CS-E4300 Network Security, 2021-2022
Project 3
Teachers responsible for the project: Mohit Sethi, Aleksi Peltonen, Jacopo Bufalino

WireGuard mesh for IoT devices and cloud servers

In this exercise, you will act as a security and networking consultant and advice Acme Inc. on how to upgrade their legacy IPsec VPN tunnels to modern and lightweight WireGuard tunnels.

Current setup: You have already helped Acme to migrate the servers from the customer sites to the cloud. This has allowed them to reduce the server instances, and now they only have two servers for IoT devices, one for each of two device vendors.

Acme now wants your help to move to modern lightweight WireGuard tunnels between IoT devices and the cloud servers. WireGuard is used for establishing direct tunnels between end hosts, which are called *peers* in WireGuard terminology. In comparison, IPsec is typically used for establishing tunnels between gateways or between a device and gateway.

ACME manages IoT devices for two different customers. Hence, ACME has two servers in the cloud, one for each customer. Based on prudent security practice, ACME wants to isolate each customer's devices and the corresponding server into a separate WireGuard subnet. While currently only a single server for each customer's devices is needed, ACME expects to scale up the number of devices, servers, and customers.

Your goal is to migrate from IPsec-based VPN setup to direct WireGuard tunnels between devices and servers.

WireGuard identifies end hosts with public keys. Asymmetric public-private key pairs can be generated with commands such as *wg genkey*. For establishing a tunnel between two hosts, you will need to configure the public key of the other end host. A sample WireGuard configuration file is shown here:

```
[Interface]
PrivateKey = <privatekey>
ListenPort = 21841

[Peer]
PublicKey = <peer publickey>
Endpoint = <peer public ip>:<peer listen port>
AllowedIPs = 192.168.2.0/24

PersistentKeepalive = 25
```

For a minimum acceptable solution, configure the WireGuard clients and servers manually and **submit the configuration files**.

Use the Vagrant testbed from exercise 2. We recommend creating a separate clone of the testbed repository for exercise 3. ACME expects functional WireGuard tunnels between customer 1 devices (client-a1, client-a2) and server (server-s1) as well as between customer 2 devices (client-b1, client-b2) and server (server-s2).

While working on this assignment, you will quickly realize that manually copying public keys between end hosts does not scale as the number of peers, i.e., IoT devices and servers increases. There are various tools that try to automate the process by collecting the public keys at a central management service:

- https://golangrepo.com/repo/seashell-drago
- https://tailscale.com/kb/1086/tailscale-vs-wireguard/

For this assignment, we have developed a simple management API server for tracking WireGuard peers and public keys. It is based on an open-source project called *Meshmash*. You can launch the containerized management API server here: https://netsec.vikaa.fi/netsec3. Create either one server per group or one per student, as needed. For full points, you should use the management server and **submit both the client software that accesses the API and the produced**

configuration for the clients and servers. The *Management API* on the management servers is used for setting up the WireGuard mesh:

Request type	URL	Request Body	Response Body
POST	/overlays	{"overlay_name":"site1"}	{"overlay_id": "a9af9b902c8b4007924c4928923111933", "subnet": "10.254.2.0/24", "overlay_name": "site1", "devices": {}}
GET	/overlays		{"overlays": ["939b14be0a4146e1bd80619b928fbc76", "a9f07e9cdd0d42199b939f120317373a", "a9af9b902c8b4007924c4928923111933"]}
POST	/devices	{"hostname": "dev2.example.com", "public_ip": "172.18.18.34", "public_key": "tHsl1dwL6rs5DbgxRftJQ/z9ZpgZznQ6QxIQCX6iODQ=", "listen_port": 51820, "device_name": "my_dev2"}	{"device_id": "ebdce3d2d21d44e4aa05279f13c7ce2a", "device_name": "my_dev2", "hostname": "dev2.example.com", "public_ip": "172.18.18.34", "public_key": "tHsl1dwL6rs5DbgxRftJQ/z9ZpgZznQ6QxIQCX6iODQ=", "listen_port": 51820, "token": "f81NOAzhJ1z9UPukhbPamQ", "expiry_ts": 1637494732.068814}
GET	/devices		{"devices": ["de1f28d603ae4beabfbf95b136fec71b", "f4aca0cf59f14e90a32cb3598d687c2d", "7d82c9a7b40941239298dde6bbfebe23"]}
GET	/devices/<device_id>		{"device_id": "ebdce3d2d21d44e4aa05279f13c7ce2a", "device_name": "my_dev2", "hostname": "dev2.example.com", "public_ip": "172.18.18.34", "public_key": "tHsl1dwL6rs5DbgxRftJQ/z9ZpgZznQ6QxIQCX6iODQ=", "listen_port": 51820, "token": "f81NOAzhJ1z9UPukhbPamQ", "expiry_ts": 1637494732.068814}
PUT	/devices/<device_id>	{"public_ip":"172.18.18.36","listen_port":51830}	{"device_id": "ebdce3d2d21d44e4aa05279f13c7ce2a", "device_name": "my_dev2", "hostname": "dev2.example.com", "public_ip": "172.18.18.36", "public_key": "tHsl1dwL6rs5DbgxRftJQ/z9ZpgZznQ6QxIQCX6iODQ=", "listen_port": 51830, "token": "f81NOAzhJ1z9UPukhbPamQ", "expiry_ts": 1637494732.068814}
GET	/overlays/<overlay_id>		{"overlay_id": "a9af9b902c8b4007924c4928923111933", "subnet": "10.254.2.0/24", "overlay_name": "site1", "devices": {}}
DELETE	/overlays/<overlay_id>		{"status": "Deleted"}

DELETE	/devices/<device_id>		{"status": "Deleted"}
POST	/overlays/<overlay_id>/devices	{"device_id":"de1f28d603ae4beabfbf95b136fec71b"}	{"status": "Device added", "tunnel_ip": "10.0.5.1"}
DELETE	/overlays/<overlay_id>/devices/<device_id>		{"status": "Device removed"}
GET	/devices/<device_id>/token		{"device_id": "de1f28d603ae4beabfbf95b136fec71b", "device_name": "my_dev2", "hostname": "dev2.example.com", "public_ip": "172.18.18.38", "public_key": "tHsl1dwL6rs5DbgxRftJQ/z9ZpgZznQ6QxIQCX6iODQ=", "listen_port": 51820, "token": "pKJW6x6GLdjZ0xAh8e_FQg", "expiry_ts": 1637501942.5853105}

Each device is identified by a server-assigned identifier. The overlays are isolated subnets, so that peers in each subnets can only connect to each other with WireGuard. The subnets of the different overlays on the same management server cannot overlap. (This is because it is possible to assign the same device to multiple subnets; however, that feature is not needed in the project.) Requests to the API are authenticated by sending an API key in the `X-Api-Key` request header. You can get the API key for your server from the exercise launcher. Remember to specify `Content-Type` when sending requests that contain a body. Example:

```
x-api-key: fd9r98roieurifddere8re980r9e
Content-Type: application/json
```

The *Device API* is accessed by the IoT devices:

GET	/overlays/<overlay_id>/devices/<device_id>/wgconfig		[Peer 1] PublicKey = tHsl1dwL6rs5DbgxRftJQ/z9ZpgZznQ6QxIQCX6iODQ= AllowedIPs = 10.254.2.2/32 Endpoint = 172.18.18.34:51820 [Peer 2] PublicKey = SIKSx7FpypvmOHCwsOgSK6DtDmCESf2jcHEBrWsON0c= AllowedIPs = 10.254.2.3/32 Endpoint = 172.18.18.36:51820
GET	/devices/<device_id>/token		{"device_id": "de1f28d603ae4beabfbf95b136fec71b", "device_name": "my_dev2", "hostname": "dev2.example.com", "public_ip": "172.18.18.34", "public_key": "tHsl1dwL6rs5DbgxRftJQ/z9ZpgZznQ6QxIQCX6iODQ=", "listen_port": 51820, "token": "X1jrdiYFW_Tc5kzGPZIv6w", "expiry_ts": 1637501942.5853105}
PUT	/devices/<device_id>	{"public_ip":"172.18.18.38"}	{"device_id": "ebdce3d2d21d44e4aa05279f13c7ce2a", "device_name": "my_dev2", "hostname": "dev2.example.com", "public_ip": "172.18.18.38", "public_key": "tHsl1dwL6rs5DbgxRftJQ/z9ZpgZznQ6QxIQCX6iODQ=", "listen_port": 51830, "token": "f81NOAzhJ1z9UPukhbPamQ", "expiry_ts": 1637494732.068814}

A device token is created at the time of device creation. Device API requests from the device are authenticated with the Bearer Authorization header that sets the token. The token expires after 30 minutes, and the device should renew

it before expiry; in a real deployment, the expiry time could be 30 days. Remember to specify `Content-Type` when sending requests that contain a body. Example:

```
Authorization: Bearer X1jrdiYFW_Tc5kzGPZIv6w
Content-Type: application/json
```

WireGuard is already part of the Linux kernel. If you are using the kernel implementation of WireGuard, it might be useful to enable *dyndebug* logging: https://www.procustodibus.com/blog/2021/03/wireguard-logs/#dyndbg to access the logs.