Securing WireGuard with an HSM

Peter Van Eenoo

A Capstone Project
submitted in partial fulfillment of the
requirements for the degree of

Master of Science in Cybersecurity Engineering

University of Washington

2022

Reading Committee:
Brent Lagesse, Chair
William Erdly
Yang Peng

Program Authorized to Offer Degree:
Computer Science and Engineering

University of Washington

Abstract

Securing WireGuard with an HSM

Peter Van Eenoo

Chair of the Supervisory Committee:
Dr. Brent Lagesse
Computing & Software Systems

WireGuard is a popular, secure, and relatively new VPN implementation that has seen widespread adoption. WireGuard's basic key management in the reference implementation leaves some weaknesses that could be exploited by threat actors to steal keys, compromising a user's identity or exploit their privileged access. In my project I combined the industry-standard practice of isolating sensitive data with cutting-edge support for Curve25519 keys on an HSM. I created a WireGuard-compatible fork called WireGuard-HSM which uses the PKCS#11 interface to securely access a user's private key and perform privileged operations on a USB security key. After performing two threat model analysis and comparing the results, I show how my modifications improve the security of the WireGuard system by decreasing the attack surface and mitigating two vulnerabilities, if the host computer is compromised. WireGuard-HSMs security improvements come without a noticeable performance penalty.

TABLE OF CONTENTS

LIST OF FIGURES

GLOSSARY

WIREGUARD: a VPN technology created in 2017 by Jason DonenFeld that operates at the network layer. It aims to be a replacement for popular TLS-based VPNs like OpenVPN and IPsec. WireGuard uses a fixed-set of cryptographic primitives and by design, has no support for negotiation of these primitives.

CURVE25519: an elliptic curve with a 256-bit key size. Designed by Daniel J. Bernstein in 2005[1]. Curve25519 has a digital signature algorithm named Ed25519 and a key derivation algorithm named X25519.

X25519: an algorithm that uses Curve25519 to implement Elliptic-curve Diffie–Hellman (ECDH) key exchange. This process is also known as a key derivation function (KDF). All KDF references refer specifically to X25519.

AEAD: Authenticated encryption with additional data. A class of encryption algorithms which enforce confidentiality and authenticity of data.

HKDF: Hashed Message Authentication Code (HMAC)-based key derivation function. RFC5869 [13]

AEAD: Authenticated encryption with additional data. A class of encryption algorithms which enforce confidentiality and authenticity of data.

CHACHA20POLY1350: an AEAD encryption algorithm which is composed of the ChaCha20 stream cipher and Poly1305 for message authentication code (MAC)

PKCS#11: a platform-independent interface standard for interacting with cyrptographic tokens governed by the OASIS technical committee[14]. It defines a common interface for programs to interact with security keys, also referred to as cryptographic tokens.

HARDWARE SECURITY MODULE: (HSM) is a dedicated, physical device that safeguards digital keys and provides limited-access to the resident keys for operations on messages such as encryption, decryption, verification and authentication.

SECURITY KEY: a general term for a class of devices that implement a limited subset of HSM functionality. Security keys store cryptographic keys and restricts access through a limited interface. These devices typically connect to a computer or smartphone via USB or NFC and are small enough to be held on a user's key-ring. Nitrokey Start[15] or YubiKey[20] are examples of such devices. I will use the terms Security key and HSM interchangeably.

Chapter 1
INTRODUCTION

WireGuard is a new implementation of a Virtual Private Network (VPN) proposed in 2017. It has had a successful mainstream adoption, as evidenced by its recent inclusion in many open-source operating systems[4] as well as a native Windows kernel module[3]. MacOS, iOS, Android, FreeBSD and OpenBSD are supported as well. WireGuard has been described as "crypto-opinionated", meaning the protocol supports only one cryptographic primitive for each cryptographic requirement which allows it to completely forego cryptographic-protocol negotiation between peers, eliminating design complexity and reducing it's attack surface.

For example, WireGuard only supports Curve25519 key-pairs for client authentication and key exchange, and ChaCha20Poly1350 for symmetric encryption[5] of data. Simplicity is a design goal of WireGuard, the core protocol is implemented in under 4,000 lines of code. WireGuard goes to great lengths to resist leaking any possible information about the connected peers; it encrypts the identity of the peers in the handshake process and no connection-state information is sent over the wire so peers mange the connection state themselves using a state machine.

WireGuard has been shown to be very fast, out-performing IPsec and OpenVPN-based VPNs, in terms of throughput and response time[6], as well as easy to quickly port to a wide variety of operating systems.

An important design simplification of WireGuard is the management of public and private keys. WireGuard does not use traditional x509 digital certificates or public key infrastructure. A peer's identity is anchored to a static Curve25519-based private key stored in the configuration file. The static public keys of the initiator and responder must be pre-shared with both parties respectively, before a successful handshake can be made. A consequence of leaving key management up to the users is that this exposes a weakness in reference implementation of WireGuard, it's plain-text configuration file is an attractive target for threat actors.

1.1 Problem Definition

After performing a threat model analysis of WireGuard, I identified local static-key handling as the primary weakness in the reference implementation. It's protected from the network but not the user's machine.

The key design choice of WireGuard-HSM is to maintain client compatibility and better protect the user's key. This can be done by storing the client's private key on a USB-security key which supports Curve25519, which will maintain compatibility with mainline WireGuard clients.

WireGuard's key management has been investigated by other researchers, I briefly discuss their work on this topic in section 1.5.

1.2 Goals

In this project, my primary goal is to increase the security of WireGuard by improving it's key handling. I want to use well established methods in the computer security field to achieve this goal, without sacrificing compatibility. Instead of proposing compatibility-breaking changes, my secondary goal was to work within the confines of the existing protocol and maintain client compatibility, while improving security. I also wanted to understand what performance implications my design choices would impose on the system. System performance is discussed in section 5.1.1

1.3 Outline

I will discuss WireGuard's authentication and authorization mechanisms during the handshake process, in enough detail so readers can understand why my changes are relevant.

I will show how weaknesses in WireGuard's key management could lead to a threat actor compromising a private key and thereby a user's identity. I will briefly discuss what happens to the WireGuard system when a user's identify is compromised and why a threat actor might want to do this. I will show how the security architecture design WireGuard-HSM safeguards a user's identity in the same scenarios. Finally I will evaluate any performance impact of WireGuard-HSM on the system and the implications for usability.

1.4 Contributions

My project is primarily focused on improving the security of WireGuard through improvements to it's static-private key handling. These improvements reduce the attack surface of the program as I will attempt to demonstrate in section 5.2.2

The Elliptic Curve, Curve25519 is being used in an increasing number of open-source and commercial products from OpenSSH to Signal[11], however hardware support in security keys is currently almost non-existent. YubiKey is currently the most popular security key manufacturer, however all of their current products lack support for X25519. One of the goals of this project, as the first open source project to combine an X25519 supported security key with a popular open source VPN like WireGuard, will encourage more manufactures to add support for X25519 across the security key industry.

1.5 Related-work

The WireGuard handshake protocol has gone through extensive formal verification using Tamarin proof system [8]. Researchers have identified some potential weaknesses in WireGuard around quantum computers[10] but no serious vulnerabilities have been identified, since it was introduced five years ago in 2017.

WireGuard's key handling is a current area of research. A paper by [19] focused on modifying WireGuard to use Trusted Execution Environment (TEE) of an Android phone to securely store the long-term static private key. The TEE is similar to the secure enclave in Intel processors. This work has the benefit that is supports a mobile operating system but it's implementation is restricted to a specific android maker's platform.

Chapter 2
BACKGROUND

Next we will discuss several key points about WireGuard's handshake process and it's major parts so readers can understand the context of my project's modifications. How peers are authenticated, how the symmetric cipher is exchanged and how the handshake process works between peers.

WireGuard's 'crypto-opinionated' design and lack of any negotiation does come with consequences. The lack of cryptographic negotiation means that if vulnerabilities are identified and a cryptographic component must be modified or replaced, this will result in complete peer incompatibility, until all peers are updated to the new protocol. This also means that project forks are limited in the scope of changes they can introduce, if they wish to maintain interoperability with mainline WireGuard peers.

WireGuard makes exclusive use of Curve25519 for user authentication and authorization. First I will discuss it's technical use in WireGuard and how it is tied to a users identity and authentication. For a complete description of the handshake process and details, see [7][5].

2.1 Curve25519 key pairs

Curve25519 is a patent-free, high-performance elliptic-curve with a reference implementation that is in the public-domain. This makes it an attractive curve for many projects [11]. A useful feature of Curve25519 is that any 32-byte value is a valid private key. Meaning no Curve25519 key requires validation which helps avoid small subgroup attacks. Curve25519 keys have also been shown to be immune to some classes of timing attacks[2][18]. Deriving the public key from the private key is fast enough that programs such as WireGuard don't need to store the client's public key in their configuration file. WireGuard only saves the private key and quickly derives the public key on program start-up.

A user's identity is rooted in their Curve25519 'long-term static public key' which is generated from their private key and shared out-out-band with every peer that a user wishes to communicate. There are no mechanisms for key revocation built into the WireGuard protocol and a user's static public key never expires.

During the handshake process, WireGuard creates short-lived Curve25519 key pairs which are independent of the long-term static public key. These short-lived key pairs are referred to as the ephemeral session keys and they are used to derive the symmetric key for each peer's sending channel, during the handshake. These ephemeral key-pairs are valid for a default of 2 minutes per session, which after that time, a peer will initiate another handshake in order to generate new keys. WireGuard has a novel key rotation mechanism which guarantees that nonce-reuse will not happen but further discussion of this is out-of-scope for this paper.

2.2 Session Establishment

WireGuard assumes that each party's public key is securely exchanged out-of-band. This exchange must take place before a handshake can be performed. WireGuard refers to parties as an initiator and a responder, since there is no strict definition of client or server. A handshake is performed using an initiation message, sent by the initiator to the responder. The first packet is authenticated by including the peers encrypted public key in the initiation message, I will discuss this further in the next section.

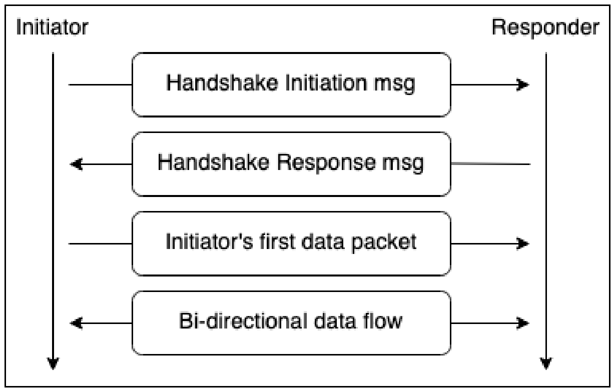


Figure 2.1: Simplified Handshake Packet Sequence Diagram

This handshake has been described as a 1.5 round-trip-time (RTT) handshake over UDP which is inherently connection-less. See Figure 2.1 for a visual description. The 1.5 RTT handshake is defined by the following steps: the initiator sends a handshake initiation message, the peer responds to a properly authenticated initiation message with a handshake response message. Finally the initiator sends the first data packet. These three message are required for the session to be considered established and compose the 1.5 RTT handshake process. Consider the similarity to TCP's three-way-handshake.

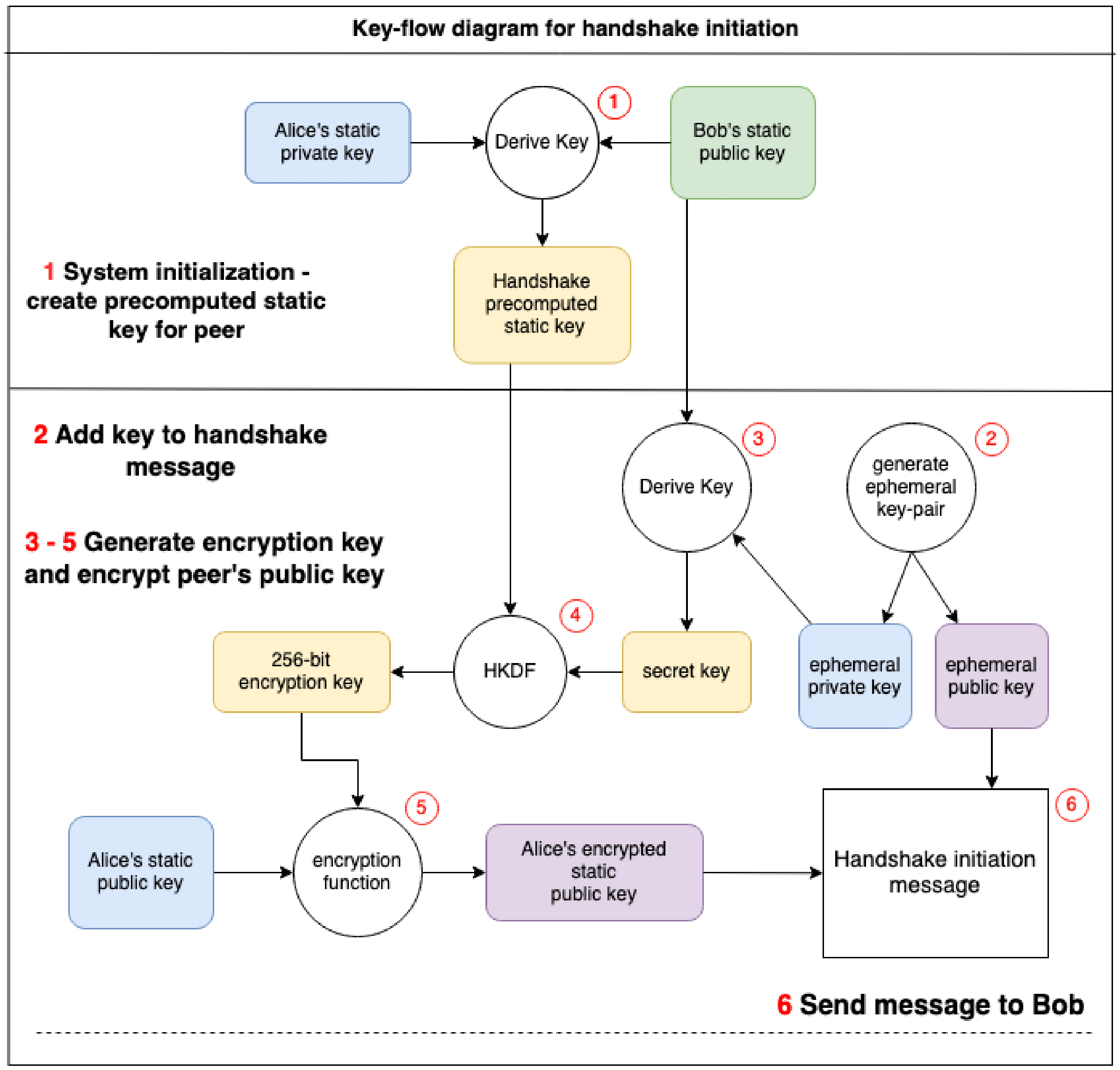

Key-flow diagram for handshake initiation
Alice's static private key
Derive Key
1
Bob's static public key
1 System initialization - create precomputed static key for peer
Handshake precomputed static key
2 Add key to handshake message
3 - 5 Generate encryption key and encrypt peer's public key
Derive Key
3
generate ephemeral key-pair
2
4
256-bit encryption key
HKDF
secret key
ephemeral private key
ephemeral public key
5
6
Alice's static public key
encryption function
Alice's encrypted static public key
Handshake initiation message
6 Send message to Bob

Figure 2.2: Detailed Handshake Initiation Process

2.2.1 Handshake initiation

Let's call the initiator Alice and the peer (the responder) Bob. The diagram in 2.2 shows the detailed process on Alice's computer that WireGuard goes through to send a handshake initiation message. First, WireGuard reads Alice's private key and Bob's public key from it's configuration file, sending them to a KDF to generate the handshake's precomputed-static key. Second, Alice generates an ephemeral Curve25519 key-pair and attaches that public key to her initiation message. The ephemeral private key and the precomputed static key from step 1, are used as input to an HKDF which generates, what WireGuard refers to as the chaining key. The chaining key is used in part as the key to the symmetric-encryption function ChaCha20Poly1350 which encrypts Alice's static public key. The encrypted form of Alice's public key is finally added to the handshake initiation message and used by Bob to authenticate that only Alice could have sent this message.

When Bob receives this message, assuming he already has Alice's public key, he will follow this process in reverse-order. Bob will generate his own ephemeral key-pair and send his ephemeral public key in his handshake response message to Alice. The chaining key mentioned earlier is also used as input into an HMAC function, so both parties can derive the sending and receiving keys for symmetric encryption of transport data. The state is now identical for both peers and they have each derived the the symmetric encryption key to decrypt data sent by the peer. It's worth noting that unlike many encryption schemes, the key to the symmetric cipher is not same in each direction because each peer chooses their own 'sending channel' cipher-key.

A benefit of requiring that the sender include their encrypted public key, is that any party running WireGuard is resistant to active network enumeration. In order to elicit a response from a node running WireGuard, the initiator must have knowledge of the responder's public key and the responder must have knowledge of the initiators public key.

WireGuard uses encrypted timestamps to avoid replay attacks and it employs a novel cookie mechanism to avoid DoS attacks by unauthenticated parties however further discussion on the handshake is out of scope for this paper.

Key Rotation

WireGuard sessions guarantee perfect forward secrecy by including a key rotation process (rekey) between peers. This rekey event occurs after a default of 120 seconds, or after 2^{60} messages have been exchanged. The key rotation process is essentially another handshake process with one key difference: if Alice is the peer performing the rekey process, then in step 1, see Figure2.2, Bob's most recent ephemeral public key is used as input to the KDF instead of his static public key. This has implications for my project which we will discuss in section 3.1.1

Identify

The WireGuard protocol has no distinction between authentication and authorization. Authentication and authorization are combined in order to reduce the handshake round-trip-time. If a peer can successfully authenticate a handshake initiation message and response, the user is allowed to communicate over the session, with whatever resources may be gated on or behind the WireGuard peer. Now that we understand the handshake initiation process and session key rotation in more detail, it should be clear how the initiators static private and public key, as well as the responders public key, form the basis for establishing a WireGuard session.

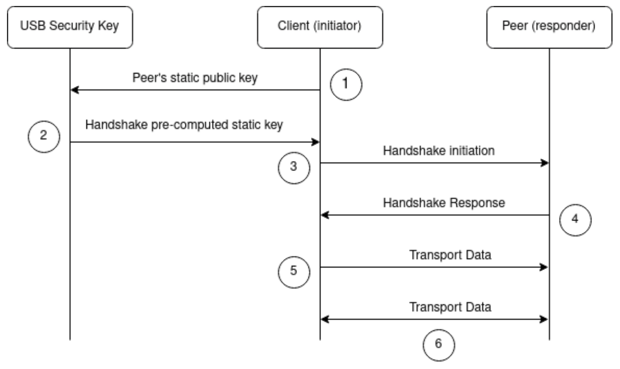
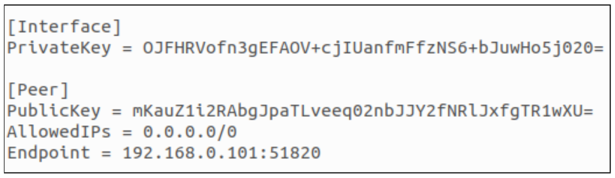


Figure 2.3: Simplified WireGuard-HSM Handshake Narration Process

2.3 Key Storage

As mentioned previously, WireGuard leaves key management up to the user. Figure 2.4 shows an example configuration file for WireGuard. The user's private, long-term static private key is under the "[Interface]" section and "[Peer]" section contains each peer's pre-shared static public key, as well as IP address information. The private key is saved in plain-text as a Base64 encoded string.



```
[Interface]
PrivateKey = OJFHRVofn3gEFAOV+cjIUanfmFfzNS6+bJuwHo5j020=

[Peer]
PublicKey = mKauZ1i2RAbgJpaTLveeq02nbJJY2fNRlJxfgTR1wXU=
AllowedIPs = 0.0.0.0/0
Endpoint = 192.168.0.101:51820
```

Figure 2.4: Example WireGuard Configuration File from the Reference Implementation

2.4 HSM - Nitrokey

An HSM is a widely used, industry-standard device that is used to safeguard digital keys. Many HSMs undergo rigorous testing and certification, to validate their compliance with internationally recognized standards such as Federal Information Processing Standard (FIPS) and Common Criteria. They also typically use the ubiquitous and platform-independent PKCS#11 interface.

Figure 2.5: Image of the Nitrokey Start HSM

At the time of writing, the only commercially available HSM that offered the right features features for the project was the "Nitrokey Start" from Nitrokey in Germany[15]. A few HSM list support for 'Curve25519' yet most only support Ed25519 while WireGuard only uses the X25519 algorithm of Curve25519.

The Nitrokey Start connects to the computer via a USB-A interface. It securely stores cryptographic keys and provides an interface using the PKCS#11 standard. An HSM safeguards private keys by only allowing authenticated users to perform a strict subset of key management and cryptographic functions on a slot. A slot holds private and public keys and is protected by a user pin. HSMs are also designed in such a way that physical tamping with the device should destroy the private-keys, making them unreadable and useless.

Chapter 3
METHODOLOGY

Here I discuss key parts of the project architecture, so readers can understand the context of the different systems that were created for my project.

The confidentiality of a user's private key is paramount to proper client authentication and authorization in WireGuard. My project WireGuard-HSM, aims to improve upon WireGuard's key handing by using an HSM to securely store the private key, without modifying WireGuard in such a way to introduce incompatibility.

3.1 Project Design

Figure 3.1 shows the major WireGuard-HSM system interaction. When WireGuard-HSM attempts to establish a new session with a peer, WireGuard-HSM interacts with a module named 'pkclient' which uses the PKCS#11 interface to talk to the connected HSM.

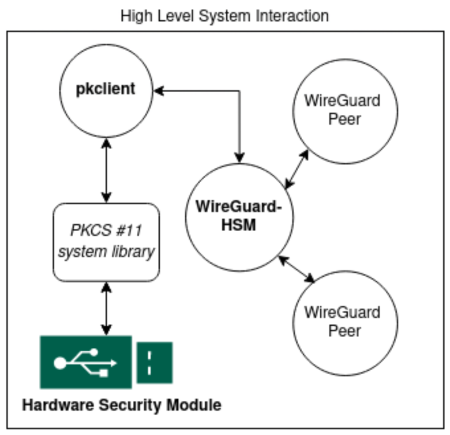


Figure 3.1: A High-Level WireGuard-HSM System Interaction Diagram

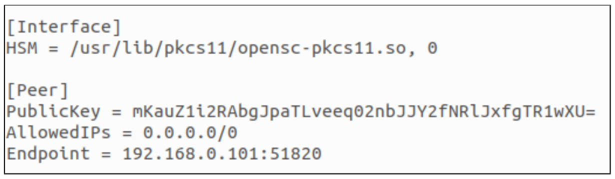
3.1.1 Pkclient

I created a standalone project called 'pkclient' to handle the interactions with the HSM and manage state. Pkclient implements functions specific to the cryptographic operations required by WireGuard but it's implemented as a standalone module, to limit the amount of modifications to WireGuard. Pkclient itself incorporates a Golang package to integrate the low-level PKCS#11 interface calls, exposing a simplified interface to callers. Pkclient handles the HSM session establishment, verification that the correct key-types are present on the HSM, user-interactions for pin authentication. From WireGuard-HSM's point of view, pkclient is responsible for key derivation functions when the long-term static private key is needed as input, which is performed by the HSM. When the peer session expires and performs the key rotation, described in section 2.2.1, the ephemeral public key of the peer is sent in a call to pkclient to derive the session's new shared secret.

Normally WireGuard has full-access to the long-term static private key for deriving the long-term static public key, so pkclient must also provide a function to return the public key contents, which enables the user to share their public key with other peers. This is the public key referenced in step 5 of Figure 2.2.

3.1.2 WireGuard-HSM

WireGuard-HSM has the long-term private static key completely removed from the configuration file, replaced by a system library path to a PKCS#11 library and the slot number on the HSM to use, see an example file in Figure 3.2. In places where WireGuard expects access to the private key, changes were made to use the interface with pkclient for those calls instead.


```
[Interface]
HSM = /usr/lib/pkcs11/opensc-pkcs11.so, 0

[Peer]
PublicKey = mKauZ1i2RAbgJpaTLveeq02nbJJY2fNRlJxfgTR1wXU=
AllowedIPs = 0.0.0.0/0
Endpoint = 192.168.0.101:51820
```

Figure 3.2: Example WireGuard-HSM Configuration File

WireGuard-HSM was designed to work in a two modes, the HSM mode and the original software mode. The 'software mode' is the original, unchanged mode that has access to the private key in the configuration file. WireGuard-HSM executes it's program flow in the same way as WireGuard-go but it used if/else statements to access the HSM or software mode.

This approach helped me ensure that no incompatibility was introduced into the system.

Simplicity and separation of privileges were important goals in my system design, so by design WireGuard-HSM abstracts much of it's functionality over to pkclient, keeping the changes to the WireGuard-go code-base to a minimum.

Some small changes were made to the 'wireguard-tools' package to allow me to modify the configuration file for WireGuard-HSM but they out of scope for our purposes and only affect program initialization.

3.2 Threat Model

Since WireGuard and all VPN clients, allow users access to protected resources across networks, there is inherent value for an attacker to gain access to these systems. The attacker may be highly motivated, targeting a specific individual or the attacker may be opportunistic, deploying malware en-mass and harvesting credentials of WireGuard nodes, for later sale or use.

In the following section, I discuss the threat models for WireGuard and WireGuard-HSM, comparing WireGuard-HSM to see the relative strengths and weaknesses. This will help us evaluate the system design and perform a comparative threat model analysis. This analysis will show the relative strengths and weaknesses of the designs and if WireGuard-HSM has improved the security of the system at all.

3.3 DREAD - Risk Assessment Model

In order to assess the relative risk in both systems, I will identify and calculate a risk score for a given weakness based upon the DREAD model. I have used the rankings and descriptions of risk from OpenStack Security Group's DREAD and OWASP process [16][17] as a basis for creating my metrics. I have also used the table of descriptions and rankings from OWASP's published DREAD model and applied them as plainly as possible to remove as much experimenter bias as I can from my DREAD model. Assigning these rankings is somewhat subjective but following other groups procedures will increase the objectivity.

3.3.1 DREAD categories

The five major categories of DREAD are defined as follows:

- **D**amage - how high is the impact of a successful attack?
- **R**eproducibility - how easy it is to reproduce the attack?
- **E**xploitability - how easy is it to launch the attack?
- **A**ffected users - how many people will be impacted?
- **D**iscoverability - how easy it is to discover the threat?

DREAD scores have been criticized as being subjective however the calculated risk score can help us identify the relative strength and weaknesses when applied to similar systems. The scores are based on a ranking of 0-10, low to high for each category, added together and divided by the number of categories to give a final risk score to the weakness. The calculation of risk is as follows:

$$Risk = \frac{Dam + Rep + Exp + Aff + Disc}{5} \tag{3.1}$$

Damage - If the vulnerability is exploited, how much damage does it cause to the system?

Rating	Description
0	None
1	Individual user data is exposed. A public key is exposed
3	All individual public keys are exposed
5	System availability impacted intermittently
7	Single user DoS possible or loss of a private key
9	Major system compromise. Loss of private keys and peer public keys. DoS possible.
10	Complete system compromise. Loss of all private and public keys. DoS possible and decryption of peer traffic possible.

Reproducibility - How reliably can the vulnerability be exploited?

Rating	Description
0	Very hard or impossible. The vulnerability is unstable and statistically unlikely to be reliably exploited
1	Very difficult to reproduce, physical access required - may be possible once or twice
3	Attacker must gain physical access to the computer - difficult to reproduce
5	One or two steps required, tooling / scripting readily available
10	Unauthenticated users can trivially and reliably exploit using only a web browser

Exploitability - How difficult is the vulnerability to exploit for the attacker?

Rating	Description
0	N/A - I assert that every vulnerability is exploitable, given enough time and effort
1	Even with direct knowledge of the vulnerability, a viable path for exploitation is uncertain
3	Advanced techniques required, custom tooling. Only exploitable by authenticated users
5	Exploit is available and usable by skilled, authenticated users
7	Exploit is available and usable by unauthenticated users
10	Trivial - using a web browser

Affected Users - How many users will be affected if the vulnerability is exploited?

Rating	Description
0	No users affected
3	A single individual user is impacted
5	At least two users are impacted
10	All users of the system are impacted

Discoverability - How easy is the vulnerability found by an attacker?

Rating	Description
0	Very hard to impossible to discover, even given source code access
5	Discoverable by observing networking traffic
9	Vulnerability details publicly available
10	The information is visible in the web browser address bar

I will use Threat Model Data-Flow Diagrams to identify where data flows across trust boundaries. I will classify vulnerabilities and apply mitigations using terms and techniques from the STRIDE method[9]. In the following section we will briefly discuss STRIDE.

3.3.2 STRIDE

The STRIDE method provides tools for threat classification and vulnerability mitigation. Each threat breaks a particular and desirable security property of the system. For example, if a threat can take advantage of a vulnerability in the system such as information disclosure then confidentiality is violated. The recommended action according to the STRIDE method is applying a mitigation that will restore confidentiality. See Table 3.1 for more details.

Threat	**Security Property**
Spoofing	Authenticity
Tampering	Integrity
Repudiation	Non-repudiation
Information Disclosure	Confidentiality
Denial of Service	Availability
Elevation of Privilege	Authorization

Table 3.1: STRIDE Method for classifying and mitigating security threats

3.3.3 DREAD Threat Rating Tables

Once we have calculated the DREAD scores, we can further simplify the total score for each threat into Critical, High, Medium and Low based on the average DREAD score, as other researchers have done when using DREAD [12]. These rankings are illustrated in Table 3.2.

Average	Rating
0.0 - 1.5	Low
1.6 - 3.5	Med
3.6 - 5.0	High
5.1 +	Crit

Table 3.2: DREAD Threat Rating Table

Chapter 4
THREAT MODELS AND COMPARISON

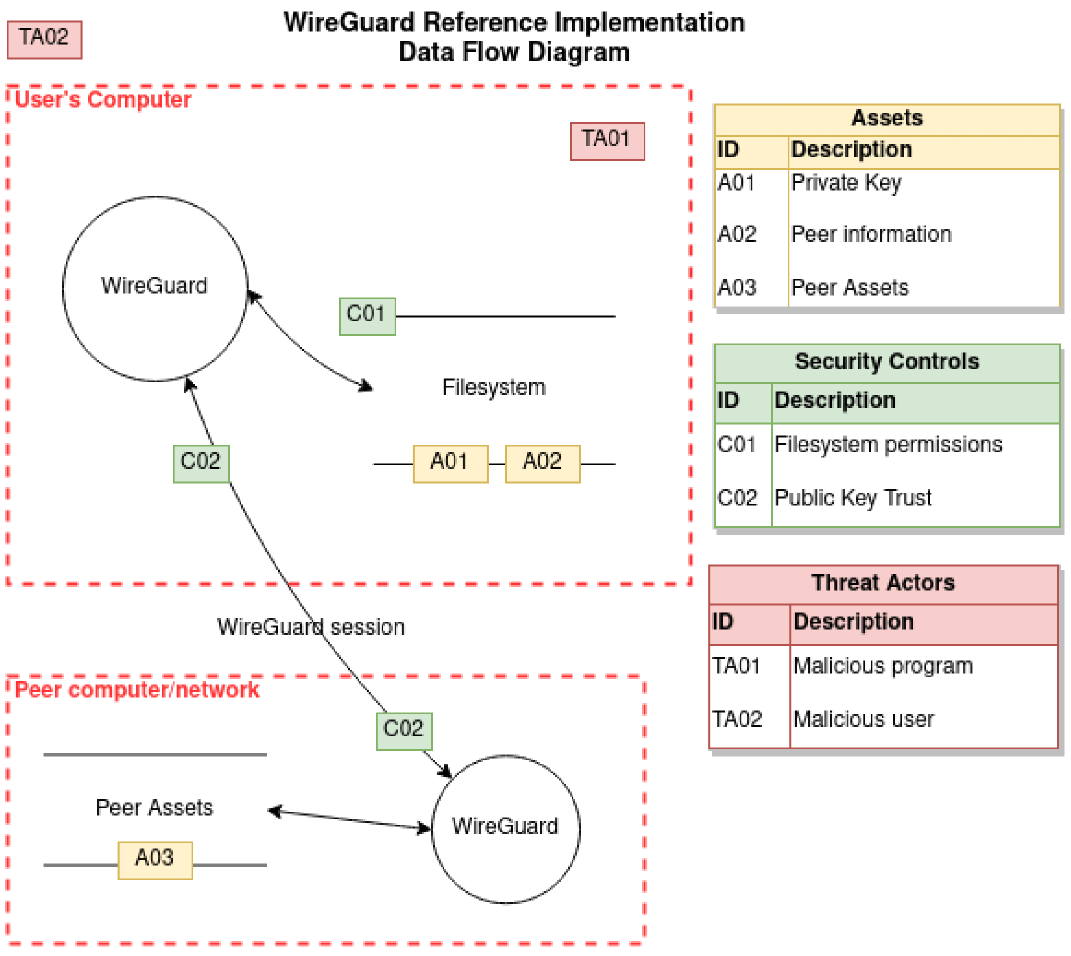
The two threat models in the following sections are primarily concerned with how data at rest is handled by both systems. WireGuard's handling of data in transit is out of scope for the analysis. First I will identify and evaluate the risks in the WireGuard reference implementation. Second I will identify and evaluate the same risks and any new risks in WireGuard-HSM. Finally I will compare the results to draw my conclusions about the modifications in WireGuard-HSM.

4.1 WireGuard - Reference Impl. Threat Model

This threat model is focused on WireGuard's 1. Asset handling related to configuration file which contains private and public keys. 2. The security controls around the assets. 3. The threats to the system. The computer running WireGuard will be considered Alice's computer and the WireGuard peer that Alice connects to is Bob. Both computers are turned on and connected to the network but Alice has not yet established a WireGuard session to Bob's computer. Refer back to section 2.2.1 on how WireGuard establishes a session with a peer.

4.1.1 Assets

The file system on Alice's computer contains the WireGuard configuration file, see Figure2.4. Asset A01 is Alice's Curve25519-based long-term static private key. Asset A02 is Bob's connection information consisting of his static public key and internet address. Asset 03 is the label for any potential assets on Bob's side of the WireGuard tunnel. Access to Bob's machine might be a highly valuable asset or Bob's network access might be the highly valuable asset. This model combines these possibilities into 'Peer Assets' for simplification.



Assets	
ID	**Description**
A01	Private Key
A02	Peer information
A03	Peer Assets

Security Controls	
ID	**Description**
C01	Filesystem permissions
C02	Public Key Trust

Threat Actors	
ID	**Description**
TA01	Malicious program
TA02	Malicious user

Figure 4.1: Threat Model for the WireGuard Reference Impl. - Data Flow Diagram

4.1.2 Security controls

Access control in WireGuard's reference implementation for A01 and A02 is C01: file system permissions. Most of the documentation indicates that sufficient file-system permissions' should be set on the WireGuard configuration file to allow only administrator access and restrict access by unprivileged users. This is not an enforced measure, meaning WireGuard will still start and not warn the user if the file permissions are insufficient.

Security control C02, access to each WireGuard peer, is the pre-shared public key system implemented by WireGuard.

4.2 Threat Actor 1

The primary threat is TA01: a piece of malware that has gained access to Alice's computer and will search for keys in configuration files. This malware could have been delivered by acquiring a zero-day exploit or by using a known vulnerability with a piece of software running on Alice's computer. Using a known vulnerability would likely lower the cost of the attack.

4.2.1 Insufficient permissions

If TA01 gains access to Alice's computer and the file system permissions were not correctly set on the WireGuard configuration file, then the threat gains access to A01 and A02. Access to these assets results in loss of A03 by bypassing C02. See Table 4.1 and 4.2.

Potential for: Information Disclosure, Spoofing, DoS

Category	Score	Rationale
Damage	9	Loss of A01, A02, A03. DoS for a single user possible
Reproducibility	6	Two steps required, tooling and scripting readily available
Exploitability	5	Exploit is available/understood, usable with only moderate skill by authenticated users or malware
Affected Users	5	At least two users are affected
Discoverability	5	Attack may be targeted - exploited by general malware infection

Dread Score: 6 Crit

Table 4.1: Attack 1 - Loss of A01 and A02 after access to configuration file by using malware delivered using a known vulnerability and the configuration file permissions are incorrectly set

Potential for: Information Disclosure, Spoofing, DoS

Category	Score	Rationale
Damage	9	Loss of A01, A02, A03. DoS for a single user possible
Reproducibility	4	Two steps required, tooling and scripting readily available
Exploitability	4	Advanced techniques required. General malware infection would have unimpeded access to access assets
Affected Users	5	At least two users are affected
Discoverability	4	Attack may be targeted - exploited by advanced malware

Dread Score: 5.2 - Crit

Table 4.2: Attack 3 - Loss of A01 and A02 after access to configuration file by using malware delivered using a zero-day exploit and the configuration file permissions are incorrectly set

4.2.2 Sufficient permissions

If C01 is configured properly, then TA01 will need exploit one more vulnerability, in-order to gain access to A01 and A02. This offers a single layer of security, so the cost of this attack cost is slightly higher. Exploit chains are common-place in today's attacks so this is a reasonable assumption. See Table 4.3 and 4.4.

Potential for: Information Disclosure, Spoofing, DoS

Category	Score	Rationale
Damage	9	Loss of A01, A02, A03. DoS for a single user possible
Reproducibility	5	Three steps required, tooling and scripting readily available
Exploitability	5	Exploit is available/understood, usable with only moderate skill by authenticated users or malware
Affected Users	5	At least two users are affected
Discoverability	4	Attack may be targeted - exploited by general malware infection

Dread Score: 5.6 - Crit

Table 4.3: Attack 2 - Loss of A01 and A02 after access to configuration file by using malware delivered using a known vulnerability and the configuration file permissions are correctly set

Potential for: Information Disclosure, Spoofing, DoS

Category	Score	Rationale
Damage	9	Loss of A01, A02, A03. DoS for a single user possible
Reproducibility	3	Two steps required, tooling and scripting readily available
Exploitability	3	Advanced techniques required. General malware infection would have unimpeded access to access assets
Affected Users	5	At least two users are affected
Discoverability	3	Attack may be targeted - exploited by advanced malware

Dread Score: 4.6 - High

Table 4.4: Attack 4 - Loss of A01 and A02 after access to configuration file by using malware delivered using a zero-day exploit and the configuration file permissions are correctly set

4.3 Threat Actor 2

TA02 is a malicious user who gains direct or indirect access to Alice's computer.

In a direct method of attack, TA02 gains physical access to Alice's laptop and uses some technical or non-technical exploit to read the hard-drive. If Alice's machine uses disk-encryption then this increases the difficulty of the exploit because a second exploit must be used to read the disk contents. See Table 4.5 and 4.6.

Potential for: Information Disclosure, Spoofing, DoS

Category	Score	Rationale
Damage	9	Loss of A01, A02, A03. DoS for a single user possible
Reproducibility	4	Two steps required. Attacker must gain physical access to the computer
Exploitability	7	Exploit is understood, usable by unskilled attacker
Affected Users	5	At least two users are affected
Discoverability	3	Attack must be targeted to user

Dread Score: 5.6 - Crit

Table 4.5: Attack 5 - Loss of A01 and A02 after access to configuration file by using physical access to unprotected machine.

Potential for: Information Disclosure, Spoofing, DoS

Category	Score	Rationale
Damage	9	Loss of A01, A02, A03. DoS for a single user possible
Reproducibility	4	Two steps required. Attacker must gain physical access to the computer
Exploitability	5	Exploit is understood, usable by moderately skilled attacker
Affected Users	5	At least two users are affected
Discoverability	3	Attack must be targeted to user

Dread Score: 5.2 - Crit

Table 4.6: Attack 6 - Loss of A01 and A02 after access to configuration file by using physical access to a protected machine.

The indirect method of attack could be performed by TA02 using social engineering tactics such as a phone-call or email, masquerading as someone from Alice's IT department, asking her to, in some way, disclose the contents of the WireGuard configuration file, leading to disclosure of assets A01 and A02. See Table 4.7

Potential for: Information Disclosure, Spoofing, DoS

Category	Score	Rationale
Damage	9	Loss of A01, A02, A03. DoS for a single user possible
Reproducibility	4	Two steps required. Attacker must find and convince user to disclose contents of configuration file
Exploitability	5	Little technical knowledge is required but trust or deception must be used
Affected Users	5	At least two users are affected
Discoverability	3	Attack is well understood

Dread Score: 6.6 - Crit

Table 4.7: Attack 7 - Loss of A01 and A02 after access to configuration file is gained by social engineering.

4.4 Vulnerabilities

Once any of the previously weaknesses and have been taken advantage of by a threat, and the threat actor gains access to assets A01 and A02, this creates two classes of vulnerabilities in our WireGuard system.

4.4.1 Spoofing

Loss of A01 and A02 together allow an attacker to spoof Alice's identity with Bob because the attacker can complete a handshake with Bob as Alice and establish an authenticated session. VPN's are commonly used as the first layer of defense for network access, so A03 is an attractive asset. In this scenario, spoofing threatens authentication in our system between because unauthorized parties can now masquerade as authorized users, potentially gaining access to protected resources.

4.4.2 Denial of Service

A Denial of Service vulnerability is also possible with the loss of A01 and A02. Once a threat can perform the impersonation of Alice, then a DoS could be leveraged against any future connection attempts that Alice's may perform with Bob. When Bob receives and establishes a session with Alice's credentials, the initiator includes a randomly generated 4-byte value as the 'sender's index'. Subsequent handshake initiation messages must include this value or the receiver will drop the packet. Once Bob has an established session with the threat, the real Alice's initiation messages won't have the correct, random sender's index according to Bob's session table and he will drop her packets.

A DoS attack threatens availability in the system because Alice's connection to Bob's resources can be denied by a malicious third-party.

4.5 Vulnerability Mitigation

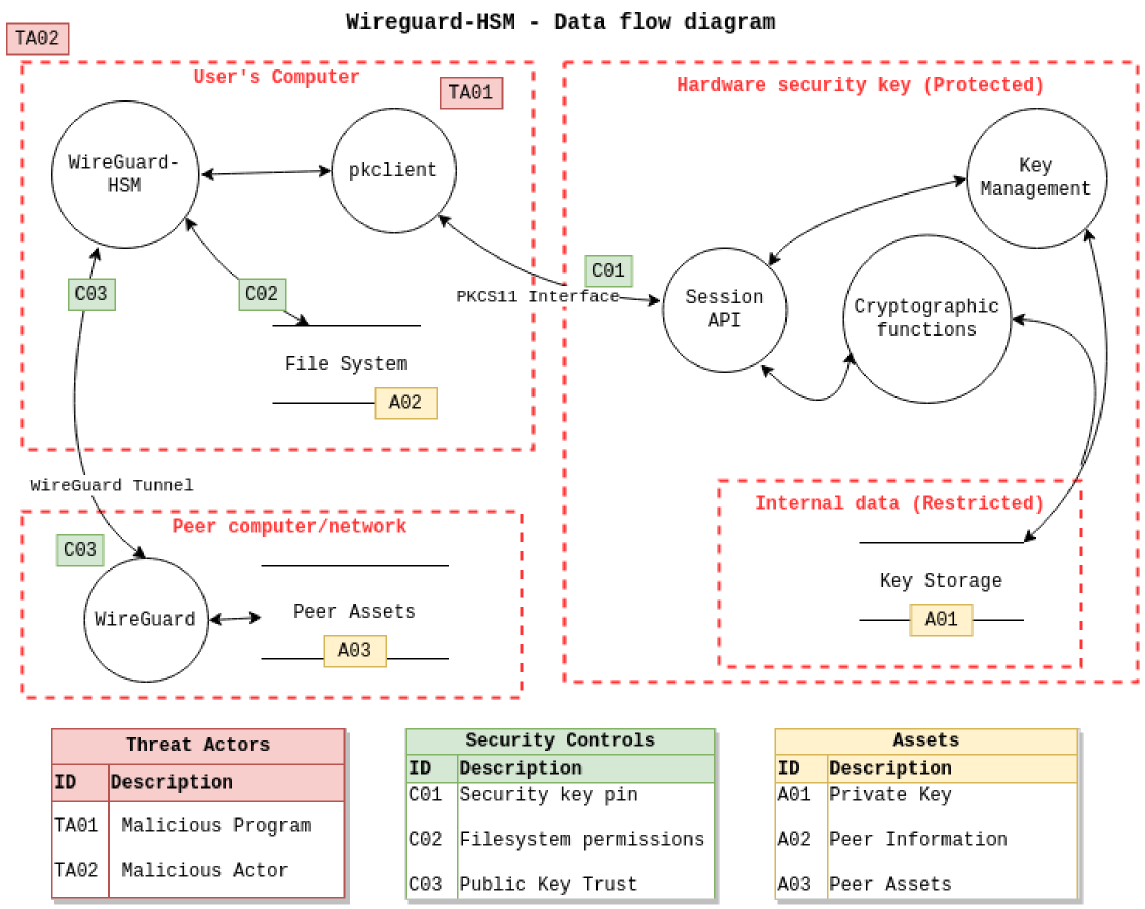
Both of the vulnerabilities that were identified, spoofing and DoS, were only enabled once the system suffered an information disclosure of the plain-text configuration file. We can address the root of both vulnerabilities by applying STRIDE's recommended mitigation for information disclosure which is confidentiality.

The confidentiality of the configuration file is a weakness in the design of the system. The entire configuration file cannot be protected or encrypted without major design changes to the program. Security is often a series of trade-offs, in which system designers must consider user-convenience vs system security. If I remove the private key from the configuration file and place a security control around it, this should decrease user convenience at the benefit of increasing security and mitigating the identified risks.

4.6 WireGuard-HSM Threat Model

This threat model is focused on WireGuard-HSM's asset handling related to the private key and peer public keys. All assumptions laid out for the previous threat model in section 4.1

are the same here.



Threat Actors	
ID	**Description**
TA01	Malicious Program
TA02	Malicious Actor

Security Controls	
ID	**Description**
C01	Security key pin
C02	Filesystem permissions
C03	Public Key Trust

Assets	
ID	**Description**
A01	Private Key
A02	Peer Information
A03	Peer Assets

Figure 4.2: A Threat Model for WireGuard-HSM - Data Flow Diagram

4.6.1 Assets

The file system on Alice's computer contains the WireGuard-HSM configuration file, example in Figure2.4. Asset A01 is Alice's Curve25519-based long-term static private key. Asset A02 is Bob's peer information consisting of his static public key and IP/DNS address. Asset 03 is the label for any potential assets on Bob's side of the WireGuard tunnel. Access to Bob's machine might be a highly valuable asset or Bob's network access might be the highly valuable asset. This model combines these possibilities into 'Peer Assets' for simplification.

4.6.2 Security controls

Access to asset A01 is behind two controls. First A01 is now on a physically separate system, the HSM. Alice must plug in the HSM and authenticate with a PIN, Control C01, before cryptographic operations can be performed on A01 via the PKCS#11 interface with the session API. File system permissions are used to protect the Peer Information, A02. Security control C03, access to each WireGuard peer, is the pre-shared public key system implemented by WireGuard.

4.7 Threat Actor 1

The primary threat is TA01: a piece of malware that has gained access to Alice's computer and will search for keys in configuration files. This malware could have been delivered by acquiring a zero-day exploit or by using a known vulnerability with a piece of software running on Alice's computer. Using a known vulnerability would likely lower the cost of the attack.

Insufficient permissions

If TA01 gains access to Alice's computer and the file system permissions were not correctly set on the WireGuard-HSM configuration file, then the threat gains access to the A02: Peer Information. Loss of A02: Peer Information, may be desirable for an attacker but this does not enable an additional vulnerability in the system. See Table 4.8 and 4.9.

Potential for: Information Disclosure

Category	Score	Rationale
Damage	1	The public key of the peer is revealed
Reproducibility	6	Two steps required, tooling and scripting readily available
Exploitability	5	Exploit is available/understood, usable with only moderate skill by authenticated users or malware
Affected Users	0	No users affected
Discoverability	5	Attack may be targeted - exploited by general malware infection

Dread Score: 3.4 - Medium

Table 4.8: Attack 1 - Loss of A02 after access to configuration file by using malware delivered using a known vulnerability and the configuration file permissions are incorrectly set.

Potential for: Information Disclosure

Category	Score	Rationale
Damage	1	Loss of A01, A02, A03. DoS for a single user possible
Reproducibility	4	Two steps required, tooling and scripting readily available
Exploitability	4	Advanced techniques required. General malware infection would have unimpeded access to access assets
Affected Users	0	No users affected
Discoverability	4	Attack may be targeted - exploited by advanced malware

Dread Score: 2.6 - Medium

Table 4.9: Attack 3 - Loss of A02 after access to configuration file by using malware delivered using a zero-day exploit and the configuration file permissions are incorrectly set.

Sufficient permissions

If C01: File system permissions, is configured properly, then TA01 will need to use an additional exploit to bypass the file system permissions and access A02, slightly increasing the cost of the attack. Loss of A02: Peer Information, may be desirable for an attacker but this does not enable an additional vulnerability in the system. See Table 4.10 and 4.11.

Potential for: Information Disclosure

Category	Score	Rationale
Damage	1	Loss of A01, A02, A03. DoS for a single user possible
Reproducibility	5	Three steps required, tooling and scripting readily available
Exploitability	5	Exploit is available/understood, usable with only moderate skill by authenticated users or malware
Affected Users	0	No users affected
Discoverability	4	Attack may be targeted - exploited by general malware infection

Dread Score: 3.0 - Medium

Table 4.10: Attack 2 - Loss of A02 after access to configuration file by using malware delivered using a known vulnerability and the configuration file permissions are correctly set.

Potential for: Information Disclosure

Category	Score	Rationale
Damage	1	Loss of A01, A02, A03. DoS for a single user possible
Reproducibility	3	Two steps required, tooling and scripting readily available
Exploitability	3	Advanced techniques required. General malware infection would have unimpeded access to access assets
Affected Users	0	No users affected
Discoverability	3	Attack may be targeted - exploited by advanced malware

Dread Score: 2.0 - Medium

Table 4.11: Attack 4 - Loss of A02 after access to configuration file by using malware delivered using a zero-day exploit and the configuration file permissions are correctly set.

4.8 *Threat Actor 2*

TA02 is a malicious user who gains direct or indirect access to Alice's computer.

Physical Access

In a direct method of attack, TA02 gains physical access to Alice's computer and uses some technical or non-technical exploit to read the hard-drive. If Alice's machine uses disk-encryption then this increases the cost of the attack because a second exploit must be used to read the disk contents. Loss of A02: Peer Information, may be desirable for an attacker but this does not enable an additional vulnerability in the system. A01 is held on Alice's HSM not on the computer. See Table 4.12 and 4.13.

Potential for: Information Disclosure

Category	Score	Rationale
Damage	1	Loss of A01, A02, A03. DoS for a single user possible
Reproducibility	4	Two steps required. Attacker must gain physical access to the computer
Exploitability	7	Exploit is understood, usable by unskilled attacker
Affected Users	0	No users affected
Discoverability	3	Attack must be targeted to user

Dread Score: 3.0 - Medium

Table 4.12: Attack 5 - Loss of A02 after access to configuration file by using physical access to unprotected machine.

Potential for: Information Disclosure

Category	Score	Rationale
Damage	1	Loss of A01, A02, A03. DoS for a single user possible
Reproducibility	4	Two steps required. Attacker must gain physical access to the computer
Exploitability	5	Exploit is understood, usable by moderately skilled attacker
Affected Users	0	No users affected
Discoverability	3	Attack must be targeted to user

Dread Score: 2.6 - Medium

Table 4.13: Attack 6 - Loss of A02 after access to configuration file by using physical access to a protected machine.

If Alice left her HSM connected to her computer, then there is a risk that the threat actor could gain physical access to both Alice's computer and her HSM. Access to the HSM is protected by C01, Alice's PIN. Brute forcing HSM PIN is very unlikely because a Nitrokey Start will become blocked after three incorrect guesses and further attempts will wipe the device.

This attack could go one step further but it would require a highly-sophisticated attacker. If the attacker had knowledge of an exploit to extract the keys from the HSM, then this attack could be successful but at a decreased likelihood due to the cost of mounting a successful attack on an HSM. Information disclosure of A01 and A02 would result in loss of A03 by allowing the bypass of C03. See Table 4.14.

Potential for: Information Disclosure, Spoofing, DoS

Category	Score	Rationale
Damage	9	Loss of A01, A02, A03. DoS for a single user possible
Reproducibility	0	Very hard or impossible. The vulnerability is unstable and statistically unlikely to be reliably exploited
Exploitability	1	Extreme level of technical exploitation needed. Possibly a Nation-State actor
Affected Users	5	At least two users are affected
Discoverability	1	Weakness is currently unknown but a targeted attacker with knowledge of the system is possible

Dread Score: 3.2 - Medium

Table 4.14: Attack 9 - Loss of A01 and A02 after TA02 breaks into the machine and uses sophisticated methods to extract the private key from the HSM.

Social Engineering

The indirect method of attack could be performed by TA02 using social engineering tactics, such as a phone-call or email, masquerading as someone from Alice's IT department, asking her to, in some way, disclose the contents of the WireGuard configuration file, leading to the disclosure of asset A02. See Table 4.15.

Potential for: Information Disclosure

Category	Score	Rationale
Damage	1	Loss of A01, A02, A03. DoS for a single user possible
Reproducibility	5	Two steps required. Attacker must find and convince user to disclose contents of configuration file
Exploitability	5	Little technical knowledge is required but trust or deception must be used
Affected Users	0	No users affected
Discoverability	9	Weakness is is well understood

Dread Score: 4 - High

Table 4.15: Attack 7 - Loss of A02 after access to configuration file is gained by social engineering.

The HSM has controls in place to prevent anyone from accessing the private key, so Alice cannot be convinced to disclose it. A successful social engineering attack could still be conceivably possible, if Alice was somehow convinced by the threat actor to give away physical possession of her HSM and PIN number, as well as the contents of the WireGuard-HSM configuration file. This represents a very high cost for a successful attack which decreases the likelihood that this weakness will be exploited by an attacker. See Table 4.16.

Potential for: Information Disclosure, Spoofing, DoS

Category	Score	Rationale
Damage	9	Loss of A01, A02, A03. DoS for a single user possible
Reproducibility	1	Very difficult to reproduce, may be possible once or twice
Exploitability	1	Extreme level of technical exploitation needed. Possibly a Nation-State actor
Affected Users	5	At least two users are affected
Discoverability	3	Intimate knowledge of the target would be needed to successfully target user

Dread Score: 3.8 - High

Table 4.16: Attack 8 - Loss of A01 and A02 after TA02 convinces Alice to give away her HSM and disclose the contents of her WireGuard configuration file.

4.9 Vulnerabilities

The two attacks described in 4.16 and 4.14 result in the same vulnerabilities in the system as described in section 4.4. These attacks require a high level of sophistication to carry out and come at a high-cost, but it is necessary to consider as many scenarios as possible when analyzing the system design.

Chapter 5
EVALUATION AND DISCUSSION OF RESULTS

First I will look at the performance benchmarks and final code size. Then I will discuss the results from the evaluation.

5.1 Evaluation

5.1.1 Performance Analysis

WireGuard-HSM's use of a USB-connected HSM means a small part of its functionality is embedded inside a much less powerful computer than the computer running the WireGuard-HSM program. I wanted to see what performance impacts this design might introduce to the system.

In order to benchmark the performance of the implementations, I added a timing function that saves the current time at the beginning of the derive key operations in WireGuard. The software-based function is called sharedSecret() in WireGuard and it's called DeriveNoise() for the HSM version in pkclient.

The benchmarks were run on two computers connected via a local Gigabit Ethernet connection. WireGuard-HSM ran on a desktop PC running Ubuntu 21.10 configured with an AMD Ryzen 5 2600 CPU @ 3.4GHz nominal speed and 64 GB DDR4 ram.

The peer computer ran an unmodified WireGuard client included with Ubuntu 21.10 on a machine with an Intel core i5-4590 CPU @ 3.30GHz nominal speed and 16 GB DDR3 ram.

Each function test was run until 32 function calls had been recorded for the target function, labeled DeriveNoise() for the security key and DeriveSoftware() for the default WireGuard software derive function. Each test was run 3 times and the results were compiled together for an total of 96 data points for each function. A descriptive statistic analysis performed on the resulting data set.

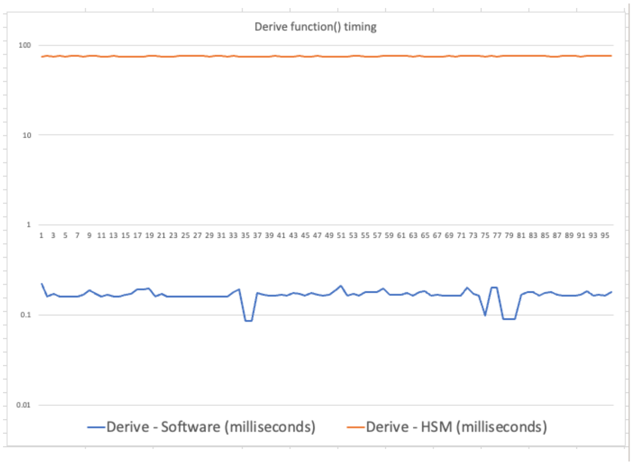


Figure 5.1: Execution Time for DeriveNoise() and softwareDerive()

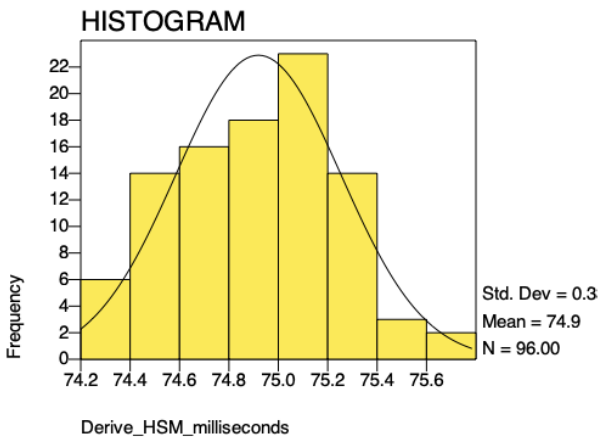


Figure 5.2: Histogram of DeriveNoise Execution Time in Milliseconds

The mean execution time for DeriveHSM() is 74.9 milliseconds and the standard deviation is 0.3 milliseconds. The variance across all data points is .11 milliseconds.

The mean execution time for softwareDerive() is 0.17 milliseconds and the standard deviation is 0.02 milliseconds. The variance across all data points was too small to be significant.

5.1.2 Code size

The total number of lines of code (LOC) changed in WireGuard-HSM when compared to WireGuard-Go is currently 104. If I only count lines of code changed to the core WireGuard protocol files, then the count drops to only 44 LOC changes. A very easy number of changes to audit and understand.

The Pkclient Golang module is 230 LOC in total which is also a small number of lines to audit and understand.

5.2 Discussion of Results

Here we will discuss the results from the previous threat model analysis and performance metrics.

5.2.1 Threat Model Comparison

Attacks 1-7 are are shared by both WireGuard and WireGuard-HSM. The impact of Attacks 1-7 on WireGuard-HSM have almost no 'damage', according to the DREAD calculations. When considering WireGuard-HSM, the vulnerabilities identified for WireGuard in section 4.4 are successfully mitigated. While the system still has a weakness that can be used to expose the public key of the peer, WireGuard-HSM manages to maintain the confidentiality of the users' private key. This demonstrates a reduction in attack-surface for WireGuard-HSM.

Table 5.2.1 shows the summary of the shared attacks for both systems.

Attack #	**WireGuard ref.**	**WireGuard-HSM**
1	6.0	3.4
2	5.6	3.0
3	5.2	2.6
4	4.6	2.0
5	5.6	3.0
6	5.2	2.6
7	6.6	4.0
Avg	5.5	2.9
σ	0.59	0.59
σ^2	0.35	0.35

Table 5.1: DREAD results for shared attacks

If we subtract WireGuard-HSM's average DREAD score from WireGuard's average DREAD score, then WireGuard-HSM reduces the average DREAD score of these shared attack scenarios by 2.6 points.

DREAD scores are a unitless measure of calculated risk. The DREAD scores help in quantifying the results in terms of risk for each system component. The higher the DREAD score for a given scenario, the more important it is to apply mitigations to that system component. Since the DREAD scores are the total average, then a reduction in 2.6 points is a significant improvement.

The results are easier to put into context if we look at the DREAD threat rating in Section 3.2. WireGuard's average DREAD score of 5.5 is rated as Critical while WireGuard-HSM's average DREAD score is rated as Medium. WireGuard-HSM's average DREAD score is reduction of severity by two levels.

In section 4.9 we considered two additional attacks for WireGuard-HSM, 4.16 and 4.14. These two attacks have an average DREAD rating of 3.5. Their risk is low but it demonstrates that no security solution is perfect. HSM vulnerabilities may be found and users must understand the possibilities of social engineering attacks. The average DREAD rating of 3.5 for these two attacks is rated as a Medium.

A successful attack would also lead to a spoofing and DoS vulnerability in WireGuard-HSM. Since the reproducibility and exploitability of those attacks is very low, I consider it an acceptable risk for the benefits to security that WireGuard-HSM demonstrated in attacks 1-7.

5.2.2 Attack Surface

The threat model analysis of WireGuard-HSM in section 4.6 shows that any attack against the system, where the attacker can read the sensitive configuration file, will no-longer result in the loss of the user's private key because of how WireGuard-HSM keeps sensitive data confidential. Since the confidentiality of the private-key is maintained the previous vulnerabilities in WireGuard a no longer exposed. This demonstrates WireGuard-HSM's significant reduction in the attack surface of the system.

The number of lines of code changed in WireGuard-HSM to the original program was minimal, less than 100 lines, making code audits fast.

5.2.3 Performance

The performance analysis shows the impact of using the Nitrokey Start, in WireGuard-HSM, is less than a tenth of a second per-handshake. If the typical use-case for WireGuard-HSM is two users establishing a session over the internet, then there is no noticeable delay when utilizing WireGuard-HSM.

5.2.4 Compatibility

All tests of WireGuard-HSM were carried out by establishing a bi-direction traffic session with a peer running a unmodified version of WireGuard on Linux. The fact that bi-directional traffic could be established and maintained, demonstrates the compatibility goal of WireGuard-HSM.

5.3 Limitations

In this section I will discuss the limitations of my project in regards to the completeness of the evaluation and the limitations of the project implementation itself. Significant time was spent to get the technical implementation of the project working, so I will discuss what improvements I would like to see, if I had more time to complete the evaluation of the entire system.

5.3.1 Formal Analysis

A formal analysis was identified as an aspirational goal for this project. If I had more time, I would perform a formal analysis of the security properties of WireGuard-HSM. The original WireGuard protocol has been verified using the Tamarin prover[8]. I would replicate this work for WireGuard-HSM and define and verify WireGuard-HSM in the Tamarin prover to create a symbolic model of the system. This would further demonstrate that the security properties of WireGuard-HSM are correctly maintained.

5.3.2 Mobile Operating Systems

I had initially planned to support mobile operating systems such as a mobile phones. Users can easily use HSMs with mobile operating systems but due to the timeout, re-key interval of the WireGuard handshake, a WireGuard-HSM user would have to leave their security key constantly connected to the phone. This requirement was deemed unreasonable so the approach was not taken. This could potentially be avoided by having users modify their WireGuard client and peer defaults to a much longer handshake timeout for re-key, but this seems like a burdensome approach and it may not be possible for the client to request a modification of peer settings.

5.3.3 HSM limitations

Adding a physical piece of hardware like an HSM into a software system adds complexity and decreases the user-convenience of the system. Another restriction is the correct physical port type must be available on the user's machine to connect the HSM. HSMs can be lost, stolen, break, or wear-out. By design, the private keys on the HSM cannot be copied, so if the HSM breaks, the user will need to purchase a new one and manually go through the out-of-band key exchange process, with a new public key.

Chapter 6
FUTURE WORK AND CONCLUSION

6.1 Future Work

There are several areas of improvement for WireGuard-HSM that have been identified over the course of implementing and evaluating the system.

6.1.1 Pin handling

When WireGuard-HSM starts, it prompts the user on the command-line for the PIN to the slot on the HSM. This prompt is easy to miss and wouldn't work very well if WireGuard-HSM was implemented as a kernel module. A better approach would be to have the prompt implemented as a system-dialog window, that appears over any existing windows, making very simple for the user to omit the pin and only enter it when using WireGuard-HSM. Finding a Golang module that opens window dialogues and supports multiple operating system was deemed out of scope for this project but it will be my next addition to the program since it would increase the ease-of-use for the program.

6.1.2 Security Key Diversity

HSM support for Curve25519 and specifically X25519 is, as of writing, very limited. Nitrokey 3 is planned to support X25519 but the hardware is still not in mass production. Many hardware-security keys have introduced support for near-field-communication (NFC) which makes using them with mobile devices almost seamless. Having a security-key with Curve25519 support, would make integrating WireGuard-HSM with mobiles devices easier and put less burden on the user.

6.1.3 Kernel-Mode Support

WireGuard-HSM is written using the Go language version of WireGuard but a kernel-mode version would be more practical for most users. The Golang-version of WireGuard warns users on start-up that if they are on a supported operating system, the kernel version has better support and offers better performance. The main challenge in writing the kernel mode version will be the PKCS#11 integration and handling user interaction between a kernel module. A small program like pkclient that handled the PKCS#11 integration and user prompts would be a good first step in expanding the functionality.

6.1.4 Biometric Authentication

It would be desirable for this project to use a biometric security key; for example, a security key that uses a small fingerprint reader to authenticate the user. A biometric security key with X25519 support is not yet offered together but they exist in independent implementations, for example Yubico sells a security key that has fingerprint reader needed for user authentication[21].

6.2 Conclusion

WireGuard-HSM helps mitigate the impact of weaknesses in the reference implementation of WireGuard by increasing the confidentiality of the user's private key. WireGuard-HSM successfully mitigates both a spoofing and DoS vulnerability found during a threat model analysis of WireGuard. WireGuard-HSM's performance impact is negligible and should be unnoticeable to users. This project offers a compelling example for more system designers to integrate HSM support for Curve25519 keys, into their projects.

BIBLIOGRAPHY

[1] Daniel J. Bernstein. Curve25519: high-speed elliptic-curve cryptography, 2005. URL: `https://cr.yp.to/ecdh.html`.

[2] Daniel J. Bernstein. SafeCurves: Introduction, March 2022. URL: `https://safecurves.cr.yp.to/`.

[3] Daniel J. Bernstein. wireguard-nt - WireGuard implementation for NT kernel, 2022. URL: `https://git.zx2c4.com/wireguard-nt/about/`.

[4] Jason A. Donenfeld. WireGuard: fast, modern, secure VPN tunnel. URL: `https://www.wireguard.com/`.

[5] Jason A. Donenfeld. WireGuard: Next Generation Kernel Network Tunnel. In *Proceedings 2017 Network and Distributed System Security Symposium*, San Diego, CA, 2017. Internet Society. URL: `https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/wireguard-next-generation-kernel-network-tunnel/`, `doi:10.14722/ndss.2017.23160`.

[6] Jason A. Donenfeld. Performance - WireGuard, 2018. URL: `https://www.wireguard.com/performance/`.

[7] Jason A. Donenfeld. Protocol & Cryptography - WireGuard, 2018. URL: `https://www.wireguard.com/protocol/`.

[8] Jason A Donenfeld and Kevin Milner. Formal Verification of the WireGuard Protocol. *None*, page 11, 2018. URL: `https://www.wireguard.com/papers/wireguard-formal-verification.pdf`.

[9] Shawn Hernan. Uncover Security Design Flaws Using The STRIDE Approach, October 2019. URL: `https://docs.microsoft.com/en-us/archive/msdn-magazine/2006/november/uncover-security-design-flaws-using-the-stride-approach`.

[10] Andreas Hülsing, Kai-Chun Ning, Peter Schwabe, Florian Weber, and Philip R. Zimmermann. Post-quantum WireGuard. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 304–321, May 2021. ISSN: 2375-1207. `doi:10.1109/SP40001.2021.00030`.

[11] IANIX. Things that use Curve25519, June 2022. URL: `https://ianix.com/pub/curve25519-deployment.html`.

[12] Komalpreet Kaur and Rishabh Sharma. Critical: Threat model for an outsourcing business. In *2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pages 1–5, July 2017. `doi:10.1109/ICCCNT.2017.8204093`.

[13] Hugo Krawczyk and Pasi Eronen. HMAC-based Extract-and-Expand Key Derivation Function (HKDF). Request for Comments RFC 5869, Internet Engineering Task Force, May 2010. Num Pages: 14. URL: `https://datatracker.ietf.org/doc/rfc5869`, `doi:10.17487/RFC5869`.

[14] Cryptsoft Pty Ltd. Cryptsoft, 2020. URL: `https://www.cryptsoft.com/pkcs11doc/`.

[15] Nitrokey. Nitrokey Start, 2022. URL: `https://shop.nitrokey.com/shop/product/nksa-nitrokey-start-6`.

[16] OpenStack. Security/OSSA-Metrics - OpenStack, June 2022. URL: `https://wiki.openstack.org/wiki/Security/OSSA-Metrics#DREAD`.

[17] OWASP. Threat Modeling - OWASP Cheat Sheet Series, July 2022. URL: `https://cheatsheetseries.owasp.org/cheatsheets/Threat_Modeling_Cheat_Sheet.html`.

[18] Pascal Sasdrich and Tim Güneysu. Implementing Curve25519 for Side-Channel–Protected Elliptic Curve Cryptography. *ACM Transactions on Reconfigurable Technology and Systems*, 9(1):3:1–3:15, November 2015. `doi:10.1145/2700834`.

[19] Yongkang Wu, Yiwei Shan, Zhichao Wang, Pengcheng Zhang, Min He, and Jihong Liu. SeWG: Security-Enhanced WireGuard for Android Based on TEE. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 1711–1717, December 2020. ISSN: 2324-9013. `doi:10.1109/TrustCom50675.2020.00235`.

[20] Yubico. Discover YubiKey 5 | Strong Authentication for Secure Login, 2021. URL: `https://www.yubico.com/products/yubikey-5-overview/`.

[21] Yubico. YubiKey Bio Series - FIDO Edition, April 2022. URL: `https://www.yubico.com/products/yubikey-bio-series/`.