

Post Module Assignment Submission Form

MODULE TITLE: Crypto-systems & Data Protection CDP 01CE

MODULE LECTURER: Peter Norris

MODULE CODE: ES94N-15

MODULE DATES: 16/11/20 – 20/11/20

STUDENT ID NUMBER: 2086937

GPG Fingerprint:

5D0A 0274 3FF9 7554 8018 D90B 954B 6926 2E9A 9BE4

Contents

List of Figures	iii
List of Tables	iii
Abbreviations	iv
1 Pass & Merit Bands	1
1.1 IPSec Netkit Model	1
1.1.1 Design & Implementation	1
1.1.2 Assurance	4
1.2 Internal x509 Certificate Authority Hierarchy	7
1.2.1 Design & Implementation	7
1.2.2 Assurance	8
2 Merit Band	10
2.1 Apache Web Server with HTTPS	10
2.1.1 Design & Implementation	10
2.1.2 Assurance	15
3 Distinction Band	19
3.1 Evaluation of WireGuard VPN	19
3.1.1 Design & Implementation	19
3.1.2 Assurance	21
3.1.3 WireGuard Compared to IPSec	23
Appendices	29
A IPSec Tunnel Established	29
B IPSec Status with all Remote Clients	30
C WireGuard Enabling	30
D WireGuard Status with all Remote Clients	31

List of Figures

1.1	IPSec VPN	2
1.2	Before IPSec Implemented Destination Unreachable	4
1.3	After IPSec Implemented a Secure Tunnel is Created	5
1.4	Packet Capture of IPSec Secure Tunnel	5
1.5	Disabling IPSec Tunnel	6
1.6	x509 Certificate Authority Hierarchy	7
1.7	OpenSSL Command Configuration in <code>x509/create-x509.sh</code>	9
2.1	Web Server Setup	10
2.2	HTTP Packet Capture	16
2.3	HTTPS Packet Capture	17
2.4	HTTP Header Capture	18
2.5	HTTPS Header Capture	18
3.1	WireGuard VPN	19
3.2	Before WireGuard Implemented Destination Unreachable	21
3.3	Once WireGuard is Enabled	22
3.4	Packet Capture of WireGuard Communication	22
3.5	Disabling WireGuard Interface	23

List of Tables

3.1	Advantages and Disadvantages of WireGuard Compared to IPSec	26
-----	---	----

Abbreviations

Anti-Virus	AV
Authentication Header	AH
Certificate Revocation List	CRL
Certification Authority	CA
Committee on National Security Systems Policy	CNSSP
Cross Site Scripting	XSS
Denial of Service	DoS
Diffe-Hellman	DH
Distributed Denial of Service	DDoS
Encapsulating Security Payload	ESP
Extensible Authentication Protocol	EAP
GNU Privacy Guard	GPG
Hyper-Text Transfer Protocol Secure	HTTPS
Internet Key Exchange	IKE
Internet of Things	IoT
Internet Protocol	IP
Internet Protocol Security protocol	IPSec
Internet Service Provider	ISP
Man-in-the-Middle	MITM
Multi-Purpose Internet Mail Extensions	MIME
National Cyber Security Centre (UK)	NCSC
National Security Agency (US)	NSA
Open Systems Interconnection model	OSI
Operating System	OS
Payment Card Industry	PCI
Perfect Forward Secrecy	PFS
Public Key Infrastructure	PKI
Secure Sockets Layer	SSL
Security Association	SA
Transmission Control Protocol	TCP
Transport Layer Security	TLS
Virtual Private Network	VPN

1 Pass & Merit Bands

1.1 IPSec Netkit Model

1.1.1 Design & Implementation

The recent demand to shift to a remote working environment has focused this organisation on creating an infrastructure which provides systematic protection for communication between Warwick and users working from home. This infrastructure needs to allow users to access assets located on Warwick's internal domain `u2086937.cyber2020.test` in a secure manner from a remote location. To achieve this, it is recommended that a VPN is used with a new strongSwan IPSec gateway `vpn1.u2086937.cyber2020.test` located at IP address `213.1.133.98` be configured and deployed immediately. In order to effectively demonstrate a working implementation of this VPN, the Netkit User Mode Linux platform has been chosen to model this configuration. A skeleton template of the organisation's infrastructure has been used for demonstration purposes. On the left side of the VPN connection are several local users who are directly connected to Warwick's internal domain `u2086937.cyber2020.test` and represent internal Warwick assets that remote workers need to access securely. This internal domain has a default gateway of `10.11.0.1` which directs traffic to VPN 1 `vpn1.u2086937.cyber2020.test`. This VPN gateway encrypts traffic leaving Warwick's internal domain and forwards this traffic to ISP A, which connects Warwick to the Internet. On the right side of the Internet (demonstrated here by the `internet` machine) is ISP B which connects three different remote clients (representing users working from home) to the Internet. Each of these remote machines (`remote1`, `remote2`, `remote3`, `remote4`, and `remote5`) uses an IPSec Remote Access configuration whereby they can establish a tunnel with `vpn1.u2086937.cyber2020.test` to send/receive encrypted data to/from the internal Warwick domain. This infrastructure can be visualised in figure 1.1.

This basic infrastructure allows users working remotely to have a secure line of communication between themselves and the internal Warwick domain `u2086937.cyber2020.test`. Traffic leaving the domain is encrypted by the VPN gateway, sent over the public Internet, and then, once it reaches the VPN client machines, it is decrypted and made accessible to remote users. This setup creates systematic protection for communications between Warwick and users working from home. However, there are several important security configurations that must be implemented to make this setup secure and scalable. These are ranked in order of significance below.

1. IPSec mode IPSec has two main modes of operation; *tunnel* mode and *transport* mode. Transport mode encrypts IP packets at the transport layer, based on the OSI model, which leaves the source and destination IP addresses visible to third parties. To hide this communication metadata it is recommended that tunnel mode is used to create an encrypted tunnel between the two VPN endpoints. This keeps the ultimate destination IP address of internal Warwick assets confidential as a third party will only be able to see the IP addresses of the two VPN endpoints, not the ultimate destination of the IP packets. This configuration is implemented in the `/etc/ipsec.conf` file with `type=tunnel` on the VPN gateway and all remote client machines.

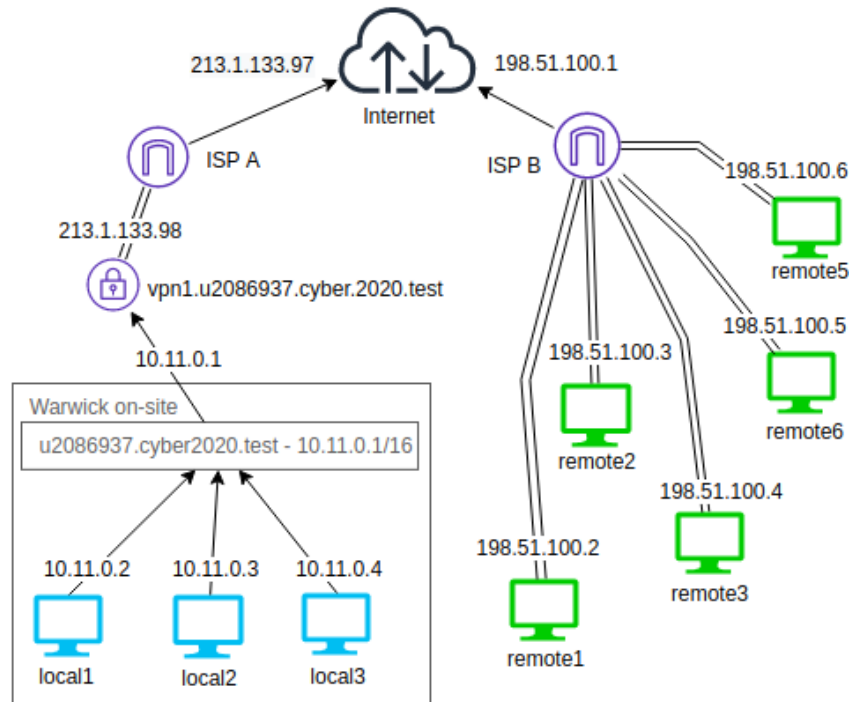


Figure 1.1: IPsec VPN

2. Cipher selection It is important to select the appropriate encryption algorithms to use for both phase one of the IPsec process – the initial Internet Key Exchange (IKE) – as well as phase two – the Encapsulating Security Payload (ESP). Both phases require the use of strong encryption algorithms as they setup, and then maintain, the tunnel between VPN endpoints. For IPsec VPNs, the NSA recommends using the following cryptographic algorithms which have been verified by the Committee on National Security System Policy (CNSSP) as of July 2nd 2020 (Barker et al., 2020) and have superseded previous NIST recommendations (Frankel et al., 2005). For IKE:

- ◇ Diffie-Hellman Group: 16
- ◇ Encryption: AES-256
- ◇ Hash: SHA-384

These are configured in the `/etc/ipsec.conf` files with `ike=aes256-sha2_384-modp4096!`. And for ESP/ISAKMP:

- ◇ Encryption: AES-256
- ◇ Hash: SHA-384
- ◇ Block Cipher Mode: CBC

These are configured in the `/etc/ipsec.conf` files with `esp=aes256-sha2_384!`. Note that an exclamation mark `!` is used at the end of each configuration to ensure that the responding endpoint uses these ciphers and not the default proposals.

3. Authorisation mechanisms The previous point on cipher selection leads into this recommendation on which authorisation mechanisms to use for authentication. IPsec lets

you configure VPNs to use Preshared Keys (PSK), other protocols like EAP, solely RSA keys, or RSA keys with certificate based authentication. This report recommends using RSA keys with certificate based authentication. Firstly, RSA keys are cryptographically stronger than PSK because they are longer, have a more entropy, and are more resistant to Man-in-the-Middle (MITM) attacks as an attacker would need to steal the private keys from both sides of the VPN tunnel (Steffen et al., 2018). The respective `/etc/ipsec.secrets` files store the private RSA keys for each side of the VPN connection. Secondly, certificate based authentication is recommend because it utilises a public key authentication mechanism that is inherently stronger when scaling than a symmetric crypto system like PSK. This Public Key Infrastructure (PKI) means certificate based authentication is cryptographically more secure than PSKs and supports trust delegation through Certificate Authorities (CAs) (Steffen et al., 2018). As such, certificate based authentication is recommend by the NCSC for most IPsec-based networks (National Cyber Security Centre, 2016b). IPsec uses the directories `/etc/ipsec.d/certs` to store the endpoint's certificate, `/etc/ipsec.d/cacerts` to store the intermediary CA's certificate, and `/etc/ipsec.d/aacerts` to store the root CA's certificate. In addition to this, certificate based authentication allows CAs to create Certificate Revocation Lists (CRLs), which is a list of certificates which have been revoked. This allows a CA to revoke a certificate that it signed if said certificate becomes compromised. For this strongSwan IPsec model CRLs are stored in the `/etc/ipsec.d/crls` directory on both the gateway and client machines. Once this model is implemented in the real world, it is recommend by this report that CRLs for end entity certificates be published every 24 hours and be valid for up to 8 days, based on the guidance of the National Cyber Security Centre (2016a). In this model that would mean manually creating and uploading the new CRLs to `vpn1.u2086937.cyber2020.test`'s `/etc/ipsec.d/crls` directory for remote clients. However, an additional CRL distribution point can be defined in the IPsec `/etc/ipsec.conf` to provide greater continuity to the VPN system. That said, adding additional secure network infrastructure is beyond the scope of this model and, as such, is left to the network architect who implements this model in the real world. To allow the network architect to do this, the code to enable CRLs is left commented out in the `/etc/ipsec.conf` file under the `config setup` section. In addition to un-commenting this code, an alternative CRL distribution point would need to be configured with similar code to:

```
ca warwick_crl
    cacert=ca_vpn.cert.pem
    crluri=http://crl2.warwick.org/ca_vpn.crl
    auto=add
```

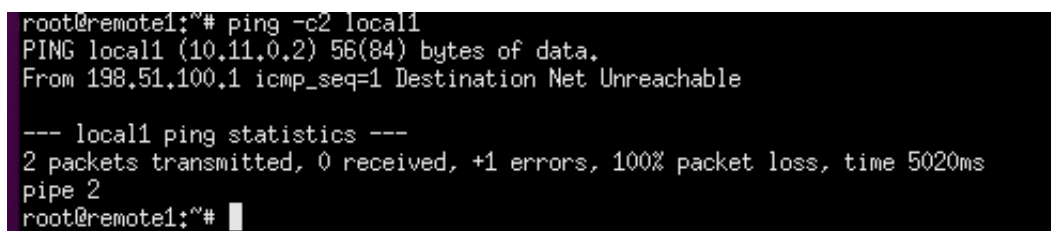
This code is also left in the `/etc/ipsec.conf` of `vpn1.u2086937.cyber2020.test` as a template. These changes can be actualised when the Netkit model is deployed in the real world.

4. Perfect Forward Secrecy In addition to using a cryptographically sound cipher, it is also important to use Perfect Forward Secrecy (PFS) when using IPsec in tunnel mode. PFS makes “endpoints negotiate a new key for each IKE and/or IPsec SA upon re-key or re-authentication event” (Steffen et al., 2018), which protects traffic if the IKE shared secret is leaked. This is configured in the `/etc/ipsec.conf` file with `ike=aes256-sha2.384-modp4096!`. In strongSwan 5.x on-wards, the IPsec VPN implementation software this model uses, appending a Diffie-Hellman (DH) group to the ESP setting automatically enables PFS (Steffen et al., 2018), as is done here.

5. Reducing the VPN gateway attack surface Another recent recommendation from the NSA for IPSec networks is to limit the gateway VPN’s attack surface by restricting accepted traffic to known VPN client/peer IP addresses when possible (Barker et al., 2020). This is possible in this model because the remote access VPN client’s have static IP addresses (for example, 198.51.100.2) so restrictions can be implemented by instantiating firewall rules which mandate connections only be made from these IP addresses (not any IP address on the Internet). In addition to this, it is recommended that the firewall is configured to limit access to UDP ports 500 (used for phase 1 of IKE) and 4500 (used for ESP) to mitigate potential attacks against IPSec. However, the model presented here is for demonstration purposes and implementing such a firewall is out of scope as it would require additional secure network architecture to be defined. Due to this limitation, this report leaves the implementation of a firewall up to the network architect who will deploy this IPSec model in the real world.

1.1.2 Assurance

Before IPSec is enabled, and an encrypted tunnel is created between the internal Warwick domain `u2086937.cyber2020.test` and a remote client, an asset connected to the internal domain is unreachable from the outside. This is shown in the figure 1.2 where machine `remote1` tries to ping an internal machine `local1`.



```
root@remote1:~# ping -c2 local1
PING local1 (10.11.0.2) 56(84) bytes of data:
From 198.51.100.1 icmp_seq=1 Destination Net Unreachable

--- local1 ping statistics ---
2 packets transmitted, 0 received, +1 errors, 100% packet loss, time 5020ms
pipe 2
root@remote1:~#
```

Figure 1.2: Before IPSec Implemented Destination Unreachable

The host `remote1` tries to ping `local1` with the command `ping -c2 local1`, however this is unsuccessful because `local1` is behind a gateway node `vpn1.u2086937.cyber2020.test` and NAT post-routing for inbound connections is not configured. This is done so internal assets are not open to generic access from the Internet. As such, clients working remotely cannot access assets inside the Warwick domain by default. In order to provide secure remote connectivity IPSec has to establish a tunnel from the gateway `vpn1.u2086937.cyber2020.test` to the remote client machine, in this example `remote1`, so packets can be transferred between local assets and remote machines. This is done by configuring IPSec on both the gateway node and any remote machines that want to connect, as described in section 1.1.1, and then enabling the encrypted tunnel on the remote machine with the command:

```
> ipsec up warwick
```

Here “warwick” is the label of the VPN connection in `remote1`’s `ipsec.config` file. This ultimately returns `connection to ‘warwick’ established successfully` once successful, but the full output from this command can be found in appendix A. Note, IPSec is automatically started on the VPN gateway machine in it’s startup file `vpn1.u2086937.cyber2020.test.startup` and remote connections through it’s `ipsec.config`.

Once IPSec has successfully created a tunnel between the remote client’s machine and the internal Warwick domain, the remote client can begin interacting with internal assets

in a secure manner. To demonstrate this a packet capture on the `internet` machine which connects `vpn1.u2086937.cyber2020.test` to each VPN client has been performed. This packet capture was done after IPsec was enabled to demonstrate the encrypted tunnel created by this protocol which allows packets to be transferred securely across the Internet. The packet capture on the `internet` machine was initiated with the command:

```
> tcpdump -s0 -v -w /hostlab/assurances/IPSec/ipsec_enabled.pcap
```

Then the packets transferred between the remote client `remote1` and the internal asset `local1` were created with the command:

```
> ping -c2 local1
```

This `ping` command was issued from `remote1` to show how an employee can securely access corporate assets from a remote environment. With IPsec enabled, the new reply to this command can be found in figure 1.3. This screenshot shows that the internal asset `local1` is now reachable and `remote1` can engage in data transfer with this asset.

```
root@remote1:~# ping -c2 local1
PING local1 (10.11.0.2) 56(84) bytes of data.
64 bytes from local1 (10.11.0.2): icmp_seq=1 ttl=63 time=0.735 ms
64 bytes from local1 (10.11.0.2): icmp_seq=2 ttl=63 time=2.63 ms

--- local1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1021ms
rtt min/avg/max/mdev = 0.735/1.684/2.633/0.949 ms
root@remote1:~# ping -c2 local1
PING local1 (10.11.0.2) 56(84) bytes of data.
64 bytes from local1 (10.11.0.2): icmp_seq=1 ttl=63 time=0.697 ms
64 bytes from local1 (10.11.0.2): icmp_seq=2 ttl=63 time=2.88 ms

--- local1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1003ms
rtt min/avg/max/mdev = 0.697/1.786/2.875/1.089 ms
root@remote1:~#
```

Figure 1.3: After IPsec Implemented a Secure Tunnel is Created

The packet capture, stored in `assurances/IPSec/ipsec_enabled.pcap`, is shown in figure 1.4 to demonstrate how this connection is secure.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	198.51.100.2	213.1.133.98	ISAKMP	762	IKE_SA_INIT MID=00 Initiator Request
2	0.055998	213.1.133.98	198.51.100.2	ISAKMP	815	IKE_SA_INIT MID=00 Responder Response
3	0.124069	198.51.100.2	213.1.133.98	ISAKMP	1290	IKE_AUTH MID=01 Initiator Request (fragment 1/3)
4	0.125339	198.51.100.2	213.1.133.98	ISAKMP	1290	IKE_AUTH MID=01 Initiator Request (fragment 2/3)
5	0.126297	198.51.100.2	213.1.133.98	ISAKMP	490	IKE_AUTH MID=01 Initiator Request (fragment 3/3)
6	0.151993	213.1.133.98	198.51.100.2	ISAKMP	1290	IKE_AUTH MID=01 Responder Response (fragment 1/3)
7	0.152181	213.1.133.98	198.51.100.2	ISAKMP	1290	IKE_AUTH MID=01 Responder Response (fragment 2/3)
8	0.152304	213.1.133.98	198.51.100.2	ISAKMP	474	IKE_AUTH MID=01 Responder Response (fragment 3/3)
9	5.058676	d6:84:84:4a:f5:f6	c6:f2:a1:74:22:74	ARP	42	Who has 213.1.133.97? Tell 213.1.133.98
10	5.058702	c6:f2:a1:74:22:74	d6:84:84:4a:f5:f6	ARP	42	213.1.133.97 is at c6:f2:a1:74:22:74
11	5.075997	c6:f2:a1:74:22:74	d6:84:84:4a:f5:f6	ARP	42	Who has 213.1.133.98? Tell 213.1.133.97
12	5.076627	d6:84:84:4a:f5:f6	c6:f2:a1:74:22:74	ARP	42	213.1.133.98 is at d6:84:84:4a:f5:f6
13	12.134583	198.51.100.2	213.1.133.98	ESP	178	ESP (SPI=0xcdd8748e)
14	12.135065	213.1.133.98	198.51.100.2	ESP	178	ESP (SPI=0xc0715dd9)
15	13.156337	198.51.100.2	213.1.133.98	ESP	178	ESP (SPI=0xcdd8748e)
16	13.157554	213.1.133.98	198.51.100.2	ESP	178	ESP (SPI=0xc0715dd9)

▶ Frame 13: 178 bytes on wire (1424 bits), 178 bytes captured (1424 bits)

▶ Ethernet II, Src: c6:f2:a1:74:22:74 (c6:f2:a1:74:22:74), Dst: d6:84:84:4a:f5:f6 (d6:84:84:4a:f5:f6)

▶ Internet Protocol Version 4, Src: 198.51.100.2, Dst: 213.1.133.98

▼ Encapsulating Security Payload

ESP SPI: 0xcdd8748e (3453514894)

ESP Sequence: 1

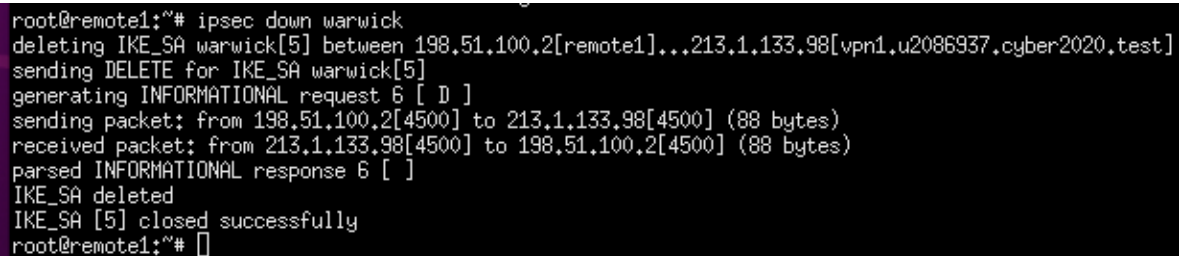
Figure 1.4: Packet Capture of IPsec Secure Tunnel

This screenshot demonstrates two things. Firstly, packet numbers 1 through 8 show the ISAKMP handshake which establishes the encrypted tunnel for `remote1` and `local1` to communicate securely across. Secondly, packets 13 (which is highlighted) and 14 show the first ICMP ping and ICMP reply packets that are exchanged between `remote1` and `local1`. However, someone who intercepts these packets cannot determine that ICMP packets have been sent because each packet is encapsulated in a Encapsulating Security Payload (ESP), thus hiding the data of the actual packet – as shown in the highlighted packet 13. In transit this ESP packet cannot be decrypted by someone who intercepts it because only the VPN gateway at `vpn1.u2086937.cyber2020.test` has the requisite key to do so, as established during the ISAKMP handshake phase. As such, the data sent from `remote1` to `local1` is secure. Also, because the IPsec tunnel protocol is being used only the VPN endpoint IP addresses can be found in the packet metadata (`198.51.100.2` for the remote client and `213.1.133.98` for Warwick's VPN gateway). Hence, the ultimate destination of packets travelling to the VPN gateway are unknown to anyone who intercepts these packets, which adds an extra layer of security.

Once the remote client concludes their business with internal Warwick assets, and no longer wants to access this domain, the encrypted tunnel can be terminated with the command:

```
> ipsec down warwick
```

This is issued by `remote1` and produces the output shown in figure 1.5 if successful.



```
root@remote1:~# ipsec down warwick
deleting IKE_SA warwick[5] between 198.51.100.2[remote1]...213.1.133.98[vpn1.u2086937.cyber2020.test]
sending DELETE for IKE_SA warwick[5]
generating INFORMATIONAL request 6 [ D ]
sending packet: from 198.51.100.2[4500] to 213.1.133.98[4500] (88 bytes)
received packet: from 213.1.133.98[4500] to 198.51.100.2[4500] (88 bytes)
parsed INFORMATIONAL response 6 [ ]
IKE_SA deleted
IKE_SA [5] closed successfully
root@remote1:~#
```

Figure 1.5: Disabling IPsec Tunnel

In this IPsec model the remote machines `remote1`, `remote2`, `remote3`, `remote4`, and `remote5` have all been configured to establish secure remote connections with the Warwick VPN gateway `vpn1.u2086937.cyber2020.test` and access assets within the `u2086937.cyber2020.test` Warwick domain. These assets are represented by local machines `local1`, `local2`, and `local3`. This setup shows that this IPsec configuration, which utilises x509 certificates for its PKI, is both secure and scalable as it can accommodate for the secure communications of multiple remote workers. Therefore, this IPsec configuration is recommended by this report. The IPsec status of `vpn1.u2086937.cyber2020.test` when all three remote clients are connected and can access internal Warwick assets can be found in appendix B.

1.2 Internal x509 Certificate Authority Hierarchy

1.2.1 Design & Implementation

To implement both a secure and scalable IPsec VPN this Netkit model utilises a public key authentication mechanism using x509 certificates. In addition to this, x509 certificates were used to protect the web transactions of Warwick's secure Apache web server. In order to create the certifications for both the IPsec VPN and the Apache web server a internal x509 certificate authority hierarchy was created for Warwick. This hierarchy allows the organisation to create, sign, distribute, and verify these certificates so that a robust Public Key Infrastructure (PKI) can be established. The x509 certificate authority hierarchy used in the Netkit model in order to satisfy the Warwick's needs is shown in figure 1.6.

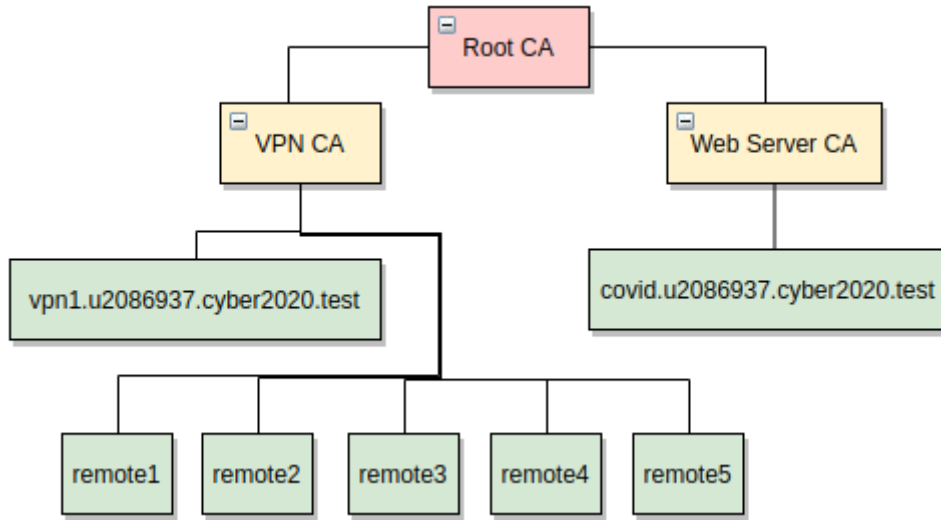


Figure 1.6: x509 Certificate Authority Hierarchy

At the top of the hierarchy is the root CA (Certification Authority). This root CA signs intermediary CA's certificates to verify that they are a legitimate Certification Authorities. Then intermediary CA's (the VPN CA and Web Server CA) are used to sign individual server/client certificates to authorise them. Individuals then use their signed certificates to establish that they are authentic by providing their public key and their certificate to an entity they are communicating with. This structure just describe, and recommended, creates a chain of certificates where a user (be it a client or server) can verify the legitimacy of another user by matching CAs on their independent chains. Intermediary CAs are used in this model to provide the organisation with business continuity by using the private key of intermediaries to sign the certificates of clients and not the root CA's private key. This means that the root CA's private key can be taken offline, used as infrequently as possible, and, if needed, be used to revoke the authority of a compromised intermediary CA by generating a Certificate Revocation List (CRL) (Nguyen, 2015). CRLs cause everything the intermediary CA has signed, or will sign in the future, to become invalid. As such, the damage caused by a key compromise only effects the certificates the intermediary CA signed, and not all the certificates the root CA signed. For example, other intermediary certificates, like the Web Server CA, will continue to be valid – along with the client certificates they have signed – despite the compromise of the VPN CA's private key. This ability to revoke an intermediary CA –

and by inheritance all the certificates it signs – allows Warwick to limit the impact of a compromised key. For this model two intermediary CA’s are used; one for VPN servers and clients and one for web servers. As Warwick currently requires certificates for these entities:

- ◇ `vpn1.u2086937.cyber2020.test`
- ◇ `remotel`
- ◇ `remote2`
- ◇ `remote3`
- ◇ `remote4`
- ◇ `remote5`
- ◇ `covid.u2086937.cyber2020.test`

These requirements may change in the future and, if so, additional intermediary CA’s can be created to sign new infrastructure assets.

Aside from using an intermediary CA for business continuity, all of these x509 certificates were created using a 4096-bit RSA key, with AES-256 symmetric encryption, to create a secure key which can be used to sign a SHA-256 signature hash. These cryptographic algorithms were used because they are not known to have any security vulnerabilities and go beyond the basic recommendations outlined by the NCSC which estimated that using 2048-bit RSA and SHA-256 would provide suitable protection for information till at least 31st December 2023 (National Cyber Security Centre, 2016b). Before this time period a new WireGuard VPN will have superseded this IPSec VPN. Therefore, implementing this recommended x509 certificate hierarchy will provide the organisation with a secure authentication mechanism to use for it’s IPSec VPN.

1.2.2 Assurance

The `x509` directory in the Netkit model contains all the certifications which need to be created to establish a secure and scalable IPSec VPN, as well as a robust HTTPS Apache web server. Along with these certificates is the file `x509/create-x509.sh` which uses OpenSSL to generate and sign all the certificates required by Warwick. Step-by-step instructions of how this is done can be found in that file. Also in each certificate’s directory are the `openssl.conf` configuration files used to create the certificate, the required RSA keys, and CRLs to revoke a certificate. These are organised into separated directories and contain the configuration options that were used to create the security recommendations stated in section 1.2.1 for each machine. For example, the code to create a 4096-bit RSA key with AES-256 symmetric encryption for the VPN CA to use, and then have the VPN CA generate a SHA-256 hash of it’s public key to be signed by the root CA to create a x509 certificate, is shown in the screenshot below in figure 1.7 and found in the `x509/create-x509.sh` file.

```
# create the vpn CA key "vpn_secret"
openssl genrsa -aes256 -out private/vpn_ca.key.pem 4096
chmod 400 private/vpn_ca.key.pem

# create the vpn cert
openssl req -config ./openssl.conf \
    -new -sha256 \
    -key private/vpn_ca.key.pem \
    -out csr/vpn_ca.csr.pem
```

Figure 1.7: OpenSSL Command Configuration in `x509/create-x509.sh`

2 Merit Band

2.1 Apache Web Server with HTTPS

2.1.1 Design & Implementation

Warwick also wants to implement a new Apache web server to support outward facing COVID aspects of the organisation’s activities. As such, this model includes an Apache web server which is outward facing named `covid.u2086937.cyber2020.test` and located at IP address `213.1.133.100`. This web server is directly connected to ISP A which is it’s route to the Internet. Note, the web server is directly on the public Internet, not behind a DNATed firewall or in a DMZ, but this will be changed in the future. In order for transactions to be secure on this web server the Netkit model uses TLS certificates to implement HTTPS. The certificate for this web server was created as part of the internal x509 certificate hierarchy previously mentioned in this report, section 1.2.1, and Figure 2.1 shows this setup.

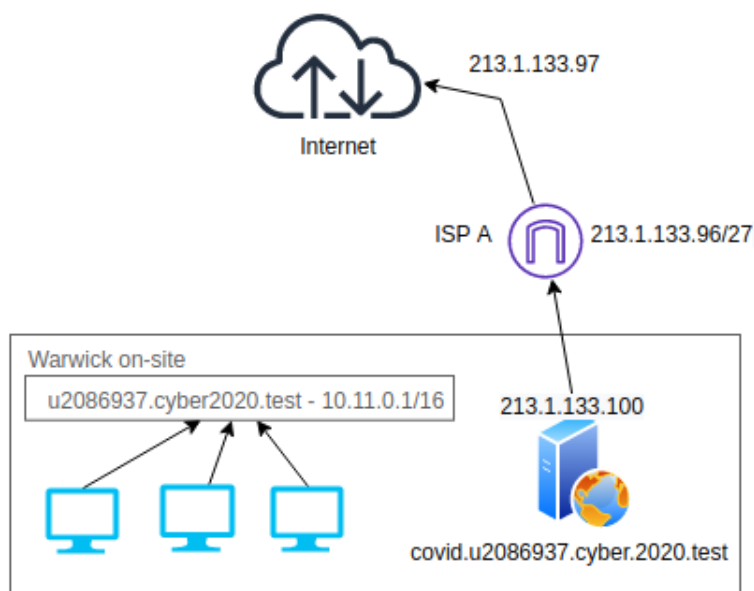


Figure 2.1: Web Server Setup

To create an Apache server which utilises HTTPS there were several configuration details that were implemented in the Netkit model:

- ◊ A default index file was created for testing purposes in `/var/www/html/index.html`
- ◊ Folders and files to store configuration modifications were created, which included:
 - `/etc/apache2/conf-available/ssl-params.conf` – this contains the TLS parameters for this web server. Importantly here are the TLS ciphers and protocols which this server allows clients to use. This web server uses strong ciphers and the most secure common TLS protocol (Kumar, 2019), all of which don’t have any known security vulnerabilities as of May 18 2020 (Russell, 2020). In terms of ciphers this means:

```
SSLCipherSuite HIGH:!MEDIUM:!aNULL:!MD5:!RC4
```

For protocols this means:

```
SSLProtocol -ALL +TLSv1.3 +TLSv1.2
```

This can be made more secure (i.e. limiting the ciphers or protocols accepted on a per page or directory basis) based on what content this web server is serving, but this content is currently unknown and it is important the web server is also accessible. Hence, TLSv1.2 and TLS 1.3 are both used, but TLS 1.3 is listed first.

- `/etc/apache2/sites-available/default-ssl.conf` – this is used to point at the web server's TLS certificate and private key.
- `/etc/apache2/ssl/certs` – this contains the x509 signed certificate for the web server.
- `/etc/apache2/ssl/private` – this contains the private key for the web server.
- `/etc/apache2/apache2.conf` – this was where the web server's domain name was specified so it matched the name on its TLS certificate.

Once these configuration files had been created the web server needed to instantiate these changes and enable TLS. To do this a simple bash script was created which contained the necessary commands. This can be found on the `covid.u2086937.cyber2020.test` machine at `/root/start-ssl.sh`. To start the Apache web server in TLS mode this script needs to be run when the Netkit model is started with `./start-ssl.sh`, and there needs to be a Internet connection to the lab (enabled in the `lab.conf` file) to download the Apache security package `mod_security`. Aside from these foundation SSL/TLS configuration details, this report looked to implement a more robust HTTPS Apache web server by adding further security configurations which are ranked in order of significance below.

1. Securing TLS To secure the TLS implementation on the Apache web server it is important to make sure all inbound traffic is directed to use HTTPS rather than the default HTTP. This is done by creating a `/etc/apache2/site-available/000-default.conf` file and implementing a configuration that applies to all virtual hosts on the Apache web server that use port 80:

```
<VirtualHost *:80>
    Redirect "/" "https://213.1.133.100/"
</VirtualHost>
```

This configuration line forces any client that connects to the web server on port 80 to be redirected to port 443 where HTTPS is being used. This is important because you want clients to be using the secure version of the site (Boucheron, 2018). To further force clients to use TLS this report recommends ensuring all headers are using TLS by instantiating the Apache Headers module and configuring the `/etc/apache2/site-available/000-default.conf` file to include:

```
<IfModule mod_headers.c>
    Header always set Strict-Transport-Security "max-age=15768000;
        includeSubDomains; preload"
</IfModule>
```

Note, that the `start-ssl.sh` script instantiates the Headers module automatically, thus running this script on startup will secure the web server's TLS configuration in this model.

2. Preventing DoS attacks The default Apache request processing model means that it is subject to a Slowloris attack – a type of DoS attack which forces Apache to wait on requests from malicious clients who take a long time to send traffic, taking up Apache resources (DreamHost, 2020). To mitigate this, the model makes use of the Apache `mod_reqtimeout` module which shuts down connections from clients that take too long to send their requests. This is configured in the `/etc/apache2/apache2.conf` file with the line:

```
RequestReadTimeout header=10-20,MinRate=500 body=20,MinRate=50
```

This allows clients with poor TCP connections to still be able to send requests, while protecting the server from the Slowloris attack (DreamHost, 2020).

3. Securing configurations There are several general security configuration which can be implemented to harden any implementation of an Apache web server, regardless of the content it is serving. These include:

- ◇ Running the Apache web server as an unprivileged user (DreamHost, 2020; Kumar, 2019). This relates to the security principle of least privilege, which states users should only be given the permissions that allow them to fulfil their role and no more. This is implemented with the following code at a Linux terminal (but done automatically in `start-ssl.sh`):

```
> groupadd apache
> useradd -g apache apache
> chown -R apache:apache /etc/apache2
> sudo vim etc/apache2/envvars
...
export APACHE_RUN_USER=apache
export APACHE_RUN_GROUP=apache
```

- ◇ Hiding Apache version and operating system. By default the web server will show the version of Apache running and the OS it is running on. This makes it easy for a potential attacker to perform “banner grabbing” and target attacks specifically for this configuration of Apache (DreamHost, 2020; Ostryzhko, 2018). Hence, it is recommended that this information is hidden by disabling `ServerTokens` and `ServerSignature` in the `/etc/apache2/conf-available/security.conf` file:

```
> sudo vim /etc/apache2/conf-available/security.conf
...
ServerTokens Prod
ServerSignature Off
```

- ◇ Disabling `.htaccess` files. These files extend the main Apache configuration file which can be convenient, but is also a security risk (DreamHost, 2020). For instance, if an attacker places one on the server it can compromise the system. To disable these files one needs to edit the `/etc/apache2/apache2.conf` file:

```
> sudo vim /etc/apache2/apache2.conf
...
<Directory /var/www/>
...
    AllowOverride None
    # or to authorise .htaccess files in AuthConfig and Indexes:
    # AllowOverride AuthConfig Indexes
```



```
...
</Directory>
```

- ◇ Disabling Apache from listing directories and following symbolic links. By default Apache will list every file and directory on the web server and follow the symbolic links between directories. This is a security vulnerability as it allows an attacker to see every file within a directory on a web server by walking directory trees (Ostryzhko, 2018). To disable this functionality, one needs to add a `-` in front of these configurations in the `/etc/apache2/apache2.conf` file:

```
> sudo vim /etc/apache2/apache2.conf
...
<Directory /var/www/>
    ...
    Options -Indexes -FollowSymLinks
    ...
</Directory>
```

- ◇ Using the open-source web application firewall Mod Security. This can be used in conjunction with Apache to provide; HTTP protection, real-time blacklist look-ups, web-based malware detection, HTTP DoS protections, common web attacks protection, automation detection, integration with AV scanning for file uploads, tracking sensitive data, Trojan protection, identification of application defects, and error detection and hiding (Kumar, 2019). To download and configure this software from a Linux terminal:

```
> sudo apt install libapache2-mod-security2
> git clone https://github.com/SpiderLabs/owasp-modsecurity-crs \
    /etc/apache2/crs
> vim /etc/apache2/apache2.conf
...
LoadModule unique_id_module modules/mod_unique_id.so
LoadModule security2_module modules/mod_security2.so
...
<IfModule *:security2_module>
    Include conf/crs/crs-setup.conf.example
    Include conf/crs/base_rules/*.conf
</IfModule>
```

This specific Mod Security configuration (“owasp-modsecurity-crs”) takes full advantage of Apache’s features by utilising core rules which are now stored in the `/etc/apache2/crs/` directory. Note, due to the ephemeral nature of the machines in this Netkit model the first command shown above `sudo apt install libapache2-mod-security2` needs to be run each time as the package is lost on reboot. This re-installation is done automatically by executing the `./start-ssl.sh` script, however the Netkit model needs to be configured to connect to the real Internet to download these packages. This is done in the `lab.conf` file with the code:

```
internet[20]=tap,192.168.0.1,192.168.0.2
```

When this model is implemented in a real world environment this will not be the case as this is a limitation of using the Netkit User Mode Linux platform.

- ◇ Disabling Trace HTTP request. By default the Trace method is enabled in Apache which opens the web server up to Cross Site Tracing attacks whereby an attack can

steal cookie information (Ostryzhko, 2018; Kumar, 2019). To prevent this, one can add the following to the `/etc/apache2/conf-available/security.conf` file:

```
TraceEnable off
```

- ◇ Disabling `Etag`. The web server's Etag header can be used by an attacker to gather sensitive information, like inode number, multi-part MIME boundary, and child processes – this needs to be fixed for the web server to be PCI compliant (Kumar, 2019). This can be done by adding the following to the `/etc/apache2/apache2.conf` file:

```
FileETag None
```

- ◇ Set cookie with `HttpOnly` and `Secure` flag. Setting these configurations mitigates against the most common Cross Site Scripting (XSS) attacks and prevents attackers manipulating web application sessions and cookies (Kumar, 2019). To do this, one first needs to ensure `mod_headers.so` is enabled (this is done in the `/root/start-ssl.sh` file with `a2enmod headers`) and add the following to the `/etc/apache2/apache2.conf` file:

```
Header edit Set-Cookie ^(.*)$ $1;HttpOnly;Secure
```

- ◇ Additional XSS protections. With XSS being a common web application security risk (OWASP, 2017), it is recommended that further security configurations are enabled on the Apache web server to mitigate this risk (Kumar, 2019). This can be done by adding the following to the `/etc/apache2/apache2.conf` file:

```
Header set X-XSS-Protection "1; mode=block"
```

- ◇ Preventing clickjacking. Clickjacking is a common web application vulnerability where an attacker gets a user to click on something different from what they perceive by using multiple transparent/opaque layers (Kumar, 2019; OWASP, 2020). This can be mitigated by adding the following to the `/etc/apache2/apache2.conf` file:

```
Header always append X-Frame-Options SAMEORIGIN
```

- ◇ Disable HTTP 1.0 Protocol. This older HTTP version is susceptible to session hijacking attacks and clients should be using HTTP 1.1 or above (Kumar, 2019). To disable this HTTP protocol you need to first enable the `mod_rewrite` Apache module (this is done in the `/root/start-ssl.sh` file with `a2enmod rewrite`) and add the following to the `/etc/apache2/apache2.conf` file:

```
RewriteEngine On
RewriteCond %{THE_REQUEST} !HTTP/1.1$
RewriteRule .* - [F]
```

These additional security configurations make this Apache web server robust (even if it was required to use HTTP instead of HTTPS one day), however there are further security configurations that can be implemented once the content the web server is hosting is known. For instance:

- ◇ Limiting file upload size if the website allows files to be uploaded.

-
- ◇ Turning off server side includes and CGI execution if the website does not require this functionality.
 - ◇ Restricting access to certain directories on the website if there are administration pages. This can be done by IP address or by using a password.
 - ◇ Allowing only certain HTTP request methods.
 - ◇ Disabling modules the website does not make use of.
 - ◇ Enabling Apache logging which is independent of, and provides more information than, system OS logging.

Above all of these security configurations is the requirement of the web server administrator to keep Apache up-to-date with security fixes once the website goes live.

2.1.2 Assurance

In order to demonstrate that the security configurations have been correctly implemented for `covid.u2086937.cyber2020.test` this report utilises packet captures taken from the `internet` machine in this Netkit model. A simple `wget` request for the default `index.html` page was made from machine `remotel` at `198.51.100.2`, which simulated the role of a generic client on the Internet, and the packets travelling between the web server and the client were captured for analysis. This was done both before and after HTTPS, and the additional security configurations, had been implemented. The commands were as follows:

- ◇ The packet capture was setup on `internet` with:

```
> tcpdump -s0 -v -w /hostlab/assurances/HTTPS/<filename>
```

- ◇ The request **before** HTTPS implementation was sent with:

```
wget http://213.1.133.100/
```

- ◇ The request **after** HTTPS implementation was sent with:

```
wget https://213.1.133.100/
```

Before HTTPS Implemented. Before the outlined security configurations were implemented, the Apache web server `covid.u2086937.cyber2020.test` located at `213.1.133.100` used the basic, insecure HTTP protocol to transfer data to clients. The packets captured while this was the case were store in `assurances/HTTPS/http.dump.pcap` and a screenshot of this can be found below in figure 2.2.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	ce:4c:08:74:8e:85	ce:4c:08:74:8e:85	ARP	42	Who has 213.1.133.100? Tell
2	0.001073	ce:4c:08:74:8e:85	ce:4c:08:74:8e:85	ARP	42	213.1.133.100 is at ce:4c:08
3	0.004929	ce:4c:08:74:8e:85	ce:4c:08:74:8e:85	ARP	42	Who has 213.1.133.97? Tell 2
4	0.004954	ce:4c:08:74:8e:85	ce:4c:08:74:8e:85	ARP	42	213.1.133.97 is at ce:4c:08
5	1.930047	198.51.100.2	213.1.133.100	TCP	74	54962 → 80 [SYN] Seq=0 Win=6
6	1.930744	213.1.133.100	198.51.100.2	TCP	74	80 → 54962 [SYN, ACK] Seq=0
7	1.931139	198.51.100.2	213.1.133.100	TCP	66	54962 → 80 [ACK] Seq=1 Ack=1
8	1.931480	198.51.100.2	213.1.133.100	HTTP	206	GET / HTTP/1.1
9	1.931768	213.1.133.100	198.51.100.2	TCP	66	80 → 54962 [ACK] Seq=1 Ack=1
10	1.932959	213.1.133.100	198.51.100.2	HTTP	643	HTTP/1.1 200 OK (text/html)
11	1.933104	198.51.100.2	213.1.133.100	TCP	66	54962 → 80 [ACK] Seq=141 Ack=
12	1.938321	198.51.100.2	213.1.133.100	TCP	66	54962 → 80 [FIN, ACK] Seq=14
13	1.938784	213.1.133.100	198.51.100.2	TCP	66	80 → 54962 [FIN, ACK] Seq=57
14	1.940316	198.51.100.2	213.1.133.100	TCP	66	54962 → 80 [ACK] Seq=142 Ack=

▶ Frame 10: 643 bytes on wire (5144 bits), 643 bytes captured (5144 bits)
 ▶ Ethernet II, Src: ce:4c:08:74:8e:85 (ce:4c:08:74:8e:85), Dst: ce:4c:08:74:8e:85 (ce:4c:08:74:8e:85)
 ▶ Internet Protocol Version 4, Src: 213.1.133.100, Dst: 198.51.100.2
 ▶ Transmission Control Protocol, Src Port: 80, Dst Port: 54962, Seq: 1, Ack: 141, Len: 577
 ▶ Hypertext Transfer Protocol
 ▶ Line-based text data: text/html (12 lines)
 <!DOCTYPE html>\n
 <html lang="en">\n
 <head>\n
 <meta charset="UTF-8">\n
 <meta name="viewport" content="width=device-width, initial-scale=1.0">\n
 <title>Welcome to Warwick CSC</title>\n
 </head>\n
 <body>\n
 \t<h1>u2086937.cyber2020.test Warwick Website</h1>\n
 \n
 </body>\n
 </html>\n

Figure 2.2: HTTP Packet Capture

Packet numbers 5 to 7 in this figure show the two machines, the client and web server, engaging in the three-way TCP handshake to establish a TCP connection. Once this is complete, packet number 8 shows the client making a HTTP request for the default `index.html` web page with `GET /`. Packet number 9 shows the server acknowledging this request with a `ACK` flagged TCP packet, and then sending `index.html` in clear text to the client in packet number 10. This packet has been highlighted in the screenshot with the 12 lines of clear text which were captured being shown. Hence, this packet capture demonstrates how an Apache web server, not configured to use HTTPS, sends traffic in the clear by default using HTTP.

After HTTPS Implemented. After the outlined security configurations were implemented, the Apache web server `covid.u2086937.cyber2020.test` located at `213.1.133.100` now uses the secure HTTPS protocol to transfer data to clients by using encryption. The new packet capture is stored in `assurances/HTTPS/https_dump.pcap` and a screenshot of this file is shown below in figure 2.3. Note, to initiate this implementation one must run the script located on the `covid.u2086937.cyber2020.test` at `/root/start-ssl.sh` and make sure the Netkit model is connected to the real Internet in the `lab.conf` file as discussed in section 2.1.1.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	198.51.100.2	213.1.133.100	TCP	74	47240 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SAC
2	0.000118	213.1.133.100	198.51.100.2	TCP	74	443 → 47240 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 M
3	0.000198	198.51.100.2	213.1.133.100	TCP	66	47240 → 443 [ACK] Seq=1 Ack=1 Win=64240 Len=0 TSval=
4	0.024138	198.51.100.2	213.1.133.100	TLSv1.3	583	Client Hello
5	0.026280	213.1.133.100	198.51.100.2	TCP	66	443 → 47240 [ACK] Seq=1 Ack=518 Win=64704 Len=0 TSva
6	0.047305	213.1.133.100	198.51.100.2	TLSv1.3	1514	Server Hello, Change Cipher Spec, Application Data
7	0.047409	213.1.133.100	198.51.100.2	TLSv1.3	1441	Application Data, Application Data, Application Data
8	0.047456	198.51.100.2	213.1.133.100	TCP	66	47240 → 443 [ACK] Seq=518 Ack=1449 Win=64088 Len=0 T
9	0.047508	198.51.100.2	213.1.133.100	TCP	66	47240 → 443 [ACK] Seq=518 Ack=2824 Win=63488 Len=0 T
10	0.074952	198.51.100.2	213.1.133.100	TLSv1.3	72	Change Cipher Spec
11	0.075082	213.1.133.100	198.51.100.2	TCP	66	443 → 47240 [ACK] Seq=2824 Ack=524 Win=64704 Len=0 T
12	0.076762	198.51.100.2	213.1.133.100	TLSv1.3	140	Application Data
13	0.076863	213.1.133.100	198.51.100.2	TCP	66	443 → 47240 [ACK] Seq=2824 Ack=598 Win=64640 Len=0 T
14	0.077862	213.1.133.100	198.51.100.2	TLSv1.3	145	Application Data
15	0.078020	198.51.100.2	213.1.133.100	TCP	66	47240 → 443 [ACK] Seq=598 Ack=2903 Win=64088 Len=0 T
16	0.078332	213.1.133.100	198.51.100.2	TLSv1.3	145	Application Data
17	0.078484	198.51.100.2	213.1.133.100	TCP	66	47240 → 443 [ACK] Seq=598 Ack=2982 Win=64088 Len=0 T
18	0.079344	198.51.100.2	213.1.133.100	TLSv1.3	228	Application Data
19	0.079555	213.1.133.100	198.51.100.2	TCP	66	443 → 47240 [ACK] Seq=2982 Ack=760 Win=64512 Len=0 T
20	0.085526	213.1.133.100	198.51.100.2	TLSv1.3	825	Application Data
21	0.085685	198.51.100.2	213.1.133.100	TCP	66	47240 → 443 [ACK] Seq=760 Ack=3741 Win=64088 Len=0 T
22	0.112625	198.51.100.2	213.1.133.100	TCP	66	47240 → 443 [FIN, ACK] Seq=760 Ack=3741 Win=64088 Le
23	0.113046	213.1.133.100	198.51.100.2	TLSv1.3	90	Application Data
24	0.113236	213.1.133.100	198.51.100.2	TCP	66	443 → 47240 [FIN, ACK] Seq=3765 Ack=761 Win=64512 Le
25	0.113483	198.51.100.2	213.1.133.100	TCP	54	47240 → 443 [RST] Seq=761 Win=0 Len=0

```

▶ Frame 14: 145 bytes on wire (1160 bits), 145 bytes captured (1160 bits)
▶ Ethernet II, Src: 06:62:e1:09:b1:93 (06:62:e1:09:b1:93), Dst: c6:f2:a1:74:22:74 (c6:f2:a1:74:22:74)
▶ Internet Protocol Version 4, Src: 213.1.133.100, Dst: 198.51.100.2
▶ Transmission Control Protocol, Src Port: 443, Dst Port: 47240, Seq: 2824, Ack: 598, Len: 79
▼ Transport Layer Security
  ▼ TLSv1.3 Record Layer: Application Data Protocol: http-over-tls
    Opaque Type: Application Data (23)
    Version: TLS 1.2 (0x0303)
    Length: 74
    Encrypted Application Data: 415b69b0cc2992a75008ef6de2ba22d0e6135dc2ede6e7b2...

```

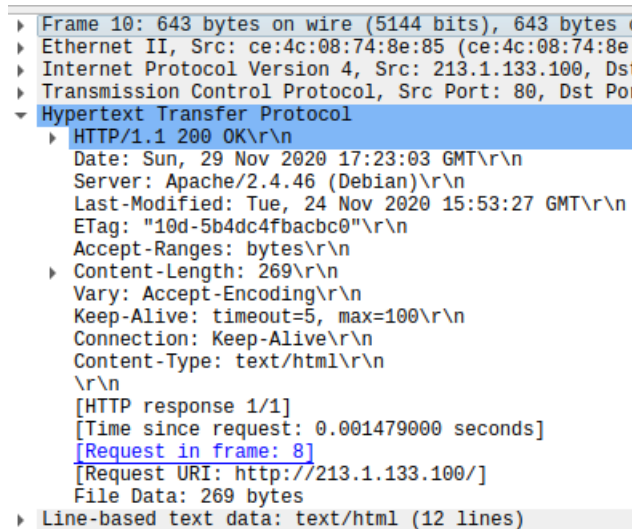
Figure 2.3: HTTPS Packet Capture

In this screenshot the TCP three-way handshake takes place in packet numbers 1 to 3. Once this is complete a TLSv1.3 handshake begins (TLSv1.3 was defined in `/etc/apache2/conf-available/ssl-params.conf` as being the preferred TLS protocol accepted by the web server as it is the most secure), which continues through packet numbers 4 to 11 with the final TCP acknowledgement being sent by the client. Then in packet number 12 the client (198.51.100.2) makes the same request before `GET /`, but this time it is over HTTPS as oppose to HTTP. As a result, packet number 12 just shows “Application Data” is being sent over TLSv1.3. The web server replies to this by sending back to the client `index.html` over several “Application Data” packets using HTTPS which is shown in the highlighted packet number 14. This time inspection of the packet only reveals that TLSv1.3 was used to send application data using “http-over-tls” as the data sent is encrypted. Hence, this packet capture demonstrates how configuring the Apache web server to use HTTPS ensures traffic is sent encrypted to a client, which makes it undecipherable to an intruder capturing packets.

Note, for `remote1` to recognise the `covid.u2086937.cyber2020.test` x509 certificate as being legitimate, and `wget` not throwing a warning, the web server certificates `covid.u2086937.cyber2020.test.cert.crt` and the root CA certificate `ca.cert.crt` need to be stored on the `remote1` machine. This is achieved for all the machines in this model by using the `shared` machine which contains the `usr/local/share/ca-certificates/` directory where the aforementioned certificates are stored. However, due to the configuration of the Netkit model every time you want to use the certificates they must be loaded with `update-ca-certificates` from the machine you want to use them on. For instance, issuing `update-ca-certificates` on the `remote1` machine means that the x509 certificate that

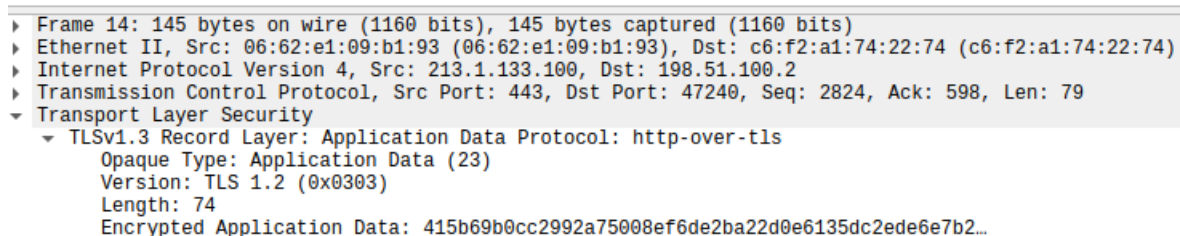
`covid.u2086937.cyber2020.test` is using will now be recognised as legitimate when HTTPS requests are made from this machine. When this model is deployed in the real world this will not be the case as the certificates will be stored permanently or a recognised CA (not a self-signed CA) will be paid to sign the certificates.

Additional Security Configurations. Aside from how the Apache web server sends HTTP data traffic in the clear by default, it is also important to analyse the HTTP header it responds with to HTTP requests from clients. Below in figure 2.4 is a screenshot of the default HTTP packet header that Apache responds with to clients. This HTTP header reveals potentially compromising information about the web server which is serving the client. For instance, it reveals configuration details in it's signature (version and OS), it shows that it does not protect against XSS, one can see the server's E-Tag, and it indicates it is susceptible to clickjacking attacks. All of this is just in the server's header. Once HTTPS has been implemented, along with the security configurations outlined above, the new header is shown in figure 2.5 as a screenshot. This header reveals nothing about the server because it is encrypted using TLSv1.3, thus significantly reduces the attack surface of the web server.



```
▶ Frame 10: 643 bytes on wire (5144 bits), 643 bytes captured (5144 bits) on interface 0
▶ Ethernet II, Src: ce:4c:08:74:8e:85 (ce:4c:08:74:8e:85), Dst: 08:00:27:00:00:00
▶ Internet Protocol Version 4, Src: 213.1.133.100, Dst: 198.51.100.2
▶ Transmission Control Protocol, Src Port: 80, Dst Port: 443
▼ Hypertext Transfer Protocol
  ▶ HTTP/1.1 200 OK\r\n
    Date: Sun, 29 Nov 2020 17:23:03 GMT\r\n
    Server: Apache/2.4.46 (Debian)\r\n
    Last-Modified: Tue, 24 Nov 2020 15:53:27 GMT\r\n
    ETag: "10d-5b4dc4fbacbc0"\r\n
    Accept-Ranges: bytes\r\n
    Content-Length: 269\r\n
    Vary: Accept-Encoding\r\n
    Keep-Alive: timeout=5, max=100\r\n
    Connection: Keep-Alive\r\n
    Content-Type: text/html\r\n
    \r\n
    [HTTP response 1/1]
    [Time since request: 0.001479000 seconds]
    [Request in frame: 8]
    [Request URI: http://213.1.133.100/]
    File Data: 269 bytes
  ▶ Line-based text data: text/html (12 lines)
```

Figure 2.4: HTTP Header Capture



```
▶ Frame 14: 145 bytes on wire (1160 bits), 145 bytes captured (1160 bits) on interface 0
▶ Ethernet II, Src: 06:62:e1:09:b1:93 (06:62:e1:09:b1:93), Dst: c6:f2:a1:74:22:74 (c6:f2:a1:74:22:74)
▶ Internet Protocol Version 4, Src: 213.1.133.100, Dst: 198.51.100.2
▶ Transmission Control Protocol, Src Port: 443, Dst Port: 47240, Seq: 2824, Ack: 598, Len: 79
▼ Transport Layer Security
  ▼ TLSv1.3 Record Layer: Application Data Protocol: http-over-tls
    Opaque Type: Application Data (23)
    Version: TLS 1.2 (0x0303)
    Length: 74
    Encrypted Application Data: 415b69b0cc2992a75008ef6de2ba22d0e6135dc2ede6e7b2...
```

Figure 2.5: HTTPS Header Capture

3 Distinction Band

3.1 Evaluation of WireGuard VPN

3.1.1 Design & Implementation

It has also been proposed by the organisation that in 6 months time the strongSwan IPsec gateway `vpn1.u2086937.cyber2020.test` is superseded by a WireGuard VPN gateway `vpn2.u2086937.cyber2020.test` at `213.1.133.99`. To determine if this new technology is fit for purpose, a WireGuard VPN configuration has been implemented in the Netkit model, in addition to the IPsec VPN, to evaluate the two VPN technologies. This WireGuard implementation can be substituted for the IPsec implementation by configuring which VPN gateway is run from within the `lab.conf` file. This is because the WireGuard VPN implementation has a similar infrastructure setup to the IPsec implementation which is shown in the figure 3.1 below.

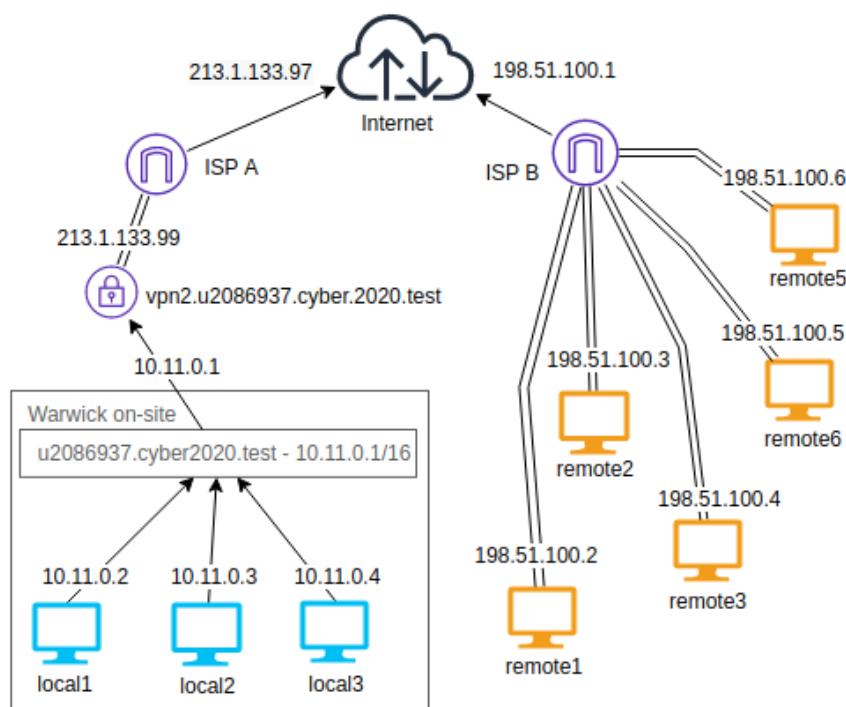


Figure 3.1: WireGuard VPN

In this diagram, there is Warwick's internal domain `u2086937.cyber2020.test` which now connects to the new WireGuard VPN gateway `vpn2.u2086937.cyber2020.test` through the default gateway IP address `10.11.0.1` (the same default gateway as before). This WireGuard VPN then connects to the Internet (represented by the `internet` machine in this Netkit model) through ISP A via its external IP address `213.1.133.99`. Then, exactly like the IPsec VPN, remote clients connect through their ISP (ISP B) to the Internet to converse with the WireGuard VPN gateway. Although the underlying infrastructure and remote access (road warrior) configuration is similar, the technology used to enable secure communications between assets on the internal Warwick domain (`local1`, `local2`,

and `local3`) and remote clients (`remote1`, `remote2`, `remote3`, `remote4`, and `remote5`) is very different.

WireGuard uses 256-bit public and private keys to secure communications between VPN endpoints and remote clients. These can be created with the `wg` WireGuard tool with the following code:

```
> umask 077
> wg genkey | tee privatekey | wg pubkey > publickey
```

The `umask` command is used here to change the default permissions of the keys generated so only the Owner has read, write, execute permissions; however, the Netkit software used to create this network model can change file permissions so to combat this `chmod 700 *` may need to be invoked from within the `/etc/wireguard/` directory on both the VPN gateway and remote client machines. A private and public key needs to be created for the WireGuard VPN gateway used by Warwick, `vpn2.u2086937.cyber2020.test`, and each client that wants to connect to this endpoint (called “peers” in WireGuard literature), like `remote1`. These keys are then used in each node’s `/etc/wireguard/wg0.conf` configuration file to authenticate themselves and the nodes that they are connecting with, much like x509 certificates were used in the IPsec configuration in section 1.1.1. The default WireGuard setup creates a site-to-site (or peer-to-peer) encrypted tunnel for communications that connects each node’s special WireGuard network interface (i.e. `wg0`, `wg1`, etc.) to the other WireGuard interfaces creating an internal IP network. This produces a “cryptokey routing” system where each internal IP address is associated with a public key which is used to route traffic through the correct tunnel and to the create destination. However, for Warwick a road warrior configuration needs to be implemented to allow remote workers to access assets within the internal Warwick domain `u2086937.cyber2020.test`. To do this, IP forwarding needs to be enabled by configuring an `iptables` firewall in the VPN gateway `/etc/wireguard/wg0.conf` configuration file that allows traffic to be forwarded to and from the internal Warwick domain. This is done for IPv4 with the following lines of code (Linode Community, 2020):

```
PostUp = iptables -A FORWARD -i wg0 -j ACCEPT; \
        iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
PostDown = iptables -D FORWARD -i wg0 -j ACCEPT; \
        iptables -t nat -D POSTROUTING -o eth1 -j MASQUERADE
```

Aside from configuring IP forwarding in the VPN gateway, each WireGuard endpoint needs to configure what IP addresses are allowed to sent packets through the encrypted tunnel it creates with other endpoints. This is done in each endpoint’s configuration files under the corresponding `[Peer]` section(s). For instance, for remote clients they only should accept traffic from the internal Warwick domain, with:

```
[Peer]
...
AllowedIPs = 10.11.0.1/16
```

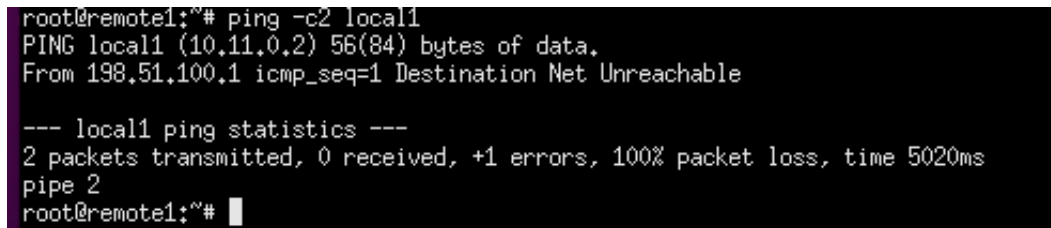
Configuring allowed IP addresses adds security to the WireGuard VPN setup as only clients who have IP addresses within this IP range will be able to use this secure channel to communicate with this remote client. Note `10.11.0.1/16` is the CIDR of the internal Warwick domain.

Apart from these basic implementation details, which turn the WireGuard VPN into a road warrior setup, there are no other security configurations to implement. This

juxtaposes IPsec and the advantages and disadvantages of this are discussed in section 3.1.3.

3.1.2 Assurance

Similar to IPsec in section 1.1.2, without the WireGuard VPN instantiated remote clients cannot communicate with assets internal to the Warwick domain. Thus, as before, a ICMP ping from a remote client generates the same “Destination Net Unreachable” error message. This is shown below in figure 3.2.



```
root@remote1:~# ping -c2 local1
PING local1 (10.11.0.2) 56(84) bytes of data:
From 198.51.100.1 icmp_seq=1 Destination Net Unreachable

--- local1 ping statistics ---
 2 packets transmitted, 0 received, +1 errors, 100% packet loss, time 5020ms
pipe 2
root@remote1:~#
```

Figure 3.2: Before WireGuard Implemented Destination Unreachable

Hence, to allow remote clients to communicate with internal assets WireGuard needs to be configured, as detailed in section 3.1.1, and instantiated on both the VPN gateway `vpn2.u2086937.cyber2020.test` and on the remote client that wants to connect (i.e. `remote1`). This is done by enabling each remote client machine’s `wg0` interface (`wg0` is the WireGuard interface configured in this Netkit model) with the command:

```
> wg-quick up wg0
```

This initiates WireGuard on the remote clients, however WireGuard also needs to be initiated on the VPN gateway. For ease-of-use this has been done by starting WireGuard as a service on the VPN gateway `vpn2.u2086937.cyber2020.test` in its startup file `vpn2.u2086937.cyber2020.test.startup` so that WireGuard is automatically started on boot. The output generated when instantiating WireGuard on a remote client and on the VPN gateway can be found in appendix C. Once running, WireGuard’s status on either machine can be checked with `wg show` to inspect the machine’s WireGuard interface and peer connections.

To demonstrate how WireGuard establishes secure communications between the internal Warwick domain and remote clients, a packet capture on the `internet` machine was performed. This was completed in the same way as it was for the IPsec VPN implementation with `remote1` sending a ping request to `local1`, in order to represent a remote worker communicating with an internal asset. The command to capture packets and invoked on the `internet` machine was:

```
> tcpdump -s0 -v -w /hostlab/assurances/IPsec/wireguard.enabled.pcap
```

And the command to generate traffic (the ping request) that was invoked on `remote1` was:

```
> ping -c2 local1
```

Like in the IPsec implementation the internal asset (`local1`) is now reachable, as shown in figure 3.3, but the packet capture which gathered these ICMP packets is very different. This packet capture is stored in the file `assurances/WireGuard/wireguard.enabled.pcap` and a screenshot of this file is shown in figure 3.4.

```

root@remote1:~# ping -c2 local1
PING local1 (10.11.0.2) 56(84) bytes of data.
64 bytes from local1 (10.11.0.2): icmp_seq=1 ttl=63 time=3.12 ms
64 bytes from local1 (10.11.0.2): icmp_seq=2 ttl=63 time=2.50 ms

--- local1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 2.495/2.807/3.120/0.312 ms
root@remote1:~#

```

Figure 3.3: Once WireGuard is Enabled

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	c6:f2:a1:74:22:74	Broadcast	ARP	42	Who has 213.1.133.99? Tell 213.1.133.97
2	0.000179	ae:56:11:72:96:0f	c6:f2:a1:74:22:74	ARP	42	213.1.133.99 is at ae:56:11:72:96:0f
3	0.000183	198.51.100.2	213.1.133.99	WireGuard	190	Handshake Initiation, sender=0xBA930087
4	0.000787	213.1.133.99	198.51.100.2	WireGuard	134	Handshake Response, sender=0x150A5150, receiver=0xBA930087
5	0.001163	198.51.100.2	213.1.133.99	WireGuard	170	Transport Data, receiver=0x150A5150, counter=0, datalen=96
6	0.001520	213.1.133.99	198.51.100.2	WireGuard	170	Transport Data, receiver=0xBA930087, counter=0, datalen=96
7	1.000852	198.51.100.2	213.1.133.99	WireGuard	170	Transport Data, receiver=0x150A5150, counter=1, datalen=96
8	1.002313	213.1.133.99	198.51.100.2	WireGuard	170	Transport Data, receiver=0xBA930087, counter=1, datalen=96
9	5.068268	ae:56:11:72:96:0f	c6:f2:a1:74:22:74	ARP	42	Who has 213.1.133.97? Tell 213.1.133.99
10	5.068295	c6:f2:a1:74:22:74	ae:56:11:72:96:0f	ARP	42	213.1.133.97 is at c6:f2:a1:74:22:74


```

▶ Frame 5: 170 bytes on wire (1360 bits), 170 bytes captured (1360 bits)
▶ Ethernet II, Src: c6:f2:a1:74:22:74 (c6:f2:a1:74:22:74), Dst: ae:56:11:72:96:0f (ae:56:11:72:96:0f)
▶ Internet Protocol Version 4, Src: 198.51.100.2, Dst: 213.1.133.99
▶ User Datagram Protocol, Src Port: 38243, Dst Port: 51820
▼ WireGuard Protocol
  Type: Transport Data (4)
  Reserved: 000000
  Receiver: 0x150a5150
  Counter: 0
  Encrypted Packet
  [Stream index: 0]

```

Figure 3.4: Packet Capture of WireGuard Communication

This packet capture shows that WireGuard uses its own bespoke network protocol, as represented in the Protocol column, to transfer encrypted traffic over the VPN created. Packet numbers 3 and 4 show the initial fast WireGuard handshake which is invoked when the remote client `remote1` sends data to the WireGuard VPN gateway `vpn2.u2086937.cyber2020.test`. Once the cryptographic data protections have been instantiated, and both gateway and client have been authenticated, packets 5 through 8 show the two ICMP ping packets sent by the client `remote1` and the corresponding two ICMP reply packets sent by the internal Warwick asset `local1`. The first of these ICMP ping packets has been highlighted in this screenshot to show that the data payload of the packet is a “Encrypted Packet” and it uses the bespoke WireGuard Protocol. Also, just like IPsec, the ultimate destination of the packet (the IP address of the internal Warwick asset `local1`) is hidden from anyone capturing packets on the network, thus adding an extra layer of security.

Similar to IPsec again, once the remote client concludes their business with any internal Warwick assets, and no longer wants to access this domain, the encrypted tunnel can be terminated with the command:

```
> wg-quick down wg0
```

This is issued by `remote1` and produces the output shown in figure 3.5 if successful.

```
root@remote1:/etc/wireguard# wg-quick down wg0
[#] ip link delete dev wg0
root@remote1:/etc/wireguard#
```

Figure 3.5: Disabling WireGuard Interface

In this WireGuard VPN model the remote machines `remote1`, `remote2`, `remote3`, `remote4`, and `remote5` have all been configured to establish secure remote connections with the Warwick VPN gateway `vpn2.u2086937.cyber2020.test` and access assets within the `u2086937.cyber2020.test` Warwick domain. These assets are represented by local machines `local1`, `local2`, and `local3`. This setup shows that WireGuard can be used to create a robust VPN configuration which is notably easier to implement than IPsec. The status of the WireGuard gateway VPN `vpn2.u2086937.cyber2020.test` when all three remote clients are connected and can communicate with internal Warwick assets can be found in appendix D.

3.1.3 WireGuard Compared to IPsec

This report demonstrated both the implementation of an IPsec VPN and a WireGuard VPN in the Netkit model. Both of these VPN solutions have their advantages and disadvantages and here this report looks to succinctly evaluate WireGuard against IPsec. WireGuard is described as being a simple, lean, and fast VPN solution that uses the latest cryptographic technologies to be the best cross-platform technology for securing data communications (WireGuard, 2020b). As such, WireGuard defines four main features which it's creators have focused on. The first of these is the technology's ease-of-use. After implementing both an IPsec VPN and a WireGuard VPN it was immediately noticeable that WireGuard took significantly less time to implement both in terms of research and configuration. This is because WireGuard takes inspiration from OpenSSH and it's key management approach as it utilises simple public keys and out-of-band key exchanges (Donenfeld, 2017). This approach leads to the next feature of WireGuard, it's minimal attack surface. By using simple public keys and leaving key distribution to protocols higher up the network stack, WireGuard does not become bloated like IKE or OpenVPN (WireGuard, 2020b) which inevitably leads to more vulnerabilities. Instead, the ease-of-use and simplicity of WireGuard means that it requires minimal code to implement and is easily auditable, especially compared to IPsec and OpenVPN (WireGuard, 2020b). For instance, to configure IPsec in this Netkit model required 24 lines of code across two files, a private RSA key, and several x509 certificates on both the VPN gateway and each remote client. In contrast, WireGuard only required 16 lines of code in one file for the VPN gateway and 7 lines of code in one file on each client. This is not mentioning the very large implementation codebase IPsec uses compared to the substantially smaller implementation codebase WireGuard uses – around 4000 lines of code for WireGuard, compared to around 400,000 for strongSwan IPsec (Salter, 2018). The third distinguishing feature of WireGuard is the “state-of-the-art” cryptography it uses to secure trusted connections (WireGuard, 2020b). WireGuard uses the following primitives – which is a collection of algorithms or protocols in WireGuard literature (WireGuard, 2020a):

- ◊ ChaCha20 for symmetric encryption, authenticated with Poly1305, using RFC7539's AEAD construction
- ◊ Curve25519 for ECDH

-
- ◊ BLAKE2s for hashing and keyed hashing, described in RFC7693
 - ◊ SipHash24 for hashtable keys
 - ◊ HKDF for key derivation, as described in RFC586

In contrast to this, IPsec uses the older AES and SHA algorithms for encryption and hashing which are “industry standard”, but have been known to have vulnerabilities, especially in their implementations (Scripcariu et al., 2018; Seals, 2020). Although, as mentioned in section 1.1.1 the cryptographic algorithms used for IPsec in this Netkit model are secure in 2020. In addition to this, WireGuard strictly limits the cryptographic agility of users to prevent downgrade or man-in-the-middle attack – users cannot choose which cryptography algorithms or protocols to use. This intentional lack of cryptographic agility is to ensure that if a vulnerability in the underlying primitives is discovered, all that needs to be done is an update at each endpoint to a newer, more secure primitive and that vulnerability is completely alleviated – thus, there is no opportunity for downgrade attacks (Donenfeld, 2017). The final main feature which WireGuard boasts is it’s outstanding performance. WireGuard’s use of efficient, high-speed cryptographic primitives and it’s direct integration into the Linux kernel means that it can perform secure networking tasks faster than both IPsec and OpenVPN (Donenfeld, 2017). This means that, theoretically, WireGuard offers improved performance over IPsec in the following areas (Taylor, 2020):

- ◊ Faster speeds
- ◊ Better battery life with phones/tablets
- ◊ Better roaming support (mobile devices)
- ◊ More reliability
- ◊ Faster at establishing connections/re-connections (faster handshake)

Strong performance in these areas makes WireGuard ideal for mobile users who need a VPN solution on-the-go or for embedded devices which have limited computing power. This demographic is comparable to the remote clients who need to access internal Warwick assets as they are likely be using mobile devices (i.e. laptops, smartphones, tables) which do not have the computing power of a desktop PC or server.

With these four main features of WireGuard in mind – easy-of-use, minimal attack surface, “state-of-the-art” cryptography, and performance – the only advantage which IPsec might have over WireGuard is that it is more configurable by the end user. For example, when configuring the IPsec VPN there was a plethora of options that could be defined in the `config setup` and `conn %default` sections within the `/etc/ipsec/ipsec.conf` file. These included; authorisation mechanism (public key, PSK, etc.), ESP encryption/authentication algorithms to use, IKE/ISAKMP SA encryption/authentication algorithms to use, lifetimes of packets, key retries, key exchange protocol, IPsec mode (tunnel, transport, etc.), PFS, AH algorithms to use, DPD settings, and so on (strongSwan, 2020). This provides a lot of flexibility for the end user when implementing an IPsec VPN, which is why it was important to specify the security configurations this report recommends in section 1.1.1, especially in terms of cipher selection. However, this can be seen as an advantage and disadvantage as it allows for more control, yet makes the implementation process considerably more complicated. This added complexity increases the likelihood of a misconfiguration or use of a weak cryptography algorithm/protocol

and, as a result, the VPN could be vulnerable. On the other hand, WireGuard gives the end user fewer options to configure and defines the cryptography primitives that will be used to make the implementation process simple and prevent potential downgrade or man-in-the-middle attacks. If stronger cryptography primitives then emerge, these automatically replace the current primitives being used when WireGuard is updated at each endpoint. This approach means the user does not have to worry about keeping track of new cryptography findings and simply needs to keep WireGuard up-to-date.

That said, having the VPN software automatically implement the best cryptography primitives and configurations is only useful if those protocols and configurations are secure. In the Netkit model the cryptography configurations for the IPsec VPN were chosen based on NSA recommendations, and x509 certificates were used for authentication with big 4096-bit RSA keys. This was done to ensure that the cryptography would remain secure while Warwick needed this VPN solution. Whereas, WireGuard uses cryptography primitives that it chooses and utilises 256-bit public keys for its authentication. Thus, on the surface it may appear that the IPsec implementation is more secure, or at least can be configured to be more secure. However, there may not necessarily be such a stark difference in the security of the two implementations. This is because the IPsec implementation relies upon old, “industry standard” cryptography protocols which require such a large key as they are much more susceptible to dictionary attacks. These attacks immediately resolve large chunks of the problem space (what a key could potentially be) and make brute forcing the key considerably easier (Salter, 2018). Due to this, IPsec (and OpenVPN) need to use tremendously long keys to expand the problem set so that it is so large that dictionary attacks are mitigated. On the other hand, WireGuard uses newer, modern cryptography primitives which do not require such drastic key sizes. Thus, so long as the algorithms that WireGuard is built on remain unbreached, a 256-bit key is strong enough to not be brute forced until quantum computing advances are able to break the elliptical curve (Curve25519) algorithm (Salter, 2018). At this point, WireGuard has some mitigation techniques to increase the strength of its public keys, like an optional pre-shared symmetric key mode, but these are in lieu of using a completely post-quantum crypto system (Donenfeld, 2017). Henceforth, the cryptography primitives implemented by WireGuard are, as of 2020, strong enough to ensure secure communications.

Finally, before implementing WireGuard at the enterprise level, it is important to consider the completeness of WireGuard as an enterprise solution. Despite WireGuard having many advantages over IPsec, it is still “currently under heavy development” (WireGuard, 2020b). This was highlighted by WireGuard’s delay in being supported cross-platform (Salter, 2018), however as of 2020 WireGuard is supported across all major platforms (Windows, Mac OS, Android, iOS, and Linux), as well as being fully integrated into several commercial VPN services (NordVPN, Surfshark, Mullvad, OVPN, and AzireVPN among others) (Taylor, 2020). Hence, with such widespread cross-platform commercial deployment, along with all the previous advantages mentioned, this report concludes that implementing WireGuard will be more efficient and simpler than implementing IPsec, with a couple of caveats. Firstly, it is important to note that an adequate key distribution solution be implemented alongside a WireGuard VPN implementation as this is purposely out of scope of WireGuard. Secondly, by default WireGuard was not built for anonymity or privacy, but instead for security and speed (Taylor, 2020). This means that there are a few privacy problems with WireGuard – such as saving connected IP

addresses on the server and not assigning dynamic IP addresses – but there are solutions to these problems (Taylor, 2020), and using WireGuard to secure the communications of remote workers is the objective of Warwick – not the anonymity of remote workers. For a summary of the advantages and disadvantages of WireGuard compared to IPsec see table 3.1 below.

Advantages	Disadvantages
easy-to-use	not configurable
minimal attack surface and easier to audit	lacks default anonymity or privacy features
“state-of-the-art” cryptography	some features are still in development
better performance	smaller key size
cross-platform and better on mobile/embedded devices	
limits cryptographic agility	
automatic updates to most secure primitives	
built into the Linux kernel	

Table 3.1: Advantages and Disadvantages of WireGuard Compared to IPsec

References

- Barker, E. et al. (2020). *Guide to IPsec VPNs*. Revision 1. [online] Gaithersburg, MD: National Standard of Standards and Technology. Available from: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-77r1.pdf> (Accessed 25 November 2020).
- Boucheron, B. (2018). *How To Create a Self-Signed SSL Certificate for Apache in Ubuntu 18.04*. [online] Available from: <https://www.digitalocean.com/community/tutorials/how-to-create-a-self-signed-ssl-certificate-for-apache-in-ubuntu-18-04/> (Accessed 26 November 2020).
- Donenfeld, J. A. (2017). WireGuard: Next Generation Kernel Network Tunnel. *Proceedings of the Network and Distributed System Security Symposium, NDSS* [online]. Available from: <https://pdfs.semanticscholar.org/de53/d55470addc0e0d52fd82c2f4bd8e4ca879c2.pdf> (Accessed 20 October 2020).
- DreamHost (2020). *The most important steps to take to make an Apache server more secure*. [online] Available from: <https://help.dreamhost.com/hc/en-us/articles/226327268-The-most-important-steps-to-take-to-make-an-Apache-server-more-secure/> (Accessed 26 November 2020).
- Frankel, S. et al. (2005). *Guide to IPsec VPNs*. [online] Gaithersburg, MD: National Standard of Standards and Technology. Available from: <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-77.pdf> (Accessed 25 November 2020).
- Kumar, C. (2019). *Apache Web Server Hardening and Security Guide*. [online] Available from: <https://geekflare.com/apache-web-server-hardening-security/> (Accessed 27 November 2020).
- Linode Community (2020). *Set Up WireGuard VPN on Ubuntu*. [online] Available from: <https://www.linode.com/docs/guides/set-up-wireguard-vpn-on-ubuntu/> (Accessed 7 December 2020).
- National Cyber Security Centre (2016a). *Provisioning and securing security certificates*. [online] Available from: <https://www.ncsc.gov.uk/guidance/provisioning-and-securing-security-certificates/> (Accessed 28 November 2020).
- (2016b). *Using IPsec to protect data*. [online] Available from: <https://www.ncsc.gov.uk/guidance/using-ipsec-protect-data#crypt/> (Accessed 25 November 2020).
- Nguyen, J. (2015). *OpenSSL Certificate Authority*. [online] Available from: <https://jamielinux.com/docs/openssl-certificate-authority/create-the-intermediate-pair.html/> (Accessed 25 November 2020).
- Ostryzhko, M. (2018). *How to Harden Your Apache Web Server on an Ubuntu 18.04 Dedicated Server or VPS*. [online] Available from: <https://hostadvice.com/how-to/how-to-harden-your-apache-web-server-on-ubuntu-18-04/> (Accessed 26 November 2020).
- OWASP (2017). *Top 10 Web Application Security Risks*. [online] Available from: <https://owasp.org/www-project-top-ten/> (Accessed 27 November 2020).
- (2020). *Clickjacking*. [online] Available from: <https://owasp.org/www-community/attacks/Clickjacking/> (Accessed 27 November 2020).
- Russell, A. (2020). *SSL/TLS Best Practices for 2020*. [online] Available from: <https://www.ssl.com/guide/ssl-best-practices/> (Accessed 27 November 2020).
- Salter, J. (2018). *WireGuard VPN review: A new type of VPN offers serious advantages*. [online] Available from: <https://arstechnica.com/gadgets/2018/08/wireguard-vpn-review-fast-connections-amaze-but-windows-support-needs-to-happen/> (Accessed 7 December 2020).
- Scripcariu, L. et al. (2018). AES Vulnerabilities Study. *10th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)* [online] Iasi, Romania, 28-30 June 2018. Available from: <https://ieeexplore.ieee.org/document/8678930> (Accessed 10 December 2020).

-
- Seals, T. (2020). *Exploit Fully Breaks SHA-1, Lowers the Attack Bar*. [online] Available from: <https://threatpost.com/exploit-fully-breaks-sha-1/151697/> (Accessed 10 December 2020).
- Steffen, A. et al. (2018). *Security Recommendations*. [online] Available from: <https://wiki.strongswan.org/projects/strongswan/wiki/SecurityRecommendations> (Accessed 25 November 2020).
- strongSwan (2020). *General Connection Parameters*. [online] Available from: <https://wiki.strongswan.org/projects/strongswan/wiki/ConnSection> (Accessed 7 December 2020).
- Taylor, S. (2020). *WireGuard VPN: What You Need to Know*. [online] Available from: <https://restoreprivacy.com/vpn/wireguard/> (Accessed 7 December 2020).
- WireGuard (2020a). *Protocol & Cryptography - WireGuard*. [online] Available from: <https://www.wireguard.com/protocol/> (Accessed 7 December 2020).
- (2020b). *WireGuard: fast, modern, secure VPN tunnel*. [online] Available from: <https://www.wireguard.com/> (Accessed 7 December 2020).

A IPsec Tunnel Established

```
root@remote1:~# ipsec up warwick
initiating IKE_SA warwick[3] to 213.1.133.98
generating IKE_SA_INIT request 0 [ SA KE No N(NATD_S_IP) N(NATD_D_IP) N(FRAG_SUP) N(HASH_ALG) N(REDIR_SUP) ]
sending packet: from 198.51.100.2[500] to 213.1.133.98[500] (720 bytes)
received packet: from 213.1.133.98[500] to 198.51.100.2[500] (773 bytes)
parsed IKE_SA_INIT response 0 [ SA KE No N(NATD_S_IP) N(NATD_D_IP) CERTREQ N(FRAG_SUP) N(HASH_ALG) N(CHDLESS_SUP) N(MULT_AUTH) ]
selected proposal: IKE:AES_CBC_256/HMAC_SHA2_384_192/PRF_HMAC_SHA2_384/MODP_4096
received cert request for "C=GB, ST=England, L=Markwickshire, O=UoW, OU=WMG_CSC_Teaching, CN=ca_root.u2086937.cyber2020.test, E=vpn1.u2086937.cyber2020.test"
received cert request for "C=GB, ST=England, L=Markwickshire, O=UoW, OU=WMG_CSC_Teaching, CN=vpn_ca.u2086937.cyber2020.test, E=vpn1.u2086937.cyber2020.test"
sending cert request for "C=GB, ST=England, L=Markwickshire, O=UoW, OU=WMG_CSC_Teaching, CN=ca_root.u2086937.cyber2020.test, E=vpn1.u2086937.cyber2020.test"
sending cert request for "C=GB, ST=England, L=Markwickshire, O=UoW, OU=WMG_CSC_Teaching, CN=vpn_ca.u2086937.cyber2020.test, E=vpn1.u2086937.cyber2020.test"
authentication of 'remote1' (myself) with RSA_EMSA_PKCS1_SHA2_384 successful
sending end entity cert "C=GB, ST=England, L=Markwickshire, O=UoW, OU=WMG_CSC_Teaching, CN=remote1, E=remote1@email.com"
establishing CHILD_SA warwick{4}
generating IKE_AUTH request 1 [ IDi CERT N(INIT_CONTACT) CERTREQ IDr AUTH SA TSi TSr N(MOBIKE_SUP) N(NO_ADD_ADDR) N(MULT_AUTH) N(ADD_4_ADDR) ]
splitting IKE message (2776 bytes) into 3 fragments
generating IKE_AUTH request 1 [ EF(1/3) ]
generating IKE_AUTH request 1 [ EF(2/3) ]
generating IKE_AUTH request 1 [ EF(3/3) ]
sending packet: from 198.51.100.2[4500] to 213.1.133.98[4500] (1244 bytes)
sending packet: from 198.51.100.2[4500] to 213.1.133.98[4500] (1244 bytes)
sending packet: from 198.51.100.2[4500] to 213.1.133.98[4500] (444 bytes)
received packet: from 213.1.133.98[4500] to 198.51.100.2[4500] (1244 bytes)
parsed IKE_AUTH response 1 [ EF(1/3) ]
received fragment #1 of 3, waiting for complete IKE message
received packet: from 213.1.133.98[4500] to 198.51.100.2[4500] (1244 bytes)
parsed IKE_AUTH response 1 [ EF(2/3) ]
received fragment #2 of 3, waiting for complete IKE message
received packet: from 213.1.133.98[4500] to 198.51.100.2[4500] (428 bytes)
parsed IKE_AUTH response 1 [ EF(3/3) ]
received fragment #3 of 3, reassembled fragmented IKE message (2760 bytes)
parsed IKE_AUTH response 1 [ IDr CERT AUTH SA TSi TSr N(AUTH_LFT) N(MOBIKE_SUP) N(ADD_4_ADDR) ]
received end entity cert "C=GB, ST=England, L=Markwickshire, O=UoW, OU=WMG_CSC_Teaching, CN=vpn1.u2086937.cyber2020.test, E=vpn1.u2086937.cyber2020.test"
using certificate "C=GB, ST=England, L=Markwickshire, O=UoW, OU=WMG_CSC_Teaching, CN=vpn1.u2086937.cyber2020.test, E=vpn1.u2086937.cyber2020.test"
using trusted intermediate ca certificate "C=GB, ST=England, L=Markwickshire, O=UoW, OU=WMG_CSC_Teaching, CN=vpn_ca.u2086937.cyber2020.test, E=vpn1.u2086937.cyber2020.test"
checking certificate status of "C=GB, ST=England, L=Markwickshire, O=UoW, OU=WMG_CSC_Teaching, CN=vpn1.u2086937.cyber2020.test, E=vpn1.u2086937.cyber2020.test"
certificate status is not available
using trusted ca certificate "C=GB, ST=England, L=Markwickshire, O=UoW, OU=WMG_CSC_Teaching, CN=ca_root.u2086937.cyber2020.test, E=vpn1.u2086937.cyber2020.test"
checking certificate status of "C=GB, ST=England, L=Markwickshire, O=UoW, OU=WMG_CSC_Teaching, CN=vpn_ca.u2086937.cyber2020.test, E=vpn1.u2086937.cyber2020.test"
using trusted certificate "C=GB, ST=England, L=Markwickshire, O=UoW, OU=WMG_CSC_Teaching, CN=ca_root.u2086937.cyber2020.test, E=vpn1.u2086937.cyber2020.test"
crl correctly signed by "C=GB, ST=England, L=Markwickshire, O=UoW, OU=WMG_CSC_Teaching, CN=ca_root.u2086937.cyber2020.test, E=vpn1.u2086937.cyber2020.test"
crl is valid: until Apr 04 11:40:17 2021
using cached crl
certificate status is good
reached self-signed root ca with a path length of 1
authentication of 'vpn1.u2086937.cyber2020.test' with RSA_EMSA_PKCS1_SHA2_384 successful
IKE_SA warwick[3] established between 198.51.100.2[remote1]...213.1.133.98[vpn1.u2086937.cyber2020.test]
scheduling reauthentication in 2924s
maximum IKE_SA lifetime 3524s
selected proposal: ESP:AES_CBC_256/HMAC_SHA2_384_192/NO_EXT_SEQ
CHILD_SA warwick{4} established with SPIs c85d458e_i c094b593_o and TS 198.51.100.2/32 === 10.11.0.0/16
received AUTH_LIFETIME of 2913s, scheduling reauthentication in 2313s
peer supports MOBIKE
connection 'warwick' established successfully
root@remote1:~#
```

Establishing an encrypted tunnel on machine `remote1` with `ipsec up warwick` - note, some of the x509 certificate names have been cropped for brevity.

B IPsec Status with all Remote Clients

```
root@vpn1:~# ipsec status
Security Associations (5 up, 0 connecting):
  rw{5}: ESTABLISHED 33 seconds ago, 213.1.133.98[vpn1.u2086937.cyber2020.test]...198.51.100.4[remote3]
  rw{5}:  INSTALLED, TUNNEL, reqid 5, ESP SPIs: c020e061_i c404fb73_o
  rw{5}:  10.11.0.0/16 === 198.51.100.4/32
  rw{4}: ESTABLISHED 42 seconds ago, 213.1.133.98[vpn1.u2086937.cyber2020.test]...198.51.100.3[remote2]
  rw{4}:  INSTALLED, TUNNEL, reqid 4, ESP SPIs: c0883140_i ce86b6d4_o
  rw{4}:  10.11.0.0/16 === 198.51.100.3/32
  rw{3}: ESTABLISHED 50 seconds ago, 213.1.133.98[vpn1.u2086937.cyber2020.test]...198.51.100.2[remote1]
  rw{3}:  INSTALLED, TUNNEL, reqid 3, ESP SPIs: c9c35b64_i cac8425d_o
  rw{3}:  10.11.0.0/16 === 198.51.100.2/32
  rw{2}: ESTABLISHED 63 seconds ago, 213.1.133.98[vpn1.u2086937.cyber2020.test]...198.51.100.6[remote5]
  rw{2}:  INSTALLED, TUNNEL, reqid 2, ESP SPIs: caac0b72_i cc187244_o
  rw{2}:  10.11.0.0/16 === 198.51.100.6/32
  rw{1}: ESTABLISHED 77 seconds ago, 213.1.133.98[vpn1.u2086937.cyber2020.test]...198.51.100.5[remote4]
  rw{1}:  INSTALLED, TUNNEL, reqid 1, ESP SPIs: c61d0b79_i c9849bde_o
  rw{1}:  10.11.0.0/16 === 198.51.100.5/32
root@vpn1:~#
```

The output of the `ipsec status` command when all remote clients (`remote1`, `remote2`, `remote3`, `remote4`, and `remote5`) are connected to Warwick's VPN gateway `vpn1.u2086937.cyber2020.test`.

C WireGuard Enabling

```
root@vpn2:/etc/wireguard# wg-quick up wg0
[#] ip link add wg0 type wireguard
[#] wg setconf wg0 /dev/fd/63
[#] ip -4 address add 10.10.10.1/24 dev wg0
[#] ip link set mtu 1420 up dev wg0
[#] iptables -A FORWARD -i wg0 -j ACCEPT; iptables -t nat -A POSTROUTING -o eth1
-j MASQUERADE
root@vpn2:/etc/wireguard# systemctl enable wg-quick@wg0
Created symlink /etc/systemd/system/multi-user.target.wants/wg-quick@wg0.service
â /lib/systemd/system/wg-quick@.service.
root@vpn2:/etc/wireguard#
```

Output from enabling WireGuard on VPN gateway `vpn2.u2086937.cyber2020.test` - note, this is done in process `vpn2.u2086937.cyber2020.test.startup` to automatically start WireGuard on boot.

```
root@remote1:/etc/wireguard# wg-quick up wg0
[#] ip link add wg0 type wireguard
[#] wg setconf wg0 /dev/fd/63
[#] ip -4 address add 10.10.10.2/24 dev wg0
[#] ip link set mtu 1420 up dev wg0
[#] ip -4 route add 10.11.0.0/16 dev wg0
root@remote1:/etc/wireguard#
```

Output from enabling WireGuard on remote client `remote1`.

D WireGuard Status with all Remote Clients

```
root@vpn2:~# wg show
interface: wg0
  public key: hCixDlUgsN3MhKEmP0dnz0zVMtRkJjwUae3FrMOSDI=
  private key: (hidden)
  listening port: 51820

peer: rYZLwc7/inT6zrNz+xFAJamM+y2my//SSFsI1k3IUmw=
  endpoint: 198.51.100.2:47468
  allowed ips: 10.10.10.2/32
  latest handshake: 7 seconds ago
  transfer: 276 B received, 220 B sent

peer: 2r9l3sMkG001K5emkbLvEgwc0yqNcGTj8zKJoBChPME=
  endpoint: 198.51.100.5:59962
  allowed ips: 10.10.10.5/32
  latest handshake: 15 seconds ago
  transfer: 308 B received, 220 B sent

peer: AKbz8+Hbmopcs/+10S6jcVFQWwBAJX42XfjQmyFStGw=
  endpoint: 198.51.100.4:47702
  allowed ips: 10.10.10.4/32
  latest handshake: 23 seconds ago
  transfer: 308 B received, 220 B sent

peer: ngbhZfDo835x22LH+WW/tKngH7MK5R/xrBmG5FQPgB8=
  endpoint: 198.51.100.3:55936
  allowed ips: 10.10.10.3/32
  latest handshake: 28 seconds ago
  transfer: 308 B received, 220 B sent

peer: 1DwktFysotQo6pLhYCavGumPFaNC/YGw1h1MxooXUzU=
  endpoint: 198.51.100.6:55802
  allowed ips: 10.10.10.6/32
  latest handshake: 1 minute, 11 seconds ago
  transfer: 436 B received, 348 B sent
root@vpn2:~#
```

The output of the `wg show` command when all `remote1`, `remote2`, `remote3`, `remote4`, and `remote5` machines are connected to Warwick's VPN gateway `vpn2.u2086937.cyber2020.test`.