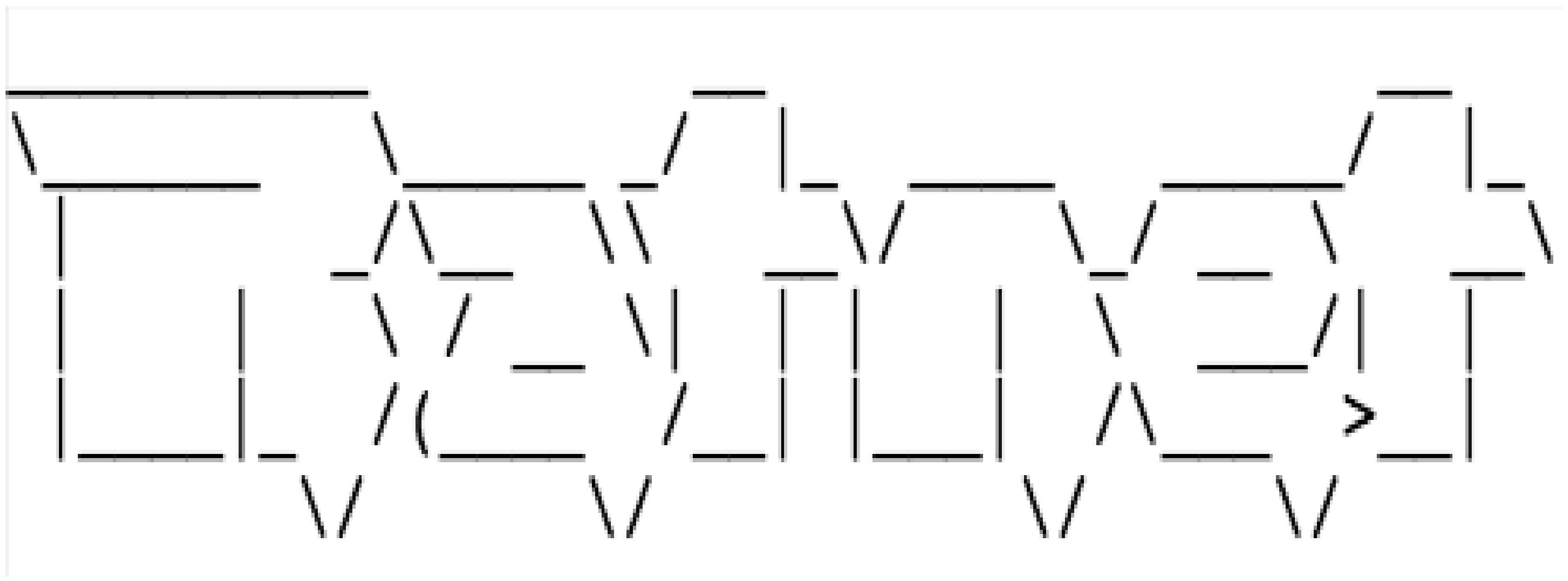


Special Thanks to: Vyrus

Intro

- RatNet is a protocol for simple one-way message passing which is: Onion-routed, Flood-routed, End-to-end encrypted, Designed to run on any hardware & over any transport.
  - Reference implementation in Go
- First, **Disclaimer**
- Then <u>Why</u>, Then <u>What</u>, Then <u>Who</u>.

Disclaimer

- The code is a proof of concept and should not be relied on for any purpose until it’s been subjected to some external scrutiny.
- I am here primarily for **peer review**.

Use Case: Iranian Election Problem

- State blocks all TLS in spurts.
- Targeted blocking of Tor, FB, Twitter, Gmail, proxies, etc.
- Deanonymization => v& (ask an Iranian)

Use Case: Protester Coordination Problems

- Cell towers go down (for some strange reason).
- Cops are on to the “organize everything on FB” plan.
- Need a quick way to enroll people.
- Trust nightmare.

Why not Tor?

- Tor is the best existing thing that anonymizes bidirectional sockets (which is impossible).
- Obviously, Tor has a lot of trouble living up to its security promises (hidden services, exit node control, no e2e encryption, browser bugs, blah blah blah).
- Tor is a best-effort solution to an impossible problem, no complaints about that…
- I have some different beef with Tor.

My Beef with Tor

- Tor was written to be difficult to embed and difficult to cross-compile. On… fscking... **purpose**.
- When asked, a dev told me: “**We don’t want people to use it for botnets**.”
- So I asked about embedded, and he said: “**We don’t want to support platforms that will only be clients.**”

Tor: Intended Consequences

- The thing is such a mess there has been only one partial port (to Java), and full version still won’t build on an ARM Chromebook. And the bugs, oh the bugs...
- Crippling functionality out of fear “the people” will misuse it is traditional gov’t treatment of crypto.
  - DES - Parity bits are super important, eh?
  - SSL – 40 bits is OK for export?

If **your** life is depending on something,
you should be able to take it apart
and put it back together.

Crippling functionality to impose an arbitrary morality on end users is just **bad design**.

Use Case: Criminal Problems

- Adversary is the state, purpose of communication is technically illegal, although possibly legitimate.
- Absolutely anything goes on the network in terms of detection, protocol blocking, and total outages.
- Deniability and hiding the intended destination of a message has value.
- Detection will improve to track anomalous behavior, we must have room to grow.
- Potentially life-and-death (why I’m here).

Why not FreeNet?

- …
- …
- …
- …
- …
- ...
- You *know* why, shut up.

Design Goals

- **Sender and Recipient should be as hidden as possible on the wire**
- Non-Internet Unidirectional Transmission (Packet Radio)
- Mesh-routable, for when there is no Internet
- Varied Internet Transports, for when Internet is hostile
- End-to-end encryption, because users

Design Goals (cont.)

- Modular everything, any layer can be replaced
- Easy to embed in an application as a transport layer
- Easy to understand (well, compared to Tor)
- Private nets, per-event nets, and botnets are all first-class citizens
- Must compile for embedded and especially ARM

Non-Goals

- Efficiency at all costs – if it was more efficient, someone would already be doing it.
- Supporting a “socket” abstraction. This makes correlation attacks too easy, plus doesn’t work well unidirectional with high latencies.
- Forcing everyone on to a global network or into using interoperable crypto or transport.
  - Go make your own net. With blackjack. And hookers. (you can use this library to do it more easily, of course)

Non-Goals (cont.)

- Preventing “unacceptable” use through code or protocol obfuscation.
  - I do pentest, I have a legal & legitimate use for a RAT.
  - Dissidents are criminals some places, it’s just that the relevant laws may be illegitimate.

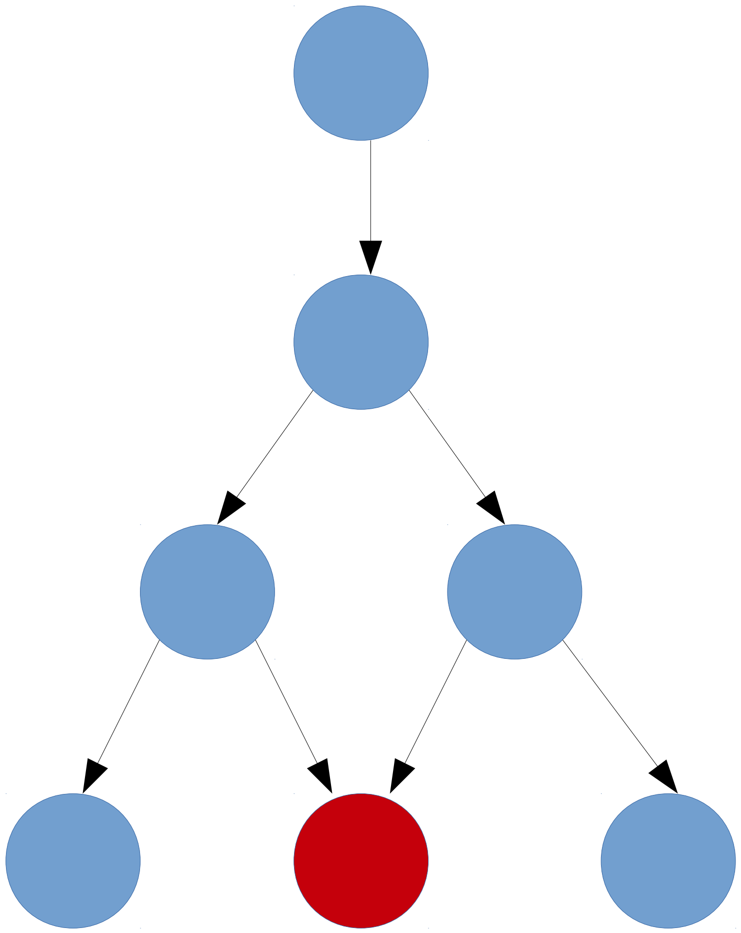
Room to Grow

- Virtual File Systems (AES-CTR via golang?)
- Key management should be portable to TPM, JavaCard, or other KMS-type thing.
- “Meta” routing protocol that can control switching between different types of physical transport.
  - … more on this later
- The more physical transports implemented, the more versatile.
- Network configuration management is going to be a huge problem for scaling, but there is a good framework to build on.

Turbo Background & Influences

- Flood Routing
- FidoNet
- UUCP
- Bang Paths
- UseNet
- Onion Routing

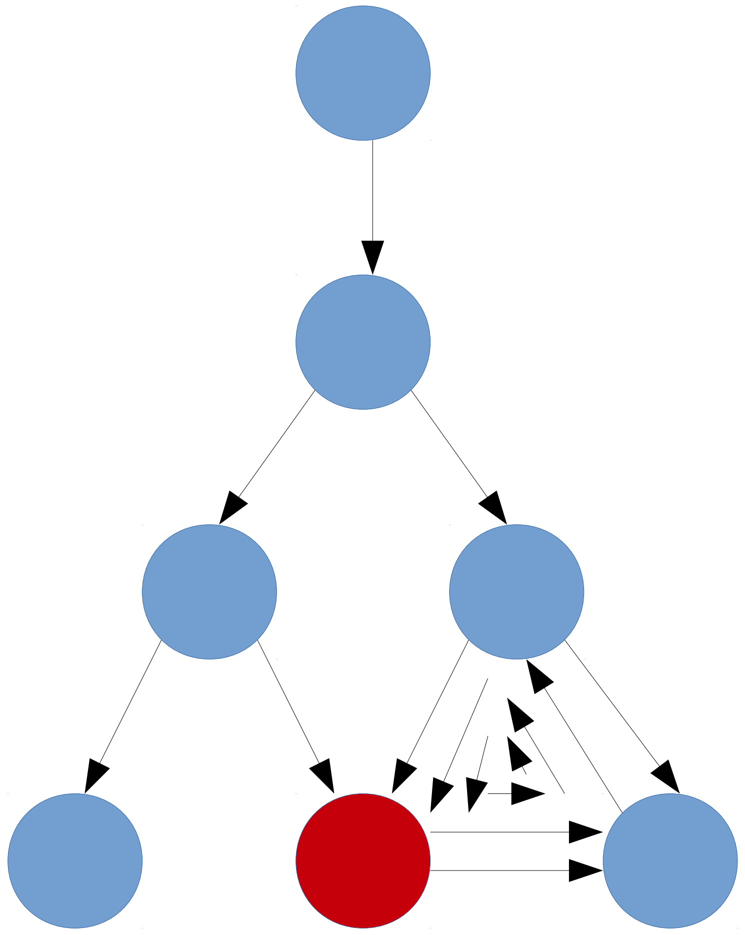
Flood Routing (* goes *)



Flood Routing / Store-and-Forward

- Pros:
  - Only real solution for mobile mesh routing:
    - Potentially long-to-infinite delays between contacts
  - Great for creating ‘whitenet’ noise.
- Cons:
  - The most inefficient strategy possible.
  - No guarantee a message will be delivered at all (this can be handled up the stack).
  - Loops are an issue.

Loop Detection



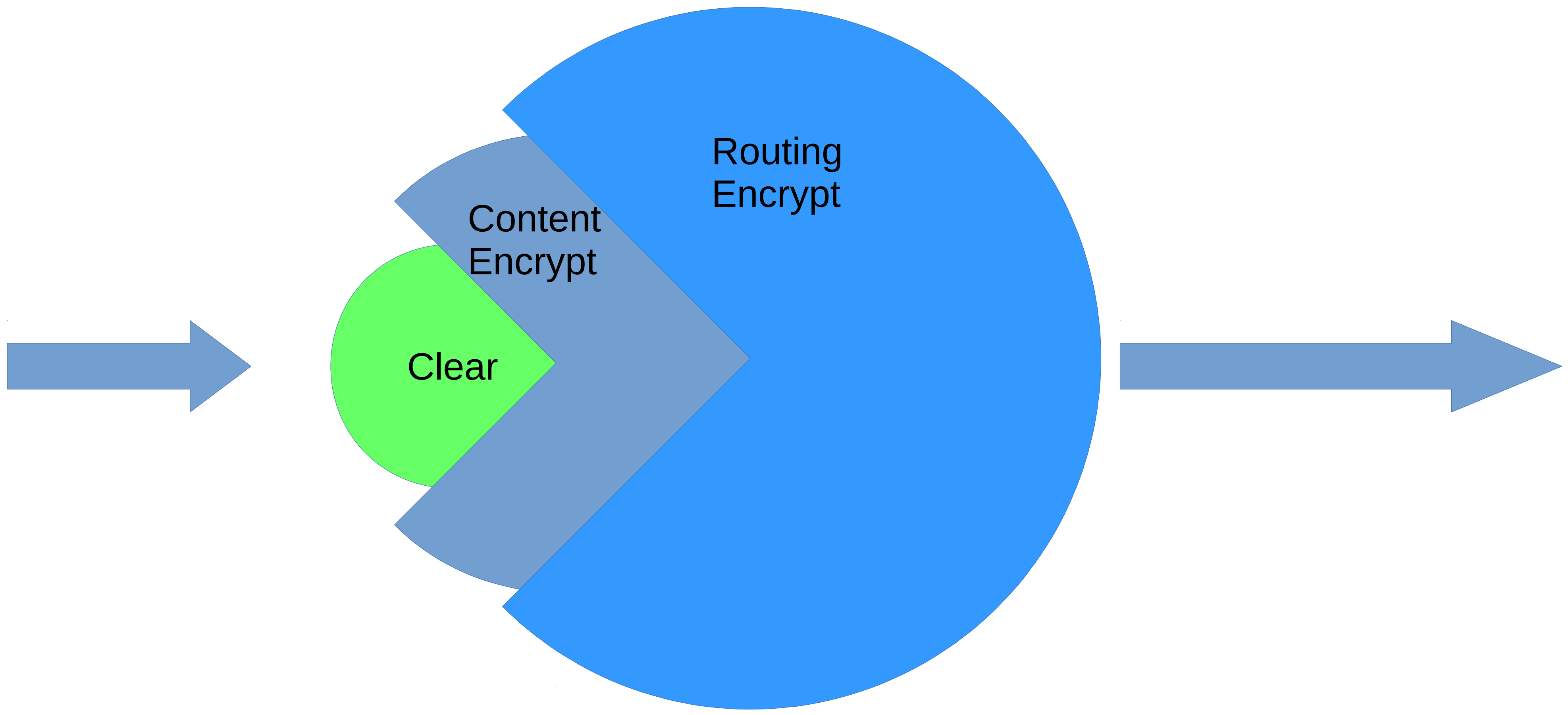
FidoNet, UUCP, & UseNet

- FidoNet
  - For modems to relay BBS msgs to each other. No crypto
  - Because long distance calls used to be a thing
    - Zone:Network/Node – all nodes knew all other nodes addresses
    - Zone mail hour
    - FidoNet user addresses: bob @ 2:331:113.1
- Unix-to-Unix copy (UUCP)
  - Bang Paths for source routing: node1!node1!node3!node4
  - This might be interesting to implement Tor in RatNet or ?

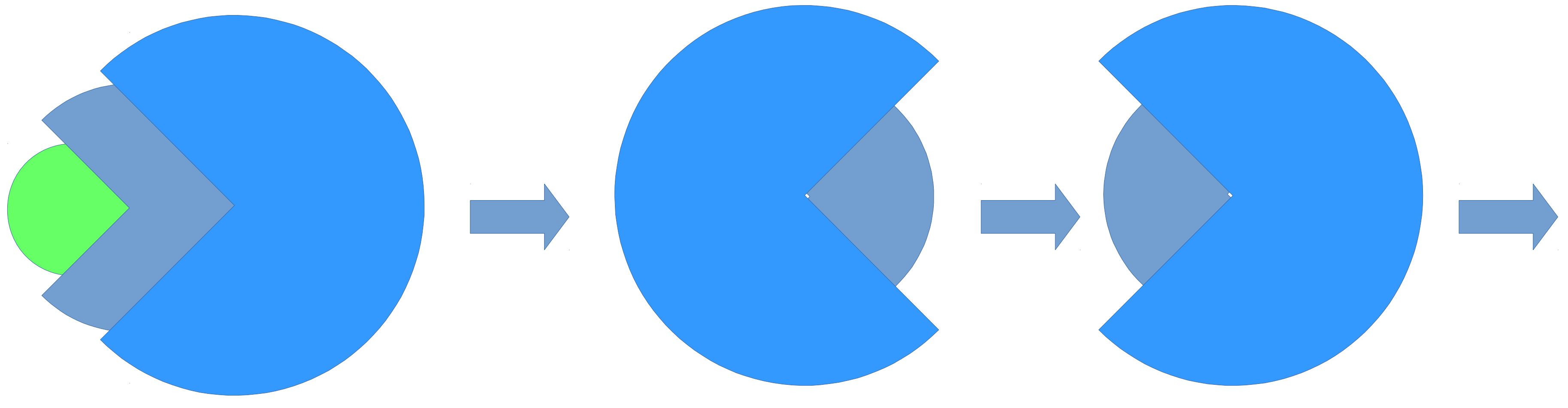
UseNet

- Everything goes everywhere, but nodes can filter in/out channels they can handle based on hierarchy:
  - alt.startrek.fan-fiction.parodies.time-travel.twentieth-century.newt-gingrich.changelings.infiltration
  - alt.binaries.erotica.genitalia.Presidential
  - alt.broke.unemployed.overeducated.misunderstood

Onion-Routing



Onion-Routing: One Step



Notes on Threat Model

- Combination of Onion and Flood routing is good for privacy, but bad for efficiency.
- Correlation attacks can always be screwed up with dynamic topography and varied latency connections.
- However, this system is an engine for Denial of Servicing itself if not careful.
- Nodes must be extra vigilant about managing their own resources and saying no to peer pressure.
- Also stolen/lost phone scenarios are critical.

What did we do?

- RatNet is an onion-routed messaging layer
  - Flood-routed, store-and-forward
  - Completely modular network transports & crypto
  - Builds to ARM (and everything else)
  - Easily embeddable
  - Supports open and private scenarios
- HushCom is an IRC-like chat client/server using Ratnet

RatNet Component

- Acts as a Key Management Service
  - Ask it to generate keys, store them, and sign/encrypt things with them, but it won’t return the private key.
- Provides an API which can be accessed locally via Go or FFI native code, or accessed remotely through any transport plugin.
- Ratnet is responsible for key management and the message queue.

RatNet Component (cont.)

- Supports simple channels ([a-zA-Z0-9]+), but no source routing or hierarchies or anything like that yet.
- Ratnet makes calls to CryptoAPI module, but the transport modules call in to RatNet…

RatNet API Flows

- Send / Send Channel:
  - DestHash (16 byte random nonce
    + KDF(destPubKey)[16])
  - **Encrypts** single msg to Destination Content Key (or Channel key), and adds to outbound queue
- Pickup
  - Bundles msgs since given time, from all or certain channels
  - **Encrypts** to Routing Key
  - Prepends local current time

RatNet API Flows (cont.)

- Deliver
  - Receives bundle from Pickup and handles/fwd’s
  - Drops repeating nonces via capped Hash Table
  - DestHashes each with <u>all known channel/private PubKeys</u>

Crypto (Batches & Singles)

Roughly ECIES (RSA implementation is similar, but sloooooow):

1) Generate new EC key. (Cheap!)

2) Determine shared key with public of destination.

3) Uses KDF to derive a symmetric encryption and a MAC key

4) Encrypts the message

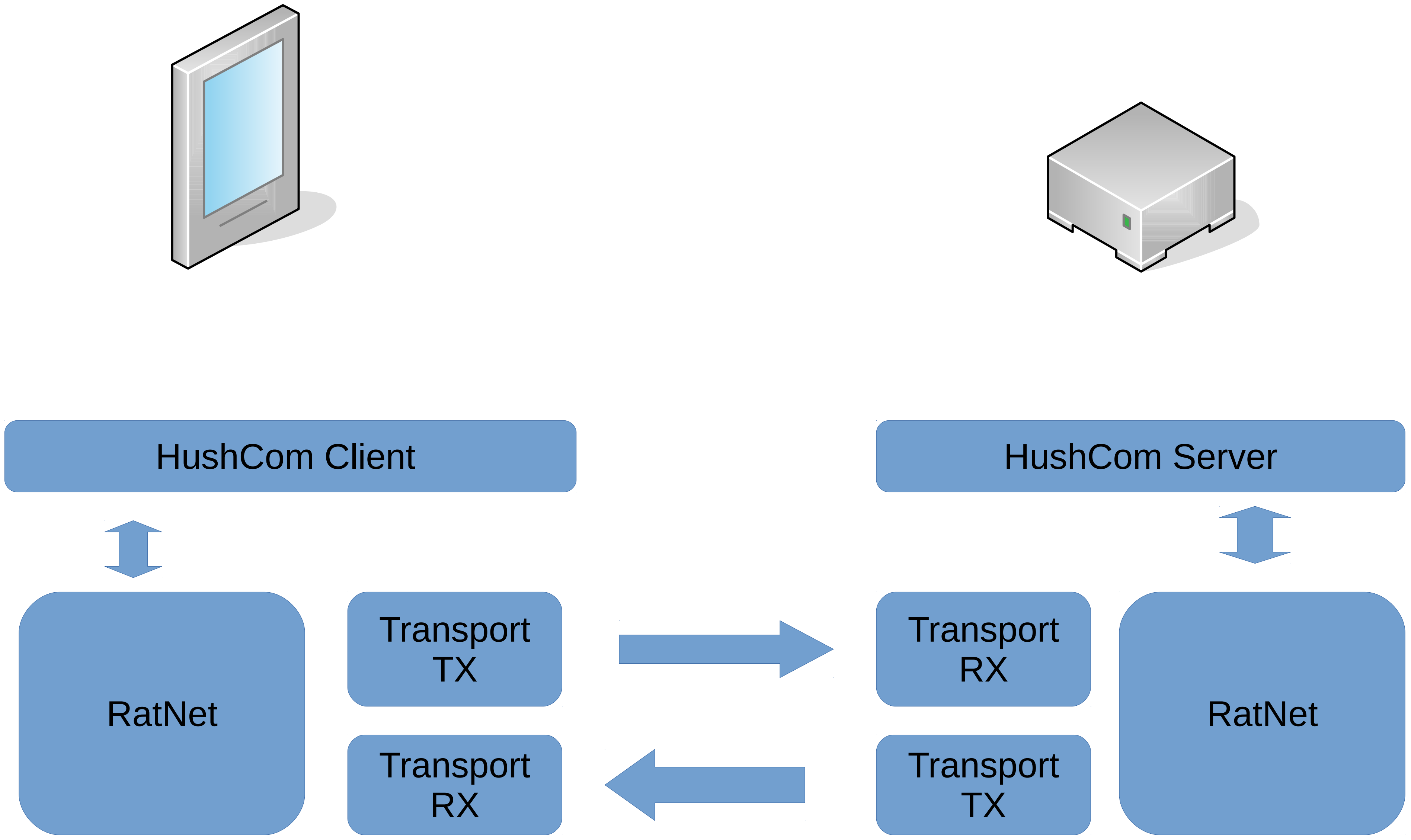
5) Computes the HMAC digest of encrypted message

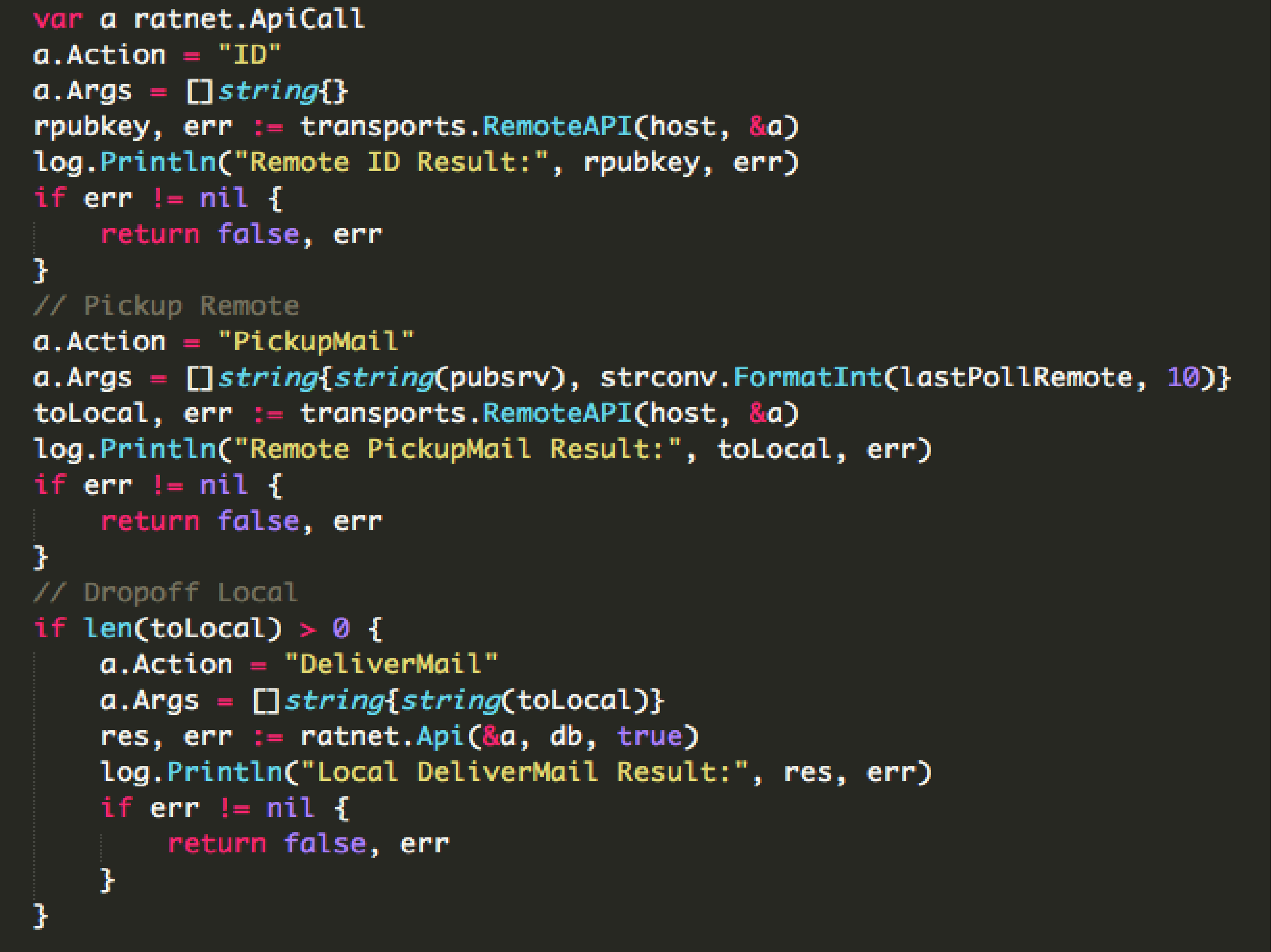
6) Outputs: SharedKey | CipherText | Digest

→ "Curve25519,AES-CBC-256,HMAC-SHA-256", etc.

HushCom

- Simple IRC-like that attempts to minimize the amount of information left on the server.
- Server stores nothing unencrypted to disk, and only caches in RAM:
  - List of registered nicks/pubkeys (NickServ)
  - List of registered channels/pubkeys (ChanServ)
- All other functionality is implemented P2P, server cannot even read contents of channel messages. At all.


HushCom Client
HushCom Server
RatNet
Transport TX
Transport RX
Transport RX
Transport TX
RatNet



```go
var a ratnet.ApiCall
a.Action = "ID"
a.Args = []string{}
rpubkey, err := transports.RemoteAPI(host, &a)
log.Println("Remote ID Result:", rpubkey, err)
if err != nil {
    return false, err
}
// Pickup Remote
a.Action = "PickupMail"
a.Args = []string{string(pubsrv), strconv.FormatInt(lastPollRemote, 10)}
toLocal, err := transports.RemoteAPI(host, &a)
log.Println("Remote PickupMail Result:", toLocal, err)
if err != nil {
    return false, err
}
// Dropoff Local
if len(toLocal) > 0 {
    a.Action = "DeliverMail"
    a.Args = []string{string(toLocal)}
    res, err := ratnet.Api(&a, db, true)
    log.Println("Local DeliverMail Result:", res, err)
    if err != nil {
        return false, err
    }
}
```

Hushcom Flows: New User

- Hello Server, here is a nick and pubkey signed with itself (inside the DestHash/encrypted usual RatNet message).
- Ok, Client, that name is untaken, I will remember you until I restart.

Hushcom Flows: New Channel

- What up Server, you remember me. I want to register a public channel with:
  - Name, Public Key
  - And I signed the request with the key you know from nick registry
- Client, OK, I will share the list of public channels and pubkeys with any registered nick until I restart.

Hushcom Flows: Channel Join

- User requests public channel list from server (or already has it), gets name & pubkey.
- For private channels, have to get the channel pubkey via QR code or some other way.
- User sends a Join request as a channel message, server forwards it to everywhere.
- If Password matches (or set to auto), user in channel will respond with the private key to the channel.
- User gets response w./key, adds key to keyring.

Future Tech

- Create a DSL for pushing dynamic routes out.
  - Dynamically reconfigure whole nets based on command, timing, or randomly.
  - Behavior-based IDS will work eventually, but this will fsck it for all time.
- More transports:
  - Packet Radio – Iranian Election Tweets
  - QUIC, UDT, ICMP, DTLS, Sneakernet, etc.
  - Wifi Direct (prototype exists), maybe BT

Future Tech II

- Two obvious moves to protect against being ‘owned from below’ or a lost phone scenario:
  - Sink the key management in RatNet into a TPM or SE or TrustZone TEE, etc.
  - Golang supports virtual filesystems. Can extend the ZipFS example to use AES-CTR mode or similar.
  - If both of these are done, the AES key can be wrapped to the TPM. Add in periodic key rolling, and this can become **super nasty** to recover anything from.

Future Tech III

- Network Simulations for routing strategy optimization.
- Investigate safety of source routing / bang path schemes.
- QR Code key exchange in HushCom client.
- Optimize~~, Optimize, Optimize~~
- This could run on a con badge or be a ‘virtual badge’… if you hadn’t worked that out yet...

O’Reilly Book Cover

“He’s not a Hairless,” the vet said. “He’s not even a dog. He’s a sewer rat—and he has rabies.”

Who?

- Hopefully, <u>you</u>.
- Looking for people to help with code/crypto review, simulations, and feature work.
- Also looking for people who want to use this as a library for their own projects!!!

Contact:

awgh@awgh.org

#ratnet on hackint

https://github.com/awgh/ratnet