Vula: automatic local area network encryption

***Abstract*—This paper introduces Vula, a protocol and suite of Free Software tools for automatically protecting network traffic between hosts in the same Local Area Network (LAN). Without any configuration, or any user awareness, Vula automatically blinds passive adversaries. With user awareness and a small amount of interaction, it also protects connections using *.local* hostnames, or any other user supplied domain, against active adversaries. The protocol additionally provides protection against a passive adversary who is recording traffic today and who may have a quantum computer tomorrow. Vula's protections persist with network topology changes which occur naturally over time, allowing users to maintain cryptographic assurances while roaming between different LANs. The software operates without requiring centralized administration, specialized network equipment, or significant performance penalties.**

I. Introduction

Mass surveillance [72], [29], [32], [30], [33], [31], [34], [35] is not only a concern for backbone [75] networks; every network [88] is potentially a target. Local Area Network (LAN) security in the form of protection against surveillance is generally lacking in home, small business, and enterprise networks. We propose Vula: a protocol for automatically securing the LAN against eavesdropping and traffic tampering by other users, and/or network infrastructure equipment. Vula combines a secure tunneling protocol for secure point-to-point communications between Vula peers, multicast DNS for publishing and discovery of Vula peer associated metadata, along with easy Vula peer verification. Vula automatically builds tunnels directly between participating Vula peers on the same local network segment. We have selected the WireGuard Virtual Private Network (VPN) protocol outlined in Section IV for our Vula tunneling protocol. Unlike most deployments of WireGuard, Vula does not require the use of a third party located on another network. Users may additionally discover and/or verify Vula peers using user-friendly QR codes [52] for protection against active adversaries.

In addition to WPA [78], [93] encrypted home networks, open public WiFi hotspots, and business networks, we designed Vula with Ethernet networks such as Internet Exchange Points (IXP) [97] in mind which are known targets [71] of mass surveillance [1]. This is not the main target of Vula or this paper, but Figure 2 and additionally Figure 4 produced by the FLENT [47], [36] tool in the Appendix show that WireGuard performance keeps up with Gigabit line speed. We note that users of *any* wireless or Ethernet network will benefit from the use of Vula [2]. With Vula's ability to be gradually deployed, every host has a notion of cryptographic identity, and we think that with this improvement it will be clearer how to solve the problem of Internet-wide end-to-end encryption without resorting to sending unecrypted IP packets, encrypted but unauthenticated IP packets, or any of the various Single Points of Failure (SPOFs) as described in Section II.

A. Motivation

Public and private personal networks are commonly deployed using wired Ethernet (802.3) or wireless LAN (802.11) standards without comprehensive protection against surveillance adversaries. Ethernet networks are commonly deployed in consumer and commercial contexts without encryption of any kind. Authentication [2] of end-user's computers may be combined with a protocol such as MACsec [50] or WPA for link encryption between an end-user's computer and their immediate upstream Ethernet or wireless link. This combination of end-user authentication and link encryption does not provide end-to-end encryption [24] between hosts on the same Ethernet segment or the same IP multicast broadcast domain. It additionally adds a per-user administrative overhead, necessitating network equipment-specific administration which must further be specialized to support the required protocols. This may also create additional logging and other user data management complexity. Logs often present themselves as a tempting target for attacks. [84].

B. The Vula Proposal

We propose a system which provides automatic end-to-end encryption of IP packets with transitionally [85] post-quantum [70] forward-secrecy [65], without requiring centralized administration or specialized equipment beyond a basic Ethernet hub, switch, and/or wireless LAN. We also experiment with other Ethernet devices such as Thunderbolt [62], [82].

The purpose of Vula is to enhance the security of LAN traffic between hosts on the same IP multicast broadcast domain. We have taken great care to avoid introducing new vectors for host compromise in the design and implementation of Vula. Overall, Vula reduces the attack capabilities of various adversaries by following the principle of least authority (POLA [66]). We use but we do not completely trust the local network infrastructure. For systems participating in the Vula protocol, we reduce nearly all attack vectors to a denial-of-service that maintains confidentiality for traffic of Vula peers, also known as participants, using the protocol, while

[1] Two examples in the IXP community are London's LINX [15] and Frankfurt's DE-CIX [64] [58]. LINX denies that they are under such an order and they promise to reveal it if they learn about it [15] while DE-CIX has been to court to fight against such an order. In the IXP community, it is commonly understood in private discussions that both are under secret orders to export large volumes of traffic that their IXP carries to their local mass surveillance adversary, GCHQ and BND respectively.

[2] Any point-to-point traffic between participating systems will be protected by WireGuard, and the protection for point-to-point traffic is much stronger than the protection afforded by an unencrypted wireless network. It is also stronger in some ways than the protections afforded by WPA or WPA2 such as forward-secrecy.

maintaining backwards compatibility, and connectivity with non-participants. Traffic exchanged with non-participants of the Vula protocol remains unchanged, and may be optionally blocked, if desired. Vula provides a number of properties:

0) No infrastructure required.
1) Functional across organizational boundaries.
2) Fully automatic: LAN traffic between hosts with Vula installed is protected without any configuration whatsoever.
3) Works on temporarily offline or airgapped networks (e.g.: link-local [19] addressing on Ethernet, ad-hoc WiFi, Thunderbolt, etc).
4) Protects traffic using *existing* IP addresses (whether DHCP-assigned, link-local, or manually configured), so applications do not need to be reconfigured.
5) Protects entire IP packets regardless of sub-protocol (e.g.: UDP, TCP, ICMP).
6) Transitional post-quantum protection.

We consider previous and related work in Section II. We specify our threat model in Section III in terms of two broad categories of network adversary capabilities: *passive adversaries* [69], which are those who can observe some or all of the network's traffic but who lack the ability or opportunity to inject packets into the network or to prevent packets from being delivered, and *active adversaries* [25], [92] who do not lack those abilities. We present Vula in Section IV, our performance measurements in Section V, our security evaluation in Section VI, conclusions in Section VII, and we also present additional related information in Appendix A.

II. Prior and related work

Previous attempts to provide security for wired and wireless networks are myriad but in practice have largely failed to protect end-users from commonly understood surveillance adversaries such as corporate or government surveillance programs.

The general state of affairs for consumer or small business connections provided by internet service providers is to require a modem of some kind. This modem usually acts as a media converter; examples such as DOCSIS [79] or VDSL2 [28] convert their respective uplink technology into an Ethernet network or a wireless network, or both. The modem device generally either acts as an IP router, or as a router with network address translation (NAT), often with basic packet filtering capabilities. Often these modems provide both Ethernet and wireless LAN capabilities which are commonly configured as a single bridged network with a single multicast domain. The security of such networks from the perspective of a surveillance adversary often hinges on the strength of a wireless passphrase, with no protection of the Ethernet side beyond physical access restrictions. Worse, even with a strong passphrase, the ability to trivially predict factory-initialized or even user-chosen long term cryptographic keys from wireless routers has been available to unskilled adversaries for years [40], [61], [94]. For any given consumer wireless network deployment, one out of dozens of public attacks may be practically feasible without extensive knowledge requirements.

Previous attempts to create automatic or opportunistic [59] end-to-end encryption with IPSec [38] have generally foregone authentication, and attempt to solve a similar set of problems at internet scale by simply attempting to build IPSec connections to every host or network block. Alternative authentication using DNS is also possible for highly technical users who have end to end reachability such as a routable IPv4 or IPv6 network address, and who are able to control their forward and reverse DNS. This is not a common situation for many internet users who sit behind a carrier-grade NAT, or where their traffic is filtered to prevent running of services without permission from their ISP.

Our proposed protocol attempts to solve a similar and the related set of problems at a smaller scale without any trusted third parties, and without attempting to create trust relationships between people who are unable to meet and verify cryptographic keys.

A. Related protocols: 802.1x and MACsec

Protocols primarily deployed in corporate and academic environments center around access-control in an attempt to address some security concerns posed by adversaries with or without permitted access to the LAN.

These networks generally provide authentication, authorization, and accounting (AAA) services. A popular example in academic environments is the Eduroam [99] network which uses WPA2-Enterprise. In passive adversary models, Eduroam protects against a local surveillance adversary by shifting the risk of the user's authentication traffic with EAP [95] to their home academic institution. In active adversary models with Eduroam, client software may or may not [99, Section 7] be configured correctly to provide active adversary protection.

Wireless networks with AAA services as part of wireless WPA2-Enterprise or Ethernet networks protected by 802.1x are often used without any additional security measures against potential surveillance adversaries after authentication. Ethernet networks may also be deployed with MACsec [50] in an attempt to thwart adversary access to the network infrastructure, not as a matter of protecting against surveillance adversaries. MACsec provides link security in the form of encryption, integrity, and authenticity between a given client's Ethernet interface and the immediate upstream switch. This scenario does not provide end-to-end security when used for access control.

While GNU/Linux and some other operating systems do support MACsec and 802.1x, it is uncommon for consumer-grade switching equipment to support it and when enterprise switching equipment offers support it typically requires a paid license; these issues hinder general adoption. It may also be subject to export control, especially when specialized hardware is required for a given platform deployment of MACsec. MACSec is typically not end-to-end encrypted but host to switch-port, traffic after the entry-switch is completely unprotected.

Switching infrastructure with MACsec generally has access to encrypted and unencrypted Ethernet frames while a passive surveillance adversary is generally only able to intercept Ethernet frames through Ethernet cable tapping or by monitoring radio emissions for a related wireless network. An active adversary may compromise the wireless drivers of a client [9], an access point, switch, and/or router to gain access to key

material. In the general case, end-to-end encryption is a much stronger and much more desirable protection than the partial protection offered by 802.1x networks even when deployed in tandem with MACsec.

Post-Quantum MACsec Key Agreement for Ethernet Networks [22] suffers from the same problems as MACsec in that it is not an end-to-end protocol, it is layer-two, and at this time it is an experimental protocol which has not been adopted by any MACsec vendors.

Wireless network security protocols attempt to tackle confidentiality, integrity, and access control in a manner which is generally not secure against surveillance adversaries as shown below. We consider the WPA1 and WPA2 personal protocols which are commonly used as they require no additional authentication servers or configuration beyond setting a passphrase. Long term monitoring of passphrase authenticated wireless networks with poor passphrase rotation policies is especially problematic. Given a password, a passive adversary is able to recover plaintext for each session for which they have recorded a successful authentication and association, in addition to the encrypted traffic that they wish to decrypt. Handshakes occur frequently, e.g. devices that enter a low power mode generally re-authenticate after waking from a sleep mode, so that adversaries arriving too late need not wait long. Many wireless networks do not support protected management frames [51] and so adversaries commonly are able to force a disassociation without knowledge of the passphrase. Users' software will commonly automatically reconnect after an adversary has forced a disassociation. This extremely common issue gives an adversary the chance to force and then observe a fresh handshake and thus mount the above-mentioned attack. Furthermore, a recording of the handshake permits mounting offline password-guessing attacks.

With Vula, these attacks are mitigated with regard to traffic confidentiality concerns. For example, when Vula is deployed for users on a WiFi network, an attacker breaking the WPA/WPA-2 access controls and thus joining the network is restricted to performing only denial-of-service attacks instead of being able to mount a full on-path active Machine-In-The-Middle (MITM) attack with access to unencrypted IP packets.

B. Comparison with other projects

There are a wide variety of tools which can be used to create end-to-end encrypted tunnels between hosts, or which share other superficial similarities with Vula. To our knowledge, however, none of them achieve Vula's design goal of providing fully-automatic end-to-end encryption of local area network traffic. We present a comparison in Table I.

Projects such as Tailscale [89], Headscale [37], and innernet [91] are similar to Vula in that they can be used to encrypt traffic between hosts on a LAN using WireGuard tunnels, but they differ in some important respects: They only create tunnels between hosts that are logged in to the same account on a centralized coordination server. Tailscale outsources the operation of this component to Amazon, a surveillance actor. Headscale and innernet provide free software implementations which can be self-hosted, but the server remains a single point of failure. These systems use a different IP range inside and outside of the tunnels, so LAN-based applications need to be reconfigured to benefit from it. They do not provide any post-quantum protection. Furthermore, Tailscale requires internet access, thus is unsuitable for offline, or airgapped networks. Tailscale also requires an additional trust relationship with at least one but likely more 3rd parties: Tailscale and one of Google, Amazon, Microsoft, or an email provider. Nebula uses a custom protocol that its authors claim is based on a Noise Protocol Framework [73] handshake and it has yet to receive the scruitiny of other instantations such as WireGuard [26]. Nebula, like Tailscale, is used to construct a similar organizational structure VPN mesh. Tailscale, Headscale, innernet, and Nebula are unsuitable for dynamically discovered peers, air-gapped network segments, and/or multi-organization protection, and these properties are not goals of the respective projects.

With the exception of TCPcrypt and IPsec OE, the other projects listed in Table I are all designed to protect traffic between hosts which are configured to be part of a single organization, whereas Vula provides automatic encryption of traffic between *all* locally-reachable hosts that are running the software. TCPcrypt is an outlier, in that it does provide opportunistic encryption between hosts without any configuration; however, it only protects TCP traffic, does not provide secure names, its key verification system requires application-specific support, and it appears to be an out-of-tree Linux kernel patch. These and other deployment impediments have prevented its adoption even after standardization [14]. For these reasons, we find TCPcrypt unsuitable for Vula's needs but we remark it is still an interesting design with important goals. IPsec OE is designed to provide opportunistic encryption, but has numerous [100] shortcomings [41], including vulnerability to quantum computers, and it has failed to gain adoption, partially because it requires manual configuration.

C. Star network WireGuard deployments

While WireGuard's architecture is defined in terms of *peers*, deployments often use a hub-and-spoke network topology wherein multiple hosts which are commonly referred to as *clients* connect to one or more centralized hosts which are commonly referred to as *servers*. While this topology can be used with WireGuard or another VPN in the context of a local Ethernet segment, it presents a number of downsides related to the server(s) being SPOFs. Bandwidth SPOF: The total bandwidth available for clients to communicate with each other is limited to the bandwidth of the server(s) they are communicating through. When that bandwidth is exhausted, performance suffers for all clients. Confidentiality SPOF: Due to the absence of end-to-end encryption, a central server holds *excess authority* allowing it to capture and/or modify traffic from many clients which are routing through it. Availability SPOF: The ability of clients to communicate with each other is entirely dependent upon the availability of their centralized servers.

D. Point-to-point VPN deployments

It is possible to manually configure WireGuard or another VPN in a point-to-point topology on a LAN to achieve some of the same properties that Vula provides. However, there are some shortcomings to a manual approach which also apply to the star network topology that Vula addresses.

	0	1	2	3	4	5	6	7	8
Tailscale [89]	✗	to specific IP addresses	✗	coordination server	✗	✗	✓	✓(client) + ✗(server)	WireGuard
Headscale [37]	✗	to specific IP addresses	✗	coordination server	✗	✗	✓	✓	WireGuard
innernet [91]	✗	to specific IP addresses	✗	coordination server	✗	✗	✓	✓	WireGuard
Nebula [86]	✗	to specific IP addresses	✗	certificate authority	✗	✗	✓	✓	Custom protocol
MACsec [50]	✗	Ethernet link w/host and switch	✓	RADIUS server	✗	✓	✗	✓(client) + ✗(switch)	MACsec
TCPcrypt [14]	✓	TCP traffic w/participating hosts	✓	none	✗	✓	✗	✓	TCPCrypt
IPsec OE [100]	✗	w/participating hosts (LAN & WAN)	✓	DNS+DNSSEC and/or CA	✗	¡	✗	✓	IPSec cipher-suite
Vula	✓	w/participating hosts (LAN)	✓	none	†	✓	✓	✓	WireGuard

Table I: **Comparison of properties:** ✗: no, ✓: yes, ¡: not default, †: transitional, 0: zero configuration, 1: encrypts, 2: works offline, 3: required infrastructure, 4: post-quantum, 5: protects traffic using existing IPs, 6: secure hostnames, 7: free software, 8: encrypted transport.

For example, Vula automatically securely computes and sets the pre-shared key (PSK) value in the WireGuard protocol for all peers. The use of an additional PSK is to add transitional post-quantum security to the WireGuard protocol, but normally requires manual configuration. We use CSIDH [16] as described in Section IV-E to compute shared symmetric keys between pairs of peers and the result is used as a PSK. Vula explicitly supports rotation of the CSIDH keypairs on a regular basis as this rotates the PSK shared between peers.

1) Management: Adding new hosts to a point-to-point WireGuard overlay network requires configuring each existing host with the new host's key and IP address, and configuring the new host with all existing hosts' keys and IPs. Vula performs this key distribution and routing configuration automatically. We explain the use of multicast in Section IV-B.

2) Addressing: In most point-to-point WireGuard configurations, the IP subnet used for VPN traffic is separate from the one used for other traffic. This means that traffic to and from typical LAN applications using mDNS [21] hostnames will not be automatically encrypted without additional configuration of each application. Vula, in contrast, encrypts all connections between participating peers while applications continue using their existing LAN IP addresses and hostnames.

III. Threat Model and Design Considerations

In the examples below, we require that Vula users have at least a single IPv4 address, and are connected to an IP network through an Ethernet switching fabric and/or a wireless LAN. To optionally protect upstream traffic, we additionally assume that any hypothetical user is on a LAN which has at least one IPv4 gateway with connectivity to the wider internet. We choose the strongest adversaries to defend against, thus assume that the adversaries may record all IP packets. This would happen for unprotected wifi, if an adversary has access to a switch mirroring port, or if the adversary has any WPA/WPA2 passphrases.

A. Unilateral Surveillance Adversary

The possibility of **Unilateral Surveillance**, such as the wideband monitoring of all wireless networks in an area, is a well-understood attack vector. With commonly deployed consumer or so-called *prosumer* [80] [56] equipment, capture of association handshakes with a wireless access point will allow an attacker to guess a passphrase and later decrypt captured wireless traffic. Interception of wireless networks is so common that there are cloud-based services [63] as well as GPU optimized key recovery tools [1] for attacking (WPA) cryptographic handshakes in service of decrypting intercepted data.

A simple and relatable example is a curious neighbor who lives in close physical proximity to a wireless network such that their basic interception equipment is within radio range. They may passively capture wireless traffic over long periods of time, decrypt it at a later date, and refrain from joining the wireless network lest their subterfuge be detected. An example of a tool that may be used by such a neighbor is the SPARROW II as seen in Appendix Figure 9. Their home devices may otherwise be compromised [7] and used by an adversary.

Another passive example is the NSA program OVERHEAD [39] which performs wireless network packet capture *in space using satellites* [83]; data from that program may be fed into systems such as XKeyscore [5].

B. End User

There are a variety of ways that **End Users** can attack each other on a Local Area Network, due to the reliance on vulnerable protocols such as the Dynamic Host Configuration Protocol (DHCP) [27], the Address Resolution Protocol (ARP) [74], and the Domain Name System (DNS) [67]. By simply sending a few malicious ARP or DHCP packets, an end user can easily intercept other users' traffic on a switched network. Users may also attempt to use packet-in-packet [43] smuggling to interfere with other users.

An additional adversary to consider would be an active adversary using NIGHTSTAND [9] as shown in Appendix Figure 8 which is an attack suite to compromise authorized wireless equipment.

C. Network Operator

We presume that the **Network Operator** is able to enable port mirroring for an entire switching fabric. This means that they are able to passively collect every packet sent within the switching fabric, as well as enabling full packet capture on an upstream router which sends all user data to and from the internet. We consider this adversary to be close to the Dolev-Yao [25] model for an attacker, in that they are able to arbitrarily disable user access, change passwords, capture packets, inject packets, delay delivery, and more. Usually this is only possible in the upstream equipment providing network access to the internet. They are able to carry out all of the other attacks enumerated. In an ideal environment, at least

one end user is actually the **Network Operator**. So while all powerful, we presume that a user will not attack themselves but rather consider what is possible if their own equipment is compromised [44].

D. Vula peer states

We distinguish peers by their *verification state* and their *pinned state*. By default, peers are either *unverified* and *unpinned*, or *unverified* and *pinned*. Unpinned should also be thought of as *replaceable* by another Vula peer, and pinned should be thought of as *permanent* where no other Vula peer may conflict with the peer's claimed resources.

In Vula each user has a cryptographic identity given by a long-term Ed25519 [11] key. These keys certify all other cryptographic keys used in the Vula protocol. Verification is an out-of-band process whereby Vula users compare these Ed25519 identity keys. The identity keys are the only keys that do not change while all other cryptographic keys may be regularly rotated. The Ed25519 public key for signatures is known as the *verification key* (vk) and it is used to sign all Vula *descriptors* - either broadcast to other peers over the network or scanned as part of a QR code verification process. Descriptor smuggling with other protocols is left as an exercise for advanced users.

In order to also protect against active attackers, continuity of vk public keys must be enforced with respect to both hostnames and IP addresses. This leads to a security-convenience trade-off: if continuity of vk public keys is enforced by default for all peers, naturally-occurring name or IP conflicts will sometimes lead to an inability to communicate. For this reason, we introduce a user-controlled boolean state for each peer called *pinned*. Continuity of vk public keys is only enforced for peers in the *pinned* state. Pinning peers allows users to have the benefit of protection against active adversaries at the cost of needing to manually resolve hostname or IP address conflicts.

Pinning a peer creates a binding from the peer's long-term *verification key* to lists of hostnames and IP addresses which that peer has been known to use. A single pinned peer may be associated with any number of hostnames and IP addresses, while a given hostname or IP address may never be associated with more than one peer.

A pinned peer is a permanent peer. A pinned peer has a permanent route and traffic for that peer is always directed into the local Vula device; pinned peers do not expire. If no Wireguard session exists between the user and their respective peer, the traffic is never emitted onto the network as the device will *fail closed* (See [77, Section 4.1] for the defnition.). Pinned peers cause a denial of service with non-participants or colliding Vula peers by design if other users obtain the same IP address or use the same host name; this is independent of those being Vula peers or non-participants. Consider the following scenario: Alice uses Vula and has pinned Bob; she is hostname.local with 10.0.0.2 as her IP address. Bob uses Vula; he is at otherhostname.local with 10.0.0.3 as his IP address. Bob leaves the network. Carol arrives on the network. Carol does not use Vula. The DHCP server gives Carol 10.0.0.3 as their IP address. Alice knows that only Bob is available at 10.0.0.3, but Bob's WireGuard does not reply, so Alice cannot talk to Carol and any attempt at communication will fail closed. Unencrypted traffic will not leak out of the WireGuard tunnel. When Bob returns, and the IP address is still in use he will obtain a new IP address which Alice will learn through a Vula broadcast. Alice will see that Bob now has at least two IP addresses, and Alice will still be unable to reach Carol until Carol obtains an IP address not used by a pinned Vula peer. Carol is oblivious to all of this.

An unpinned peer is a temporary peer. Unpinned peers remain until the user's system moves to another network segment, until the peer descriptor expires, or until a new peer announces resources that conflict with this replaceable peer. To securely reach the peer, Vula adds specific routes for peer addresses to the `vula` device, and Vula additionally configures the same addresses as being associated with the cryptographic keys for the peer on the `vula` device. When the peer is replaced or expires, the routes are removed, and the cryptographic keys are removed from the `vula` device.

Often participants and non-participants are mixed on private network segments that use commonly allocated private [68] IP addresses. To prevent denial of service for potentially communicating hosts, unpinned peers *fail open* for the benefit of non-participant hosts.

E. Cryptographic choices

A protocol is said to have *perfect forward-secrecy* (PFS) [65] if compromise of long-term keys does not compromise past session keys, Vula brings this property to IP traffic for participating systems. From the perspective of public-key cryptography, the attack targets in Vula are reducible to a few specific problems. An attacker wishing to forge Vula peer descriptors must be able to forge Ed25519 signatures to break authenticity of the peer discovery and key exchange mechanism. If the authentication process is not broken, the attacker wishing to recover plaintext traffic must record traffic, and then they must break X25519 [10] as used in WireGuard, and CSIDH-512 to recover the PSK.

F. Automatic protection against passive adversaries

As Vula automatically encrypts traffic between hosts while they are connected to the same broadcast domain, in the absence of an active attacker it will always deny passive adversaries the opportunity to decrypt traffic that they capture.

G. Automatic protection against active adversaries

Encryption relying on an unauthenticated key exchange is, of course, intrinsically vulnerable to key-substitution attacks by active adversaries who are present at the time of the initial key exchange. The concept of authentication, however, is meaningless in the absence of a notion of identity. In the LAN setting in which Vula operates, there are several notions of identity, such as hostnames, IP addresses, and MAC addresses, but none of these are intrinsically authenticatable. Therefore, without manual key verification or dependence on some sort of public key infrastructure, it is not possible to automatically authenticate the initial communication between two hosts on a LAN. However, it is possible to automatically provide protection against active adversaries who only become active after that point, by following the trust-on-first-use [98]

(TOFU) pattern often employed by users of SSH: Keys are implicitly assumed to be valid for hosts which have never been contacted before, and *c*ontinuity of vk public keys is enforced for any subsequent communication. Unlike SSH, where users are prompted to explicitly make the TOFU decision, Vula has a configuration option called *pin_new_peers* which causes newly-discovered peers to be automatically marked as *pinned*. This is not the recommended default as it imposes user interface awareness requirements on users as explained in Section VI-D8 and shown in Figure 1.

Peers automatically *pinned* in the *pin_new_peers* state are vulnerable to an active attack *only at the time that they discover peers for the first time*. If their initial discover was not compromised, Vula protects them against active attacks at any later time.

For full protection against active attackers, including those who could be present at the time of first contact, manual key verification is necessary. When a peer is manually verified, it is marked as *pinned* and is also marked as *verified* to allow the user to distinguish it from peers pinned automatically by the *pin_new_peers* state.

Vula provides a convenient-to use QR code-based tool for performing peer verification. We describe this verification process further in Section 6. To protect against active adversaries who are present at the time of initial contact, it is necessary to manually verify fingerprints.

H. Security-convenience trade-off

We consider the default behavior for Vula protocol implementations with regard to the usability and security outcomes.

1) pin_new_peers = true: As stated above, using the *pin_new_peers* mode has the advantage that unverified peers for whom the initial contact was not compromised are automatically protected against any subsequent active attacks. The disadvantage is that when an IP address which has been previously used by a Vula peer is later reassigned to a new host, Vula users who learned about the previously associated IP and are using *pin_new_peers* as their default mode will be unable to communicate with the new host, regardless of whether it runs Vula itself, until they explicitly remove the IP address as associated with the previously existing public key or the new host moves to a previously unassigned IP address. Pinned peers accumulate IP addresses and hostnames where manual removal may be necessary.

2) pin_new_peers = false: Marking new peers *unpinned* by default has the disadvantage that new peers will remain vulnerable to active attacks until they are explicitly marked as *pinned* or *verified*. It has the advantage that it will gracefully handle IP address reassignment and/or hostname collisions without requiring any user interaction, so Vula could conceivably be widely-deployed and enabled by default without causing significant inconvenience while also thwarting passive adversaries. Unpinned peers do not accumulate IP addresses and hostnames, they are replaced by any conflicting announcements, and they expire automatically.

I. Summary of protections

Vula should always provide confidentiality with respect to passive adversaries. For peers that are *pinned*, it will also protect against active adversaries as long as those did not compromise the first contact. For peers that are manually verified, a successful verification ensures security against attackers which were active even at the time of the first contact as any key-substitution attack would make manual verification fail.

Although Vula protects the confidentiality of network traffic between verified peers against both passive and active attackers, we do not claim to be able to prevent traffic analysis attacks which may be revealing. We also do not attempt to prevent packet delaying or Denial-of-Service attacks, and we admittedly do allow for some new minor avenues by which DoS attacks can potentially be executed as explained in Section VI-D. However, these are not significantly different from the DoS vulnerabilities which are inherent in the LAN setting. We remark that there is a need to design and deploy enhancements to the underlying LAN protocols such as DHCP and ARP.

IV. Detailed Protocol Description

The Vula protocol does not rely on any infrastructure and is purely a peer-to-peer protocol. Every participant that wishes to use the protocol must install the Vula software on the computer system which has traffic it wishes to protect with the Vula protocol. As a concrete example, a router running the Vula software is able to provide a locally secured WireGuard tunnel to any downstream clients who also run Vula. Downstream clients may then communicate through the router to the internet with all traffic protected between their respective systems and the router itself. If the router itself does not have a Vula peer upstream or another VPN tunnel, the traffic will be unencrypted as it traverses subsequent routers. The benefit of running this software on a router is that normally clients may intercept each other's traffic with minimal effort as explained in Section VI-D1, and with Vula, they would need to violate some assumption of the protocol which is protected by strong cryptography.

A. WireGuard

The Vula protocol relies on a secure tunneling protocol for protecting IP packets between participating systems. We have selected WireGuard [26] as our encrypted tunneling protocol on the basis that it is well understood, peer-reviewed, extremely efficient, performs exceptionally fast packet transformation even under heavy system load, and is now a part of the Linux kernel shipping with a number of GNU/Linux distributions. Unlike IPsec, it is not suspected of being sabotaged by the NSA. IP traffic between any given pair of hosts participating in the Vula protocol is protected by WireGuard. WireGuard is modeled after the Noise Framework IK pattern [73, Section 7.5] which in turn has been updated to reflect some of the needs of WireGuard. The IK pattern optionally allows any pair of peers to use a symmetric pre-shared key (PSK) to make the WireGuard protocol transitionally post-quantum in addition to keys derived from both ephemeral and long term keys. We take advantage of this and generate a pair-wise shared secret with CSIDH. To a third party observer, the use of a PSK is indistinguishable from other WireGuard traffic which does not use a PSK. WireGuard presents an interesting constraint: peers must be configured, and thus keys must be known before the protocol is ready for use. This raises a number of

questions about efficient key exchange, as well as questions about rotation of keys used in the protocol. Session keys rotate every few minutes under normal usage conditions, though long term keys must be rotated manually. WireGuard leaves discovery of peer public keys, as well as configuration, as a problem for the user to solve. Vula automates everything that WireGuard has left for users to otherwise manually configure.

B. mDNS/DNS-SD: decentralized Vula peer discovery

Each user's Vula descriptor contains their long term WireGuard public key, along with their CSIDH public key. We have chosen to automatically distribute Vula descriptors using multicast DNS (mDNS [21]) and DNS Service Discovery (DNS-SD [20], [23]), on the local network segment. DNS-SD specifies structure for DNS records which may be used to facilitate service discovery using DNS. When mDNS and DNS-SD are combined together, DNS queries for local hosts do not leave the local network segment to be properly resolved. Each Vula peer must publish a *Service Name* under *_opabinia._udp.local.* [3] for their host. A query for the respective Service Name should return a TXT record containing a list of values representing a Vula *descriptor*. The values required for Vula are enumerated and briefly explained in Table II.

key	example value	description
addrs	192.168.6.9	List of addresses for Vula peer
c	36e8...c764	CSIDH public key for deriving pair-wise PSKs
dt	86400	Seconds after vf that descriptor is valid
e	0	Flag indicating that a peer is ephemeral
hostname	alice.local	Hostname
pk	cW8Ek...R0=	WireGuard Curve25519 public key
port	5354	WireGuard UDP port number
r	1	IP forwarding services are available to peers
s	adsLEe...a=	Ed25519 signature over Vula descriptor
vf	1601388653	Starting validity of descriptor in seconds since 1970
vk	ptKKc...0M=	Ed25519 public key used to sign Vula descriptor

Table II: mDNS/DNS-SD Vula descriptor key, value examples

We presume that the generally insecure nature [45] of DNS, even in the local LAN context with mDNS and DNS-SD, is understood. As an alternative to DNSSEC [81] or other proposals [44], Vula enhances the security of DNS-SD service records with cryptographic signing of the service descriptors. All computers which wish to deploy Vula must be able to send IP packets to and receive from **224.0.0.251:5353** or **[FF02::FB]:5353** with the correct multicast MAC addresses for any IP packet they send or receive respectively. These packets should only contain properly formatted mDNS queries or answers.

When a peer publishes its descriptor, all of the values besides the signature *s* but including the verification key *vk* are ordered and serialized into a string. A signature over the string is computed and its value is stored in the final item, *s*; the resulting set of name-value pairs is then added to the DNS-SD Service record.

The verification key (vk) is used for authenticating Vula protocol messages; currently the messages are used for peer discovery and peer consistency, stateless rotation of IP addresses such as when a DHCP server gives a DHCP client a new IP address, for rotation of the CSIDH public key and derived PSKs, and for rotation of the WireGuard Curve25519 public key used by the `vula` device. This device appears as a normal network interface with the name *vula* in various system configuration tools.

[3] See Opabinia Regalis from the Middle Cambrian

C. Vula Protocol logic

In this subsection of the paper, we walk step-by-step through a full protocol run for two peers on the same LAN segment.

The same protocol scales to n possible peers. One peer queries for the DNS-SD service and n devices may answer. The only limit to the number of peers is the number of addresses on the local segment, any internal limit on peers that the WireGuard implementation may impose on configuring the network interface, and on system memory.

When a peer receives a new descriptor, it evaluates it according to a policy engine which considers the peer's current state and enforces various constraints. We define the policy engine as a pure function which computes the next policy state from the current state and some event: `ProcessEvent(PolicyEngineState, Event)` $\Longrightarrow$ `PolicyEngineState'`. Events include incoming descriptors, changes in the system state such as an IP address being configured or unconfigured by a DHCP client, or some user action. The `Event` objects contain a timestamp indicating when the `Event` occurred, which allows `ProcessEvent()` to make decisions which include time despite being a pure function. A flowchart showing an overview of the policy engine's handling of the *incoming descriptor* event is shown in Figure 1.

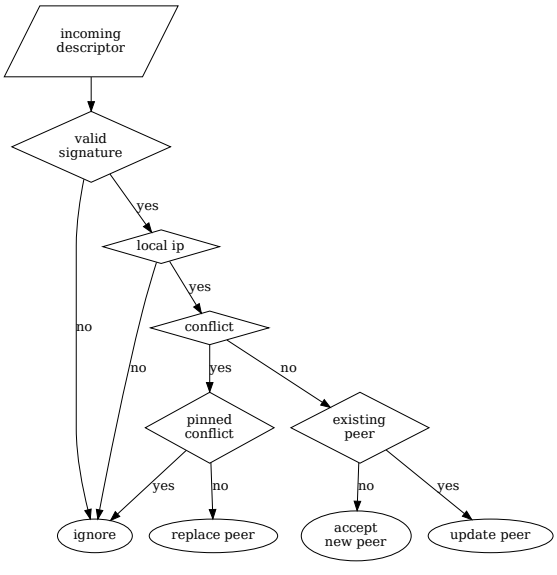

Figure 1: Incoming descriptor processing state engine

If the *vk* in a descriptor corresponds to a known peer and the descriptor is different from the latest descriptor previously seen from that peer, then the peer state is updated to reflect the new descriptor; otherwise, a new peer entry is created. The

peer state includes a list of all hostnames and IP addresses which each peer has ever announced, along with an *enabled* flag for each. Newly announced IP addresses are marked as *enabled* only if they are both in an acceptable subnet and they do not collide with any *pinned* peers' enabled IPs. Acceptable subnets are those which are both on the list of *allowed_subnets* and where the evaluating peer also has an IP bound at the time that the descriptor is first seen. Hostnames are likewise protected against collisions, and only accepted if they end with an allowed suffix on the *local_domains* list which by default is set to *.local*.

Collisions with unpinned peers' hostnames and IPs are governed by the *overwrite_unpinned* policy option; if it is set, then unpinned peers' can have their hostnames and IPs immediately disabled and reassigned to newly discovered peers. Unpinned peers are automatically removed from the database when they have not made an announcement in at least *expire_time* seconds, which defaults to 3600. Descriptors must have a valid from (*vf*) value that is smaller than the current time, and for already-known peers the value must be greater than the previous descriptor from that peer. If the descriptor sets the IP router flag, and the receiving peer processing it sees that the announced IP address matches the current default route, and the *accept_default_route* policy option is enabled, then the peer's *use_as_gateway* flag is set, which will cause *vula organize* to configure the remote peer as the receiving system's default route and adjust the peer's AllowedIPs value accordingly.

D. Protocol steps

0) Phase 0: Alice and Bob both start with three keypairs each. The Curve25519 keypair is used for the WireGuard peer identity for the Vula device, the CSIDH keypair is used to establish PSKs with other peers, and the Ed25519 keypair is used for signing Vula protocol messages.
1) Phase 1: Alice creates a protocol message ℵ which contains a cryptographic signature over the contained list of values as shown in Table II.
2) Phase 2: The ℵ value is used to construct a TXT record for the mDNS service associated with Alice's hostname and all records (A, SRV, and TXT) are published by Alice upon request.
3) Phase 3: Bob sends a query to the multicast address for the network and queries for the Vula service *_opabinia._udp.local*.
4) Phase 4: Bob receives Alice's ℵ protocol message and any other messages of participating hosts for *_opabinia._udp.local*.
   Bob verifies the ℵ descriptor is properly formatted and that the signature is valid. Bob will then process the ℵ protocol message according to a set of constraints. Any failure to meet the constraints as enumerated in Section IV-C will result in rejecting the Vula descriptor ℵ.
5) Phase 5: If the descriptor is not rejected, the Vula device will also be reconfigured and system routes added as needed. If there was a new IP address announced for an existing peer, it will become the new endpoint for the peer. Traffic to the new IP address will now be routed via the Vula device. If the peer is in the *pinned* state, traffic to its previous address or addresses will also continue to be routed via the Vula device.
6) Phase 6: optionally verify peers: The final step of the protocol is optional and highly recommended. To complete this phase of the protocol, the end-user may verify a peer's *vk* either manually or with a convenient-to use QR code. When the user has verified a peer, the peer's state is mutated to reflect that the peer is now both *pinned* and *verified*.

E. Implementation

We have implemented Vula in Python 3 for GNU/Linux. All of our software and changes to related software are Free Software, and are available at the anonymous site https://vula.link/.

Vula is separated into three distinct services: a publishing daemon, a discovery daemon, and a configuration daemon. We have implemented each of these daemons to have minimal attack surface. Each participating host must run all of the services listed here to properly use the Vula protocol with other hosts on their LAN segment.

Our initial implementation of Vula made use of three different CSIDH implementations depending on the platform where it would be used. We first started with the reference implementation [17], it is a generic C program which runs on systems with little-endian architecture and word size of 64 bits. We found it lacking in portability and in side-channel protections which is to be expected for a proof of concept implementation. The x86_64 implementation [18] claims to be constant-time and extremely fast, while the ARM64 [53] implementation is constant-time and extremely slow computing key derivation. The performance of each C implementation relied on specific CPU features which made portability extremely difficult. Additionally, each implementation had its own serialization formats which were incompatible. We later adopted a pure Python CSIDH implementation [4] [3]. While significantly slower than the reference or other implementations in C as mentioned in Section IV-I, the Python implementation allowed for supporting any CPU architecture where Python is available with a single implementation. Python does not require separate builds for each of our systems' CPUs (RISC-V, AMD64, POWER9, ARM64, ARM32, x86) and it provides memory safety. All available CSIDH implementations in C use unportable Intel or ARM assembly. Vula's handshake is not performance-sensitive and key derivation is cached for previously seen public keys. We additionally implemented serialization formats for CSIDH keypairs which should allow for greater interoperability. Bulk encryption of IP packets is handled efficiently by in-kernel WireGuard.

Slow key derivation may be a denial of service vector for embedded devices which decide to deploy a constant-time implementation over the reference implementation. We have extended each of the previously mentioned CSIDH implementations to include a basic tool for key generation and key derivation as well as shared secret generation. These tools are not currently used as Vula has chosen portability over performance at this time. After a shared secret is generated, we use a standard HKDF [57] construction to hash the secret value before use in any cryptographic context.

F. Multi-daemon systemd integration or monolithic mode

The Vula implementation operates by default in multi-daemon mode with `vula organize`, `vula discover`, and `vula publish` daemons. Multi-daemon mode includes systemd configuration files to run as several systemd services at install time. Each of the daemon services is run as a systemd service with minimal privileges, e.g.: as an unpriviledged user which has minimal access to the overall system. The services are grouped in a systemd slice called `vula.slice`. Each daemon follows the principle of least authority: each service has the minimum set of capabilities and permissions required to accomplish the specific tasks of the daemon. Further details about Vula systemd integration are available in Appendix C.

For systems that do not support systemd or for systems where only a single daemon is desired, the Vula implementation can also run all required services as a single process. The monolithic mode combines the `vula organize`, `vula discover`, and `vula publish` daemons into a single daemon, `vula`, which retains the superset of all other required daemon privileges which are normally compartmentalized away.

G. Vula peer tunnel considerations

During the `vula organize` daemon startup the local peer cache is loaded before new configuration information is accepted from the `discovery` daemon. After configuration of previously known peers, the `organize` daemon sits idle until a new descriptor is sent by the `discover` daemon or until another network event changes the system state. Key changes, IP address information, route updates, and interprocess communication from the command line interface are handled by this daemon.

The `vula` device is a normal WireGuard network interface which is entirely managed by the Vula `organize` process. This device has a single long-term identity which corresponds to the Curve25519 public key in the Vula descriptor announcements. Unlike normal usage of WireGuard, this key may be rotated at any time as long as the newly generated public key is announced to the local network or the descriptor is otherwise shared with Vula peers. WireGuard peers on the `vula` device always have a pre-shared key set. This key is derived from the CSIDH public key of the peer, and the CSIDH private key of the device owner. This key may also be rotated at any time as long as the new public key is also announced to the local network.

1) IP packet marking: IP packet marking is required to ensure that unencrypted packets are encrypted by the `vula` WireGuard device when appropriate as well as to mitigate routing loops of already encrypted packets. An important corner case with any point-to-point tunnel is to guarantee that packets which should be encrypted are encrypted. When a failure to encrypt happens and an unencrypted packet is sent over a device other than the VPN device, it is generally called a *bypass* or a *leak*. IP packet marking allows Vula to use WireGuard in a way that prevents this class of catastrophic failures that are common with point-to-point VPN software. Other VPN software that does not use IP packet marking suffers from catastrophic traffic bypass issues [77] which may be exploited by an adversary. One example where a bypass may occur is that WireGuard devices are configured with a peer at a given endpoint IP address, UDP port, and a list of AllowedIPs. Without IP packet marking, the endpoint address cannot be inside of any IP range in the AllowedIPs list unless AllowedIPs is 0.0.0.0/0, and with marking, desired traffic always traverses the Vula device, and it does not leak unencrypted IP packets.

H. Memorable and Secure: Pet-Names

Vula's network based discovery and publication is built on top of the trivially insecure mDNS protocol. Local active attackers are able to trivially forge responses to queries broadcast to the local network segment. It is for this reason that we turn select hostnames under the existing .local namespace into a secure petname [87] system.

Vula learns hostnames automatically as part of peer discovery. As currently implemented the Vula descriptor includes a .local hostname in its signed mDNS descriptor announcements. The signed .local hostnames in announcements from permanent peers are accumulated in a similar fashion as IP addresses already are: if a name is not already claimed, it will be added to the list of previously accepted names which that key has announced, all of the key's names resolve to the latest IP announced. By default, Vula scopes the name to only allow for claiming names under .local, or by a user setting a specific policy. This prevents an attacker from claiming a popular hostname while allowing them to claim a locally relevant hostname [4].

The Vula hosts file is used by a Name Service Switch (NSS) module [42] which requires reconfiguration of `/etc/nsswitch.conf`; our Vula implementation provides packages that perform this configuration automatically at package install time. Therefore, Vula provides protection of the authenticity of mDNS hostnames of participating Vula systems. The Vula vk is currently scoped to the hostname and only one vk may be the claimant of any single hostname, though in principle many hostnames is fine, none may conflict amongst all peers.

I. Post-Quantum considerations by the CSIDH

Several approaches have been proposed for enhancing WireGuard with regard to attacks from quantum computers. In Tiny WireGuard Tweak [8], the authors explain that to gain resistance to attacks by quantum computers, the Curve25519 public key used by WireGuard peers must be further concealed. The suggested enhancement is incompatible with Vula as the WireGuard public keys must be published with mDNS. We considered privacy improvements to mDNS and think this area is worth exploring in a future publication. However, absent privacy protections for mDNS service publications, we found the hiding of public keys to be impractical at this time.

In Post-quantum WireGuard [49], the authors proposed a post-quantum enhancement which effectively replaces the current WireGuard protocol with a post-quantum WireGuard

[4] After peer processing, the `vula organize` daemon writes a hosts file to disk in `/var/lib/vula-organize/hosts` which contains the current list of known hostnames and their respective IPv4 endpoints in classic `/etc/hosts` format.

protocol. Adopting this underlying protocol would add post-quantum protections for IP packets from attacks posed by universal quantum computers. The Post-Quantum WireGuard protocol has a great deal of promise. It additionally has practical implementation drawbacks for our envisoned deployment of Vula. Like WireGuard, it requires pre-configuration of peers by their public keys, and unlike WireGuard, it uses much larger public keys that do not easily fit in a single IP packet. The current implementation [48] is only available as a Linux kernel patch and it is incompatible with all other WireGuard implementations which makes cross platform support impractical.

One promising method to achieve post-quantum protection for traffic protected by the current WireGuard protocol is to set a per-peer pre-shared key. We had the idea to derive the PSKs by computation, using a different cryptosystem, rather than simply setting a symmetric key. If that system is a post-quantum key exchange, IP traffic will be further protected. We chose to use CSIDH for Vula to achieve transitional post-quantum security.

Each Vula peer announces their CSIDH-512 public key. This only adds 93 bytes to the mDNS service announcement when encoded as base64 and while accounting for DNS-SD overhead. All Vula descriptor data continues to fit into a single packet. Vula could rely on an architure specific C implementation where computing a PSK for a peer with CSIDH would take roughly 111 milliseconds on x86_64 [18]. However, we have chosen portability over performance. This choice results in signfigiantly longer computation time with a pure Python CSIDH implementation as shown in Table III and Table IV. Implementation details and platform specifics may dictate other constraints. Even with the pure Python CSIDH, we find it to be a negligible computational cost for potential protection against an adversary with a quantum computer.

Regarding the selection of CSIDH-512, CSIDH adds to the X25519 security already built into WireGuard. The purpose of this addition is to protect against the risk of quantum computers being built that are large enough to break X25519. Most post-quantum encryption options [70] are Key Encapsulation Mechanisms (KEMs), which need point-to-point communication, leaving CSIDH as the only practically deployable non-interactive key exchange (NIKE) choice compatible with broadcast channels. Current estimates to break CSIDH-512 with a quantum computer take around 2^{60} qubit operations, each of those costing as much as roughly 2^{40} bit operations.

The use of a post-quantum signature system such as SPHINCS+-128s [12] could replace the use of Ed25519 in the Vula protocol as long as the mDNS record size does not exceed 9000 bytes [21] split over multiple 1500 byte Ethernet frames if necessary. The SPHINCS+-128s signatures are 7856 bytes for 128-bit post-quantum security levels and the typical record size of a Vula key, value descriptor is around 300 bytes. We found that while the standard does allow larger record size, the underlying mDNS libraries we use did not. Furthermore, by using larger signatures we would move from a single 1500 byte packet for informing the entire local multicast group of a systems' descriptor to between five and six packets. It remains an open research question if such overhead is worthwhile absent an active attacker who possesses a universal quantum computer.

V. Performance

In this section we consider the performance of Vula's cryptographic choices. Post-quantum protections provided by CSIDH is explored in Section V-A. Bulk encryption is handled by WireGuard in-kernel, and we consider its performance in Section V-B. Additional measurements are available in Section B in the Appendix.

A. CSIDH performance evaluation

At peer discovery time, Vula uses CSIDH to generate a pairwise PSK. Each peer computes a shared key computation using its own secret key and the respective public key of each other peer. We show the performance time in Table III and in Table IV for five popular CPU architectures.

Table III: Python CSIDH-512 shared key computation execution time in seconds averaged over 128 runs

Arch	amd64	amd64
CPU	i7-9750H	Zen R1606G
Frequency	3.4Ghz	1.39Ghz
CSIDH	6.25	9.38

Arch	aarch64	ppc64le	riscv64
CPU	Cortex-A72	POWER9	rv64imafdc
Frequency	1.5Ghz	3.2GHz	1.5Ghz
CSIDH	26.16	20.01	130.324

Table IV: Python CSIDH-512 shared key computation execution time in seconds averaged over 128 runs

The performance of the pure python CSIDH leaves much to be desired and is an area in need of architecture-specific performance improvements. A pure C implementation that has speed as the only goal is around two orders of magnitude faster than the pure Python approach. A hybrid approach of extending the SIBC Python module with C for architecture specific operations is almost certainly an ideal compromise. Caching of CSIDH derived symmetric keys as well as background computation for the shared key computation additionally improve performance. We consider that CSIDH computations present a possible denial of service vector to Vula until native C bindings are developed.

B. Network performance evaluation

We examined the performance in both ideal lab conditions and in an actual home network deployment. Performance greatly varies by CPU architecture.

We observed that performance is not an issue with the underlying WireGuard transport. We found that WireGuard was able to sustain a consistent 1Gb/s in each direction using full duplex Ethernet devices as seen in Figure 2 using a FLENT [47], [36] TCP bidirectional measurement test. The solid green and solid orange lines represent the upload and download performance for IP traffic processed by WireGuard. The dotted green and dotted orange lines represent the upload and download performance for IP traffic without any protection from WireGuard on the same system. The latency of IP traffic is represented by the solid purple line for WireGuard and the dotted purple line is without any protection from WireGuard.

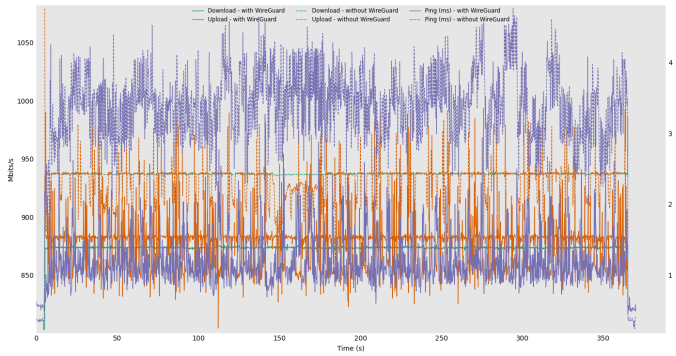

Figure 2: FLENT throughput and latency measurement

Notice that Figure 2 shows that IP traffic latency performance is sometimes *better* when IP packets are encrypted with WireGuard. This is surprising as we would expect packet processing to take a constant amount of time and for WireGuard encryption to incur an extra cost in addition; this is true and due to kernel scheduling, WireGuard packets appear to be processed faster in many cases under load.

Our primary test systems for this evaluation were an Intel NUC running Ubuntu 20.04 with an Intel i7-8705G CPU and an AMD Ryzen Embedded R1606G with Ubuntu 20.10. The NUC has an Intel I219-LM Gigabit Ethernet device and the AMD system uses an Intel I211 Gigabit Ethernet device. The switching fabric used is the prosumer Unifi Gigabit Ethernet by Ubiquiti. Latency is naturally increased as a side effect of sustained 2Gb/s traffic over time. When not under extreme network load, the latency is nearly indistinguishable.

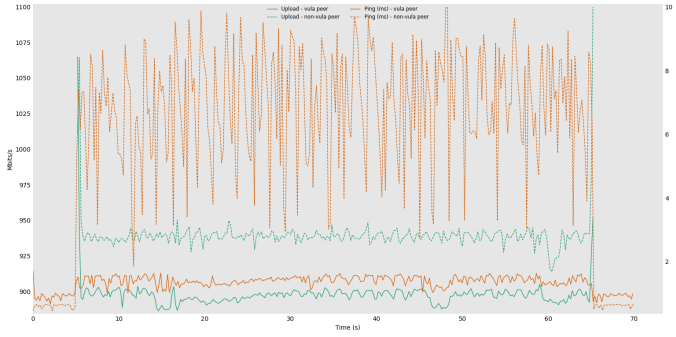

Figure 3: FLENT throughput and latency measurement

In Figure 3 we examine transmission of multiple flows with high performance switching equipment from Allies Telesis (x930 series; 48 1Gb/s ports) using two Dell PowerEdge R240 systems (Intel(R) Xeon(R) E-2124 CPU @ 3.30GHz with BCM5720 gigabit network card). We see that the throughput and latency for transmitted packets are again consistently lower and with significantly less variability. The solid green line represents traffic to another Vula peer and this traffic is protected by WireGuard. The dotted green line represents traffic to a non-Vula peer. The solid orange line shows latency with a Vula peer and the dotted orange line shows latency to a non-Vula system. The difference in total bytes of payload sent remains to be investigated, and may be related to maximum transmission unit (MTU) of the underlying Ethernet network.

VI. Security Evaluation

The security of IP traffic protected by Vula is provided by the WireGuard protocol as outlined in the 2017 NDSS paper [26], which relies on Curve25519, ChaCha20, Poly1305, and BLAKE2. It is further enhanced by setting the optional WireGuard peer-wise pre-shared symmetric key which the Vula protocol generates using CSIDH-512.

Vula's protection against active adversaries on descriptor announcements as described in Section IV-C is dependent on the security of Ed25519 signatures, and a specific order of operations as outlined in section VI-A through section VI-E.

A. Vula Security Goals

The Vula protocol aims to automatically protect IP traffic for the local ethernet segment with end-to-end encryption. When Alice wishes to transmit an IP packet to Bob, and this traffic would otherwise be sent directly on the local segment, Vula will configure the local system to automatically upgrade the security of the IP packets by sending them over the *Vula* WireGuard interface. The Vula interface does not have an IP address assigned to it; the IP packets have the same IP addresses inside and outside of the tunnel.

B. Data and Metadata

Users of wireless and wired Ethernet networks leave behind a number of unique data points. The Vula protocol seeks to reduce the data and metadata sent unencrypted overall. However, broadcast traffic such as mDNS data, including Vula announcements, any layer-two traffic such as ARP traffic with the MAC address [5] or addresses of each system, and traffic to hosts which are not using Vula, will be unencrypted as usual. Users of the Vula system will additionally generate a signed descriptor that may be verified by any third party.

C. Formal verification

We have proven that our model of the Vula protocol in Listing 1 in the Appendix is secure against both passive and active adversaries with Verifpal [54], [55], a symbolic formal verification tool. Verifpal has cryptographic constructions that make modeling protocols a straight-forward, easy to read, easy to understand exercise. Verifpal models form a basis for security and privacy-property-centric queries and thus proofs of protocol properties. Our Verifpal model captures the conditions and constraints expressed in Section III-D, and the Verifpal analysis confirms that the Vula protocol is secure in the passive attacker model without any public key verification, and that

[5] As it is understood that MAC addresses are used as a kind of covert communication channel about a systems's cryptographic state. An example is that the MAC address of a common router platform may be used as a lookup for its initial cryptographic state. Private correspondence regarding 0day BULLRUN-style [90] cryptographic "enabling" in ARM CPU configuration for a popular router brand. It also is understood that some adversaries collect MAC address information from drones and satellites in outer space [83] for at least geolocation reasons. As these MAC addresses are collected broadly, we encourage users to change the default MAC address at least once in the lifetime of their computer. This change should be inconsequential to everyone except saboteurs [13]. As this kind of sabotage has become known as *sigint enabling*, we consider counter actions to be *sigint disabling*.

it is secure in the active adversary model if the long term vk public keys are verified. The queries show that descriptor updates are fresh, signed, authenticated, and that long term secret keys stay confidential:

```
queries[
  freshness? sig_a_0
  freshness?  time_stamp_a_0
  authentication? Glenn -> Laura: sig_b_0
  confidentiality? ss_a
]
```

The first asks about the freshness of the signature from Alice. The second asks about the freshness of the *vf* timestamp. The third asks about the authentication of the signature from Alice. The last query asks about the confidentiality of the shared secret computed by Alice. Verifpal confirms all these (and similar queries for Bob) pass. Verifpal outputs:

```
Verifpal * All queries pass.
```

As always, formal analyses have limitations. Major risks that would not be ruled out by this analysis include the following: breaks in the WireGuard integration, cryptographic breaks in any of the assumed perfect cryptographic primitives, and/or any possible issues with Verifpal itself.

We have taken steps to secure the Vula traffic against active adversaries with our use of Ed25519. We have also taken steps to secure the Vula traffic against passive quantum adversaries who are able to record traffic and then later attack the recorded data's cryptography with their quantum computer. The protocol will need to be revised when quantum computers become available as Vula does not currently resist an active quantum adversary. Such an adversary should be able to forge Ed25519 signatures and would be able to publish new Vula descriptors to their advantage which would completely break Vula.

D. Active attacks against Vula

Here we discuss some attacks against systems on a local area network with and without Vula.

1) Address Resolution Protocol: Unrelated to the Vula protocol, the Address Resolution Protocol (ARP) allows for selectively targeting users by carrying out an ARP poisoning [76] attack. An attacker able to successfully ARP poison a target is able to place themselves into an on-path position. This attacker may delay, drop, or modify traffic depending on the protection available for any given IP packet.

With Vula in place, an on-path attacker is no longer able to modify traffic protected by Vula, adversaries will only be able to delay or drop encrypted packets. They may be able to interfere with mDNS and other unencrypted broadcast traffic.

2) Dynamic Host Configuration Protocol: Users who use the Dynamic Host Configuration Protocol (DHCP) to obtain an IP address automatically from the network are susceptible to DHCP related attacks. Attackers acting as a DHCP server may assign a targeted user any address on any subnet, and they may change the lease of any system on the network segment to a new IP address. References to previous IP address assignments would then be stale until a new Vula descriptor containing the new IP address is broadcast to the network.

3) MAC address vs IP address vs hostname security: Vula's use of petnames is important to security and paricipants should use the local names to address peers. To see the importance, consider either the ARP attack vector or the DHCP attack vector. Our Pet-Name system from Section IV-H is a required part of ensuring Vula's security claims in either case. Hostnames for Vula peers are cached locally into hostname, IP address pairs based on atomic processing of Vula peer descriptors. Additionally, and most critically: some IP addresses are routed via the Vula WireGuard device and some are not. If users use the Vula protected hostname, they will receive the latest IP that is routed through WireGuard for that respective Vula peer. This ensures that unless there is a valid WireGuard session, the packets will buffer, or drop before being sent or if there is a valid session but the peer has moved, they will be sent encrypted but no replies are expected.

The Vula Pet-Name system from Section IV-H does not prevent the DHCP attack where Alice is connecting to Bob's actual IP. Mallory published her key before she tricked Bob into moving there. However, if users understand that they should rely on names to connect to Bob's system rather than IP addresses when referring to Vula peers, then from Alice's perspective bob.local will continue to resolve to the last announcement from Bob which she considered valid. As Bob unbound his old IP when Mallory tricked him, Alice will not be able to reach him there, but the use of the name system has downgraded Mallory's attack from a confidentiality break to a mere Denial-of-Service. The same applies to the simplier ARP attack: as long as the packets first pass through the WireGuard device, they will be protected, and only by using hostnames is this guaranteed. This is why using secure names and having Vula perform the resolution is mandatory for Vula's security.

4) Traffic analysis: An attacker monitoring traffic as either an on-path or off-path attacker has the ability to perform traffic analysis such as website fingerprinting [46] or other traffic classification [6]. This capability may allow for targeted on-path selective blocking even when traffic is protected. WireGuard, and thus Vula, does not attempt to resist traffic analysis through timing obfuscation, padding, or other schemes such as generating dummy traffic or mixing.

5) Selective blocking: Attackers can selectively prevent certain packets, such as Vula announcements, from being delivered. This can prevent new peers from being automatically discovered, and can prevent existing peers from learning about each other's address changes, which can cause Denial-of-Service. Such an attack does not allow a breach of confidentiality between peers that are already *pinned* at the time of the attack because of the secure petname system.

6) Continuity of Verification Keys: We learn about peers and index them by their verification key. All other keys are considered peer-specific state and we allow rotation of those values. This means that the WireGuard and CSIDH public key may be rotated by issuing a new descriptor signed by the vk keypair. It is for this reason that the Verification Key must not change as it is the root of trust, and additionally, the hostname list, and the IP address list, are indexed under the Verification Key. No two peers may have overlapping hostnames or IP addresses.

7) Key substitution: Absent pinning as introduced in Section III-D, peers are *replaceable*, and active adversaries may announce their own descriptors, with conflicting resources, and with their own keys for two or more peers, enabling an active MITM attack. Peer substitution is possible against Vula peers which are not *pinned*, and new peers when they are making first contact. Using this attack, an active adversary can observe and bidirectionally forward IP packets between pairs of victims, or even in a single direction. This attack can be performed by an adversary with the capability to drop or replace targeted Vula peer mDNS announcements, and may be combined with other attacks such as ARP spoofing as mentioned in Section III-B. Adversaries are not able to breach confidentiality for *pinned* peers. Peers in the replaceable state are only secure against passive adversaries, and peers in the pinned state are secure against active adversaries from peer replacement attacks.

8) Further protection against active adversaries: Vula makes some trade-offs by default which may be adjusted according to a specific deployment's desired security properties. All options are implemented in our Python reference implementation. Users of Vula concerned about active adversaries must configure Vula for their use case. By default, Vula does not require users to be aware of the software or any configuration option after installation. As mentioned in the peer replacement attack section, Vula peers are replaceable by default and this is only safe against a completely passive adversary. In the unpinned, replaceable state, peers expire, and any conflicting peer descriptor will simply replace the original peer entirely. Expiration of peers ensures that non-Vula systems will retain connectivity in the event of IP address reuse. In the pinned, permanent state, a user must resolve any resource conflicts manually, and the first peer descriptor to arrive and be pinned will always remain during automatic descriptor conflict resolution. To accommodate users who are unable or unwilling to manually resolve IP address conflicts, Vula defaults to all peers starting in a replaceable state. Changing the default is straightforward at install or run time. The *pin_new_peers* configuration option default state is *false*. The benefit is that normally-occurring IP address or hostname collisions will be handled normally as if Vula were not in use; the disadvantage is that active attackers are able to replace peers, and are *not* thwarted even while being detectable. This default configuration is intended to be suitable for all deployments without requiring any user awareness of Vula at all. For protection against active attackers who are not present before first contact, a knowledgeable user can choose pin automatically by setting *pin_new_peers* to *true*. This setting means that naturally-occurring IP address or hostname collisions will sometimes lead to an inability to communicate with affected Vula or non-Vula hosts. Pinned hosts may always update their own resources, and their descriptors must not conflict with any other pinned Vula peers or it will be rejected entirely. In any case, a user may always pin or verify a peer regardless of defaults, and they may set their own defaults at install time.

E. Adversary evaluation

We consider the adversary definitions from Section III and their respective attack vectors. As described in Section III-B all layer-two traffic remains unprotected such as ARP as well as layer-three IP broadcast traffic. Any active adversary as described in Section III may delay, drop, and/or store any traffic where they have successfully performed an *ARP poisoning* attack. When operating with Vula, nearly all intranet traffic to participating systems will be protected in a forward-secret manner which defeats passive adversaries automatically. Using a trusted third party would allow for automatic trust decisions, but there is no suitable trusted third party in the context of every LAN, and across organizational boundaries.

1) Unilateral Surveillance: Vula completely defeats the **Unilateral Surveillance Adversary** in a forward-secret manner that is not dependent on a wireless passphrase. Thus, regardless if there is wireless encryption or wireless passphrase rotation policies, Vula successfully defeats the **Unilateral Surveillance Adversary** for the types of traffic which are protected by Vula. When the router deploys Vula, *all* traffic to the internet may be protected from interception in the local LAN context.

2) End User: Vula *partially* defeats the **End User** adversary for the types of the traffic which are protected by Vula. It reduces the adversary capabilities to denial-of-service as they are only able to delay or drop Vula protected traffic. Recording of the traffic is now largely useless thanks to the forward secrecy provided by the underlying WireGuard transport. The protection is bound by time of adversary arrival, default peer pinning status, and by peer verification status. If the **End User** adversary arrives before some other users, the adversary is able to claim possession of any currently unclaimed IP addresses, including addresses which may conflict with DHCP leases of newly arriving users. This would allow the adversary to be a Vula peer for an IP address assigned by DHCP to another user, thus impacting all subsequent users on the network. Active attacks by long-term adversaries can only be detected and defeated by manual key verification.

3) Network Operator: Vula successfully defeats the **Network Operator** adversary for the majority of the traffic which is protected by Vula. Intra-network traffic is protected between any set of systems deploying Vula, and peers should be pinned, as well as verified.

If the router also deploys Vula, *all* traffic to the internet is protected from interception in the local LAN context. The **Network Operator** is still able to monitor it on the gateway itself. Using an additional protection mechanism such as a layered VPN may reduce the adversary capabilities to delaying or dropping traffic which is destined for the internet.

VII. Conclusions

Vula enhances the confidentiality, integrity, and authenticity of IP traffic which is routed to or through another Vula peer. The cryptographic overhead with regard to performance for Gigabit networks is negligible. Without any configuration, or even any user awareness that Vula exists on their system, the protection provided by Vula completely defeats a passive adversary, and active adversaries arriving after the first contact with automatic pinning. With user awareness and peer verification, all active adversaries are also defeated until such a time in which they have the ability to forge Ed25519 signatures, e.g. with access to an universal quantum computer. Our implementation has been released anonymously [96] as Free Software available for deployment, and is now deployed on our home and other production networks.

References

[1] A. Abdelrahman, H. Khaled, E. Shaaban, and W. S. Elkilani, "WPA-WPA2 PSK cracking implementation on parallel platforms," in *2018 13th International Conference on Computer Engineering and Systems (ICCES)*. IEEE, 2018, pp. 448–453.

[2] B. D. Aboba, J. Malinen, P. Congdon, J. A. Salowey, and M. Jones, "RADIUS Attributes for IEEE 802 Networks," RFC 7268, Jul. 2014, https://rfc-editor.org/rfc/rfc7268.txt.

[3] G. Adj, J.-J. Chi-Domínguez, and F. Rodríguez-Henríquez, "On new Vélu's formulae and their applications to CSIDH and B-SIDH constant-time implementations," IACR Cryptol. ePrint Arch., vol. 2020/1109, 2020, https://eprint.iacr.org/2020/1109.

[4] ——, "SIBC Python library," https://github.com/JJChiDguez/sibc/, 2021.

[5] J. Appelbaum, A. Gibson, J. Goetz, V. Kabisch, L. Kampf, and L. Ryge, "NSA targets the privacy-conscious," 07 2014, https://daserste.ndr.de/panorama/aktuell/nsa230_page-1.html.

[6] ——, "NSA XKeyscore source code," 07 2014, https://daserste.ndr.de/panorama/xkeyscorerules100.txt.

[7] J. Appelbaum, A. Gibson, C. Grothoff, A. Müller-Maguhn, L. Poitras, M. Sontheimer, and C. Stöcker, "Inside the NSA's War on Internet Security," https://www.spiegel.de/international/germany/inside-the-nsa-s-war-on-internet-security-a-1010361.html, 12 2014.

[8] J. Appelbaum, C. Martindale, and P. Wu, "Tiny WireGuard Tweak," in *Progress in Cryptology - AFRICACRYPT 2019 - 11th International Conference on Cryptology in Africa, Rabat, Morocco, July 9-11, 2019, Proceedings*, ser. Lecture Notes in Computer Science, J. Buchmann, A. Nitaj, and T. Rachidi, Eds., vol. 11627. Springer, 2019, pp. 3–20, https://doi.org/10.1007/978-3-030-23696-0_1.

[9] J. Appelbaum, L. Poitras, M. Rosenbach, C. Stöcker, J. Schindler, and H. Stark, "Documents reveal top NSA hacking unit," December 2013, https://tinyurl.com/4ewxxejj.

[10] D. J. Bernstein, "Curve25519: new diffie-hellman speed records," in *Public Key Cryptography-PKC 2006*. Springer, 2006, pp. 207–228.

[11] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B. Yang, "High-Speed High-Security Signatures," in *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, ser. Lecture Notes in Computer Science, B. Preneel and T. Takagi, Eds., vol. 6917. Springer, 2011, pp. 124–142, https://doi.org/10.1007/978-3-642-23951-9_9.

[12] D. J. Bernstein, A. Hülsing, S. Kölbl, R. Niederhagen, J. Rijneveld, and P. Schwabe, "The SPHINCS+ Signature Framework," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 2129–2146, https://doi.org/10.1145/3319535.3363229.

[13] D. J. Bernstein, T. Lange, and R. Niederhagen, "Dual EC: A standardized back door," in *The New Codebreakers - Essays Dedicated to David Kahn on the Occasion of His 85th Birthday*, ser. Lecture Notes in Computer Science, P. Y. A. Ryan, D. Naccache, and J. Quisquater, Eds., vol. 9100. Springer, 2016, pp. 256–281, https://doi.org/10.1007/978-3-662-49301-4_17.

[14] A. Bittau, D. B. Giffin, M. J. Handley, D. Mazieres, Q. Slack, and E. W. Smith, "Cryptographic Protection of TCP Streams (tcpcrypt)," RFC 8548, May 2019, https://rfc-editor.org/rfc/rfc8548.txt.

[15] D. Campbell, "London Internet Exchange members vote no to constitution tweak; Peering peers reject proposed rules on keeping quiet about secret govt gagging orders," 2017, https://www.theregister.com/2017/02/22/linx_members_vote_to_block/.

[16] W. Castryck, T. Lange, C. Martindale, L. Panny, and J. Renes, "CSIDH: an efficient post-quantum commutative group action," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2018, pp. 395–427.

[17] ——, "CSIDH reference implementation source code," 2018, https://yx7.cc/code/csidh/csidh-latest.tar.xz.

[18] D. Cervantes-Vázquez, M. Chenu, J. Chi-Domínguez, L. D. Feo, F. Rodríguez-Henríquez, and B. Smith, "Stronger and faster side-channel protections for CSIDH," in *Progress in Cryptology - LATINCRYPT 2019 - 6th International Conference on Cryptology and Information Security in Latin America, Santiago de Chile, Chile, October 2-4, 2019, Proceedings*, ser. Lecture Notes in Computer Science, P. Schwabe and N. Thériault, Eds., vol. 11774. Springer, 2019, pp. 173–193, https://doi.org/10.1007/978-3-030-30530-7_9.

[19] S. Cheshire, B. Aboba, and E. Guttman, "Dynamic Configuration of IPv4 Link-Local Addresses," RFC 3927, May 2005, https://rfc-editor.org/rfc/rfc3927.txt.

[20] S. Cheshire and M. Krochmal, "DNS-Based Service Discovery," RFC 6763, Feb. 2013, https://rfc-editor.org/rfc/rfc6763.txt.

[21] ——, "Multicast DNS," RFC 6762, Feb. 2013, https://rfc-editor.org/rfc/rfc6762.txt.

[22] J. Y. Cho and A. Sergeev, "Post-Quantum MACsec Key Agreement for Ethernet Networks," in *Proceedings of the 15th International Conference on Availability, Reliability and Security*, ser. ARES '20. New York, NY, USA: Association for Computing Machinery, 2020, https://dl.acm.org/doi/10.1145/3407023.3409220.

[23] D. Crocker, "DNS Attrleaf Changes: Fixing Specifications That Use Underscored Node Names," RFC 8553, Mar. 2019, https://rfc-editor.org/rfc/rfc8553.txt.

[24] W. Diffie, "Securing Networks: End-to-End Encryption vs. Link Encryption and Trusted Systems," in *Proceedings of the 1983 IEEE Symposium on Security and Privacy, Oakland, California, USA, April 25-27, 1983*. IEEE Computer Society, 1983, pp. 136–138, https://doi.org/10.1109/SP.1983.10021.

[25] D. Dolev and A. C. Yao, "On the security of public key protocols," *IEEE Trans. Information Theory*, vol. 29, no. 2, pp. 198–208, 1983.

[26] J. A. Donenfeld, "Wireguard: Next generation kernel network tunnel," in *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society, 2017, https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/wireguard-next-generation-kernel-network-tunnel/.

[27] R. Droms, "Dynamic Host Configuration Protocol," RFC 2131, Mar. 1997, https://rfc-editor.org/rfc/rfc2131.txt.

[28] P.-E. Eriksson and B. Odenhammar, "VDSL2: Next important broadband technology," *Ericsson Review*, vol. 1, pp. 36–47, 2006.

[29] European Court of Human Rights (ECHR), "Klass and Others v. Germany," p. 28, 1978.

[30] ——, "Malone v. United Kingdom," p. 14, 1984.

[31] ——, "Leander v. Sweden," p. 433, 1987.

[32] ——, "Weber and Saravia v. Germany," p. 1173, 2006.

[33] ——, "Kennedy v. United Kingdom," p. 682, 2010.

[34] ——, "Roman Zakharov v. Russia," p. 1065, 2015.

[35] ——, "Szabó and Vissy v. Hungary," p. 579, 2016.

[36] Flent Authors, "Flent source code and web page," https://flent.org/, 2017.

[37] J. Font, "An open source, self-hosted implementation of the Tailscale control server," https://github.com/juanfont/headscale, 2020.

[38] S. Frankel and S. Krishnan, "IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap," RFC 6071, Feb. 2011, https://rfc-editor.org/rfc/rfc6071.txt.

[39] R. Gallagher, "Inside Menwith Hill: The NSA's British Base at the Heart of U.S. Targeted Killing," 2016, https://theintercept.com/2016/09/06/nsa-menwith-hill-targeted-killing-surveillance/.

[40] P. Geissle, "upc keys," 2016, https://haxx.in/upc_keys.c.

[41] J. Gilmore, "Re: [Cryptography] Opening Discussion: Speculation on "BULLRUN"," https://www.mail-archive.com/cryptography@metzdowd.com/msg12325.html, 2013.

[42] GNU Project, "glibc: System Databases and Name Service Switch," 1992-2020, https://www.gnu.org/software/libc/manual/html_node/Name-Service-Switch.html.

[43] T. Goodspeed, S. Bratus, R. Melgares, R. Shapiro, and R. Speers, "Packets in Packets: Orson Welles' In-Band Signaling Attacks for Modern Radios," in *5th USENIX Workshop on Offensive Technologies, WOOT'11, August 8, 2011, San Francisco, CA, USA, Proceedings*, D. Brumley and M. Zalewski, Eds. USENIX Association, 2011, pp. 54–61, http://static.usenix.org/event/woot11/tech/final_files/Goodspeed.pdf.

[44] C. Grothoff, M. Wachs, M. Ermert, and J. Appelbaum, "Toward secure name resolution on the internet," *Computers & Security*, vol. 77, pp. 694–708, 2018.

[45] C. Grothoff, M. Wachs, M. Ermert, J. Appelbaum, D. Larousserie, Y. Eudes, and L. Poitras, "MoreCowBells: Nouvelles révélations sur les pratiques de la NSA," *Le Monde*, no. 24.1.2015, January 2015.

[46] D. Herrmann, R. Wendolsky, and H. Federrath, "Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier," in *Proceedings of the 2009 ACM workshop on Cloud computing security (CCSW '09)*. New York, NY, USA: ACM, October 2009, pp. 31–42, http://epub.uni-regensburg.de/11919/1/authorsversion-ccsw09.pdf.

[47] T. Høiland-Jørgensen, C. A. Grazia, P. Hurtig, and A. Brunstrom, "Flent: The flexible network tester," in *Proceedings of the 11th EAI International Conference on Performance Evaluation Methodologies and Tools*, 2017, pp. 120–125.

[48] A. Hülsing, K. Ning, P. Schwabe, F. Weber, and P. R. Zimmermann, "Post-quantum WireGuard source code," 2020, https://cryptojedi.org/crypto/data/pqwireguard-20200402.tar.bz2.

[49] ——, "Post-Quantum WireGuard," in *2021 IEEE Symposium on Security and Privacy (S&P)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2021, pp. 511–528, https://doi.ieeecomputersociety.org/10.1109/SP40001.2021.00030.

[50] IEEE, "IEEE Standard for Local and Metropolitan Area Networks: Media Access Control (MAC) Security," *IEEE Std 802.1AE-2006*, pp. 1–150, 2006.

[51] IEEE, "IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 4: Protected Management Frames," *IEEE Std 802.11w-2009 (Amendment to IEEE Std 802.11-2007 as amended by IEEE Std 802.11k-2008, IEEE Std 802.11r-2008, and IEEE Std 802.11y-2008)*, pp. 1–111, 2009.

[52] ISO Central Secretary, "Information technology - Automatic identification and data capture techniques - QR Code bar code symbology specification," International Organization for Standardization, Geneva, CH, Standard ISO/IEC TR 18004:2015, 2016, https://www.iso.org/standard/62021.html.

[53] A. Jalali, R. Azarderakhsh, M. M. Kermani, and D. Jao, "Towards optimized and constant-time CSIDH on embedded devices," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2019, pp. 215–231.

[54] N. Kobeissi, "Verifpal: Cryptographic Protocol Analysis for Students and Engineers," *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 971, 2019, https://eprint.iacr.org/2019/971.

[55] N. Kobeissi, G. Nicolas, and M. Tiwari, "Verifpal: Cryptographic Protocol Analysis for the Real World," in *Progress in Cryptology - INDOCRYPT 2020 - 21st International Conference on Cryptology in India, Bangalore, India, December 13-16, 2020, Proceedings*, ser. Lecture Notes in Computer Science, K. Bhargavan, E. Oswald, and M. Prabhakaran, Eds., vol. 12578. Springer, 2020, pp. 151–202, https://doi.org/10.1007/978-3-030-65277-7_8.

[56] P. Kotler, "The prosumer movement," in *Prosumer Revisited*. Springer, 2010, pp. 51–60.

[57] H. Krawczyk and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)," IBM Research, IETF Internet Draft – RFC5869, May 2010, https://tools.ietf.org/html/rfc5869.

[58] K. Landefeld, "G10, BND-Gesetz und der effektive Schutz vor Grundrechten," 2018, https://fahrplan.events.ccc.de/congress/2018/Fahrplan/events/10016.html.

[59] A. Langley, "Opportunistic encryption everywhere," *In W2SP*, 2009, https://www.ieee-security.org/TC/W2SP/2009/papers/s1p2.pdf.

[60] Linux man-pages project, *capabilities(7) - Linux manual page*, 7 2020.

[61] E. N. Lorente, C. Meijer, and R. Verdult, "Scrutinizing WPA2 password generating algorithms in wireless routers," in *9th USENIX Workshop on Offensive Technologies (WOOT-15)*, 2015.

[62] A. T. Markettos, C. Rothwell, B. F. Gutstein, A. Pearce, P. G. Neumann, S. W. Moore, and R. N. M. Watson, "Thunderclap: Exploring Vulnerabilities in Operating System IOMMU Protection via DMA from Untrustworthy Peripherals," in *Proceedings of the Network and Distributed Systems Security Symposium (NDSS)*, 2 2019.

[63] M. Marlinspike, "Cloud crack," 2012, https://web.archive.org/web/20160319232206/https://www.cloudcracker.com/.

[64] A. Meister, "How the German Foreign Intelligence Agency BND tapped the Internet Exchange Point DE-CIX in Frankfurt, since 2009," 2015, https://shorturl.at/gmrMU.

[65] A. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996, http://cacr.uwaterloo.ca/hac/.

[66] M. S. Miller, "Robust Composition: Towards a Unified Approach to Access Control and Concurrency Control," Ph.D. dissertation, Johns Hopkins University, Baltimore, Maryland, USA, May 2006.

[67] P. Mockapetris, "Domain names - implementation and specification," RFC 1035, Nov. 1987, https://rfc-editor.org/rfc/rfc1035.txt.

[68] R. Moskowitz, D. Karrenberg, Y. Rekhter, E. Lear, and G. J. de Groot, "Address Allocation for Private Internets," RFC 1918, Feb. 1996, https://rfc-editor.org/rfc/rfc1918.txt.

[69] S. J. Murdoch and G. Danezis, "Low-cost traffic analysis of Tor," in *2005 IEEE Symposium on Security and Privacy (S&P'05)*. IEEE, 2005, pp. 183–195.

[70] NIST, "Post-Quantum Round 3 Submissions," https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions, 2021.

[71] G. Nomikos and X. A. Dimitropoulos, "traixroute: Detecting ixps in traceroute paths," in *Passive and Active Measurement - 17th International Conference, PAM 2016, Heraklion, Greece, March 31 - April 1, 2016. Proceedings*, ser. Lecture Notes in Computer Science, T. Karagiannis and X. A. Dimitropoulos, Eds., vol. 9631. Springer, 2016, pp. 346–358, https://doi.org/10.1007/978-3-319-30505-9_26.

[72] E. C. of Human Rights, "Big Brother Watch and others v. the United Kingdom," Sep 2018, applications nos. 58170/13, 62322/14 and 24960/15 and http://hudoc.echr.coe.int/eng?i=001-186048.

[73] T. Perrin, "The Noise Protocol Framework," 2018, http://www.noiseprotocol.org/noise.html.

[74] D. C. Plummer, "An Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware," RFC 826, Nov. 1982, https://rfc-editor.org/rfc/rfc826.txt.

[75] L. Poitras, M. Rosenbach, and H. Stark, "How GCHQ Monitors Germany, Israel and the EU," https://tinyurl.com/2krrdswx, 12 2013.

[76] V. Ramachandran and S. Nandi, "Detecting ARP spoofing: An active technique," in *Information Systems Security, First International Conference, ICISS 2005, Kolkata, India, December 19-21, 2005, Proceedings*, ser. Lecture Notes in Computer Science, S. Jajodia and C. Mazumdar, Eds., vol. 3803. Springer, 2005, pp. 239–250, https://doi.org/10.1007/11593980_18.

[77] M. Ray, J. Appelbaum, K. Koscher, and I. Finder, "vpwns: Virtual Pwned Networks," in *2nd USENIX Workshop on Free and Open Communications on the Internet, FOCI '12, Bellevue, WA, USA, August 6, 2012*, R. Dingledine and J. Wright, Eds. USENIX Association, 2012, https://www.usenix.org/conference/foci12/workshop-program/presentation/appelbaum.

[78] B. I. Reddy and V. Srikanth, "Review on wireless security protocols (WEP, WPA, WPA2 & WPA3)," *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 2019.

[79] D. J. Rice, "DOCSIS 3.1® technology and hybrid fiber coax for multi-Gbps broadband," in *2015 Optical Fiber Communications Conference and Exhibition (OFC)*. IEEE, 2015, pp. 1–4.

[80] G. Ritzer and N. Jurgenson, "Production, consumption, prosumption: The nature of capitalism in the age of the digital 'prosumer'," *Journal of consumer culture*, vol. 10, no. 1, pp. 13–36, 2010.

[81] S. Rose, M. Larson, D. Massey, R. Austein, and R. Arends, "DNS Security Introduction and Requirements," RFC 4033, Mar. 2005, https://rfc-editor.org/rfc/rfc4033.txt.

[82] B. Ruytenberg, "Breaking Thunderbolt Protocol Security: Vulnerability Report," 2020, https://thunderspy.io/assets/docs/breaking-thunderbolt-security-bjorn-ruytenberg-20200417.pdf.

[83] L. Ryge, "add wifi geolocation from space to the list of things that once sounded crazy but actually happens /ty," 2016, https://nitter.net/wiretapped/status/773136872323317760#m.

[84] C. Savage, 12 2020, https://www.nytimes.com/2020/12/03/us/politics/section-215-patriot-act.html.

[85] J. M. Schanck, W. Whyte, and Z. Zhang, "Circuit-extension handshakes for tor achieving forward secrecy in a quantum world," *Proc. Priv. Enhancing Technol.*, vol. 2016, no. 4, pp. 219–236, 2016, https://doi.org/10.1515/popets-2016-0037.

[86] Slack Inc, "Nebula," https://github.com/slackhq/nebula, 11 2019.

[87] M. Stiegler, "An introduction to petname systems," *Advances in Financial Cryptography*, 2005, https://www.financialcryptography.com/mt/archives/000499.html.

[88] C. Stöcker, "GCHQ Surveillance: The Power of Britain's Data Vacuum," https://tinyurl.com/4azdtvc8, 7 2013.

[89] Tailscale, "Tailscale," https://tailscale.com/, 2020.

[90] The Guardian, "How US and UK spy agencies defeat internet privacy and security," http://www.theguardian.com/world/2013/sep/05/nsa-gchq-encryption-codes-security, 2013.

[91] Tonari Inc, "innernet," https://github.com/tonarino/innernet, 2021.

[92] M.-T. Tran, T.-T. Nguyen, and I. Echizen, "Pool-Based APROB Channel to Provide Resistance against Global Active Adversary under Probabilistic Real-Time Condition," in *2008 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, vol. 2, 2008, pp. 257–263.

[93] M. Vanhoef and E. Ronen, "Dragonblood: Analyzing the Dragonfly Handshake of WPA3 and EAP-pwd," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 517–533.

[94] S. Viehböck, "Brute forcing Wi-Fi Protected Setup," 2011, https://sviehb.files.wordpress.com/2011/12/viehboeck_wps.pdf.

[95] J. Vollbrecht, J. D. Carlson, L. Blunk, D. B. D. Aboba, and H. Levkowetz, "Extensible Authentication Protocol (EAP)," RFC 3748, Jun. 2004, https://rfc-editor.org/rfc/rfc3748.txt.

[96] Vula Authors, "Vula: automatic local area network encryption," https://vula.link, 2021.

[97] B. Wagner and P. Mindus, "Multistakeholder Governance and Nodal Authority–Understanding Internet Exchange Points," NoC Internet Governance Research Project: Case Studies, Case Study 7, 2015, https://publixphere.net/i/noc/page/IG_Case_Study_Multistakeholder_Governance_and_Nodal_Authority_Understanding_Internet_Exchange_Points.html.

[98] D. Wendlandt, D. G. Andersen, and A. Perrig, "*Perspectives:* improving ssh-style host authentication with multi-path probing," in *2008 USENIX Annual Technical Conference, Boston, MA, USA, June 22-27, 2008. Proceedings*, R. Isaacs and Y. Zhou, Eds. USENIX Association, 2008, pp. 321–334, http://www.usenix.org/events/usenix08/tech/full_papers/wendlandt/wendlandt.pdf.

[99] K. Wierenga, S. Winter, and T. Wolniewicz, "The eduroam Architecture for Network Roaming," RFC 7593, Sep. 2015, https://rfc-editor.org/rfc/rfc7593.txt.

[100] P. Wouters, "History and implementation status of Opportunistic Encryption for IPsec," https://tinyurl.com/s3dp7z98, 2013.

APPENDIX

A. Verifpal verification

The following listing models the Vula protocol and shows our security queries. For an executable version see our anonymous page [96].

```
attacker[active]

principal Laura[
      knows public _hkdf_salt
      knows public _hkdf_info
      generates time_stamp_a_0
      knows private vk_a
      vk_a_pk = G^vk_a
      generates csidh_a
      csidh_a_pk = G^csidh_a
      descriptor_a_pt0 = CONCAT(
          time_stamp_a_0, csidh_a_pk)
      ha_0 = HASH(descriptor_a_pt0)
      sig_a_0 = SIGN(vk_a, ha_0)
]

principal Glenn[
      knows public _hkdf_salt
      knows public _hkdf_info
      generates time_stamp_b_0
      knows private vk_b
      vk_b_pk = G^vk_b
      generates csidh_b
      csidh_b_pk = G^csidh_b
      descriptor_b_pt0 = CONCAT(
          time_stamp_b_0, csidh_b_pk)
      hb_0 = HASH(descriptor_b_pt0)
      sig_b_0 = SIGN(vk_b, hb_0)
]

Laura -> Glenn: [vk_a_pk], time_stamp_a_0,
    csidh_a_pk, sig_a_0

Glenn -> Laura: [vk_b_pk], time_stamp_b_0,
    csidh_b_pk, sig_b_0

principal Laura[
      x_0 = SIGNVERIF(vk_b_pk, HASH(CONCAT
          (time_stamp_b_0, csidh_b_pk)),
          sig_b_0)?
      ss_a = HKDF(_hkdf_salt, HASH(
          csidh_b_pk^csidh_a), _hkdf_info)
]

principal Glenn[
      y_0 = SIGNVERIF(vk_a_pk, HASH(CONCAT
          (time_stamp_a_0, csidh_a_pk)),
          sig_a_0)?
      ss_b = HKDF(_hkdf_salt, HASH(
          csidh_a_pk^csidh_b), _hkdf_info)
]

queries[
      freshness? sig_a_0
      freshness? sig_b_0
      freshness? time_stamp_a_0
      freshness? time_stamp_b_0
      authentication? Glenn -> Laura:
          sig_b_0
      authentication? Laura -> Glenn:
          sig_a_0
      confidentiality? ss_a
      confidentiality? ss_b
]
```

Listing 1: "Verifpal Vula model protocol"

B. Additional FLENT performance graphs

The following graphs show some of the performance characteristics with different CPU architectures, and microarchitectures. The solid green and solid orange lines represent the upload and download performance for IP traffic processed by WireGuard. The dotted green and dotted orange lines represent the upload and download performance for IP traffic without any protection from WireGuard on the same system. The latency of IP traffic is represented by the solid purple line for WireGuard and the dotted purple line is without any protection from WireGuard.

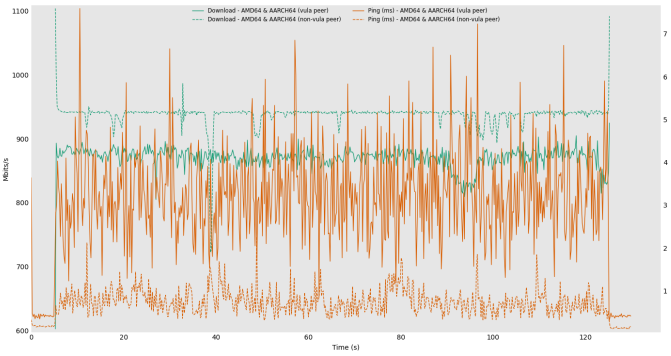

Figure 4: FLENT 12 stream down with ping; graph with and without WireGuard. AMD64 and AARCH64.

In Figure 4 we see the performance of a twelve stream iperf3 test with and without WireGuard between an AMD64 machine and an AARCH64 (ARM64) machine. The performance for gigabit traffic is as expected and fills roughly all available bandwidth modulo measurement noise.

In Figure 5 we see the performance of a twelve stream iperf3 test with and without WireGuard between an AMD64 machine and an RISC-V machine. The RISC-V machine has performance issues. It is not even able to sustain a full gigabit of traffic without WireGuard. Adding WireGuard shows a steady 200Mb/s which indicates that the RISC-V platform would greatly benefit from an optimized WireGuard implementation. That WireGuad works everywhere that Linux works helps with deployment and performance improvements may be made as needed for each CPU architecture.

In Figure 6 we see the performance of a twelve stream iperf3 test with and without WireGuard between an AMD64 (Intel i7) machine and an AMD64 (zen) machine. The performance difference between these two microarchitectures is

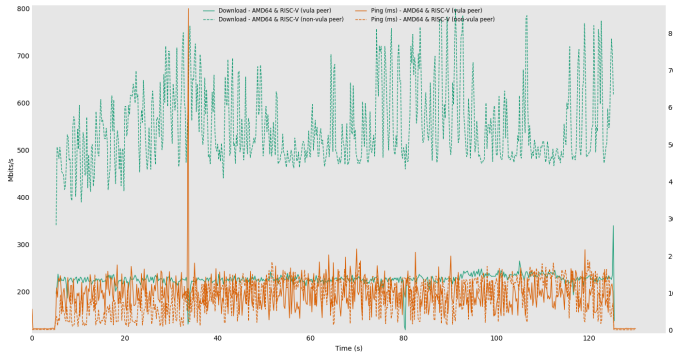

Figure 5: FLENT 12 stream down with ping; graph with and without WireGuard. AMD64 and RISC-V.

nominal, and while improvements may be useful for speeds in excess of one gigabit, they are suitable for full gigabit saturation. The AMD64 architecture is extremely common and many home users likely have only AMD64 machines as their laptop or desktop endpoints.

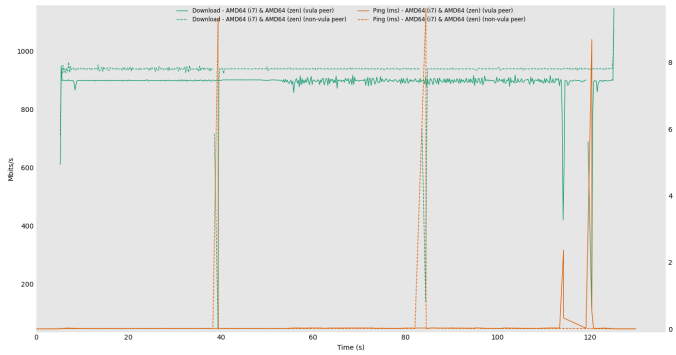

Figure 6: FLENT 12 stream down with ping; graph with and without WireGuard. AMD64 (i7) and AMD64 (zen).

In Figure 7 we see the performance of a twelve stream iperf3 test with and without WireGuard between a POWER9 machine and an AMD64 machine. The performance for gigabit traffic is as expected and fills roughly all available bandwidth modulo measurement noise.

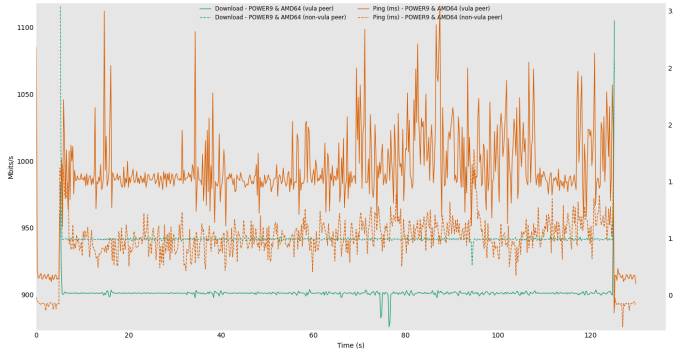

Figure 7: FLENT 12 stream down with ping; graph with and without WireGuard. POWER9 and AMD64.

The performance characteristics clearly show the benefits of architecture specific optimization. WireGuard is able to saturate gigabit Ethernet connections bidirectionally when two peers use modern AMD64 CPUs. WireGuard performance on more esoteric or otherwise new CPU architectures leaves something to be desired by comparison to optimized versions of itself on other platforms.

C. *systemd integration details*

Vula is integrated into the system as multiple daemons managed by systemd.

1) vula.slice: The `vula.slice` limits memory and other resources to ensure that none of the daemons run with systemd are able to consume excessive resources. All Vula daemons are a part of the `vula.slice`.

2) Discovery daemon: `vula-discover.service` runs the `vula discover` daemon which monitors for Vula publications on the local network. It runs as user `vula-discover` and as group `vula`. It requires access to the local network segment.

`vula discover` listens for mDNS service announcements under the DNS-SD label of `_opabinia._udp.local.` and it outputs each discovered mDNS WireGuard service. The output is a peer descriptor contained in a single line for each discovered WireGuard service. The peer descriptor contains all the information needed to reach and configure the newly discovered WireGuard peer. For each Vula service detected, it constructs a descriptor which is then sent to the `vula organize` daemon.

3) Publish daemon: `vula-publish.service` runs the `vula publish` daemon and publishes the mDNS Service record on the local network. It runs as user `vula-publish` and as group `vula`.

`vula publish` is a standalone mDNS service announcer which does not conflict with other mDNS programs commonly found on GNU/Linux systems such as avahi-daemon. It receives instructions from the `vula organize` daemon via `d-bus`, or via a python function call in monolithic mode, and publishes service records containing specifically formatted data signed under a Vula specific Ed25519 private key.

4) Configuration daemon: `vula-organize.service` runs the `vula organize` daemon which reads peer descriptors from a systemd managed socket. It runs as user `vula-organize` and as group `vula`. It does not access the network and its primary purpose is to configure the local `vula device` WireGuard interface. It retains the capability [60] `CAP_NET_ADMIN` to ensure it has the relevant authority and permission to modify the interface.

5) Additional implementation details: `vula organize` will generate cryptographic keys and write out data to the following files:

0) `/var/lib/vula-organize/keys.yaml` CSIDH secret key, Curve25519 private key, and the Ed25519 private key for the `vula organize` daemon.

1) `/var/lib/vula-organize/vula-organize.yaml` Configuration file containing relevant Vula state for the `vula organize` daemon.
2) `/etc/systemd/system/vula-organize.service` A systemd daemon configuration file.
3) `/etc/systemd/system/vula-publish.service` A systemd daemon configuration file.
4) `/etc/systemd/system/vula-discover.service` A systemd daemon configuration file.
5) `/etc/systemd/system/vula.slice` A systemd slice to contain and constrain the aforementioned systemd services.

`vula configure` will add a firewall rule using `ufw` to allow traffic to the `vula` interface (`ufw allow 5354/udp`):

```
To                         Action      From
--                         ------      ----
5354/udp                   ALLOW IN    Anywhere
5354/udp (v6)              ALLOW IN    Anywhere
```

D. Adversary realities

Often when writing about adversaries it is difficult to point to specific tools that may motivate specific design goals. Thanks to some very special whistleblowers, we have evidence from inside one of the largest, and well funded state level adversaries on the planet. We know that cryptography is a hard barrier [7] for surveillance by such adversaries. It is reasonable to expect that other large state adversaries have similar limitations, similar tools, or even access to the same tools based on geopolitical agreements. We have included two relevant internal documents from the NSA for posterity.

1) NIGHTSTAND: NIGHTSTAND as seen in Figure 8 is a so-called close access operation tool for attacking wireless devices.

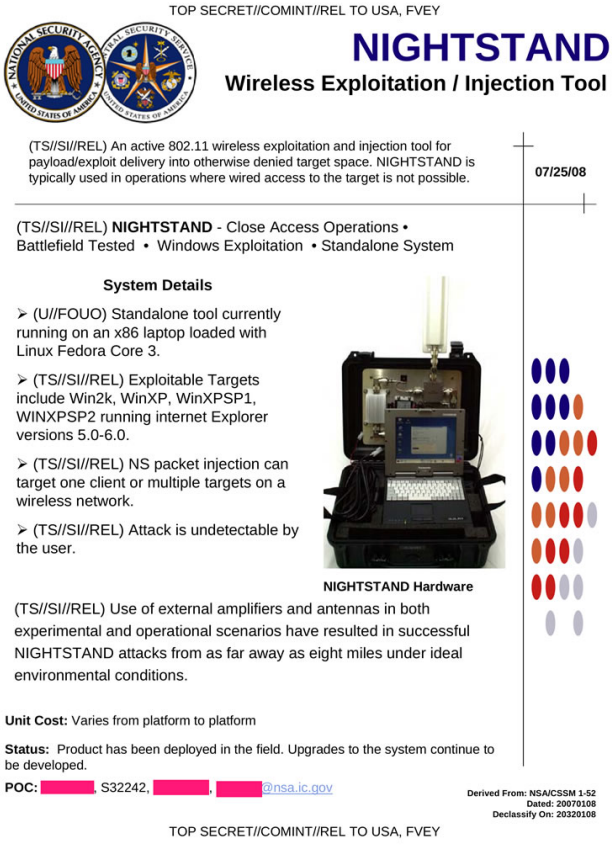

Figure 8: NSA internal product advertisement

2) SPARROW II: SPARROW II as seen in Figure 9 is a so-called Airborne Operations tool for monitoring wireless networks.

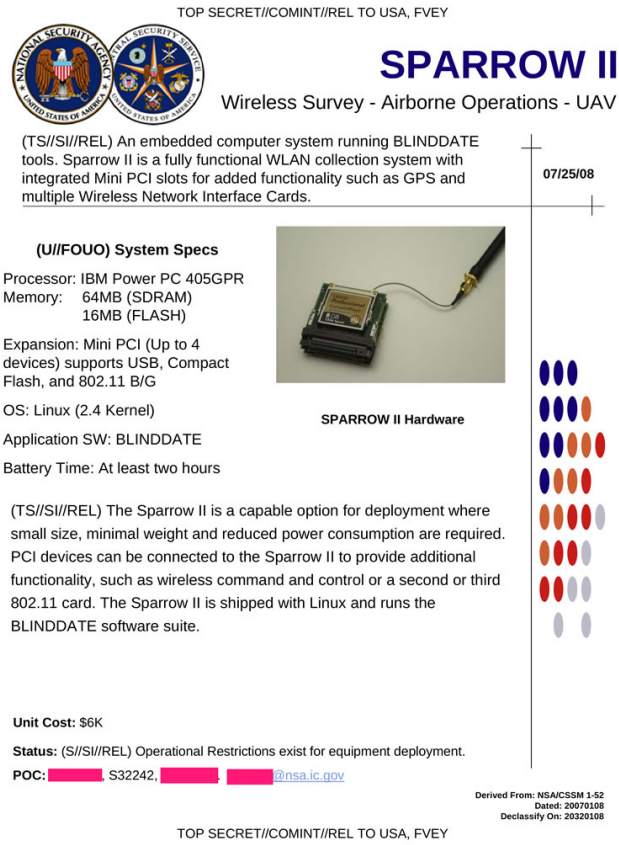

Figure 9: NSA internal product advertisement