a Coq proposition. tprop is the basic building block for specifying the trace properties of programs. An invariant property has type Inv, which is precisely tprop. The partial correctness type, denoted Pc, takes as an additional argument the type of the return value of a program. A program e has type ST (fun i ub $ue \Rightarrow \varphi_1$) (fun $(x : A)$ i ub $ue \Rightarrow \varphi_2$), if any trace containing unfinished execution of e by thread I between time U_B and U_E satisfies $\varphi_1[U_B, U_E, I/ub, ue, i]$, and any trace containing completed execution of e by thread I between time U_B and U_E satisfies the formula $\varphi_2[U_B, U_E, I, v/ub, ue, i, x]$ provided that e returns v at time U_E.

Assertions for silent threads are encoded using the type Silent. A proof of the type Silent (fun i ub $ue \Rightarrow \varphi$) means that formula $\varphi[U_B, U_E, I/ub, ue, i]$ is true on any trace where thread I idles between time U_B and U_E. The silent assertions are needed because in a distributed system threads execute interleavingly and are allowed to idle while other threads make progress.

STReturn is the assertion for the return statement. The invariant property states that before the return, the thread is idle. The partial correctness property states that this thread doesn't perform any actions while this return statement is being evaluated, and that the value being returned is v. STWeakening weakens trace properties. Both the invariant and partial correctness properties of program e can be weakened, as long as proofs can be constructed to show that the resulting properties are logically implied by the original properties. STBind is the assertion for sequencing statements. The partial correctness of a sequencing statement is the conjunction of the partial correctness properties of both statements, since the trace containing the complete execution of the sequence contains the complete execution of both statements. The invariant property is the disjunction of three cases: the program is idle, the first statement has not finished, and the first one finished but the second one has not. We use STFix to encode a fixed point function. Finally, we use the following syntactic sugar for ease of presentation: $x \leftarrow c_1; c_2$ for sequencing and $\{\{\{c\}\}\}$ for weakening.

Protocols are encoded using these monads. Each instruction is encoded as a ST monad specifying its properties. These instructions are then sequentially composed using STBind. For instance, the following is the encoding of a fragment of the destination's key setup program that encrypts and signs the key shared between the destination and the source, and sends the list of keys to the source.

```
ckS ← STAsymEnc (symkeyToMsg (msgToSymKey keyS)) pks;
sigS ← STSign (pair (prinToMsg upstream)
                  (pair (symkeyToMsg (msgToSymKey keyS))
                        (pkeyToMsg pks)))
                   (sk p);
x ← STSend (pair keysigsS (pair ckS sigS));
STReturn unitMsg
```

Here, STAsymEnc, STSign, and STSend are the state monads for encrypting and signing messages, and sending messages over the network. Their type specifications state the trace properties of these actions. The above program maps straightforwardly to the pseudo code in Figure 1. The tricky part is to specify a type for the program and to prove, using LS^2, that the type specification is correct. Once Coq type checks the program against its type, the program is verified to have the invariant and partial correctness properties specified in the type.

Honesty. The HONESTY rule asserts properties of traces that contain the execution of trusted programs (shown below). Predicate honestThread i $prog$ t states that thread i starts to run program $prog$ at time t. The Honesty rule states that after the thread starts,

```
Definition tprop := threadId → time → time → Prop.
Definition Pc T := T → tprop.
Definition Inv := tprop.
Parameter ST : Inv → ∀ T, Pc T → Set.
Parameter Silent : Inv → Set.

Parameter STReturn : ∀T (v: T) (phi1: Inv) (phi2: Inv),
  Silent phi1 → Silent phi2 →
  ST phi1 (fun x i ub ue ⇒ phi2 i ub ue ∧ x = v).

Parameter STWeaken : ∀ T (inv1 inv2: Inv) (pc1 pc2: Pc T),
  ST inv1 pc1
  → (∀i ub ue, inv1 i ub ue → inv2 i ub ue)
  → (∀x i ub ue, pc1 x i ub ue → pc2 x i ub ue)
  → ST inv2 pc2.

Parameter STBind : ∀T1 T2 (phi: Inv) inv1 (pc1: Pc T1)
  inv2 (pc2: T1→ Pc T2), Silent phi →
  ST inv1 pc1 → (∀ x:T1, ST (inv2 x) (pc2 x)) →
  ST (fun i ub ue ⇒
      phi i ub ue
      ∨ (∃um, ub ≤ um ∧ um≤ue ∧ phi i ub um ∧ inv1 i um ue)
      ∨ (∃um1 um2 y, ub≤ um1 ∧ um1 ≤ um2 ∧ um2≤ ue ∧
             phi i ub um1 ∧ pc1 y i um1 um2 ∧ inv2 y i um2 ue)
     (fun x i ub ue ⇒ ∃um1 um2 y, ub≤um1 ∧ um1≤um2 ∧
                        um2≤ ue ∧ phi i ub um1 ∧
                        pc1 y i um1 um2 ∧ pc2 y x i um2 ue).

Parameter STFix : ∀T1 T2 inv1 (pc1: ∀x:T1, Pc (T2 x))
  (F: (∀v: T1, ST (inv1 v) (pc1 v))→ (∀v: T1, ST (inv1 v) (pc1 v)))
  (v: T1), ST (inv1 v) (pc1 v).
```

Figure 3: Coq encoding of ST monad for LS^2

the invariant property of the thread holds throughout the execution.

```
Parameter honestThread: ∀T inv (pc:Pc T), threadId →
ST inv pc → time → Prop.

Axiom HONESTY: ∀T inv (pc: Pc T) ub ue i,
ue > ub → ∀e: ST inv pc, honestThread i e ub →inv i ub ue.
```

Rely-guarantee. Rely-guarantee reasoning principles (described by Garg et al. [25]) are needed to prove invariant properties of the form $\forall t$:time, $\varphi(t)$. To prove that at all time t, the property φ holds, we need to prove (1) that φ holds initially and (2) that φ has held before time t implies that φ holds at t. By induction over time, we can conclude from (1) and (2) that $\forall t :$ time, $\varphi(t)$. In a distributed system, (2) can be further refined into two conditions concerning the local guarantees of a set of threads We list the conditions below. We define time as natural numbers because we conflate time with the number of steps the system has taken so far.

($\mathbf{RG}_1$) $\varphi(0)$
($\mathbf{RG}_2$) $\forall u, (\forall u', u'{<}u \rightarrow \varphi(u')) \rightarrow (\forall i, \iota(i) \rightarrow \psi(i,u))$
($\mathbf{RG}_3$) $\forall u, (\forall u', u'{<}u \rightarrow \varphi(u')) \rightarrow (\forall i, \iota(i) \rightarrow \psi(i,u)) \rightarrow \varphi(u)$

We call $\varphi(u)$ a global invariant, and $\psi(i,u)$ a local guarantee by thread i. Condition $\mathbf{RG}_1$ ensures that the invariant property holds initially. Condition $\mathbf{RG}_2$ checks that local guarantees are met assuming the global invariant has held so far. Condition $\mathbf{RG}_3$ checks that local guarantees imply the global invariant. Using the above conditions, we can prove that $\forall u, \varphi(u)$ by inducting over time. Here, $\iota(i)$ selects the set of relevant threads whose local behavior is central to the security of the protocols. $\psi(i,u)$ is a property of the trace that is specific to thread i. Condition $\mathbf{RG}_3$ is simpler than the corresponding condition described by Garg et al. [25]: it does not mention violations of the invariant property. We have proved in Coq that conditions $\mathbf{RG}_1$–$\mathbf{RG}_3$ imply $\forall u, \varphi(u)$.

```
Inductive hassymKey: threadId → symKey → time → Prop :=
...
with has: threadId → msg → time → Prop :=
...
| hasComp: ∀ i m t, mayComp i m t → has i m t
| hasRecv: ∀ i m t, recv i m @ t → has i m t
| hasHad: ∀ i m t1 t2, has i m t1 → t1 < t2 → has i m t2
with mayComp: threadId → msg → time → Prop :=
| compSymEnc: ∀ i k m t,
      has i (symkeyToMsg k) t → has i m t →
      mayComp i (symEncMsg m k) t
| compSymDec: ∀ i k m t,
      has i (symkeyToMsg k) t → has i (symEncMsg m k) t →
      mayComp i m t
| compSig: ∀ i k m t,
      has i (skeyToMsg k) t → has i m t →
      mayComp i (signMsg m k) t
| compMac: ∀ i k m t,
      has i (symkeyToMsg k) t → has i m t →
      mayComp i (macMsg k m) t
| compPair: ∀ i m1 m2 t,
    has i m1 t → has i m2 t → mayComp i (pair m1 m2) t
| compProj1: ∀ i m1 m2 t,
    has i (pair m1 m2) t → mayComp i m1 t
...
```

Figure 4: Principal's knowledge.

3.2 Protocol Specific Constructs

Inductively defined data structures. Network messages are defined as an inductive data type. The constructors for messages include principals and keys; constructors for creating ciphertext, hashes, signatures, MACs, and pseudo-random functions; and a constructor for pairs. An important definition is a principal's knowledge, which is crucial to modeling adversary capabilities. We define has $i\ m\ t$ as an inductively defined data type, which means that thread i knows message m at time t. We show part of the definition in Figure 4. A principal has a message if it has received that message earlier, has generated it, or can compute it. For instance, if a principal has an encryption key k and a message m, then it can compute the encryption of m using k; if a principal has a cyphertext and the decryption key, then it can obtain the plaintext. Using these definitions, we can prove lemmas about the properties of cryptographic functions. For instance, we can prove the following lemma by induction over time and the structure of has.

```
Lemma hasMacCnt:∀ i m t m1 k,
 has i m t → contain m (macMsg k m1) →
 has i (symkeyToMsg k) t
 ∨∃m2 tr, tr≤t ∧ recv i m2 @ tr ∧
           contain m2 (macMsg k m1).
```

This lemma states that if a thread i has a message that contains a MAC message, then it either has the key for computing the MAC, or it has received a message that contains that MAC earlier. Here, contain $m1$ $m2$ means that $m2$ can be computed from $m1$ and it is inductively defined.

Axioms. We assert several axioms that can only be proven sound based on the semantics of actions. We list a few below. Axiom Recv states that if one receives a message, then someone must have sent it. Axiom sendHas states that if i sends a message m, then i must have that message. Axiom Verify states that if sig is the signature of m using the public key of an honest principal p, then some thread owned by p must have signed m using p's private key.

Axiom Recv: ∀ i m t, recv i m @ t → ∃ j t', t'<t ∧ send j m @ t'.
Axiom sendHas: ∀ i m t, send i m @ t → has i m t.
Axiom Verify: ∀ i m p t sig, verify i sig m pk(p) t ∧ honest p →
∃ j t', t'<t ∧ owns j p∧ sign j m pk^{-1}(p) sig t'.

These axioms together with the definition of has specify our adversaries' capabilities.

4. THE SECRECY AND AUTHENTICITY OF THE DRKEY PROTOCOL

Two important properties of the DRKey protocol are *Secrecy:* each symmetric key K_{Si} (resp. K_{Di}) generated by the router R_i is known only to S (resp. D) and R_i, and the symmetric key K_D generated by the destination is known only to S and D; and *Authenticity:* the list of keys accepted by the source (resp. destination) is computed using the correct public key of S (resp. D) for that session and the router's local secret for the source (resp. destination).

4.1 Property specifications

We summarize main predicates in Figure 5. safeMsg $M\ s\ K$ is borrowed from earlier work on verifying secrecy properties using PCL [41]. We define it inductively over the structure of M. For instance, the following two rules define when s is safe in an encrypted message. Here pubencMsg $m\ pk$ denotes the encryption of m using public key pk. Either s is already safe in m, or pk's corresponding private key pk^{-1} belongs to the set K. For compactness, definitions are shown in the style of inference rules, rather than the Coq code.

$$\frac{\mathsf{safeMsg}\ m\ s\ K}{\mathsf{safeMsg}\ (\mathsf{pubencMsg}\ m\ pk)\ s\ K}$$

$$\frac{\mathsf{asymKeyPair}\ pk\ pk^{-1} \qquad pk^{-1} \in K}{\mathsf{safeMsg}\ (\mathsf{pubencMsg}\ m\ pk)\ s\ K}$$

Next we define HasOnlyPath $p\ path\ keys$ to state that every key k in $keys$, such that k is the i^{th} key in $keys$, is only known to the principal p and the i^{th} principal r in $path$, if r is honest.

$$\frac{}{\mathsf{HasOnlyPath}\ p\ ()\ ()}$$

$$\frac{\mathsf{hasOnly}\ (p :: r :: nil)\ k \qquad \mathsf{HasOnlyPath}\ p\ path\ keys}{\mathsf{HasOnlyPath}\ p\ (path, r)\ (keys, k)}$$

Finally we define GenkPath $keys\ path\ pk\ sv$ to mean that for every k in $keys$, where k is the i^{th} key in $keys$, k must be the message generated by applying the pseudo random function to the i^{th} principal on $path$, r's secret $sv(r)$ and the hash of the key pk, if r is honest. Here sv is either SV_S or SV_D. PrfMsg is the constructor for messages generated using pseudo random functions.

$$\frac{}{\mathsf{GenkPath}\ ()\ ()\ pk\ sv} \qquad \frac{\mathsf{honest}\ r \qquad k = \mathsf{PrfMsg}\ sv(r)\ pk \quad \mathsf{GenkPath}\ keys\ path\ pk\ sv}{\mathsf{GenkPath}\ (keys, k)\ (path, r)\ pk\ sv}$$

The secrecy and authenticity properties of the keys are defined in terms of HasOnlyPath $p\ path\ keys$ and GenkPath $keys\ path\ pk\ sv$. The following two theorems state that our key setup protocol has key secrecy and authenticity for both the source and the destination.

THEOREM 1 (KEY SECRECY AND AUTHENTICITY (DST)). *For all $i\ S\ D\ path\ keysD\ pkD\ t$,* owner $i\ D$*,* honest D*, and* acceptedD $i\ S\ D\ path\ keysD\ pkD@t$ *imply all of the following*

- GenkPath $keysD\ path\ pkD\ \mathsf{SV}_D$
- *there exists a pkS such that* session $S\ D\ path\ pkS\ pkD$ *and* HasOnlyPath $D\ path\ keysD$

Theorem 1 states that when the destination (D) finishes validating the keys ($keysD$) w.r.t. the path ($path$) and the session key (pkD), it is the case that (1) the list of keys it accepts is generated by intermediate routers if they are honest, (2) each key is known only to the destination and the router who generated it (if that router is honest), and (3) the current session is identical to the session that the destination has shared with the source.

When the source accepts a list of keys, a similar property of the keys holds, stated below.

THEOREM 2 (KEY SECRECY AND AUTHENTICITY (SRC)). *For all $i\ S\ D\ path\ keysS\ pkS\ t$,* owner $i\ S$*,* honest S*, and* acceptedS $i\ S\ D\ path\ keysS\ pkS@t$ *imply all of the following*

- GenkPath $keysS\ (path, D)\ pkS\ \mathsf{SV}_S$
- *there exists a pkD such that* session $S\ D\ path\ pkS\ pkD$ *and* HasOnlyPath $S\ (path, D)\ keysS$

4.2 Proofs of Key Secrecy and Authenticity

We explain proofs of Theorem 1. Theorem 2 can be proved similarly. We make the following assumptions, which are encoded as axioms in Coq. (KeyS) and (KeyD) state that if a principal is honest and declares that a public-key component of a key pair is being used as a session key, then the corresponding private key must be known only to that principal. (SecS) and (SecD) state that the local secrets of an honest router are known only to itself.

(KeyS) $\forall S\ D\ path\ pkS\ pkD\ pkS^{-1}$,
honest S ∧ session $S\ D\ path\ pkS\ pkD$∧
asymKeyPair $pkS\ pkS^{-1}$ → hasOnly $(S :: nil)\ pkS^{-1}$
(KeyD) $\forall S\ D\ path\ pkS\ pkD\ pkD^{-1}$,
honest D ∧ session $S\ D\ path\ pkS\ pkD$∧
asymKeyPair $pkD\ pkD^{-1}$ → hasOnly $(D :: nil)\ pkD^{-1}$
(SecS) $\forall rt$, honest rt → hasOnly $(rt :: nil)\ \mathsf{SV}_S(rt)$
(SecD) $\forall rt$, honest rt → hasOnly $(rt :: nil)\ \mathsf{SV}_D(rt)$

honest p	Threads owned by the principal p run either the source, destination, or router's key setup or forwarding program.
ownerIn $i\ P$	One of the principals in the list of principals P owns the thread i.
hasOnly $P\ s$	s is only known to principals in P.
safeMsg $M\ s\ K$	All occurrences of s in message M are protected by one of the keys in K (s is safe in M w.r.t. K)
sendsSafeMsg $i\ s\ K$	s is safe in all messages sent by thread i w.r.t. K.
safeNet $s\ K\ u$	s is safe in all messages sent by any thread (including adversarial ones) w.r.t. K up to time u.

Figure 5: Predicates used in key setup.

Authenticity proofs. To prove the authenticity property of the list of keys accepted by the destination, we first consider the authenticity property of one key (Lemma 3). Assume that rt is an honest router and pkD is a session key for the destination D. If sig is a signature of the tuple $(k, pkD, downstream)$ signed using the private key of rt, then we can prove that k is computed by applying the pseudo random function to the correct arguments.

LEMMA 3 (KEY AUTHENTICATION). *For all* $i\ sig\ d\ rt\ pkD\ k\ t$, $\mathsf{verify}\ i\ sig\ (k, pkD, downstream)\ \mathsf{pk}(rt)@t$, $\mathsf{honest}\ rt$ *imply* $k = \mathsf{PrfMsg}\,\mathsf{SV}_S(rt)\ pkD$

To prove Lemma 3, we use Axiom Verify and conclude that an honest thread must have signed the tuple. Then we use LS^2 inference rules and prove that an honest node only signs a key, k, that is generated by applying the pseudo random function to the thread's local secret and the hash of the public key that it encrypts k with. The conclusion follows directly. By induction over the length of the trace, we can prove the first condition in Theorem 1.

Secrecy proofs. The secrecy proofs follow similar strategies as described by Garg et al. [25], where rely-guarantee reasoning principles are used to show that the keys have been safe in all network messages w.r.t. secret keys only known to the trusted principals. We first consider one key. Assume that rt and D are honest. For a given session, we prove that the symmetric key k generated by rt to be shared with D is always protected by the private key pkD^{-1}, formalized below. Recall pkD is the public session key for D in that session.

LEMMA 4 (SAFENET). *For all* $S\ D\ path\ pkS\ pkD\ pkD^{-1}\ rt\ k$, $\mathsf{honest}\ rt$, $\mathsf{honest}\ D$, $\mathsf{asymKeyPair}\ pkD\ pkD^{-1}$, $k = \mathsf{PrfMsg}\,\mathsf{SV}_D(rt)\ pkD$ *and* $\mathsf{session}\ S\ D\ path\ pkS\ pkD$ *imply that for all* u, $\mathsf{safeNet}\ k\ (pkD^{-1} :: nil)\ u$

The proof of the above lemma uses the rely-guarantee reasoning principles ($\mathbf{RG}_1$-$\mathbf{RG}_3$) outlined in Section 3.1 with

$$\begin{aligned} \iota(i) &= \exists node, \mathsf{owner}\ i\ node \wedge \mathsf{honest}\ node. \\ \psi(i,u) &= \forall rt, \mathsf{sendsSafeMsg}\ \ i\ \mathsf{PrfMsg}\,\mathsf{SV}_D(rt)\ pkD) \\ &\qquad\qquad (pkD^{-1} :: nil)@u. \end{aligned}$$

The three conditions in rely-guarantee principles (introduced in Section 3.1) are now instantiated as follows: Condition $\mathbf{RG}_1$ says that initially these keys are safe in all network messages. This is trivially true because no message is sent at time 0. Condition $\mathbf{RG}_2$ says that if a router-generated key is protected by pkD^{-1} in all network messages, up to u (excluding u), then honest threads will protect this key in all messages they send out at time u. This condition can be proved by reasoning about the protocol code. Condition $\mathbf{RG}_3$ says that if a router-generated key is protected by pkD^{-1} in all messages that have been sent out at any time before u, and that all honest nodes protect this key, using pkD^{-1} in all messages they send at time u, then this key is also protected by pkD^{-1} in all messages sent by all threads up to time u inclusive. Next we explain how $\mathbf{RG}_3$ is proven.

We prove another lemma (shown below), which states that if the key $\mathsf{PrfMsg}\,(\mathsf{SV}_D(rt))\ pk$, denoted m, has been safe in all network messages protected by keys in K, then anyone with knowledge of m must either have the router's secret or a key k in K.

LEMMA 5 (POS). *For all* $i\ m\ rt\ pk\ t$, $\mathsf{safeNet}\ m\ K\ t$, $m = \mathsf{PrfMsg}\,\mathsf{SV}_D(rt)\ pk$, *and* $\mathsf{has}\ i\ m\ t$ *imply* $\mathsf{has}\ i\ \mathsf{SV}_D(rt)\ t$ *and there exists* k *in* K *such that* $\mathsf{has}\ i\ k\ t$

The proof of $\mathbf{RG}_3$ uses Lemma 5 to show that if a thread i can compute this key at time u, it must be the case that i knows either rt's local secret value or pkD^{-1}. According to our axioms, we know that i must be owned by either rt or D, both of which are honest. Therefore, condition $\mathbf{RG}_3$ holds.

From Lemma 4 and Lemma 5, we can prove that the key k generated by rt for D is known only to rt and D (formalized below).

LEMMA 6 (HASONLY). *For all* $S\ D\ path\ pkS\ pkD\ pkD^{-1}\ rt\ k$, $\mathsf{honest}\ rt$, $\mathsf{honest}\ D$, $\mathsf{asymKeyPair}\ pkD\ pkD^{-1}$, $k = \mathsf{PrfMsg}\,\mathsf{SV}_D(rt)\ pkD$ *and* $\mathsf{session}\ S\ D\ path\ pkS\ pkD$ *imply* $\mathsf{hasOnly}\ (rt :: D :: nil)\ k$

The second condition in Theorem 1 can be proved by induction on the length of the trace and Lemma 6. Combining this with the authenticity proofs, we have successfully verfied that the DRKey protocol has the desired security properties.

5. THE ORIGIN AND PATH AUTHENTICITY OF THE OPT PROTOCOL

We prove two origin and path authenticity theorems for packet forwarding: one for intermediary routers and the other for the destination. We only present the source-based origin and path authenticity for routers. The corresponding theorem for the destination differs only in that source is not assumed to be trusted. A direct benefit of constructing proofs is that we are able to verify these topology-independent properties.

Property specifications. A list of key predicates is summarized in Figure 6. Next we define the predicate ($\mathsf{goodPathSrc}\ n\ S\ D\ pkS\ pkt\ path\ t$) to mean that a packet with payload pkt has path authenticity w.r.t. source S, destination D, the session public key pkS, a path $path$, and a time point t at index n. It is inductively defined over n, a position on the path. pkS is S's session public key and $path$ is the path on which the source has intended to send the packet in this session. The time point, t, is used to enforce the order in which the packet has traversed the network.

$$\frac{}{\mathsf{goodPathSrc}\ 0\ S\ D\ pkS\ pkt\ path\ t}$$

$$\frac{\begin{array}{c}\sim \mathsf{honest}\ R \qquad \mathsf{pathN}\ path\ R\ (n{+}1) \\ \mathsf{goodPathSrc}\ n\ S\ D\ pkS\ pkt\ path\ t\end{array}}{\mathsf{goodPathSrc}\ n{+}1\ S\ D\ pkS\ pkt\ path\ t}$$

$$\frac{\begin{array}{l}\mathsf{honest}\ R \qquad \mathsf{pathN}\ path\ R\ (n{+}1) \qquad \mathsf{owner}\ i\ R \\ \text{if } n = 0 \text{ then } rt' = S \\ \text{if } n = n'{+}1 \text{ then } \mathsf{pathN}\ path\ rt'\ n \\ \mathsf{recvFrom}\ i\ rt'\ (S', D', path', \mathsf{hashMsg}(pkS), \\ \qquad\qquad kDh, pvfS, pvfD, opvs, pkt)@tr \\ tr < t \qquad ts \leq t \qquad ts > tr \\ \mathsf{send}\ i\ (S', D', path', \mathsf{hashMsg}(pkS), kDh, \\ \qquad pvfS1, pvfD1, opvs, pkt)@ts \\ pvfS1 = \mathsf{macMsg}\,(\mathsf{PrfMsg}\,\mathsf{SV}_S(R)\ \mathsf{hashMsg}(pkS))\ pvfS \\ pvfD1 = \mathsf{macMsg}(\mathsf{PrfMsg}\,\mathsf{SV}_D(R)\ kDh) \\ \qquad\qquad\qquad (pvfD, rt', \mathsf{hashMsg}(pkt)) \\ \mathsf{goodPathSrc}\ n\ S\ D\ pkS\ pkt\ path\ tr\end{array}}{\mathsf{goodPathSrc}\ n{+}1\ S\ D\ pkS\ pkt\ path\ t}$$

In the base case when $n = 0$, there are no additional requirements. There are two inductive cases for when $n = j + 1$. Let the j^{th} and $(j+1)^{th}$ routers on $path$ $(R_1, R_2, \cdots, R_n)$ be R_j and R_{j+1}, respectively. When R_{j+1} is dishonest, pkt is required to have path authenticity w.r.t. S, $path$, and t at index j, and there

pathN $path\ R\ n$	Router R is the n^{th} node on $path$ from the left ($path$=R_1,R_2,...,R_n).
acceptRt $i\ pkt\ pvfS\ rt'\ kSh$ @ t	Thread i has accepted payload pkt from router rt' with PVFS = $pvfS$ verified using key kSh at time t.
flagSRsF $S\ path\ keys\ pkS$	S has set up keys $keys$ for path $path$ using public key pkS.
flagSR $k\ S\ rt\ pkS$	S has set up key k with router rt using public key pkS.
session $S\ D\ path\ pkS\ pkD$	S and D has agreed on a session to forward along $path$ based on key pkS (for S) and pkD (for D).
goodPvfS $pvfS\ S\ D\ pkS\ path\ keys\ pkt$	$pvfS$ has the correct format of PVFS$_n$ w.r.t. $S\ D\ pkS\ path\ keys\ pkt$
goodVrf $opv\ S\ D\ pkS\ path\ keys\ pkt\ k\ rt$	opv has the correct format of OPV$_n$ w.r.t. $S\ D\ pkS\ path\ keys\ pkt\ k\ rt$

Figure 6: Predicates in forwarding

is no further requirement for router R_{j+1}. When R_{j+1} is honest, there exist two time points tr and ts earlier than t, such that router R_{j+1} receives from R_j payload pkt at tr and sends pkt out at a later time ts, and pkt has path authenticity w.r.t. S, $path$, and tr at index j.

Theorem 7 states that the forwarding protocol provides origin and path validation for each intermediary router. It is source-based validation because S needs to be honest. However, we do not assume the destination to be trusted.

THEOREM 7 (SOURCE-BASED OPT). *For all S, D, rt, pkD, kD, pkS^{-1}, path, keys, pkt, pkS, pvfS rt$'$, ta, i,*

- session $S\ D\ path\ pkS\ pkD$
- asymKeyPair $pkS\ pkS^{-1}$
- honest S, honest rt
- flagSRsF S $(path,\ D)$ $(keys,\ kD)$ pkS
- owner $i\ rt$, acceptRt $i\ pkt\ pvfS\ rt'$ hashMsg(pkS) @ ta

then exists n j ms ts opvs pvf0 kD pkD$'$, such that

- pathN $(path,\ D)$ $(S\ n)$ rt
- goodPathSrc $n\ S\ D\ pkS\ pkt$ $(path,\ D)$ ta
- $ts \leq ta$, owner $j\ S$, send $j\ ms$ @ ts
- $pvf0$ = macMsg kD (hashMsg(pkt))
- ms = $(S,\ D,\ path,$ hashMsg(pkS), hashMsg(pkD), $pvf0,\ pvf0,\ opvs,\ pkt)$.

Theorem 7 states that if a router accepts a packet, then the payload, pkt, originates from S and has traversed all the honest nodes on the path intended by S, up to the router, in the correct order.

High-level descriptions of proofs. The correctness of the forwarding protocol relies on the following properties related to verified MACs. **MACProp** 1: once the router R_i validates the origin and path validation field (opv) in the packet header, opv must be the correct OPV$_i$ as shown in Section 2.2.3. **MACProp** 2: a validated origin and path validation field (opv) indicates that the packet must have originated from the source. **MACProp** 3: The presence of (PVFS$_{i-1}$) of the correct format indicates that the packet must have traversed the path from S to R_1 to R_{i-1}.

MACProp 1 holds because opv is a MAC that can only be computed by S, and S's program guarantees to only compute OPV fields of the right form. **MACProp** 2 holds for the same reason. **MACProp** 3 holds because only R_k and S have the key K_{Sk} to compute PVFS$_{i-1}$, where $0<k<i$. From **MACProp** 1, we can derive that once the router R_i verifies the OPV$_i$ field, it must be the case that the path validation field it received in the packet is the same as PVFS$_{i-1}$. Origin and path authenticity follows from **MACProp** 2 and **MACProp** 3. An interesting proof technique is the use of rely-guarantee reasoning to prove **MACProp** 1–3, as they are invariant properties about MACs in PVFS, PVFD, and OPV. Next we show three lemmas that formalize these properties.

Proof of MACProp 1. We define a predicate sendsSafeVrf $i\ S\ D\ rt\ pkS\ opv\ t$ stating that if a thread i sends out a message m that contains a message, opv, which is the MAC of some message $pvfS$, the hash of pkt, and a principal rt' using key k, then opv must be a valid origin and path validation field (OPV) for some $path$ and $keys$, w.r.t. $S\ D\ pkS\ pkt$ and k.

sendsSafeVrf $i\ S\ D\ rt\ pkS\ opv\ t$ =
$\forall k\ m\ pkt\ pvfS\ rt'$,
  send $i\ m$ @ $t \rightarrow$ contain $m\ opv \rightarrow$
  opv = macMsg k ($pvfS$, hashMsg(pkt), rt') $\rightarrow$
  $\exists\ path\ keys$, goodVrf $opv\ S\ D\ pkS\ path\ keys\ pkt\ k\ rt$.

Next we define safeNetVrf to mean that sendsSafeVrf $i\ S\ D\ rt\ pk\ opv\ t$ has held since the beginning till u for all threads i. safeNetVrf is the first invariant property that we prove for origin path validation field (OPV), which is **MACProp** 1 stated above.

safeNetVrf $S\ D\ rt\ pk\ opv\ u$ =
$\forall\ i\ u', u' \leq u \rightarrow$ sendsSafeVrf $i\ S\ D\ rt\ pk\ opv\ u'$.

The following lemma states that the formula (safeNetVrf $S\ D\ rt\ pk$ (macMsg k (pvf,rt',hp)) u) holds for all time u, provided that k is the symmetric key computed using pkS and the secret of the router rt. The first two premises are assumptions about the key setup for this session: S and D have decided that they will use pkS (for S) and pkD (for D) to forward along path $path$; further, pkS and pkS^{-1} are an asymmetric key pair where pkS is the public key in this pair. These two premises are needed for using key secrecy and authenticity properties of the setup phase. The next two premises state the honesty assumption about the principals; both S and rt have to run the correct forwarding protocol.

LEMMA 8 (SAFEVRF). *For all rt S D k pkS pkD pkS^{-1} path,* session $S\ D\ path\ pkS\ pkD$, asymKeyPair $pkS\ pkS^{-1}$, honest rt, honest S, *and k =* (PrfMsg SV$_S(rt)$ hashMsg(pkS)) *imply for all u pvfS rt' hp,*
safeNetVrf $S\ D\ rt\ pkS$ (macMsg k ($pvfS$,rt',hp)) u.

This lemma is proven using rely-guarantee reasoning. The threads selection function $\iota(i)$ is (ownerIn i (S::rt::nil)). The local guarantee $\psi(i,u)$ is

$\forall pvf\ rt'\ hp$,
  sendsSafeVrf $i\ S\ D\ rt\ pkS$ (macMsg k ($pvfS$,rt', hp)) u

The three conditions in rely-guarantee reasoning are as follows: Condition **RG**$_1$ states that initially, if any thread i sends out a message that contains a validation field, then it has the right format.

Condition $\mathbf{RG}_2$ states that if up to time u, all validation fields that have been sent out have the correct format, then threads owned by honest S and R send validation fields of the correct format. Condition $\mathbf{RG}_3$ states that if up to time u, all validation fields that have been sent out have the correct format and that all honest principals S and R send out correct validation fields at time u, then all threads (including malicious ones) only send out correct verification fields at time u.

The proof of $\mathbf{RG}_1$ uses the axiom that initially there are no send actions. The proof of $\mathbf{RG}_2$ relies on reasoning about the protocol code, and $\mathbf{RG}_3$ uses the properties of MAC to show that if a thread i sends out a validation field that is computed using key k, then i must either (i) be owned by S and R, or (ii) have received it earlier. In either case, if i sends out a verification field, it must have the correct format. Case (i) is proven using LS^2 rules to reason about honest protocol code. Case (ii) is proven by directly using the assumption that safeNetVrf has held so far.

Proof of MACProp 2. Next, we define a second invariant property related to the orgin and path validation field. This property is central to proving source authenticity. sendsSafeVrfT and safeNetVrfT follow similar format as sendsSafeVrf and safeNetVrf. The difference is that instead of asserting goodVrf, we assert goodVrfT $S\ D\ pk\ pkt\ t$.

$\mathsf{sendsSafeVrfT}\ i\ S\ D\ rt\ pkS\ opv\ t =$
$\forall k\ m\ pkt\ pvfS\ rt',$
$\quad \mathsf{send}\ i\ m\ @\ t \rightarrow \mathsf{contain}\ m\ opv \rightarrow$
$\quad opv = \mathsf{macMsg}\ k\ (pvf, rt', \mathsf{hashMsg}(pkt)) \rightarrow$
$\quad \mathsf{goodVrfT}\ S\ D\ pkS\ path\ pkt\ t.$

$\quad \mathsf{safeNetVrfT}\ S\ D\ rt\ pkS\ opv\ u =$
$\quad \forall\ i\ u', u' \leq u \rightarrow \mathsf{sendsSafeVrfT}\ i\ S\ D\ pkS\ opv\ u'.$

$\mathsf{goodVrfT}\ S\ D\ pkS\ pkt\ t =$
$\exists\ j\ m\ ts\ path\ keys\ opvs\ pkD\ pvf\ kD,$
$\quad ts \leq t \wedge \mathsf{owner}\ j\ S \wedge \mathsf{send}\ j\ m@ts\wedge$
$\quad \mathsf{flagSRsF}\ S\ (path, dst)\ (keys, kD\ pkS \wedge$
$\quad pvf0 = \mathsf{macMsg}\ kD\ (\mathsf{hashMsg}(pkt)) \wedge$
$\quad m = (S, D, path, pkt, pkS, pkD, pvf0, pvf0, opvs)$

The predicate goodVrfT $S\ D\ pkS\ pkt\ t$ is true if the source S has sent out a packet of the right format, where the initial PVF_0 is computed using the key (kD) established between S and D for this session. The flagSRsF predicate helps us identify arguments in the packet that S has sent out, as well as kD.

The following lemma states that safeNetVrfT is an invariant property under the assumptions about this session and the honesty of principals. The proof is similar to that of Lemma 8.

LEMMA 9 (SAFEVRFT). *For all $rt\ S\ D\ k\ pkS\ pkD\ pkS^{-1}$ path,* session $S\ D\ path\ pkS\ pkD$, asymKeyPair $pkS\ pkS^{-1}$, honest rt, honest S, *and* k = (PrfMsg $\mathsf{SV}_S(rt)$ hashMsg(pkS)) *imply* $\forall\ u\ pvfS\ rt'\ hp$,
safeNetVrfT $S\ D\ rt\ pkS$ (macMsg k ($pvfS$,rt', hp)) u.

Proof of MACProp 3. The final lemma is about an invariant property associated with a path validation message. Similar to the previous two lemmas, we first define the invariant property PVFINV $kS\ pkS\ pkt\ pvfS\ pvfS'\ S\ D\ t$, where kS is the key shared between a router and S, pkS is S's public session key, $pvfS$ is the MAC that is the path validation field, and $pvfS'$ is the path validation field used to compute $pvfS$. We omit the formal definition for PVFINV. It states that either S has sent out an initial packet where $pvfS'$ is the hash of the payload; or there exists a router rt that has

- received a packet of the correct packet format and
- *pvf'* is the PVFS_{i-1} in the packet and
- a verification field in the packet is of the form and $\mathsf{macMsg}\ K_{Si}\ (pvf', R_{i-1}, \mathsf{hashMsg}(pkt))$
- and rt has sent out a packet of the correct format with PVFS_i and PVFD_i updated.

Next we define sendsSafePvfS $i\ S\ D\ pkS\ pvfS\ pkt\ t$ to mean that whenever a thread i sends out a message that contains a valid PVFS nested inside a chained MAC, the invariant PVFINV holds. Predicate safeNetPvfS $S\ D\ pkS\ pvfS\ pkt\ u$ asserts that for all threads i, sendsSafePvfS $i\ S\ D\ pkS\ pvfS\ pkt\ t$ has held up to time t.

$\mathsf{sendsSafePvfS}\ i\ S\ D\ pkS\ pvfS\ pkt\ t =$
$\forall\ m\ pvfS'\ k\ pvfn,\ \mathsf{send}\ \ i\ m@t \rightarrow \mathsf{contain}\ m\ pvfn \rightarrow$
$\mathsf{subPvf}\ pvfS\ pvfn \rightarrow pvfS = \mathsf{macMsg}\ k\ pvfS' \rightarrow$
$(\exists\ path\ keys, \mathsf{goodPvfS}\ pvfS\ S\ D\ pkS\ path\ keys\ pkt) \rightarrow$
$\mathsf{PVFINV}\ k\ pkS\ pkt\ pvfS\ pvfS'\ S\ D\ t.$

$\mathsf{safeNetPvfS}\ S\ D\ pkS\ pvfS\ pkt\ u=$
$\forall\ i\ u', u' \leq u \rightarrow \mathsf{sendsSafePvfS}\ i\ S\ D\ pkS\ pvfS\ pkt\ u'.$

LEMMA 10 (SAFEPVFS). *For all $rt\ S\ D\ k\ pkS\ pkD\ pkS^{-1}$ path,* session $S\ D\ path\ pkS\ pkD$, asymKeyPair $pkS\ pkS^{-1}$, honest rt, honest S, *and* flagSR $k\ S\ rt\ pkS$ *imply for all u,* safeNetPvfS $S\ D\ pkS$ (macMsg $k\ m$) $pkt\ u$.

The proof of Lemma 10 also uses rely-guarantee reasoning. This lemma provides us with the condition required for path authenticity at an honest router rt.

We have explained how to prove **MACProp** 1–3. From there, the origin and path authenticity properties of the OPT protocol can be proved as outlined in the beginning of this section.

6. DISCUSSION

The formal verification process has helped us identify subtle flaws in initial protocol designs. We show a few examples in Section 6.1. A practical implication of formally verified security guarantees is that the verified protocol can defend against certain classes of attacks. In Section 6.2, we elaborate on this connection.

6.1 Attacks on Alternative Designs

We manually construct attack traces based on failed proofs. We examine derivations leading up to the failed proofs, which provide hints for constructing attacks.

Router needs to sign the public key and directionality. In the key setup protocol, a router signs the triple: the symmetric key k, the hash of the public key used to generate k, and a direction: *downstream* or *upstream*. Removing either the public key or the direction will break the key secrecy and authenticity properties.

Consider the scenario where the router only signs the symmetric key k. Let pkS_1 be the public key that the source S sends in the packet and pkS_2 be an attacker's public key. The attacker changes the header field of the packet and inserts its own public key pkS_2. Let kS denote the symmetric key generated by the router using pkS_2. The router encrypts kS using pkS_2, generating ckS_2, and signs kS, generating $sigS_2$. The attacker can decrypt ckS_2, obtain kS, and encrypt kS using pkS_1, generating ckS_1. The attacker sends ckS_1 and the signature $sigS_2$ to the source. The source will accept this symmetric key. However, this key is not a secret shared only by the source and the router, violating the secrecy property;

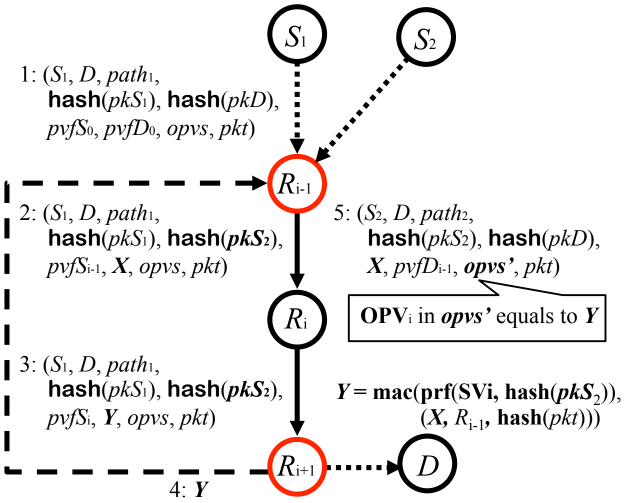
S_1
S_2
1: (S_1, D, $path_1$, **hash**(pkS_1), **hash**(pkD), $pvfS_0$, $pvfD_0$, $opvs$, pkt)
R_{i-1}
2: (S_1, D, $path_1$, **hash**(pkS_1), **hash**($\boldsymbol{pkS_2}$), $pvfS_{i-1}$, $\boldsymbol{X}$, $opvs$, pkt)
5: (S_2, D, $path_2$, **hash**(pkS_2), **hash**(pkD), $\boldsymbol{X}$, $pvfD_{i-1}$, $\boldsymbol{opvs'}$, pkt)
OPV$_i$ in $\boldsymbol{opvs'}$ equals to $\boldsymbol{Y}$
R_i
3: (S_1, D, $path_1$, **hash**(pkS_1), **hash**($\boldsymbol{pkS_2}$), $pvfS_i$, $\boldsymbol{Y}$, $opvs$, pkt)
$\boldsymbol{Y}$ = **mac(prf(SVi, hash(**$\boldsymbol{pkS_2}$**)), (**$\boldsymbol{X}$**,** R_{i-1}**, hash(**pkt**)))**
R_{i+1}
D
4: $\boldsymbol{Y}$

Figure 7: Attack scenario.

furthermore, the symmetric key is not computed using pkS_1, violating the authenticity property.

Next we show an attack to key authenticity if the router does not sign the *downstream* or *upstream* field. In this scenario, the attacker swaps the position of the session public keys of the source and destination. The source ends up accepting a key generated using the secret meant for the destination ($\mathsf{SV}_D(R_i)$). The consequence is that, during the forwarding phase, the router will drop legitimate packets, because the source will compute origin path validation fields using the wrong keys.

Routers need two secrets. Each router has two secrets, one for computing the key shared with the source and the other for computing the key shared with the destination. We illustrate in an example that, if we were to use one secret for both keys, origin and path authenticity can be violated. The high-level intuition is that with only one secret, an attacker can trick an honest router R_i to generate a bogus PVFD_i while R_i thinks it is computing a valid PVFS_i for a different path; and generate a bogus OPV_i while R_i thinks it is computing a valid PVFD_i for a different path. We show one scenario here. A second scenario is described in the companion technical report [49].

Attack scenario: PVFD is used to attack the source-based path validation of routers. The attack is illustrated in Figure 7. There are two different paths: $path_1$ is from S_1 to D and $path_2$ is from S_2 to D. Routers R_{i-1} and R_{i+1} are malicious and R_i is honest. We show how R_{i-1} and R_{i+1} collude to trick R_i into accepting a packet that R_i will believe to have originated from S_2 and traversed $path_2$, but was originated from S_1 and traversed $path_1$. S_1's session key is pkS_1 and S_2's session key is pkS_2.

A1: S_1 sends payload pkt down the path $path_1$.
A2: Malicious R_{i-1} on $path_1$ replaces the key hash for the destination with $\mathsf{hash}(pkS_2)$, places an arbitrary value $\mathbf{X}$ as PVFD_{i-1}, and sends the packet to R_i
A3: R_i verifies OPV_i, and computes PVFD_i, denoted $\mathbf{Y}$, and sends the packet to R_{i+1}. $\mathbf{Y}{=}\mathsf{mac}(\mathsf{prf}(\mathsf{SV}_\mathbf{i}, \mathsf{hash}(\boldsymbol{pkS_2})), (\mathbf{X}, \mathbf{R_{i-1}}, \mathsf{hash}(pkt)))$.
A4: R_{i+1} forwards $\mathbf{Y}$ to R_{i-1}
A5: R_{i-1} sends a packet with payload pkt, key hash of the source $\mathsf{hash}(pkS_2)$, $\mathbf{X}$ as PVFS_{i-1}, and $\mathbf{Y}$ as OPV_i to R_i.
A6: R_i validates OPV_i, and thinks that this packet was from S_2 on path $path_2$, which is not the case.

If R_i has two secrets, $\mathbf{Y}$ cannot pass as an OPV_i field.

6.2 Defending Against Attacks

Source and data spoofing. The source authenticity property of OPT (Theorem 7) ensures that a successful verification of the path validation field implies that there can be no source or data spoofing attacks to R_i, provided that the source is trusted.

Path deviation attack. The path authenticity property of OPT (Theorem 7) ensures that a successful verification by R_i (or the destination) implies that the packet R_i (or the destination) received has traversed all the honest nodes in the source-intended path in the correct order, assuming that the source is honest. Malicious routers cannot skip honest nodes, nor can they cause the packet to traverse the honest nodes in a different order than specified by the source. This indicates that if a malicious router selects a path not intended by the source, an honest intermediary router will reject the packet. However, a malicious node can send the packet to other routers that are not on the intended path.

Collusion. The path and source validation are conditioned upon whether a router is honest, i.e., correctly runs the protocol. It is clear from this property—and can be proven as a corollary—that if all the preceding routers are honest, then upon validating OPV_i, a router R_i knows that the packet originates from the source and all links in the intended path before R_i are traversed in the right order. Further, no other routers have received the packet if the links are secure.

When there are multiple adjacent malicious nodes on the intended path (R_{j1} to R_{jn}), a wormhole is present: an honest node down the path can only conclude that the packet has entered the hole via R_{j1} and exited the hole from R_{jn}, but has no knowledge of which nodes were traversed between these two points.

7. RELATED WORK

Secure routing protocols. Proposed secure routing infrastructures range from security extensions to BGP (Secure-BGP (S-BGP) [28], ps-BGP [46], so-BGP [47]), to more recent "clean-slate" Internet architectural redesigns such as SCION [50], ICING [37], and OPT [29]. ICING and OPT focus on packet forwarding, while the rest of the protocols establish routes among network nodes. ICING and OPT's goal is to provide source authentication and path validation to the routers in the network. Both protocols use the chained MAC of the packet content to attest to the authenticity of the packet. OPT is much lighter-weight than ICING in that each router needs to set up just two keys, one with the source and one with the destination, whereas ICING routers need to set up keys between each pair of routers on the path. The proof techniques presented here can be straightforwardly applied to analyzing ICING's origin and path authenticity properties.

Automated protocol verification tools. Numerous model checking tools [24, 12, 8, 32, 2, 13, 34, 33, 7, 45, 15, 19, 5, 35, 21, 42, 14, 26, 44] have been successfully applied to analyzing security protocols. However, these tools are rarely used in analyzing network protocols because such protocols are considerably more complicated than cryptographic protocols: they often compute local state, they are recursive, and their security properties need to be shown to hold for arbitrary network topologies. As the number of possible topologies is infinite, the number of models is also infinite, model-checking-based tools, in general, cannot be directly used to prove the security properties of networking protocols.

Proof-based techniques for reasoning about protocols. Our verification technique is built on prior work on reasoning about trace properties of systems [22, 23, 25]. The proof of the secrecy property of keys follows the same strategy as the secrecy proofs of Kerberos [41, 25]. The proofs of origin and path authenticity properties use a variant of the rely-guarantee reasoning principles [25]. The challenging part of the proofs is to identify the invariant properties associated with each MAC in the protocol. Identifying and verifying these invariants is one of main our technical contributions.

Various techniques have been applied to the verification of recursive cryptographic protocols [40, 6, 30]. All of them require abstract representation of the protocol behavior. In contrast, we directly verify local properties about the protocol in the same framework using LS^2. Closest to our work is work by Paulson, where recursive security protocols are verified by encoding the protocol and the Dolev-Yao adversary model in Isabelle/HOL [40]. The encoding of messages and the attacker in Paulson's work is similar to ours. In Paulson's encoding, a protocol is summarized as trace extensions allowed by the protocols. The correctness of such assertions is not verified in Paulson's work. We can verify such assertions using LS^2 inference rules. Because we encode the full protocol in Coq, we do not need to specify valid trace extensions. Instead, we specify trace properties that are relevant to the proofs and verify them using a state monad. Finally, our proofs use rely-guarantee principles which were not needed in the case studies that Paulson examined.

Another direction in verifying protocols is to use type systems built directly or indirectly on the work of Abadi [1] and Gordon and Jeffrey [27]. The most recent such systems are RCF [10] and its extensions [11, 43]. RCF is based on refinement types. RCF's theory has been implemented for the language F# in the refinement typechecker F7, backed by the SMT solver Z3 for discharging logical assertions. It has been used to automatically verify security properties of thousands of lines of code. Verified properties include weak secrecy properties and correspondence assertions [48]. Our proofs of authenticity properties require induction over the length of the trace. F7 would need to incorporate inductive principles to verify such properties.

Verification of network protocols. Recently, several papers investigated the verification of route authenticity properties of specific wireless routing protocols for mobile networks [3, 4, 20]. They have shown that identifying attacks on route authenticity can be reduced to constraint solving, and that the security analysis of a specific route authenticity property that depends on the topologies of network instances can be reduced to checking these properties on several four-node topologies. Their techniques are tightly tied to the protocol that they verify, and therefore, cannot be directly applied to other networking protocols, including ours. Chen et al. investigated verifying security properties of secure extensions of BGP [17]. They verified route authenticity properties on variants of S-BGP using a combination of manual proofs and Proverif [13]. This technique is specific to S-BGP and cannot be applied to our setting. Their subsequent work proposes a general framework that leverages a declarative programming language for verification and empirical evaluation of routing protocols [18]. Their program logic for the declarative language could be used in place of LS^2. However, constructing the full proof would still require augmenting their framework to include domain-specific definitions such as messages and the attacker's knowledge, in a similar approach to ours.

8. CONCLUSION

We have mechanized proofs of recursive secrecy and authenticity properties of the OPT source authentication and path validation protocols. These properties hold for all network topologies. By using LS^2, a program logic for reasoning about programs that run in adversarial environments, we are able to make minimal assumptions about the protocol code and directly verify the invariant properties of the pseudo code in Coq. We believe our Coq encoding is general enough to be used in verifying other protocols.

9. ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable feedback. This research was supported in part by CyLab at Carnegie Mellon, NSF grants CNS-1040801, CNS-1115706, CNS-1018061, the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement 617605, and a gift from KDDI.

10. REFERENCES

[1] M. Abadi. Secrecy by typing in security protocols. *Journal of the ACM*, 46(5):749–786, 1999.

[2] A. Armando and L. Compagna. SATMC: A SAT-Based Model Checker for Security Protocols. In *JELIA*, 2004.

[3] M. Arnaud, V. Cortier, and S. Delaune. Modeling and Verifying Ad Hoc Routing Protocols. In *Proc. CSF*, 2010.

[4] M. Arnaud, V. Cortier, and S. Delaune. Deciding security for protocols with recursive tests. In *Proc. CADE*, 2011.

[5] D. A. Basin. Lazy Infinite-State Analysis of Security Protocols. In *CQRE*, 1999.

[6] D. A. Basin, S. Capkun, P. Schaller, and B. Schmidt. Formal Reasoning about Physical Properties of Security Protocols. *ACM Trans. Inf. Syst. Secur. 14(2):16*, 2011.

[7] D. A. Basin, S. MÃűdersheim, and L. ViganÃš. Ofmc: A symbolic model checker for security protocols. *Int. J. Inf. Sec.*, 4(3):181–208, 2005.

[8] J. Bau and J. Mitchell. A Security Evaluation of DNSSEC with NSEC3. In *Proc. NDSS*, 2010.

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

[10] J. Bengtson, K. Bhargavan, C. Fournet, A. D. Gordon, and S. Maffeis. Refinement types for secure implementations. *TOPLAS*, 33(2):8:1–8:45, 2011.

[11] K. Bhargavan, C. Fournet, and A. D. Gordon. Modular Verification of Security Protocol Code by Typing. In *Proc. POPL*, 2010.

[12] B. Blanchet. Automatic verification of correspondences for security protocols. *J. Comput. Secur.*, 17(4), Dec. 2009.

[13] B. Blanchet and B. Smyth. ProVerif 1.86: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial. `http://www.proverif.ens.fr/manual.pdf`.

[14] Y. Boichut, P.-C. Heam, O. Kouchnarenko, and F. Oehl. Improvements on the Genet and Klay Technique to

Automatically Verify Security Protocols. In *Proc. AVIS*, 2004.

[15] L. Bozga, Y. Lakhnech, and M. PÃl'rin. HERMES: An Automatic Tool for Verification of Secrecy in Security Protocols. In *CAV*, 2003.

[16] K. Butler, T. R. Farley, P. McDaniel, and J. Rexford. A Survey of BGP Security Issues and Solutions. *Proc. the IEEE*, 98:100–122, January 2010.

[17] C. Chen, L. Jia, B. T. Loo, and W. Zhou. Reduction-based Security Analysis of Internet Routing Protocols. In *WRiPE*, 2012.

[18] C. Chen, L. Jia, H. Xu, C. Luo, W. Zhou, and B. T. Loo. A Program Logic for Verifying Secure Routing Protocols. In *Proc. FORTE*, 2014.

[19] E. M. Clarke, S. Jha, and W. Marrero. Verifying Security Protocols with Brutus. *ACM Trans. Softw. Eng. Methodol.*, 9:443–487, 2000.

[20] V. Cortier, J. Degrieck, and S. Delaune. Analysing routing protocols: four nodes topologies are sufficient. In *Proc. POST*, 2012.

[21] C. J. Cremers. The Scyther Tool: Verification, Falsification, and Analysis of Security Protocols. In *Proc. CAV*, 2008.

[22] A. Datta, A. Derek, J. C. Mitchell, and A. Roy. Protocol Composition Logic (PCL). *Electronic Notes in Theoretical Computer Science*, 172:311–358, 2007.

[23] A. Datta, J. Franklin, D. Garg, and D. Kaynar. A Logic of Secure Systems and its Application to Trusted Computing. In *Proc. IEEE S&P*, 2009.

[24] S. Escobar, C. Meadows, and J. Meseguer. A rewriting-based inference system for the NRL protocol analyzer: grammar generation. In *Proc. FMSE*, 2005.

[25] D. Garg, J. Franklin, A. Datta, and D. Kaynar. Compositional System Security in the Presence of Interface-Confined Adversaries. *Electronic Notes in Theoretical Computer Science*, 265:49–71, 2010.

[26] T. Genet and F. Klay. Rewriting for Cryptographic Protocol Verification. In *Proc. CADE*, 2000.

[27] A. D. Gordon and A. Jeffrey. Authenticity by typing for security protocols. *Journal of Computer Security*, 11(4):451–519, July 2003.

[28] S. Kent, C. Lynn, J. Mikkelson, and K. Seo. Secure Border Gateway Protocol (S-BGP). *IEEE Journal on Selected Areas in Communications*, 18:103–116, 2000.

[29] T. H.-J. Kim, C. Basescu, L. Jia, S. B. Lee, Y.-C. Hu, and A. Perrig. Lightweight Source Authentication and Path Validation. In *Proc. of ACM SIGCOMM*, 2014.

[30] R. Küsters and T. Wilke. Automata-Based Analysis of Recursive Cryptographic Protocols. In *Proc. STACS*, 2004.

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

[32] G. Lowe. An Attack on the Needham-Schroeder Public-key Authentication Protocol. *Inf. Process. Lett.*, 56(3):131–133, 1995.

[33] C. Meadows. The NRL Protocol Analyzer: An Overview. *J. Log. Program.*, 26:113–131, 1996.

[34] J. K. Millen, S. C. Clark, and S. B. Freedman. The Interrogator: Protocol Security Analysis. *IEEE Trans. Software Eng.*, 13:274–288, 1987.

[35] J. C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using mur-phi. In *Proc. IEEE S&P*, 1997.

[36] A. Nanevski, G. Morrisett, A. Shinnar, P. Govereau, and L. Birkedal. Ynot: Reasoning with the awkward squad. In *Proc. ICFP*, 2008.

[37] J. Naous, M. Walfish, A. Nicolosi, D. Mazieres, M. Miller, and A. Seehra. Verifying and enforcing network paths with ICING. In *Proc. CoNEXT*, 2011.

[38] O. Nordström and C. Dovrolis. Beware of BGP attacks. *SIGCOMM Computer Communication Review*, 34:1–8, 2004.

[39] B. Parno, A. Perrig, and D. Andersen. SNAPP: Stateless Network-Authenticated Path Pinning. In *Proc. ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, 2008.

[40] L. C. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. *J. Comput. Secur.*, 6(1-2):85–128, Jan. 1998.

[41] A. Roy, A. Datta, A. Derek, J. C. Mitchell, and J.-P. Seifert. Secrecy Analysis in Protocol Composition Logic. In *Proc. ASIAN*, 2006.

[42] D. X. Song, S. Berezin, and A. Perrig. Athena: A Novel Approach to Efficient Automatic Security Protocol Analysis. *J. Comput. Secur.*, 9:47–74, 2001.

[43] N. Swamy, J. Chen, C. Fournet, P.-Y. Strub, K. Bhargavan, and J. Yang. Secure Distributed Programming with Value-Dependent Types. In *Proc. ICFP*, 2011.

[44] M. Turuani. The CL-Atse Protocol Analyser. In *Proc. RTA*, 2006.

[45] L. Viganò. Automated Security Protocol Analysis With the AVISPA Tool. *Electron. Notes Theor. Comput. Sci.*, 155:61–86, 2006.

[46] T. Wan, E. Kranakis, and P. C. Oorschot. Pretty secure BGP (psBGP). In *Proc. NDSS*, 2005.

[47] R. White. Securing BGP Through Secure Origin BGP (soBGP). *The Internet Protocol Journal*, 6(3):15–22, 2003.

[48] T. Y. C. Woo and S. S. Lam. A Semantic Model for Authentication Protocols. In *Proc. IEEE S&P*, 1993.

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

[50] 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 *Proc. IEEE S&P*, 2011.