Securing ARP/NDP From the Ground Up

Dave (Jing) Tian, Kevin R. B. Butler, *Member, IEEE*, Joseph I. Choi,
Patrick McDaniel, *Fellow, IEEE*, and Padma Krishnaswamy

***Abstract*— The basis for all IPv4 network communication is the address resolution protocol (ARP), which maps an IP address to a device's media access control identifier. ARP has long been recognized as vulnerable to spoofing and other attacks, and past proposals to secure the protocol have often involved in modifying the basic protocol. Similarly, neighbor discovery protocol (NDP) is the basis for all IPv6 network communication, yet suffers from the same vulnerabilities as ARP. This paper introduces *arpsec*, a secure ARP/RARP protocol suite which a) does not require protocol modification, b) enables continual verification of the identity of the target (respondent) machine by introducing an address binding repository derived using a formal logic that bases additions to a host's ARP cache on a set of operational rules and properties, c) utilizes the trusted platform module (TPM), a commodity component now present in the vast majority of modern computers, to augment the logic-prover-derived assurance when needed, with TPM-facilitated attestations of system state achieved at viably low-processing cost, and d) supports IPv6 NDP (*ndpsec*) by extension of our previous work. Using commodity TPMs as our attestation base, we show that *arpsec* incurs an overhead ranging from 7% to 15.4% over the standard Linux ARP implementation, a comparable overhead against the standard Linux NDP implementation, and provides a first step towards a formally secure and trustworthy networking stack for both IPv4 and IPv6.**

***Index Terms*— Access protocols, logic programming, message authentication, network security.**

I. Introduction

THE ADDRESS Resolution Protocol (ARP) [36] is a fundamental part of IPv4 network connectivity. Operating below the network layer, ARP binds an IP address to the Media Access Control (MAC) identifier of a network device, e.g., an Ethernet card or a Wi-Fi adapter, which in turn completes the process of routing the packet to its intended destination. Such communication relies on the last hop for correct delivery. ARP is subject to a variety of attacks including spoofing and cache poisoning, as originally described by Bellovin [5]. Tools such as *dsniff* [42] and *nemesis* [28] can be used respectively to easily launch such attacks. An attack on ARP can subsequently enable more sophisticated denial-of-service (DoS) and man-in-the-middle (MitM) [30] attacks.

While numerous methods have been proposed to secure ARP [21], [31], [33], [46], [51], they fall short of offering a comprehensive solution to these problems. First, a successful security solution must ensure that the basic ARP protocol itself remains unchanged. There is no "flag day" on which all ARP implementations embedded into the large variety of Internet-connected IPv4 devices will change. Second, the overhead of the implementation should be as small as possible in order to optimize system performance. Third, the ARP security mechanism should be flexible and reliable. Hard-coded security policies may not be applicable to varying network environments. Last, we need to know if the remote machine can be trusted. Trust here applies to both the authentication and the system integrity state of the remote machine, e.g., even if a binding is correct, we may not wish to add a remote host that cannot attest to the correctness of its operation. While past proposals have ranged from localized solutions to those involving public key infrastructures [6], [15], [23], they have not been widely deployed, either due to requiring specific network configurations, creating large system overheads, or requiring fundamental changes to ARP.

In this paper, we propose *arpsec*, an ARP security approach based on logic and the use of the Trusted Platform Module (TPM) [48], to implement security guarantees. *arpsec* does not change or extend the ARP itself. Instead of hard-coded security policies, *arpsec* formalizes the ARP system binding using logic. A logic prover then reasons about the validity of an ARP reply from the remote machine based on the codified logic rules and the previously stored binding history on the local system. A TPM attestation protocol is also implemented to challenge the remote machine if the logic layer fails to determine the trustworthiness of the remote machine. Using TPM hardware, we can authenticate (establish the identity) of the remote and discover whether the remote machine is in a good integrity state (i.e., not compromised). *arpsec* defends from most categories of ARP attacks by tethering address bindings to trusted hardware, establishing the basis for a trustworthy networking stack.

We have implemented *arpsec* in the Linux 3.2 kernel, using commodity TPMs and a Prolog engine. Our experiments show that *arpsec* only introduces a small system overhead, ranging from 7% to 15.4% compared to the original ARP and incurs the lowest overhead when compared to the two PKI-based ARP security proposals, S-ARP [6] and TARP [23].

Manuscript received October 29, 2016; revised February 28, 2017; accepted April 3, 2017. Date of publication April 19, 2017; date of current version June 20, 2017. This work was supported by the U.S. National Science Foundation under Grant CNS-1540217. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Qian Wang. *(Corresponding author: Kevin R. B. Butler.)*

D. Tian, K. R. B. Butler, and J. I. Choi are with the Department of Computer Science and Engineering, University of Florida, Gainesville, FL 32601 USA (e-mail: daveti@ufl.edu; butler@ufl.edu; choijoseph007@ufl.edu).

P. McDaniel is with the Institute for Network and Security Research, School of Electrical Engineering and Computer Science, The Pennsylvania State University, State College, PA 16802 USA (e-mail: mcdaniel@cse.psu.edu).

P. Krishnaswamy is with the Federal Communications Commission, Washington, DC 20554 USA (e-mail: pkrishnaswamy2017@gmail.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIFS.2017.2695983

We have then expanded on this previous work, which appeared in CODASPY'15 [45], by applying the idea of *arpsec* to support the Neighbor Discovery Protocol (NDP). ARP is replaced by the Neighbor Discovery Protocol (NDP) [27] in IPv6. NDP, like ARP is used to establish bindings between a device's IP address and link-layer, or MAC, address. NDP also incorporates the ICMP-based functionalities of router discovery and redirect. Unfortunately, NDP shares many of the vulnerabilities of its predecessor, being vulnerable to three main classes of attack: Denial-of-Service (DoS), address spoofing, and router spoofing [27]. Our new work, *ndpsec*, builds on our previous framework for *arpsec* with the goal of providing a more comprehensive solution to host protection, including in the IPv6 setting. Our evaluation shows that *ndpsec* outperforms SEND [3], [7], an alternative solution that relies heavily on cryptography, by one order of magnitude.

In summary, our contributions include:

- Design and implementation of *arpsec* to secure ARP by leveraging TPM hardware and logic reasonings.
- Comparison of *arpsec* with two other ARP security solutions, S-ARP and TARP.
- Design and implementation of *ndpsec* to secure NDP, by extending *arpsec* into IPv6.
- Comparison of *ndpsec* with the existing NDP security solution SEND.

To the best of our knowledge, our work is the first solution to consider building trusted networking for IPv4/IPv6 domains starting from ARP/NDP using TPM, and ours is the first to implement the defense algorithm using logic. This is the first step to build a trusted and verifiable networking environment.

The remainder of this paper is structured as follows. Section 2 outlines the background on ARP and NDP security issues as well as on trusted computing. Section 3 details the design and architecture of *arpsec* and *ndpsec*. Section 4 shows details and tradeoffs during the implementation. Section 5 describes a general procedure to deploy *arpsec* and *ndpsec* in the wild. Section 6 provides the performance evaluation of *arpsec* and *ndpsec*. Section 7 discusses potential issues with *arpsec* and *ndpsec* alongside possible solutions. Section 8 reviews the past efforts on ARP and NDP security, and Section 9 concludes.

II. Background

We first discuss ARP and NDP security issues based on their current design and implementations, before explaining common attacks against either protocol. As both *arpsec* and its extension, *ndpsec*, use the TPM, a brief review of trusted computing is provided.

A. ARP Security Issues

ARP [36] is the glue between Layer 3 and Layer 2 in IPv4 networks, binding IP addresses to medium access control (MAC) addresses unique to a particular network interface card (NIC).[1] Before an IP packet is sent out from a NIC (e.g., an Ethernet card), the host's ARP cache is queried to find the MAC address assigned to the target IP address of the packet. If the MAC/IP binding is not found, an ARP *request* will be broadcast to the entire network segment (the *broadcast domain*). Only the host with the target IP address should send back an ARP *reply* containing its MAC address.

[1]Reverse ARP [11] is generally obsolete in favor of other bootstrapping protocols such as DHCP and BOOTP.

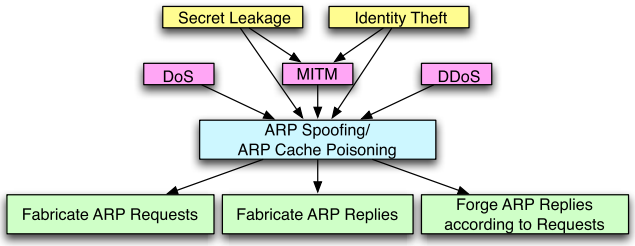


Fig. 1. An attack tree for ARP.

In reality, every machine in the network could send an ARP reply claiming that it has the requested MAC address, as there is no ARP reply authentication mechanism. In this case, most operating systems either accept the first reply or the latest one if multiple replies respond to the same request. They further optimize performance by processing ARP requests from other machines and adding MAC/IP bindings for future use. Though all bindings in the ARP cache have some Time-To-Live (TTL) control, the timer is usually large and designed for performance rather than security. As an example, Linux always accepts the first ARP reply to the request and ignores others. It also rejects ARP replies without a request while processing ARP requests from other machines. The TTL for each entry in the Linux ARP cache is around 20 minutes [6]. Solaris and Windows have similar optimizations and hence, similar security issues [26], [43].

One basic attack against ARP is message spoofing. The adversary could inject a new MAC/IP binding into the victim's ARP cache simply by sending a forged ARP request or reply to the victim. The other basic ARP attack is cache poisoning [51], where the adversary generates the ARP reply using a certain MAC address given the request from the victim. Both spoofing and poisoning attacks attempt to insert a malicious MAC/IP binding in the victim's ARP cache.

As shown in Figure 1, the attacks described above act as enablers for other adversary actions, such as man-in-the-middle (MitM) attacks [30] and denial of service (DoS) [51] attacks. For a DoS attack, the adversary can inject the victim's MAC address into a particular machine or substitute the victim's MAC address with another one. In the former case, all the IP traffic from that machine targeting a certain address will be redirected to the victim, while in the latter case, the victim would never receive the messages intended for it to provide the service. MitM attacks are particularly serious, since with the help of ARP spoofing/poisoning, the adversary can interpolate himself into the traffic between victims by injecting his MAC/IP binding into both victims' ARP cache. Both attacks are also quite simple to implement, with small usable scripts widely available, and these in turn can lead to attacks compromising user identity or allowing the leakage of secret information.

B. NDP Security Issues

NDP [27] is the replacement to ARP in IPv6 networks. For a particular link, NDP is involved in the following [27]: router discovery by host devices, prefix discovery for enumerating on-link destinations, discovery of either link (e.g., MTU) or Internet (e.g., hop limit) parameters, stateless address auto-configuration, address resolution (as with ARP, determine the link layer address of an on-link destination given its IP addresses), next-hop determination, neighbor unreachability detection (NUD), duplicate address detection (DAD), and redirect. The above functionality is provided by five types of ICMP packets. The corresponding message types are [27]:

1) Router Solicitation (RS): sent by hosts, requesting routers to generate Router Advertisement messages.
2) Router Advertisement (RA): periodically sent by routers, making their presence and relevant parameters known.
3) Neighbor Solicitation (NS): sent to determine a neighbor's link-layer address or confirm reachability.
4) Neighbor Advertisement (NA): sent as a response to a NS or as an announcement of link-layer address change.
5) Redirect: sent by routers, identifying a better first hop.

NDP does ignore packets from off-link senders and check Hop Limit fields, but these measures alone are insufficient [27].

Despite being a more robust counterpart for ARP in IPv6 networks, NDP remains vulnerable to three main classes of attack: Denial-of-Service (DoS), address spoofing, and router spoofing [27]. NDP's vulnerability to such attacks results from there being no means to verify whether a particular node is authorized to send a particular type of message. This means, for example, any non-router node may launch a DoS attack by falsely advertising itself as a router. Spoofed router solicitation or advertisement messages can also allow man-in-the-middle attacks, enable parameter and prefix spoofing, or seemingly kill router(s), thus invoking direct Neighbor Discovery exchanges between nodes on the link [29]. Similar effects may be achieved via spoofed redirect messages. Spoofed neighbor solicitation or advertisement messages can cause packet redirection, block Neighbor Unreachability Detection, and leverage Duplicate Address Detection to deny access to newly entering hosts [29], introducing similar security problems as in ARP spoofing/poisoning.

C. Trusted Platform Module (TPM)

A Trusted Platform Module (TPM) is a cryptographic chip embedded in motherboards. Though implemented by various vendors, all TPM chips follow the TPM specification [48] designed by the Trusted Computing Group (TCG). In conjunction with the system BIOS, TPMs can be used to form a root of trust in a system and to build the trust chain for the software along the software stack, including boot loaders, operating systems, and applications [16], [17], [32], [38].

TPMs can help to determine the true identity of a remote host via the Attestation Identity Key (AIK) verification during the TPM attestation. After creating an AIK pair, the TPM hardware communicates with a Privacy Certification Authority (PCA) or Attestation Certification Authority (ACA) using the information embedded in itself to prove its identity and get the AIK credentials. A remote machine proves its integrity state by reporting the values of its Platform Configuration Registers (PCRs). If the *measurement* of PCR values during a TPM attestation is different from what is expected, the remote may be compromised and thus not be trustworthy. It is important to realize that the AIK private key and the measurement of PCRs are all stored in the TPM itself. Unless the TPM hardware is compromised [44], there is no disclosed method of hacking into the TPM through software and changing PCR values.

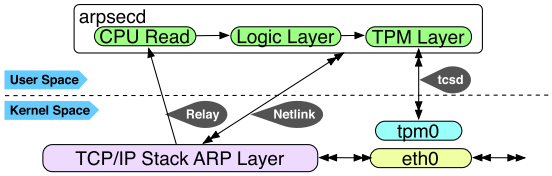


Fig. 2. The architecture of *arpsec*.

III. DESIGN

A. Threat Model

The hardware, BIOS, boot loader, operating system and the corresponding system libraries, as well as the *arpsecd* daemon, are trusted components in our local host. However, except for the TPM hardware in the remote machine, we do not trust anything generated by the remote machine. Moreover, the adversary may have compromised the remote machine and gained root permission, through which any ARP or NDP attacks can be launched, including ARP message spoofing and ARP cache poisoning. The adversary may also leak secret information he has gotten from the victim and use this information to impersonate the victim on another machine while taking the victim machine offline. In short, for the local *arpsec* host machine, the TCB includes all hardware and system software required to start *arpsecd*; the local host machine should also only trust the TPM hardware within the remote machine during the ARP or NDP processing.

It is important to note that the TPM hardware attacks, like TPM deconstructing [44] and TPM reset attack [17], are not considered in this paper, nor are potential TPM relay attacks, whose preferred solution is a special-purpose hardware interface [32]. Also, *arpsec* is not designed for DoS/DDoS attacks, though it has the ability to defend against simple DoS attacks, which we discuss further in Section VII.

B. System Design

Compared to proposals such as S-ARP and TARP, which take advantage of the PKI system to extend ARP, *arpsec* formalizes ARP address binding and validates ARP messages using both a logic prover and TPM attestations, all without requiring any changes to the original protocol. The architecture of *arpsec*, shared by *ndpsec*, is illustrated in Figure 2.

In the user space, *arpsecd* is the daemon process of *arpsec* that runs in the local machine and takes control of processing of all the ARP messages from the kernel. There are three major components in the *arpsecd* daemon: the CPU read, the logic layer and the TPM layer. The CPU read component

Algorithm 1 ARP Message Processing Within *arpsec*

```
while there is an ARP msg from the kernel do
    check the msg type;
    if msg.type == ARP request then
        if msg is for us then
            reply the request;
        else
            drop the request;
        end
    else if msg.type == ARP reply then
        if msg is for us then
            if msg passes the logic layer then
                add the MAC/IP binding into the ARP cache;
            else
                if msg passes the TPM layer then
                    add the MAC/IP binding into the ARP
                    cache;
                else
                    drop the reply;
                end
            end
        else
            if msg passes the logic layer then
                add the MAC/IP binding into the ARP cache;
            else
                drop the reply;
            end
        end
    end
end
```

retrieves all ARP request/reply messages from the kernel space and passes the preprocessed, logic-friendly messages to the logic layer component. The logic layer component then tries to handle these messages based on the message type, system state and the logic rules. We will detail the logic layer in the following section. For the ARP reply, if the logic layer is unable to validate the message, the TPM layer will then challenge the remote machine using the TPM attestation. Only the MAC/IP bindings (in the ARP reply) validated by the logic layer or TPM attestation could be added into the local ARP cache. The pseudo code of *arpsecd* ARP processing is listed in Algorithm 1. Note that *arpsec* ignores ARP requests not for itself. These are usually processed by ARP implementations for performance but leave the ARP cache vulnerable.

C. Logic Formulation

The logic layer in *arpsec* is the first filter used to testify the trustiness of an ARP/RARP reply message. The logic layer imposes minimal performance costs when compared to the TPM layer, using a logic prover and a list of ARP logic rules. To leverage the power of the logic reasoning, firstly, we introduce an ARP system binding logic formulation.

1) Intuition: The logic layer tracks statements (attestations) by systems that particular media addresses are mapped to network addresses. The timing of these statements are tracked such that the logic can "prove" exactly which binding is the most authoritative at a given time. The logic judges a binding to be authoritative if it is the most recent one received from a trusted system. At runtime, the system generates, if possible, the proof of a binding before using it for network communication.

An instance of an ARP binding system is defined as $\mathcal{A} = \{\mathcal{N}, \mathcal{M}, \mathcal{T}, \mathcal{S}, \bar{\mathcal{S}}, \bar{\mathcal{R}}\}$, where

$$\begin{aligned}\mathcal{T} &= \mathbb{P}\\ \mathcal{N} &= (\epsilon, n_0, \ldots, n_a)\\ \mathcal{M} &= (\epsilon, m_0, \ldots, m_b)\\ \mathcal{S} &= (s_0, \ldots, s_c)\\ \bar{\mathcal{S}} &= \mathcal{S} \times \mathcal{T}\\ \bar{\mathcal{R}} &= \mathcal{S} \times \mathcal{N} \times \mathcal{M} \times \mathcal{T}\end{aligned}$$

Intuitively, $\mathcal{T}$ is a set of all positive integers representing an infinite and totally ordered set of time epochs. $\mathcal{N}$ is the collection of network addresses and $\mathcal{M}$ is the collection of media addresses. For convenience, both address sets contain a special address ϵ representing the lack of binding assignment, described below. $\mathcal{S}$ is the set of systems that makes assertions about the address bindings within the network. $\bar{\mathcal{S}}$ represents the timing of system trust validations (e.g., system attestations); $\bar{s}_{i,j} \in \bar{\mathcal{S}}$ where system s_i was successfully vetted at time t_j. $\bar{\mathcal{R}}$ is the binding assertions made in the course of operation of the ARP protocol, where $\bar{R}_{i,j,k,l} \in \bar{\mathcal{R}}$ if system s_i asserts the binding (n_j, m_k) at time t_l. Lastly, for ease of exposition, we introduce the following derived binding and trust time-state elements within the system:

$$\begin{aligned}\mathcal{A} &= (A_0, \ldots, A_{|\mathbb{P}|})\\ \mathcal{B} &= (B_0, \ldots, B_{|\mathbb{P}|})\end{aligned}$$

$\mathcal{R}$ is the key conceptual element here; each element of $\mathcal{R}$ captures the fact that system s_i stated (through an attestation) a binding of network address n_j to media address m_k at time t_l. The remainder of the logic simply reasons from the set of statements which binding should be considered authoritative at a given time.

2) Trust State: The trust state $\mathcal{A}$ of the system is a totally ordered set of subsets of $\mathcal{S}$ representing the instantaneous set of systems that have been determined to be in trusted state in each epoch (e.g., have been vetted through system attestations). The trust state of the A at time t_k, A_k is:

$$A_k^h = \{s \mid \exists j, (k-h) \leq j < k : (s, j) \in \bar{\mathcal{S}}\}$$

Or simply, A_k is the set of all systems $s_i \in \mathcal{S}$ that have been vetted as trustworthy within the last h epochs. The security parameter h represents the durability of a system trust state. In the initial state of the system all systems are untrusted, e.g., $A_o = \{\emptyset\}$.

3) Binding State: We refer to the B_k as the binding state at time $\mathcal{T}_k$. The states of the binding system $\mathcal{B}$ are a totally ordered sequence of B_k, which is a relation over $\mathcal{N}$ and $\mathcal{M}$ representing the instantaneous binding of network to media

addresses, where:

$$\forall B_k \in \mathcal{B} : B_k \subset \mathcal{N} \times \mathcal{M}$$

It is worth noting further that each B_k is constrained by a set of *coherency* properties that define correct operation of the binding protocol. Namely, $\forall B_k \in \mathcal{B}$:

$$\begin{aligned}
&(1) \quad \forall\ n_l \in \mathcal{N} : \exists\ (n_l, m_o), m_o \in \mathcal{M} \\
&(2) \quad \forall\ m_o \in \mathcal{M} : \exists\ (n_l, m_o), n_l \in \mathcal{N} \\
&(3) \quad \nexists\ (n_l, m_o), (n_p, m_q) : n_l = n_p \neq \epsilon \\
&(4) \quad \nexists\ (n_l, m_o), (n_p, m_q) : m_o = m_q \neq \epsilon
\end{aligned}$$

That is, all network addresses (constraint 1) and media addresses (2) must have an assignment at each epoch. Further, the network address not bound to the unassigned element ϵ must be bound to exactly one media address (3), and the media address not bound to the unassigned element ϵ must be bound to exactly one network address (4).

We define the set of rules with operational properties for the binding set. We state that $(n_j, m_k) \in B_l$ if and only if:

$$\begin{aligned}
(5) \quad &\exists\ \bar{R}_{i,j,k,x} \in \bar{\mathcal{R}}, \quad x \leq l, \quad s_i \in A_x, \\
&\nexists\ \bar{R}_{v,j,p,y} \in \bar{\mathcal{R}}, \quad p \neq k, \quad y > x, \ s_v \in A_y, \\
&\nexists\ \bar{R}_{v,q,k,y} \in \bar{\mathcal{R}}, \quad j \neq q, \quad y > x, \ s_v \in A_y
\end{aligned}$$

Constraint (5) indicates that any binding in B_l was asserted at or prior to time t_l by a trusted system, and no later assertion for that network or media address was subsequently received at or before t_l was asserted.

Finally, by definition, all network and media addresses are unassigned in the initial state B_0:

$$B_0 = \forall n_l \in \mathcal{N}, (n_l, \epsilon) \bigcup \forall m_o \in \mathcal{M}, (\epsilon, m_o)$$

In general, constraint (5) is the core property used by the logic prover to implement the ARP security. The logic layer stores all the verified bindings with the remote system identifiers and the time epochs. For any given MAC/IP binding in the ARP/RARP reply message from the remote, if there exists a binding record from the same (trusted) remote in the past that is no older than a pre-defined number of epochs (security parameter h) when compared to the current epoch. the logic layer would trust this binding, add the binding to the local ARP cache, and add this binding record into the logic prover for future reasoning. Security parameter h represents a tradeoff between reliability and performance, as it determines the time range of the past we trust to validate the current event.

The Prolog engine continuously consumes assertions received at an end host and infers B_k at each time epoch using the above constraints. That generation thus provides a proof of authority; if a binding $(n_i, m_j) \in B_k$, then it is authoritative and can be used for communication at time t_i.

In summary, the security parameter h dominates how long we trust the remote machine given the current MAC/IP binding, which is attested and trusted, by essentially setting a timer. When the binding is updated (e.g., either by the host machine solicitation or the remote machine), the timer gets extended as long as the binding does not change and the timer has not expired yet. This design introduces the minimum runtime overhead given a benign environment, as the attestation happens only once at the very start of the communication. When the timer expires, or the binding changes, a new attestation will be triggered before this binding is added into the local cache, making sure that either the remote machine is still in a good state or the remote machine is the one owning this binding. The logic layer in *ndpsec* is extended from that of *arpsec* and testifies to the trustiness of NDP messages.

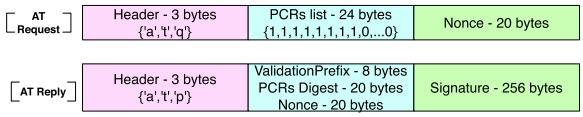


Fig. 3. The AT Request/Reply.

D. TPM Attestation

If an incoming ARP/RARP request or reply, or NDP message, cannot be validated by the logic layer, *arpsec/ndpsec* turns to a TPM layer as a second line of defense. To establish trust in the remote host, we use a TPM attestation [48], whose general operation is described in Section II. A *measurement* is taken based on the current state of the underlying hardware, BIOS, boot loader, and operating system, with each value stored in a PCR. The TPM is tamper-resistant and access to PCRs is not possible except through expensive hardware attacks. The Attestation Identification Key (AIK) thus provides identity while the PCRs determine system integrity state.

We design the *arpsec/ndpsec* Attestation (AT) protocol for communication between the local machine (also known as the challenger) and the remote machine (also known as the attester), as shown in Figure 3. The request contains a header, a list of PCRs, and a nonce to prevent replay. The PCR list indicates which registers are of interest - for our purposes, these are registers 0 through 7.[2] When the host receives this challenge, it responds with a TPM *Quote*, which includes the nonce, PCR values and their corresponding digest, signed by the AIK private key, in the AT reply. If at this point any of these values fail and the signature cannot be validated, the address binding is purged from the ARP or NDP neighbor cache. Note that we do not put any MAC/IP binding into the AT reply. Comparing to the TPM, a MAC/IP binding is easy to fake and thus not trustworthy.

IV. Implementation

We have implemented *arpsec* in Linux with the 3.2.0.55 kernel, using C and Prolog. The implementation details of the *arpsecd* daemon is shown in Figure 4. The primary goal of the implementation is high performance, in order to minimize the overhead of *arpsec*. We also focus on *incremental deployment*, such that *arpsec* and standard ARP messages can coexist in the same network.

Depicted in Figure 4, the *relay* mechanism [52] transports ARP messages from kernel to user space, as it is designed

[2]PCRs 0 through 7 cover the measurement for hardware, BIOS, boot loader and even OS [48] Other PCRs could be extended to cover system libraries and even applications.

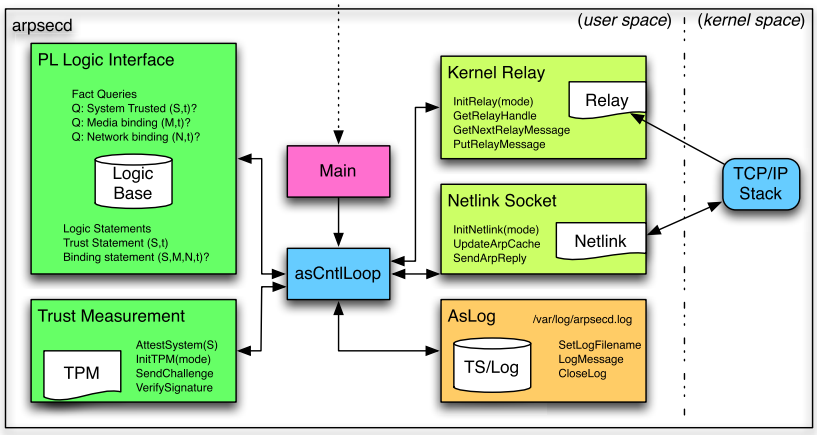


Fig. 4. The implementation of *arpsec* daemon (*arpsecd*).

to manage large amounts of asymmetric traffic. We also use a netlink socket to communicate from user to kernel space, in order to manipulate the ARP cache. This provides similar functionality to the `ioctl()` calls for cache management but uses the low-level kernel APIs to get rid of the extra locking in `ioctl()`. Using this netlink socket, we could also trigger the kernel to send the ARP reply given any request, at which point it is relayed to user space for efficient processing.

In user space, the logic formulation of the ARP binding system is implemented in GNU Prolog (GProlog) [9]. We integrate the GProlog-based logic prover into our C-based *arpsecd* using the GProlog-C interfaces, providing a 50X performance improvement compared to IPC between *arpsecd* and the GProlog interpreter. We set a 5-second security parameter, meaning that every 5 seconds we expect a new attestation of the ARP binding.

We also implemented a whitelist and two blacklists before logic processing occurs. The whitelist contains the MAC/IP bindings known to be good under all conditions. The two blacklists contain potentially malicious MAC addresses or IP addresses, respectively. Currently, only the MAC/IP bindings which failed the TPM attestation will be added into the black list. All entries in the blacklists have the same TTL of 200 seconds, at which point they are removed.

The *arpsec* TPM component is built on the top of the Trousers API [47] following the TPM 1.2 specification [48]. TPM information (PCRs and AIK public keys) of remote hosts is stored in an internal database. A TPM daemon (*tpmd*) is also implemented for the remote machines to process the TPM attestation from *arpsecd*. The PrivacyCA architecture is also used to get the AIK credential for the TPM hardware. In our current implementation and testing environment, we are interested in the PCR `0-7`. More PCRs could be covered and extended if the measurement of applications is also desired.

Like *arpsec*, *ndpsec* has both kernel and user space components. Unlike IPv4, the Linux kernel has imposed some defensive checks in IPv6 before treating the NDP message to be valid, including multicast address checking, DAD checking, and ICMPv6 ND option checking. Once the NDP message is validated by the kernel, it is relayed to the user-space daemon *arpsecd*[3] before being processed by the kernel. The netlink socket used by *arpsec* has been extended to support NDP message delivery of NS and NA messages. Accordingly, we have extended *arpsecd* to support NDP message processing as well. Similarly, *tpmd* is augmented to respond to the IPv6-based TPM attestation. No changes are needed by the Prolog engine, which is transparent towards IPv4 or IPv6.

V. Deployment

To deploy *arpsec/ndpsec* in the wild, we need a TPM Information Management Server (TIMS) that acts in a similar manner to a Privacy CA as defined by the TCG. Below, we detail the management of AIK key pairs by the TIMS.

As one TPM could generate multiple AIK pairs, we require each new machine to create a new AIK pair using the TIMS when firstly joining the network. Before the AIK generation, the new machine has to verify that it is talking with the TIMS. Instead of trying to verify the MAC/IP binding of the TIMS, the new machine should challenge the TIMS using TPM. For sure, all the new machines have to be configured with the AIK public key of the TIMS before hand. Once the procedure is done, the new machine has to send its MAC address and PCR values (thought to be good values) via a secure channel, (e.g., using SSL, via a *httpd* daemon with a SSL module), to the TIMS. The TIMS then creates a new entry for this new MAC/AIK/PCR binding in its database for this new machine and distributes this new TPM Information Entry (TIE) to all *arpsecd* daemons running on machines within the network via a secure channel. Upon receiving the new TIE from the TIMS, the *arpsecd* daemon will insert the entry into its own internal database for future reference. It is worth noting that the AIK and PCRs within a TIE, are also used for TPM attestation by the TIMS for TIE update and revocation down below.

There are some special use cases we need to be cognizant of within the *arpsec/ndpsec* network. If the MAC address is changed for one machine, the machine must notify the TIMS. In this case, the machine could send a TIE Update message containing the old and new MAC addresses to the TIMS.

[3]Instead of creating a new *ndpsecd*, we enhanced *arpsecd* to support NDP.

The TIMS should first launch a TPM attestation (just like the *arpsecd*) towards the requesting machine. Only upon the success of the attestation will the TIMS update the existing TIE with the new MAC address and distribute the TIE Update to other *arpsecd* daemons. If a new MAC address is added into the machine, different actions will be taken based on whether the existing AIK is reused or not. If the new MAC address is bound with an existing AIK, a TIE Add message containing the new MAC address and the existing AIK should be sent to the TIMS. Again the TIMS will do the TPM attestation before creating a new TIE in its database and distributing this TIE Add to other *arpsecd*s. If the new MAC address tries to use a new AIK, then it is the same procedure as when a new machine joins into the network except an extra TPM attestation by the TIMS before the new AIK pair generation, making sure the machine is in a good integrity state. Any time one machine wants to discard its existing AIKs or change the TPM hardware or upgrade the BIOS/operating system (and thus changing the PCRs), it has to send a TIE Remove with all the registered MAC addresses before it starts a new registration. The TIMS will then clear all the TIEs with those MAC addresses and notify all *arpsecd*s to remove them as well. Though a hybrid network is not desired, the *arpsecd* has implemented a white list for the old machines, which may not have TPM hardware but have to be trusted anyhow, like gateway routers or DNS servers.

Besides the one-time effort of AIK creation, either TIE update or revocation may be a major management overhead. However, we argue here that both AIKs and OS (PCRs) are stable compared to session keys and application software. The distributing and updating for TPM information usually happens when hardware/OS upgrades, which is far less frequent than updates from applications. TIE revocation is directly related with AIK revocation, which usually happens when the ownership of TPM hardware is changed - a TPM reset in the BIOS. Once the TPM hardware is in use, we assume AIK revocation is rare.

VI. Performance Evaluation

To fully understand the overhead of *arpsec*, we compare our implementation with standard ARP as well as the proposals that most closely mirror the security guarantees that we provide, S-ARP and TARP. We follow the experiment settings of TARP, providing macro- and microbenchmarks.

We also investigate the overhead of *ndpsec*, comparing it to standard NDP as well as Secure Neighbor Discovery (SEND). According to the report by the Rocky Mountain IPv6 Task Force [10], there are three SEND implementations publicly available: Easy-SEND [8], NDprotector [7], and IPv6-Send-CGA [13]. Easy-SEND is a Java GUI-based implementation, which imposes extra overhead besides SEND processing. IPv6-Send-CGA is an old kernel patch based on Linux kernel 2.6, which is too old for current Linux systems. Unfortunately, there is no official SEND implementation in the Linux kernel (which also reflects the limited deployment of SEND in the wild). We choose NDprotector because it is a Python and C based implementation, which uses libpcap to capture and process IP packets in the user space efficiently.

Our testing environment involves 4 Dell Optiplex 7010 desktop PCs with quad-Core Intel i5-3470 3.20 GHz CPU, 8GB memory with Intel Pro/1000 full duplex Ethernet cards, running Ubuntu LTS 12.04 (x86-64) with Linux kernel version 3.2.0.55. All machines are equipped with TPM hardware from STM (version 1.2 and firmware 13.12), running Trousers API 1.2 rev 0.3. To eliminate the impact from exterior network traffic, all machines are isolated on a 1000-Mbps HP ProCurve switch. As S-ARP and TARP were written on Linux kernel 2.6, we have forward-ported the S-ARP and TARP implementations to our testing environment. We have also fixed some bugs in NDprotector and made sure it works correctly before running the evaluation.

A. Macro-Benchmark Testing of arpsec

We benchmark performance based on the round-trip-time (RTT) using `ping` to provide overhead from an application's or user's perspective. This benchmark we used is also consistent with what was used by S-ARP and TARP. Like TARP, we also implemented a custom `ping` command: `ncping` (no-cache ping), which clears the local ARP cache before each ICMP echo request is sent. With `ncping`, we can get the performance evaluation in the worst case and reveal the true overhead of different methods.

We have performed three groups of experiments: (*a*) `ping` with the target MAC/IP binding in the ARP cache, (*b*) `ping` without the target MAC/IP binding in the ARP cache, and (*c*) `ncping`. Each test consists of 1000 ICMP echo requests or 10×1000 requests for the `ping` without caching.

Figure 5a shows the RTT average (mean), min ($mean - 2\sigma^2$) and max ($mean + 2\sigma^2$) from the `ping` command with the target binding in the ARP cache. In all experiments, internal caching of S-ARP and TARP is enabled to maximize performance. Once the target binding is in the ARP cache, RTT average values of all these methods look similar, ranging from 0.210 to 0.240 ms. The max and min values among these methods are also comparable, which is intuitive given no ARP processing is occurring. We attribute *arpsec*'s slightly faster processing time to efficiency of processing in user space and of the relay system.

Figure 5b demonstrates the most common scenario, where the target binding is initially absent from the ARP cache. The first `ping` now takes much more time, as the ARP request will be broadcast and the corresponding reply will be handled before the binding can be added to the cache. Once the reply is processed and the binding is added, the performance is the same as in Figure 5a and the average RTTs converge to be similar to standard ARP.

To show the average time of the first-ARP-Reply processing, we repeated the 1000-run `ping` for 10 times. S-ARP, TARP and *arpsec* daemons were restarted each time to show the real processing time without caching. As shown in the figure, the left bar is the average over all 1000 pings, and the right bar is the average of 10 first-time `ping`s. The left bars show the amortized costs are close to cached processing. From the right bars, we see that after standard ARP, *arpsec* has the smallest overhead by 15.4%. S-ARP, without the help of caching,

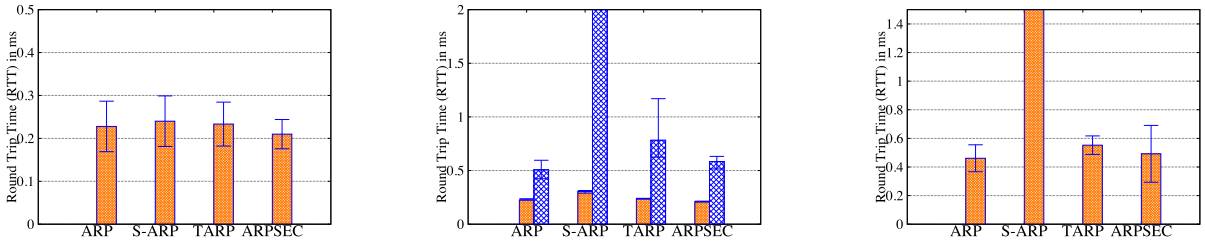


Fig. 5. The macro-benchmark of *arpsec*. (a) The average RTT (*ms*) of `ping` command with the target binding in the ARP cache for 1000 ICMP echo requests. (b) The average RTT (*ms*) of `ping` command without target binding in the ARP cache for 1000 ICMP echo requests over 10 runs. The right bar is the average RTT of the first `ping` during the 10 runs. (c) The average RTT (*ms*) of `ncping` command for 1000 ICMP echo requests.

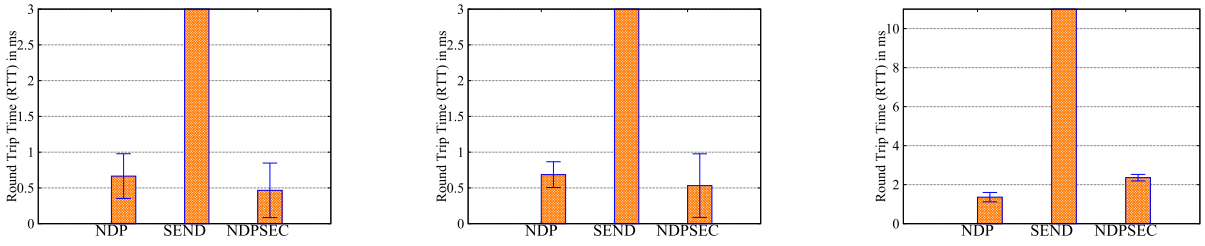


Fig. 6. The macro-benchmark of *ndpsec*. (a) The average RTT (*ms*) of `ping` command with the target binding in the neighbor cache for 1000 ICMP echo requests. (b) The average RTT (*ms*) of `ping` command without target binding in the neighbor cache for 1000 ICMP echo requests. (c) The average RTT (*ms*) of `ncping` command for 1000 ICMP echo requests.

introduces the biggest overhead, taking on average 64 *ms* for the new MAC/IP binding.[4]

Figure 5c displays worst-case performance using the `ncping` command, where the ARP cache will be flushed before each ICMP echo request is sent. With the help of internal caching and one-setup signature validation, TARP introduces a small overhead of 19.9% comparing with the original ARP. Even with caching, S-ARP still shows the largest overhead with the RTT average value 8.4 *ms* (not shown in the figure) because of the time synchronization and communication with the AKD. Comparing with S-ARP and TARP, *arpsec* performs the best, introducing a 7% overhead. Note that the TPM attestation is not triggered until the logic prover fails. The security parameter used by the prover is 5 seconds in our testing. Also, the RTT value of *arpsec* does not mean that TPM operation is fast, only that a quote can be amortized in the overhead. Because of the asynchronous operation of the TPM, the RTT value of *arpsec* is free from the degradation caused by the TPM attestation, but only limited by the user vs. kernel space communication and the logic prover. We will detail this in the later section.

B. Macro-Benchmark Testing of ndpsec

In a similar manner to the benchmarking of *arpsec* for IPv4, we benchmark the performance of *ndpsec* based on the round-trip-time (RTT) using `ping` to provide overhead from an application's or user's perspective. We test *ndpsec* against base NDP without any modifications and against SEND using NDprotector. We also use a modified `ncping` to clear the local neighbor cache before each ICMP echo request is sent.

We have performed three groups of experiments, with each experiment consisting of 1000 ICMP echo requests: (*a*) `ping` with the target MAC/IP binding in the neighbor cache, (*b*) `ping` without the target MAC/IP binding in the neighbor cache, and (*c*) `ncping`.

In all experiments, internal caching of base NDP and SEND is enabled to maximize performance. Though we did not encounter such a problem when evaluating performance in the IPv4 setting, neighbor cache bindings in the IPv6 setting are periodically cleared more frequently than the ARP cache bindings in the IPv4 setting, regardless of the protocol in use. As a result, after every 10-20 ICMP echo requests, the RTT of the following request would take time rivaling that of the request when no binding is available. Instead of tuning the expiration timer for the neighbor bindings, we leave the original system parameters unchanged, and include these values when calculating the average RTTs in order to remain faithful to the actual behavior of neighbor discovery in our default IPv6 setting.

Figure 6a shows the RTT average (mean) of the `ping` command with the target binding in the neighbor cache. The error bars show the range of values from $mean - 2\sigma^2$ to $mean + 2\sigma^2$. The RTT average for *ndpsec* and base NDP is similar, 0.467 *ms* for *ndpsec* and 0.665 *ms* for NDP. The range of values is also similar, though *ndpsec* posts a larger standard

[4]Although the overhead improvements of *arpsec* over S-ARP and TARP are less than 1 ms in most cases, even such small improvements are significant for kernel operations. Also note the averaging of times over 1000 runs makes the improvement less obvious for a simple operation as single ICMP echo requests.

deviation of 0.191 *ms* vs. NDP's 0.156 *ms*. The greater variation seen in *ndpsec* is a result of the longer intermittent pings, when compared to NDP. The RTT average for SEND is much larger at 10.71 *ms*, with a standard deviation of 54.82 *ms*. As was the case for *arpsec*, we attribute *ndpsec*'s slightly faster processing time to offloading the NDP processing from the kernel space to user space, which essentially makes the ICMPv6 software stack of the kernel simpler.

Figure 6b demonstrates the most common scenario, in which the target binding is initially absent from the neighbor cache. Shown are the RTT average (mean), $mean - 2\sigma^2$, and $mean + 2\sigma^2$. The first `ping` now takes much more time, as the neighbor solicitation and corresponding neighbor advertisement messages need be transmitted before the binding is entered into the cache. Once this step is completed and the binding is added, the performance is no different from that reflected in Figure 6a. Because of the longer intermittent pings, the first `ping` was no longer the sole outlier, so the timing of the first `ping` was not excluded. Again, the RTT average for *ndpsec* and base NDP is similar, being 0.532 *ms* for *ndpsec* and 0.686 *ms* for NDP. The range of values is also similar, but *ndpsec* again shows more performance variation, partly due to its larger max RTT (that of the initial `ping`) of 2.64 *ms* compared to the 1.49 *ms* max of NDP. The longer intermittent pings also contributed to the greater variation of *ndpsec*. The standard deviation for *ndpsec* was 0.222 *ms*, while *ndp* results showed a deviation of 0.090 *ms*. SEND's performance is well behind, with an average RTT of 9.144 *ms* and a standard deviation of 50.67 *ms*.

Figure 6c displays worst-case performance using the `ncping` command, which removes the target binding from the neighbor cache before each ICMP echo request is sent. Shown are the RTT average (mean), $mean - 2\sigma^2$, and $mean + 2\sigma^2$. In this worst-case setting, *ndpsec* introduces a 73.3% overhead compared to base NDP. The average RTT for *ndpsec* was 2.359 *ms*, compared to NDP's 1.361 *ms*; the difference between average RTTs is only about 1 millisecond. SEND, on the other hand, with its average RTT of 307 *ms* (not shown in the figure), takes almost 225x the time of base NDP. Furthermore, *ndpsec* actually has the least variation of the three when performing `ncping`, with a standard deviation of 0.083 *ms* vs. NDP's 0.120 *ms* and SEND's 7.276 *ms*.

C. Micro-Benchmark of arpsec

Using the GProlog-C interfaces, the logic prover can run as a pure C component without affecting performance of *arpsecd*. The prominent bottleneck in *arpsec* is then the TPM hardware, which is slow compared to a CPU [39].

Table I shows the key generation time of different methods. Here TARP* shows the ticket generation instead of the public/private key pair generation. *arpsec* is the cost of TPM AIK pair generation and the AIK public key certification. S-ARP has a mean key generation time of 90.364 *ms*. TARP is the fastest with the mean time 32.012 *ms* (public/private key pair + ticket). By contrast, *arpsec* is the slowest with the mean time 12.841 seconds, because we use the PrivacyCA to certify the AIK public key generated by the local TPM, following the complicated AIK certificate enrollment scheme described in the previous section. Fortunately, the AIK generation and certification is one-time effort. After this, the AIK private key is stored in the TPM and could be used in a secure manner. Moreover, both S-ARP and TARP have either the key expiration or the ticket expiration issue, which means after a certain time, either the key or the ticket has to be re-generated for each host within the network. In the long run, the time of *arpsec*'s one-time, offline key generation will be amortized by the key/ticket re-generation of S-ARP or TARP.

TABLE I
THE KEY GENERATION TIME (*ms*) WITH THE KEY LENGTH 1024 BITS AVERAGED BY 100 RUNS

Protocol	Min	Avg	Max	Mdev
ARP	36.16	90.36	330.7	34.79
TARP	5.17	31.01	69.48	10.83
TARP*	0.47	1.007	1.068	0.022
arpsec	3879	12841	46759	6062

TABLE II
THE TPM OPERATION TIME (*ms*) AVERAGED BY 100 RUNS

TPM	Min	Avg	Max	Mdev
AIKgen	864	9385	43716	5932
Rand	10.92	11.40	11.47	0.035
Quote	324.5	336.1	336.5	0.698
SigVerify	0.120	0.199	0.213	0.006
AttVerify	0.208	0.307	0.344	0.009

Table II profiles some TPM operations used by *arpsec*: AIK generation, cost of obtaining a random value, a TPM quote, signature verification, and TPM attestation verification, respectively. *AIKgen* is time consuming, as we saw from Table I. Otherwise, the quote operation is the slowest with the mean time 336.109 *ms*. We summarize the comparison among S-ARP, TARP and *arpsec* in Table III.

D. Micro-Benchmark of ndpsec

To better understand the disparity between *ndpsec* and SEND, we inspect the operation costs of each. Table IV presents the timing of operations used by SEND: key generation, cryptographically generated address (CGA) generation and verification, and signature generation and verification. The greatest cost is incurred when generating the CGA, taking 681.45 *ms* on average, but this is a one-time cost. RSA signature generation and verification together takes 218.14 *ms*. Though hosts typically need only perform a few such operations as part of neighbor discovery [3], these crypto operations take most time in runtime, and are one concern of SEND's adoption in the real world. Since there is no change in the TPM attestation protocol and the Prolog engine, *ndpsec* shares the same performance bottleneck as *arpsec* does, including the TPM hardware speed in Table I and Table II.

E. Effectiveness and Advantages of arpsec/ndpsec

Like S-ARP and TARP, *arpsec* is designed to defend against ARP spoofing attacks; *ndpsec* is designed to defend against spoofed neighbor solicitation or advertisement messages.

TABLE III
GENERAL COMPARISON AMONG S-ARP, TARP AND *arpsec*

Protocol	Mechanism	Formal Proofs	Remote Integrity	Change to ARP?	Change to Kernel?	Support for IPv6?	Overhead
S-ARP	PKI	N	N	Y	Y	N	Large
TARP	Ticket-based PKI	N	N	Y	N	N	Small
arpsec	Logic+TPM	Y	Y	N	Y/N	Y (*ndpsec*)	Small

TABLE IV
THE SEND OPERATION TIME (*ms*) AVERAGED BY 100 RUNS OF NDPROTECTOR USING ITS BENCHMARKING TOOL

SEND	Min	Avg	Max	Mdev
KEYgen	0.537	0.846	0.932	0.016
CGAgen	32.67	603.2	3893	404.0
CGAVerify	69.63	78.25	79.39	0.493
SigGen	142.9	146.8	149.7	0.928
SigVerify	67.69	71.34	83.17	0.531

We use one machine to craft an ARP spoofing response with the victim's IP address and its own MAC address. This spoofing message is sent to the target machine, which is running *arpsec* and has a cache mapping of the victim's IP and true MAC address. The logic layer within *arpsec* then fails the verification, since the MAC address presented in the spoofed response does not match that in the old MAC/IP binding. The TPM attestation which follows fails as well since *arpsec* cannot verify the signature of the attestation response using the victim's AIK public key, let alone the PCR measurement. Thus, the spoofing attack is defeated. *ndpsec* can defend against NDP spoofing attacks in similar fashion.

Note that unlike S-ARP and TARP, which can be defeated once the private key is leaked or the ticket is forged, e.g., when the host kernel is compromised, *arpsec/ndpsec* does not rely on the integrity of the remote machine, because the AIK private key is generated and saved inside the TPM. The fact that attackers cannot forge the TPM attestation response provides *arpsec/ndpsec* the strongest guarantee, in addition to introducing less system overhead than alternative solutions.

VII. DISCUSSION

arpsec and *ndpsec* are comprised of both a Logic Layer and a TPM Layer; we now discuss the implications of each of these layers. The logic formulation for the ARP binding system we have created (extensible to NDP) is simple, straightforward and intuitive, due in part to the simple design of ARP. Even with these simple logic rules, we are able to record all ARP cache update events, which implicitly capture the provenance history of the ARP cache. This could potentially allow the logic prover to act as a forensic system identifying compromised hosts, and the logic system itself can be extended to formalize other network protocols that build on ARP/NDP.

To implement the TPM attestation protocol, every machine must run a TPM daemon, which runs in user space. Even if this daemon is compromised, the attacker would be unable to circumvent the TPM attestation protocol. This would require either a forged TPM Quote on known-good PCR values, or for the attacker to possess the AIK private key. An attacker would be unable to recover AIK or edit the PCRs because they are stored onboard the TPM.

By adding a host to the ARP/NDP cache and purging it if the attestation fails, we make a design trade-off. In the worst case, we have made an incorrect binding for 300-500 *ms* until a TPM quote fails and a binding is removed from the cache. This can be exacerbated by TCP transmission delays. Currently, we set the TCP socket timeout to be 2 *s*. To optimize for security of the binding, we can purge it immediately after the challenge is sent and wait for the attestation before we update the cache again. This creates considerable overhead, however. Alternately, the ARP request could carry the challenge and the ARP reply could encapsulate the AT reply, at the cost of creating a protocol change to ARP or similar change to NDP.

Another limitation of using TPM attestation is that it only attests to what was loaded into the system at boot time (or load-time using integrity measurement). Runtime integrity checking provides more guarantees at the cost of requiring extra processors or significant overhead. The need for extra security co-processors or the incurring of significant system overhead prevents it from actual deployment. Integrity systems such as IMA [38] or PRIMA [16] could be integrated with *arpsec/ndpsec*. To further reduce the TCB size, we could potentially run *arpsec/ndpsec* in an isolated root of trust environment with a dynamic root of trust as offered by Flicker [25] or TrustVisor [24]. Even more, to detect the correct IP and MAC settings of the machine and endorse them, a trusted path system [53], [54] is needed for *arpsec/ndpsec* to securely retrieve settings from NICs.

Though not designed to defend against DoS attacks, *arpsec* could handle certain attacks against ARP, as could *ndpsec* for NDP attacks. As mentioned before, once the TPM attestation fails, the malicious MAC address or IP address will be added into the corresponding black list. When the same MAC address or IP address is contained in the following ARP/RARP reply, the reply will be dropped without processing. However, if the malicious MAC address or IP address keeps changing, *arpsec* has to examine each message, as the black list does not help in this case. Moreover, if the DoS attack is triggered from a higher-level network protocol, this would be out of the scope of *arpsec*. Such protection could be helpful against spanning tree attacks and VLAN hopping, however [41].

Each machine has to prove its identity using the TPM hardware and provide the good PCRs for future reference before joining the network. As long as the PCRs stay unchanged, we trust this machine. However, machines with good PCRs could still play MitM attacks by injecting a new TIE with a new MAC address into the TIMS. The IP address could

include the TIE, but IP addresses can be changed, and the solution would also burden the management of the TIE and complicate the design of the TIMS.

The performance of *arpsec/ndpsec* is limited by the TPM hardware. The TPM chip is designed to be cheap - only a few dollars. While the low price helps embed a TPM chip into each machine, even in mobile phones, it limits the scope of TPM usages. As shown in Table II, TPM Quotes impose a 336 *ms* delay when TPM attestation is required. As the TPM 2.0 library specification is published for review now, new TPM implementations based on it could further reduce the cost of Quote operations.

While TPMs have been widely deployed in servers, desktops, laptops and even mobile devices, many legacy machines lack them. For these machines, software TPMs could be used as a replacement, such as `libtpm` mentioned above, or vTPMs [34] in cloud environments. However, as the TPM Quote command occurs in software rather than hardware, a secure, trusted and isolated execution path [53], [54] is needed to guarantee trustworthiness. For the Cloud Computing environment, vTPM [34] can provide a working base. Substituting a software TPM may also result in more incurred computational overhead by both *arpsec* and *ndpsec*.

Intel's Software Guard Extensions (SGX) may provide an alternative to TPMs. SGX provides a means of verifying identity of both hardware environment and enclave through remote attestation [2]. Proving integrity is not so easy, as a quote only confirms the integrity of a particular enclave, as opposed to that of the entire system [14]. If stronger guarantees are required beyond those provided by PCR measurements, some sensitive portions of neighbor discovery may be placed into an enclave for attestation purposes. Requests by a possibly compromised remote machine could then be ignored if the associated code or data does not match what is expected.

VIII. Related Work

The security of address binding operations in IPv4 contexts, particularly ARP, has received considerable attention, focused on the problems summarized in previous sections. Although the threats are in a LAN context, as they impact correct packet delivery to destinations, it is critical that countermeasures to them be successfully employed.

Alternatives range from early suggestions for static bindings [5], which at normal scale on any type of network with frequent host additions/removals is intractable; to ARP modifications in S-ARP [6] and TARP [23] which introduce signed attestations, in the form of addresses bound to a public key or a ticket. S-ARP participants self-generate key pairs and register the public key in the central Authoritative Key Distributor (AKD). The AKD maintains the public key/MAC bindings and distributes these to all S-ARP hosts. TARP relies on Kerberos-style tickets and a central ticket-granting service to provide authentication, and is hence faster due to the use of symmetric keys. These approaches all require modifications to ARP itself, which limits adoption.

Another solution that does not require a modification to the ARP protocol examines a middleware approach to preventing ARP spoofing through a STREAMS based subsystem used to implement a Cache Poisoning Checker [46] that intercepts ARP requests and responses and inspects them for correctness. A more wide-ranging approach that deals with detecting spoofing of addresses in general and considers confidentiality is described in [40]. If IP address assignment is dynamic, as is likely to be the case, given the shortage of IPv4 addresses, this approach could be limited in its usability. It also does not address the problem of IP to lower level address binding, either in the discovery or allocation phases.

Other solutions use security policies to prevent ARP attacks. ArpON [33] defines different ARP binding policies for different networks, including static, dynamic, or hybrid networks. Instead of a centralized management server, hosts within the network all run the ArpON daemon and respect the same policies, thus adding complexity to defining and updating policies for different network environments.

Few proposals have considered a more holistic perspective of overall protocol design. A class of effort which incorporates a more fundamental evaluation of what can be done to integrate network security from protocol design to verification is proposed in [50]. It illustrates examples using BGP and metarouting but does not look at address binding. Generally there has been little emphasis as yet on approaches that tie together the use of system attestations utilizing hardware roots of trust with formally verifiable correct operation of infrastructural Internet protocols. In the case of *arpsec*, this is done using logic adjuncts to supplement sections of the existing protocol stack, so as to retrofit stricter trust properties to specific portions of the IPv4 network subsystem. From a hardware standpoint, while the Trusted Computing Group defines the TPM standard for roots of trust, and standardizes uses of the same through protocols categorized into different contexts such as trusted network connect (network access control), protocol operations supporting Internet infrastructure however have not yet been considered [12], [49].

The use of IPV4, and thus ARP, is relevant for the foreseeable future, but IPv6 deployment is increasing. The IPv6 Neighbor Discovery Protocol (NDP) [27] has capabilities beyond ARP, relying on autoconfiguration based on address binding. However, these new features do not immediately translate into heightened security, as many of the issues faced by ARP also carry over into NDP.

The IPv6 Neighbor Discovery specification suggests a couple of cryptographic solutions to NDP security: SEND [3] and IPsec for IP-layer authentication [20]. SEND relies on router authorization over certification paths, verification of senders' address ownership using cryptographically generated addresses (CGAs), RSA signatures, nonces, and timestamps. Besides NDprotector, user space implementations of SEND are provided by send-0.2, Easy-SEND, and Windows Secure Neighbor Discovery (WinSEND) [37], while kernel implementations include send-0.3 and ipv6-send-cga [1]. The practical adoption of SEND is limited by the high overhead of time-consuming cryptographic operations used in both generation and verification of CGAs and signatures. The IPv6 Router Advertisement Guard (RA Guard) [22] is proposed to overcome the deployment issues of SEND by providing lightweight filtering in the layer 2 network. However, as its name

implies, RA Guard focuses on router discovery/advertisement, leaving end users vulnerable to spoofing attacks.

The IPsec solution is composed of either one or both of IP Authentication Headers (AH) [18] and IP Encapsulating Security Payloads (ESP) [19]. This solution operates on the IP layer above NDP. AH provides integrity and data origin authentication, while ESP also provides confidentiality [20]. Despite the potential of IPsec, the complexity of manual key exchange, along with slowed performance, has discouraged its adoption to solve even network-layer problems.

An alternative Trust Based Neighbor Discovery Protocol (T-NDP) uses elliptic curve cryptography as a substitute for RSA to reduce both transmission delay and traffic overhead [35]. T-NDP incurs a 9% traffic overhead when a network contains 50 nodes, an improvement over SEND, but *ndpsec* outperforms this in the average case and in smaller local networks.

Besides attempts to secure the underlying protocol, another line of work is concerned with monitoring NDP activities. NDPMon is such a tool which monitors node activity and link-layer address changes, detecting attacks based on variations in behavior [4]. Such work is promising, though a monitor does not provide a complete solution and needs to be paired with adequate countermeasures to defeat detected attacks.

IX. Conclusion

This work has proposed *arpsec*, a secure ARP protocol that provides a logic prover to reason about the validity of ARP/RARP replies and uses TPM attestation to guarantee the trust in remote hosts. Compared to the original ARP, *arpsec* introduces only $7\% - 15.4\%$ system overhead. *ndpsec* is then implemented to apply the idea of *arpsec* for the NDP protocol in IPv6. The evaluation shows that *ndpsec* outperforms SEND by one order of magnitude. Both *arpsec* and *ndpsec* use a logic prover and TPM hardware and minimize system overhead without impacting current implementations. These formally-defined protocols based on bottom-up trust provide a first step towards a formally secure and trustworthy networking stack for both IPv4 and IPv6.

Acknowledgments

The authors wish to thank Stephen L. Squires and Jim Just for their helpful comments and discussion of the problem space. The views expressed in this paper are solely those of its authors and do not necessarily reflect the view of the Federal Communications Commission or the United States Government on the subject matter and related concepts.

References

[1] A. AlSa'deh and C. Meinel, "Secure neighbor discovery: Review, challenges, perspectives, and recommendations," *IEEE Security Privacy*, vol. 10, no. 4, pp. 26–34, Jul./Aug. 2012.

[2] I. Anati, S. Gueron, S. Johnson, and V. R. Scarlata, "Innovative technology for CPU based attestation and sealing," in *Proc. Workshop Hardw. Archit. Support Secur. Privacy (HASP)*, vol. 13. 2013.

[3] J. Arkko, J. Kempf, B. Zill, and P. Nikander. (Mar. 2005). *SEcure Neighbor Discovery (SEND)*. [Online]. Available: http://tools.ietf.org/html/rfc3971

[4] F. Beck, T. Cholez, O. Festor, and I. Chrisment, "Monitoring the neighbor discovery protocol," in *Proc. 2nd Int. Workshop IPv6 Today-Technol. Deployment (IPv6TD)*, 2007, p. 57.

[5] S. M. Bellovin, "Security problems in the TCP/IP protocol suite," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 19, no. 2, pp. 32–48, Apr. 1989.

[6] D. Bruschi, A. Ornaghi, and E. Rosti, "S-ARP: A secure address resolution protocol," in *Proc. 19th Annu. Comput. Secur. Appl. Conf. (ACSAC)*, 2003, pp. 66–74.

[7] T. Cheneau. (2013). *NDprotector*. [Online]. Available: https://github.com/tcheneau/NDprotector

[8] S. Chiu. (Apr. 2013). *Easy-SEND*. [Online]. Available: https://sourceforge.net/projects/easy-send/

[9] D. Diaz *et al. The GNU Prolog Web Site*, accessed on Apr. 2017. [Online]. Available: http://gprolog.org/

[10] J. Duncan. (Nov. 2011). *IPv6 Secure Neighbor Discovery (SeND) and CGA*. [Online]. Available: http://www.rmv6tf.org/wp-content/uploads/2012/11/IPv6_SeND_PPT1.pdf

[11] R. Finlayson, T. Mann, J. Mogul, and M. Theimer. (Jun. 1984). *A Reverse Address Resolution Protocol*. [Online]. Available: http://tools.ietf.org/rfc/rfc903.txt

[12] S. Frankel, R. Graveman, J. Pearce, and M. Rooks. (2010). Guidelines for the secure deployment of IPv6. NIST. [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-119/sp800-119.pdf

[13] Huawei Technologies. (Dec. 2009). *IPv6-Send-CGA*. [Online]. Available: https://code.google.com/archive/p/ipv6-send-cga/

[14] *Intel Software Guard Extensions Enclave Writer's Guide. Revision 1.02*, Intel Corp., Santa Clara, CA, USA, 2015.

[15] B. Issac, "Secure ARP and secure DHCP protocols to mitigate security attacks," *Int. J. Netw. Secur.*, vol. 8, pp. 107–118, Mar. 2009.

[16] T. Jaeger, R. Sailer, and U. Shankar, "PRIMA: Policy-reduced integrity measurement architecture," in *Proc. 11th ACM Symp. Access Control Models Technol. (SACMAT)*, 2006, pp. 19–28.

[17] B. Kauer, "OSLO: Improving the security of trusted computing," in *Proc. 16th USENIX Secur. Symp.*, 2007, pp. 1–9.

[18] S. Kent. (Dec. 2005). *IP Authentication Header*. [Online]. Available: https://tools.ietf.org/html/rfc4302

[19] S. Kent. (Dec. 2005). *IP Encapsulating Security Payload (ESP)*. [Online]. Available: http://tools.ietf.org/html/rfc4303

[20] S. Kent and K. Seo. (Dec. 2005). *Security Architecture for the Internet Protocol*. [Online]. Available: http://tools.ietf.org/rfc/rfc4301

[21] Lawrence Berkeley National Laboratory Network Research Group. (2006). *arpwatch: The Ethernet Monitor Program*. [Online]. Available: http://ee.lbl.gov/

[22] E. Levy-Abegnoli, G. van de Velde, C. Popoviciu, and J. Mohacsi. (Feb. 2011). *IPv6 Router Advertisement Guard*. [Online]. Available: https://tools.ietf.org/html/rfc6105

[23] W. Lootah, W. Enck, and P. McDaniel, "TARP: Ticket-based address resolution protocol," in *Proc. 21st Annu. Comput. Secur. Appl. Conf. (ACSAC)*, 2005, pp. 108–116.

[24] J. M. McCune *et al.*, "TrustVisor: Efficient TCB reduction and attestation," in *Proc. 31st IEEE Symp. Secur. Privacy (IEEE S&P)*, May 2010, pp. 143–158.

[25] J. M. McCune, B. Parno, A. Perrig, M. K. Reiter, and H. Isozaki, "Flicker: An execution infrastructure for TCB minimization," in *Proc. 3rd ACM Eur. Conf. Comput. Syst. (EuroSys)*, 2008, pp. 315–328.

[26] M. Technet. *Address Resolution Protocol*, accessed on Apr. 2017. [Online]. Available: http://technet.microsoft.com/en-us/library/cc940021.aspx

[27] T. Narten, E. Nordmark, W. Simpson, and H. Soliman. (Sep. 2007). *Neighbor Discovery for IP Version 6 (IPv6)*. [Online]. Available: https://tools.ietf.org/html/rfc4861

[28] J. Nathan. (2004). *Nemesis*. [Online]. Available: http://nemesis.sourceforge.net/

[29] P. Nikander, J. Kempf, and E. Nordmark. (May 2004). *IPv6 Neighbor Discovery (ND) Trust Models and Threats*. [Online]. Available: https://tools.ietf.org/html/rfc3756

[30] A. Ornaghi and M. Valleri. (2003). Man in the middle attacks Demos. Blackhat. [Online]. Available: http://www.blackhat.com/presentations/bh-europe-03/bh-europe-03-valleri.pdf

[31] A. P. Ortega, X. E. Marcos, L. D. Chiang, and C. L. Abad, "Preventing ARP cache poisoning attacks: A proof of concept using OpenWrt," in *Proc. 6th Latin Amer. Netw. Oper. Manage. Symp. (LANOMS)*, 2009, pp. 1–9.

[32] B. Parno, "Bootstrapping trust in a 'trusted' platform," in *Proc. 3rd USENIX Summit Hot Topics Secur. (HotSec)*, 2008, pp. 1–6.

[33] A. D. Pasquale. (2008). *ArpOn: ARP Handler Inspection*. [Online]. Available: http://arpon.sourceforge.net/index.html

[34] R. Perez, R. Sailer, and L. van Doorn, "vTPM: Virtualizing the trusted platform module," in *Proc. 15th USENIX Security Symp.*, 2006, pp. 305–320.
[35] K. Perumal and M. J. P. J. Priya, "Trust based security enhancement mechanism for neighbor discovery protocol in IPv6," *Int. J. Appl. Eng. Res.*, vol. 11, no. 7, pp. 4787–4796, 2016.
[36] D. C. Plummer. (Nov. 1982). *An Ethernet Address Resolution Protocol or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware*. [Online]. Available: http://tools.ietf.org/search/rfc826
[37] H. Rafiee, A. AlSa'deh, and C. Meinel, "WinSEND: Windows secure neighbor discovery," in *Proc. 4th Int. Conf. Security Inf. Netw. (SIN)*, 2011, pp. 243–246.
[38] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn, "Design and implementation of a TCG-based integrity measurement architecture," In *Proc. 13th USENIX Secur. Symp.*, 2004, pp. 1–17.
[39] J. Schmitz, J. Loew, J. Elwell, D. Ponomarev, and N. Abu-Ghazaleh, "TPM-SIM: A framework for performance evaluation of trusted platform modules," in *Proc. 48th Design Autom. Conf. (DAC)*, 2011, pp. 236–241.
[40] C. Schridde, M. Smith, and B. Freisleben, "TrueIP: Prevention of IP spoofing attacks using identity-based cryptography," in *Proc. 2nd Int. Conf. Secur. Inf. Netw. (SIN)*, 2009, pp. 128–137.
[41] L. Senecal, "Understanding and preventing attacks at layer 2 of the OSI reference model," in *Proc. 4th Annu. Commun. Netw. Services Res. Conf. (CNSR)*, May 2006, pp. 6–8.
[42] D. Song. (2000). *Dsniff*. [Online]. Available: http://monkey.org/~dugsong/dsniff/
[43] Symantec. (Dec. 20, 2000). *Solaris Kernel Tuning for Security*. http://www.symantec.com/connect/articles/solaris-kernel-tuning-security
[44] C. Tarnovsky, "Deconstructing a 'secure' processor," Black Hat Briefings Federal, Washington, DC, USA, Feb. 2010.
[45] J. D. Tian, K. R. B. Butler, P. D. McDaniel, and P. Krishnaswamy, "Securing ARP from the ground up," in *Proc. 5th ACM Conf. Data Appl. Secur. Privacy (CODASPY)*, 2015, pp. 305–312.
[46] M. V. Tripunitara and P. Dutta, "A middleware approach to asynchronous and backward compatible detection and prevention of ARP cache poisoning," in *Proc. 15th Annu. Comput. Secur. Appl. Conf. (ACSAC)*, 1999, pp. 303–309.
[47] TrouSerS. *The Open-Source TCG Software Stack*, accessed on Apr. 2017. [Online]. Available: http://trousers.sourceforge.net/
[48] Trusted Computing Group. *TPM Main Specification*, accessed on Apr. 2017. [Online]. Available: http://www.trustedcomputinggroup.org/resources/tpm_main_specification
[49] Trusted Computing Group. *Trusted Computing Group Glossary*, accessed on Apr. 2017. [Online]. Available: http://www.trustedcomputinggroup.org/developers/glossary
[50] A. Wang, L. Jia, C. Liu, B. T. Loo, O. Sokolsky, and P. Basu, "Formally verifiable networking," in *Proc. 8th Workshop Hot Topics Netw. (HotNets)*, 2009, pp. 1–8. [Online]. Available: http://dblp.uni-trier.de/db/conf/hotnets/hotnets2009.html#WangJLLSB09
[51] S. Whalen. (2001). *An Introduction to ARP Spoofing*, accessed on Apr. 2017. [Online]. Available: http://rootsecure.net/content/downloads/pdf/arp_spoofing_intro.pdf
[52] T. Zanussi *et al. Relay (Formerly Relayfs)*, accessed on Apr. 2017. [Online]. Available: http://relayfs.sourceforge.net/
[53] Z. Zhou, V. D. Gligor, J. Newsome, and J. M. McCune, "Building verifiable trusted path on commodity x86 computers," in *Proc. 33rd IEEE Symp. Secur. Privacy (IEEE S&P)*, May 2012, pp. 616–630.
[54] Z. Zhou, M. Yu, and V. D. Gligor, "Dancing with giants: Wimpy kernels for on-demand isolated I/O," in *Proc. 35th IEEE Symp. Secur. Privacy (IEEE S&P)*, May 2014, pp. 308–323.

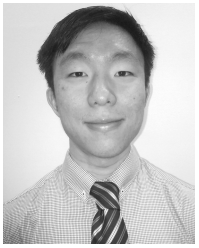
Dave (Jing) Tian received the B.S. degree from Qingdao Technological University, China, in 2006, and the M.E. degree from the Ocean University of China in 2009. He is currently pursuing the Ph.D. degree in computer science with the Department of Computer and Information Science and Engineering, University of Florida.

He was a Software Developer with the Alcatel-Lucent Research and Development Linux Control Platform Group, Qingdao, China, for four years. He is the Lead Graduate Student of the Florida Institute for Cybersecurity Research and is a Research Assistant, supervised by Prof. K. Butler. His research direction involves system infrastructure, security, and storage. He also has interests in Linux kernel hacking and compilers.

Kevin R. B. Butler (M'03) received the B.Sc. degree in electrical engineering from Queen's University, Kingston, in 1999, the M.S. degree in electrical engineering from Columbia University in 2004, and the Ph.D. degree in computer science and engineering from The Pennsylvania State University in 2010.

He is an Associate Professor with the Department of Computer and Information Science and Engineering, University of Florida, where he leads research in computer systems security within the Florida Institute for Cybersecurity Research. His current research interests include the security of systems and data, with a concentration on storage and embedded systems, mobile security and privacy, and cloud security. He also has interests in Internet security and applied cryptography.

Prof. Butler is a member of ACM and USENIX. He was a recipient of the National Science Foundation CAREER Award in 2013 and the Symantec Research Labs Graduate Fellowship in 2009. He is the Conference Chair at the 2017 IEEE Symposium on Security and Privacy and a Vice Chairman of the International Telecommunications Union's Focus Group on Digital Financial Services.



Joseph I. Choi received the B.S. degree in computer science from the University of Miami, FL, USA, in 2015. He is currently pursuing the Ph.D. degree in computer science from the Department of Computer and Information Science and Engineering, University of Florida.

Since 2016, he has been a Research Assistant with the Florida Institute for Cybersecurity Research, under his advisor Prof. K. Butler. His current research interests include systems and network security, cryptography, and Android/iOS.

Patrick McDaniel (F'14) received the B.S. degree in computer science from Ohio University in 1989, the M.S. degree in computer science from Ball State University in 1991, and the Ph.D. degree in computer science and engineering from the University of Michigan in 2001.

He is a Distinguished Professor with the School of Electrical Engineering and Computer Science and the Director of the Institute for Networking and Security Research, The Pennsylvania State University (Penn State). Prior to joining Penn State in 2004, he was a Senior Research Staff Member with AT&T Labs-Research. His research centrally focuses on a wide range of topics in security and technical public policy.

Prof. McDaniel is a fellow of the ACM and serves as the Program Manager and Lead Scientist with the Army Research Laboratory's Cyber-Security Collaborative Research Alliance.

Padma Krishnaswamy received the B.Tech. degree in electrical engineering from IIT Delhi, New Delhi, India, in 1982, and the M.S. degree in electrical and computer engineering from Oakland University, Rochester, MI, USA, in 1984.

She is a Senior Engineer with the Office of Engineering and Technology at the Federal Communications Commission, Washington, DC. Her interests include network architectures, measurement and characterization, and cybersecurity. A part of her responsibilities at the FCC is the initiation and facilitation of technical collaborations in relevant areas with academia, other Government Agencies and the industry at large. Her pre-FCC career spans over 20 years in the private sector in a variety of roles in Internet-focused research and development and program management, principally involving protocol specification, network systems design, and information assurance, with tours of duty at Bell Communications Research, Lucent Technologies, SAIC, and Juniper Networks, among others.