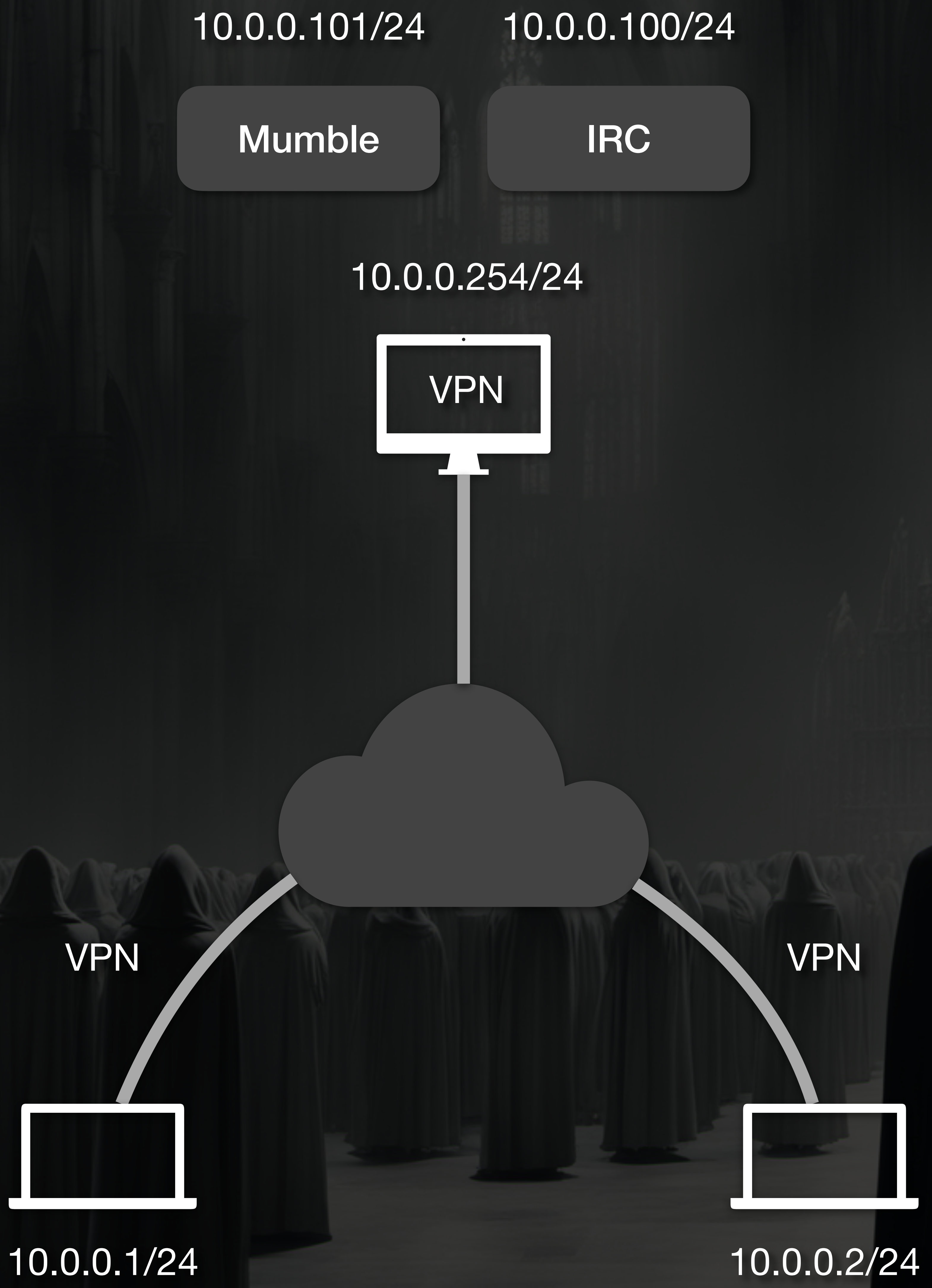
The Voices Of Confession

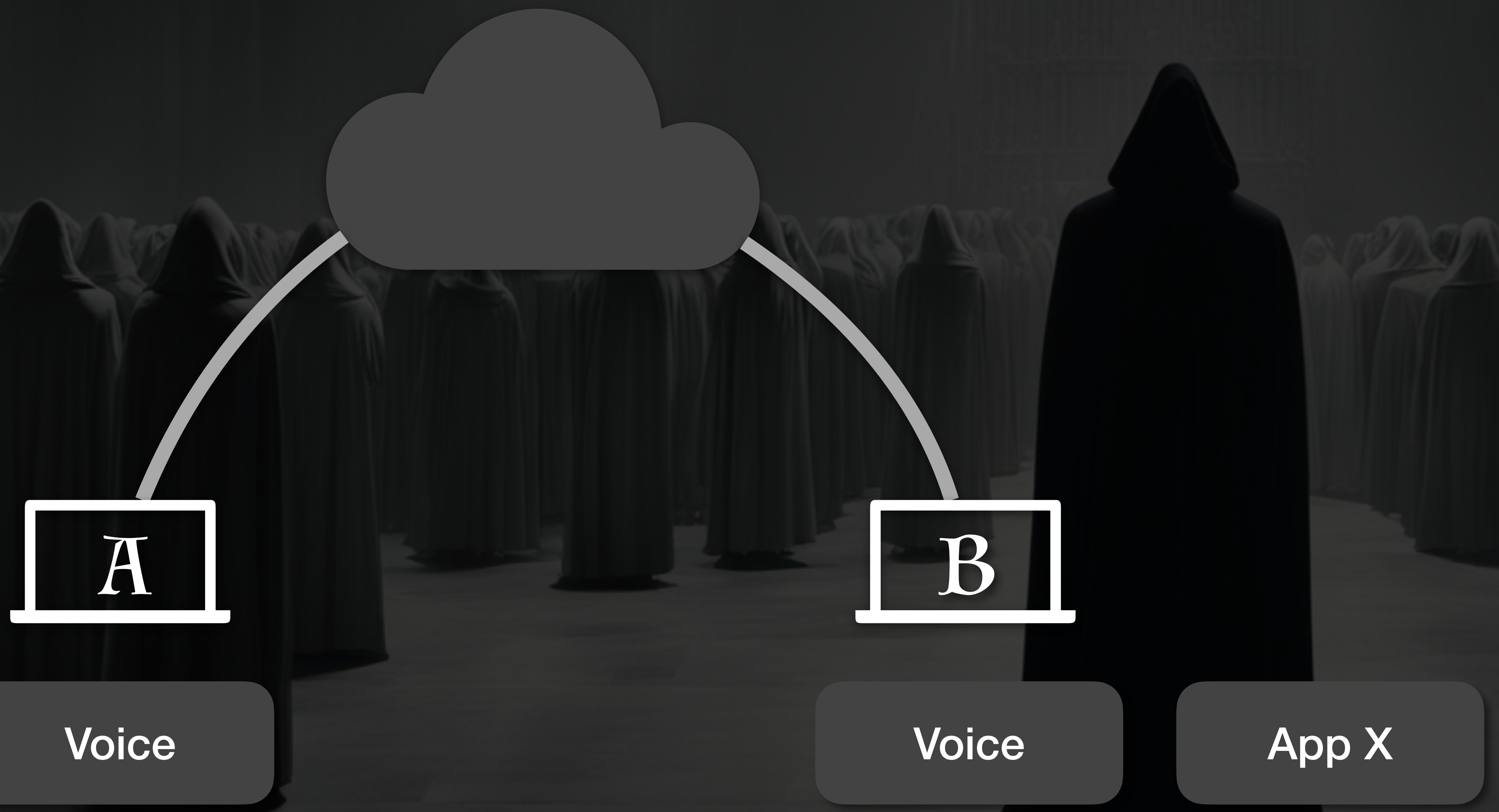
Gospel

- I stand before you as an independent hacker.
- I care deeply about secure communication.
- Distributed secure communication infrastructure.
- Been working on this project for about 2 years in my spare time.

Gospel


10.0.0.101/24
10.0.0.100/24
Mumble
IRC
10.0.0.254/24
VPN
VPN
VPN
10.0.0.1/24
10.0.0.2/24

Gospel



Gospel

- Commercial offerings exist, for example Tailscale
- IP only.
- WireGuard (TM) as their DataPlane.
- ControlPlane are closed platforms.
- Open source implementations exist but companies can decide to kill these at any time.
- NordVPN recently killed off their mesh-net without warning.

How can we build better?

How can we build more resilient?

Principium

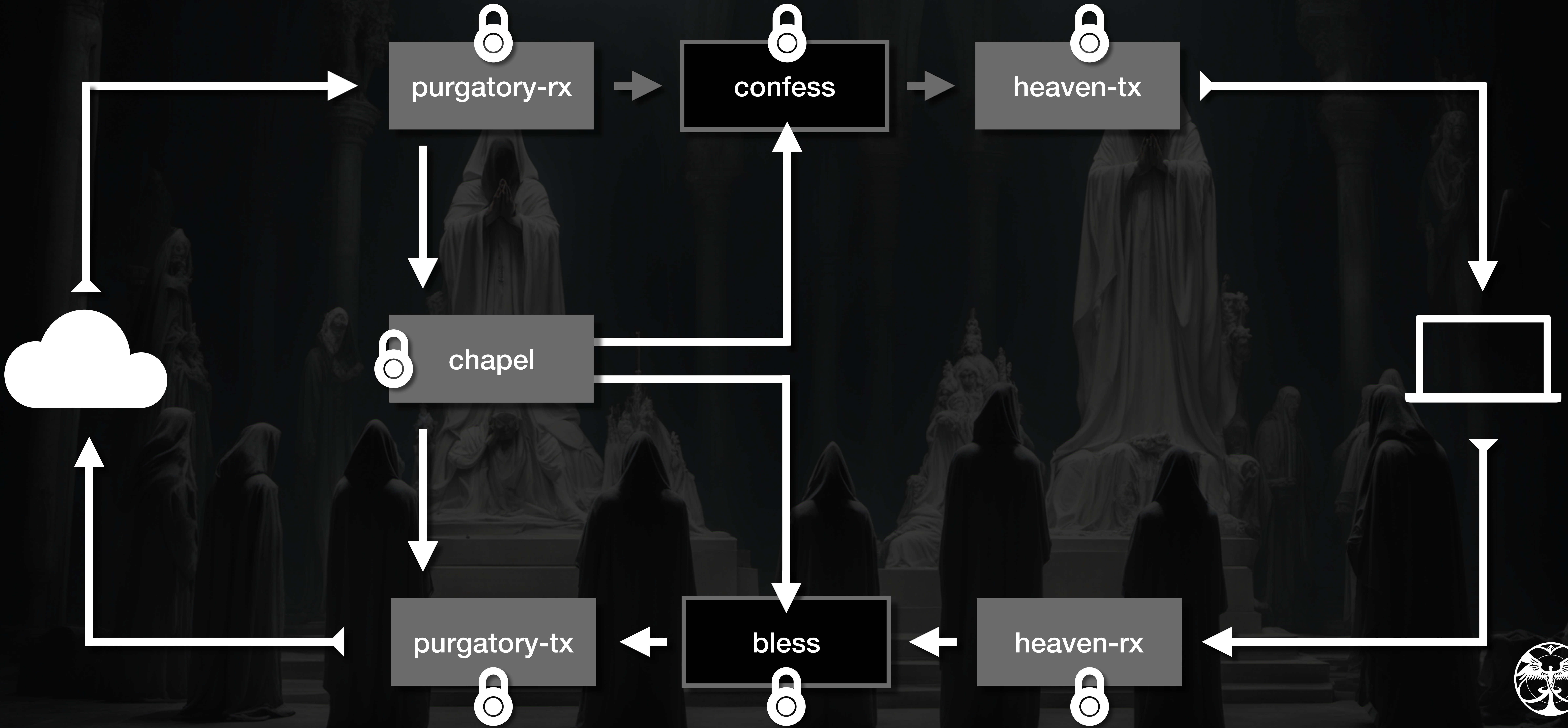
- The Sanctum
- The Cathedrals
- The Library
- The Conclave
  - Confession
  - Litany
- The Reliquary

Sanctum

Sanctum

- Sanctum, a VPN daemon with many novel approaches.
  - ISC licensed, fully free and open.
  - Fully privilege separated at every level.
  - Sandboxed with modern techniques.
  - Implements an easy to understand protocol.

Sanctum



purgatory-rx
confess
heaven-tx
chapel
purgatory-tx
bless
heaven-rx

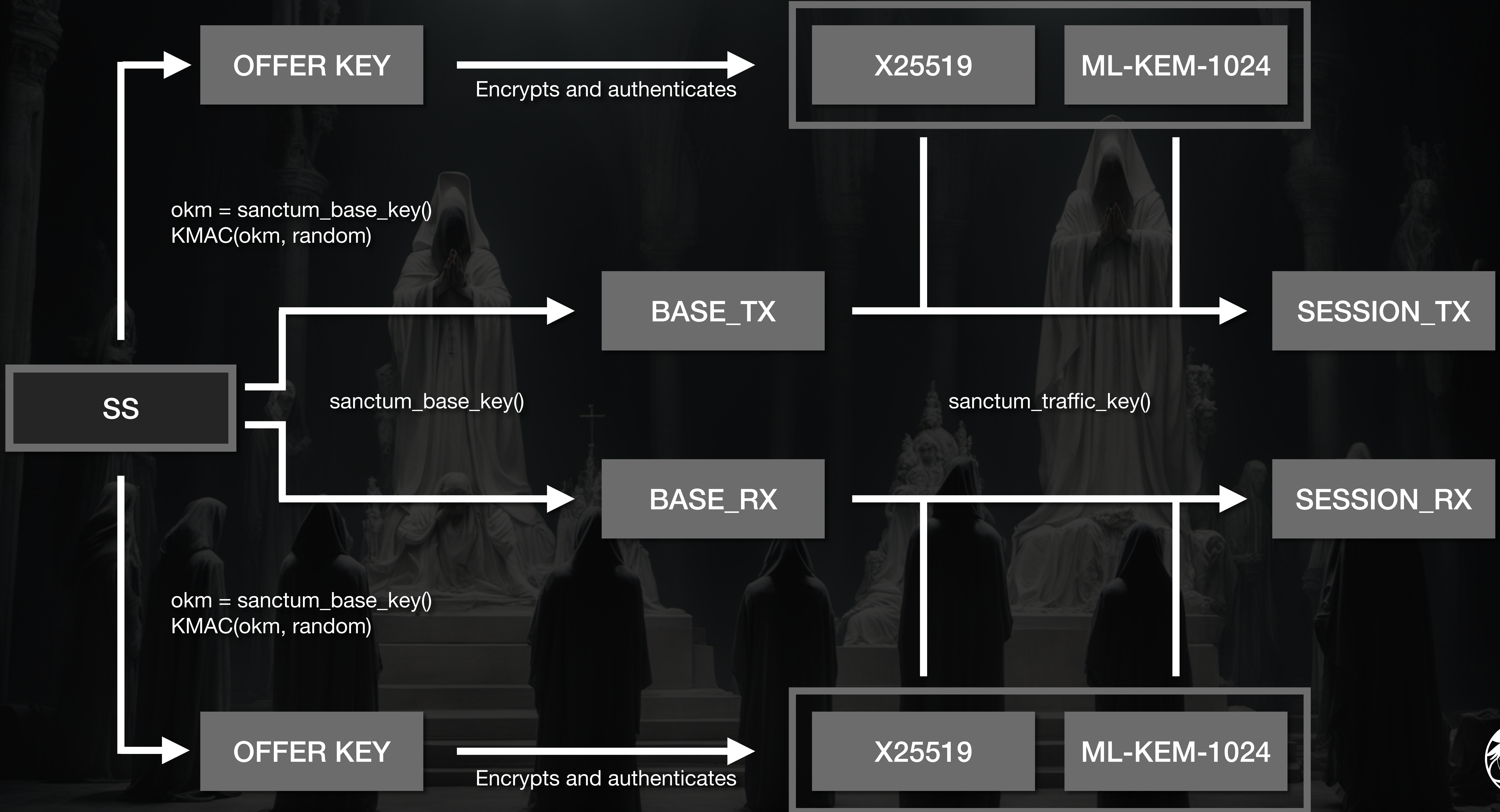
Sanctum

- Different modes
  - Tunnel mode (direct connection)
  - One-way mode (in case you have a diode)
  - Liturgy mode (auto discover peers + start tunnel mode)
  - Cathedral mode (what we’re going to talk about)

Sanctum

- Hybridised key exchange
  - Symmetrical key
  - Classical ECDH - x25519
  - PQ-secure - ML-KEM-1024
- 3 secret inputs to KDF for session key derivation.

Sanctum


OFFER KEY
Encrypts and authenticates
X25519
ML-KEM-1024
okm = sanctum_base_key()
KMAC(okm, random)
BASE_TX
SESSION_TX
SS
sanctum_base_key()
sanctum_traffic_key()
BASE_RX
SESSION_RX
okm = sanctum_base_key()
KMAC(okm, random)
OFFER KEY
Encrypts and authenticates
X25519
ML-KEM-1024

Cathedrals

Cathedrals
Design tenets

- Not security critical from a confidentiality point of view.
- Cathedrals cannot inject, decrypt or manipulate traffic.
- Cathedrals may be run on cloud platforms.
- Cathedrals are ephemeral.

Cathedrals

Discovery

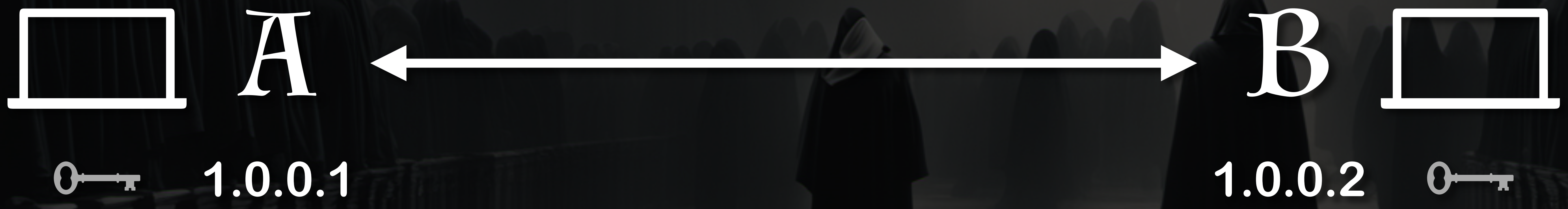
Key Distribution

Resilience

Cathedrals

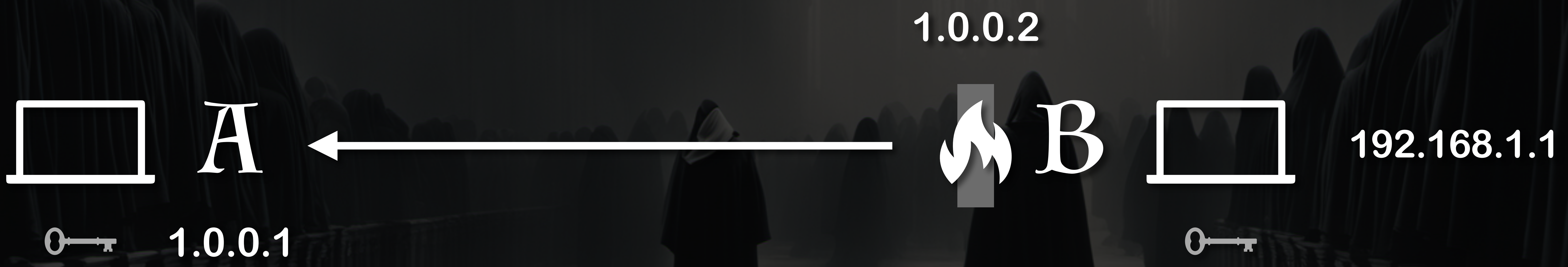
Discovery

Cathedrals
Discovery



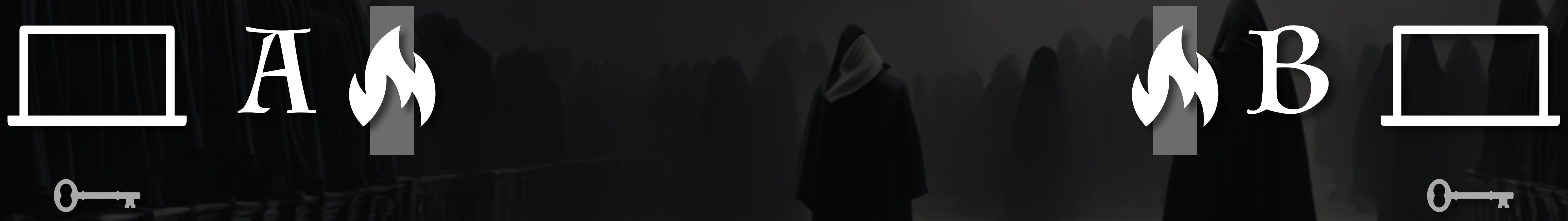
Cathedrals

Discovery



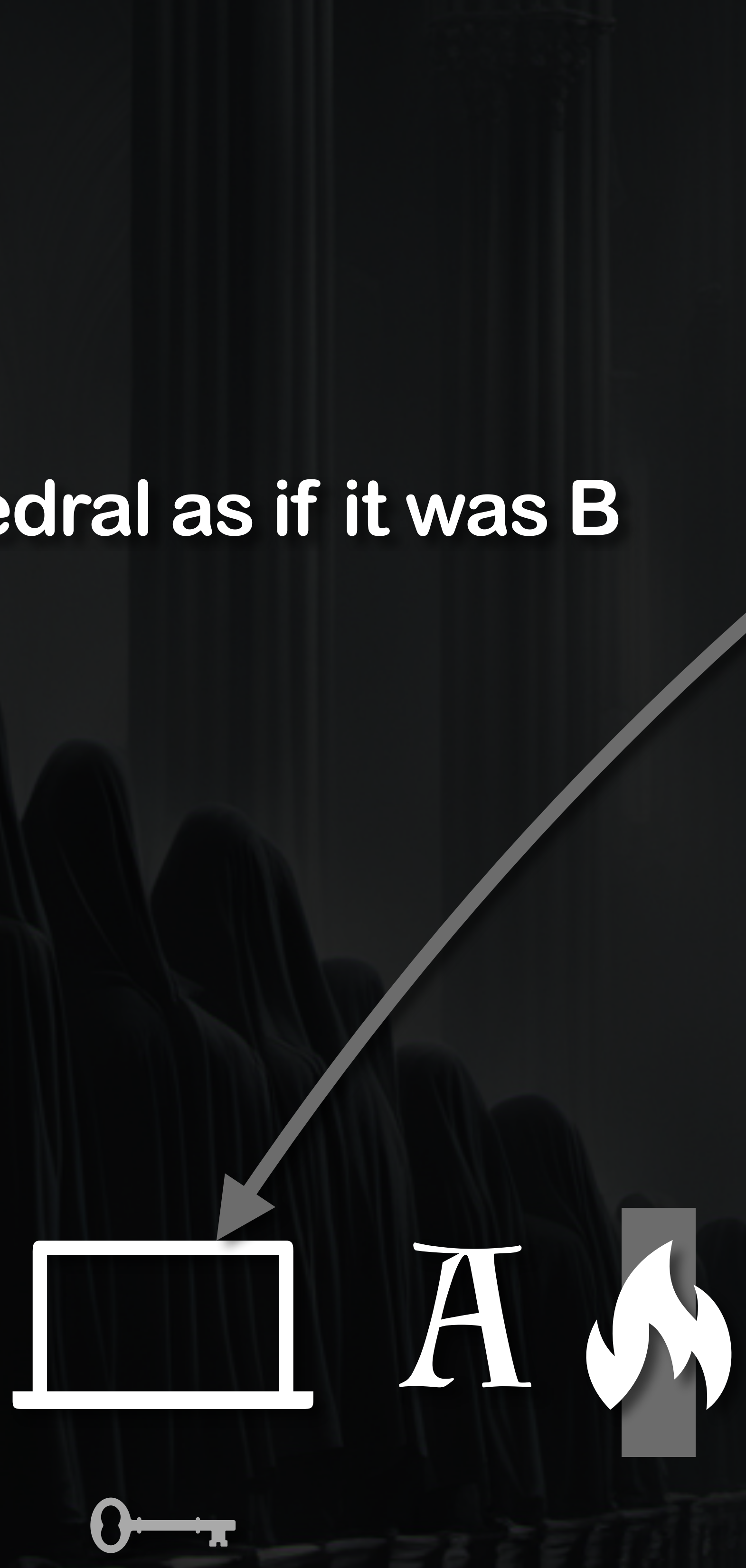
Cathedrals
Discovery

How does A communicate with B if both are behind a firewall?

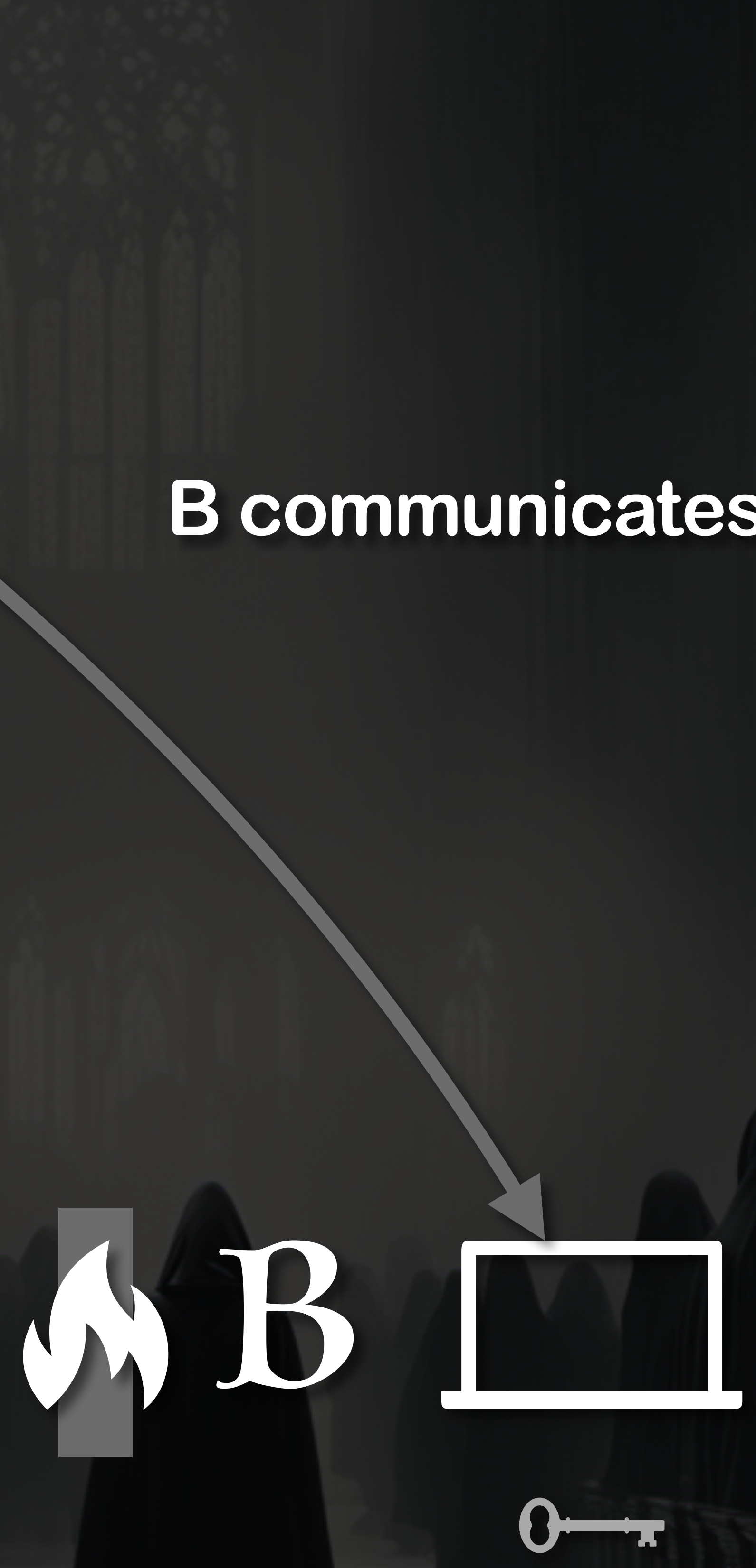


Cathedrals

Discovery



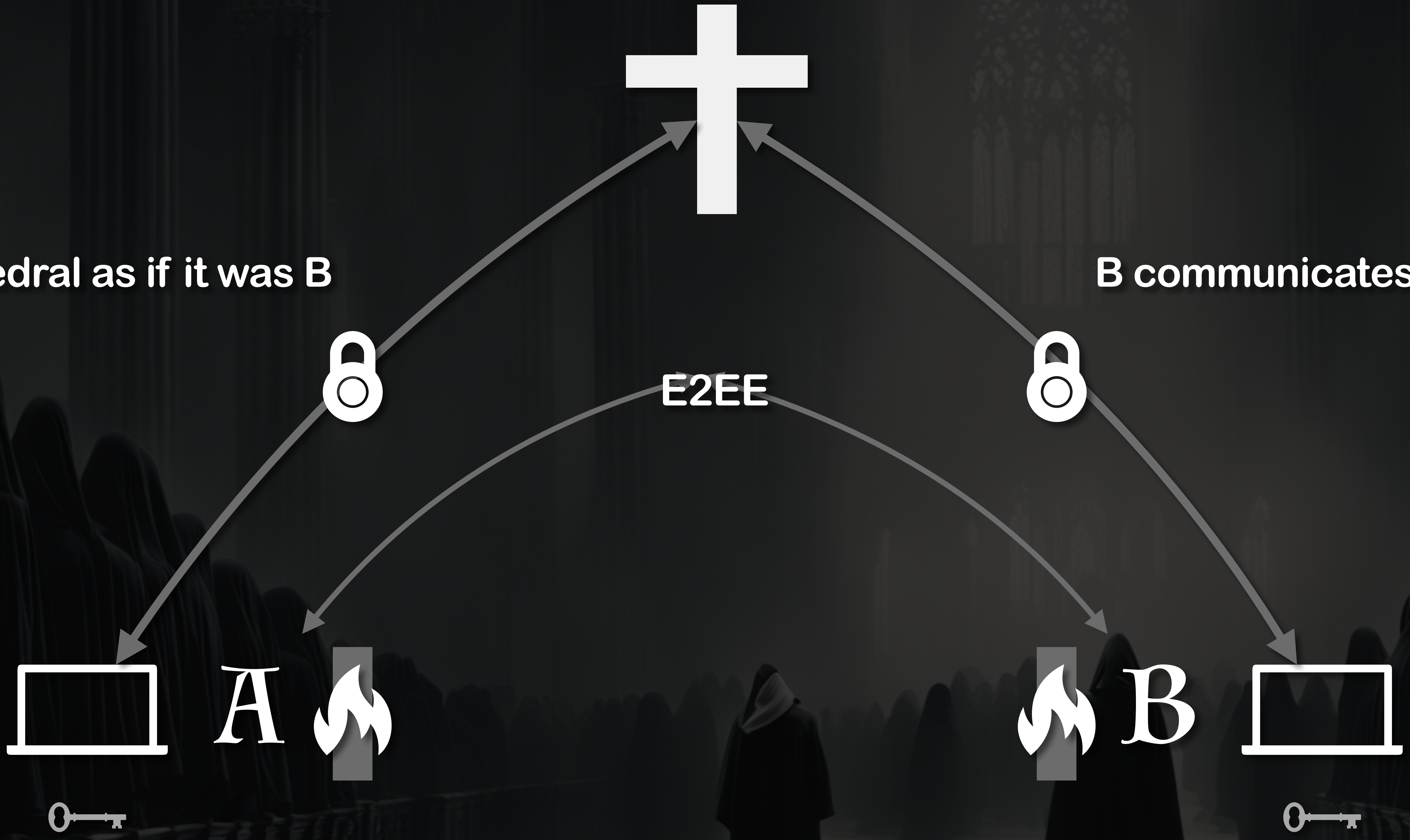
A communicates with cathedral as if it was B



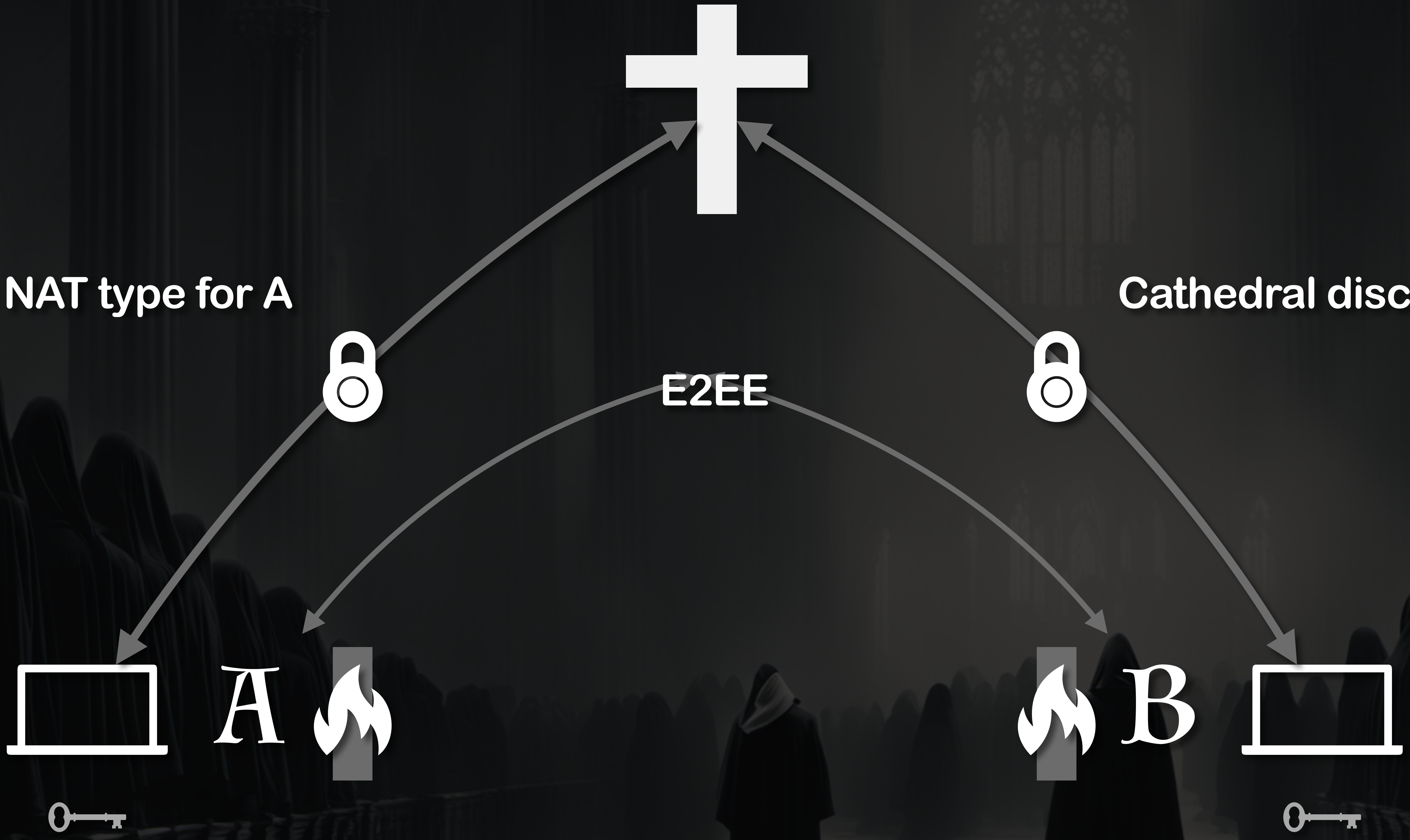
B communicates with cathedral as if it was A

Cathedrals

Discovery

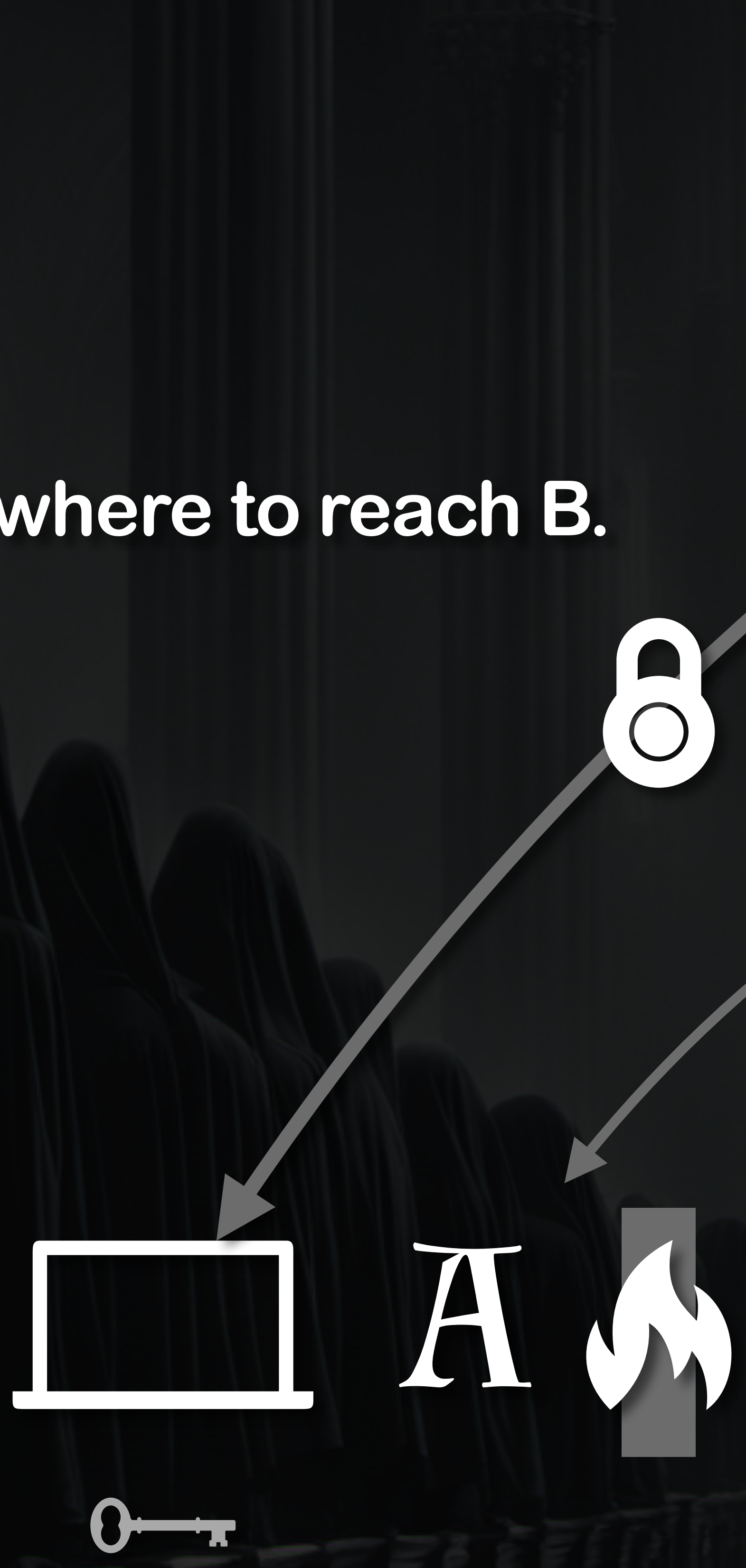

A communicates with cathedral as if it was B
B communicates with cathedral as if it was A
E2EE
A
B

Cathedrals
Discovery


Cathedral discovers NAT type for A
Cathedral discovers NAT type for B
E2EE
A
B

Cathedrals

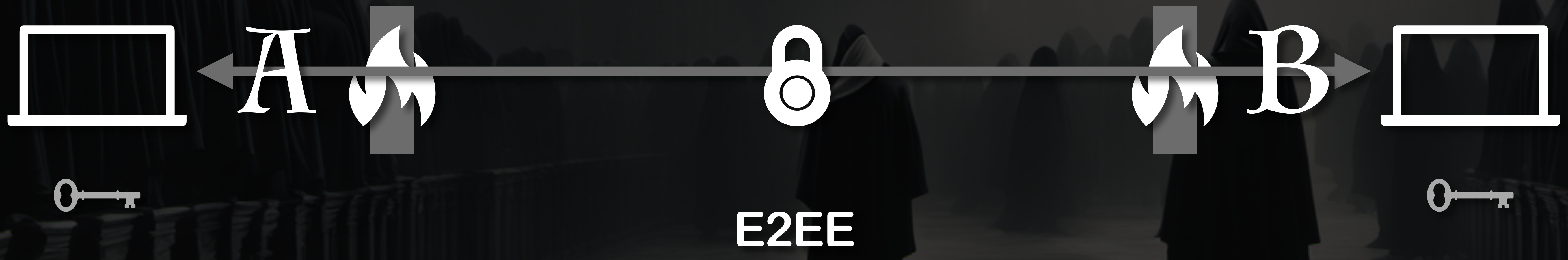
Discovery


Cathedral tells A where to reach B.
Cathedral tells B where to reach A.
E2EE
A
B

Cathedrals
Discovery

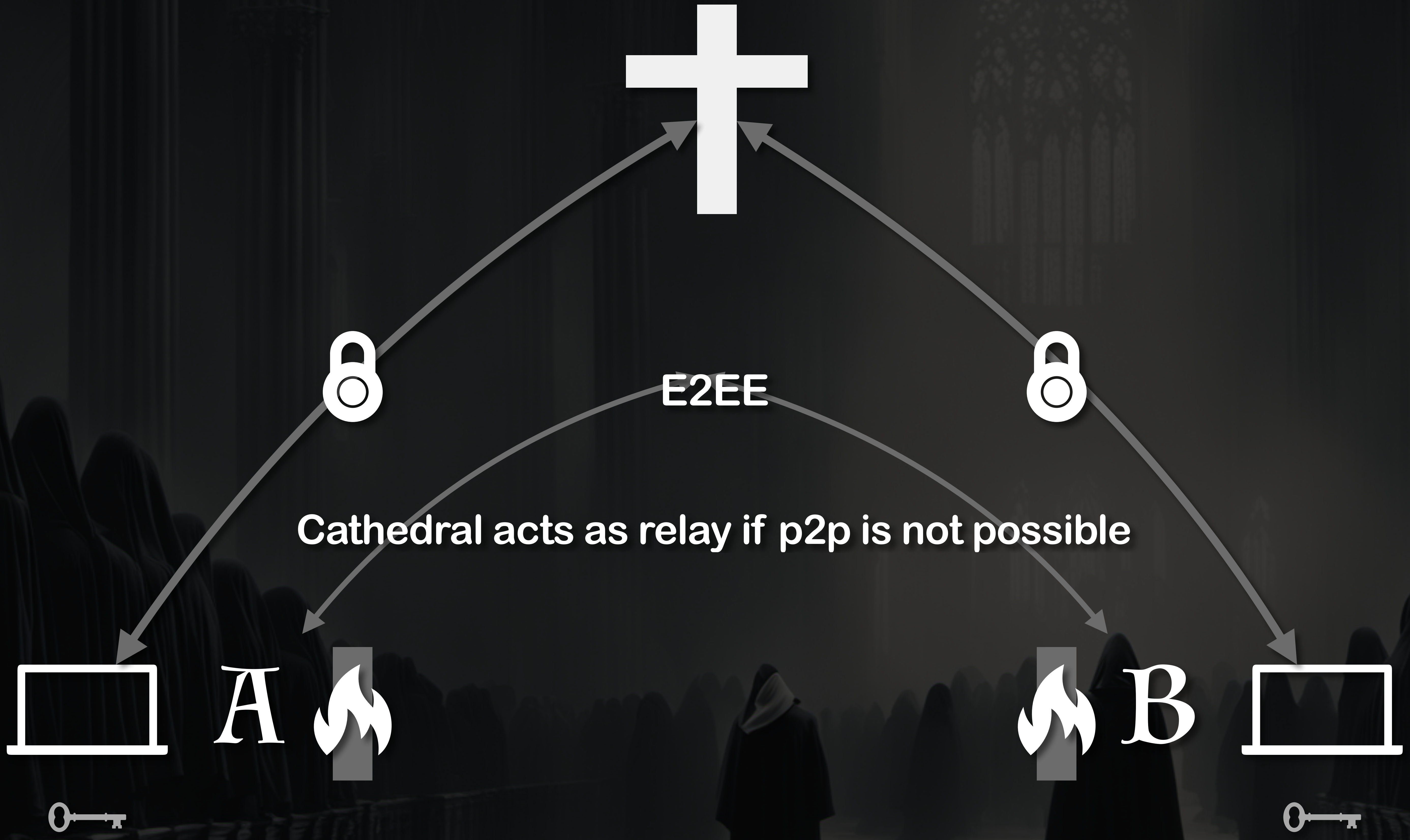
Hole punching to update NAT states on firewall.

A and B now send traffic directly to each other.



Cathedrals

Discovery



Cathedrals

Discovery

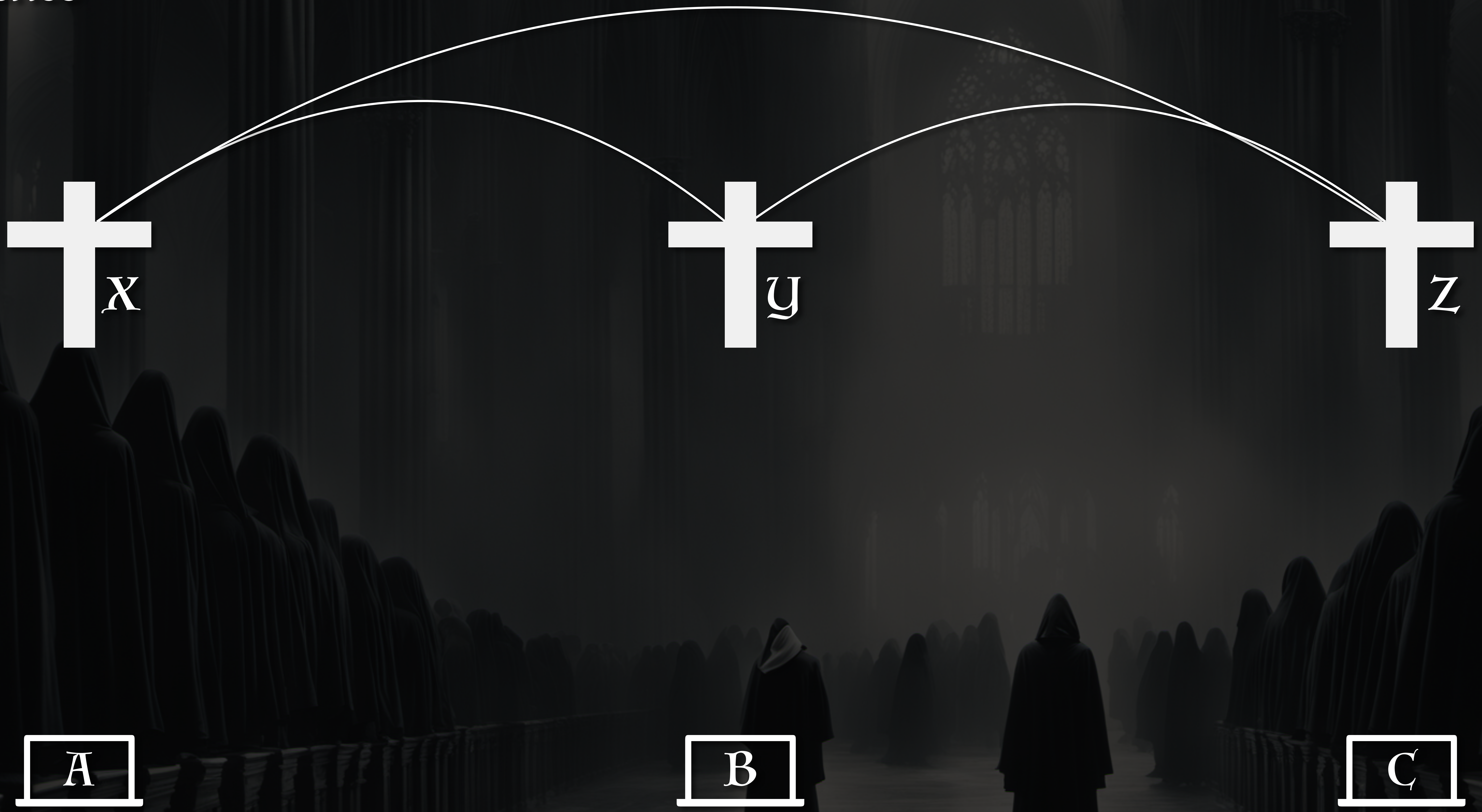
- Cathedrals help devices find each other.
- Cathedrals facilitate tunnel establishment.
- Your devices are reachable no matter where they are.
  - No need for firewall adjustments.
  - It just works.

Cathedrals

Resilience

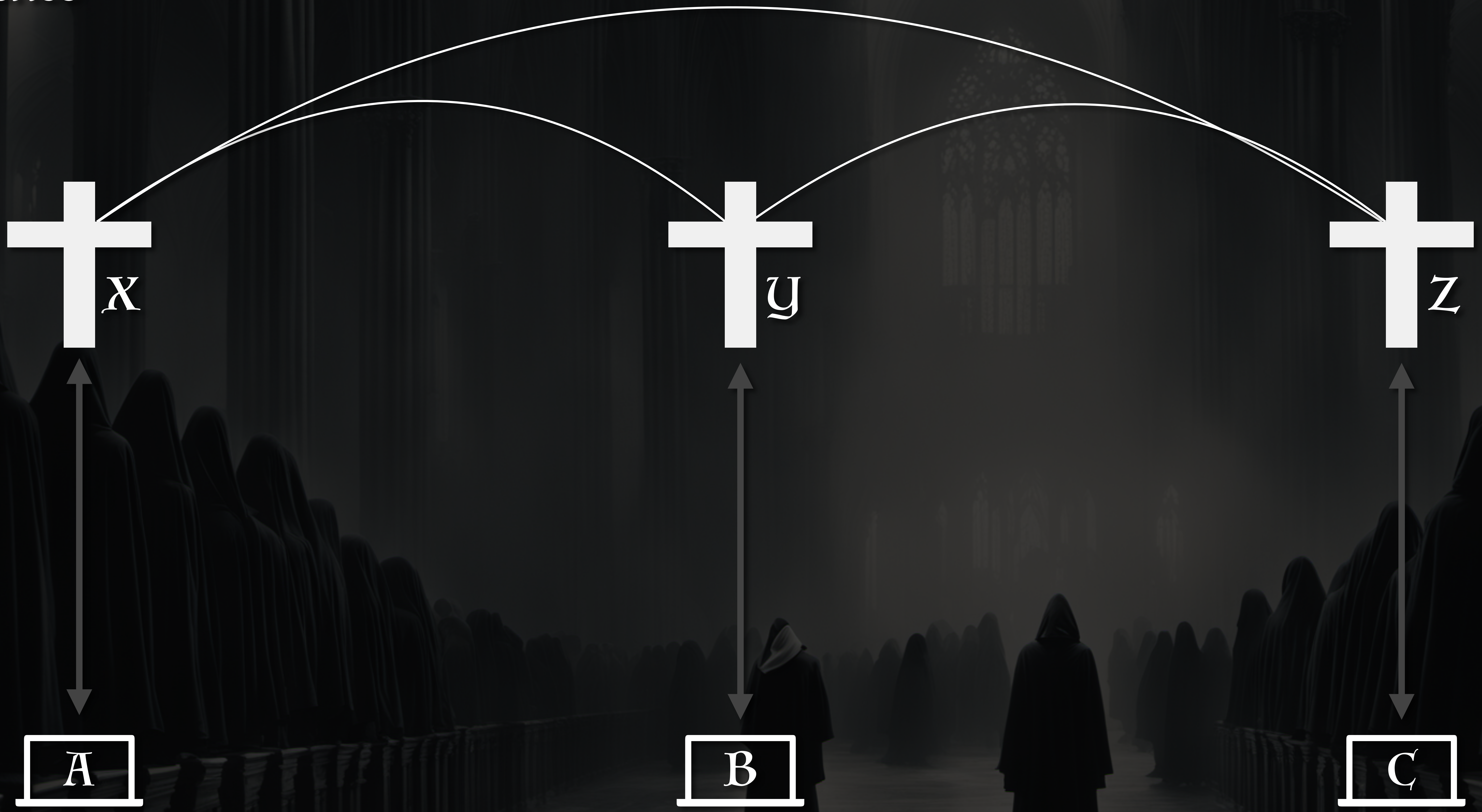
Cathedrals

Resilience



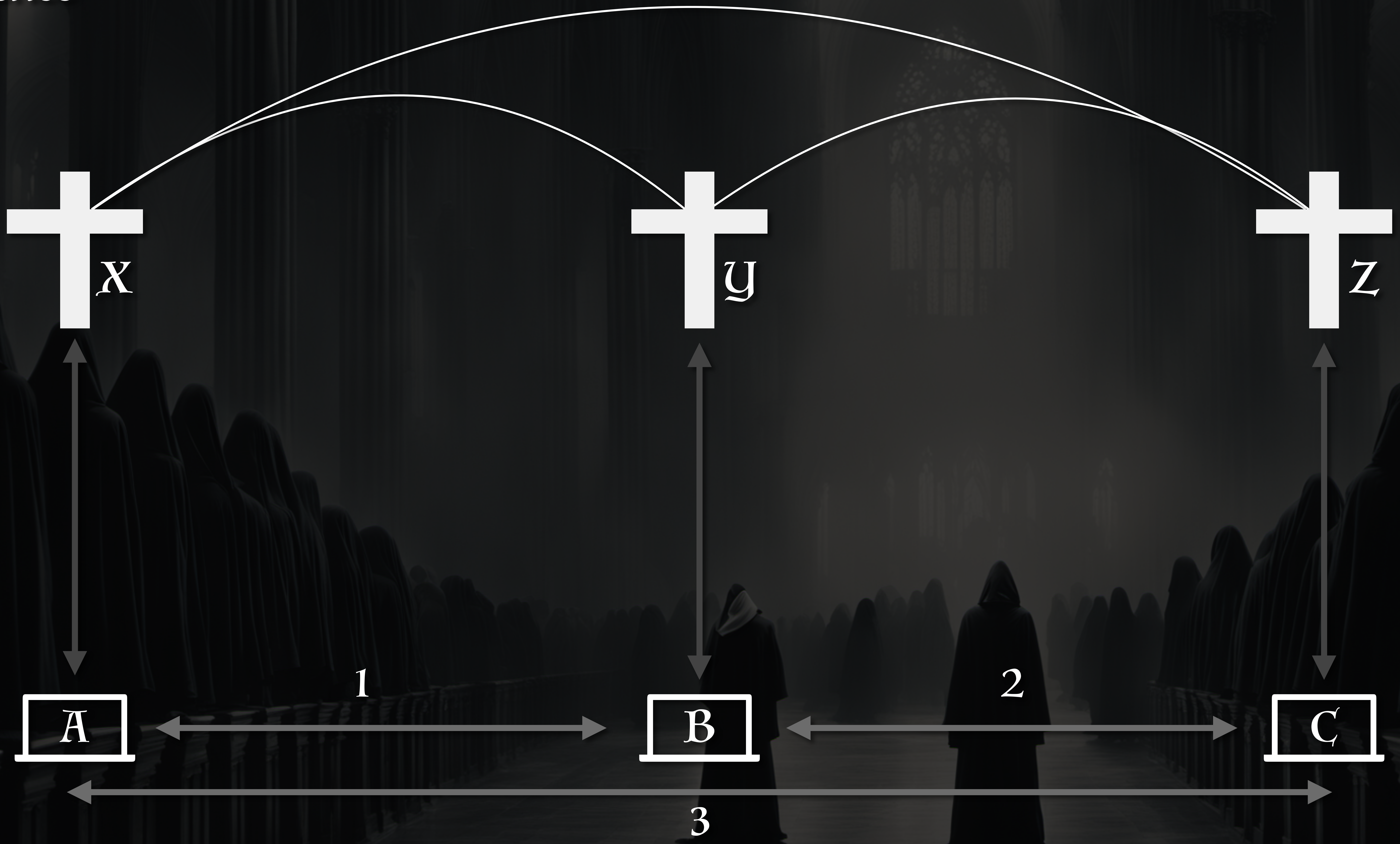
Cathedrals

Resilience



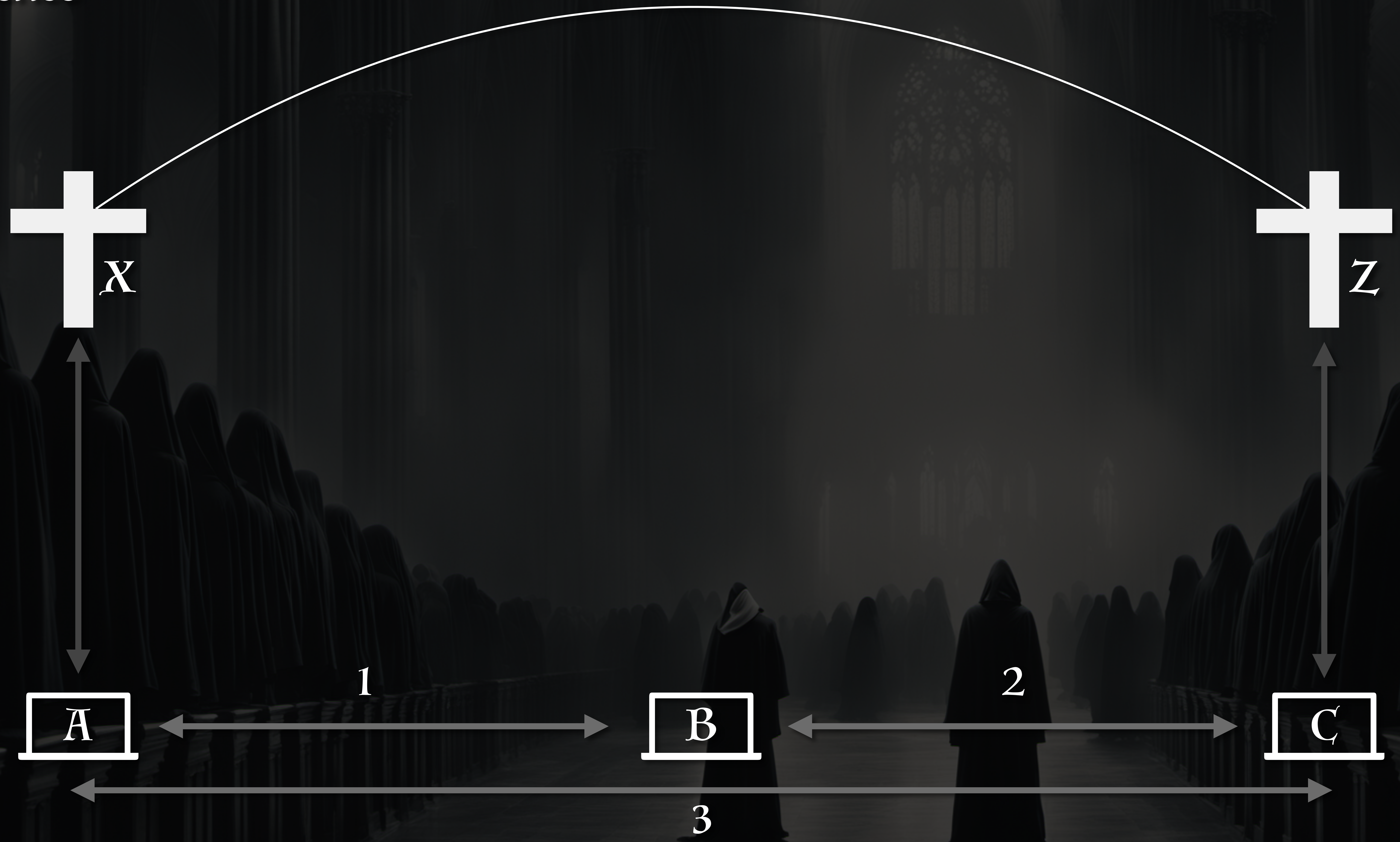
Cathedrals

Resilience

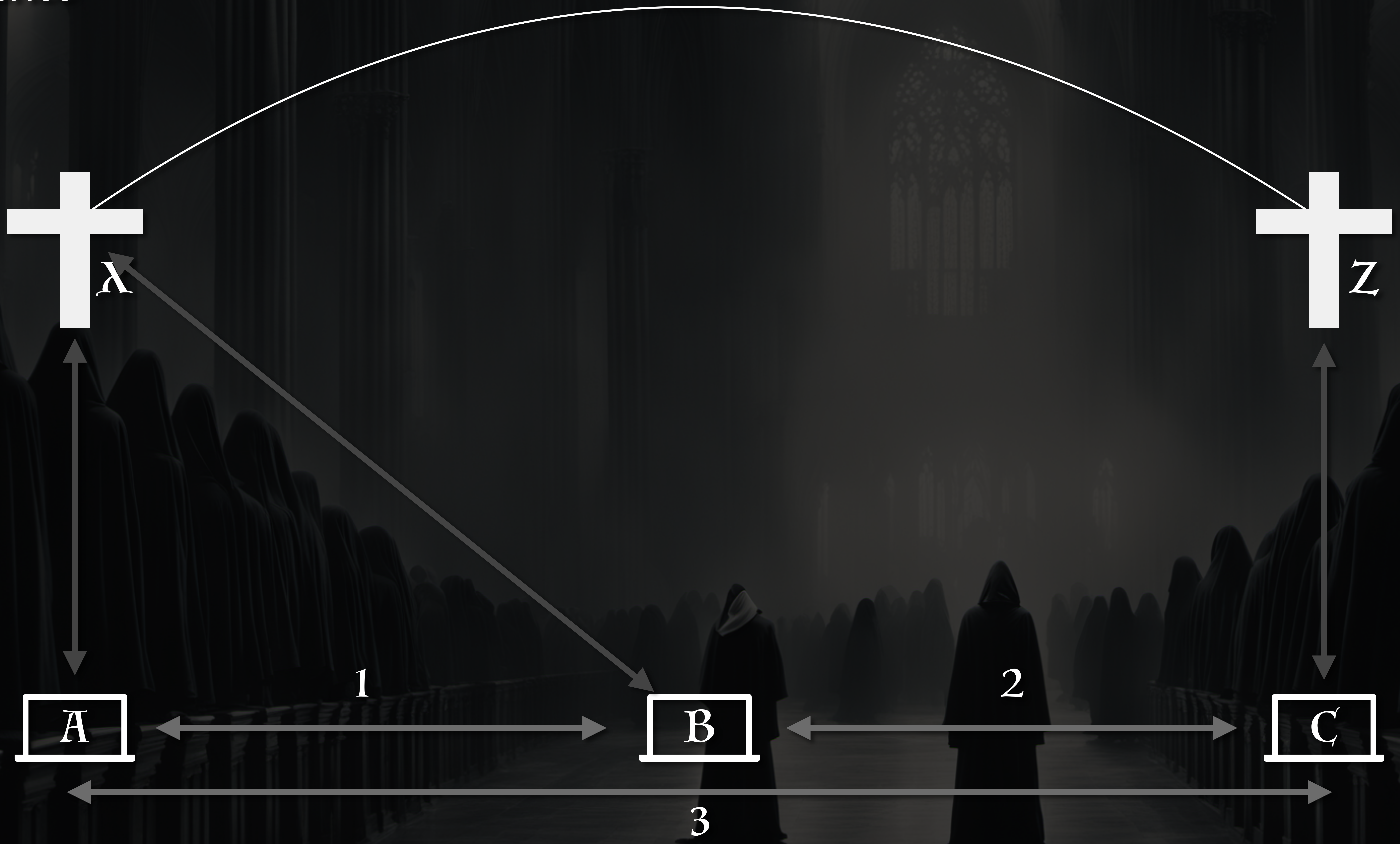


Cathedrals

Resilience



Cathedrals
Resilience



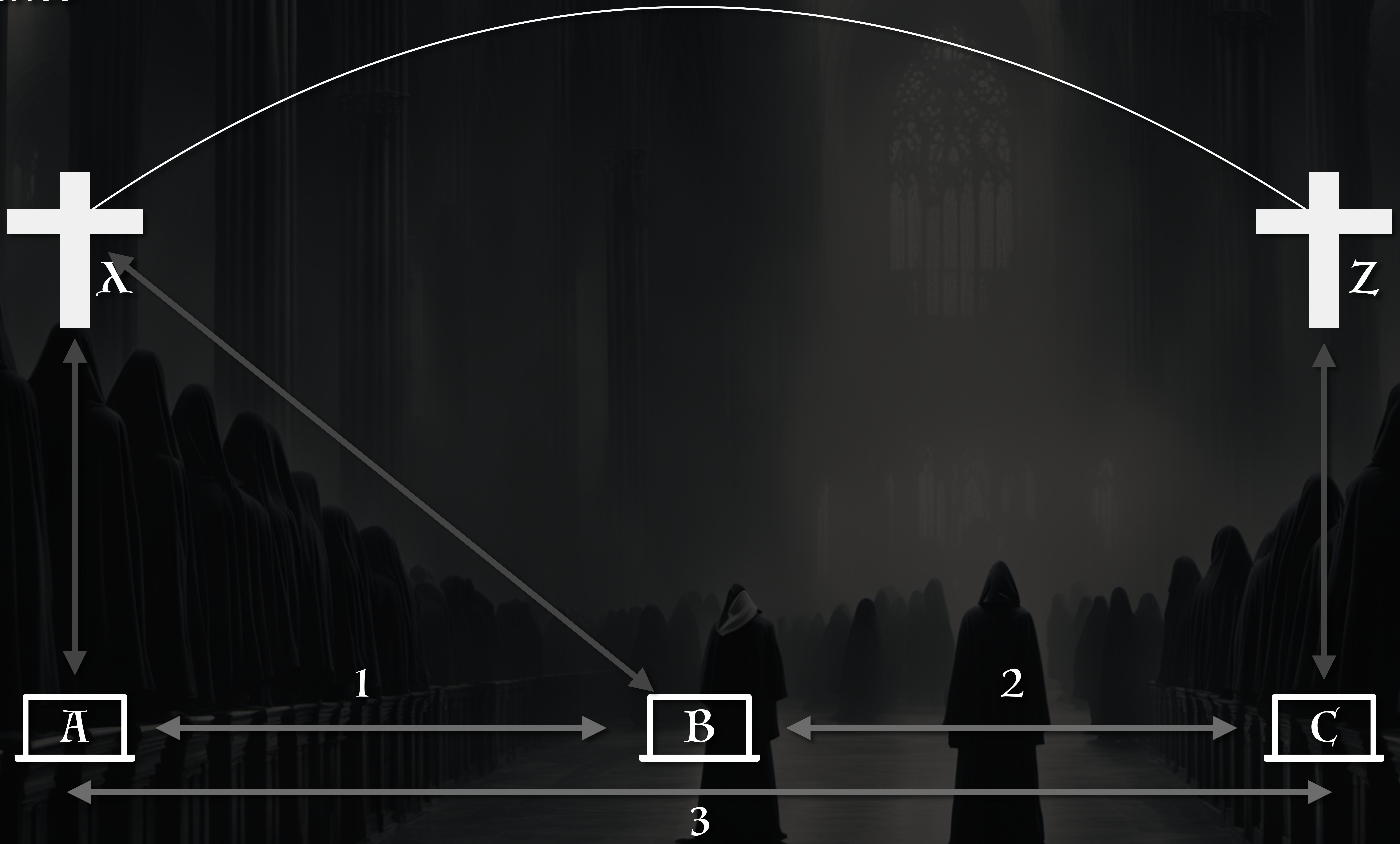
Cathedrals

Resilience

- Federated cathedrals update each other about their devices.
- Devices do not have to be talking to the same cathedral.
- Cathedrals will tell devices about other cathedrals it knows.
- Devices can failover to other cathedrals if required.

Cathedrals

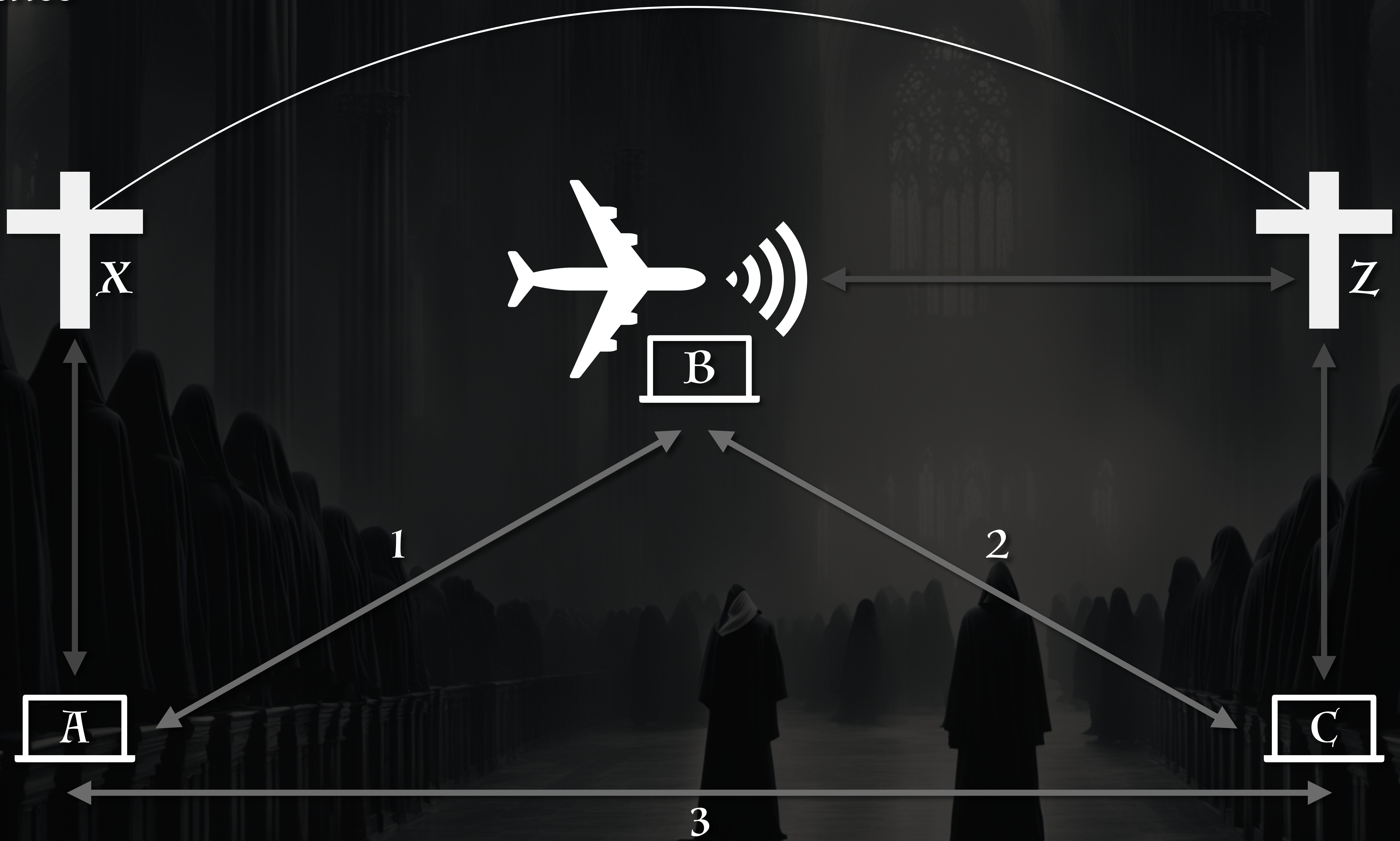
Resilience



SEC-T 2025

Cathedrals

Resilience



[redacted]
[redacted]

Cathedrals
Resilience

- All cathedrals must be taken offline at the same time to prevent NEW tunnels from being established.
  - Established p2p tunnels are entirely unaffected.
- Almost impossible to take down.
  - Distribute cathedrals over different providers.
  - Clever automation using chef, ansible, etc.
- Time to recover.

Cathedrals

Key Distribution

Cathedrals
Key Distribution

- Cathedrals act as a key distribution point.
- Cathedrals hold encrypted shared secrets for tunnels.
  - Wrapped with unique per device Key-Encryption-Keys (KEKs).
  - Wrapped keys are called Ambries.
  - Cathedral cannot modify or read Ambries.
- Cathedrals distribute these to the correct peers when needed.

Cathedrals
Key Distribution

- Ambries can be updated at any time.
  - Automate your key flow.
- Devices automatically rekey upon receiving a new Ambry.
  - Roll out keys very quickly.

Cathedrals

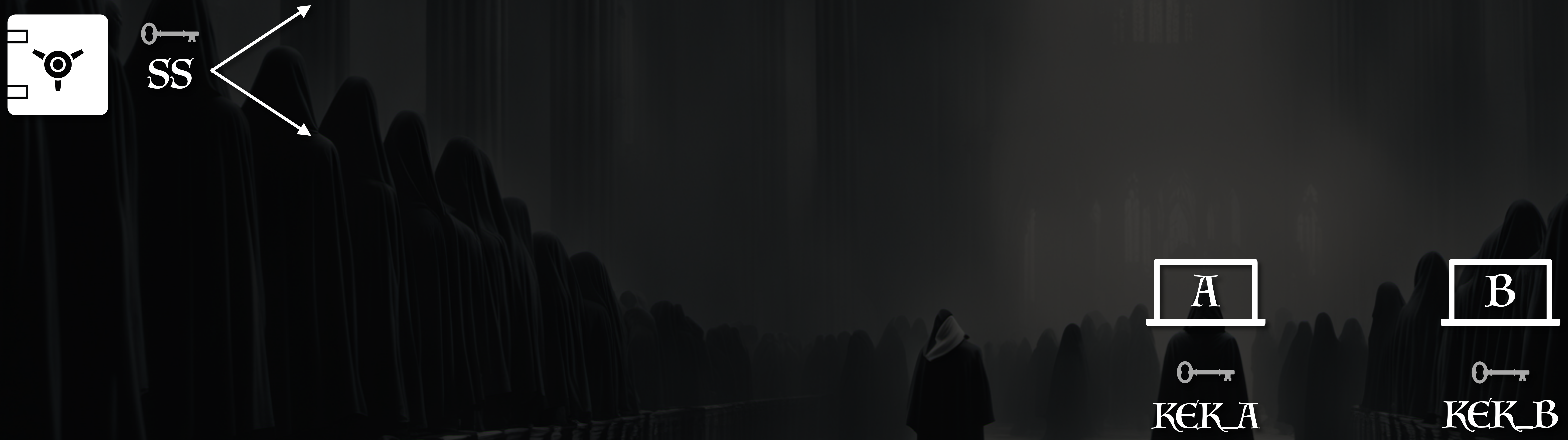
Key Distribution

Cathedrals

Key Distribution

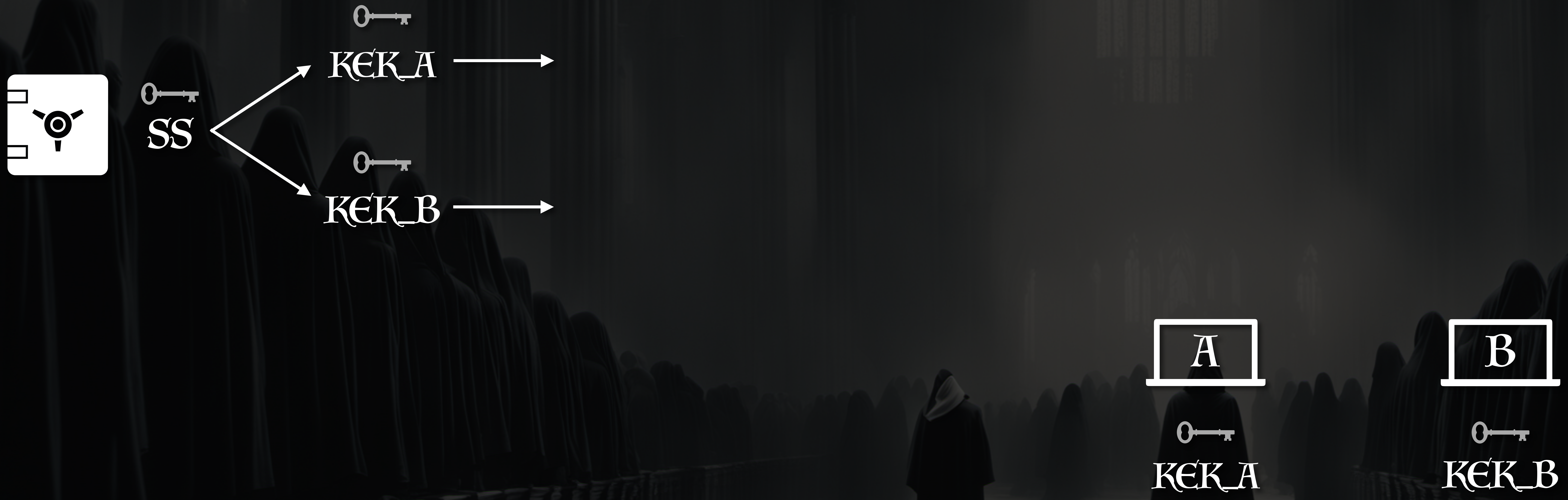
Cathedrals

Key Distribution



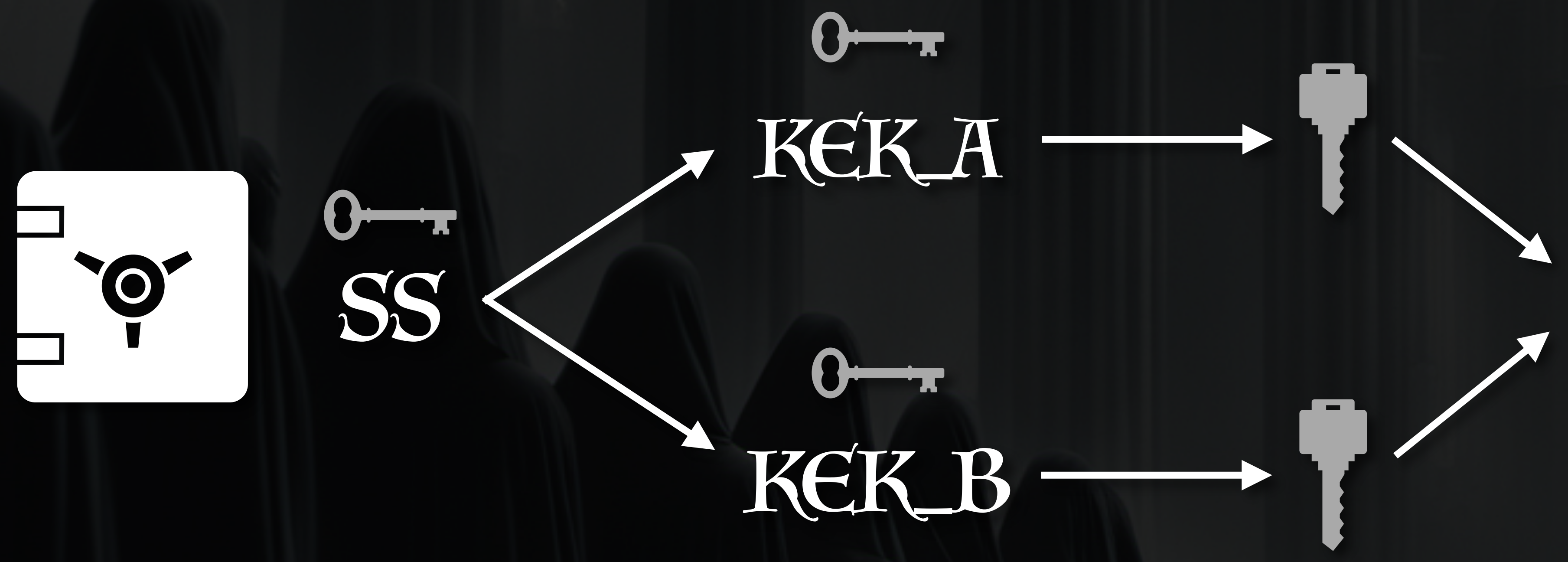
Cathedrals

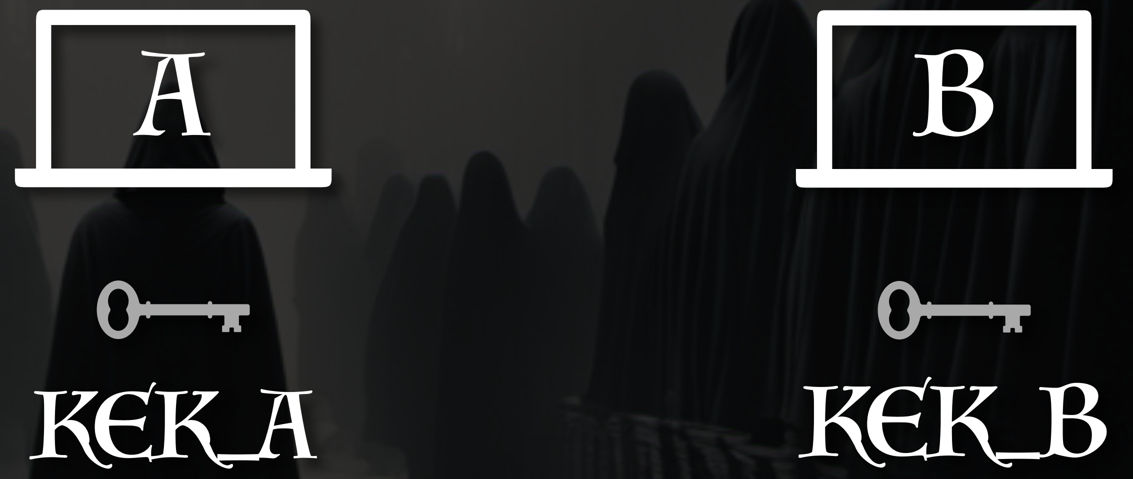
Key Distribution



Cathedrals

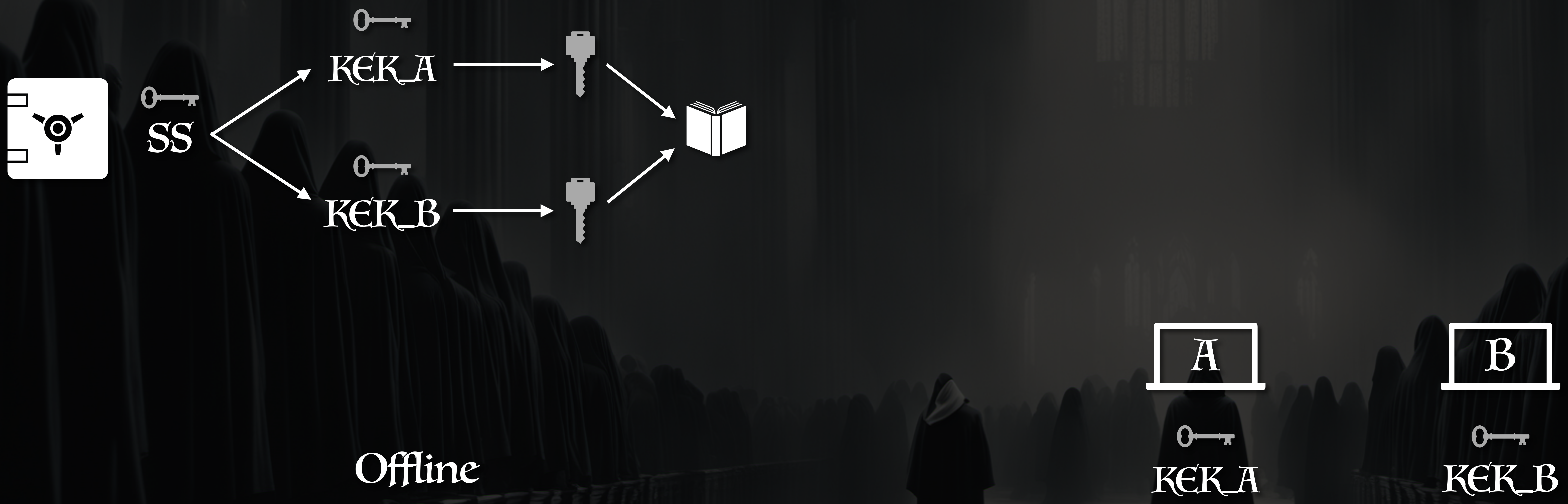
Key Distribution





Cathedrals

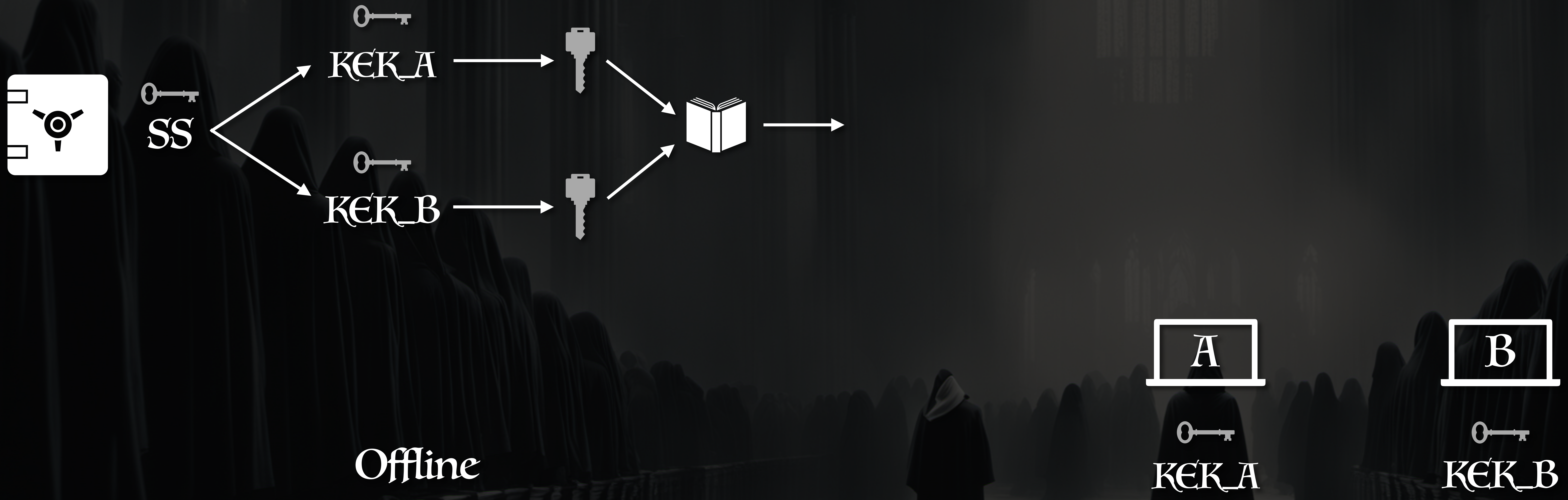
Key Distribution



Cathedrals

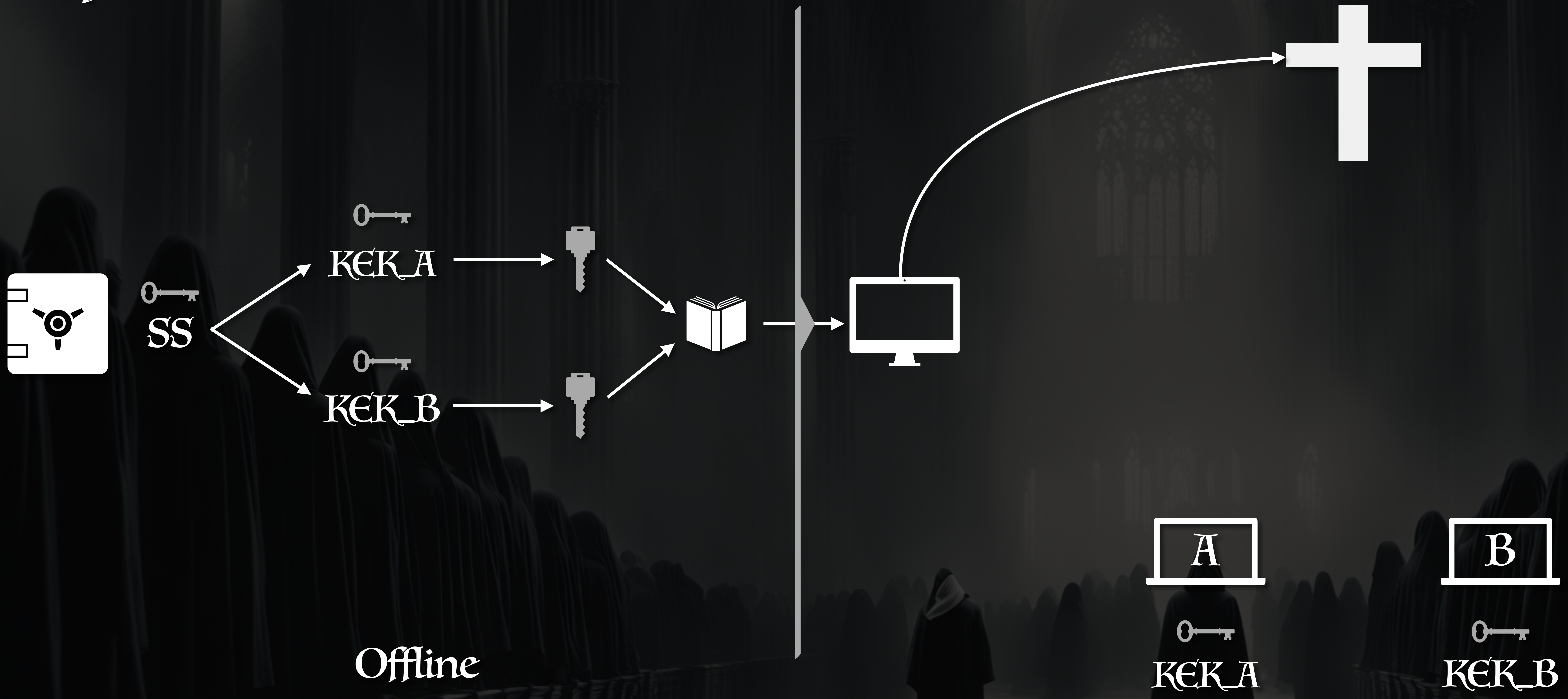
Key Distribution

Online

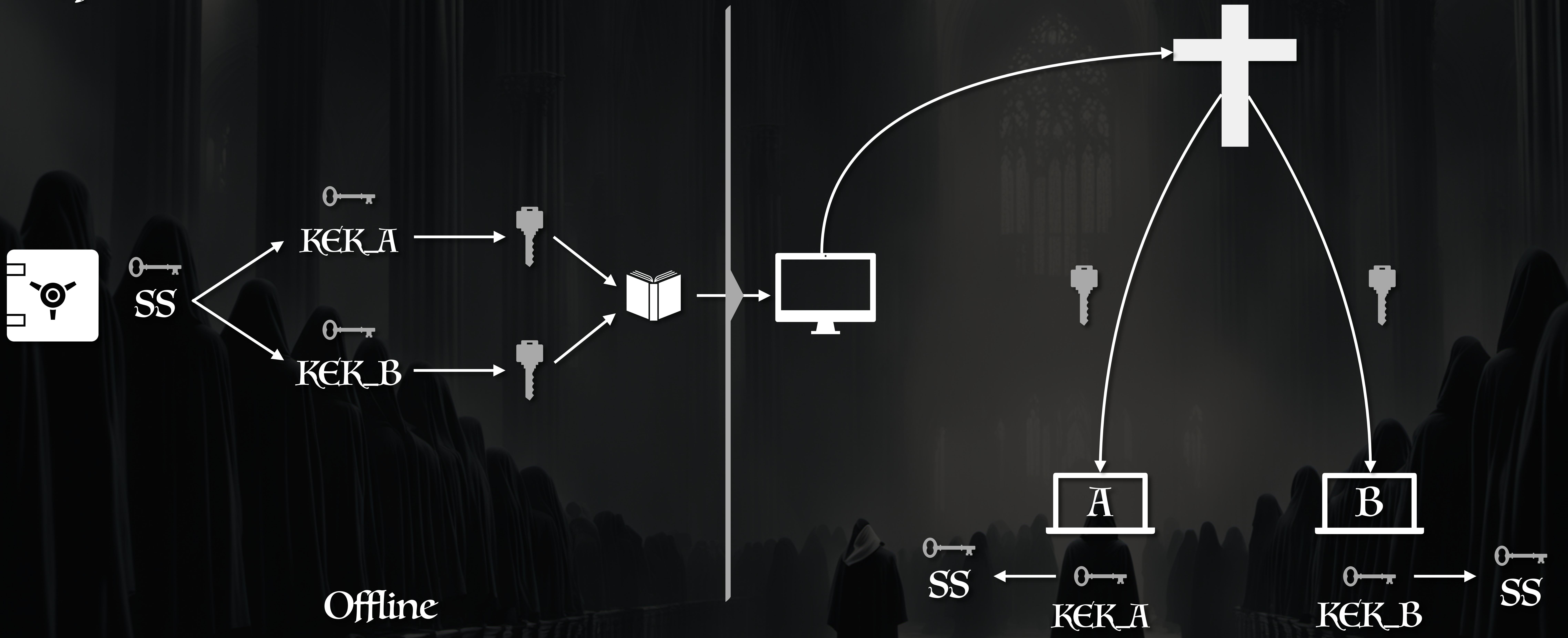


Cathedrals

Key Distribution

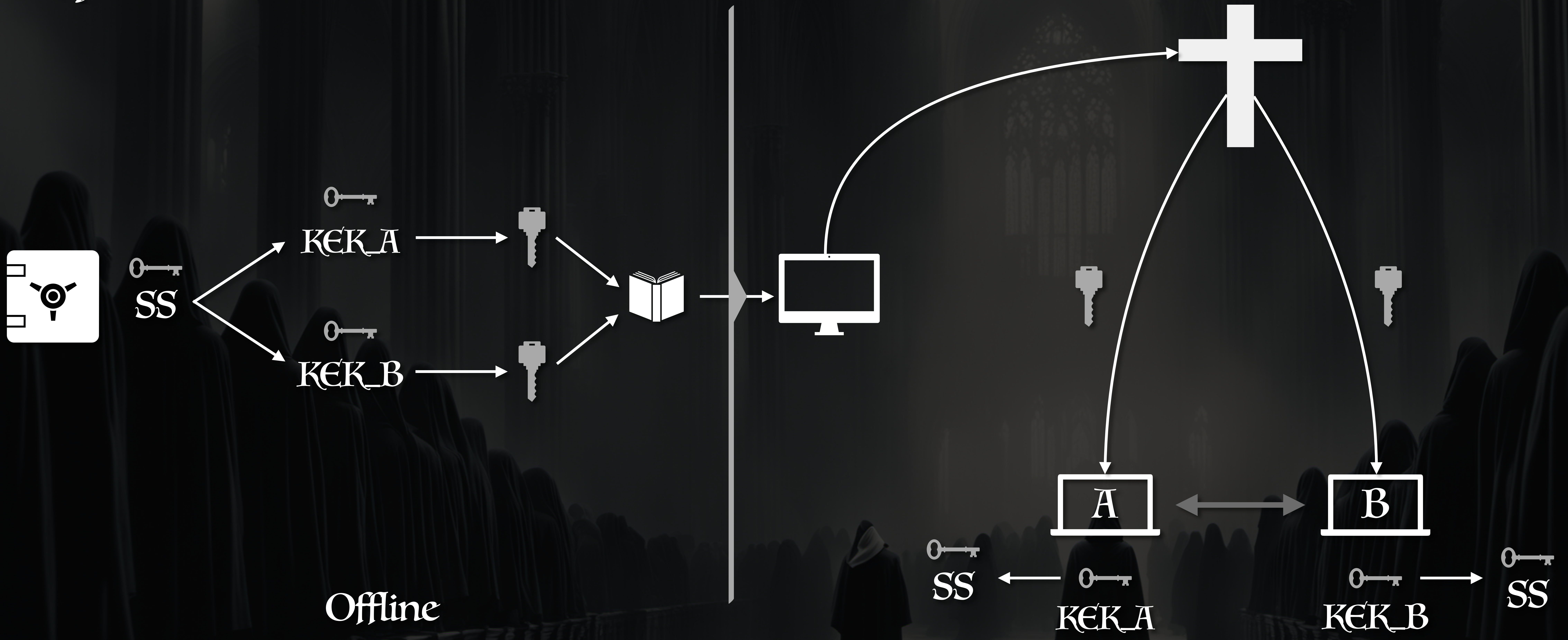


Cathedrals
Key Distribution



Cathedrals

Key Distribution



Cathedrals

Key Distribution

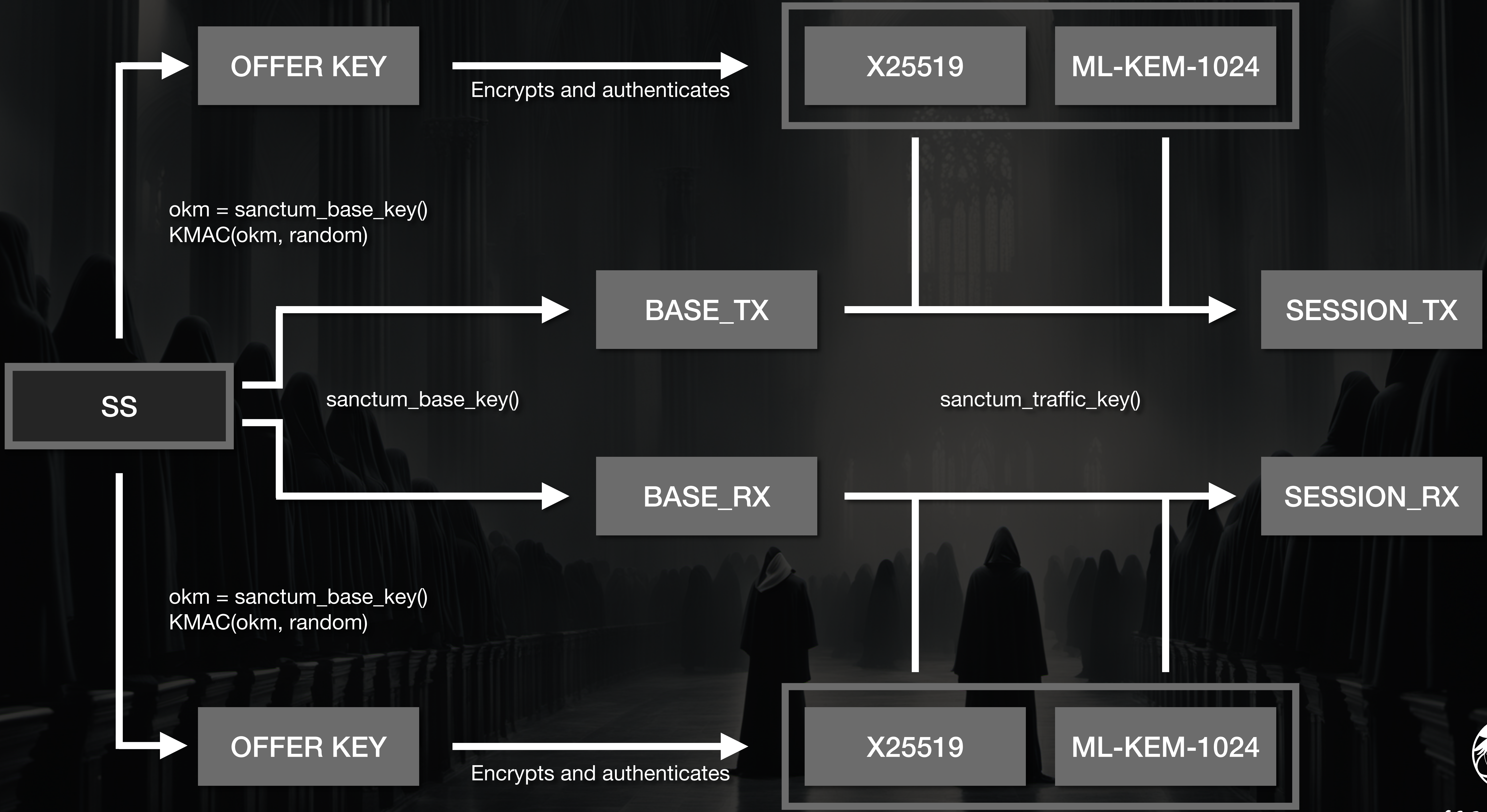
```
archael@gotyon ~ % ambry generate 0x53616e6374756d
generating device KEKs under 53616e63747500 ... done
deriving internal flock KEKs ... done
archael@gotyon ~ % ambry bundle 0x53616e6374756d 0x53616e6374756d ambry.bundle
ambry.bundle: generated 32385 tunnels, generation 0x649af776
archael@gotyon ~ %
```

Cathedrals

Key Distribution

$$\begin{tikzcd} & & \text{AMBRY} \arrow[d] \\ \text{KEK} \arrow[r, "\text{sanctum\_base\_key()}"'] & \text{KEK\_UNWRAP} \arrow[r] & \text{DECRYPT} \arrow[d] \\ & & \boxed{\text{SS}} \end{tikzcd}$$

Sanctum


OFFER KEY
Encrypts and authenticates
X25519
ML-KEM-1024
okm = sanctum_base_key()
KMAC(okm, random)
BASE_TX
SESSION_TX
SS
sanctum_base_key()
sanctum_traffic_key()
BASE_RX
SESSION_RX
okm = sanctum_base_key()
KMAC(okm, random)
OFFER KEY
Encrypts and authenticates
X25519
ML-KEM-1024

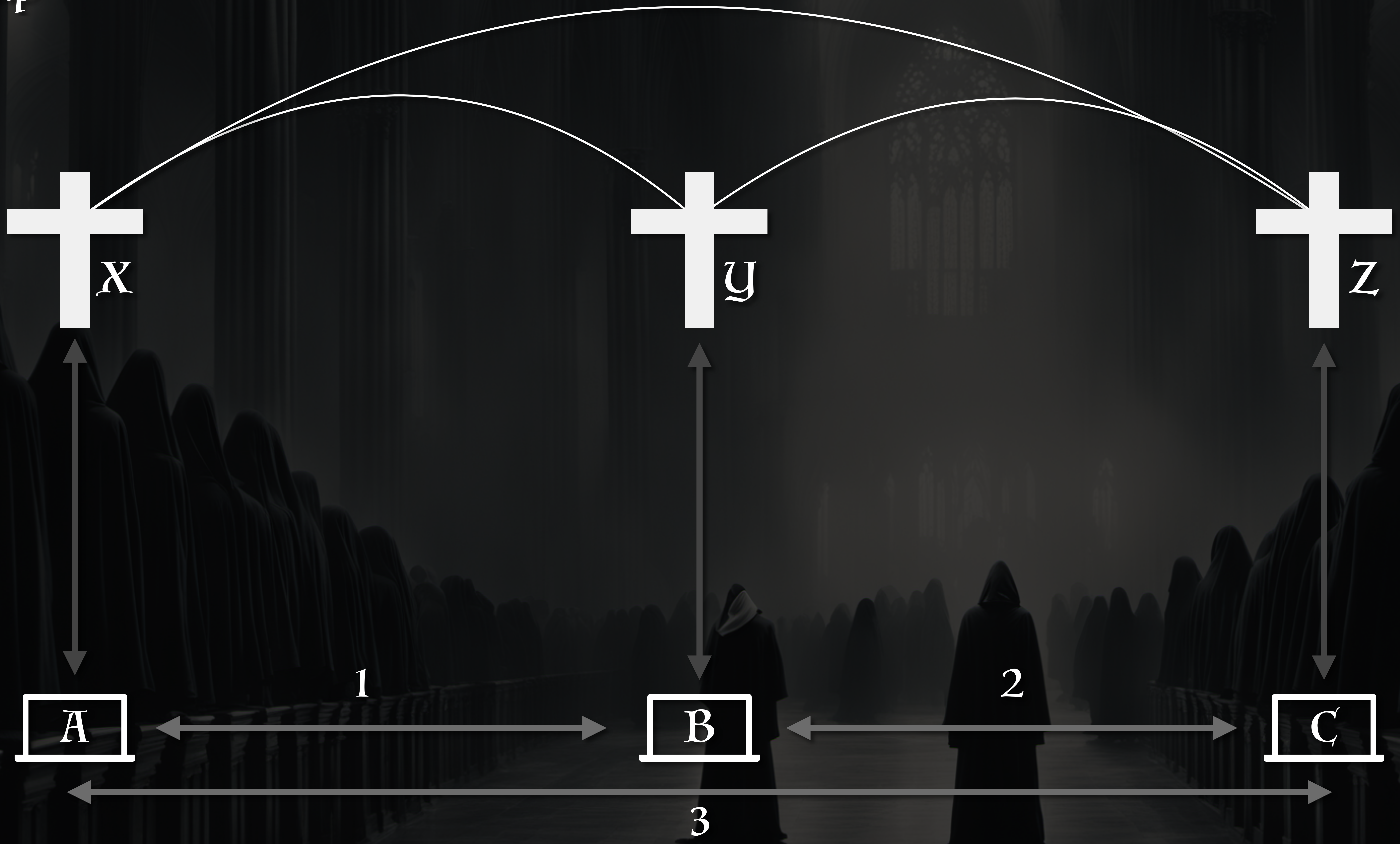
Cathedrals

Recap

- Discovery
  - Find and establish tunnels to devices, no matter where they are.
- Distributed
  - Federate with other cathedrals to create global network.
- Resilience
  - As long as one cathedral survives, communication survives
  - Established p2p e2ee tunnels unaffected

Cathedrals

Recap



The Library

The Library

- libkyrka
- A complete implementation of the sanctum protocol.
- Embeddable into your application.
- Written in good old C99.
- Runs on Linux, OpenBSD, MacOS, Windows, Android, iOS.

The Library

- Designed to be IO agonistic.
  - Tunnels over IP
  - Tunnels over Serial
  - Tunnels over USB
  - Tunnels over Discord (pls don’t ban me)
  - Your imagination is the limit.

The Library

- Designed to be IO agonistic (*).
  - Tunnels over IP
  - Tunnels over Serial
  - Tunnels over USB
  - Tunnels over Discord (pls don't ban me)
  - Your imagination is the limit.

* cathedrals require IP connectivity.

The Library

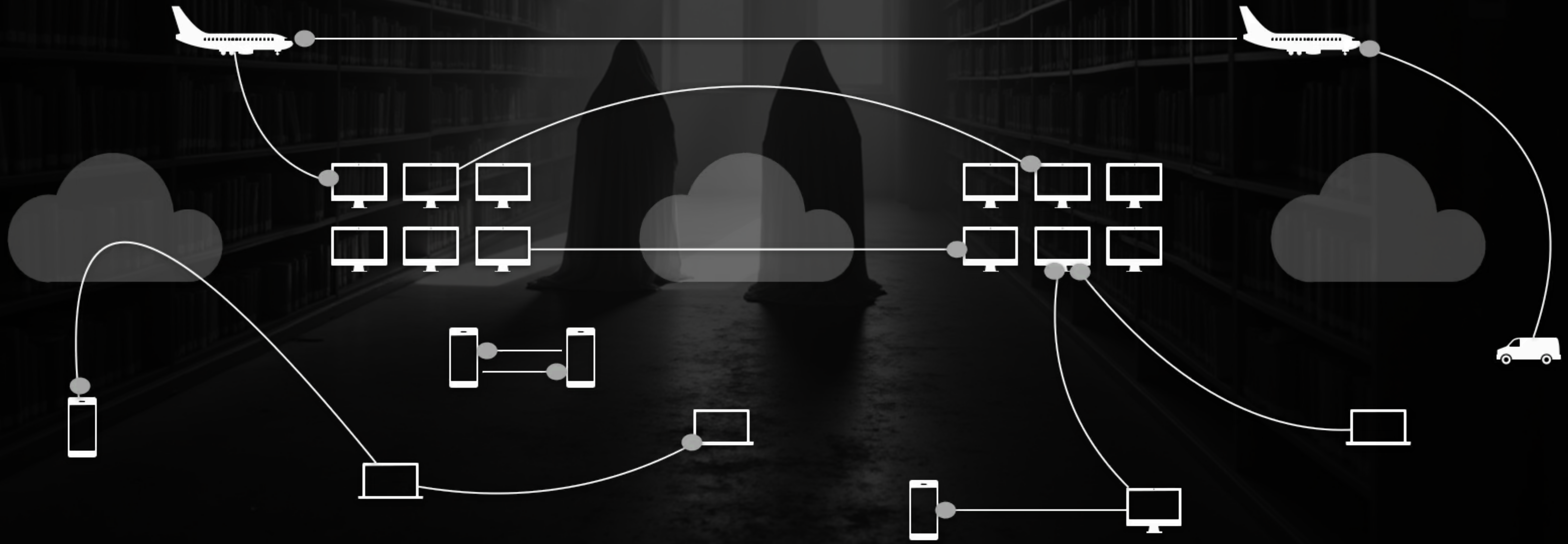
- No real option for sandboxing, that is up to applications.
- Non time-critical sensitive assets masked in memory.
- Prevents easy or accidental exfiltration.
- Makes it harder on adversaries to obtain secrets.
- Does not make it impossible, but substantially raises the bar.

The Library

- Opens up the encrypted tunnel ecosystem to much more.
- Voice, Data, Text, Images, Inference data, MCP, etc.

The Library

- Opens up the encrypted tunnel ecosystem to much more.
- Voice, Data, Text, Images, Inference data, MCP, etc.



The Library

```c
static struct tunnel *
tunnel_setup(void)
{
        struct tunnel                   *tun;

        if ((tun = calloc(1, sizeof(*tun))) == NULL)
                fatal("calloc");

        if ((tun->fd = socket(AF_INET, SOCK_DGRAM, 0)) == -1)
                fatal("socket: %s", strerror(errno));

        if (bind(tun->fd, (const struct sockaddr *)&local, sizeof(local)) == -1)
                fatal("bind: %s", strerror(errno));

        if ((tun->ctx = kyrka_ctx_alloc(tunnel_event, tun)) == NULL)
                fatal("kyrka_ctx_alloc: failed");

        if (kyrka_heaven_ifc(tun->ctx, tunnel_plaintext, tun) == -1)
                fatal("kyrka_heaven_ifc: %d", kyrka_last_error(tun->ctx));

        if (kyrka_purgatory_ifc(tun->ctx, tunnel_ciphertext, tun) == -1)
                fatal("kyrka_purgatory_ifc: %d", kyrka_last_error(tun->ctx));

        if (kyrka_secret_load_path(tun->ctx, "secret.key") == -1)
                fatal("kyrka_secret_load_path: %d", kyrka_last_error(tun->ctx));

        if (kyrka_encap_key_load(tun->ctx, tek, sizeof(tek)) == -1)
                fatal("kyrka_encap_key_load: %d", kyrka_last_error(tun->ctx));

        return (tun);
}
```

The Library

```c
static struct tunnel *
tunnel_setup(void)
{
        struct tunnel                   *tun;

        if ((tun = calloc(1, sizeof(*tun))) == NULL)
                fatal("calloc");

        if ((tun->fd = socket(AF_INET, SOCK_DGRAM, 0)) == -1)
                fatal("socket: %s", strerror(errno));

        if (bind(tun->fd, (const struct sockaddr *)&local, sizeof(local)) == -1)
                fatal("bind: %s", strerror(errno));

        if ((tun->ctx = kyrka_ctx_alloc(tunnel_event, tun)) == NULL)
                fatal("kyrka_ctx_alloc: failed");

        if (kyrka_heaven_ifc(tun->ctx, tunnel_plaintext, tun) == -1)
                fatal("kyrka_heaven_ifc: %d", kyrka_last_error(tun->ctx));

        if (kyrka_purgatory_ifc(tun->ctx, tunnel_ciphertext, tun) == -1)
                fatal("kyrka_purgatory_ifc: %d", kyrka_last_error(tun->ctx));

        if (kyrka_secret_load_path(tun->ctx, "secret.key") == -1)
                fatal("kyrka_secret_load_path: %d", kyrka_last_error(tun->ctx));

        if (kyrka_encap_key_load(tun->ctx, tek, sizeof(tek)) == -1)
                fatal("kyrka_encap_key_load: %d", kyrka_last_error(tun->ctx));

        return (tun);
}
```

The Library

```
static struct tunnel *
tunnel_setup(void)
{
        struct tunnel                   *tun;

        if ((tun = calloc(1, sizeof(*tun))) == NULL)
                fatal("calloc");

        if ((tun->fd = socket(AF_INET, SOCK_DGRAM, 0)) == -1)
                fatal("socket: %s", strerror(errno));

        if (bind(tun->fd, (const struct sockaddr *)&local, sizeof(local)) == -1)
                fatal("bind: %s", strerror(errno));

        if ((tun->ctx = kyrka_ctx_alloc(tunnel_event, tun)) == NULL)
                fatal("kyrka_ctx_alloc: failed");

        if (kyrka_heaven_ifc(tun->ctx, tunnel_plaintext, tun) == -1)
                fatal("kyrka_heaven_ifc: %d", kyrka_last_error(tun->ctx));

        if (kyrka_purgatory_ifc(tun->ctx, tunnel_ciphertext, tun) == -1)
                fatal("kyrka_purgatory_ifc: %d", kyrka_last_error(tun->ctx));

        if (kyrka_secret_load_path(tun->ctx, "secret.key") == -1)
                fatal("kyrka_secret_load_path: %d", kyrka_last_error(tun->ctx));

        if (kyrka_encap_key_load(tun->ctx, tek, sizeof(tek)) == -1)
                fatal("kyrka_encap_key_load: %d", kyrka_last_error(tun->ctx));

        return (tun);
}
```

The Library

```c
static struct tunnel *
tunnel_setup(void)
{
	struct tunnel			*tun;

	if ((tun = calloc(1, sizeof(*tun))) == NULL)
		fatal("calloc");

	if ((tun->fd = socket(AF_INET, SOCK_DGRAM, 0)) == -1)
		fatal("socket: %s", strerror(errno));

	if (bind(tun->fd, (const struct sockaddr *)&local, sizeof(local)) == -1)
		fatal("bind: %s", strerror(errno));

	if ((tun->ctx = kyrka_ctx_alloc(tunnel_event, tun)) == NULL)
		fatal("kyrka_ctx_alloc: failed");

	if (kyrka_heaven_ifc(tun->ctx, tunnel_plaintext, tun) == -1)
		fatal("kyrka_heaven_ifc: %d", kyrka_last_error(tun->ctx));

	if (kyrka_purgatory_ifc(tun->ctx, tunnel_ciphertext, tun) == -1)
		fatal("kyrka_purgatory_ifc: %d", kyrka_last_error(tun->ctx));

	if (kyrka_secret_load_path(tun->ctx, "secret.key") == -1)
		fatal("kyrka_secret_load_path: %d", kyrka_last_error(tun->ctx));

	if (kyrka_encap_key_load(tun->ctx, tek, sizeof(tek)) == -1)
		fatal("kyrka_encap_key_load: %d", kyrka_last_error(tun->ctx));

	return (tun);
}
```

The Library

```c
static void
tunnel_plaintext(const void *data, size_t len, u_int64_t seq, void *udata)
{
	printf("<< %.*s\n", (int)len, (const char *)data);
}

static void
tunnel_ciphertext(const void *data, size_t len, u_int64_t seq, void *udata)
{
	struct tunnel		*tun;

	tun = udata;

	if (sendto(tun->fd, data, len, 0,
	    (const struct sockaddr *)&peer, sizeof(peer)) == -1)
		fatal("sendto: %s", strerror(errno));
}
```

The Library

```
static void
tunnel_plaintext(const void *data, size_t len, u_int64_t seq, void *udata)
{
        printf("<< %.*s\n", (int)len, (const char *)data);
}

static void
tunnel_ciphertext(const void *data, size_t len, u_int64_t seq, void *udata)
{
        struct tunnel           *tun;

        tun = udata;

        if (sendto(tun->fd, data, len, 0,
            (const struct sockaddr *)&peer, sizeof(peer)) == -1)
                fatal("sendto: %s", strerror(errno));
}
```

The Library

```c
static void
tunnel_read(int fd, KYRKA *ctx)
{
        ssize_t                 ret;
        u_int8_t                pkt[1500];

        if ((ret = recv(fd, pkt, sizeof(pkt), MSG_DONTWAIT)) == -1) {
                if (errno == EWOULDBLOCK || errno == EAGAIN)
                        return;
                fatal("recv: %s", strerror(errno));
        }

        if (kyrka_purgatory_input(ctx, pkt, ret) == -1 &&
            kyrka_last_error(ctx) != KYRKA_ERROR_NO_RX_KEY)
                fatal("kyrka_purgatory_input: %d", kyrka_last_error(ctx));
}
```

The Library

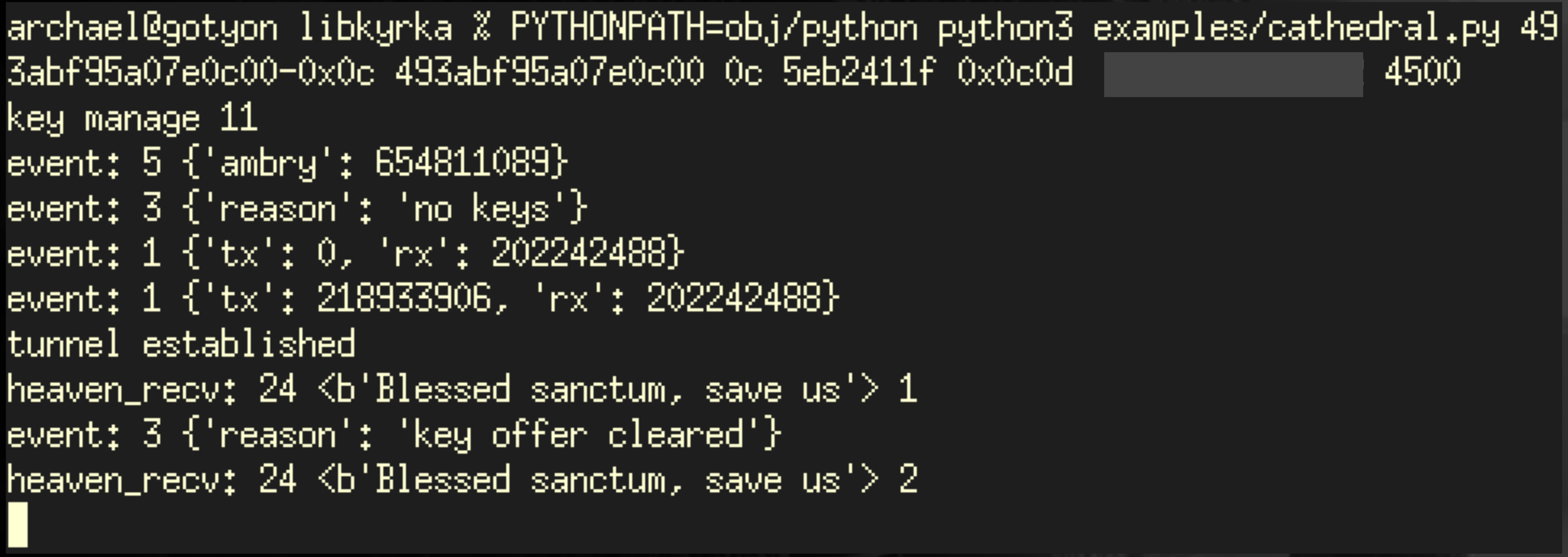
```
static void
tunnel_read(int fd, KYRKA *ctx)
{
        ssize_t         ret;
        u_int8_t        pkt[1500];

        if ((ret = recv(fd, pkt, sizeof(pkt), MSG_DONTWAIT)) == -1) {
                if (errno == EWOULDBLOCK || errno == EAGAIN)
                        return;
                fatal("recv: %s", strerror(errno));
        }

        if (kyrka_purgatory_input(ctx, pkt, ret) == -1 &&
            kyrka_last_error(ctx) != KYRKA_ERROR_NO_RX_KEY)
                fatal("kyrka_purgatory_input: %d", kyrka_last_error(ctx));
}
```

The Library

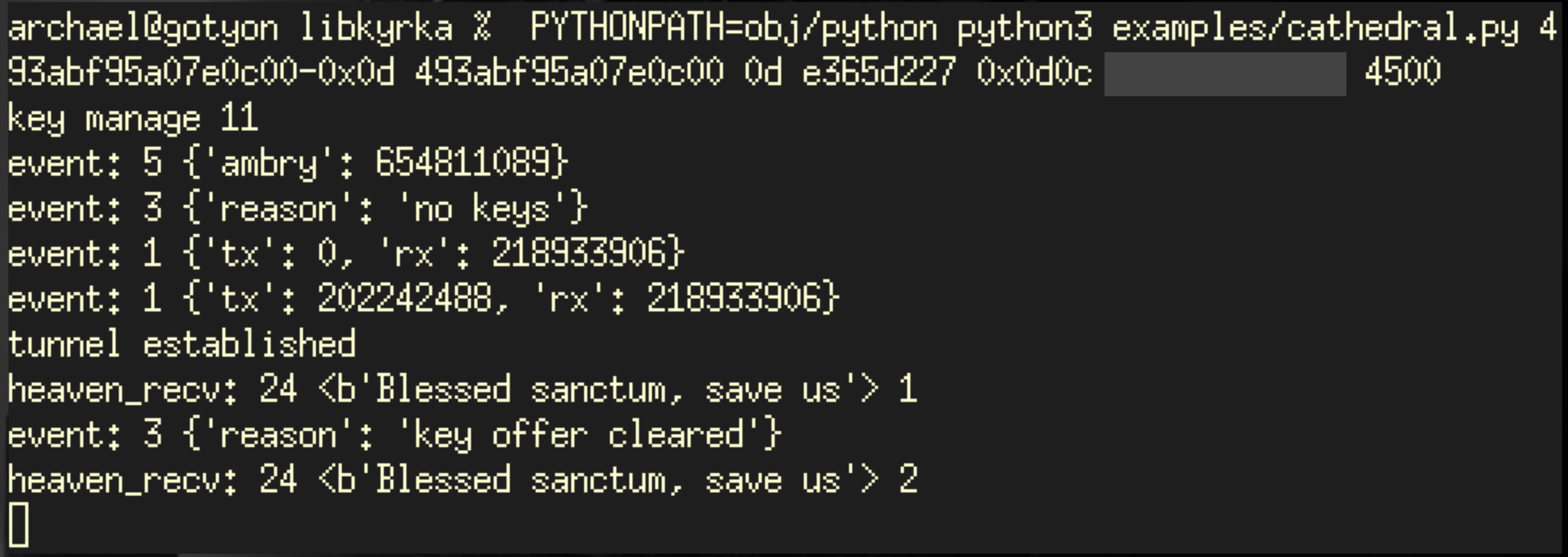
- **Python module included**

```python
ctx = libkyrka.alloc()

ctx.event_callback(event, None)
ctx.heaven_callback(heaven_recv, None)
ctx.purgatory_callback(purgatory_send, None)

try:
    with open("secret.key", "rb") as f:
        ctx.secret_load(f.read())
except Exception as e:
    print(f"error loading secret.key: {e}")
    quit()
```

The Library



```
archael@gotyon libkyrka % PYTHONPATH=obj/python python3 examples/cathedral.py 49
3abf95a07e0c00-0x0c 493abf95a07e0c00 0c 5eb2411f 0x0c0d [redacted] 4500
key manage 11
event: 5 {'ambry': 654811089}
event: 3 {'reason': 'no keys'}
event: 1 {'tx': 0, 'rx': 202242488}
event: 1 {'tx': 218933906, 'rx': 202242488}
tunnel established
heaven_recv: 24 <b'Blessed sanctum, save us'> 1
event: 3 {'reason': 'key offer cleared'}
heaven_recv: 24 <b'Blessed sanctum, save us'> 2
█
```



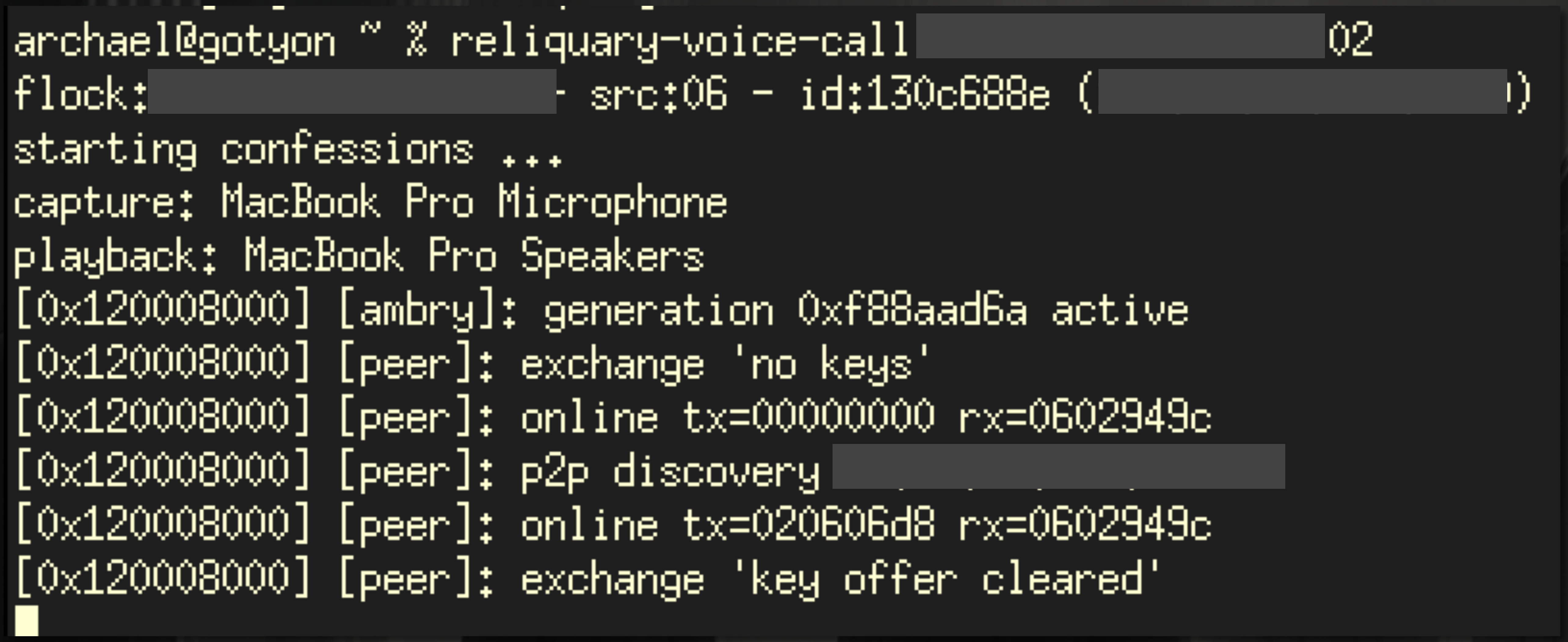
```
archael@gotyon libkyrka %  PYTHONPATH=obj/python python3 examples/cathedral.py 4
93abf95a07e0c00-0x0d 493abf95a07e0c00 0d e365d227 0x0d0c [redacted] 4500
key manage 11
event: 5 {'ambry': 654811089}
event: 3 {'reason': 'no keys'}
event: 1 {'tx': 0, 'rx': 218933906}
event: 1 {'tx': 202242488, 'rx': 218933906}
tunnel established
heaven_recv: 24 <b'Blessed sanctum, save us'> 1
event: 3 {'reason': 'key offer cleared'}
heaven_recv: 24 <b'Blessed sanctum, save us'> 2
▯
```

Confessions

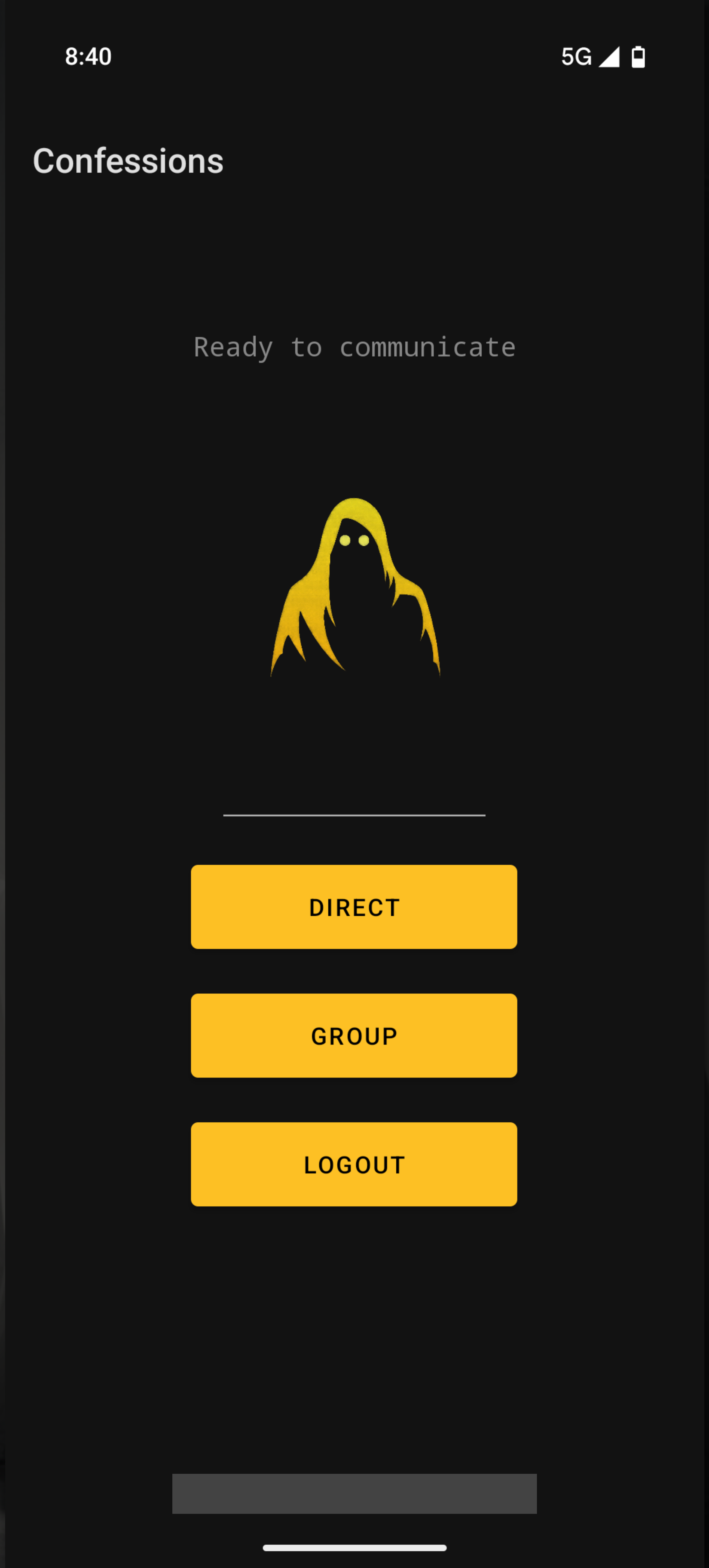
Confessions

- Confessions is a voice program written in C99.
  - Linux, OpenBSD, MacOS, Windows and Android.
- Uses libkyrka to establish secure tunnels to its peers.
  - Fully implements the sanctum protocol and can thus failover to other cathedrals and use Ambries etc.
  - All communication P2P and E2EE.
- Voice is carried over the tunnels instead of IP packets.

Confessions



```
archael@gotyon ~ % reliquary-voice-call [redacted]02
flock:[redacted]- src:06 - id:130c688e ([redacted])
starting confessions ...
capture: MacBook Pro Microphone
playback: MacBook Pro Speakers
[0x120008000] [ambry]: generation 0xf88aad6a active
[0x120008000] [peer]: exchange 'no keys'
[0x120008000] [peer]: online tx=00000000 rx=0602949c
[0x120008000] [peer]: p2p discovery [redacted]
[0x120008000] [peer]: online tx=020606d8 rx=0602949c
[0x120008000] [peer]: exchange 'key offer cleared'
```



8:40 5G

Confessions

Ready to communicate

[redacted]

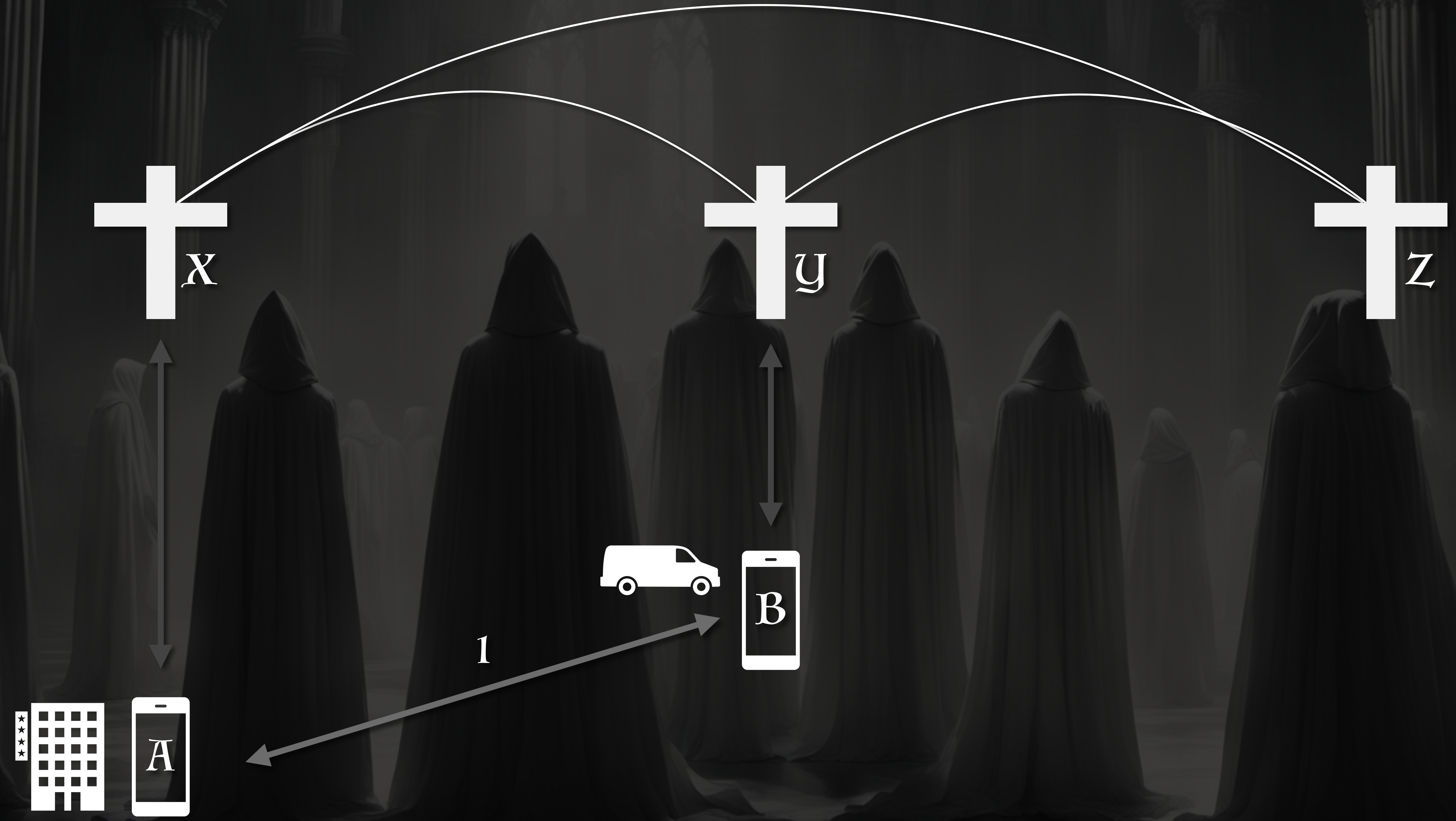
Confessions

- One-to-one calls.
  - Single tunnel to your peer.
- Group calls.
  - Establish unique tunnels to each participant in the group.
  - More resource demanding but works fine.
  - Don’t let anyone else claim differently.

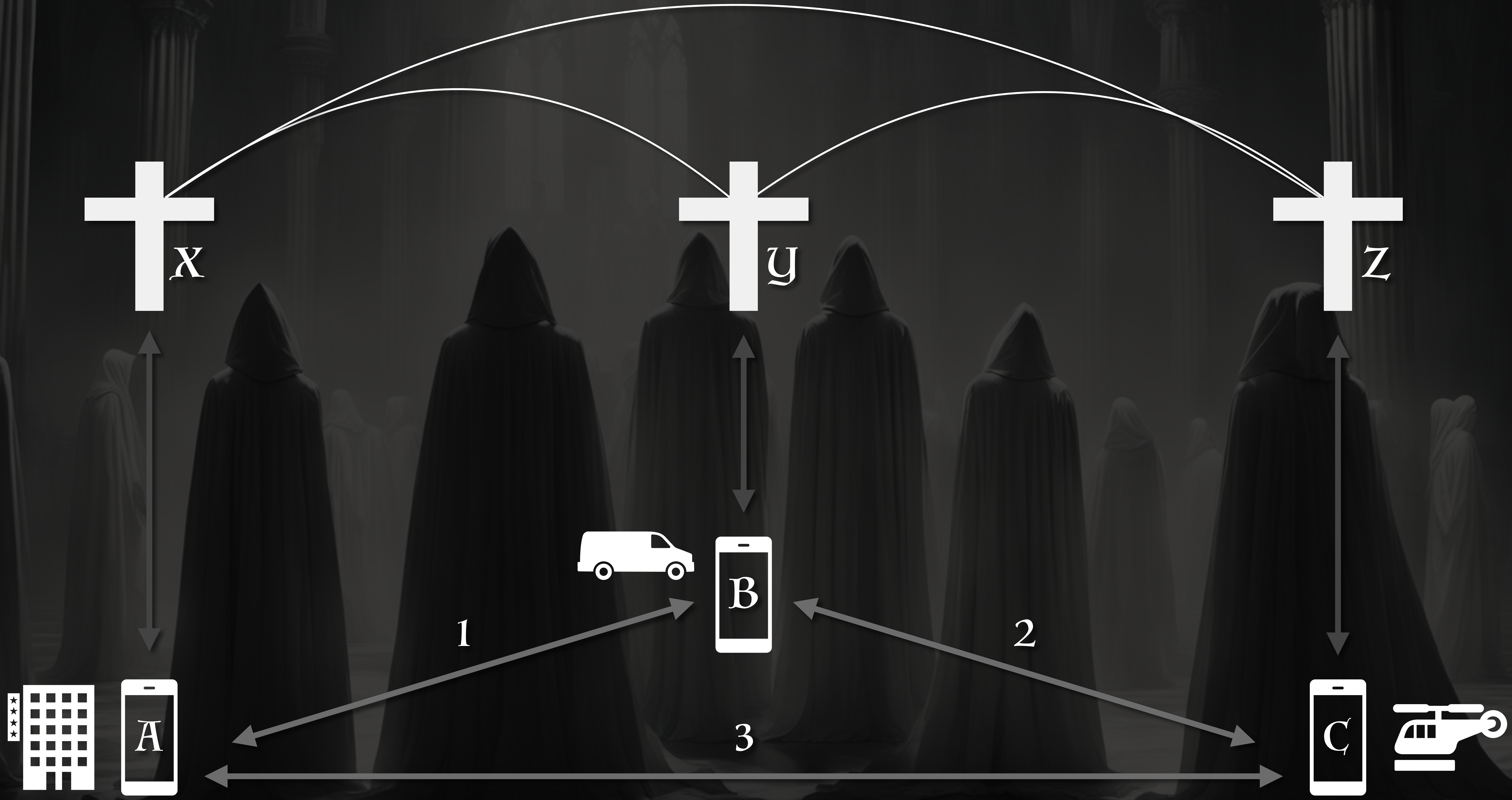
Confessions

- Group calls make use of liturgies to auto-discover peers.
- Peers can join and leave group automatically.
- The good, any peer in your flock can join the group.
- The bad, any peer in your flock can join the group.
- Depending on use-case this is either good or bad.

Confessions



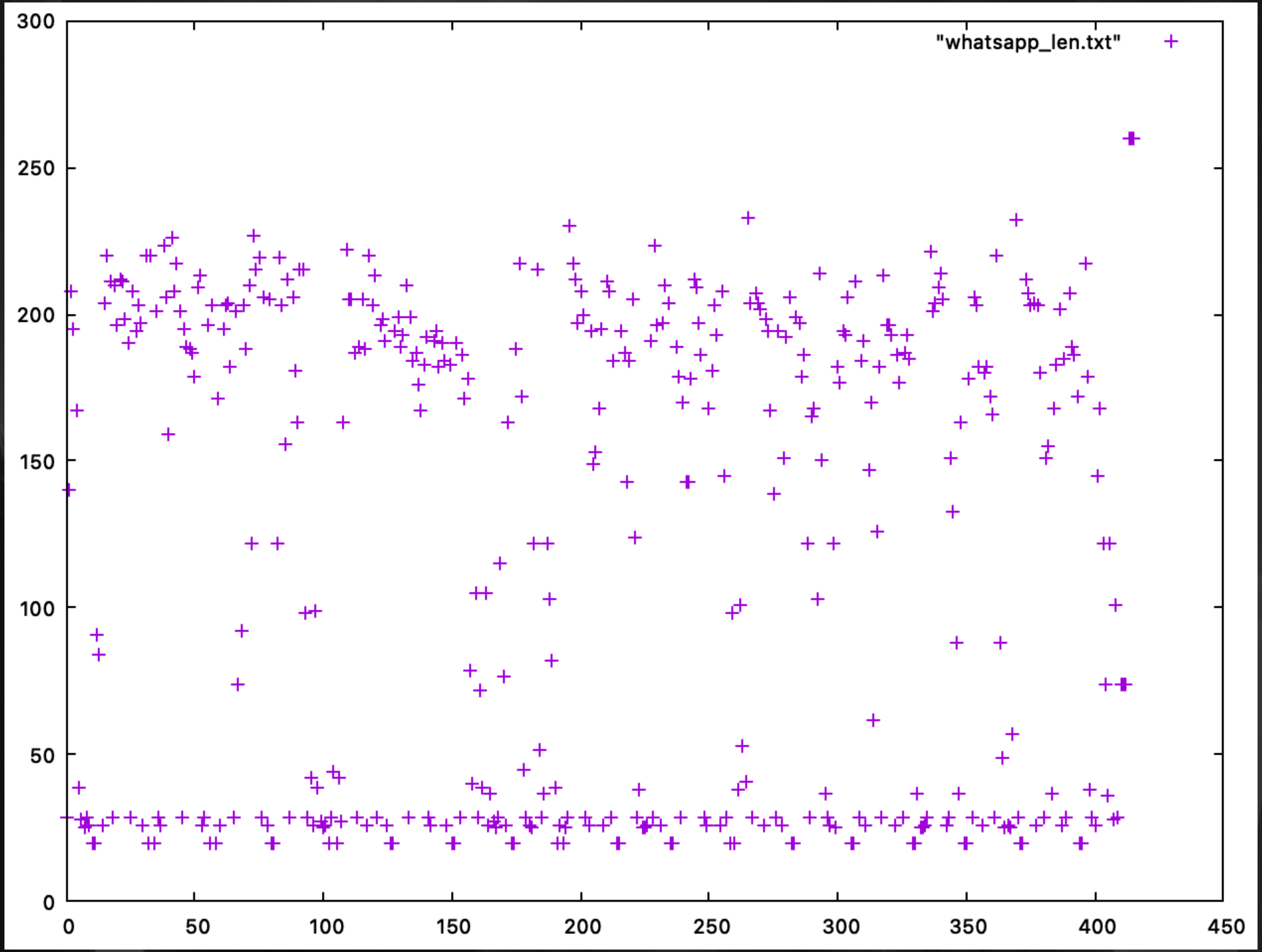
Confessions



Confessions

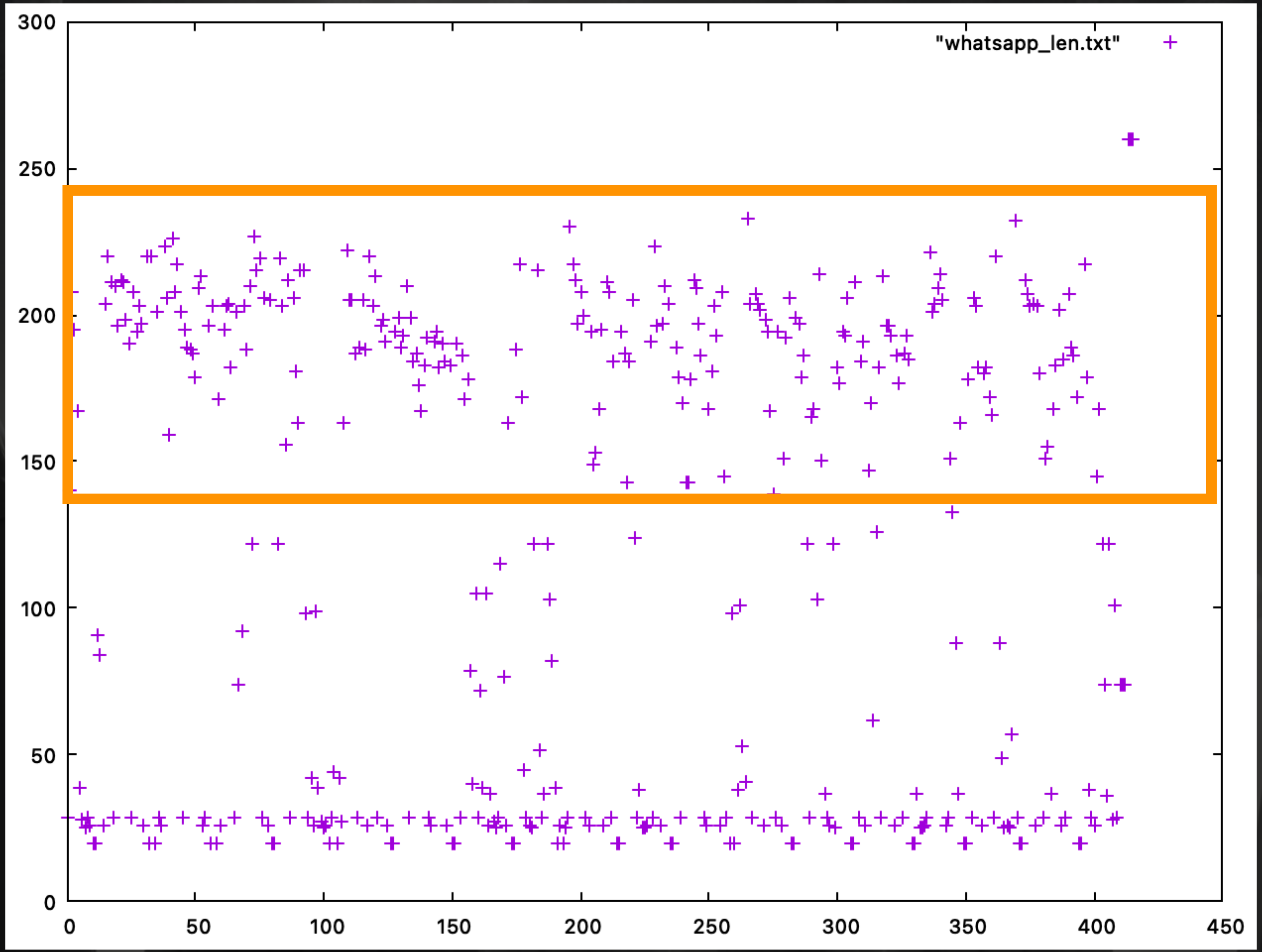
- Voice encoded using OPUS.
- VBR codecs (like OPUS) are problematic:
  - Length of encoded packets leak information.
- Papers from 2010 talks about being able to recover spoken words even when encrypted.
- All you need is a prerecording of a persons voice.
  - And a 2010 neural network (where are we today?)

Confessions



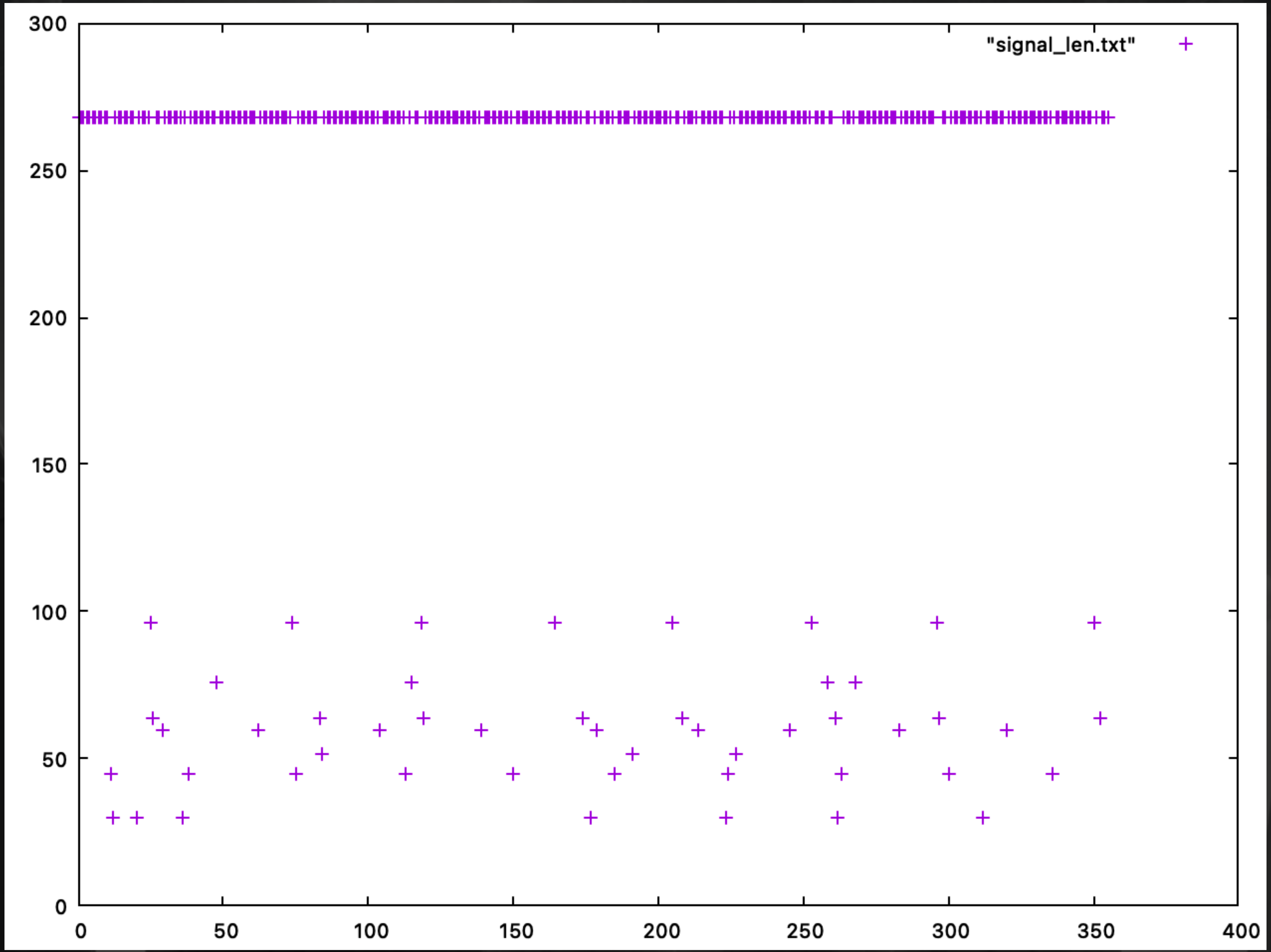
WhatsApp voice call

Confessions



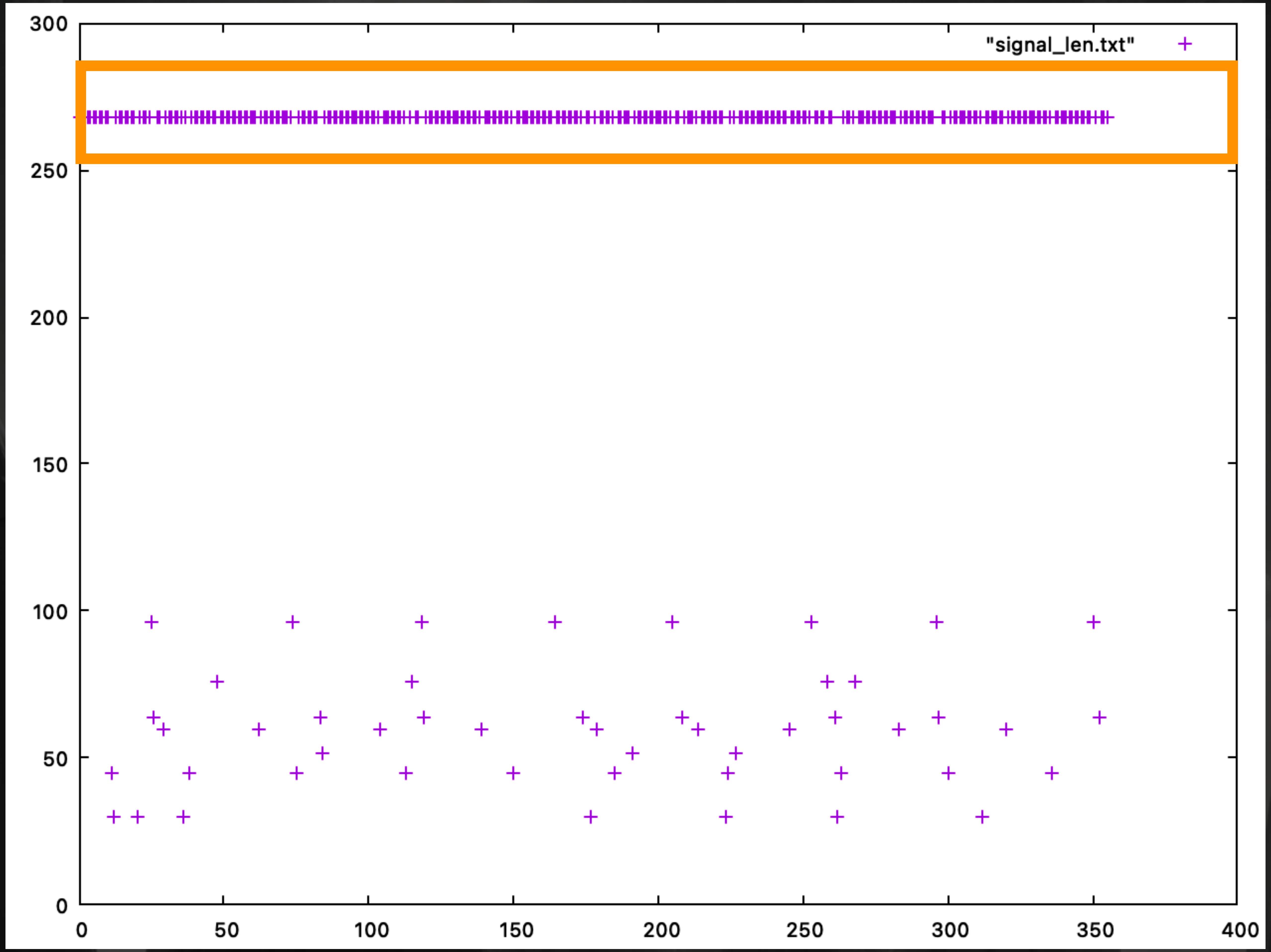
WhatsApp voice call

Confessions



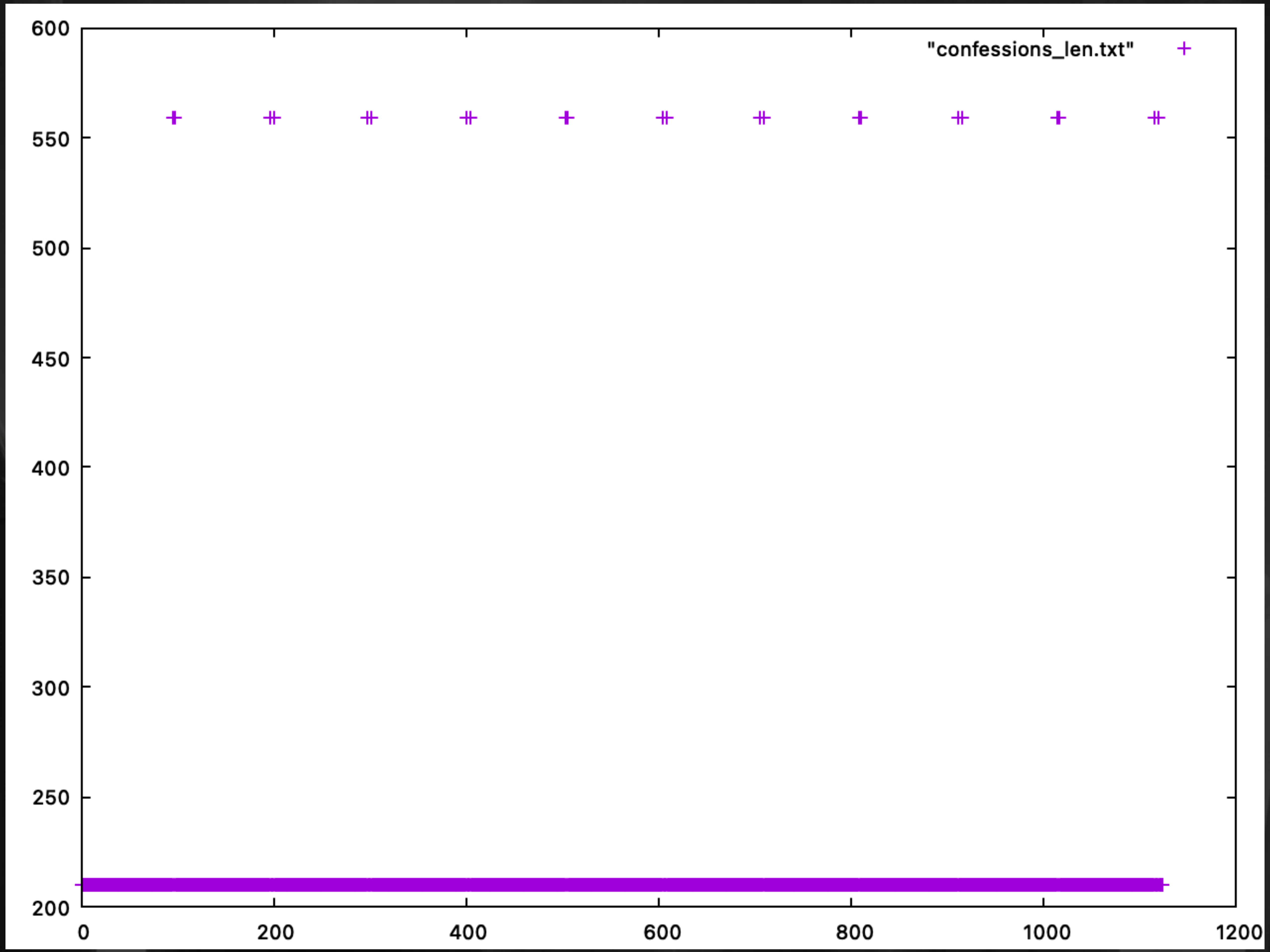
Signal voice call

Confessions



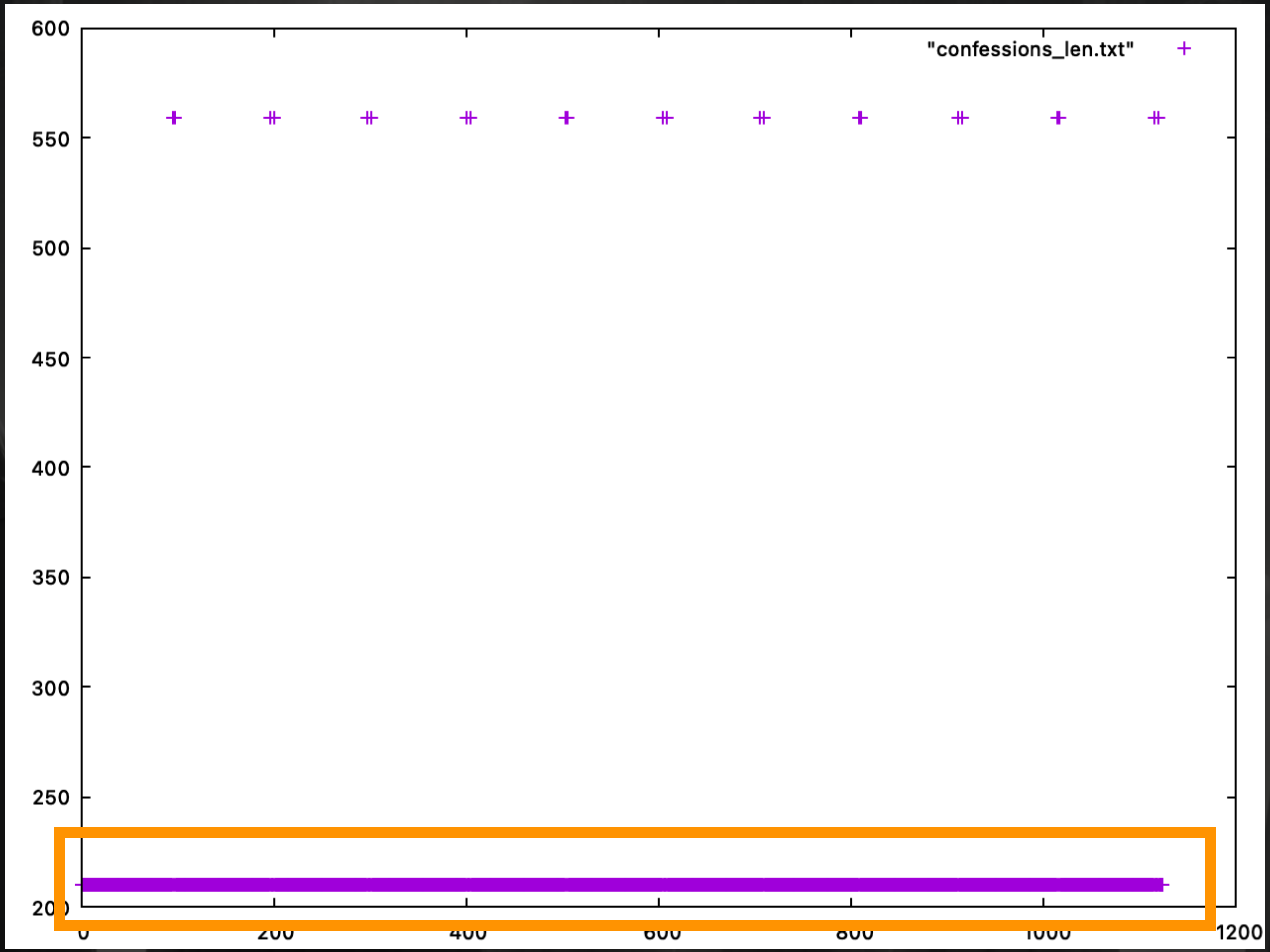
Signal voice call

Confessions



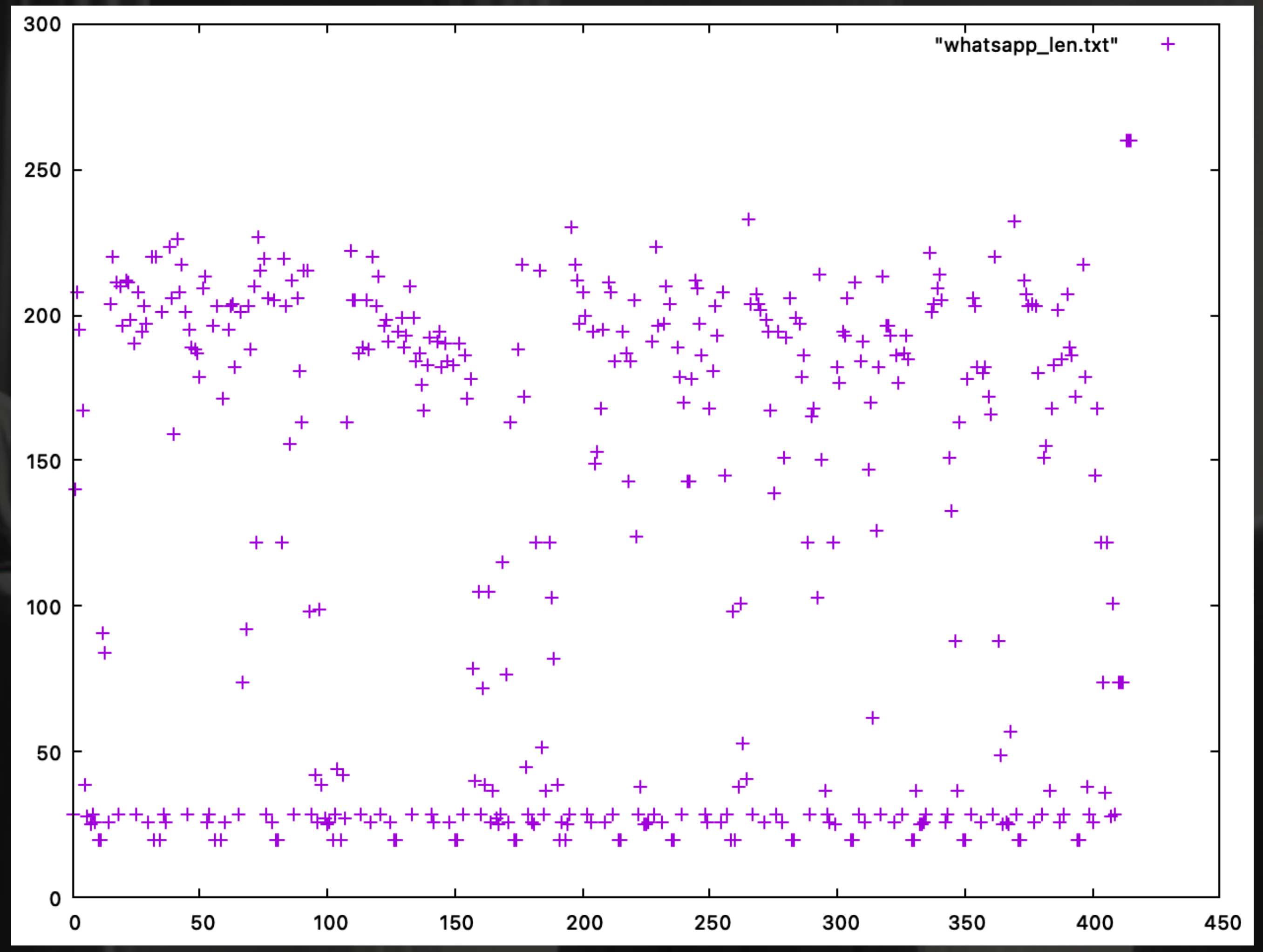
Confessions voice call

Confessions

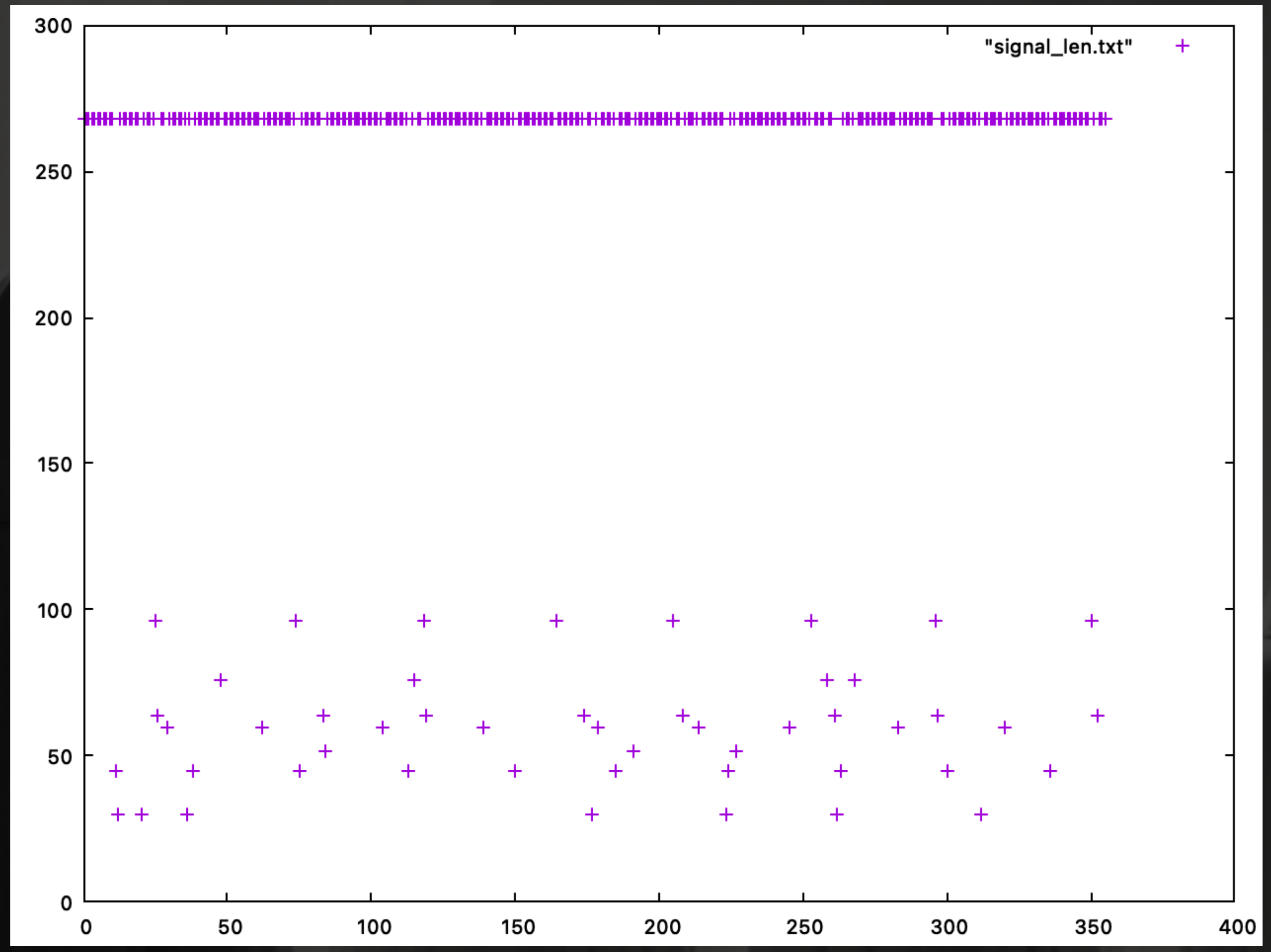


Confessions voice call

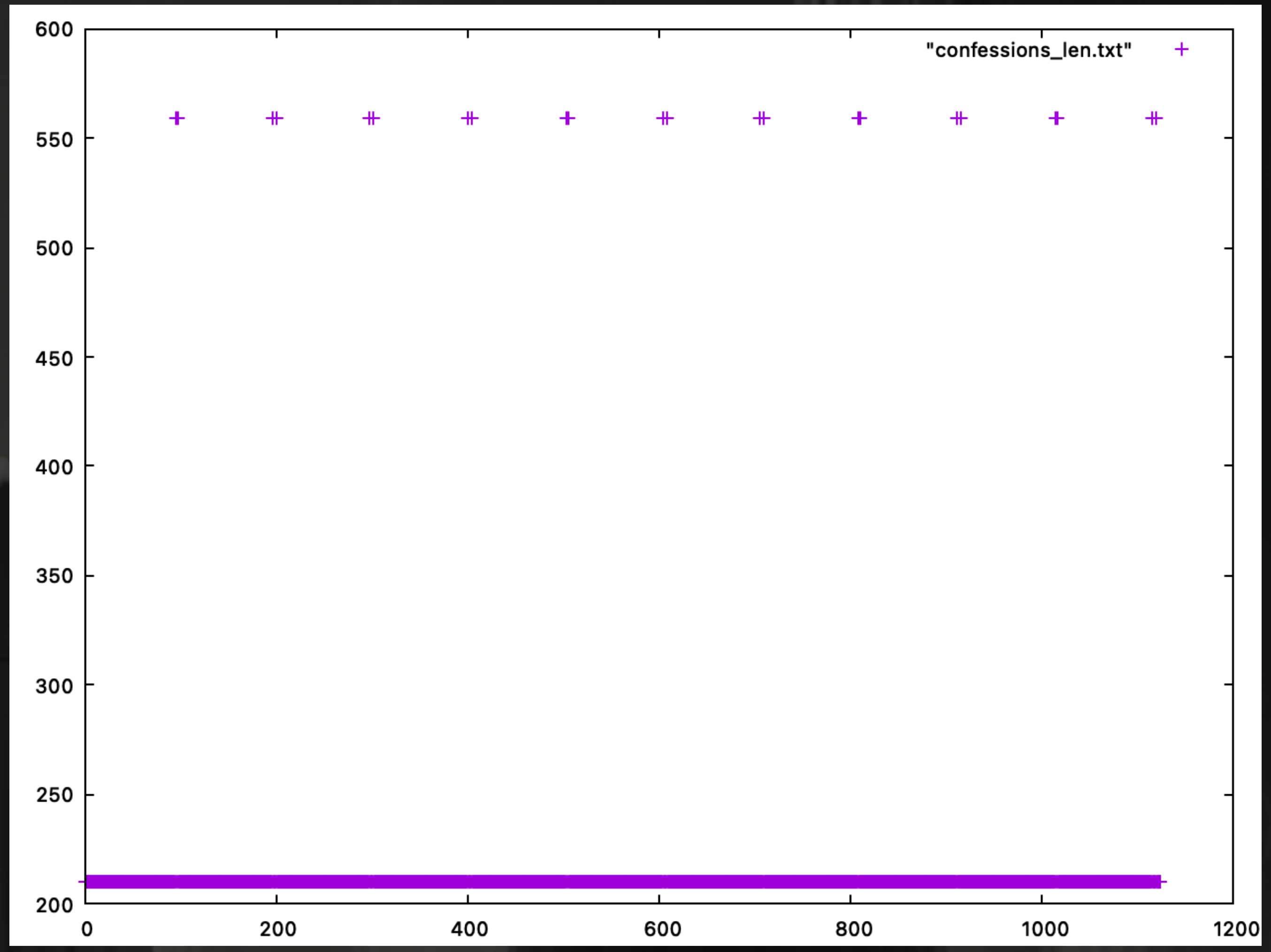
Confessions



WhatsApp



Signal



Confessions

Confessions

- Confessions uses fixed-size packets.
  - OPUS payload is carried inside fixed-size packets.
- DTX feature detects when someone talks.
  - If nobody is talking, packet sizes and frequency changes.
  - Also a side-channel, can distinguish when someone talks.
  - Confessions turns this off.

Confessions

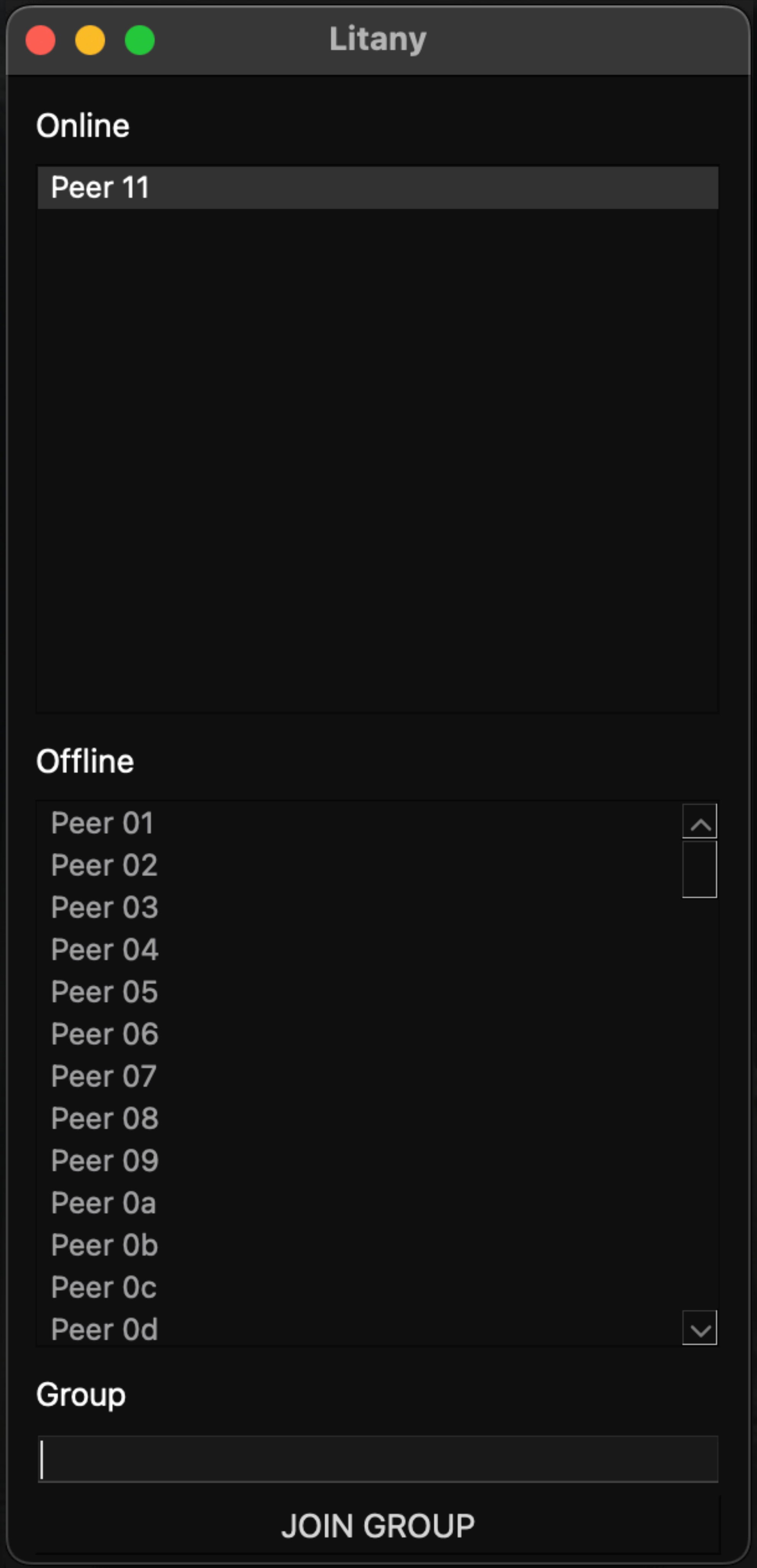
- Kind reminder that WebRTC uses OPUS by default.
- WebRTC is used everywhere for voice (browsers, apps, …)
- You must explicitly ask for CBR in SDP by setting cbr=1.
- If not present, VBR is used.
- How many WebRTC consumers do this you think?

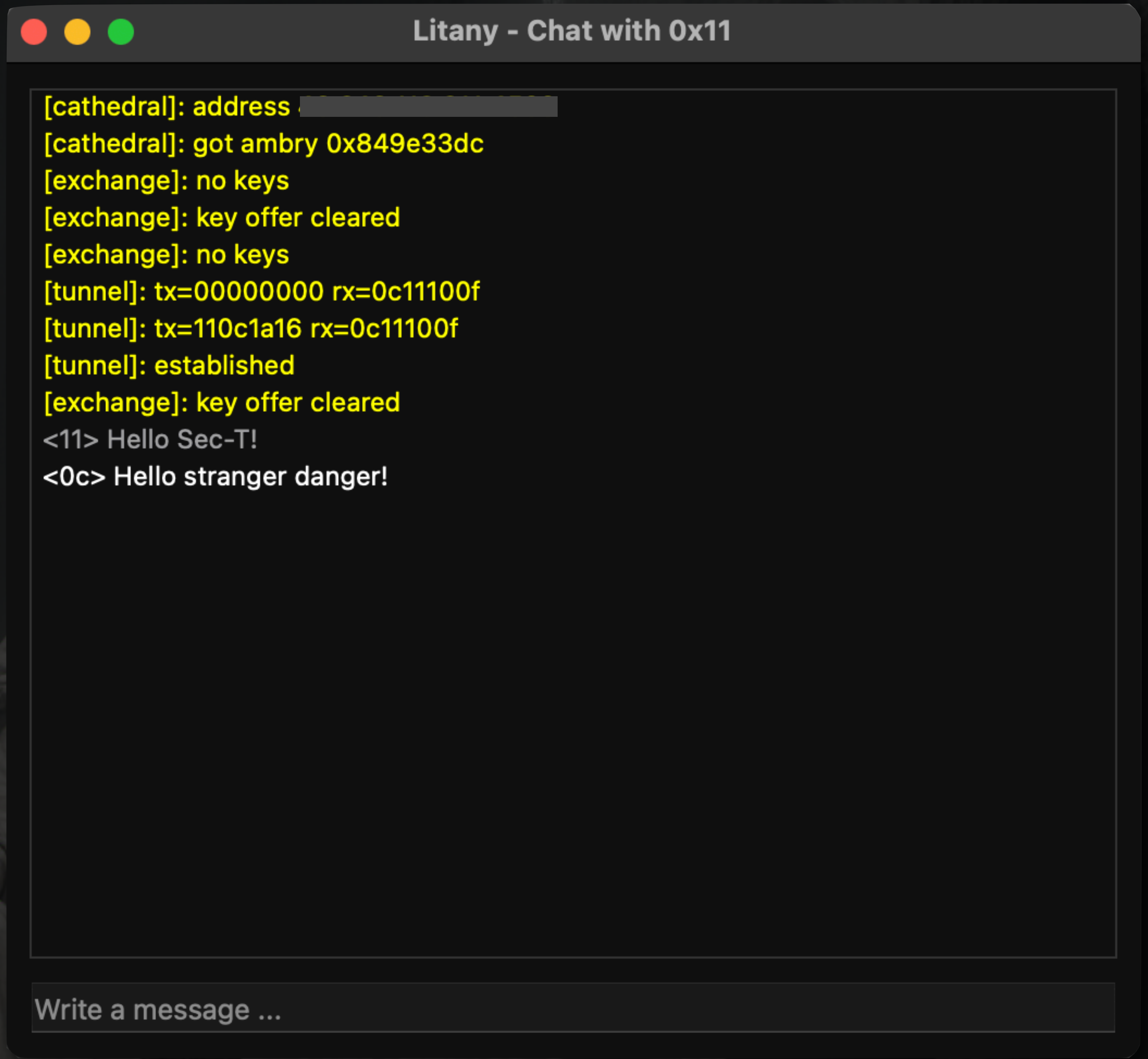
Litany

Litany

- Litany is a real-time chat program written in C++ using Qt.
  - Linux, OpenBSD, MacOS and Windows.
- Libkyrka is used to transport messages to peers.
- Text is carried instead of IP packets.

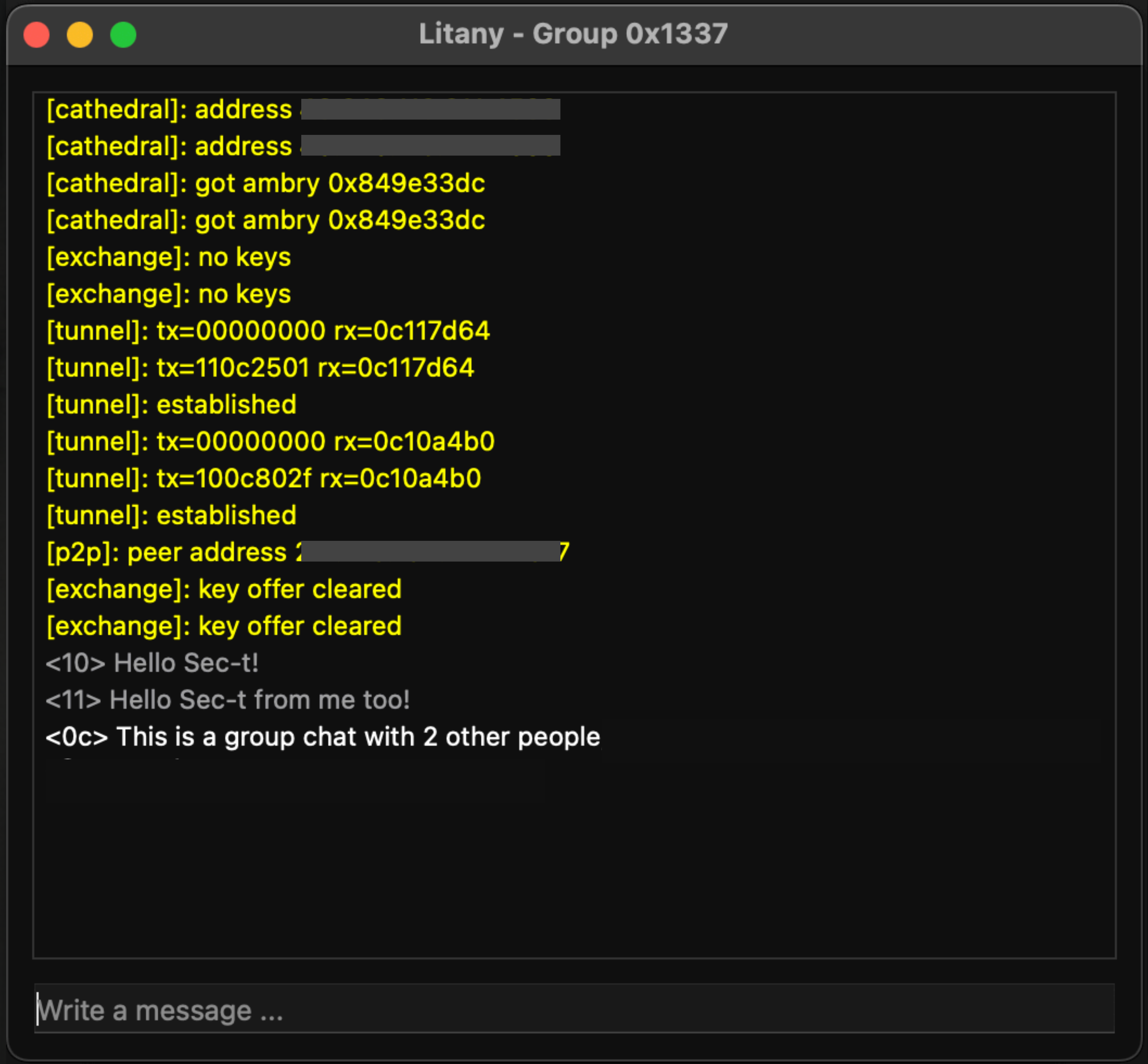
Litany





```
[cathedral]: address [redacted]
[cathedral]: got ambry 0x849e33dc
[exchange]: no keys
[exchange]: key offer cleared
[exchange]: no keys
[tunnel]: tx=00000000 rx=0c11100f
[tunnel]: tx=110c1a16 rx=0c11100f
[tunnel]: established
[exchange]: key offer cleared
<11> Hello Sec-T!
<0c> Hello stranger danger!
```

Litany



Litany - Group 0x1337

```
[cathedral]: address [redacted]
[cathedral]: address [redacted]
[cathedral]: got ambry 0x849e33dc
[cathedral]: got ambry 0x849e33dc
[exchange]: no keys
[exchange]: no keys
[tunnel]: tx=00000000 rx=0c117d64
[tunnel]: tx=110c2501 rx=0c117d64
[tunnel]: established
[tunnel]: tx=00000000 rx=0c10a4b0
[tunnel]: tx=100c802f rx=0c10a4b0
[tunnel]: established
[p2p]: peer address [redacted]7
[exchange]: key offer cleared
[exchange]: key offer cleared
<10> Hello Sec-t!
<11> Hello Sec-t from me too!
<0c> This is a group chat with 2 other people
```

Write a message ...

Litany

- One-on-one chats
  - Direct tunnel between peers.
- Group chats
  - One unique tunnel for each peer joined into the group.
  - No complex group key establishment protocol needed.

Litany

- Several different liturgies in different flock domains:
  - A discovery one to see who’s online.
  - A signalling one to indicate if someone wants to talk to you.
  - Another discovery for open group chats.

Litany

- Traffic analysis prevention:
  - All protocol messages are fixed-size.
  - Chat messages carried inside of these protocol messages.
- Reliable delivery of chat messages:
  - Chat messages must be ACK'd or they are periodically resent by sender.
- Quite noisy on the wire (on purpose).

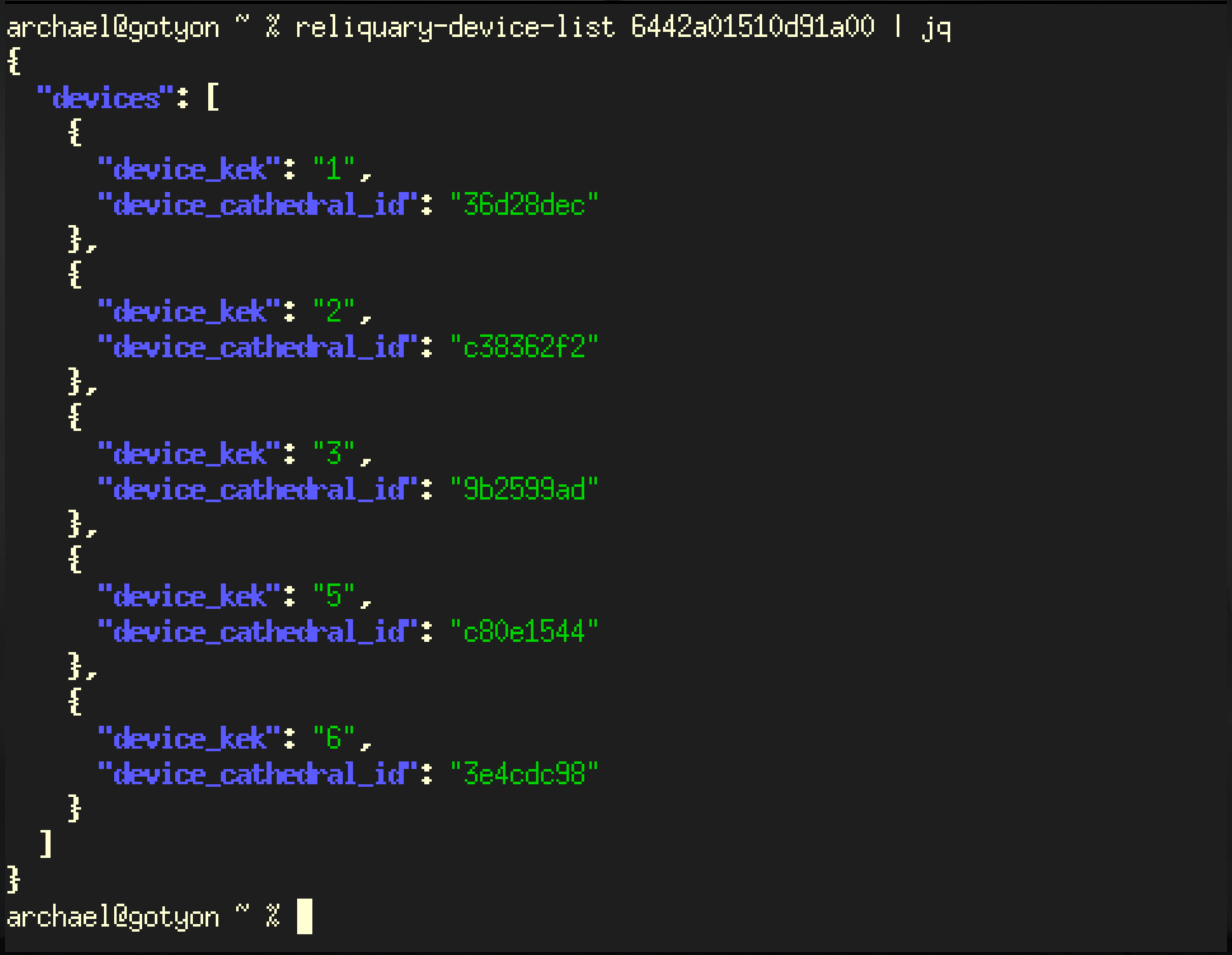
The Reliquary

SEC-T 2025

The Reliquary

- A community-driven cathedral network.
  - Multi-tenant
- Provides APIs and handy scripts:
  - Account management.
  - Create and manage flocks and devices.
  - Upload your Ambry bundles.
- Cathedrals spread over Europe.
- Relayed tunnels are capped at 25mbit/sec.

The Reliquary

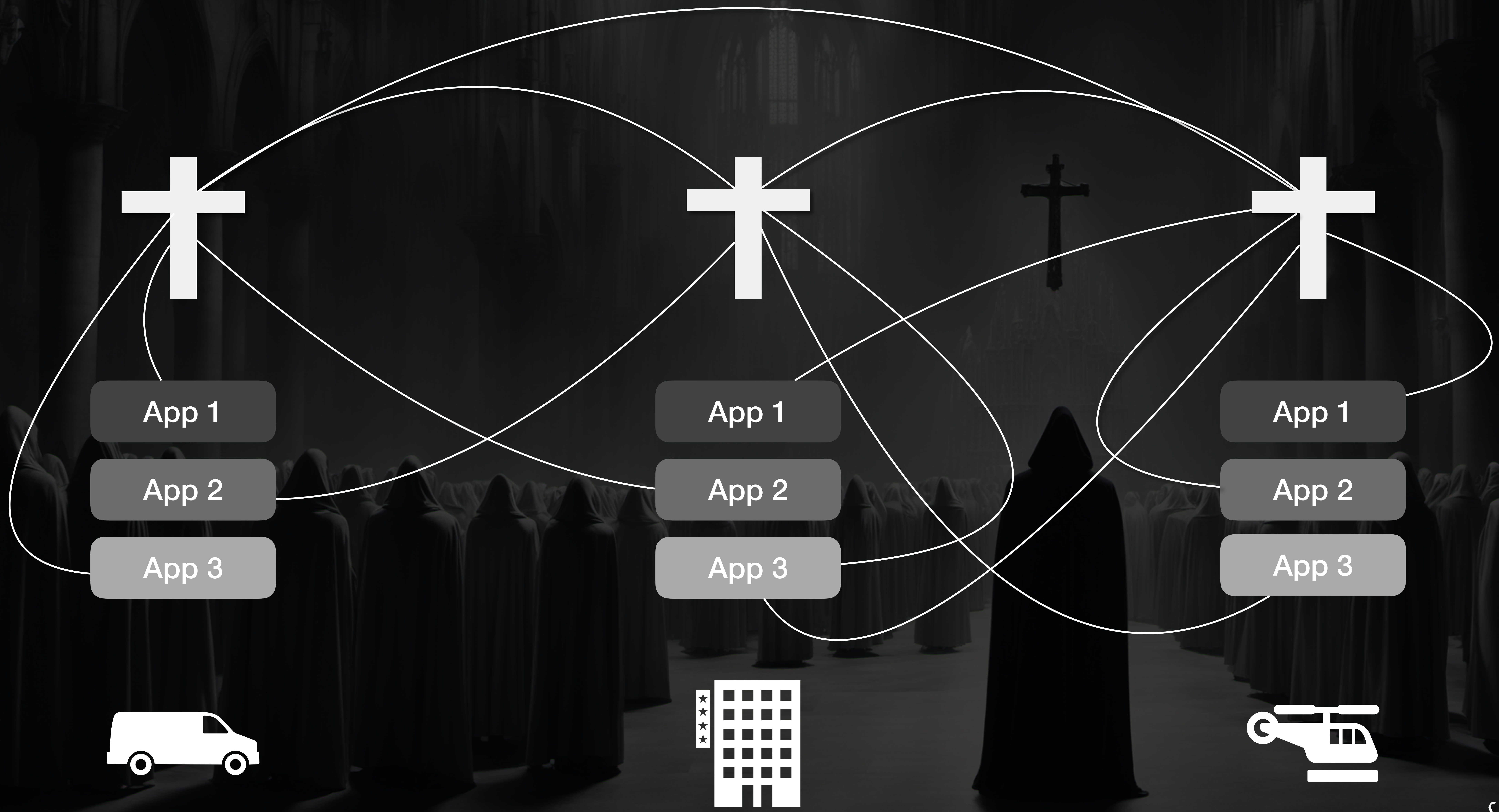


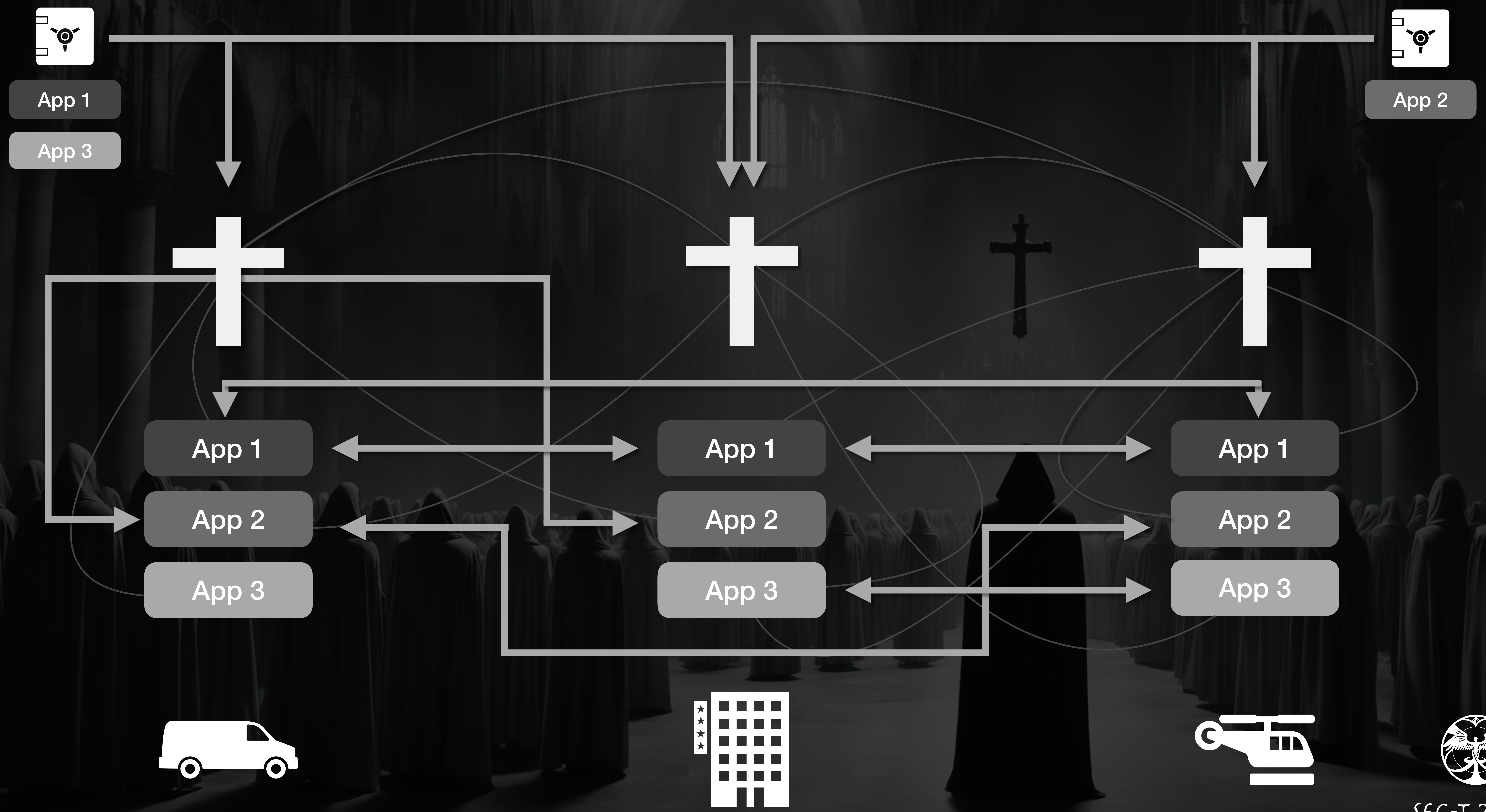
```
archael@gotyon ~ % reliquary-device-list 6442a01510d91a00 | jq
{
  "devices": [
    {
      "device_kek": "1",
      "device_cathedral_id": "36d28dec"
    },
    {
      "device_kek": "2",
      "device_cathedral_id": "c38362f2"
    },
    {
      "device_kek": "3",
      "device_cathedral_id": "9b2599ad"
    },
    {
      "device_kek": "5",
      "device_cathedral_id": "c80e1544"
    },
    {
      "device_kek": "6",
      "device_cathedral_id": "3e4cdc98"
    }
  ]
}
archael@gotyon ~ %
```

Recap

Recap

- Sanctum
  - Build highly available and truly distributed secure communication infrastructure using its cathedrals.
  - Secure key distribution.
- Libkyrka
  - Run any type of application on that infrastructure.
- Confessions and Litany
  - Voice and text without central nodes, only the cathedral infrastructure.


App 1
App 2
App 3
App 1
App 2
App 3
App 1
App 2
App 3


App 1
App 3
App 2
App 1
App 2
App 3
App 1
App 2
App 3
App 1
App 2
App 3

Recap

As long as we can communicate securely

Everything will be fine

https://conclave.se

Thank you for listening