Wireguard: A Modern VPN Protocol

Jinank Jain, Rayhaan Jaufeerally

July 17, 2018

ETH Zürich

Introduction

Why VPN?

- Necessary for point to point security between campuses (e.g. DC's, corporate offices, . . .)
  - As early as the 1970's governments have tapped undersea cables for intelligence,
  - Operation Ivy Bells in 1971 tapped Russian communicatioons to millitary bases[1],
- Necessary for end users to get a clean connection:
  - ISP's doing DNS hijacking to serve inappropriate content,
  - Open WiFi networks when travelling,
  - Geoblocking,

[1] https://en.wikipedia.org/wiki/Operation_Ivy_Bells

Necessity in real life

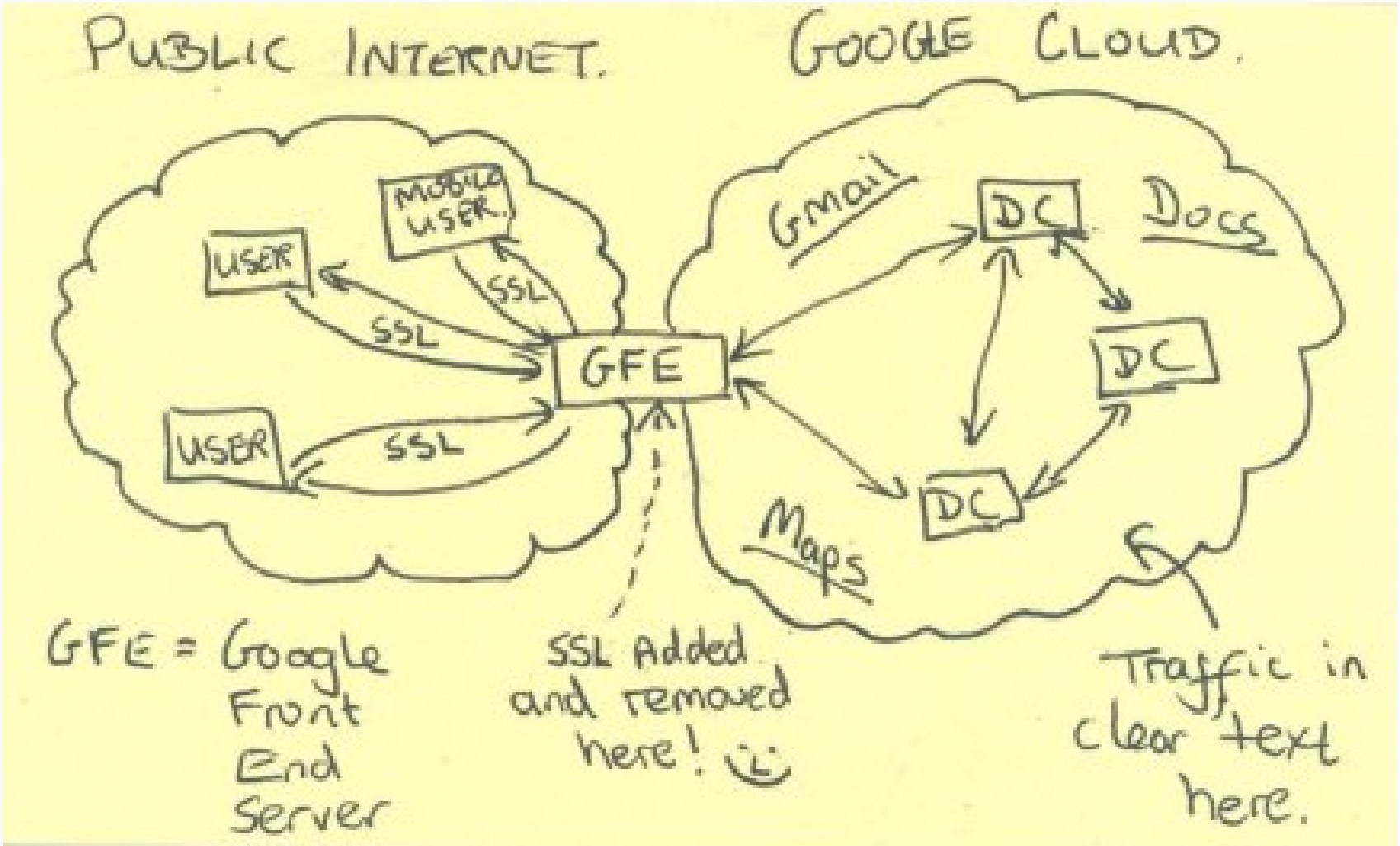


Figure 1: Nation state surveillance of user data including SPII.

History of VPN protocols

- IPSEC
  - Popular for site to site connections with dedicated router hardware,
  - Tedious to set up and high degree of complexity,
  - Large attack surface between IKE (v2), SA mechanisms, XFRM in Linux,
  - Legacy protocol support,
  - IP in IP,
- OpenVPN
  - Implemented in userspace with TUN/TAP (slow),
  - Complex confoguration vulnerable to leaks,
  - Stateful protocol which is brittle in real networks,
  - Large codebase / attack surface,

What is Wireguard?

- Opinionated Layer 3 secure network tunnel for IPv4 and IPv6.
- Lives in the Linux kernel, but cross platform userspace implementations are available.
- UDP based. Punches through firewall.
- Conservative and modern cryptographic principles.
- Emphasis on simplicity and single user auditability.
- Authentication model similar to SSH's `authorized_keys`.

Easily Auditable

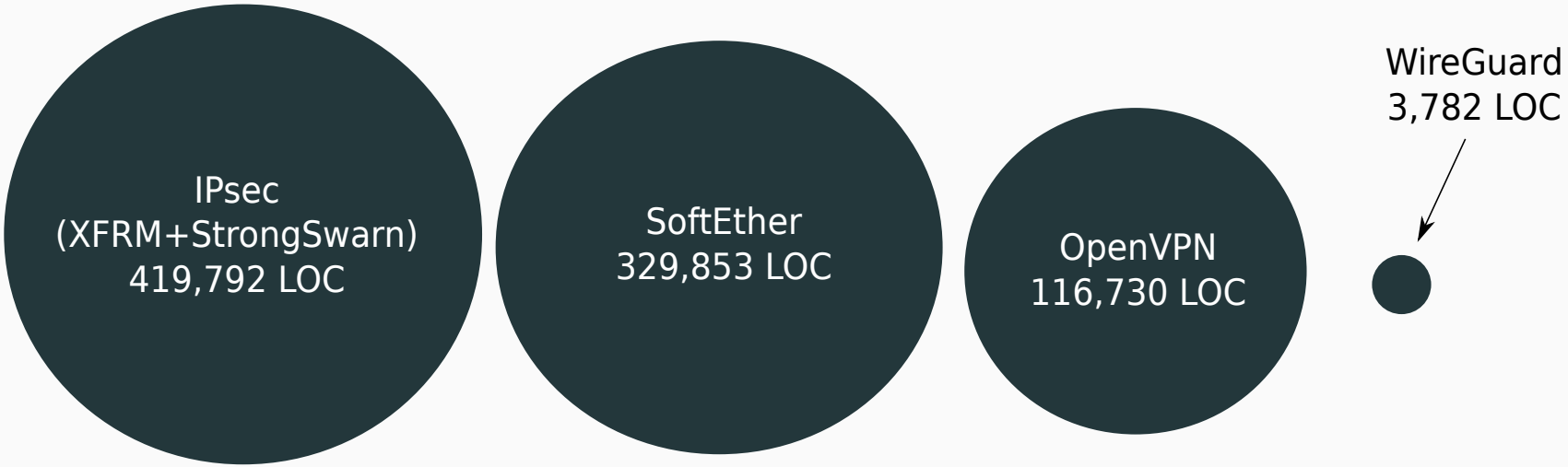


Figure 2: Comparing different VPN protocols in terms of LOC

Minimialistic Interface

"Developers should write programs that can communicate easily with other programs"

— Unix Philosophy

- Wireguard presents a normal network interface

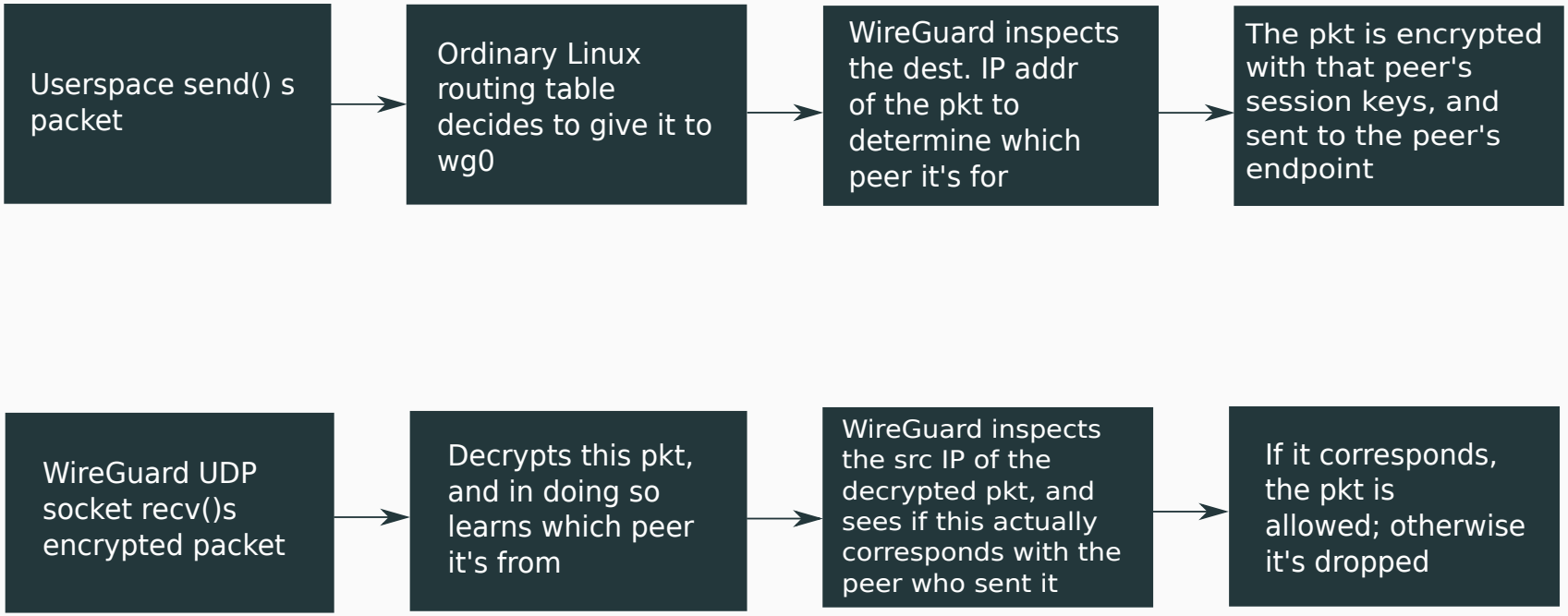
```
# ip link add wg0 type wireguard
# ip address add 10.0.32.1/24 dev wg0
# ip route add default via wg0
```

- By using a standard interface it becomes easier to administer using the existing iproute2 utilities for example

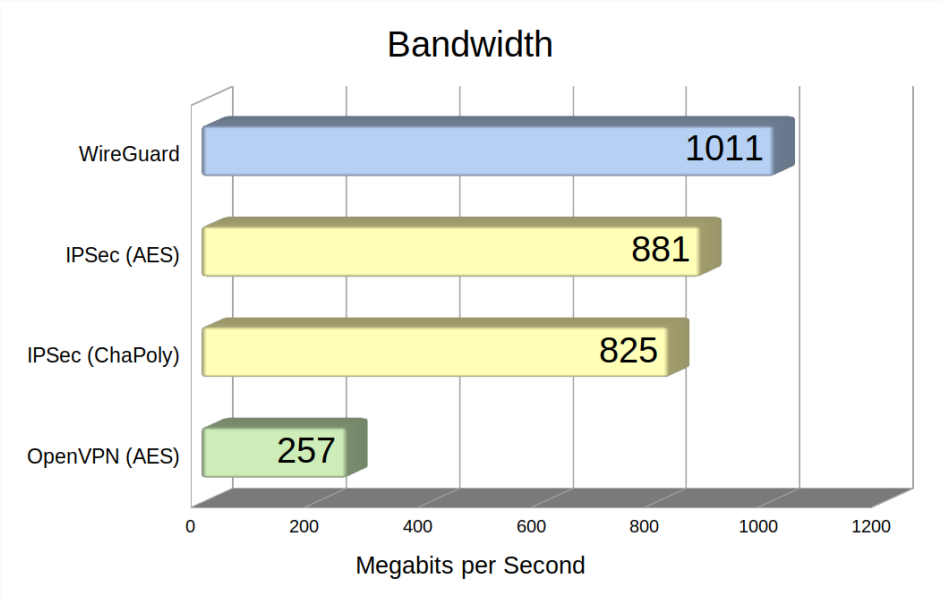
Cryptokey Routing

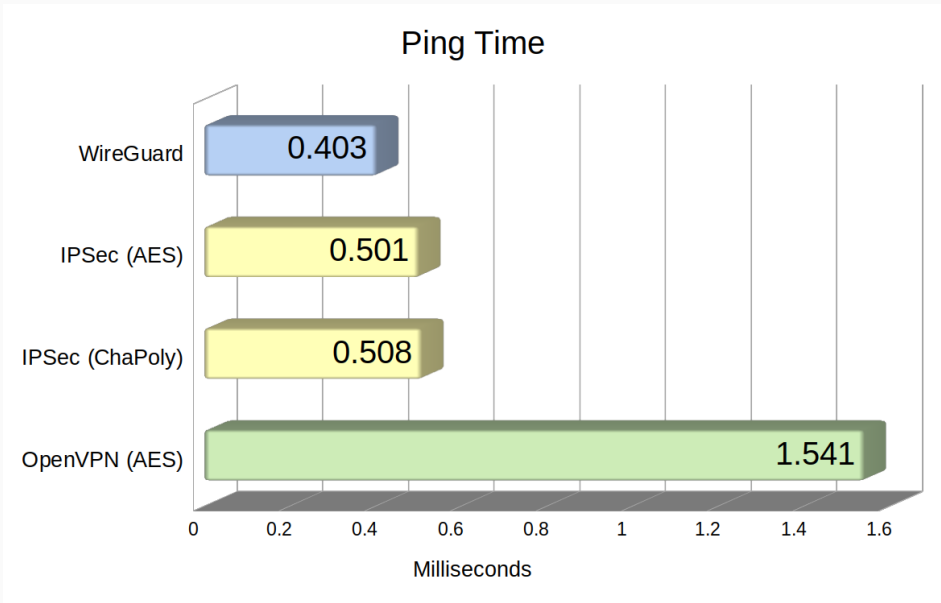
- Fundamental concept of any VPN service
  - Create **mapping** between **public keys of peers** and their **IPs**.
- WireGuard interface has:
  - A private key
  - A listening UDP port
  - A list of peers
- Peer has
  - A public key
  - A list of associated tunnel IPs
  - Optionally has an endpoint IP and port

Cryptokey Routing



Performance





Stealth

- Behaves like a rootkit in some sense.
- Should not respond to any unauthenticated packets.
- Hinder scanners and service discovery.
- Service only responds to packets with correct crypto.
- Not chatty at all.

Protocol design

Handshake DoS

- Verifying authenticity using Curve25519 is expensive,
- Under load a server may send a cookie which needs to be echoed back,
- This proves IP ownership, hence IP rate limiting can be used (e.g. token bucket).
- The cookie is the MAC over the source IP with a secret which changes every 120s.

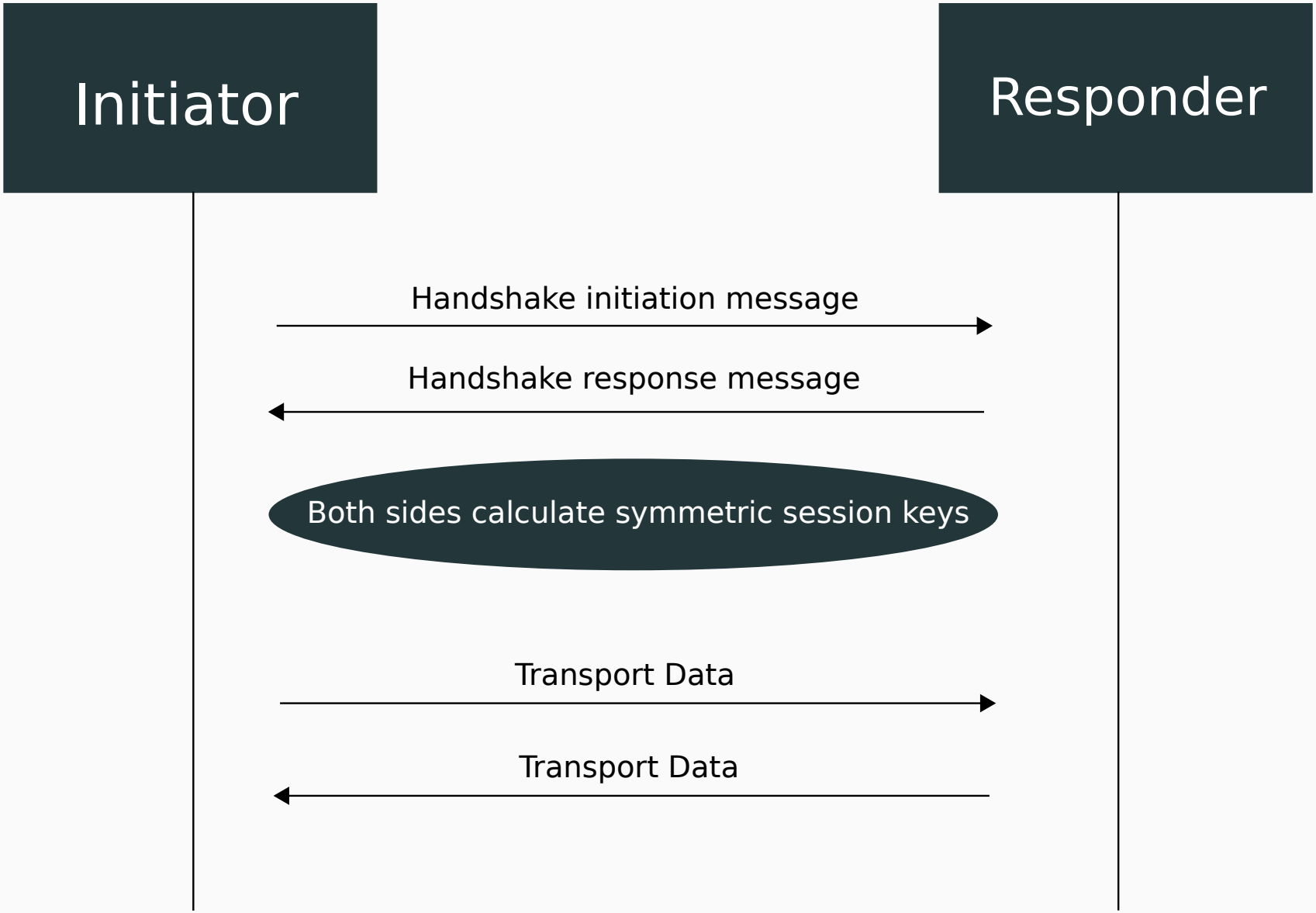
Flaws to be addressed

- Indescriminate cookie responses violate silence property,
- Cleartext cookies are vulnerable to MiTM replay attacks,
- Initator itself could be DoS'ed by being sent fake cookies

Issue 1: Silence

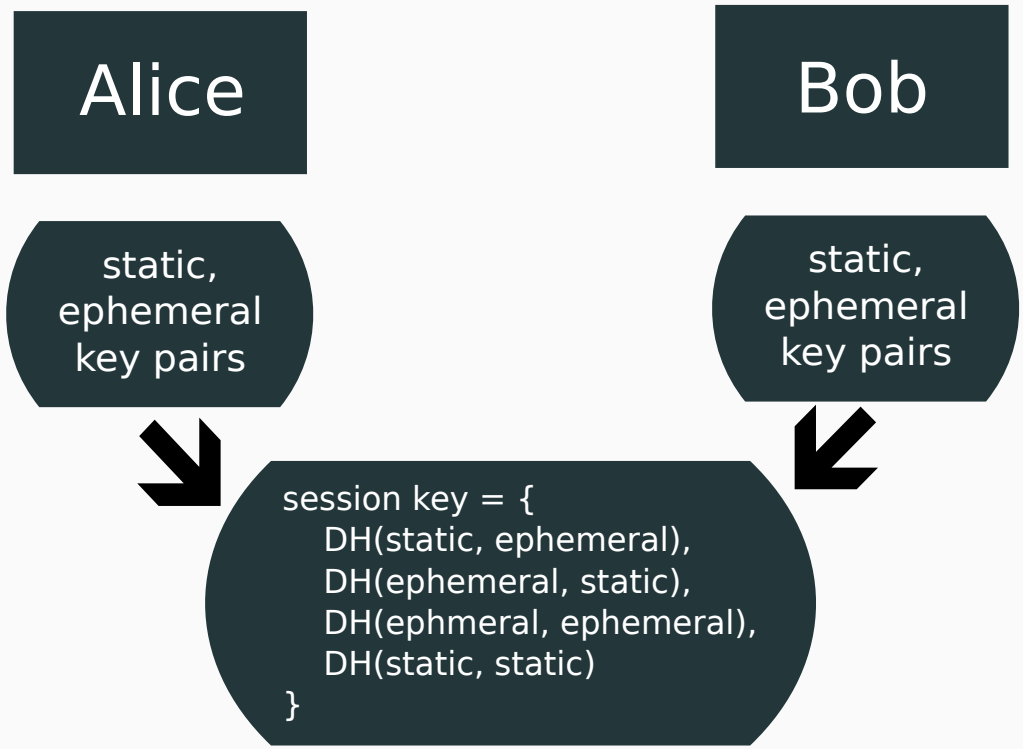
- For the responser to remain silent, all messages have a first MAC using the responder's public key,
- This proves that a peer knows to whom it is talking,
- While this public key is not secret, it is acceptable in the threat model to say if the initator knows the public key, then it knows of the existance of the server,
- This MAC is included in all packets as `msg.mac1`

The Key Exchange



Session key derivation - NoiseIK

- One peer is the initiator; the other is responder
- Each peer has their static identity - their long term *static keypair*
- For each new handshake, each peer generates an *ephemeral keypair*



Message 1: Initator to responder

Type := 1 (1 byte)	Reserved := 0 (3 bytes)
Sender := I_i (4 bytes)	
Ephemeral (32 bytes)	
Static (32 bytes)	
Timestamp (12 bytes)	
mac1 (16 bytes)	mac2 (16 bytes)

Figure 3: Initator to responder packet. NOT TO SCALE.

Message 2: Responder to initiator

type := 0x2 (1 byte)	reserved := 0 (3 bytes)
sender := I_r (4 bytes)	receiver := I_i (4 bytes)
Ephemeral (32 bytes)	
Empty (0 bytes)	
mac1 (16 bytes)	mac2 (16 bytes)

Figure 4: Responder to initiator packet. NOT TO SCALE

Transport data messages

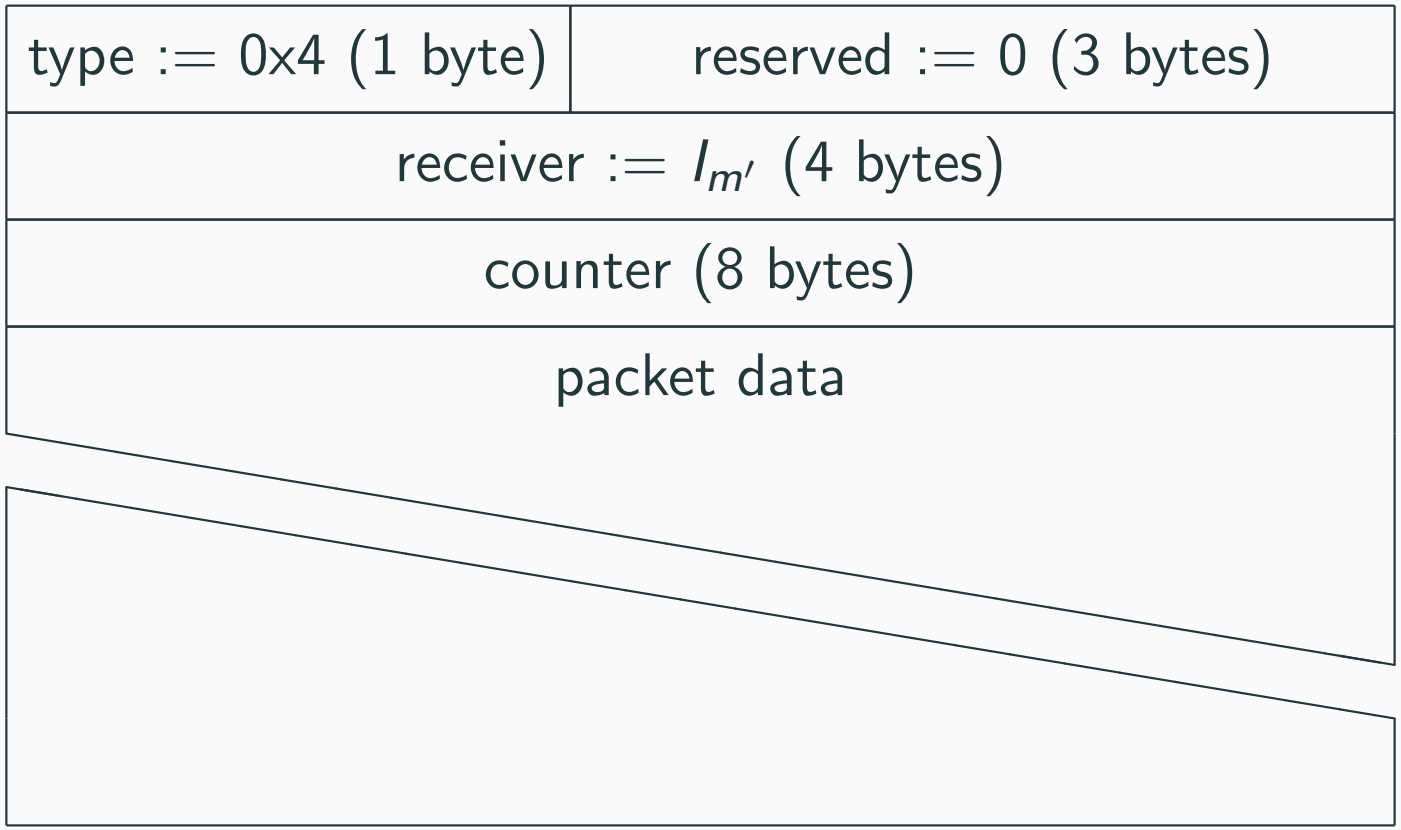


Figure 5: Payload packet. NOT TO SCALE