

# Gosling: Build Anonymous, Secure, and Metadata-Resistant Peer-to-Peer Applications using Tor Onion Services

morgan (she/they)  
**[morgan@blueprintforfreespeech.net](mailto:morgan@blueprintforfreespeech.net)**

# Introductions

- Morgan
  - Software Developer working for Blueprint for Free Speech[1]
    - Blueprint for Free Speech
      - An international non-profit dedicated to promoting the right to freedom of expression.
      - Develop and maintain
        - Gosling Rust library (and some of its dependencies)
        - Ricochet-Refresh
  - Applications Team Lead at the Tor Project
    - Manage a team of software engineers developing Tor Browser, Tor Browser for Android, and Mullvad Browser

**1. Blueprint for Free Speech: <https://blueprintforfreespeech.net>**

# What is Gosling and What Does it Do?

- Rust library which provides authenticated peer-to-peer connectivity with the following features built-in:
  - End-to-End Encryption with Forward Secrecy
  - Anonymity
  - Hole Punching
  - Censorship Circumvention
  - Client Authentication
  - Metadata Resistance
  - Optional Application-Specific Authorisation Extensions

# How Does It Work?

- Each user has a unique id like:
  - 6l62fw7tqctlu5fesdqukvpoxezkaxbzllrafa2ve6ewuhzphxczszyd
- Users have only to share their id with other users, successfully complete a handshake, and they can connect to and send traffic to each other with all the afore-mentioned properties!

# Right.. But How Does It Work?

- Built on Tor and Tor Onion Services

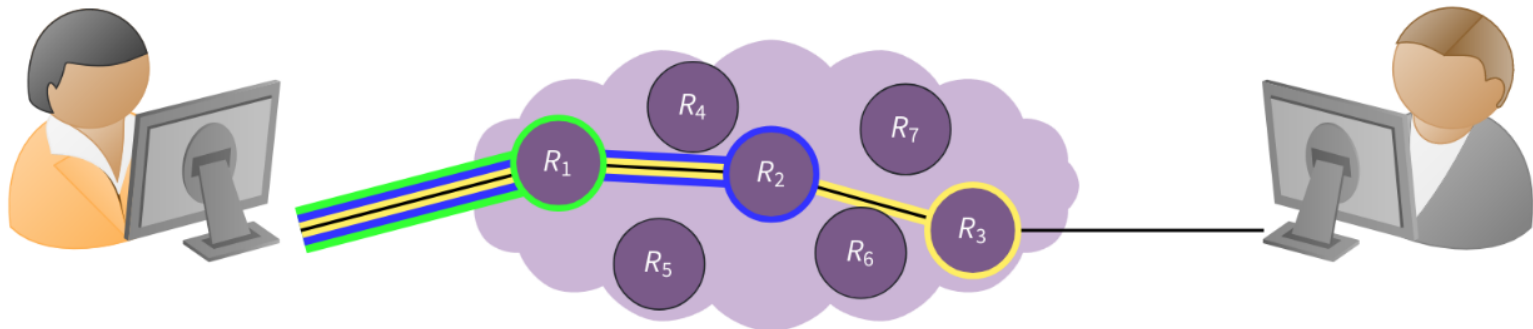
# Tor

- Tor Network is a community of relay operators, each running tor aka little-t tor, c-tor, or the legacy tor daemon
- Users create circuits to their destination within the Tor Network:
  - 1st Hop - Guard Relay: knows IP address of user and guard relay
  - 2nd Hop - Middle Relay: knows the guard relay and the exit relay
  - 3rd Hop - Exit Relay: knows middle relay, final destination and contents of traffic

Alice

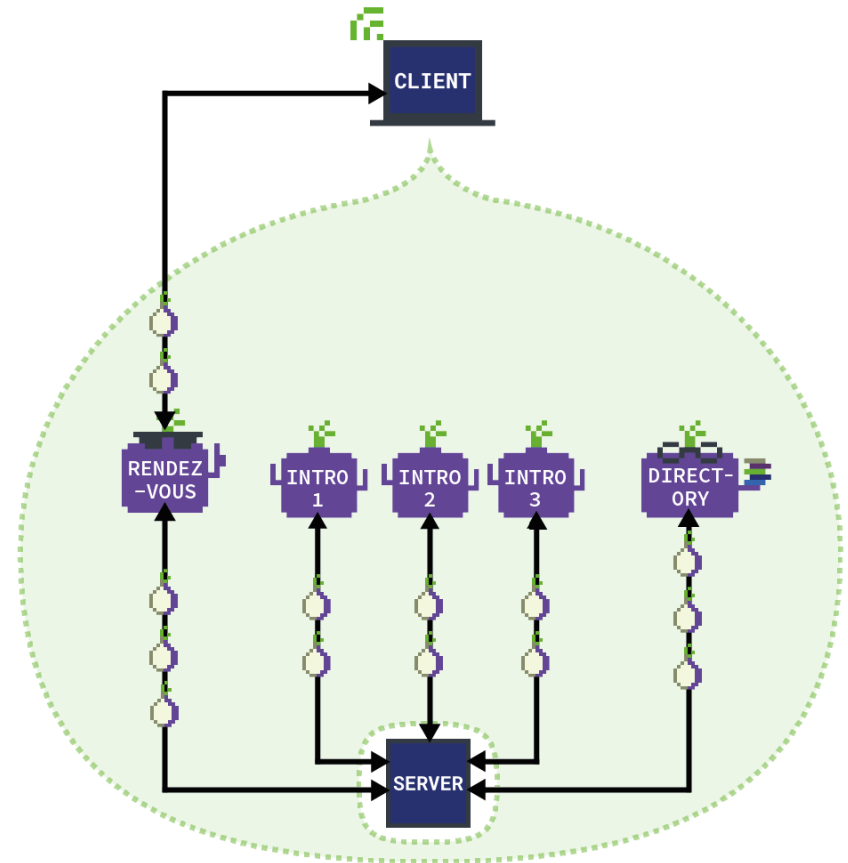
Anonymity Network

Bob



# Onion Services

- Onion Service traffic never leaves the Tor Network
  - Onion Service defines a set of introduction points within the Tor Network
  - Onion Services registers these introduction points in a distributed database in the Tor Network
  - Client connects to one of these introduction points, and negotiates a rendezvous point on another relay
  - Client + Onion Service each create circuits to the rendezvous point and begin talking



# How Does It Work? (cont)

- Every user has an id, an onion-service id:
  - 6l62fw7tqctlu5fesdqukvpoxezkaxbzllrafa2ve6ewuhzphxczszyd.onion
- This id serves dual purpose:
  - a destination (an Onion Service) for connecting peers
  - an identifier used for authenticating clients when connecting to other peers (Onion Services)
- Each peer hosts an Onion Service, which other peers may connect to

# End-to-End Encryption

- All communications between peers are end-to-end encrypted with forward secrecy

# Anonymous

- Peers do not need to know each other's real IP address to communicate
  - The *required* usage of Tor makes it impossible to accidentally leak the users IP address through usage of Gosling
- No central registrar, no sign-ups, no need to associate a peer with a phone number or other Id

# Hole Punching

- Peers do not need to have publicly accessible open ports for other peers to connect to them
- Peers only need to make outgoing connections

# Censorship-Circumvention

- There is no authority on who can or cannot run an Onion Service which could block a peer from receiving connections
- All peer-to-peer traffic stays within the Tor Network
- If you can connect to the Tor Network, then you have full access to other peers
  - (maybe a big 'if')

# Censorship-Circumvention (cont)

- Suppose you are in a place which blocks Tor such as:
  - China, Iran, Russia
  - Schools, Universities, Libraries
  - Offices, Government Buildings
- We can use pluggable transports to circumvent the block!
- Pluggable transports disguise your traffic as something else
- For example:
  - Snowflake[1] disguises your traffic as WebRTC

**1. Snowflake: <https://gitlab.torproject.org/tpo/anti-censorship/pluggable-transport/snowflake>**

# Wait A Second...

- So you may be thinking something like: "Ok, so you have a library which routes your traffic through the Tor Network, uses Onion Services, and inherits all their features. Good job, so what?"
- Bear with me!

# Client Authentication

- Thanks to clever cryptography (\*hand waving\*), Onion Services are self-authenticating
- But clients are not, you do not need any authentication to connect to an Onion Service
- Clients do not have Onion Service Ids
- **Problem:** This is supposed to be a peer-to-peer system! How does an Onion Service verify connecting clients are who they say they are?

# Client Authentication (cont)

- If a user connects to your service, and claim they are the owner of onion service id abcd...234.onion, what they are *really* claiming is they control the *private* key which maps to the *public* key which is encoded in their onion service id.
- To verify the client is telling the truth, we ask them to sign a carefully crafted message[1] with their *private* key, and the onion service verifies the signature using the client's provided *public* key (derived from their claimed onion service id)

# Optional Application-Specific Extensions

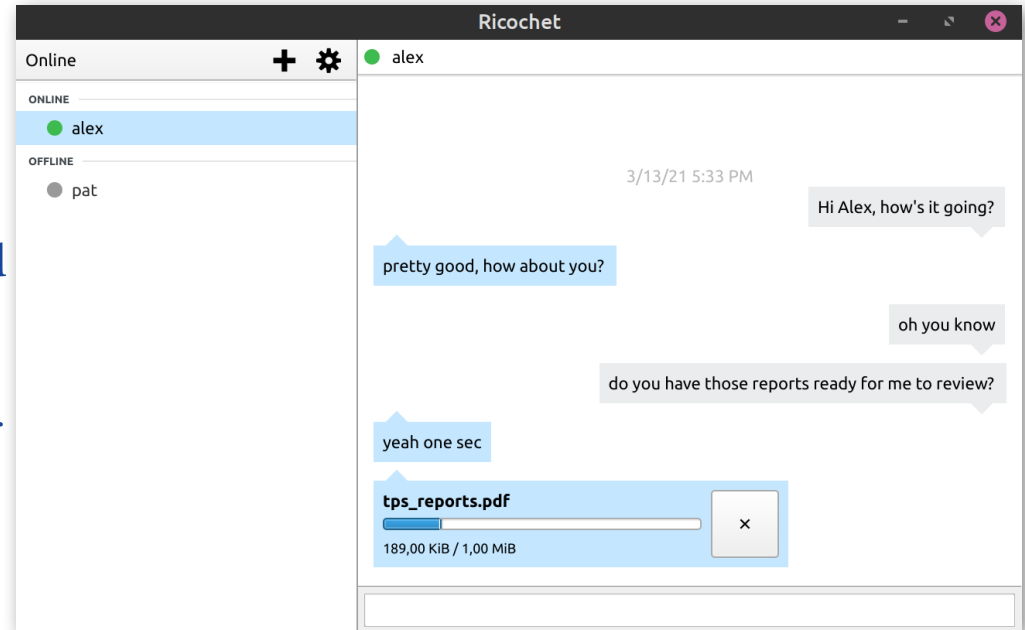
- Protocol has some flexibility to allow for some additional application-specific authentication barriers or requirements such as:
  - Peer block/allow lists
  - Shared secrets/invite codes
  - Proof-of-Work/Stake schemes

# Metadata-Resistance

- Communication contents are fully end-to-end encrypted, and stay entirely within the Tor Network
- Clients' real identities are unknown to each other
- No way to determine who peers have connected to; no way to generate a 'social graph' of peers
- Sounds great, so what's the problem?

# Some History: Ricochet-Refresh

- Peer-to-peer instant messenger via tor onion services
- Anonymous chat + file transfer
- Similar peer (contact/friend) authentication mechanism as described previously
- At least one of the peers must be running an Onion Service for the other peer to connect to in-order to chat
- A Security Review a few years ago alerted us to a problem...



# An Interesting Property of Onion Services

- Anyone (authenticated peer or not) can attempt to connect to your Onion Service and determine if it is currently online
- Therefore, a profile of the Onion Service's online/offline status can be built by repeatedly doing this
- Not really a big deal if your Onion Service is for a website or some other service that is meant to be always online
- *Kind* of a big deal when that Onion Service is running in a personal computing environment because PC online/offline status maps pretty closely to human user using/not using their computer
- Tor's Onion Service Client Authorisation feature does not fix this problem, only decreases the time resolution

# Whoops, Metadata Leak!

# Cyber-Stalking

- Malicious 3rd parties can easily 'cyber-stalk' users by simply trying to connect to them to identify their online/offline status
- Quite malicious 3rd parties could *also* discover your guard node (e.g. by simultaneously knocking guard nodes offline and cyber-stalking users)
- Quite malicious+capable 3rd parties can therefore de-anonymise users if they can see who a guard node is connected to with various methods (e.g. legal wiretaps, running malicious guard nodes and waiting, etc)

# So keep it secret...

- If you only share your Onion Service Id privately with your trusted contacts, then you do not need to worry about this metadata leak (presuming your contacts also do not leak to 3<sup>rd</sup> parties)
- Obviously a bit of a problem for general use if the act of simply sharing your Onion Service Id allows capable and malicious adversaries to de-anonymise you

# What We Would Like...

- Authenticated peers should be able to connect to and communicate with each other
- Unauthenticated peers should not be able to determine each others online/offline status
- Unauthenticated peers should be able to become Authenticated
- **Problem 1:** You can't do all three at once...
- **Problem 2:** Core functionality *requires* sharing your Onion Service Id with potentially untrustworthy users

# Gosling's Solution:

- Spread our Onion Service's responsibility across more Onion Services:
  - 1 'identity' service
  - N 'endpoint' services (one for each authenticated peer)
- Identity service acts as the gatekeeper for authenticating and accepting or rejecting new peers and distributing endpoint service credentials
- Endpoint services are where the actual peer-to-peer application-specific communications happen

**1. Gosling Specification:** <https://gosling.technology/gosling-spec.xhtml>

# Gosling's Solution (cont):

- This division of labor across multiple Onion Services means the Identity Service is not *required* for core functionality, meaning it can be disabled if there is no need to acquire more peers
  - e.g. in an instant messaging app, you can enable only your contact's associated Endpoint Services without losing normal functionality
    - potential new contacts won't be able to connect to you, but you can connect to them
- This prevents unauthenticated users from cyber-stalking you, even if you have shared your Onion Service Id publicly
- Ultimately about giving users control over their visibility

# Example Chat Applications (Rust and C++)

```
Welcome to example_chat_rs!  
Type help for a list of commands  
> help  
Available commands:  
  help COMMAND          Print help for COMMAND  
  init-context           Initialise the gosling context  
  start-identity         Start the identity onion-service  
  stop-identity          Stop the identity onion-service  
  request-endpoint       Connect to identity onion-service and request an endpoint  
  start-endpoint         Start an endpoint onion-service  
  stop-endpoint          Stop an endpoint onion-service  
  connect-endpoint       Connect to a peer's endpoint onion-service  
  drop-peer             Drop a connection to a peer  
  list-peers             List all of the currently connected peers  
  chat                  Send a message to a connected peer  
  exit                  Quits the program  
=====
```

> init-context|

Examples: <https://github.com/blueprint-freespeech/gosling/tree/main/source/examples>

# Current + Future Work

- Next Ricochet-Refresh v4 will use Gosling for client-authentication and have feature parity (and limited backward compatibility) with Ricochet-Refresh v3 as well as a UI overhaul
  - Further Gosling iteration+improvements as we run into software engineering realities
- Ricochet-Refresh v5 will focus on new user-facing features

# Links

## Gosling

- Website: <https://gosling.technology/>
- GitHub: <https://github.com/blueprint-freespeech/gosling>

## Ricochet-Refresh

- Website: <https://ricochetrefresh.net>
- GitHub: <https://github.com/blueprint-freespeech/ricochet-refresh>

Blueprint For Free Speech: <https://blueprintforfreespeech.net>