MAP OF ZK in 2024

Payments
ZCASH
firo
Daimo
payy
IRON FISH

L1s
MINA
PENUMBRA
Aleo
HORIZEN
ALEPH ZERO
anoma

Hardware
Irreducible
INGONYAMA
SUPRA NATIONAL
CYSIC
FABRIC

L2s
Aztec
polygon
ZKsync
Linea
taiko
STARKNET
Scroll
MANTA

DeFi
RENEGADE
PANTHER
Dark.fi
PENUMBRA
offshift

ZKML/AI
EZKL
NOYA
Modulus Labs
ZkAGI
ZKML

Coprocessor
AXIOM
BREVIS
HERODOTUS
marlin
lagrange
RITUAL
vlayer

Gaming
dark forest
Immutable
BLADE
ZYPHER NETWORK
ZORDLE
zkHoldem
TileVille
cartridge

Verifiable Compute
Jolt
RISC ZERO
NEXUS
Succinct
NovaNet

Cross-chain
Hyperlane
zeko
=nil;
polygon
Succinct
Union

ID/Wallets
Privado.iD
semaphore
rarimo
WORLDCOIN
OpenPassport
Pass
verida
zkLogin
Sui
demox labs
Aptos Keyless

ZKTLS
PADO
PLUTO
tlsn
ZK EMAIL

Prover Network
GEVULOT
Succinct
FERMAH
lagrange
ZKPOOL
RISC ZERO

Proof Verif.
NEBRA
ALIGNED
HYLE
zkVerify
π^2
Soundness Layer

R&D/Audit
0xPARC
geometry RESEARCH
reilabs
Zellic
ZKS
NETHERMIND
LAMBDA CLASS
ZK HACK
Veridise

ZK in BTC
Alpen
BVM
Citrea

Misc/Tools
Nouns DAO
TACEO
Anonymous DAO
Delphinus Lab
Snarkify
CLIQUE
Cursive
powdr
Argument Computer Corporation
RAILGUN

Source: ZK Podcast

MAP OF ZK in 2024

Payments
ZCASH
firo
Daimo
payy
IRON FISH

L1s
MINA
Aleo
PENUMBRA
HORIZEN
ALEPH ZERO
anoma

Hardware
Irreducible
INGONYAMA
SUPRA NATIONAL
CYSIC
FABRIC

L2s
Aztec
polygon
ZKsync
Linea
taiko
STARKNET
Scroll
MANTA

DeFi
RENEGADE
PANTHER
DarkFi
PENUMBRA
offshift

ZKML/AI
EZKL
NOYA
Modulus Labs
ZkAGI
ZKML

Coprocessor
AXIOM
BREVIS
HERODOTUS
marlin
lagrange
RITUAL
vlayer

Gaming
dark forest
Immutable
BLADE
ZYPHER NETWORK
ZORDLE
zkHoldem
TileVille
cartridge

Verifiable Compute
Jolt
RISC ZERO
NEXUS
μ
Succinct
NovaNet

Cross-chain
Hyperlane
zeko
=nil;
polygon
Succinct
Union

ID
zkLogin
Sui

ZKTLS
PADO
PLUTO
tlsn
ZK EMAIL

Prover Network
GEVULOT
Succinct
FERMAH
lagrange
ZKPOOL
RISC ZERO

Proof Verif.
NEBRA
ALIGNED
HYLE
zkVerify
π^2
Soundness Layer

R&D/Audit
0xPARC
geometry RESEARCH
reilabs
Zellic
ZKS
NETHERMIND
LAMBDA
ZK HACK
Veridise

ZK in BTC
Alpen
BitVM
Citrea

Misc/Tools
Nouns DAO
TACEO
Anonymous DAO
Snarkify
Delphinus Lab
CLIQUE
Cursive
powdr
Argument Computer Corporation
RAILGUN

Source: ZK Podcast

MystenLabs Sui COSIC Soundness Labs

zkLogin: Onboarding the next billion users to web3

Mahdi Sedaghat

Jointly with Foteini Baldimitsi | Kostas Chalkias | Yan Ji | Jonas Lindstrøm | Deepak Maram | Ben Riva | Arnab Roy | Joy Wang

To Appear at ACM CCS'24.

Presented at SBC'24.

There are around

100 million

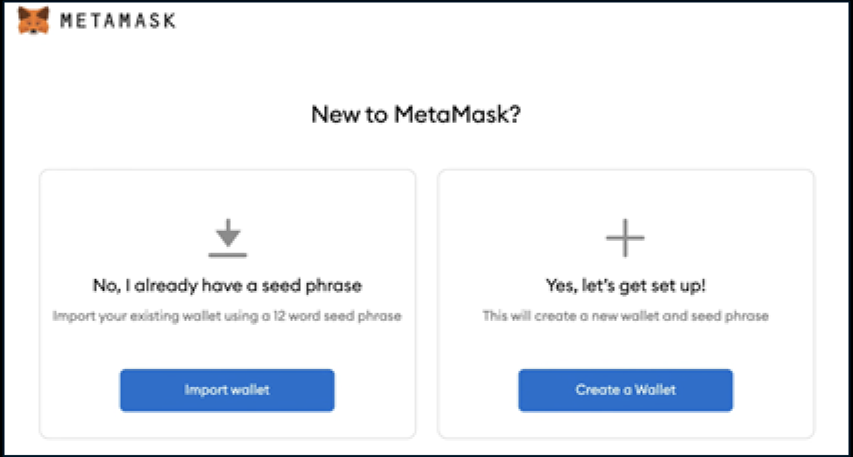
active crypto wallets

and there are several

BILLIONS

of web2 accounts

Web3 has an onboarding problem


METAMASK

New to MetaMask?

No, I already have a seed phrase

Import your existing wallet using a 12 word seed phrase

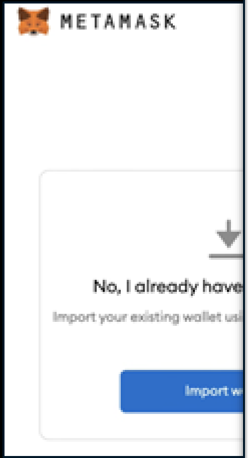
Import wallet

Yes, let's get set up!

This will create a new wallet and seed phrase

Create a Wallet

Web3 has an onboarding problem

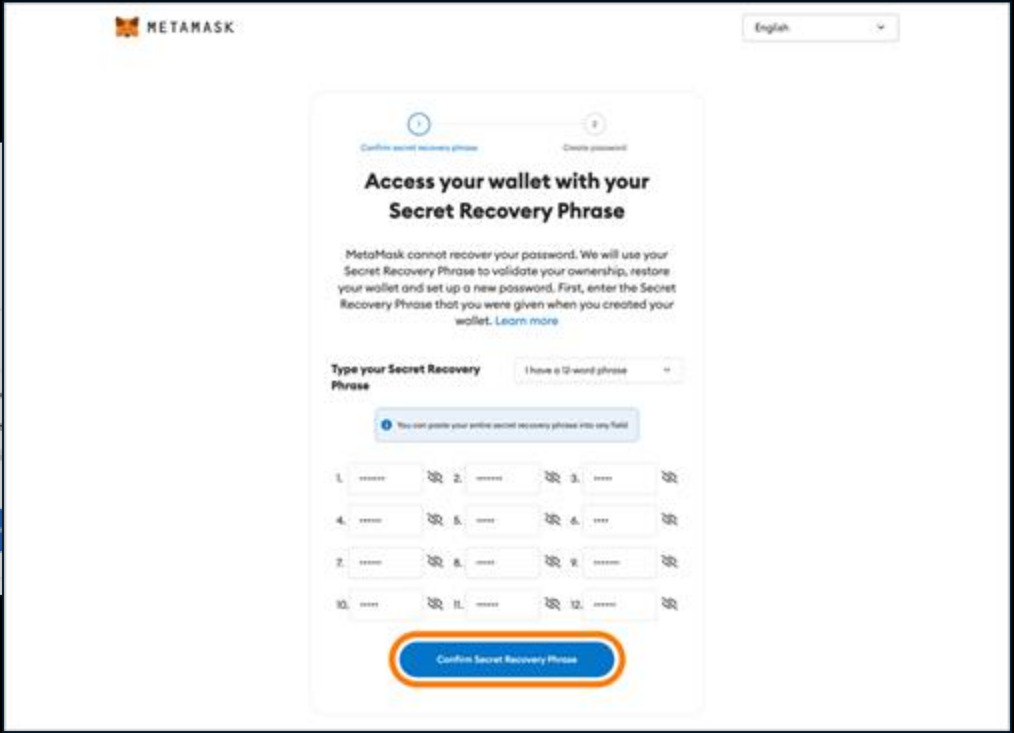


METAMASK

No, I already have

Import your existing wallet usi

Import w



METAMASK

English

Confirm secret recovery phrase

Create password

Access your wallet with your Secret Recovery Phrase

MetaMask cannot recover your password. We will use your Secret Recovery Phrase to validate your ownership, restore your wallet and set up a new password. First, enter the Secret Recovery Phrase that you were given when you created your wallet. Learn more

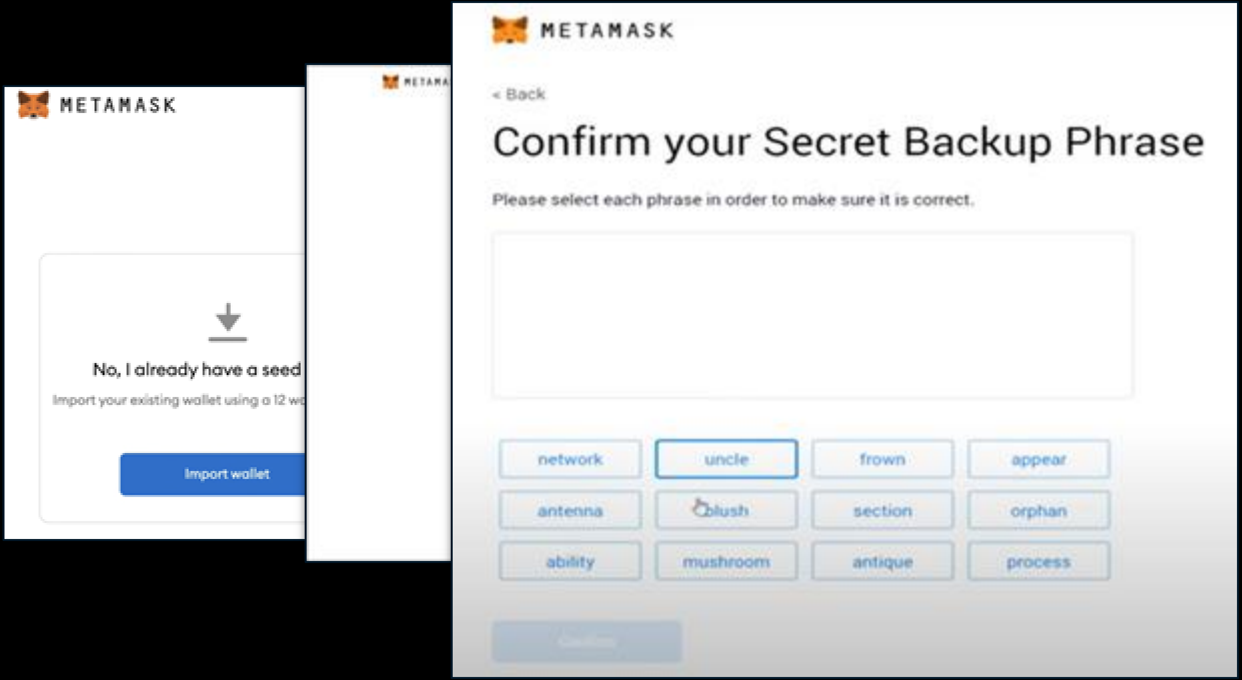
Type your Secret Recovery Phrase: I have a 12-word phrase

You can paste your entire secret recovery phrase into any field

1. •••••••	2. •••••••	3. •••••
4. ••••••	5. •••••	6. ••••
7. ••••••	8. •••••	9. •••••••
10. •••••	11. ••••••	12. ••••••

Confirm Secret Recovery Phrase

Web3 has an onboarding problem



METAMASK

No, I already have a seed

Import your existing wallet using a 12 wo

METAMASK

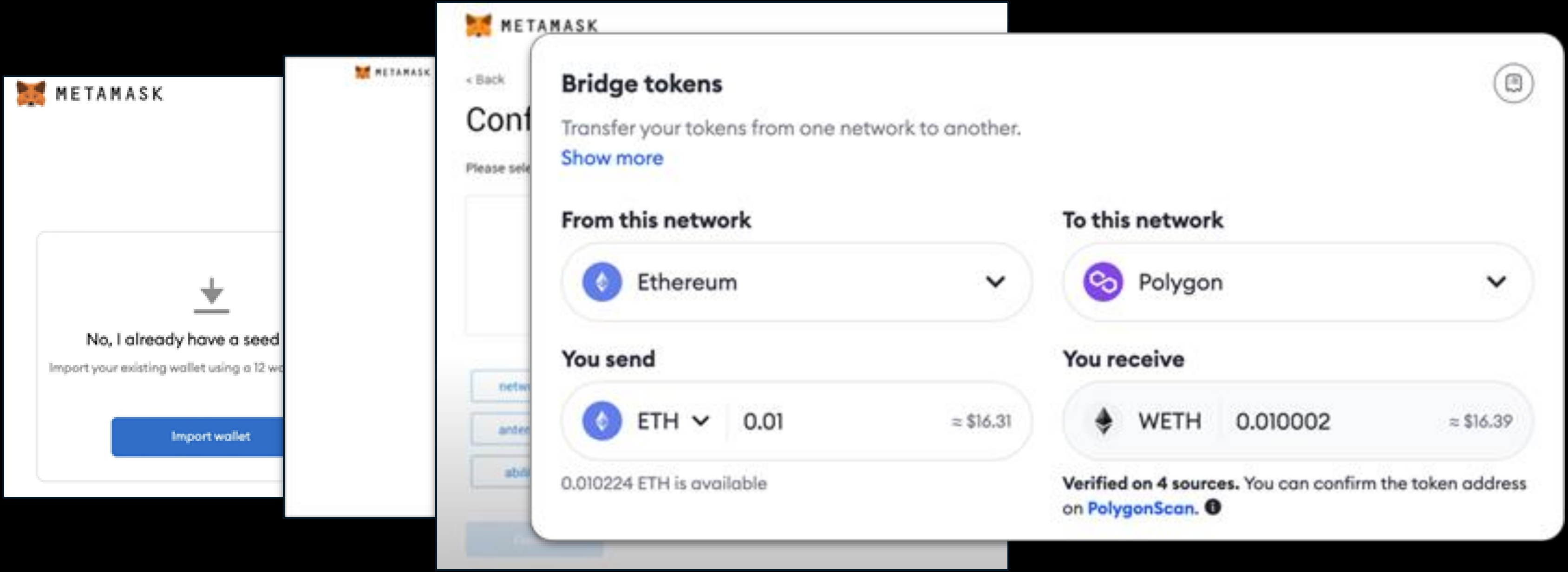
< Back

Confirm your Secret Backup Phrase

Please select each phrase in order to make sure it is correct.

network	uncle	frown	appear
antenna	blush	section	orphan
ability	mushroom	antique	process

Web3 has an onboarding problem


METAMASK

No, I already have a seed

Import your existing wallet using a 12 wo

Import wallet

METAMASK

METAMASK

< Back

Conf

Please sele

Bridge tokens

Transfer your tokens from one network to another.

Show more

From this network: Ethereum

To this network: Polygon

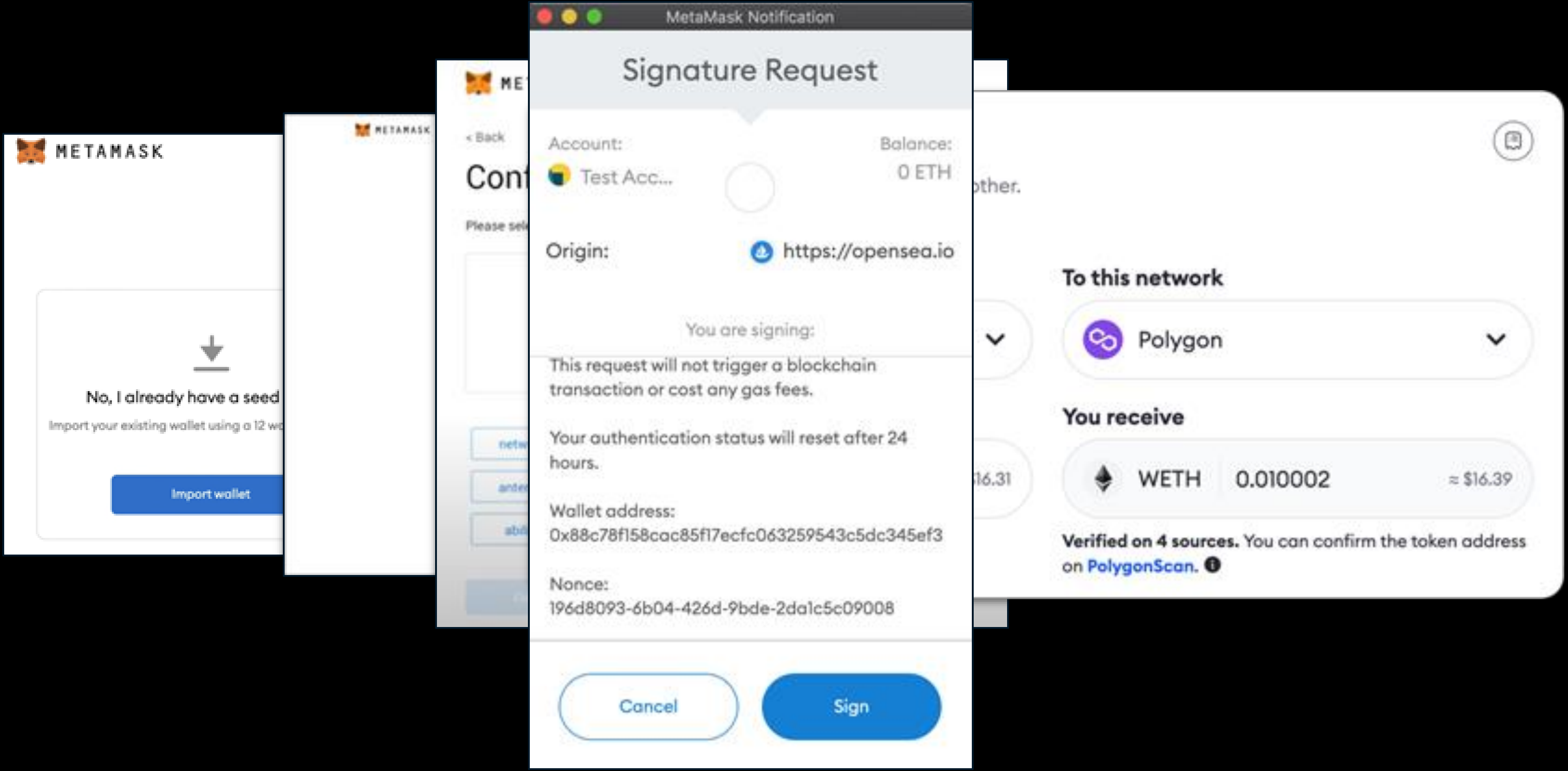
You send: ETH 0.01 ≈ $16.31

You receive: WETH 0.010002 ≈ $16.39

0.010224 ETH is available

Verified on 4 sources. You can confirm the token address on PolygonScan.

Web3 has an onboarding problem


METAMASK
No, I already have a seed
Import your existing wallet using a 12 w
Import wallet
METAMASK
ME
< Back
Conf
Please sel
netw
ante
abil

MetaMask Notification

Signature Request

Account: Test Acc...

Balance: 0 ETH

Origin: https://opensea.io

You are signing:

This request will not trigger a blockchain transaction or cost any gas fees.

Your authentication status will reset after 24 hours.

Wallet address:
0x88c78f158cac85f17ecfc063259543c5dc345ef3

Nonce:
196d8093-6b04-426d-9bde-2da1c5c09008

To this network: Polygon

You receive: WETH 0.010002 ≈ $16.39

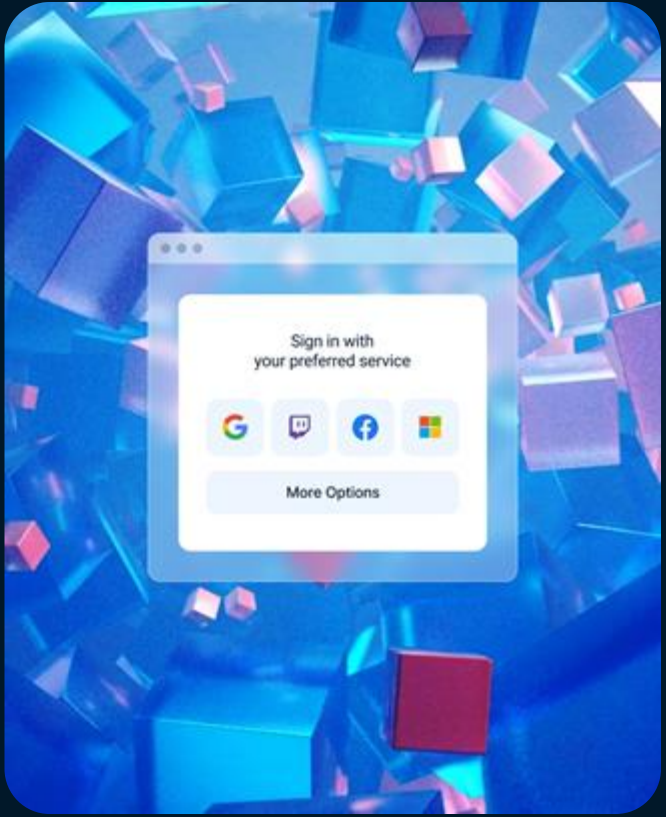
Verified on 4 sources. You can confirm the token address on PolygonScan.

Mnemonics and keys are not going to get us mass adoption.

Complexity is the killer of adoption.
The ultimate killer dApp for blockchain, is accessibility.

Can we make it as easy as signing in with Google, Facebook and co?

- People don’t want to use separate passwords for each and every app, each and every web2 service
- Extremely likely they already have a Google, Facebook, Amazon account
- Solution: use OAuth to leverage these already existing accounts



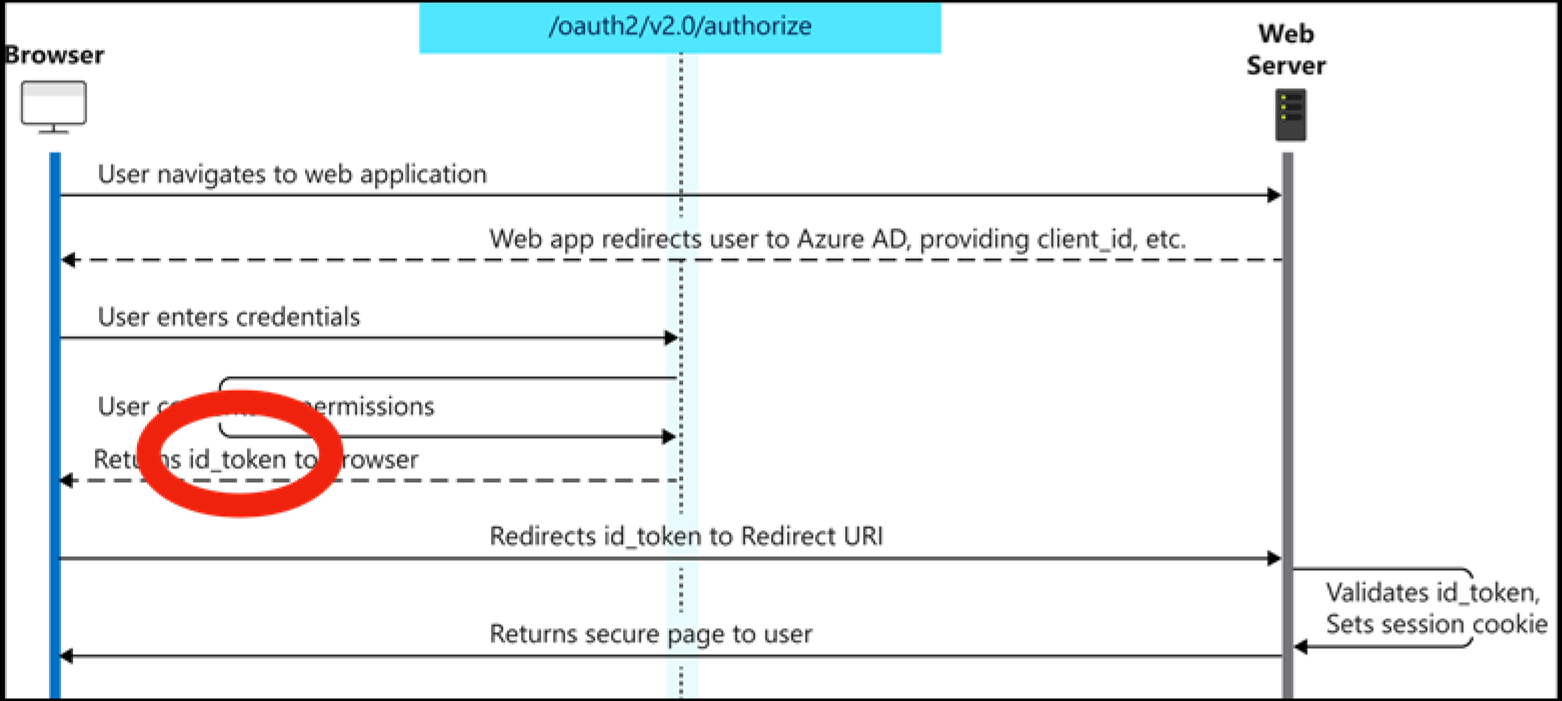
zkLogin:
OAuth + Zero Knowledge Proof

Non-custodial

User-friendly

Privacy-preserving

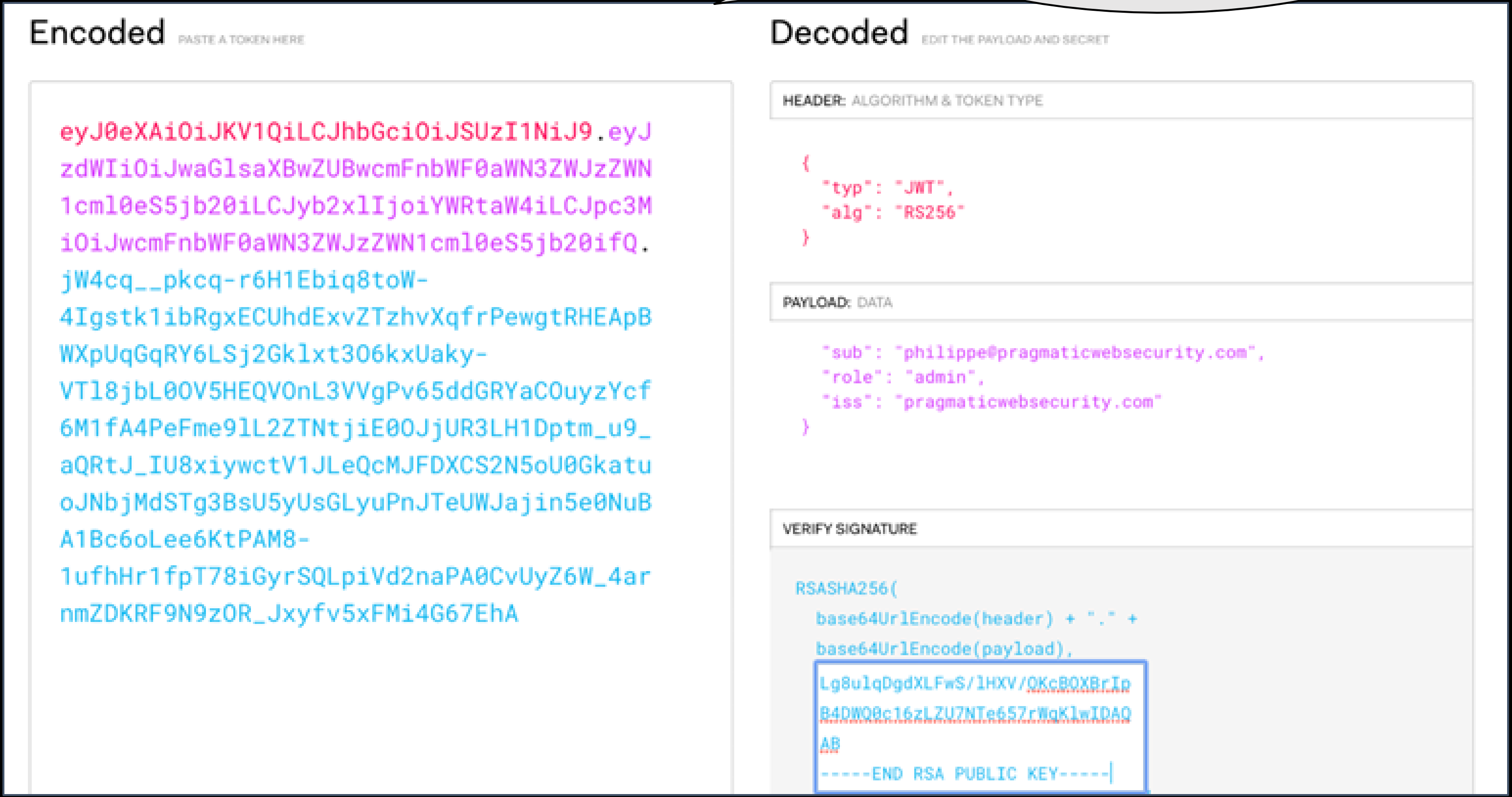
OpenID Connect (an extension of OAuth 2.0)


/oauth2/v2.0/authorize
Browser
Web Server
User navigates to web application
Web app redirects user to Azure AD, providing client_id, etc.
User enters credentials
User c ermissions
Ret ns id_token to rowser
Redirects id_token to Redirect URI
Validates id_token, Sets session cookie
Returns secure page to user

JWT: JSON Web Token

Base64-encoded, RSA-signed

JWT as an alternative to a private key?



Encoded PASTE A TOKEN HERE

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJ
zdWIiOiJwaGlsaXBwZUBwcmFnbWF0aWN3ZWJzZWN
1cml0eS5jb20iLCJyb2xlIjoiYWRtaW4iLCJpc3M
iOiJwcmFnbWF0aWN3ZWJzZWN1cml0eS5jb20ifQ.
jW4cq__pkcq-r6H1Ebiq8toW-
4Igstk1ibRgxECUhdExvZTzhvXqfrPewgtRHEApB
WXpUqGqRY6LSj2Gklxt3O6kxUaky-
VTl8jbL0OV5HEQVOnL3VVgPv65ddGRYaCOuyzYcf
6M1fA4PeFme9lL2ZTNtjiE0OJjUR3LH1Dptm_u9_
aQRtJ_IU8xiywctV1JLeQcMJFDXCS2N5oU0Gkatu
oJNbjMdSTg3BsU5yUsGLyuPnJTeUWJajin5e0NuB
A1Bc6oLee6KtPAM8-
1ufhHr1fpT78iGyrSQLpiVd2naPA0CvUyZ6W_4ar
nmZDKRF9N9zOR_Jxyfv5xFMi4G67EhA
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "typ": "JWT",
  "alg": "RS256"
}
```

PAYLOAD: DATA

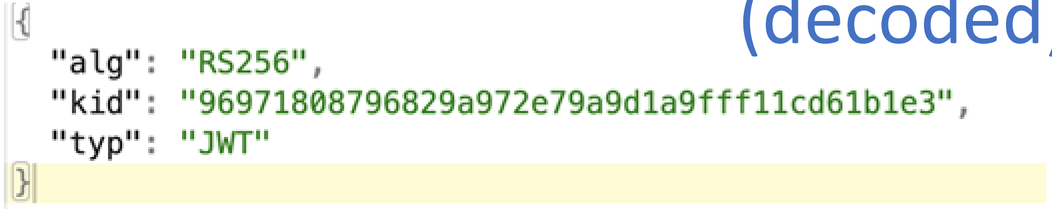
```
  "sub": "philippe@pragmaticwebsecurity.com",
  "role": "admin",
  "iss": "pragmaticwebsecurity.com"
}
```

VERIFY SIGNATURE

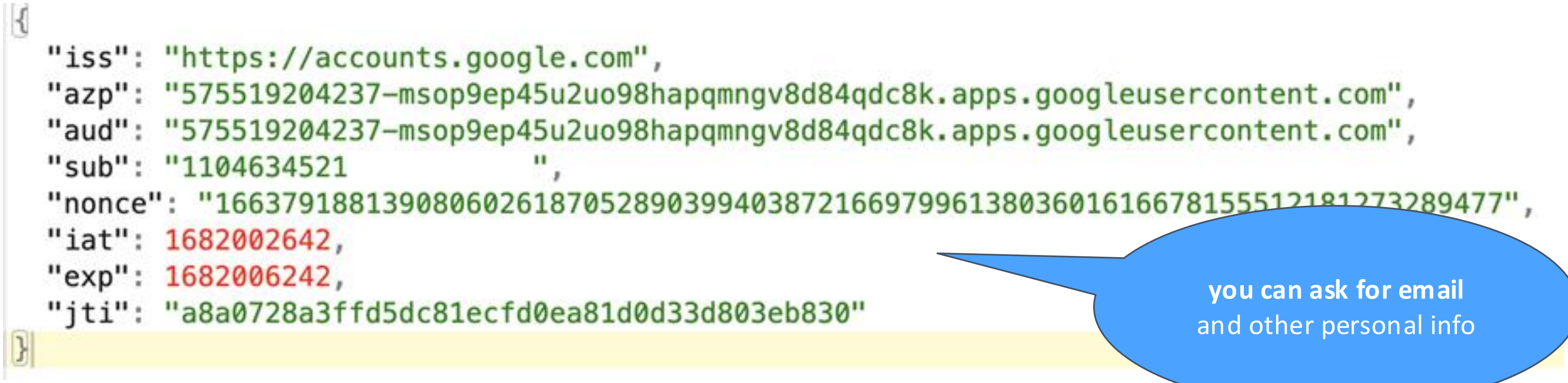
```
RSASHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  Lg8ulqDgdXLFwS/lHXV/QKcBOXBrIp
  B4DWQ0c16zLZU7NTe657rWqKlwIDAQ
  AB
  -----END RSA PUBLIC KEY-----
```

A Google-issued JWT (decoded)

Sign in with Google



```
{
  "alg": "RS256",
  "kid": "96971808796829a972e79a9d1a9fff11cd61b1e3",
  "typ": "JWT"
}
```



```
{
  "iss": "https://accounts.google.com",
  "azp": "575519204237-msop9ep45u2uo98hapqmngv8d84qdc8k.apps.googleusercontent.com",
  "aud": "575519204237-msop9ep45u2uo98hapqmngv8d84qdc8k.apps.googleusercontent.com",
  "sub": "1104634521            ",
  "nonce": "16637918813908060261870528903994038721669799613803601616678155512181273289477",
  "iat": 1682002642,
  "exp": 1682006242,
  "jti": "a8a0728a3ffd5dc81ecfd0ea81d0d33d803eb830"
}
```

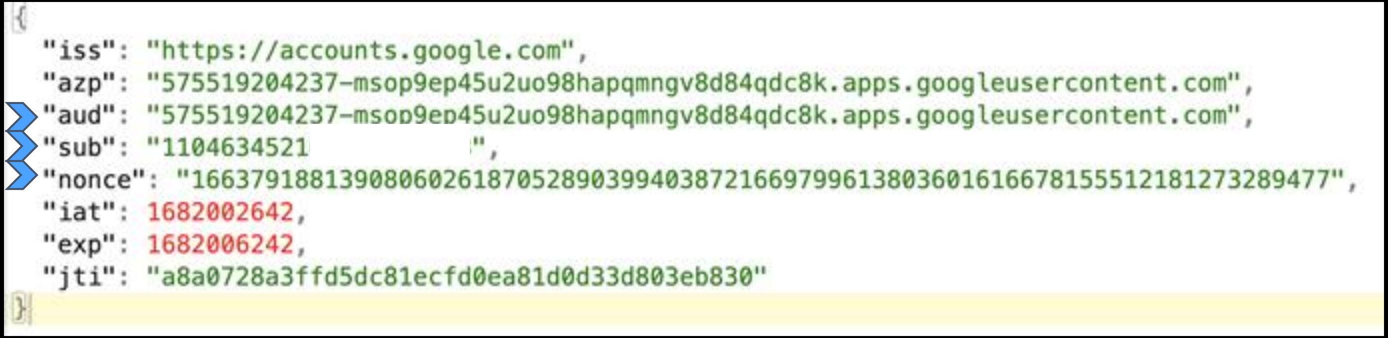
you can ask for email
and other personal info

zkLogin tricks

sample openID JWT token
signed by Google / FB

aud = walletID
sub = userID

we could ask for email too



```
{
  "iss": "https://accounts.google.com",
  "azp": "575519204237-msop9ep45u2uo98hapqmngv8d84qdc8k.apps.googleusercontent.com",
  "aud": "575519204237-msop9ep45u2uo98hapqmngv8d84qdc8k.apps.googleusercontent.com",
  "sub": "1104634521[redacted]",
  "nonce": "16637918813908060261870528903994038721669799613803601616678155512181273289477",
  "iat": 1682002642,
  "exp": 1682006242,
  "jti": "a8a0728a3ffd5dc81ecfd0ea81d0d33d803eb830"
}
```

nonce = eph. pubKey + expiration

+

add **salt**

inject **eph key**

+ **ZK proof** =

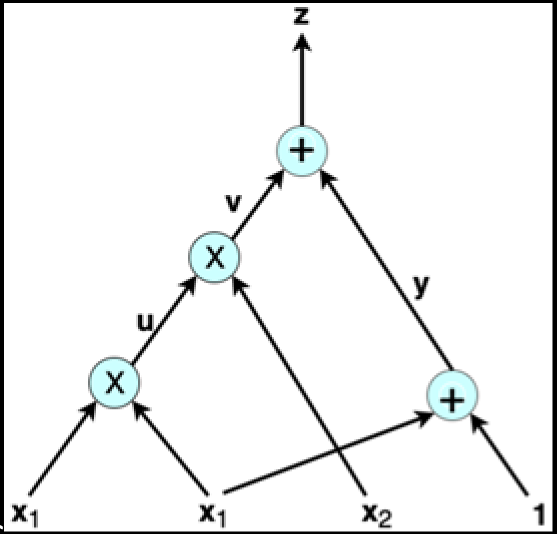
ADDRESS
~hash(providerID + zkhash(walletID + userID + zkhash(salt)))

&

verify ZKproof + **verify eph key sig**

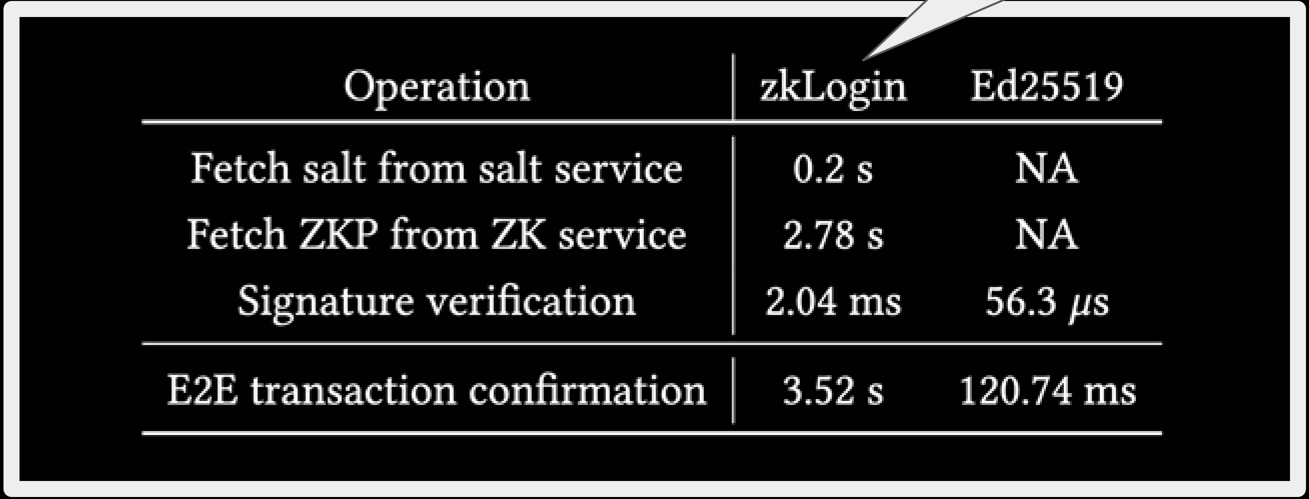
Circuit details

- Implemented in circom: ~1M R1CS constraints
- Key operations
  - SHA-2 (66%)
  - RSA signature verification (14%) using tricks from [KPS18]
  - JSON parsing, Poseidon hashing, Base64, extra rules (20%)
- Prover based on rapidsnark



zkLogin latency

These numbers correspond only to the **first transaction of a session**



Operation	zkLogin	Ed25519
Fetch salt from salt service	0.2 s	NA
Fetch ZKP from ZK service	2.78 s	NA
Signature verification	2.04 ms	56.3 μs
E2E transaction confirmation	3.52 s	120.74 ms

Latency for most zkLogin transactions is **very similar** to traditional ones!

MystenLabs Sui COSIC Soundness Labs

Q & A

ZK for authentication

How to SNARK sign-in with Google, Apple & FB

Paper

Sui docs

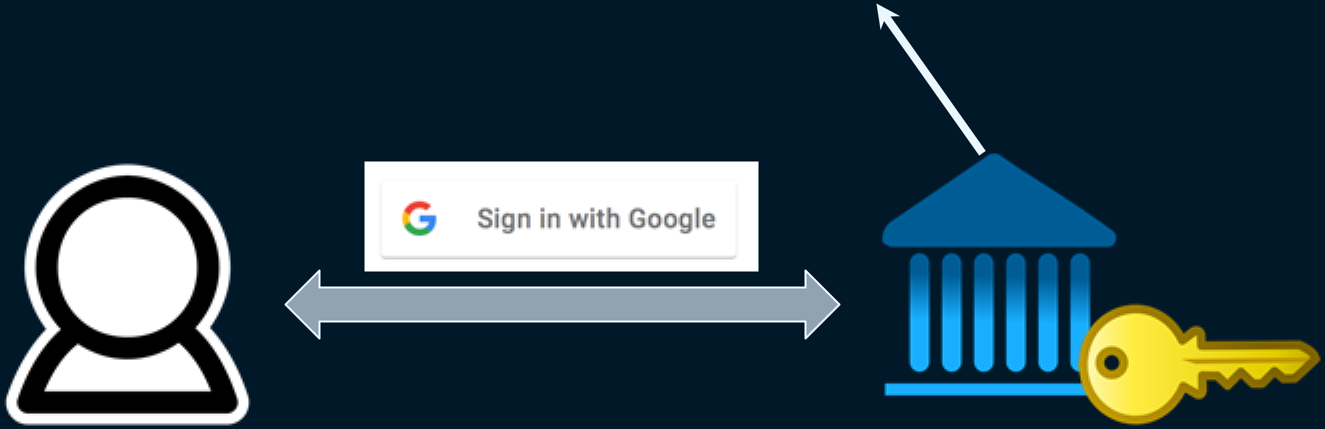
Demo

Contact: mahdi@soundness.xyz

Slides credited to Mysten Labs crypto team.

Backup slides

Naive solution: OAuth + Custodian



Can we avoid the trusted custodian?

zkLogin goodies

Native auth, cheap

Not via smart contracts, same gas cost as regular sig verification.

ID-based wallets

Create email or phone number based accounts.

Can also reveal identity of an existing account (e.g., email) fully or partially (e.g., reveal a suffix like @xyz.edu)

Embedded wallet

Mobile apps or websites can natively integrate zkLogin without the need for a wallet popup!

2FA

Can do a 2-out-of-3 between Google, Facebook and Apple. Salt can also serve as a second factor.

Hard to lose!

Thanks to robust recovery paths of Google, Facebook.

ADDRESS

hash(providerID + zkhash(walletID + userID + zkhash(salt)))

+ ZK proof

zkLogin

single-click accounts w/

- Google
- Facebook
- Twitch
- Apple
- Slack
- Microsoft

native authenticator

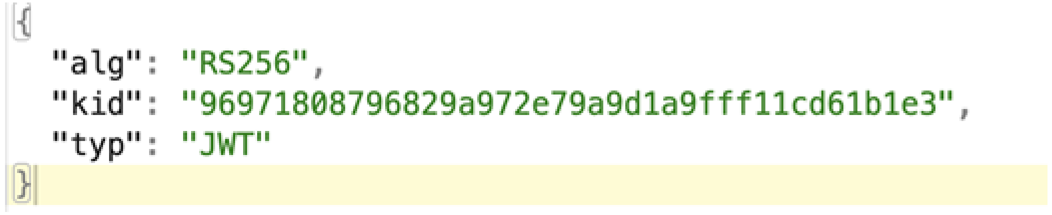
non-custodial

*discoverable, claimable

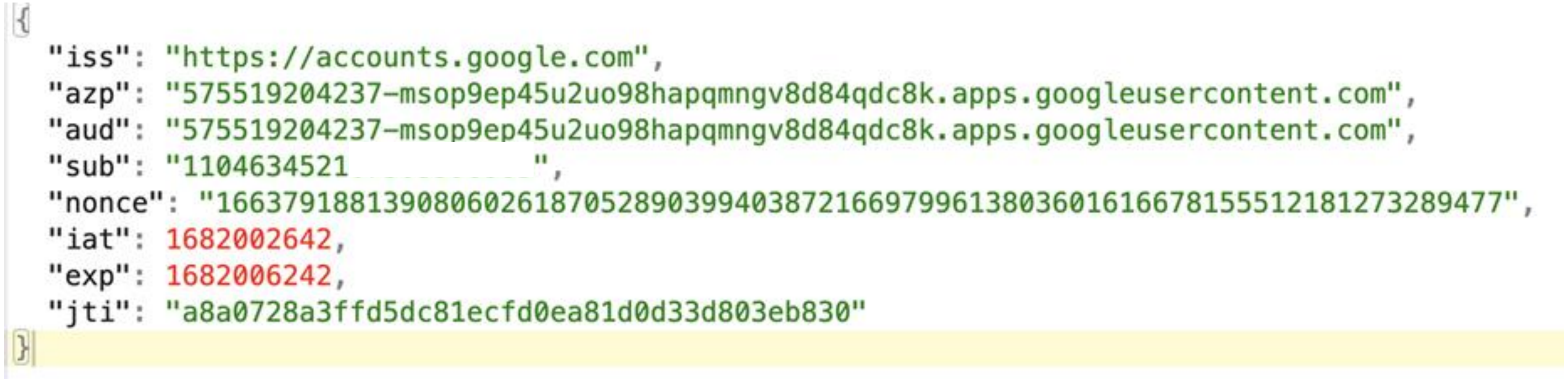
invisible wallets

semi-portable, 2FA

Challenge 1: How to authorize a tx with a JWT?

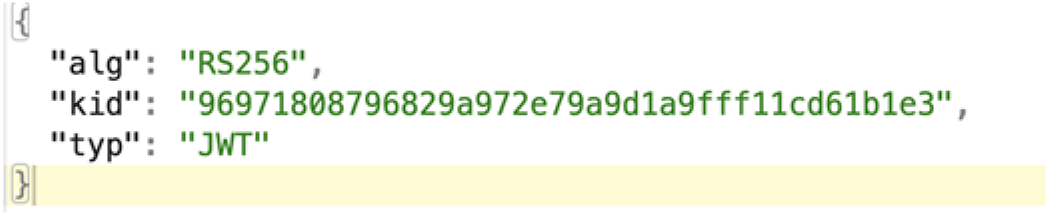


```
{
  "alg": "RS256",
  "kid": "96971808796829a972e79a9d1a9fff11cd61b1e3",
  "typ": "JWT"
}
```

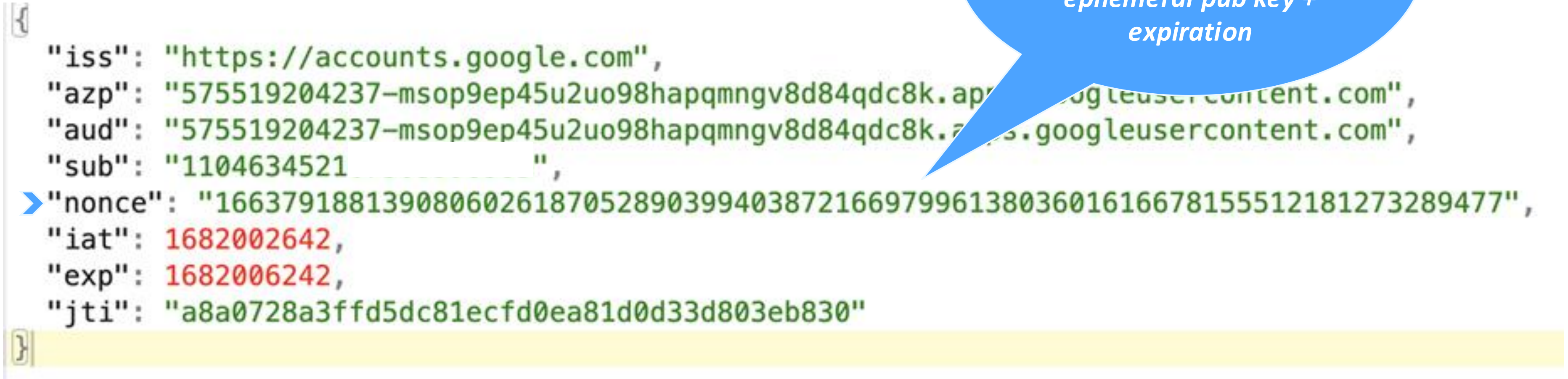


```
{
  "iss": "https://accounts.google.com",
  "azp": "575519204237-msop9ep45u2uo98hapqmngv8d84qdc8k.apps.googleusercontent.com",
  "aud": "575519204237-msop9ep45u2uo98hapqmngv8d84qdc8k.apps.googleusercontent.com",
  "sub": "1104634521           ",
  "nonce": "16637918813908060261870528903994038721669799613803601616678155512181273289477",
  "iat": 1682002642,
  "exp": 1682006242,
  "jti": "a8a0728a3ffd5dc81ecfd0ea81d0d33d803eb830"
}
```

Inject a fresh pub key into JWT!



```
{
  "alg": "RS256",
  "kid": "96971808796829a972e79a9d1a9fff11cd61b1e3",
  "typ": "JWT"
}
```



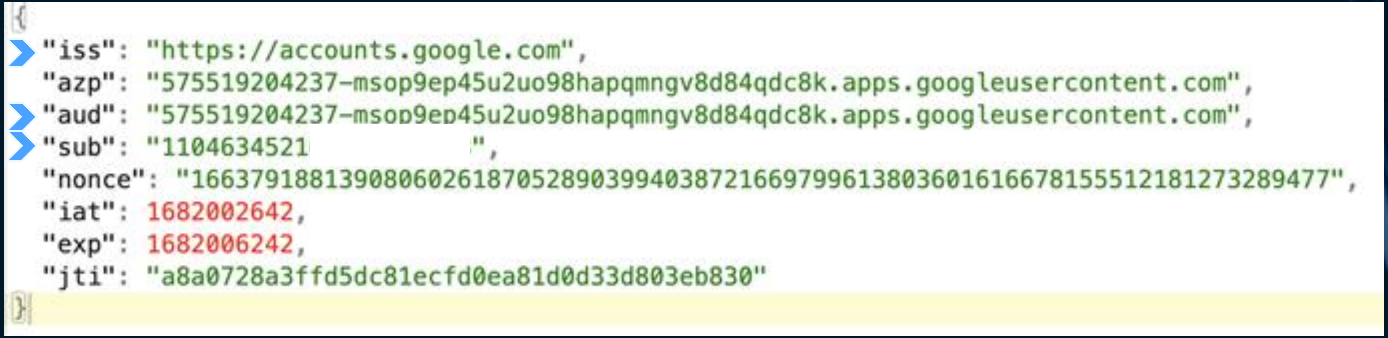
```
{
  "iss": "https://accounts.google.com",
  "azp": "57519204237-msop9ep45u2uo98hapqmngv8d84qdc8k.ap          ogleusercontent.com",
  "aud": "57519204237-msop9ep45u2uo98hapqmngv8d84qdc8k.a    s.googleusercontent.com",
  "sub": "1104634521            ",
 >"nonce": "16637918813908060261870528903994038721669799613803601616678155512181273289477",
  "iat": 1682002642,
  "exp": 1682006242,
  "jti": "a8a0728a3ffd5dc81ecfd0ea81d0d33d803eb830"
}
```

replace ***nonce*** with user provided data:

ephemeral pub key + expiration

We have a DIGITAL CERT over our fresh key + expiration

Challenge 2: How to identify the user without linking identities?



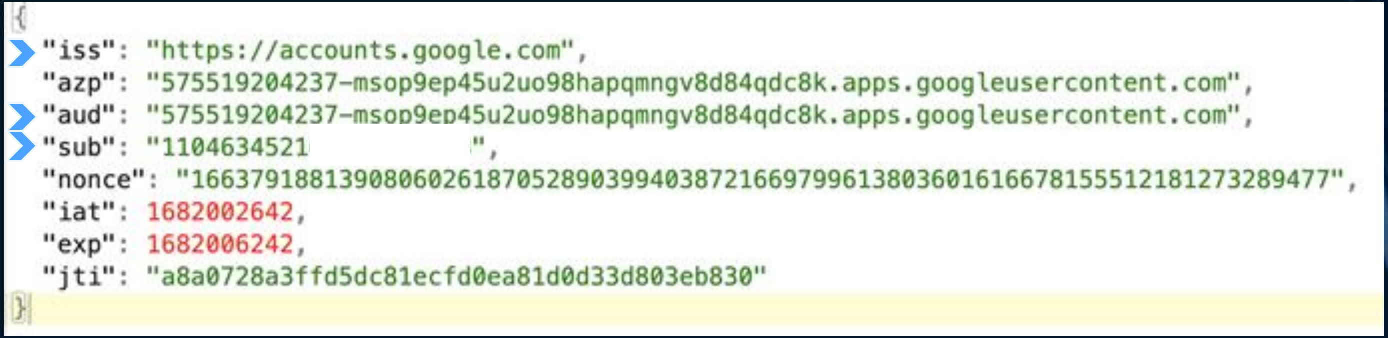
```
{
"iss": "https://accounts.google.com",
"azp": "575519204237-msop9ep45u2uo98hapqmngv8d84qdc8k.apps.googleusercontent.com",
"aud": "575519204237-msop9ep45u2uo98hapqmngv8d84qdc8k.apps.googleusercontent.com",
"sub": "1104634521[redacted]",
"nonce": "16637918813908060261870528903994038721669799613803601616678155512181273289477",
"iat": 1682002642,
"exp": 1682006242,
"jti": "a8a0728a3ffd5dc81ecfd0ea81d0d33d803eb830"
}
```

aud = walletID
sub = userID

we could ask for email too

ADDRESS
???

Add a persistent randomizer: salt



```
{
  "iss": "https://accounts.google.com",
  "azp": "575519204237-msop9ep45u2uo98hapqmngv8d84qdc8k.apps.googleusercontent.com",
  "aud": "575519204237-msop9ep45u2uo98hapqmngv8d84qdc8k.apps.googleusercontent.com",
  "sub": "1104634521[redacted]",
  "nonce": "16637918813908060261870528903994038721669799613803601616678155512181273289477",
  "iat": 1682002642,
  "exp": 1682006242,
  "jti": "a8a0728a3ffd5dc81ecfd0ea81d0d33d803eb830"
}
```

aud = walletID
sub = userID

we could ask for email too

ADDRESS
hash(providerID + walletID + userID + salt)

Salt: A persistent per-user secret for **unlinkability**

Who maintains the salt?

- Client-side on-device management
  - Edge cases, e.g., cross-device sync, device loss need handling

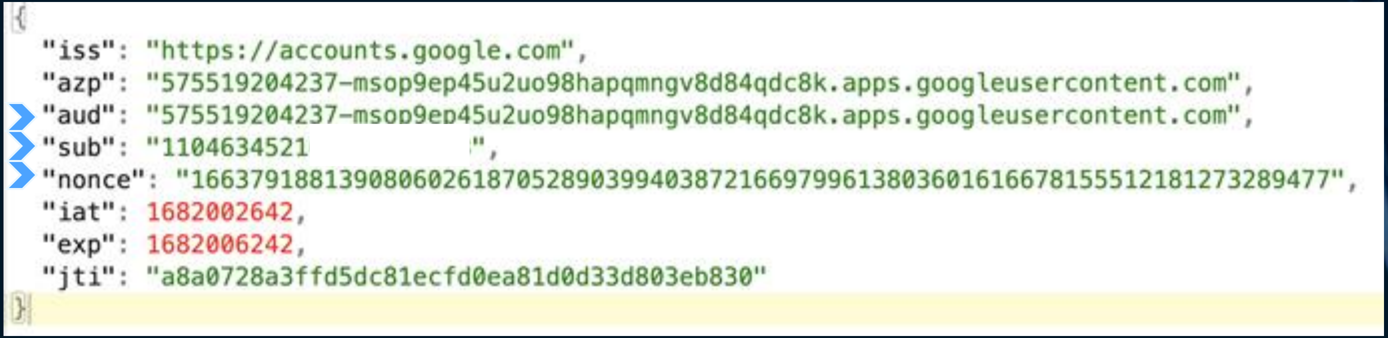
- Server-side management by a “salt service”
  - Each wallet can maintain their own service / delegate it
  - Privacy models: Store salt either in TEE / MPC / plaintext
  - Auth policies to the service: Either JWT or 2FA

ADDRESS

hash(providerID + walletID + userID + salt)

Salt: A persistent per-user secret for **unlinkability**

Challenge 3: How to hide the JWT? SNARKs to the rescue!



```
{
  "iss": "https://accounts.google.com",
  "azp": "575519204237-msop9ep45u2uo98hapqmngv8d84qdc8k.apps.googleusercontent.com",
  "aud": "575519204237-msop9ep45u2uo98hapqmngv8d84qdc8k.apps.googleusercontent.com",
  "sub": "1104634521[redacted]",
  "nonce": "16637918813908060261870528903994038721669799613803601616678155512181273289477",
  "iat": 1682002642,
  "exp": 1682006242,
  "jti": "a8a0728a3ffd5dc81ecfd0ea81d0d33d803eb830"
}
```

aud = walletID
sub = userID

we could ask for email too

nonce = eph. pubKey + expiration

Goal: Prove you have a valid JWT + you know the salt + you injected the ephemeral key into JWT

- Verify JWT's signature using Google's public key
- Verify the ephemeral public key is injected into the JWT's nonce
- Verify that the address is derived correctly from the JWT's userID, walletID providerID + user's salt

Yellow => private inputs
Blue => public inputs

Challenge 4: Prove + RTT in <3s

- We chose Groth16 due to its small proofs + rich ecosystem + fast prover
- But.. proofs are slow to generate on end-user devices
    - Make ZKP efficient: Hand-optimized circuit that selectively parses relevant parts of the JWT + string slicing tricks + …
    - Delegate proving to an untrusted ZKP service
        - Open problem: How to delegate with privacy?