



# Secure Sessions for Ad Hoc Multiparty Computation in MPyC

## Master thesis preparation phase

Emil Nikolov

Id nr: 0972305

[emil.e.nikolov@gmail.com](mailto:emil.e.nikolov@gmail.com)

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

# Outline

Introduction

Technical Survey

Reference Implementation

Demo

Planning

# Introduction

# Introduction

What is Secure Multi-Party Computation?

- Joint computation of a function
- Secret inputs
- Only final result is revealed

Examples:

- Millionaire's problem
- Secret Santa
- Electronic voting
- Auctions
- Machine learning

# MPC Basics

- Lagrange Interpolation
  - ▶ Set of  $t + 1$  points uniquely identify a polynomial of degree  $\leq t$
- Shamir's Secret Sharing
  - ▶  $(t, m)$ -threshold secret sharing scheme based on Lagrange Interpolation
  - ▶  $\geq t + 1$  shares to reconstruct the secret  $S$
  - ▶ Choose random polynomial  $f(x)$  of degree  $t$  where  $f(0) = S$
  - ▶ Share  $s_i = f(i)$ , for  $i \in [1, m]$
- Secure Multi-Party Computation
  - ▶  $m$  parties jointly compute a function  $f(S_1, S_2, \dots, S_m)$ , from their secret inputs
  - ▶ Party  $i$  secret shares its private input  $S_i$  with the others
  - ▶ Interactive protocol to reconstruct a polynomial  $g(x)$ , where  $g(0) = f(S_1, S_2, \dots, S_m)$

# Problem Description

MPyC:

- Python framework for MPC developed at TU/e
- No service discovery yet
- Target users of different level of expertise
  - ▶ Casual
  - ▶ Power
  - ▶ Enterprise
- MPC is Peer-to-Peer
- Local networks are tricky
  - ▶ Limited supply of IPv4 addresses
  - ▶ Slow adoption of IPv6
  - ▶ Network Address Translation (NAT)

## Research Questions

*How can MPyC be extended to enable casual users, power users and enterprises with limited prior knowledge of each other to discover each other and perform a secure multiparty computation under diverse networking conditions?*

- deployment strategies?
- identity?
- first contact?
- connectivity?
  - ▶ security?
  - ▶ privacy?
  - ▶ performance?

## Preparation Phase Scope

- Technical Survey
- Extensible Evaluation Environment ( $E^3$ ) - network of host machines for MPC
  - ▶ Simple
  - ▶ Extensible
  - ▶ Cross region
  - ▶ Cross platform
  - ▶ Automated
  - ▶ Reproducible
  - ▶ Disposable
- Implementation Phase Planning



## Technical Survey

# Technical Survey

- Deployment tools
- Connectivity approaches

# Infrastructure as Code (IaC)

## Tools:

- Provisioning - Terraform, CloudFormation
- Deployment - Ansible, Puppet, Chef

## Specification:

- Imperative - describes the steps to execute
- Declarative - describes the desired state

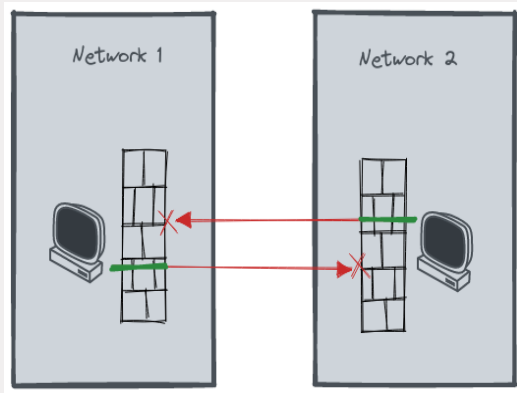
## Operating Systems:

- Most Linux distributions are imperatively managed
- NixOS
  - ▶ Declarative
  - ▶ Deployment tools: NixOps, Colmena, morph, deploy-rs

# Virtual Private Networks (VPNs)

- Centralized VPNs - OpenVPN, IPSec
  - ▶ Emulate a real network
  - ▶ Transparent to the other programs on the host
  - ▶ Single point of failure
  - ▶ Can be bottlenecked
- Mesh VPNs - Tailscale, Nebula, Tinc
  - ▶ Peer-to-Peer traffic
  - ▶ Discovery can happen via a public service or from a known peer

# Network Address Translation (NAT)



- Parties behind a NAT device (usually their router)
  - ▶ Can initiate a connection to a public endpoint
  - ▶ Cannot be discovered from the outside
  - ▶ Neither party can initiate the connection to the other

Figure 1: “Two parties behind separate NATs”

# NAT Traversal

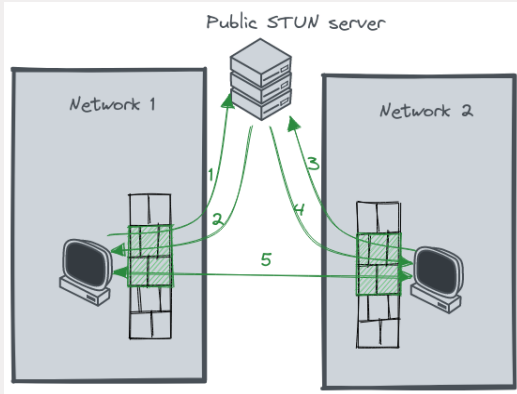


Figure 2: "NAT traversal via STUN"

- Session Traversal Utilities for NAT (STUN)
  - ▶ Parties connect to a public STUN server (can be another party)
  - ▶ The server reports the IPs it "sees" the parties at
  - ▶ User Datagram Protocol (UDP) hole punching
    - ▶ Reverse channel for the STUN server to talk back to a party
    - ▶ Appropriated by the other parties for their own traffic

## Other approaches

- Decentralized identifiers (DIDs) and DIDComm
  - ▶ Lack of sessions
  - ▶ Inefficient for MPC
- The Onion Router
  - ▶ Privacy
  - ▶ Onion services
- Peer-to-peer applications
  - ▶ Bit Torrents
  - ▶ Ethereum
  - ▶ IPFS

## Reference Implementation



## Reference Implementation

- DigitalOcean - cloud provider
- RaspberryPi - ARM based Single Board Computer (SBC)
- NixOS - declarative Linux distribution
- Terraform - declarative provisioning
- Colmena - declarative NixOS deployment
- Tailscale - mesh VPN as a Service
- prsync - sync directories to multiple hosts over ssh
- pssh - execute commands on multiple hosts in parallel over ssh

# Nix - Basic flake.nix

```
1 {  
2   inputs = {  
3     nixpkgs.url = "github:nixos/nixpkgs/nixos-unstable";  
4   };  
5  
6   outputs = inputs@{ self, nixpkgs, ... }:  
7     let  
8       pkgs = import nixpkgs {  
9         system = "x86_64-linux";  
10      };  
11     in  
12     {  
13       myHello = pkgs.hello;  
14     };  
15 }
```

# Nix - flake.lock

```
1 {  
2   "nodes": {  
3     "nixpkgs": {  
4       "locked": {  
5         "lastModified": 1666377499,  
6         "narHash": "sha256-dZZCGvWcxc7oGnUgFVf0UeNHsJ4VhkTM0v5JRe8EwR8=",  
7         "owner": "nixos",  
8         "repo": "nixpkgs",  
9         "rev": "301aada7a64812853f2e2634a530ef5d34505048",  
10        "type": "github"  
11      },  
12      "original": {  
13        "owner": "nixos",  
14        "ref": "nixos-unstable",  
15        "repo": "nixpkgs",  
16        "type": "github"  
17      }  
18    },  
19    ...  
20  }  
21 }
```

# Nix - Development Shell

```
1 devShell.x86_64-linux = pkgs.mkShell {
2   shellHook = ''
3     export PYTHONPATH=./
4   '';
5
6   nativeBuildInputs = [
7     pkgs.curl pkgs.jq
8     pkgs.colmena pkgs.pssh
9     (pkgs.terraform.withPlugins
10      (tp: [
11        tp.digitalocean tp.null
12        tp.external tp.tailscale
13        tp.random
14      ]))
15     mpyc-demo
16   ];
17 };
18
```

# Nix - MPyC Package

```
1 { pkgs, dir }:
2 (pkgs.poetry2nix.mkPoetryEnv {
3   python = pkgs.python3;
4   projectDir = dir;
5   extraPackages = (ps: [(pkgs.python3Packages.buildPythonPackage {
6     name = "mpyc";
7     src = dir;
8   })]);
9   overrides = pkgs.poetry2nix.overrides.withDefaults (
10     self: super: {
11       gmpy2 = pkgs.python3Packages.gmpy2;
12     }
13   );
14 })
```

# Nix - DigitalOcean Image (1)

```
1  ## flake.nix
2  {
3    inputs = {
4      nixpkgs.url = "github:nixos/nixpkgs/nixos-unstable";
5    };
6    outputs = inputs@{ self, nixpkgs, ... }:
7      let
8        mpyc-demo = (import ./nix/mpyc-demo.nix { inherit pkgs; dir = ./.; });
9        pkgs = import nixpkgs {
10          system = "x86_64-linux";
11        };
12        digitalOceanConfig = import ./nix/digitalocean/image.nix {
13          inherit pkgs;
14          extraPackages = [ mpyc-demo ];
15        };
16      in
17      {
18        packages.digitalOceanImage = (pkgs.nixos digitalOceanConfig).digitalOceanImage;
19      };
20  }
```

## Nix - DigitalOcean Image (2)

```
1  ## nix/digitalocean/image.nix
2  { pkgs, extraPackages ? [ ], ... }:
3  {
4    imports = [ "${pkgs.path}/nixos/modules/virtualisation/digital-ocean-image.nix" ];
5    system.stateVersion = "22.11";
6    environment.systemPackages = with pkgs; [
7      jq
8    ] ++ extraPackages;
9
10   services.tailscale.enable = true;
11
12   networking.firewall = {
13     enable = true;
14     checkReversePath = "loose";
15     trustedInterfaces = [ "tailscale0" ];
16   };
17 }
```

# Nix - RaspberryPi Image

```
1 let
2   mpyc-demo = (import ./nix/mpyc-demo.nix { inherit pkgs; dir = ./.; });
3
4   pkgs = import nixpkgs {
5     system = "aarch64-linux";
6   };
7 in
8 {
9   packages.raspberryPi4Image = (pkgs.nixos ({ config, ... }: {
10     system.stateVersion = "22.11";
11     imports = [
12       ("${pkgs.path}/nixos/modules/installer/sd-card/sd-image-aarch64-installer.nix")
13     ];
14
15     environment.systemPackages = [
16       mpyc-demo
17     ];
18   })).sdImage;
19 }
```



# Terraform - Image Import

```
1 resource "digitalocean_spaces_bucket_object" "nixos-image" {
2     region = digitalocean_spaces_bucket.tf-state.region
3     bucket = digitalocean_spaces_bucket.tf-state.name
4     key    = basename(var.nixos-image-path)
5     source = var.nixos-image-path
6     acl    = "public-read"
7     etag   = filemd5(var.nixos-image-path)
8 }
9
10 resource "digitalocean_custom_image" "nixos-image" {
11     name      = "nixos-22.11"
12     url       = "https://${digitalocean_spaces_bucket.tf-state.bucket_domain_name}/${digitalocean_spaces_bucket_object.nixos-image.key}"
13     regions  = local.all_regions
14     tags      = ["nixos"]
15
16     lifecycle {
17         replace_triggered_by = [
18             digitalocean_spaces_bucket_object.nixos-image
19         ]
20     }
21 }
```

# Terraform - Hostname Generation

```
1
2 locals {
3   node_definitions = var.DESTROY_NODES != "" ? [] : [
4     { region = "ams3", num = 3 },
5     { region = "sfo3", num = 1 },
6     { region = "nyc3", num = 1 },
7     { region = "sgp1", num = 1 },
8   ]
9   nodes_expanded = flatten([
10    for node in local.node_definitions : [
11      for i in range(node.num) :
12        merge(node, {
13          name = "mpyc-demo--${node.region}-${i}"
14        })
15    ]
16  ])
17  nodes = {
18    for node in local.nodes_expanded :
19    node.name => merge(node, {
20      hostname = "${node.name}-${random_id.mpyc-node-hostname[node.name].hex}"
21    })
22  }
23 }
```

# Terraform - DigitalOcean Droplets

```
1 resource "digitalocean_droplet" "mpyc-node" {
2   for_each = local.nodes
3
4   image   = digitalocean_custom_image.nixos-image.id
5   name    = each.value.hostname
6   region = each.value.region
7   size    = "s-1vcpu-1gb"
8   ssh_keys = [for key in digitalocean_ssh_key.ssh-keys : key.fingerprint]
9
10
11   provisioner "remote-exec" {
12     inline = [
13       "mkdir -p /var/keys/",
14       "echo ${tailscale_tailnet_key.keys.key} > /var/keys/tailscale",
15       "tailscale up --auth-key file:/var/keys/tailscale"
16     ]
17   }
18 }
19
20 resource "tailscale_tailnet_key" "keys" {
21   ...
22 }
```

# Colmena

```
1
2 let
3   mpyc-demo = (import ./nix/mpyc-demo.nix { inherit pkgs; dir = ./.; });
4
5   pkgs = import nixpkgs {
6     system = "x86_64-linux";
7   };
8
9   digitalOceanConfig = import ./nix/digitalocean/image.nix {
10     inherit pkgs;
11     extraPackages = [ mpyc-demo ];
12   };
13 in
14 {
15   packages.colmena = {
16     meta = {
17       nixpkgs = pkgs;
18     };
19     defaults = digitalOceanConfig;
20   } // builtins.fromJSON (builtins.readFile ./hosts.json);
21 };
```

## Colmena - hosts.json

```
1 ## hosts.json
2 {
3   "mpyc-demo--ams3-0-15e53f39": {},
4   "mpyc-demo--ams3-1-b4791c55": {},
5   "mpyc-demo--ams3-2-7f09fb08": {},
6   "mpyc-demo--nyc3-0-7dd0d9f6": {},
7   "mpyc-demo--sfo3-0-5bffc60e": {},
8   "mpyc-demo--sgp1-0-92700733": {}
9 }
```

# Runtime

```
1 prsync -h hosts.pssh -zarv -p 4 ./ /root/mpyc
2
3 pssh -h hosts.pssh -iv -o ./logs/$t "cd /root/mpyc && ./prun.sh"
```

```
1
2
3 # assemble $args and $MY_PID from the hosts.pssh file
4
5 ...
6
7
8 if [ $MY_PID = -1 ]
9 then
10     echo Only $i parties are allowed. $HOSTNAME will not participate in this MPC session
11 else
12
13 cmd="python ./demos/secretsanta.py 3 --log-level debug \
14     -I ${MY_PID} \
15     ${args}"
16
17 echo $cmd
18 $cmd
19 fi
20
```

Demo

## Demo

- Provisioning with Terraform
- Deployment with Colmena
- Running a distributed MPyC program
- Destruction of the infrastructure



# Planning

## Planning - connectivity implementations

- Headscale
- Nebula - IP allocation, Certificate authority, Certificate distribution
- Mesh VPN with alternative identity management
  - ▶ MPC based CA
  - ▶ Decentralized Identifiers
- DIDComm
  - ▶ sessions
  - ▶ NAT traversal
- TOR, Ethereum, IPFS
- Carbyne stack

## Planning - analysis

Compare the implementations in terms of:

- Security
- Performance
- Ease of use
- Privacy