Tiny WireGuard Tweak

Jacob Appelbaum, Chloe Martindale, and Peter Wu

Department of Mathematics and Computer Science
Eindhoven University of Technology, Eindhoven, Netherlands
jacob@appelbaum.net, chloemartindale@gmail.com, peter@lekensteyn.nl

Abstract. We show that a future adversary with access to a quantum computer, historic network traffic protected by WireGuard, and knowledge of a WireGuard user's long-term static public key can likely decrypt many of the WireGuard user's historic messages. We propose a simple, efficient alteration to the WireGuard protocol that mitigates this vulnerability, with negligible additional computational and memory costs. Our changes add zero additional bytes of data to the wire format of the WireGuard protocol. Our alteration provides transitional post-quantum security for any WireGuard user who does not publish their long-term static public key – it should be exchanged out-of-band.

Keywords: WireGuard · post-quantum cryptography · mass surveillance · network protocol · privacy · VPN · security

1 Introduction

WireGuard [12] is a recently introduced Virtual Private Network (VPN) protocol which is both simple and efficient. It aims to replace other protocols such as IPsec [22] and OpenVPN [44] for point-to-point tunnels with a secure protocol design that rejects cryptographic agility. WireGuard uses a fixed set of sound cryptographic primitives and does not negotiate them – in stark contrast to nearly every other major VPN protocol. Unlike many protocols, WireGuard *requires* out-of-band peer configuration information to be exchanged before it may be used. All peers *must* exchange fixed pairwise-unique long-term static public keys as well as Internet host name or address information out-of-band. WireGuard optionally allows peers to fix a pairwise-unique static symmetric value known as a Pre-Shared Key (PSK). A well-known VPN provider, Mullvad, has a worldwide deployment [31] of WireGuard that uses this PSK [32] as a method of adding post-quantum transitional security to the protocol. WireGuard does not require, nor use a PSK by default. A protocol is post-quantum

* Author list in alphabetical order; see https://www.ams.org/profession/leaders/culture/CultureStatement04.pdf. This work was done when the third author was a master student at Eindhoven University of Technology under the supervision of Jacob Appelbaum and Tanja Lange. This work was supported in part by the Netherlands Organization for Scientific Research (NWO) under grants 639.073.005 and 651.002.004 (CHIST-ERA USEIT). Permanent ID of this document: tue-wireguard-africacrypt-2019. Date of this document: May 11, 2019.

transitionally secure when it is secure against a passive adversary with a quantum computer [40]. If this transitionally secure protocol is used today, it is not possible for a quantum attacker to decrypt today's network traffic, *tomorrow*.

If a future adversary has access to a quantum computer, historic network traffic protected by WireGuard, and knowledge of *one* WireGuard user's long-term static public key, this threatens the security of the protocol for all related WireGuard users, as explained in Section 5. In this paper we propose a tiny tweak to the WireGuard protocol that makes WireGuard traffic flows secure against such an adversary; if our alteration is incorporated into the WireGuard protocol, a user's historic traffic will not be able to be decrypted by such an adversary if they do not release their long-term static public key to the network, as explained in Section 6. We accomplish this with both extremely minimal costs and minimal changes to the original protocol, as detailed in Section 6.1.

Note that our analysis applies to the current version of WireGuard [14] as implemented in the Linux kernel [18] as opposed to the older version described in the NDSS paper [12]. A major difference exists in the application of the PSK during the handshake which results in two incompatible protocols.

2 Realistic adversary concerns

It is well-documented and indisputable that a number of nation-state-sponsored adversaries are unilaterally conducting mass surveillance of the Internet as a whole. This has created new notions of realistic threat models [35,38,3] in the face of such pervasive surveillance adversaries. Some of these adversaries have an openly stated interest in *"collecting it all"* [25] and have directly stated that they use this data as actionable information, for example, for use in internationally contested drone strikes against unknown persons. The former director of the CIA, General Michael Hayden, famously said: *"We kill people based on metadata"* [9]. We additionally see that these adversaries target encrypted protocols and for example seek to exploit properties of handshakes, which may allow them to launch other subsequent attacks. These types of attacks are documented in the publication of numerous internal documents [29,30,1] that show attacks, claims, and results against a number of VPNs and other important cryptographic protocols. Development of quantum computers for attacking cryptographic protocols is explicitly a budget line item [4]. We consider it prudent to analyze WireGuard as a protocol that is, among others, of interest to these adversaries.

We consider nation-state mass surveillance adversaries (for example NSA [5,7] using XKeyscore [26]) as one of the primary adversaries to users of encrypted network tunnels, and we find that WireGuard will be vulnerable when these adversaries gain access to a quantum computer (see Section 5 for details). This is primarily due to the fact that large-scale [27] surveillance data sets which contain logged encrypted traffic are explicitly kept for later attempts at decryption [23].

We also consider less powerful adversaries which are directly coercive, oppressive, or political (COPs). These adversaries are able to take possession of any endpoint, such as through theft or other ill-gotten means, which includes a

long-term public static cryptographic key pair. This type of attack is regularly carried out against VPN providers and is commonly understood as a kind of compulsion [11] attack.

3 WireGuard overview

In this section we present an overview of the WireGuard protocol, briefly consider relevant implementations, and discuss traffic analysis considerations.

3.1 WireGuard implementations

WireGuard is implemented in multiple languages and is easy to understand. The primary implementation is available as a patch to the Linux kernel and is written in C [18]. Implementations targeting MacOS and iOS [19], Android [17], and Windows [20] use the wireguard-go [15] implementation which is written in the Go programming language. An experimental implementation in the Rust programming language is also available, wireguard-rs [16].

The first author has implemented a user space Python implementation for experimentation using Scapy [8] for use on GNU/Linux. The third author has implemented a protocol dissector [43] for WireGuard in Wireshark [10], a software program that can capture and analyze network traffic. Our implementations are based on the published WireGuard paper [12] and the evolving white paper [14].

3.2 WireGuard as a tunneling protocol

WireGuard is a point-to-point protocol for transporting IP packets. It uses the UDP protocol for transporting protocol messages. It is implemented as a device on common operating systems and users of WireGuard route IP packets into the WireGuard device to securely send those packets to their WireGuard peer. WireGuard does not have state for any IP packets that it transmits and it does not re-transmit packets if they are dropped by the network.

To start using the WireGuard protocol, a user must first generate a long-term static Curve25519 [6] key pair and acquire the long-term static public key of their respective peer. This precondition for running the WireGuard protocol is different from common Internet protocols as users *must* exchange these keys out of band. This is in contrast to services such as OpenVPN which may only need to exchange a user name or password for access control reasons. Example methods of distributing WireGuard keys include using a camera on a smart phone to import the peer public keys with a QR code, or by manually entering the data. This *must* be done before attempting to run the WireGuard protocol and the would-be agents running the protocol are designed to not emit packets to parties which do not have possession of previously exchanged public keys. Users are also required to exchange a DNS name or an IP address along with a UDP port number for at least one of the two parties. To use the WireGuard tunnel, the peers additionally have to exchange the expected *internal* IP addressing

information for their respective WireGuard tunnel endpoints. This again is in contrast to other VPN solutions which usually include some sort of automatic IP addressing scheme to ease automatic configuration of internal tunnel endpoint addresses.

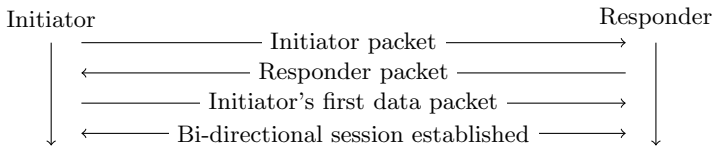

Fig. 1. Informal protocol narration of the 1.5 Round Trip Time (1.5-RTT) handshake valid for a ninety second session; parties may change roles in subsequent sessions; for additional information see Figure 7 and Algorithm 1

After configuring the endpoints with the respective public keys and IP addresses, peers will be able to create new cryptographic WireGuard sessions with each other as shown in Figure 1.

3.3 WireGuard's cryptographic handshake

The Noise Protocol framework [34] abstractly defines different Diffie-Hellman handshakes with different security, and privacy properties for use in cryptographic protocols. Protocol designers select a Noise Protocol pattern and then select the ideal abstract handshake properties. They must then select concrete objects such as an authenticated encryption scheme and a Diffie-Hellman primitive. WireGuard's cryptographic handshake [14] is a variant of IKpsk2 pattern from the Noise Protocol [34, Section 9.4] framework. A WireGuard handshake consists of the initiator sending an initiation message (see Figure 3) and the responder replying with a corresponding responder message (see Figure 4).

WireGuard selected Curve25519 [6] for Diffie-Hellman non-interactive key exchange messages, BLAKE2s [39] for hashing operations, HKDF [28] as the key derivation function (KDF), and ChaCha20Poly1305 [33] for authenticated encryption with additional data (AEAD).

WireGuard additionally augments the Noise protocol in certain areas that weaken conventional security assumptions relating to identity hiding; WireGuard reduces the identity hiding properties of the Noise IK protocol as part of a trade-off strategy to reduce computational costs and to resist detection by untargeted Internet-wide scanning. The popular Wireshark traffic analysis program displays a peer's identity and associates it with flows of traffic. We observe that preconditions of the protocol more closely resemble the Noise KK pattern; KK assumes that both parties know their peer's respective long-term static public key while IK assumes that only the responder's long-term static public key is known by the initiator. However, it is strictly weaker than the KK pattern in that the initiator

always reveals their own long-term static public key identity to the responder, and thus to the network, encrypted to the responder's long-term public key. Unlike other protocols, the roles of initiator and responder do also reverse [14]. This happens automatically when the responder attempts to send a data packet without a valid session.

3.4 Handshake details

The initiator's long-term static public key is encrypted using the ChaCha20Poly1305 AEAD using a key derived from the responder's long-term static public key and a per-session ephemeral Curve25519 key pair generated by the initiator. The resulting ciphertext is decrypted, and the public key of the initiator is found, and matched to a corresponding data structure previously initialized for cryptographic operations on the responder side; see Algorithm 1 for details. In 5.2, we describe an attack based on the transmission of the encrypted long-term static public key.

Notes on Algorithm 1:

- As in the WireGuard protocol, we use the following notation for symmetric encryption with a nonce and additional authenticated data (AEAD): $\mathsf{ciphertext} = \mathsf{aead\text{-}enc}(\mathsf{key}, \mathsf{nonce}, \mathsf{message}, \mathsf{associated\ data})$.
- Algorithm 1 gives a *simplified* version of the WireGuard key agreement process; the only fundamental simplifications that we have applied are:
  - We introduce Laura and Julian as parties in the role of Initiator and Responder.
  - Compressing the application of multiple hash function operations from $H(H(x)||y)$ to a single $H(x||y)$.
  - Omission of some constants in the initial hash and KDF salt.
  - Omission of details about construction of the 96-bit nonce. This value also serves as a counter for replay detection within a given session.
  - Compressing the application of multiple KDF's to a set of variables to the application of a single KDF to the set of variables.

4 Traffic analysis

WireGuard traffic visible to a third party observer is subject to trivial fingerprinting and confirmation that the WireGuard protocol is in use. The protocol is not designed to resist traffic analysis: session identifiers, sequence numbers, and other values are visible. For any surveillance adversary, writing a comprehensive network protocol dissector is quick work as evidenced in our Wireshark and Scapy implementations. There are four message types. Three of these types have a fixed length and each has static values which act as distinguishers or network selectors [36]. The fourth type has variable length, it additionally has static distinguishers and is linkable to other packets in any given flow. WireGuard does

Algorithm 1 Simplified WireGuard key agreement process

Public Input: Curve25519 $E/\mathbb{F}_p$, base point $P \in E(\mathbb{F}_p)$, hash function H, an empty string ϵ, key derivation function KDF_n returning n derived values index by n, and a MAC function Poly1305.

Secret Input (Laura): secret key $\mathsf{sk}_L \in \mathbb{Z}$, public key $\mathsf{pk}_L = \mathsf{sk}_L \cdot P \in E(\mathbb{F}_p)$, Julian's pre-shared public key $\mathsf{pk}_J \in E(\mathbb{F}_p)$, shared secret $\mathsf{s} = \mathsf{DH}(\mathsf{sk}_L, \mathsf{pk}_J)$, message time, PSK $Q \in \{0,1\}^{256}$; $Q = 0^{256}$ by default.

Secret Input (Julian): secret key $\mathsf{sk}_J \in \mathbb{Z}$, public key $\mathsf{pk}_J = \mathsf{sk}_J \cdot P \in E(\mathbb{F}_p)$, Laura's pre-shared public key $\mathsf{pk}_L \in E(\mathbb{F}_p)$, shared secret $\mathsf{s} = \mathsf{DH}(\mathsf{sk}_J, \mathsf{pk}_L)$, PSK $Q \in \{0,1\}^{256}$; $Q = 0^{256}$ by default.

Output: Session keys.

1: Both parties choose ephemeral secrets: $\mathsf{esk}_L \in \mathbb{Z}$ for Laura, $\mathsf{esk}_J \in \mathbb{Z}$ for Julian.
2: Laura publishes $\mathsf{epk}_L \leftarrow \mathsf{esk}_L \cdot P$.
3: Laura computes $\mathsf{se}_{JL} \leftarrow \mathsf{esk}_L \cdot \mathsf{pk}_J$; Julian computes $\mathsf{se}_{JL} \leftarrow \mathsf{sk}_J \cdot \mathsf{epk}_L$.
4: Both parties compute $(\mathsf{ck}_1, \mathsf{k}_1) \leftarrow \mathsf{KDF}_2(\mathsf{epk}_L, \mathsf{se}_{JL})$.
5: Laura computes $\mathsf{h}_1 \leftarrow H(\mathsf{pk}_J || \mathsf{epk}_L)$.
6: Laura computes and transmits $\mathsf{enc\text{-}id} \leftarrow \mathsf{aead\text{-}enc}(\mathsf{k}_1, 0, \mathsf{pk}_L, \mathsf{h}_1)$.
7: Julian decrypts enc-id with $\mathsf{aead\text{-}dec}(\mathsf{k}_1, 0, \mathsf{enc\text{-}id}, \mathsf{h}_1)$ and verifies that the resulting value (pk_L) is valid user's public key; aborts on failure.
8: Both parties compute $(\mathsf{ck}_2, \mathsf{k}_2) = \mathsf{KDF}_2(\mathsf{ck}_1, \mathsf{s})$.
9: Laura computes $\mathsf{h}_2 \leftarrow H(\mathsf{h}_1 || \mathsf{enc\text{-}id})$.
10: Laura computes and transmits $\mathsf{enc\text{-}time} \leftarrow \mathsf{aead\text{-}enc}(\mathsf{k}_2, 0, \mathsf{time}, \mathsf{h}_2)$.
11: Both parties compute $\mathsf{pkt} \leftarrow \mathsf{epk}_L || \mathsf{enc\text{-}id} || \mathsf{enc\text{-}time}$.
12: Laura computes and transmits $\mathsf{mac1} \leftarrow \mathsf{MAC}(\mathsf{pk}_J, \mathsf{pkt})$.
13: Julian verifies that $\mathsf{mac1} = \mathsf{MAC}(\mathsf{pk}_J, \mathsf{pkt})$; aborts on failure.
14: Julian computes $\mathsf{time} = \mathsf{aead\text{-}dec}(\mathsf{k}_2, 0, \mathsf{enc\text{-}time}, \mathsf{h}_2)$; aborts on failure.
15: Julian transmits $\mathsf{epk}_J \leftarrow \mathsf{esk}_J \cdot P$.
16: Laura computes $\mathsf{se}_{LJ} \leftarrow \mathsf{sk}_L \cdot \mathsf{epk}_J$; Julian computes $\mathsf{se}_{LJ} \leftarrow \mathsf{esk}_J \cdot \mathsf{pk}_L$.
17: Laura computes $\mathsf{ee} \leftarrow \mathsf{esk}_L \cdot \mathsf{epk}_J$; Julian computes $\mathsf{ee} \leftarrow \mathsf{esk}_J \cdot \mathsf{epk}_L$.
18: Both parties compute $(\mathsf{ck}_3, \mathsf{t}, \mathsf{k}_3) \leftarrow \mathsf{KDF}_3(\mathsf{ck}_2 || \mathsf{epk}_J || \mathsf{ee} || \mathsf{se}_{LJ}, Q)$.
19: Julian computes $\mathsf{h}_3 \leftarrow H(\mathsf{h}_2 || \mathsf{enc\text{-}time} || \mathsf{epk}_J || \mathsf{t})$.
20: Julian computes and transmits $\mathsf{enc\text{-}e} \leftarrow \mathsf{aead\text{-}enc}(\mathsf{k}_3, 0, \epsilon, \mathsf{h}_3)$.
21: Laura verifies that $\epsilon = \mathsf{aead\text{-}dec}(\mathsf{k}_3, 0, \mathsf{enc\text{-}e}, \mathsf{h}_3)$.
22: Both parties compute shared secrets $(T_i, T_r) \leftarrow \mathsf{KDF}_2(\mathsf{ck}_3, \epsilon)$.
23: **return** (T_i, T_r).

not attempt to hide that the WireGuard protocol is in use from a surveillance adversary, and it additionally does not attempt to hide information that allows sessions within network flows to be distinguished. WireGuard does attempt to resist active probing by requiring any initiating party to prove knowledge of the long-term static public key of the responder.

4.1 Example WireGuard protocol run

To create a WireGuard session, the protocol is broken into several phases. The initiating party is called an *initiator*, and the receiving party which must be reachable, is called the *responder*. The first phase is a handshake protocol described in detail in Section 3.3, and the second phase is a time-limited data-transfer window. The third phase is reached when a time limit or a data-transfer limit is reached, at which point a new cryptographic session is established. Unlike other cryptographic protocols, the WireGuard protocol has no session renegotiation, peers simply start again as if they have never had a session in the first place.

After a successful handshake, once the initiator has received a responder message, it may proceed to send transport data messages (see Figure 6) which contain encrypted IP packets. The responder is only permitted to send data messages after successfully receiving and authenticating the transport data packet sent by the initiator. Data messages with an encrypted empty payload act as Keep-Alive messages. These are trivially distinguishable messages by their type and length as shown in Figure 2.

Fig. 2. Flow graph between two WireGuard peers as seen in Wireshark

An example interaction taken from a packet capture between two WireGuard peers can be found in Figure 2, and an informal protocol narration in Figure 1.

If either initiator or responder are under heavy computational load, they may send a Cookie message (see Figure 5) in response to an initiation or responder message without making further progress in completing the handshake. The recipient of a Cookie message should decrypt the cookie value and use it to

calculate the MAC2 value for use in the next handshake attempt. It will not retransmit the same handshake message under any circumstances. If a handshake is unsuccessful, the initiator will try to start a new handshake.

There is no explicit error or session-tear-down signaling. A session is invalidated after a fixed duration of time; session lifetimes are currently around ninety seconds.

4.2 Packet formats

We display the four packet formats. The protocol includes only these four wire message formats, though there is an implied fifth type: an empty data message may be used as keep alive message. Each message is encapsulated entirely inside of an IP packet with UDP payload.

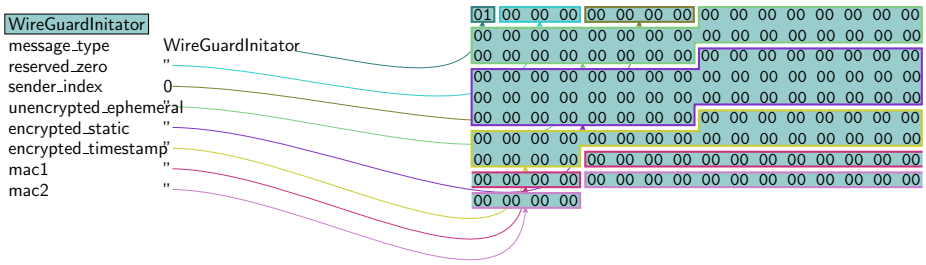

Fig. 3. 148 byte initiator packet payload

In Figure 3, the initiator message is shown. It is a fixed-size frame of 148 bytes. The MAC2 field is set to zero unless the sender has received a Cookie message before. This message is larger than the responder's message intentionally to prevent misuse such as amplification attacks using forged source addresses.

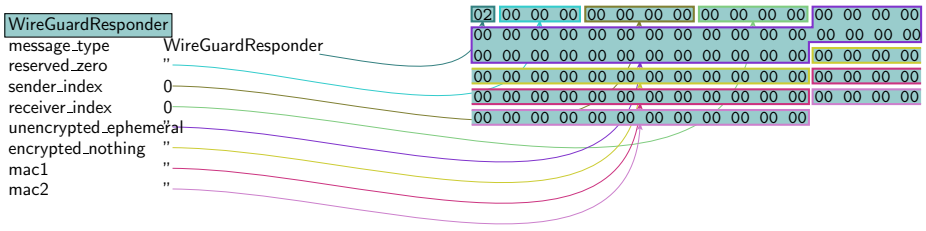

Fig. 4. 92 byte responder packet payload

In Figure 4, the responder message is shown. It is a fixed-sized frame of 92 bytes. Unlike the initiator packet, it does not contain a long term static public key.

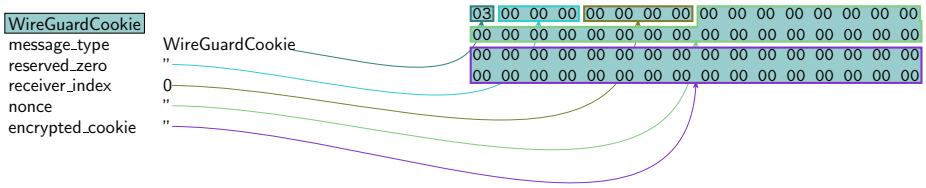

Fig. 5. 64 byte Cookie packet payload

In Figure 5, the cookie message is shown. It is a fixed-sized frame of 64 bytes. This is not used for each run of the WireGuard protocol. This message is only sent by the initiator or responder when they are "under load". The recipient must decrypt the cookie value and store it for inclusion in future handshake messages.

While all handshake messages (Figure 3, Figure 4, Figure 5) have fixed lengths, the Transport Data message (Figure 6) has a variable length. At minimum it is 32 bytes in length. This includes the Transport Data message headers and the authentication tag for the encrypted payload. For any given WireGuard protocol run, the maximum size of a generated UDP packet depends on the maximum transmission unit (MTU) of the network interface. These are typically much smaller than the theoretical limits of an IP packet.

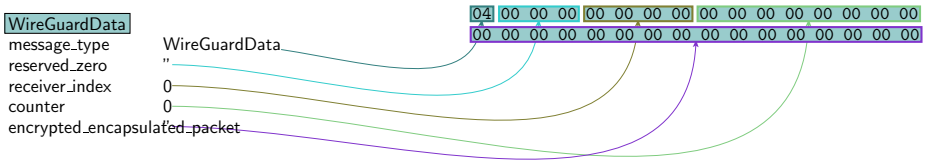

Fig. 6. Variable length (32 up to $\infty + 16$) byte data packet payload. See Table 1 for implementation specific notes

The UDP layer has a theoretical maximum length of $2^{16} - 1$, this length also includes eight bytes of the UDP header so the actual maximum length for the UDP payload is $2^{16} - 1 - 8$ bytes. The theoretical maximum length for Transport Data messages is shown in Table 1.

$2^{16} - 1 - 8$	IPv4 with fragmentation
$2^{16} - 1 - 20 - 8$	IPv4 without fragmentation nor IP options
$2^{16} - 1 - 40 - 8$	IPv6 without extension headers
$2^{32} - 1 - 40 - 8 - 8$	IPv6 with Jumbograms

Table 1. Theoretical maximum sizes for UDP payloads

While WireGuard itself does not impose a maximum length, implementations on various platforms might be constrained by their environment. For example, the Linux kernel does not support IPv6 Jumbograms [21] and FreeBSD currently does not support IPv6 Jumbograms with UDP due to the lack of a physical medium [24].

5 Security and privacy issues

We consider both the mass surveillance adversary and the less powerful local adversary conducting targeted attacks from Section 2.

Initial handshake message creation and processing

Laura	**Julian**
$\mathsf{pk}_L, \mathsf{sk}_L, \mathsf{time}$, secret key Q	$\mathsf{pk}_J, \mathsf{sk}_J$
..... Out-of-band key exchange: $\mathsf{pk}_L, \mathsf{pk}_J, PSK$ Q	
$(\mathsf{epk}_L, \mathsf{esk}_L) = \mathsf{EphemeralKey}()$	
Compute enc-id, enc-time, mac1	
epk_L, enc-id, enc-time, mac1 $\longrightarrow$ Initiator packet	
..... Responder receives initiator packet	
	Compute pkt, verify mac1
	Compute emphemeral DH
	Decrypt enc-id to a known pk
	Find session for resulting pk
	Decrypt enc-time to get time
	VerifyAntiReplay(time)
..... Handshake continues	

Fig. 7. Informal protocol narration of sending and receiving an initiator packet. (For definitions of terms and details on how to compute, decrypt, and verify, see Algorithm 1)

5.1 Identity hiding weakening

Throughout this section, suppose, as was justified in Section 2 to be a realistic situation, that a WireGuard user has released its long-term static public key. We analyze a handshake involving this user with this user in the role of responder.

The initiation packet contains the static public key of the initiator and it is encrypted as previously described with an ephemeral key pair used in conjunction with the responder's static key pair. The initiation packet is augmented with what WireGuard's design describes as a MAC. Under our assumptions, the input, which is an initiator or a responder packet, and the MAC key, which is the static public key of the receiving party, are both *public* values.

Third party observers are able to passively confirm the identity of both peers when their public keys are known to the observer. This is strictly worse than NoiseIK's identity hiding properties and allows non-sophisticated attackers to link known static public keys to individual flows of traffic.

Ostensibly the additional MAC over the whole packet is done primarily as a verification step: to prevent arbitrary packets (e.g. from an adversary) from causing the responder to compute a Diffie-Hellman key-exchange. This is a known deficiency in OpenVPN [13].

The MAC check also prevents practical Internet-wide scans from finding *unknown* WireGuard responders. While a verification step may be necessary to prevent unknown parties from exhausting resources or forcing a responder message, this additional MAC verification method is strongly divergent from the identity hiding properties of the Noise IK pattern; because of this identity hiding property, it is easier for a quantum adversary to attack, as we show below.

A simple shared secret value, set either on a per-site or per-peer basis would provide a similar protection without revealing the identity of one or both of the peers.

5.2 Quantum attack

Consider an attacker capable [1] of running Shor's algorithm [41]. Shor's algorithm breaks the discrete logarithm problem in any group in time polynomial in the size of the group; observe that this includes elliptic curve groups. Suppose that the long-term static public key of some WireGuard user U_0 is known to an adversary. We show in Algorithms 2 and 3 that in this situation, Shor's algorithm will apply to users of the WireGuard protocol, as given in Algorithm 1.

Recall from Section 4 that network traffic is visible to a third-party observer. In particular, an adversary can detect when a handshake takes place between U_0 and any other WireGuard user. We describe in Algorithm 2 how to extract the long-term static secret key of any initiator with a quantum computer when U_0 is the responder.

Of course after computing the ephemeral keys, an adversary who has access to the static secret and public keys of both the initiator and the responder of a WireGuard handshake can completely break the protocol (assuming the responder U_0 and the initiator use the default WireGuard settings, i.e. no PSK).

Now suppose an adversary wishes to attack some user U_n. Suppose also that there exists a *traceable path* from U_0 to U_n, that is, if by analyzing the

[1] See [37] for a recent estimate of the resources needed by an attacker to carry out such an attack using Shor's algorithm.

traffic flow the adversary can find users $U_1, \ldots, U_{n-1}$ for which every pair of 'adjacent' users U_i and U_{i+1} have performed a WireGuard handshake. We show in Algorithm 3 how the adversary can then compute U_n's long-term static key pair. Recall from Section 4 that the information of which pairs of users have performed a WireGuard handshake is freely available; if such a path exists then an adversary can easily find it.

An important remark on this attack: if two WireGuard users do not publish their static public keys, and *both* users do not interact with any other WireGuard users, then this attack does not apply to those two users.

Algorithm 2 Extract Initiator's Long-term Static Key Pair

Input: Long-term static public key pk_J of the responder; Ephemeral public key epk_L of the initiator (transmitted over the wire in Step 2 of Algorithm 1); $\mathsf{enc\text{-}id}$ as sent over the wire by the initiator in Step 6 of Algorithm 1.

Output: Long-term static key pair $\mathsf{sk}_L, \mathsf{pk}_L$ of the initiator.

1: Using Shor's algorithm, compute esk_L from epk_L.
2: Compute k_1 and h_1 as in Steps 4 and Steps 5 respectively of Algorithm 1.
3: Compute $\mathsf{pk}_L = \mathsf{aead\text{-}dec}(\mathsf{k}_1, 0, \mathsf{enc\text{-}id}, \mathsf{h}_1)$.
4: Compute sk_L from pk_L using Shor's algorithm.
return $\mathsf{sk}_L, \mathsf{pk}_L$.

Algorithm 3 Extract User U_n's Long-term Static Key Pair

Input: Long-term static public key of some WireGuard User U_0; A traceable path from U_0 to WireGuard User of interest U_n.

Output: Long-term static key pair of WireGuard User U_n.

1: **for** $i := 0, \ldots, n-1$ **do**
2: $U_i \leftarrow$ Responder (without loss of generality, c.f. Section 3.3).
3: $U_{i+1} \leftarrow$ Initiator (also without loss of generality).
4: Compute long-term static key pair of U_{i+1} using Algorithm 2.
5: **end for**
return Long-term static key pair of U_n.

5.3 A brief comment on extra security options

In Section 5.2 we analyzed the *default* use of the WireGuard protocol. There is an option open to WireGuard users to also preshare another secret key, i.e., to use a PSK Q as an additional input for the KDF in Step 18 of Algorithm 1. If the user does not configure a PSK, the default value ($Q = 0^{256}$) will be used.

Use of a secret PSK will not prevent a quantum adversary from computing $\mathsf{sk}_L, \mathsf{pk}_L$ using the method described in Section 5.2. It does however prevent

compromise of session keys T_i and T_r in Step 22 of Algorithm 1 as the adversary no longer has enough information to compute ck_3 in Step 18 of Algorithm 1.

A prudent user may still be concerned about an adversary stealing their PSK; the tiny protocol tweak presented in Section 6 addresses this concern as well as protecting those who use the default mode of the WireGuard protocol.

Of course our tweak cannot protect against an adversary who steals the static long-term public key of both the initiator and the responder in a WireGuard handshake.

6 Blinding flows against mass surveillance

We propose a tiny tweak to the Wireguard handshake which thwarts the quantum attack outlined in the previous section: In Step 6 and Step 7 of Algorithm 1, replace pk_L by $H(\mathsf{pk}_L)$. We suggest to use BLAKE2s as the hash function H as it is already used elsewhere in WireGuard. Naturally, the unhashed static public key pk_L of the initiator has still been exchanged out-of-band, so the responder can still perform Diffie-Hellman operations with the initiator's static public key pk_L, and is able to compute the hash $H(\mathsf{pk}_L)$. In Step 7 and Step 16 of Algorithm 1, the responder will use the decrypted value $H(\mathsf{pk}_L)$ to look up the corresponding key pk_L.

The hashing process conceals the algebraic structure of the static public key of the initiator and replaces it with a deterministic, predictable identifier. This requires no extra configuration information for either of the peers. BLAKE2s is a one-way hashing function and a quantum adversary cannot easily [42] deduce the initiator's static public or secret key from this hash value unless the hash function is broken.

An attacker as described in Section 5.2 may confirm a guess of a known long-term static public key. If the guess is correct, they may carry out the attack as in the unchanged WireGuard protocol. However, the tweak protects sessions where the public keys are not known.

We claim only *transitional security* with this alteration. That is, that a future quantum adversary will not be able to decrypt messages sent before the advent of practical quantum computers, if the messages are encrypted via an updated version of WireGuard that includes our proposed tweak. The tweaked protocol is not secure against active quantum attacks with knowledge of both long-term static public keys and a known PSK value. With knowledge of zero or only one long-term static public key, the protocol remains secure. A redesign of the WireGuard protocol to achieve full post-quantum security is still needed.

There are of course other choices of values to replace the static public key in Step 6 and Step 7 of Algorithm 1 to increase security. One alternative choice of value is an empty string, as in the case with the message sent in response to initiator packets by the responder. This would change the number of trial decryptions for the responder for initiator messages to $\mathcal{O}(n)$ where n is the number of configured peers. This change would allow any would-be attacker to force the

responder to perform many more expensive calculations. It would improve identity hiding immensely but at a cost that simply suggests using a different Noise pattern in the first place. A second alternative choice of value is a random string which is mapped at configuration time, similar to a username or a numbered account, which is common in OpenVPN and similar deployments. This provides $\mathcal{O}(1)$ efficiency in lookups of session structures but with a major loss in ease of use and configuration. It would also add a second identifier for the peer which does not improve identity hiding. Both alternative choices have drawbacks. The first method would create an attack vector for unauthenticated consumption of responder resources and the second method would require additional configuration. Both weaken the channel binding property of Noise [34, Chapter 14] as the encrypted public key of the initiator is no longer hashed in the handshake hash. The major advantage of our proposed choice is that it does not complicate configuration, nor does it require a wire format change for the WireGuard protocol. Assuming collision-resistance of the hash function, the channel binding property is also preserved. Our proposal concretely improves the confidentiality of the protocol without increasing the computation in any handshake. It increases the computation for peer configuration by only a single hash function for each configured public key.

This change does not prevent linkability of flows as it exchanges one static identifier for another, and it does preclude sharing that identifier in a known vulnerable context.

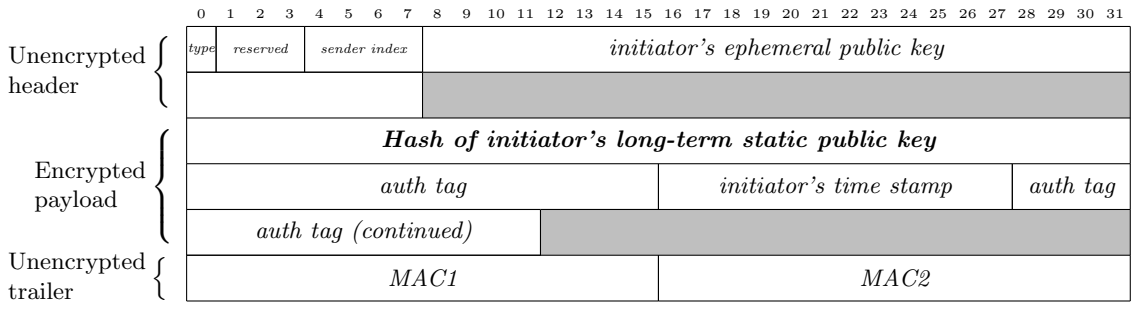

Fig. 8. Tweaked initiator packet (in bytes)

6.1 Modified protocol costs

Our modification obviously requires implementation changes. We study the effect on the proposed Linux kernel implementation as outlined in the WireGuard paper [14] as well as the effect on the alternative implementations.

The hash function input of the initiator's static public key and the output value have an identical length, thus the wire format and internal message structure definitions do not need to change to accommodate the additional hash operation.

Initiators only have a single additional computational cost, calculation of the hash over their own static public key. This could be done during each handshake at no additional memory cost, or during device configuration which only requires an additional 32 bytes of memory in the device configuration data structure to store the hash of the peer's long-term static public key.

Responders must be able to find the peer configuration based on the initiation handshake message since it includes the peer's static public key, optional PSK, permitted addresses, and so on. In the unmodified protocol, a hash table could be used to enable efficient lookups using the static public key as table key. At insertion time, a hash would be computed over the table key. The Linux kernel implementation uses SipHash2-4 [2] as hash function for this table key [14, Section 7.4]. Our modification increases the size of the per-peer data structure by 32 bytes and requires a single additional hash computation per long-term static public key at device configuration time. There are no additional memory or computational costs during the handshake.

The wireguard-go [15, device/device.go] implementation uses a standard map data type using the static public key as map key. Again, a single additional hash computation is required at configuration time with no additional memory usage.

Recall that WireGuard is based on the Noise protocol framework. Our modification is not compatible with the current version of this framework, and thus implementations that rely on a Noise library to create and process handshake messages must be changed to use an alternative Noise implementation. This affects the Rust implementation [16].

6.2 Alternative designs and future work

In theory, an alternative WireGuard implementation could accept any initiator that connects to it and successfully completes the handshake. Additional authorization could then be performed after the handshake. Our modification would make it impossible to create such implementations as it ensures that the assumed pre-condition of requiring an out-of-band exchange of long-term static public key is not violated.

Our proposed modification is generic and also applies to other protocols based on the Noise IK pattern. A new *pattern modifier* could be defined in the Noise specification that enables new protocols to improve transitional post-quantum security in the case where static public keys have been exchanged before, and only an identity selector needs to be transmitted.

7 Conclusions

We show that a future adversary with access to a quantum computer, historic network traffic protected by WireGuard, and knowledge of a WireGuard user's long-term static public key can likely decrypt many WireGuard users' historic messages when the optional PSK was not used or was compromised. We present a simple solution to this problem: hashing the long-term static public key before

it is sent encrypted over the wire, resulting in the destruction of the algebraic structure of the elliptic-curve point which otherwise could be exploited by quantum computers via Shor's algorithm. The resulting hashed public key is the same size as the original public key and does not increase the size of any of the protocol messages. The required input for a quantum adversary to run Shor's algorithm would not be available from the network flow alone and it would thwart such an attacker from using a database of network flows to decrypt those very same flows. Targeted quantum attacks would still be possible in the case that the long-term keys of both parties, initiator and responder, are known. Active quantum attacks may still be possible, but our alteration provides transitional security. Our improvement requires zero extra bytes of data transmitted on the wire, potentially zero or 32 extra bytes for each peer data structure in memory, and completely negligible computational costs for cooperating honest parties.

8 Acknowledgements

We would like to thank Jason A. Donenfeld for WireGuard and for insightful discussions about possible ways to improve WireGuard against quantum adversaries including for suggesting hashing of public keys. We would like to thank various anonymous helpers for their reviews of earlier drafts of this paper. We would also like to thank those in the TU/e coding theory and cryptology group and the cryptographic implementations group including Gustavo Banegas, Daniel J. Bernstein, and especially Tanja Lange for their valuable feedback.

References

1. Adams, A.A.: Report of a debate on Snowden's actions by ACM members. SIGCAS Computers and Society **44**(3), 5–7 (2014). https://doi.org/10.1145/2684097.2684099, https://doi.org/10.1145/2684097.2684099
2. Aumasson, J., Bernstein, D.J.: Siphash: A fast short-input PRF. In: Galbraith, S.D., Nandi, M. (eds.) Progress in Cryptology - INDOCRYPT 2012, 13th International Conference on Cryptology in India, Kolkata, India, December 9-12, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7668, pp. 489–508. Springer (2012). https://doi.org/10.1007/978-3-642-34931-7_28, https://doi.org/10.1007/978-3-642-34931-7_28
3. Barnes, R.L., Schneier, B., Jennings, C., Hardie, T., Trammell, B., Huitema, C., Borkmann, D.: Confidentiality in the Face of Pervasive Surveillance: A Threat Model and Problem Statement. RFC **7624**, 1–24 (2015). https://doi.org/10.17487/RFC7624, https://doi.org/10.17487/RFC7624
4. Barton Gellman and Greg Miller: 'Black budget' summary details U.S. spy network's successes, failures and objectives (2013), https://www.washingtonpost.com/world/national-security/black-budget-summary-details-us-spy-networks-successes-failures-and\-objectives/2013/08/29/7e57bb78-10ab-11e3-8cdd-bcdc09410972_story.html, news article

5. Bellare, M., Paterson, K.G., Rogaway, P.: Security of symmetric encryption against mass surveillance. In: Garay, J.A., Gennaro, R. (eds.) Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I. Lecture Notes in Computer Science, vol. 8616, pp. 1–19. Springer (2014). https://doi.org/10.1007/978-3-662-44371-2_1, https://doi.org/10.1007/978-3-662-44371-2_1
6. Bernstein, D.J.: Curve25519: New Diffie-Hellman Speed Records. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) Public Key Cryptography - PKC 2006, 9th International Conference on Theory and Practice of Public-Key Cryptography, New York, NY, USA, April 24-26, 2006, Proceedings. Lecture Notes in Computer Science, vol. 3958, pp. 207–228. Springer (2006). https://doi.org/10.1007/11745853_14, https://doi.org/10.1007/11745853_14
7. Bieker, F.: Can courts provide effective remedies against violations of fundamental rights by mass surveillance? The case of the United Kingdom. In: Aspinall, D., Camenisch, J., Hansen, M., Fischer-Hübner, S., Raab, C.D. (eds.) Privacy and Identity Management. Time for a Revolution? - 10th IFIP WG 9.2, 9.5, 9.6/11.7, 11.4, 11.6/SIG 9.2.2 International Summer School, Edinburgh, UK, August 16-21, 2015, Revised Selected Papers. IFIP Advances in Information and Communication Technology, vol. 476, pp. 296–311. Springer (2015). https://doi.org/10.1007/978-3-319-41763-9_20, https://doi.org/10.1007/978-3-319-41763-9_20
8. Biondi, P.: Scapy, http://www.secdev.org/projects/scapy/, website (2010)
9. Cole, D.: Michael Hayden: "we kill people based on metadata", https://www.justsecurity.org/10311/michael-hayden-kill-people-based-metadata/, David Cole quoting former director of the CIA Michael Hayden (2014)
10. Combs, G., et. al.: Wireshark (1998–2019), https://www.wireshark.org/
11. Danezis, G., Clulow, J.: Compulsion resistant anonymous communications. In: International Workshop on Information Hiding. pp. 11–25. Springer (2005), https://www.freehaven.net/anonbib/cache/ih05-danezisclulow.pdf
12. Donenfeld, J.A.: 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/
13. Donenfeld, J.A.: Wireguard Black Hat 2018 talk slides (2018), https://www.wireguard.com/talks/blackhat2018-slides.pdf, see slide 41
14. Donenfeld, J.A.: WireGuard: Next generation kernel network tunnel (2018), https://www.wireguard.com/papers/wireguard.pdf, version 416d63b 2018-06-30
15. Donenfeld, J.A.: Source code for the Go implementation of WireGuard (2019), https://git.zx2c4.com/wireguard-go, commit c2a2b8d739cb
16. Donenfeld, J.A.: Source code for the Rust implementation of WireGuard (2019), https://git.zx2c4.com/wireguard-rs, commit a7a2e5231571
17. Donenfeld, J.A.: WireGuard Android application source (2019), https://git.zx2c4.com/wireguard-android/
18. Donenfeld, J.A.: WireGuard Linux kernel source (2019), https://git.zx2c4.com/WireGuard, tag 0.0.20190227, commit ab146d92c353
19. Donenfeld, J.A.: WireGuard MacOS and iOS application source (2019), https://git.zx2c4.com/wireguard-ios/
20. Donenfeld, J.A.: WireGuard Windows application source (2019), https://git.zx2c4.com/wireguard-windows/
21. Dumazet, E.: Linux kernel patch: ipv6: Limit mtu to 65575 bytes (2014), https://git.kernel.org/linus/30f78d8ebf7f514801e71b88a10c948275168518

22. Dunbar, N.: IPsec networking standards – an overview. Inf. Sec. Techn. Report **6**(1), 35–48 (2001). https://doi.org/10.1016/S1363-4127(01)00106-6, https://doi.org/10.1016/S1363-4127(01)00106-6
23. Erwin, M.: The Latest Rules on How Long NSA Can Keep Americans' Encrypted Data Look Too Familiar (2015), https://www.justsecurity.org/19308/congress-latest-rules-long-spies-hold-encrypted-data-familiar/, blog entry
24. FreeBSD: Chapter 8. IPv6 Internals - Jumbo Payload, https://www.freebsd.org/doc/en/books/developers-handbook/ipv6.html#ipv6-jumbo
25. Greenwald, G.: The crux of the NSA story in one phrase: 'collect it all' (2013), https://www.theguardian.com/commentisfree/2013/jul/15/crux-nsa-collect-it-all, news article
26. Greenwald, G.: "XKeyscore: NSA tool collects 'nearly everything a user does on the internet' " (2013), https://www.theguardian.com/world/2013/jul/31/nsa-top-secret-program-online-data
27. Hogan, M.: Data flows and water woes: The Utah data center. Big Data & Society **2**(2), 2053951715592429 (2015), https://journals.sagepub.com/doi/abs/10.1177/2053951715592429
28. Krawczyk, H., Eronen, P.: HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC **5869**, 1–14 (2010). https://doi.org/10.17487/RFC5869, https://doi.org/10.17487/RFC5869
29. Landau, S.: Making sense from Snowden: What's significant in the NSA surveillance revelations. IEEE Security & Privacy **11**(4), 54–63 (2013). https://doi.org/10.1109/MSP.2013.90, https://doi.org/10.1109/MSP.2013.90
30. Landau, S.: Highlights from making sense of Snowden, part II: what's significant in the NSA revelations. IEEE Security & Privacy **12**(1), 62–64 (2014). https://doi.org/10.1109/MSP.2013.161, https://doi.org/10.1109/MSP.2013.161
31. Mullvad: Introducing a post-quantum VPN, Mullvad's strategy for a future problem, https://mullvad.net/en/blog/2017/12/8/introducing-post-quantum-vpn-mullvads-strategy-future-problem/, blog post
32. Mullvad: mullvad-wg-establish-psk, https://github.com/mullvad/oqs-rs/tree/master/mullvad-wg-establish-psk, source code post
33. Nir, Y., Langley, A.: ChaCha20 and Poly1305 for IETF Protocols. RFC **8439**, 1–46 (2018). https://doi.org/10.17487/RFC8439, https://doi.org/10.17487/RFC8439
34. Perrin, T.: The Noise protocol framework (2018), https://noiseprotocol.org/noise.html
35. Preneel, B.: Post-Snowden Threat Models. In: Weippl, E.R., Kerschbaum, F., Lee, A.J. (eds.) Proceedings of the 20th ACM Symposium on Access Control Models and Technologies, Vienna, Austria, June 1-3, 2015. p. 1. ACM (2015). https://doi.org/10.1145/2752952.2752978, https://doi.org/10.1145/2752952.2752978
36. Privacy and Civil Liberties Oversight Board: Report on the Surveillance Program Operated Pursuant to Section 702 of the Foreign Intelligence Surveillance Act (2014), https://www.pclob.gov/library/702-Report.pdf, july 2nd, 2014; see page 12
37. Roetteler, M., Naehrig, M., Svore, K.M., Lauter, K.E.: Quantum resource estimates for computing elliptic curve discrete logarithms. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong

Kong, China, December 3-7, 2017, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10625, pp. 241–270. Springer (2017). https://doi.org/10.1007/978-3-319-70697-9_9, https://doi.org/10.1007/978-3-319-70697-9_9
38. Rogaway, P.: The moral character of cryptographic work. IACR Cryptology ePrint Archive **2015**, 1162 (2015), http://eprint.iacr.org/2015/1162
39. Saarinen, M.O., Aumasson, J.: The BLAKE2 cryptographic hash and message authentication code (MAC). RFC **7693**, 1–30 (2015). https://doi.org/10.17487/RFC7693, https://doi.org/10.17487/RFC7693
40. Schanck, J.M., Whyte, W., Zhang, Z.: Circuit-extension handshakes for Tor achieving forward secrecy in a quantum world. Proceedings on Privacy Enhancing Technologies **4**, 219–236 (2016), https://eprint.iacr.org/2015/287.pdf
41. Shor, P.W.: Algorithms for quantum computation: Discrete logarithms and factoring. In: 35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994. pp. 124–134. IEEE Computer Society (1994). https://doi.org/10.1109/SFCS.1994.365700, https://doi.org/10.1109/SFCS.1994.365700
42. Wiener, M.J.: The full cost of cryptanalytic attacks. J. Cryptology **17**(2), 105–124 (2004). https://doi.org/10.1007/s00145-003-0213-5, https://doi.org/10.1007/s00145-003-0213-5
43. Wu, P.: Bug 15011 - Support for WireGuard VPN protocol (2018), https://bugs.wireshark.org/bugzilla/show_bug.cgi?id=15011
44. Yonan, J.: OpenVPN, https://openvpn.net/, last fetched Nov 11th, 2018