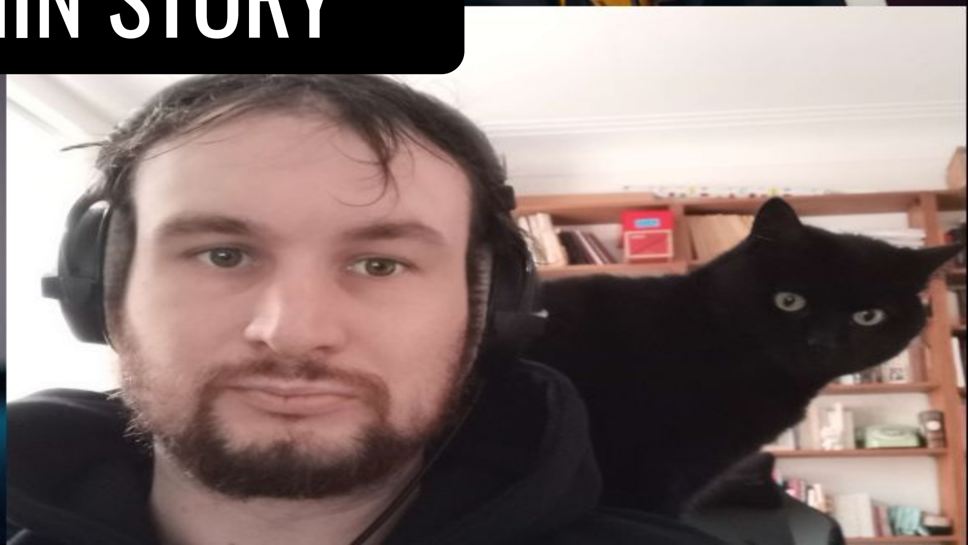
Cryptography vs AI

Deepfake resistant WebRTC video calls,
trustless P2P networks and other shenanigans

Yup, this is a clickbait title 🤷‍♂️

THE ORIGIN STORY



Project requirements

- A fully featured video+audio chat with minimal server requirements
- No centrally stored identity…
- …but a basic auth mechanism, to ensure genuine identity

SIGNALLING SERVER

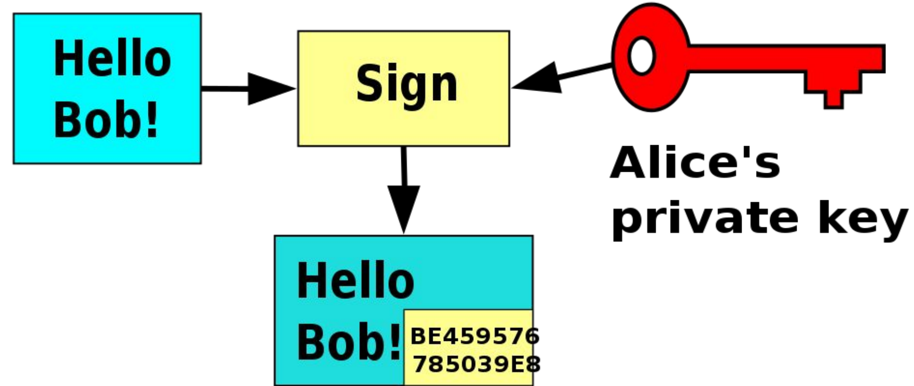
BUT CAN I TRUST THE PEERS?

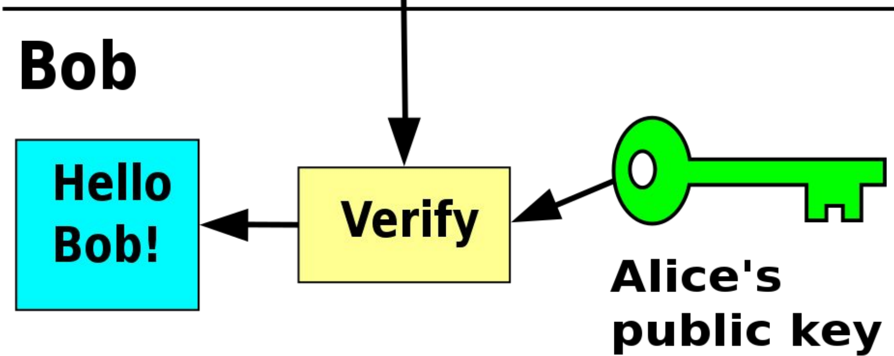
LET’S USE
A KEY!

Why tho?

- Digital signatures are everywhere and foolproof
- Keypairs are widespread, choose your poison:
  - FIDO/WebAuthn
  - Crypto wallets
  - Good ol’ `ssh-keygen -t ecdsa`

Alice





Beyond sign and verify*, you can even generate or derive keys from your browser !

*(that’s what this talk is about btw)

HERE COMES
THE PLAN

Basic overview

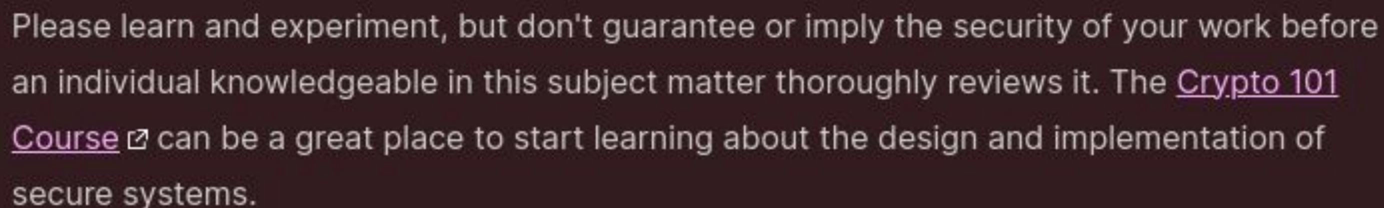
- A first client (= host) creates a new room on the signaling server (with a unique ID), by opening a websocket connection
- The host can send a special type of message on the socket which lets them whitelist a set of public keys
- Whenever a new peer wants to connect to the chat, the server asks the newcomer to sign a special payload containing the room ID, alongside a timestamp (to avoid replay attacks)
- If the signature matches the payload, and the public key is in the whitelist, let them reach the other peers
- Rinse and repeat

SECURITY
DISCLAIMER

⚠ **Warning:** This API provides a number of low-level cryptographic primitives. It's very easy to misuse them, and the pitfalls involved can be very subtle.

Even assuming you use the basic cryptographic functions correctly, secure key management and overall security system design are extremely hard to get right, and are generally the domain of specialist security experts.

Errors in security system design and implementation can make the security of the system completely ineffective.



Please learn and experiment, but don't guarantee or imply the security of your work before an individual knowledgeable in this subject matter thoroughly reviews it. The Crypto 101 Course can be a great place to start learning about the design and implementation of secure systems.

tl;dr: This project is for fun,
don’t screw up with key management

Cryptography is typically bypassed, not penetrated.

Adi Shamir

LET'S DIVE IN!

Importing a private key (client side)

```
const loadKey = async (data: BufferSource) => {
   const key = await crypto.subtle.importKey(
        'pkcs8',
        data,
        {
            name: 'ECDSA',
            namedCurve: 'P-256',
        },
        true,
        ['sign']
   )

   return key
}
```

PKCS8 is the standard format used by ssh-keygen

We’re using Elliptic Curve signature

We specify that we want to use this key for signing

Websocket handshake overview

```
type ServerPayloadType = {
    roomId: string,
    issuedAt: string,
}
```

This is the payload we're getting from the server.
issuedAt is a ISO-8601 timestamp

```
type SignedPayloadType = {
    payload: ServerPayloadType,
    signature: ArrayBuffer,
    publicKey: JsonWebKey,
}
```

This is the signed payload that we send back to the server

Signing a payload (client side)

```
const sign = async (
    key: CryptoKey,
    payload: ServerPayloadType
    ): Promise<SignedPayloadType > => {
    const signable = new TextEncoder().encode(
        JSON.stringify(payload)
    )
    const signature = await crypto.subtle.sign(
        {
            name: 'ECDSA',
            hash: { name: 'SHA-256' },
        },
        key,
        signable
    )

    return {
        payload,
        signature,
        publicKey: await crypto.subtle.exportKey('jwk', key)
    }
}
```

Verifying the payload (server side)

```
const verifyPayload = async (payload: SignedPayloadType): boolean => {
    const key = await crypto.subtle.importKey(
        'jwk', payload.publicKey, 'ECDSA', false, ['verify']
    )
    const signable = new TextEncoder().encode(
        JSON.stringify(payload.payload)
    )
    return crypto.subtle.verify(
        {
            name: 'ECDSA',
            hash: { name: 'SHA-256' },
        },
        key,
        payload.signature,
        signable
    )
}
```

A quick look at the signaling server (1/3)

```
socket.on('message', (message) => {
    const data = JSON.parse(message);

    if(data.type === 'request')
        return sendAuthPayload()
    if(data.type === 'auth')
        return connectPeer()
    // ...
});
```

A quick look at the signaling server (2/3)

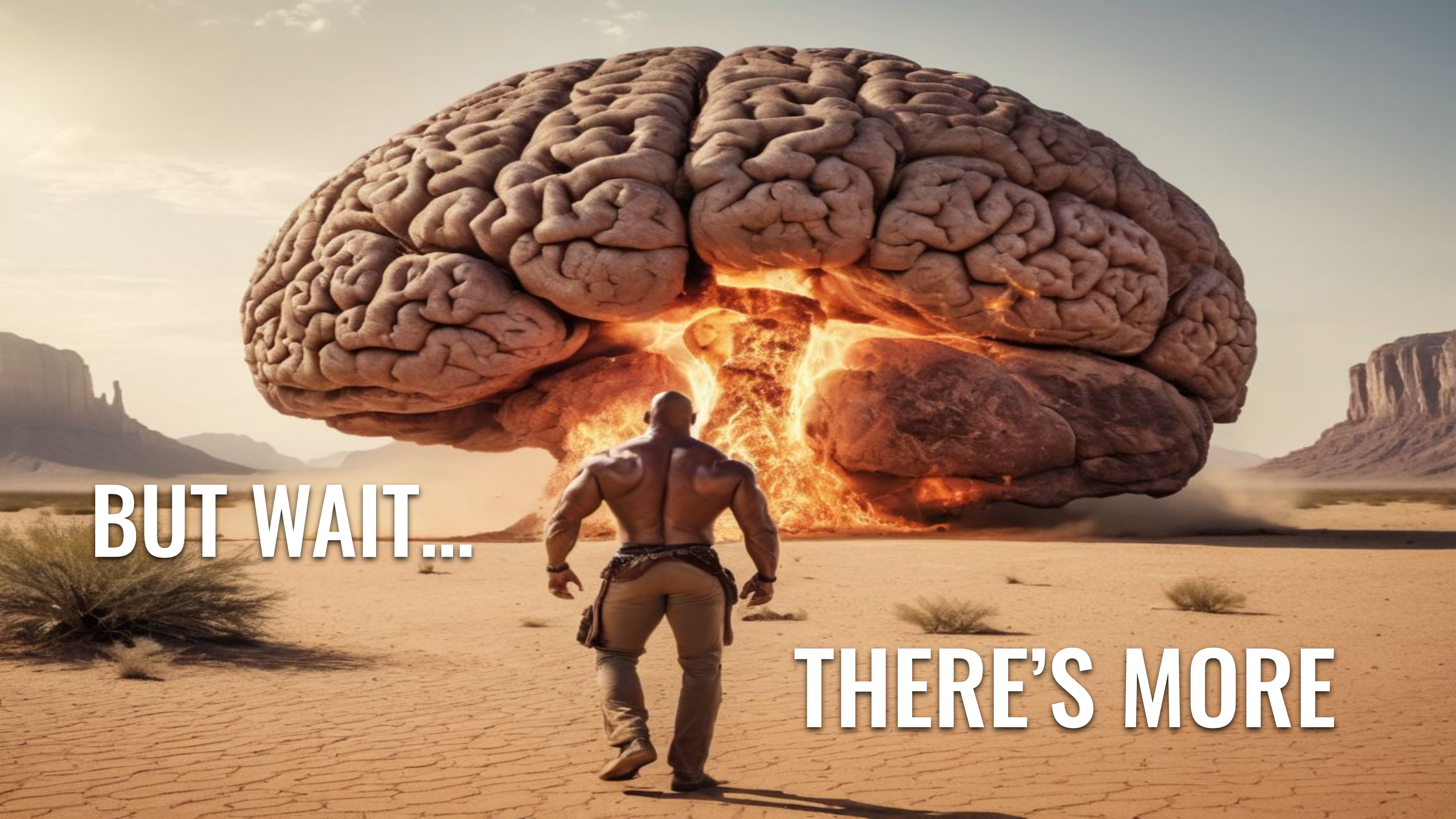
The connectPeer function looks if:

- The public key is valid for the roomId
- The issuedAt timestamp is still in an acceptable range
- The signature matches the payload and the public key

If all is good it broadcasts (= emits to all connected socket client except the one that sent the auth request) the peer information to all the other connected peers.

A quick look at the signaling server (3/3)

RTCPeerConnection objects are a pain in the bum to manage, @feross simple-peer simplifies the process of generating the objects.


BUT WAIT...
THERE'S MORE

Requesting identity checks from peer to peer

Now that we have established a link between client A and B, clients can create custom logic to revoke a peer connection.

e.g.: A “confirm your identity!” button that asks for a new signature of the peers from client A to client B.

Using subtleCrypto.encrypt() / .decrypt() on datachannels

WebRTC can also be used to stream arbitrary data between peers, using datachannels.
The subtleCrypto API also features a set of helpers around asymmetric encryption/decryption methods.

e.g.: One peer could decide to send their public encryption key (ex: RSA) to the others as to create a seamless end-to-end encrypted stream.
(short reminder that you should not use the same keypair for encryption and signing 😉)

This is your call to go further down the p2p rabbit hole…

THEY ALREADY
TRUST
THE PROCESS

Webtorrent

WebTorrent is a streaming torrent client for node.js and the browser. In the browser, WebTorrent uses WebRTC (data channels) for peer-to-peer transport.

IPFS

The InterPlanetary File System (IPFS) is a protocol, hypermedia and file sharing peer-to-peer network for storing and sharing data in a distributed file system. IPFS uses content-addressing to uniquely identify each file in a global namespace connecting IPFS hosts.

aleph.im

aleph.im is an open-source peer-to-peer network and decentralized cloud computing solution built on top of IPFS.

HELPDESK
SESSION

@BjrInt

@BonjourInternet

ifunny.co