Department of Mathematics and Computer Science
Coding Theory and Cryptology Group

Secure Sessions for Ad Hoc Multiparty Computation in MPyC

Master thesis

Emil Nikolov

Id nr: 0972305
emil.e.nikolov@gmail.com

Supervisor : Dr. ir. L.A.M. (Berry) Schoenmakers

April 2, 2023

Contents

List of Abbreviations

E^3 Extensible Evaluation Environment. 11

ACL Access Control List. 9

CGNAT Carrier-Grade NAT. 6

CIDR Classless Inter-Domain Routing. 5

DERP Designated Encrypted Relay for Packets. 9

ECDH Eliptic Curve Diffie-Hellman. 5

IP Internet Protocol. 5, 6, 7

IPSec Internet Protocol Security. 5

ISP Internet Service Provider. 5

LAN Local Area Network. 5, 6, 7

NAT Network Address Translation. 5, 6

NAT-PMP NAT Port Mapping Protocol. 5

OSI Open Systems Interconnection. 3

P2P Peer to Peer. 6, 7, 8

PCP Port Control Protocol. 5

STUN Session Traversal Utilities for NAT. 5, 6

TCP Transmission Control Protocol. 5, 6

TLS Transport Layer Security. 5, 6

TURN Traversal Using Relays around NAT. 5

UDP User Datagram Protocol. 5, 6

UPnP Universal Plug and Play. 5

VPN Virtual Private Network. 5, 6, 7, 8

List of Figures

Chapter 1

Introduction

Chapter 2

Related work

In this chapter we will provide a high level overview of relevant solutions for connecting the parties of an MPyC multiparty computation over the internet. We will use the Open Systems Interconnection (OSI) model to help us categorize those solutions and highlight the similarities and differences between them. The OSI model has 7 layers with different responsibilities:

Layer 7 - Application: high level protocols that interact with user-facing services

Layer 6 - Presentation: translation of data between a networking service and an application, e.g. encoding, compression, encryption

Layer 5 - Session: session setup, management, tear-down, authentication, authorization

Layer 4 - Transport: sending data of variable length over a network while maintaining quality-of-service, e.g. ports, connections, packet fragmentation

Layer 3 - Network: sending data packets between two nodes, routed via a path of other nodes, e.g. addressing, routing

Layer 2 - Data link: sending data frames between two nodes, directly connected via a physical layer, e.g. on a LAN

Layer 1 - Physical: sending raw bits over a physical medium

While many protocols implement aspects of several layers and do not strictly fit inside the OSI model, it still a useful visualization tool. The diagram below shows an approximate OSI model mapping of several protocols and network overlay solutions from the point of view of the systems that use them and the arrows show dependency relations between them.

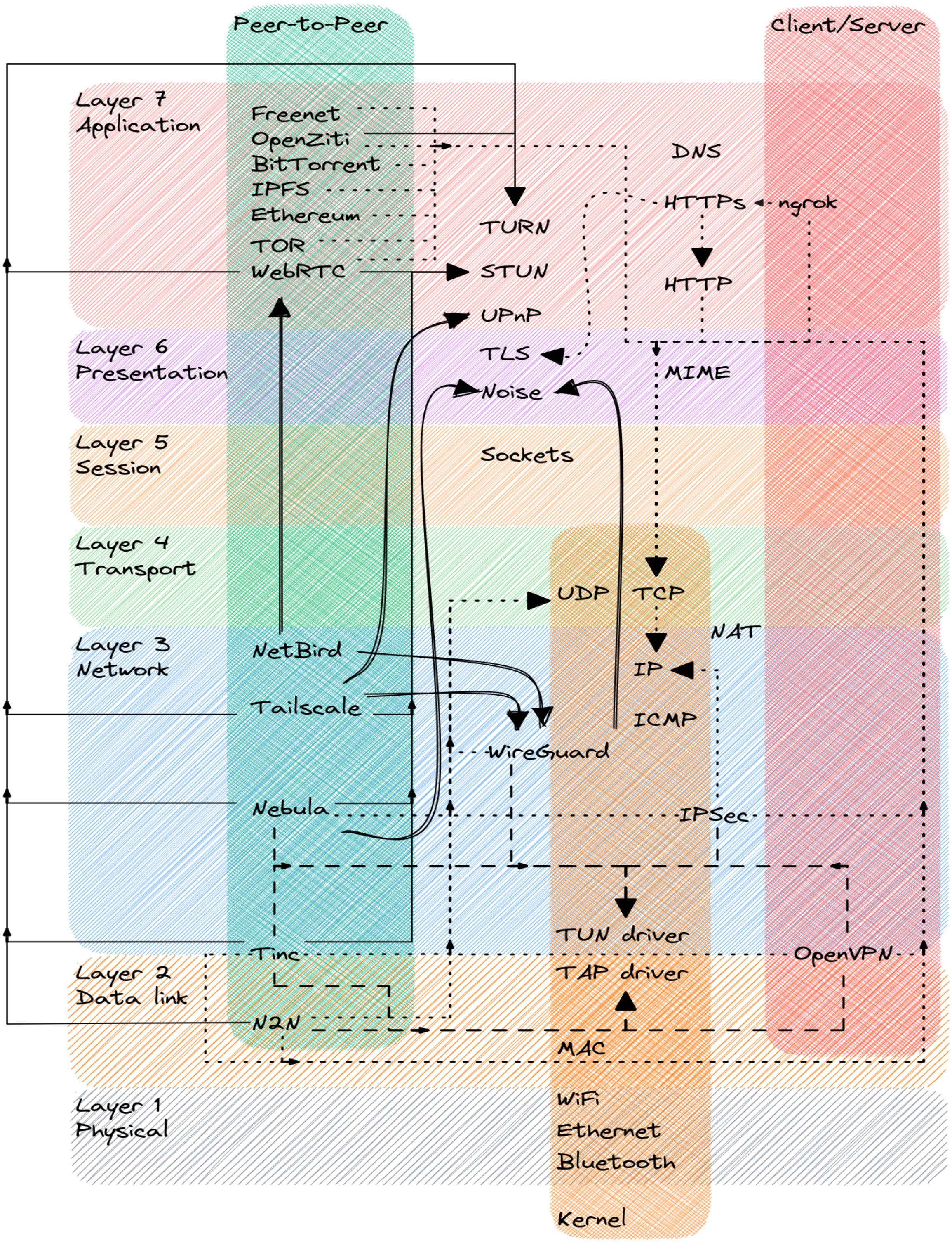

Figure 2.1: OSI model mapping of various protocols

2.1 The Internet

The modern Internet is a global multi-tiered network of devices that communicate using the protocols of the Internet Protocol Suite (TCP/IP). Typically, an Internet Service Provider (ISP) manages the physical infrastructure that connects an end-user's devices with the rest of the internet.

The **Internet Protocol (IP)** is a Network (Layer 3) protocol of the Internet Protocol Suite that is responsible for delivering datagrams between devices across the boundaries of their Local Area Networks (LANs) by possibly routing traffic via multiple intermediate devices (routers). A datagram is a connectionless packet that is delivered on a best effort basis. It has a header that contains fields such as the IP addresses of its source and destination, and a payload that encapsulates the data from the protocols of the layers above. Routers are devices that are part of multiple networks and relay datagrams between them based on a routing table that maps IP address ranges (Classless Inter-Domain Routing (CIDR)) to networks. IPv4 is the version of the Internet Protocol that was originally deployed globally and it is still the most widely used version today. It uses 32-bit numbers as IP addresses, allowing for up to with 32 bitThe IP addresses has a limited address space of 32 w globally deployed Nowadays, the most widespread version of the Internet Protocol is IPv4, which versions of the Internet Protocol IPv4 and IPv6 are

User Datagram Protocol (UDP) and **Transmission Control Protocol (TCP)** are Transport (Layer 4) protocols that add the concept of ports to allow having multiple communication channels between two devices. UDP provides best-effort delivery, while TCP is a reliable transport with delivery guarantees. TCP maintains stateful connections and handles acknowledgements and retransmissions in case of packets lost in transit.

Transport Layer Security (TLS) is a protocol that adds encryption on top of a reliable transport protocol such as TCP. It is usually placed in the Presentation Layer (Layer 6), but it does not strictly fit in any single OSI layer. tls

The **Noise Protocol Framework** [Per18] is a more recent effort that applies the ideas of TLS in a simplified way by serving as a blueprint for designing use-case specific protocols for establishing secure communication channels based on Eliptic Curve Diffie-Hellman (ECDH) handshake patterns. It powers the end-to-end encryption in messaging applications such as WhatsApp and Signal, and Virtual Private Network (VPN) software such as WireGuard and Nebula. transport agnostic

limited cipher suites

- **Network Address Translation (NAT)**
  - **Network Address Translation (NAT) traversal**
    - **Universal Plug and Play (UPnP), Port Control Protocol (PCP), NAT Port Mapping Protocol (NAT-PMP)**
      - allow configuring port forwarding in a router
      -
    - **Session Traversal Utilities for NAT (STUN), Traversal Using Relays around NAT (TURN)**
- **Internet Protocol Security (IPSec)**
  - part of the Internet Protocol Suite
  - used inside VPN software

  - has implementations in both user and kernel space as well as hardware implementations
  - rewrites the IP headers

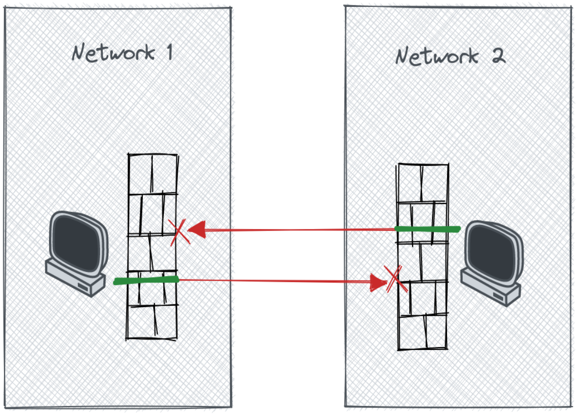

Figure 2.2: Two parties behind separate NATs

As we mentioned in the introduction chapter, the devices in a typical home network can only initiate connections to public endpoints (via NAT) but cannot be discovered from outside their LAN. This poses a challenge when two parties who want to communicate via a direct link are both behind separate NATs 2.2 and neither can be contacted by the other one first. Mesh VPNs solve this issue via NAT traversal techniques such as User Datagram Protocol (UDP) hole punching based on concepts from Session Traversal Utilities for NAT (STUN). The machines of each party can contact a public STUN server 2.3, which will note what Internet Protocol (IP) addresses the connections come from and inform the parties. Since the parties initiated the connection to the STUN server, their routers will keep a mapping between their local IP addresses and the port that was allocated for the connection in order to be able to forward the incoming traffic. Those "holes" in the NATs were originally intended for the STUN server, but mesh VPNs use the stateless "fire and forget" UDP protocol for their internal communication, which does not require nor provides a mechanism for the NATs to verify who sent a UDP packet. With most NATs, this is enough to be able to (ab)use the "punched holes" for the purpose of Peer to Peer (P2P) traffic from other parties. Mesh VPNs implement the stateful Transmission Control Protocol (TCP) and Transport Layer Security (TLS) protocols on top of UDP and expose an regular network interface to the other programs, keeping them shielded from the underlying complexities. Other NAT implementations such as Symmetric NAT and Carrier-Grade NATs (CGNATs) can be more difficult to "punch through" due to their more complex port mapping strategies. In those cases, establishing P2P connections might involve guess work or even fail and require falling back to routing the (encrypted) traffic via another party or service.

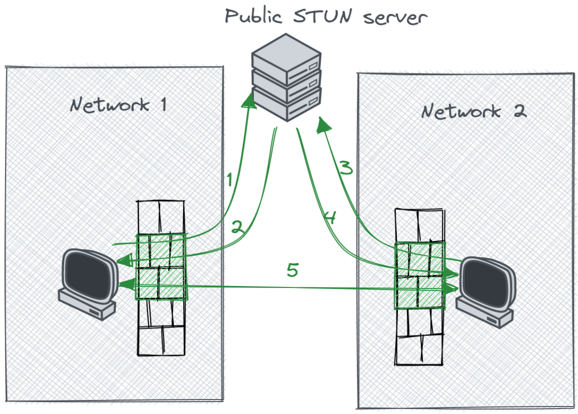

Figure 2.3: NAT traversal via STUN

2.2 Network overlays

2.2.1 TUN/TAP driver

- Layer 2 vs Layer 3 Networks
  - Layer 2 overlays bridge networks
    - virtual network switch
    - remote machines are on the same virtual LAN and can share the same IP address range
    - allows broadcast/multicast
    - TAP driver
  - Layer 3 overlays route traffic between separate local networks
    - virtual network router
    - remote machines are on separate LANs
    - simpler to configure
    - TUN driver

2.2.2 Traditional VPNs

VPNs are implemented as Layer 2 or 3 network overlays. They are commonly used for securely connecting machines from different LANs. They provide software emulation of a network interface controller via a TUN/TAP driver on the operating system level and allow other software to transparently use the functionality of the IP suite without requiring extra changes. Traditional VPNs such as IPSec[**ipSecDocs**] and OpenVPN[Ope22] use a centralized service that all (encrypted) client communications must pass through. This introduces a single point of failure and a potential bottleneck that might negatively impact the performance of the multi-party computations due to their P2P nature.

2.2.3 WireGuard

WireGuard[Don17] is a more recent protocol with a design informed by lessons learned from IPSec and OpenVPN and a key management approach inspired by SSH. It is a lower level protocol that focuses on configuration simplicity while network topology, peer discovery and key distribution are left as a responsibility of higher level systems that use it as a building block. Wireguard is implemented as a Layer 3 overlay over UDP tunnels. WireGuard has both user space implementations that use a TUN driver and also has direct support built into the Linux Kernel since version 5.6 (May 2020). The kernel implementation allows for better performance because it does not need to copy packets between the kernel and user-space memory.

The snippets below show a minimal set of configuration options that need to be provided in order for two peers to be able to form secure tunnels with each other.

```
# peer1.conf
[Interface]
Address = 101.0.0.1/32
ListenPort = 53063
PrivateKey = ePTiXXhHjvAHdWUr8Bimk30n0gh3m241RAzsN0JZDW0=

[Peer]
PublicKey = BSn0ejd1Y3bKuD+Xpg0ZZe0f+Ies/oql0NZxw+S0mkc=
AllowedIPs = 101.0.0.2/32
Endpoint = peer1.example.com:38133
```

```
# peer2.conf
[Interface]
Address = 101.0.0.2/32
ListenPort = 38133
PrivateKey = sN/d6XUPEVPGSziVgCC0n0ivDK+qAoYC3nxnssQ5Rls=

[Peer]
PublicKey = e/TxvPmrgcc1G4cSH2bHv5J0PRHXKjYxTFoU8r+G93E=
AllowedIPs = 101.0.0.1/32
```

Each peer has a public/private key pair that is used for authentication and encryption based on the Noise Protocol Framework[Per18]. The `Address` field specifies the virtual IP address that the local network interface will use, while the `AllowedIPs` field specifies what virtual IP addresses are associated with a peer's public key. A peer's `Endpoint` field specifies the URL at which it can be reached. Only one of the peers must be configured with a reachable endpoint for the other one. In the above example once `peer1` initiates communication with `peer2`, `peer2` will learn the current endpoint of `peer1` and will be able to communicate back with it.

2.2.4 Mesh VPNs

- Tinc
- N2N
- Tailscale
- Nebula
- ZeroTier

Mesh VPNs such as Tinc[Sli22], Tailscale[Tai] and Nebula[Def22] utilize NAT Traversal techniques in order to create direct P2P links between the clients for the data traffic. Authentication, authorization and traffic encryption are performed using certificates based on public key cryptography.

All three are open-source, with the exception of Tailscale's coordination service which handles the peer discovery and identity management. Headscale [Fon22] is a community driven open-source alternative for that component. Tinc is the oldest of the three but has a relatively small community. It is mainly developed by a single author and appears to be more academic than industry motivated. Nebula and Tailscale are both business driven. Tailscale was started by a number of high profile ex-googlers and is the most end-user focused of the three, providing a service that allows people to sign up using a variety of identity providers including Google, Microsoft, GitHub and others. They also provide an Admin console that allows a user to easily add their personal devices to a network or share them with others. It also has support for automation tools like Terraform for creating authorization keys and managing an Access Control List (ACL) based firewall. Nebula was originally developed at the instant messaging company Slack to create overlay networks for their cross region cloud infrastructure, but the authors later started a new company and are currently developing a user-centric platform similar to Tailscale's. Nebula is more customizable than Tailscale and since it is completely open-source it can be adapted to different use cases, but it is also more involved to set up. A certificate authority needs to be configured for issuing the identities of the participating hosts. Furthermore, publicly accessible coordination servers need to be deployed to facilitate the host discovery. Tailscale employs a distributed relay network of Designated Encrypted Relay for Packets (DERP) servers, while Nebula can be configured to route via one of the other peers in the VPN.

2.2.5 Layer 7 overlays

- WebRTC
- OpenZiti
- ngrok
- TOR
- BitTorrent
- IPFS
- Ethereum
- Teleport
- Freenet

Chapter 3

Testing methodology

In the following chapters we will design and implement several solutions for ad hoc MPC sessions based on a subset of the previously discussed related works:

- Internet protocol
- Wireguard
- Tailscale
- Headscale
- ? Headscale with DID identity?
- ? WebRTC?
- Custom solution that automates the wireguard configuration by visiting a web page

Additionally we will analyse and compare them in terms of performance, security and usability

3.1 Measuring performance

During the preparation phase of the project we developed the Extensible Evaluation Environment (E^3) framework which simplifies and automates the process of deploying machines in different geographical regions, connecting them via an overlay network and executing multiparty computations between them, where each machine represents a different party.

To summarize, E^3 is a set of scripts that use a number of automation tools:

- Terraform - declarative provisioning
- NixOS - declarative Linux distribution
- Colmena - declarative deployment for NixOS
- PSSH - parallel execution of remote scripts over ssh
- DigitalOcean - a cloud provider

It allows us to quickly provision cloud virtual machines in multiple regions and reproducibly deploy all necessary software for running a multiparty computation over a chosen network overlay solution. The source code of E^3 can be found on GitHub

Each solution will be deployed using the E^3 framework and the performance will be quantitatively measured in terms of the time it takes to execute a number of MPyC demos. The selected demos have different complexities in terms of communication rounds and message sizes which will allow us to observe their impact on the overall performance.

1. Secret Santa - high round complexity with small messages
2. Convolutional Neural Network (CNN) MNIST classifier - low round complexity with large messages

The demos will be configured at three different input size levels

- Low,
- Medium
- High

Furthermore, the demos will be executed in several networking scenarios:

1. 1-10 parties in the same geographic region
2. 1-10 parties evenly distributed across two nearby regions
3. 1-10 parties evenly distributed across two distant regions
4. 1-10 parties distributed across multiple distant regions

3.2 Security

We will analyze aspects such as

- key distribution
- trust model - are there any trusted third parties and what would be the consequences if they are corrupted or breached
- traffic encryption
- identity strength

3.3 Usability

For each solution we will describe the steps that the parties need to perform in order to execute a joint multiparty computation. Those steps will be analyzed in terms of:

- Complexity - how much technical expertise is expected from the parties in order to be able to execute the steps
- Initial effort - how much effort is each party expected to put in preparing for their first joint computation
- Repeated effort - after the initial setup, how much effort is required to perform another computation
  - with the same set of parties
  - with another set of parties
- Finalization effort - how much effort is required to finalize the MPC session once it is complete and clean up any left-over artifacts or resources so that the machine of each party is in its original state

Chapter 4

Internet protocol

This solution focuses on directly using the internet protocol without involving an overlay network. Our goal is to analyze the implications of using only the functionalities that MPyC directly supports to serve as the reference for our later experiments.

4.1 Implementation details

We will manually set up the multiparty computations via the public IP addresses of the machines and DNS.

4.2 Performance analysis

4.3 Security analysis

4.4 Usability analysis

Chapter 5

WireGuard

This solution creates an overlay network by manually configuring WireGuard on each machine.

5.1 Implementation details

5.2 Performance analysis

5.3 Security analysis

5.4 Usability analysis

Chapter 6

Tailscale

Tailscale is a VPN solution that configures a mesh of direct Wireguard tunnels between the peers.

6.1 Implementation details

6.2 Performance analysis

6.3 Security analysis

6.3.1 Trust model

There is a centralized service that deals with the key distribution, which needs to be trusted to provide the correct public keys for the correct parties

6.3.2 Identity

Identity is based on third party identity providers such as Microsoft and GitHub

- Magic DNS
-

6.4 Usability analysis

With tailscale each party needs to

- register a Tailscale account
- Download and install tailscale on the machine they want to run a multiparty computation
- Run tailscale on their machine and logs into their account in order to link it to their own Tailnet
- Share their Tailscale machine with the Tailnets of each of the other parties
- Download the demo they want to run
- Form the flags for running the chosen demo

  - add -P $HOST:$PORT for each party using their Tailscale hostname/virtual IP
- Run the demo

Chapter 7

Headscale

This solution is similar to the Tailscale one, but it uses Headscale - a self-hosted open-source alternative to the closed-source Tailscale coordination service.

7.1 Implementation details

7.2 Performance analysis

7.3 Security analysis

7.3.1 Trust model

There still is a centralized service like in the Tailscale solution, but here it is self-deployed.

7.3.2 Identity

7.4 Usability analysis

Bibliography

[Def22] Defined. *Nebula: Open Source Overlay Networking | Nebula Docs*. 2022. URL: https://docs.defined.net/docs/ (visited on 12/01/2022) (cit. on p. 8).

[Don17] Jason A. Donenfeld. “WireGuard: Next Generation Kernel Network Tunnel”. In: *Proceedings 2017 Network and Distributed System Security Symposium*. Network and Distributed System Security Symposium. San Diego, CA: Internet Society, 2017. ISBN: 978-1-891562-46-4. DOI: 10.14722/ndss.2017.23160. URL: https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/wireguard-next-generation-kernel-network-tunnel/ (visited on 09/28/2022) (cit. on p. 7).

[Fon22] Juan Font. *Juanfont/Headscale*. Dec. 6, 2022. URL: https://github.com/juanfont/headscale (visited on 12/06/2022) (cit. on p. 9).

[Ope22] OpenVPN. *Community Resources*. OpenVPN. 2022. URL: https://openvpn.net/community-resources/ (visited on 11/30/2022) (cit. on p. 7).

[Per18] Trevor Perrin. “The Noise Protocol Framework”. In: (July 1, 2018) (cit. on pp. 5, 8).

[Sli22] Guus Sliepen. *Tinc Docs*. Nov. 30, 2022. URL: https://www.tinc-vpn.org/docs/ (visited on 11/30/2022) (cit. on p. 8).

[Tai] Tailscale. *Tailscale*. Tailscale. URL: https://tailscale.com/kb/ (visited on 11/30/2022) (cit. on p. 8).