Chapter 1

Intro

- Emphasize on the difference between the Solutions Test Framework (E3) and the actual Solutions
- The Test Framework is an engineering problem of its own which deals with the scaffolding necessary to demonstrate the candidate solutions in realistic scenarios

Chapter 2

Related work

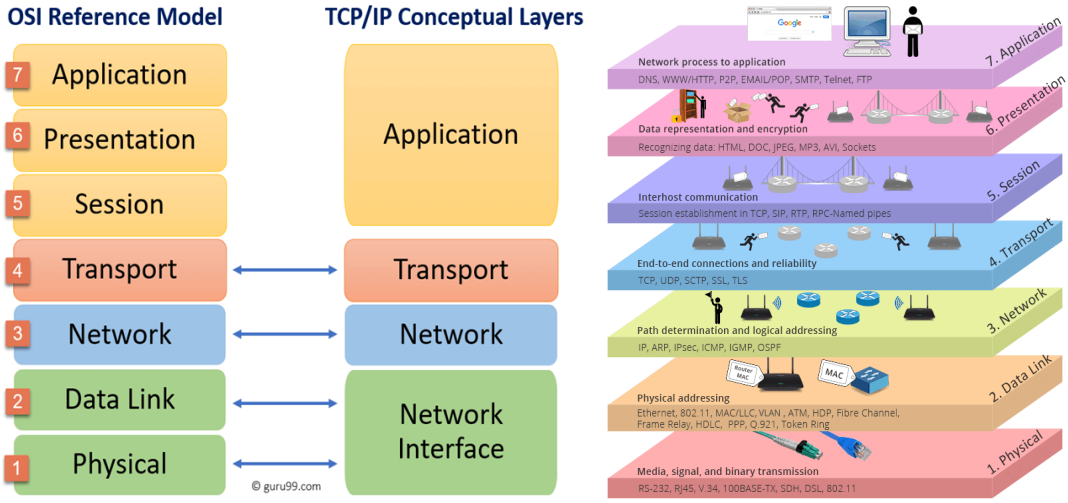
- There is a huge amount of primitives and partial solutions that are relevant to our problem, which makes it difficult to review them all, so we will try to design a classification system for them and focus on one or two solutions per class while only briefly mentioning their alternatives
- Make a distinction between
  - primitives - lower level concepts, techniques or frameworks that cannot be used as a solution directly, but are used inside higher level solutions
    - STUN/TURN/ICE
    - Noise Protocol Framework
    - Identity solutions
  - In terms of topology
    - Star shaped
    - Peer-to-peer
  - In terms of OSI layers:
    - Virtual Network Interface Controller - solutions that emulate a physical network controller in the OS using TUN/TAP driver. Most networked applications are already designed to work with the Internet Protocol Stack (Suite - definitions; Stack - implementations) in the host operating system and they do not need to do anything extra to work with a virtual NIC. Those solutions are usually more general purpose as they allow any other application on the host machine to use the overlay network. Installing an application that also uses this type of a solution may be unintuitive because it will essentially install 2 separate programs - the actual application and another one that manages the virtual network, while the virtual network will not be limited to our application but will also be usable by the others.
      - IPSec
      - Wireguard
      - Tailscale
      - .
    - Application layer - require the applications to be implemented in a way that supports the protocols of the solution by using additional libraries or SDKs

to facilitate the communications. Those solutions result in more purpose built applications as it is not possible for an application to use the overlay network if it wasn't specifically designed to be able to.

- WebRTC
- OpenZiti
-

2.1 OSI Model

- Create a figure that maps the various related components to OSI model layers
- It can be a "more or less accurate" artistic impression since the components can cover multiple layers and the layers are somewhat fluid



The Open Systems Interconnection (OSI) model distinguishes 7 layers in computer networks:

OSI Layer	Description	Protocols	Network overlays
7. Application	High level protocols	HTTP, HTTPS, DNS, FTP, SMTP	WebRTC, OpenZiti, ngrok, TOR, BitTorrent, Ethereum, Teleport, Freenet
6. Presentation	Translation of data between a networking service and an application, e.g. encoding, compression, encryption	MIME, TLS*	Noise Protocol Framework
5. Session	continuous exchange of information		
4. Transport	Variable length data over a network while maintaining quality-of-service, e.g. ports, connections, packet splitting	UDP, TCP	
3. Network	data packets between two nodes, routed via a path of other nodes, e.g. addressing, routing	IP, ICMP	OpenVPN, IPSec, Tinc, Wireguard, Tailscale, Nebula, ZeroTier
2. Data link	data frames between two nodes, directly connected via a physical layer		N2N
1. Physical	sending raw bits over a physical medium		

- Layer 7 - Application - high level protocols
  - Protocols
    - HTTP
    - HTTPS
    - DNS
    - FTP
    - SMTP
  - Solutions
    - WebRTC
    - OpenZiti
    - ngrok
    - TOR
    - BitTorrent
    - IPFS
    - Ethereum
    - Teleport
    - Freenet
- Layer 6 - Presentation - Translation of data between a networking service and an application

  - Concepts
    - encoding
    - compression
    - encryption
  - Protocols
    - MIME
    - SSL/TLS*
  - Solutions
    - Noise Protocol Famework

- Layer 5 - Session - continuous exchange of information
- Layer 4 - Transport - Variable length data over a network while maintaining quality-of-service
  - Concepts:
    - Ports
    - Stateful Connections / Connectionless
    - Splitting of variable length data into multiple smaller packets
  - Protocols:
    - TCP
    - UDP
- Layer 3 - Network - data packets between two nodes, routed via a path of other nodes
  - Concepts:
    - Routing
    - IP Addresses
  - Protocols:
    - IP
    - ICMP
  - Solutions
    - IPSec
    - OpenVPN
    - Tinc
    - Wireguard
- Layer 2 - Data link - data frames between two nodes, directly connected via a physical layer
  - Solutions
    - OpenVPN in network bridge mode
    - Tinc
    - N2N
- Layer 1 - Physical - sending raw bits over a physical medium
- Resources
  - The OSI model doesn't map well to TCP/IP
  -

2.2 Primitives

2.2.1 IP Stack

- IP addresses
- Routing
- DNS
- TLS

Network Address Translation (NAT) Traversal

https://bford.info/pub/net/p2pnat/

https://www.jordanwhited.com/posts/wireguard-endpoint-discovery-nat-traversal/

Session Traversal Utilities for NAT (STUN)

- Uses a STUN server for discovery and UDP hole-punching
- Communications are peer-to-peer
- Examples:
  - https://github.com/shawwwn/Gole
  - https://github.com/malcolmseyd/natpunch-go

Traversal Using Relays around NAT (TURN)

- Peers use a relay server as a mediator to route traffic

Interactive Connectivity Establishment (ICE)

- Umbrella term covering STUN/TURN and other related techniques

Designated Encrypted Relay for Packets (DERP)

- TURN-like protocol by Tailscale
- Relaying encrypted Wireguard traffic over HTTP
- Routing based on the Peer's public key
- Overview - https://tailscale.com/kb/1232/derp-servers/
- Source code
  - https://github.com/tailscale/tailscale/tree/main/cmd/derper
  - https://github.com/tailscale/tailscale/blob/main/derp/derp.go

2.2.2 Noise Protocol Framework

- Framework for building protocols
- Spec - http://www.noiseprotocol.org/noise.pdf
- Suite of channel establishment protocols
- Similar to TLS
- Based on Elliptic-curve Diffie–Hellman (ECDH) Handshakes
- Used in WhatsApp, Signal, Wireguard
- Resources:

  - Crypto Layers
    * Low-level primitives - AES, ChaCha20.
    * Usefully combined primitives - AES-OCB, NaCl secretbox...
    * High-level protocols - TLS, Noise.
    * (sometimes) crypto spoken over the encrypted protocols, often for E2E crypto e.g. GPG over SMTPS, or CloudFlare blinded CAPTCHA tokens over HTTPS
  - Design and Explore Noise Handshake Patterns
  - An Introduction to the Noise Protocol Framework
  - The Noise Protocol Framework (Video)
  -

2.2.3 Identity

- Identity based on a third party Identity Provider (Google, Microsoft, Government, ...)
- Self Sovereign Identity (SSI)

2.3 Basic Internet Protocol solution

- Solution with 0 overhead to be used as a reference for the performance of the other solutions
- No encryption
- Requires public IP addresses or DNS
- Easy for us to create a demo for measuring the performance because we control all of the machines that represent the different parties
- In a real life scenario with separate parties that are not under a shared control, it will be difficult to use this solution because it requires a lot of manual configuration and coordination between the parties

2.4 Virtual Private Networks (VPNs)

"A virtual private network [4] is a secure logical network that is tunnelled though another network. VPNs are often used for implementing secure point-to-point communications through the public Internet. Therefore they usually feature user authentication and content encryption." – n2n whitepaper

- Traditional centralized VPNs
  - OpenVPN
  - IPSec
- Mesh VPNs
  - Direct peer-to-peer tunnels between the participants when possible
  - Relaying via an intermediary when necessary
  - Tinc has been around for a long time (first release in 2000)
  - Newer solutions like Tailscale are modernized and more accessible
- Resources:
  - A Framework for IP Based Virtual Private Networks (RFC 2764)
  -

2.5 Wireguard

- Low level VPN Protocol
- Used by Tailscale
- Whitepaper - https://www.wireguard.com/papers/wireguard.pdf
- Built with the Noise Protocol Framework
- Added to Linux Kernel 5.6 in May 2020
- Typically used as a building block in more complicated systems
- Layer 3 over UDP
  - Linux Kernel module
    - faster - packets are not copied between kernel memory and userspace memory
  - Userspace virtual TUN device
    - easier to update because it does not require specific kernel modules
    - available on windows
- Simple configuration
  - Each peer has a public/private key pair for authentication and traffic encryption
  - Each peer has a config file:

```
[Interface]
Address = 101.0.0.1/32
ListenPort = 53063
PrivateKey = ePTiXXhHjvAHdWUr8Bimk3On0gh3m241RAzsN0JZDW0=

[Peer]
PublicKey = BSn0ejd1Y3bKuD+Xpg0ZZe0f+Ies/oql0NZxw+S0mkc=
AllowedIPs = 101.0.0.2/32
Endpoint = 142.93.135.154:38133
PersistentKeepalive = 25
```

- Creates a virtual network interface in the operating system that looks like an additional network card and can be used for TCP/IP communications
- Handles the encryption of traffic
  - deals with handshakes and generating symmetric session keys
- Cryptokey routing - associates public/private key pairs with IP addresses
- Out of scope:
  - key distribution - managed manually or via other software that builds on top of wireguard
  - peer discovery - for each pair of peers, one needs to have an endpoint that can be reached by the other peer
- Resources
  - WireGuard: The Next-Gen VPN Protocol
  - WireGuard Endpoint Discovery and NAT Traversal using DNS-SD
  - Kernel Commit
  - Arch Linux Wiki

2.6 Tailscale

- Mesh VPN
- Built on top of Wireguard
- Coordination service
  - Closed source
  - Facilitates STUN/TURN for peer discovery and NAT traversal
  - Distributes wireguard public keys
  - Magic DNS
- Client
  - Open source
  - Interacts with the Coordination service
  - Configures Wireguard
- Implementations based on Tailscale
  - Tailnet per party
    - Parties have their own tailscale accounts and manage their own tailnet
    - Each party shares adds the machine they will use for MPC to their tailnet
    - They share that machine with the tailnets of the other parties
  - "Host" party manages a tailnet
    - The host party creates authorization keys for the machines of the other parties

        * Each party runs the tailscale client with their authorization key which lets them join the host party's tailnet

2.7 Headscale

- Mostly the same as Tailscale, but with an open source coordination service that must be self-hosted
- Describe the differences in the deployment and how it affects usability
- We implemented a headscale based solution in order to be able to gain some insights into how they deal with peer discovery via STUN/TURN and configure Wireguard, and whether we can modify it or use parts of it in our own custom solution that is a better fit for ad hoc mpc sessions.

2.8 Nebula

- Open source Mesh VPN
- Similar to Tailscale
- Does not use Wireguard
- NAT traversal via Lighthouses
- Built using the Noise Protocol Framework (used in Wireguard)
- Uses a Certificate Authority that needs to sign each Peer's certificate
  - Certificates contain
    * Peer's Virtual IP address
    * Peer's public key

2.9 ZeroTier

- Closed source Mesh VPN
- Similar to Tailscale
- Does not use Wireguard
-

2.10 N2N

- Open source Mesh VPN
- Initial release - 27.03.2008
- Layer 2 TAP in userspace over Layer 3
  - offers a virtual ethernet jack
  - virtual devices have MAC addresses
  - Multicast/broadcast via the Supernode
- Similar to Nebula
- Does not use Wireguard
- Does not use the Noise protocol
- NAT traversal by relaying via Supernodes
  - No STUN - https://github.com/ntop/n2n/issues/57

- Resources:
  - Whitepaper

2.11 Tinc

- Mesh VPN released in 2000
- Less modern
- Fewer features
- Slower development
- More academic
- Runs in userspace as a TAP device
- Resources:
  - Manual

2.12 DIDComm v2

- Protocol for communicating that uses DID for identity management
- Spec: https://identity.foundation/didcomm-messaging/spec/
- No sessions
  - messages are always encrypted with the public keys of the peers
- Routing happens via mediators
- Resources:
  - Decentralized Identifiers: Implications for Your Data, Payments and Communications

2.13 ngrok

- Paid service for creating public URLs for local services
- Never peer-to-peer
- The traffic always goes through their centralized service
-

2.14 Ethereum's P2P Protocol

2.15 IPFS

- Content Addressable Storage
- Network is for discovering data, not for executing computations
-

2.16 TOR

- The Onion Router has the concept of Onion Services, which receive an address under the .onion pseudo top level domain and correspond to a public key (e.g. vww6ybal4bd7szmgncyruucpgfkqahzddi37ktceo3ah7ngmcopnpyyd.onion)

- Can be used as a privacy layer in other solutions to hide the real IP addresses of the parties

2.17 Freenet

- Whitepaper
- Friend2Friend network
- Peer to Peer Dark Web network
- Network that directly connects the machines of people who know each other
- Services and files accessible only by other people on the network
- Application layer
- Similar software:
  - retroshare
  - gnunet
  - waste
  - peerkeep
  - camlistore

2.18 Teleport

- Layer 7
-

2.19 BitTorrent

- No concept of identity based communication or peer discovery

2.20 WebRTC

- Peer to peer communications for browsers
  - can also work without a browser
  - Mainly used for multimedia communications - Peer-to-peer Audio/Video/VoIP
- Spec - https://www.w3.org/TR/webrtc/
- Uses nat STUN/TURN/ICE
- Data is encrypted
- Identity
  - Session Description Protocol (SDP)
  - peer certificates are generated and announced over SDP
  - ICE Candidates are negotiated for STUN/TURN connections
- Not a VPN
  - I think it can't serve as a TCP/IP network overlay that other applications can use
- Does not require additional plugins or native apps
- We could design a solution based on WebRTC in a browser by compiling the MPyC demos to web assembly or using PyScript.
  - https://www.win.tue.nl/~berry/mpyc/pyscript.html

  - https://pyscript.net/
- There seem to be many publicly available services that can be used as ICE servers for WebRTC
  - stun.l.google.com:19302
  - https://gist.github.com/zziuni/3741933
- Examples
  - https://github.com/pion/webrtc
    - Go
  - https://github.com/pojntfx/weron
- Resources
  - https://webrtcforthecurious.com/
  - https://temasys.io/guides/developers/webrtc-ice-sorcery/
  - https://web.dev/webrtc-basics/

2.21 OpenZiti

- Network Overlay
- No STUN
- Works by relaying traffic through intermediaries
- Focused on Services
- Allows embedding into apps via an SDK
- Tunneler is a proxy that allows non-ziti aware applications to use the overlay network by intercepting their traffic

Chapter 3

Implementation notes

3.1 Headscale

- Docker seems to have a 50-100% performance penalty (possibly due to docker's internal NAT) which makes the performance results of the headscale setup worse than they should be
- Modify the deployment setup to
  - not depend on all machines being on the same tailscale network
  - use the `*.demo.mpyc.tech` hostnames
  - the nodes should switch between the tailscale and headscale network depending on the demo script
  -

Chapter 4

Brainstorm for custom solutions

- Initial state
  - Here's my identity, here are the identities of the other parties
- Desired result
  - Executed MPC

With tailscale we'd need to

- Each party:
  - registers a Tailscale account
  - Downloads and installs tailscale on the machine they want to run the MPC on
  - Runs tailscale on their machine and logs into their account in order to link it to their own Tailnet
  - Shares their Tailscale machine with the Tailnets of each of the other parties
  - Downloads 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

- we don't need to use the same route for the communication Party A → Party B and Party B → Party A
- we can have something like an asynchronous STUN
- Party A sends a QR code/public URL/json object/DID document containing
  - Party A's public key
  - Party A's Mediator URL
- The mediator is a STUN/TURN/DERP server
  - other parties can either use it as a STUN server to find out how to access the hole in party A's NAT punched by the mediator
  - or use it as a relayer so that Party B can send encrypted packets to party A via the mediator
- Instead of a QR code, the information could be stored on a public ledger and could be resolved via DIDs

---

- There is a generic MPC wrapper program that deals with supporting tasks like generating identities and pulling MPC demos
- One party creates a "lobby" for an MPC session in their program and get a session id/public URL/QR code that can be shared with the other parties
-

- Public Website is visited by everyone
- They prove their identities using a SSI wallet or OIDC
- They get a QR code that serves as an invitation
  - contains their STUN based endpoint and identity
- Somehow everyone needs to scan each other's qr codes

- One party creates a "lobby" for an MPC session by visiting a public website
  - They provide their identity via OIDC/SSI wallet
- They get a public link/QR code that can be shared with the other parties
- The parties visit the URL and also provide their identities
- The parties obtain the MPC program they want to run
  - MPC program distribution could be done separately via cloning the github repo?
  - They could choose a DEMO and download it from the website?
  - There could also be a program running on the host machines that deals with the source code distribution. Similar to downloading custom maps for warcraft 3 or dota 2?
  - They could specify the source code when creating the MPC session in the website?
  - If the demo is not symmetrical where different parties have different roles and need to execute different programs, the roles could be assigned by the host or the people could choose their preferred role themselves?
- The parties download a configuration file that contains information on how to connect to the other parties
- They run the demos with the downloaded config file
- A temporary Wireguard mesh VPN is created between all parties

Chapter 5

Writing

- Ask questions and answer them
- Don't try to get the writing perfect on the first pass
- Write anything first and then iterate on it later

Chapter 6

Preparation phase report outline

6.1 Introduction

- – What is the purpose of this report?
  – How does it relate to the thesis assignment?
- Background information
  - What do I need to know to in order to be able to understand the problem that this thesis is trying to solve and its importance?
    * What is MPC?
    * What does MPC achieve?
    * When is MPC useful?
    * How does MPC work?
    * What is MPyC?
- Problem description
  - What is the issue with MPyC that this assignment is trying to solve?
- Research questions
- Scope

6.2 Technical survey

- Deployment
- Connectivity
- Summary

6.3 Implementation details

- Reproducible development
- NixOS image
- Provisioning

- Deployment
- Runtime execution

6.4 Conclusions

Chapter 7

Thesis outline

7.1 Introduction

- Background information
- Problem description
- Research questions
- Scope

7.2 Related work / Literature review

- Are there any existing solutions that can be applied to the problem in some capacity?

7.3 Methodology

- How are the solutions going to be evaluated?
- Performance
  - Which performance characteristics are relevant to multiparty computations?
  - How are the chosen solutions going to be implemented and deployed?
  - How are the performance metrics going to be measured?
- Security
  - What security aspects are relevant to multiparty computations?
  - How are they going to be analyzed?
- Usability
  - How is the usability of a solution going to be evaluated?

7.4 Solutions

- For each solution
  - What is the solution?
  - How was it implemented?
  - What are the security implications?
  - How can it be used by the parties?

  - What is the performance of the solution?

7.5 Conclusions

- How do the proposed solutions compare to each other?
- Which solution is preferred under which circumstances?

Chapter 8

Latex

- use \input instead of \include to avoid issues with missing directories and .aux files