ZKPROOF

ZK for modern authentication

How & Why to S{T}NARK our login

Kostas Kryptos Chalkias | Yan Ji | Jonas Lindstrøm | Deepak Maram | Ben Riva
Arnab Roy | Mahdi Sedaghat | Joy Wang | Foteini Baldimtsi

MystenLabs Sui

Berlin | May 24, 2024

ZK auth flavors

ZK friendly signatures

Ziggy

"I know a secret preimage **sk** such that **pk = H(sk)**"

sig = HMAC(sk, msg)

*Prove **sk** matches (in H and HMAC)*

RedJubjub

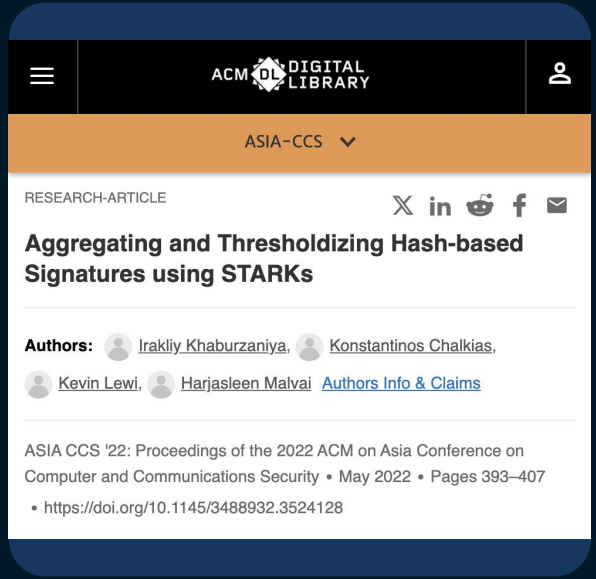
RedDSA (variant of EdDSA) over the Jubjub zk-friendly twisted Edwards curve.

zk-based Sig Aggregation

+ threshold MPC → ZK wallets

STARKs for Sphincs

Aggregate or thresholdize PQ signatures via a PQ zk-scheme.

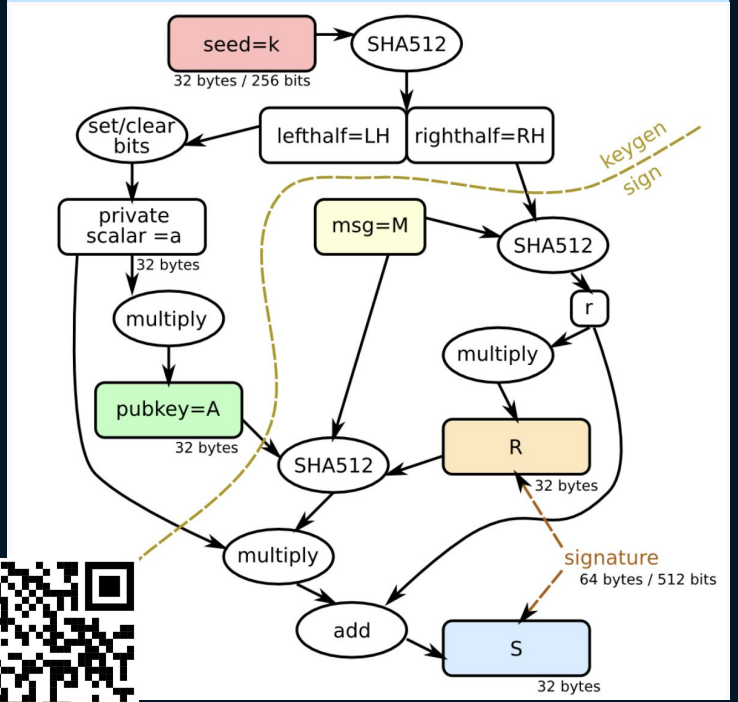


PQ fallback

W-OTS(+) up my Sleeve

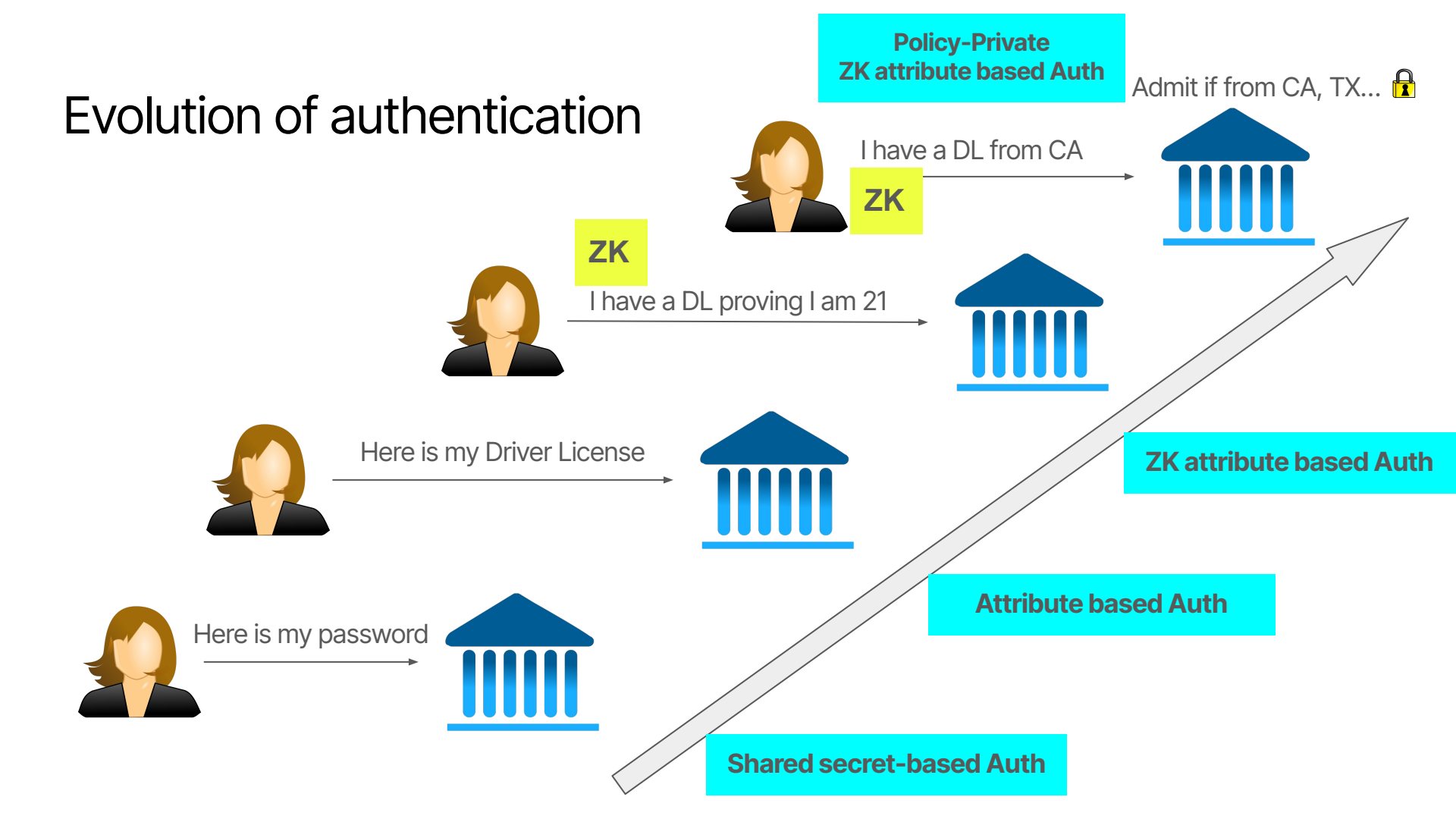
Chaum et al., then Vitalik
most private keys are the result of hash calculations
(BIP32 mnemonic to priv key)

Is EdDSA keygen PQ-ready?

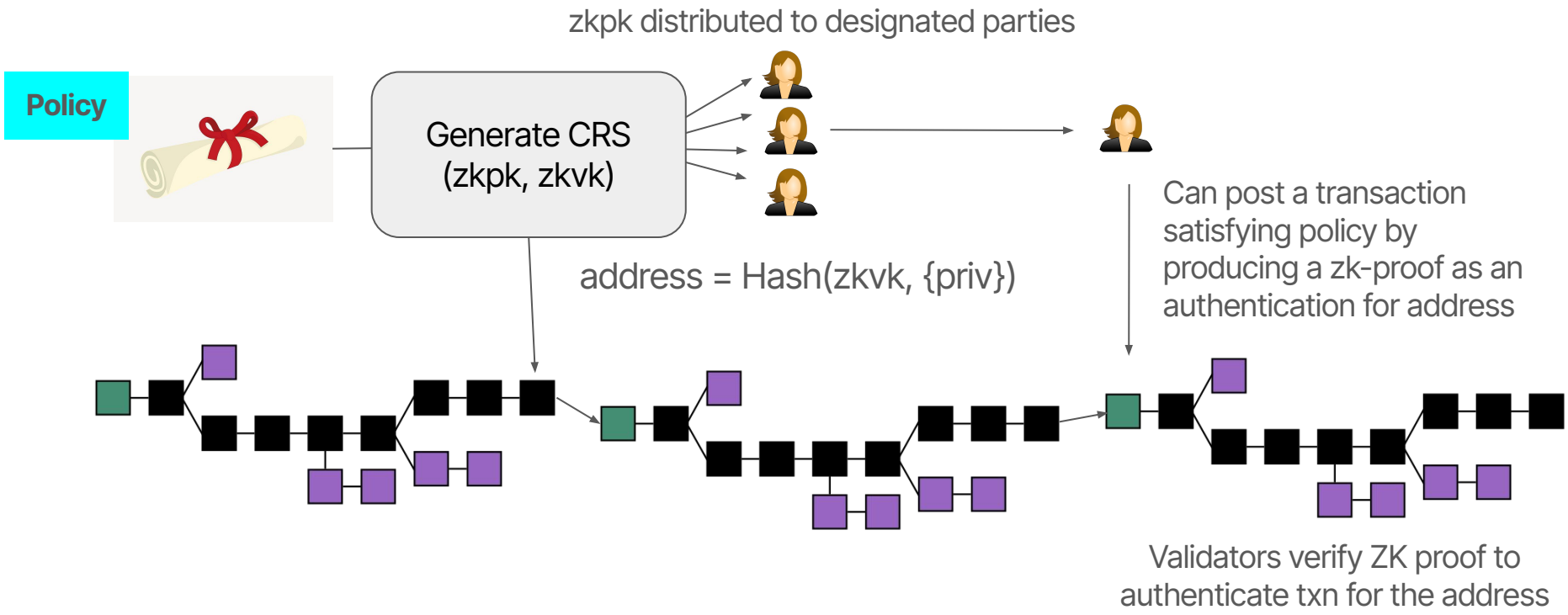


pq-zkproof(seed → priv key)

Evolution of authentication



ZK Authenticator on Blockchains

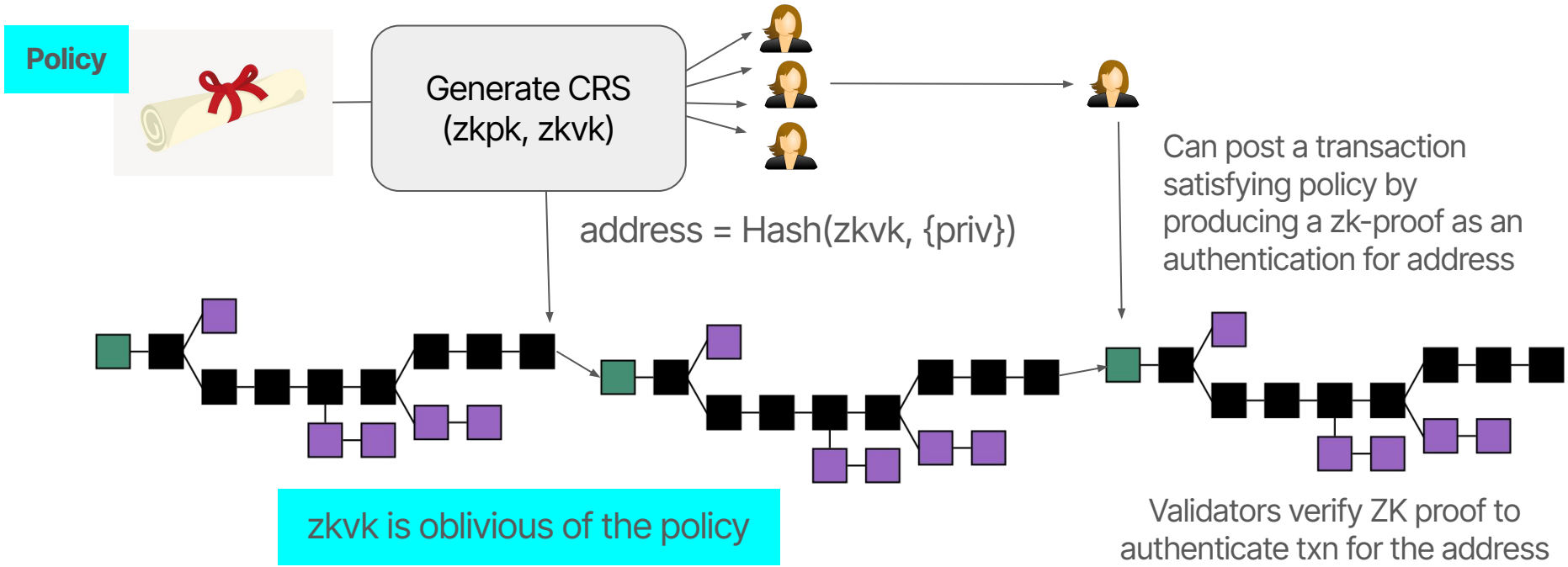


Let’s prove Groth16 is NOT dead

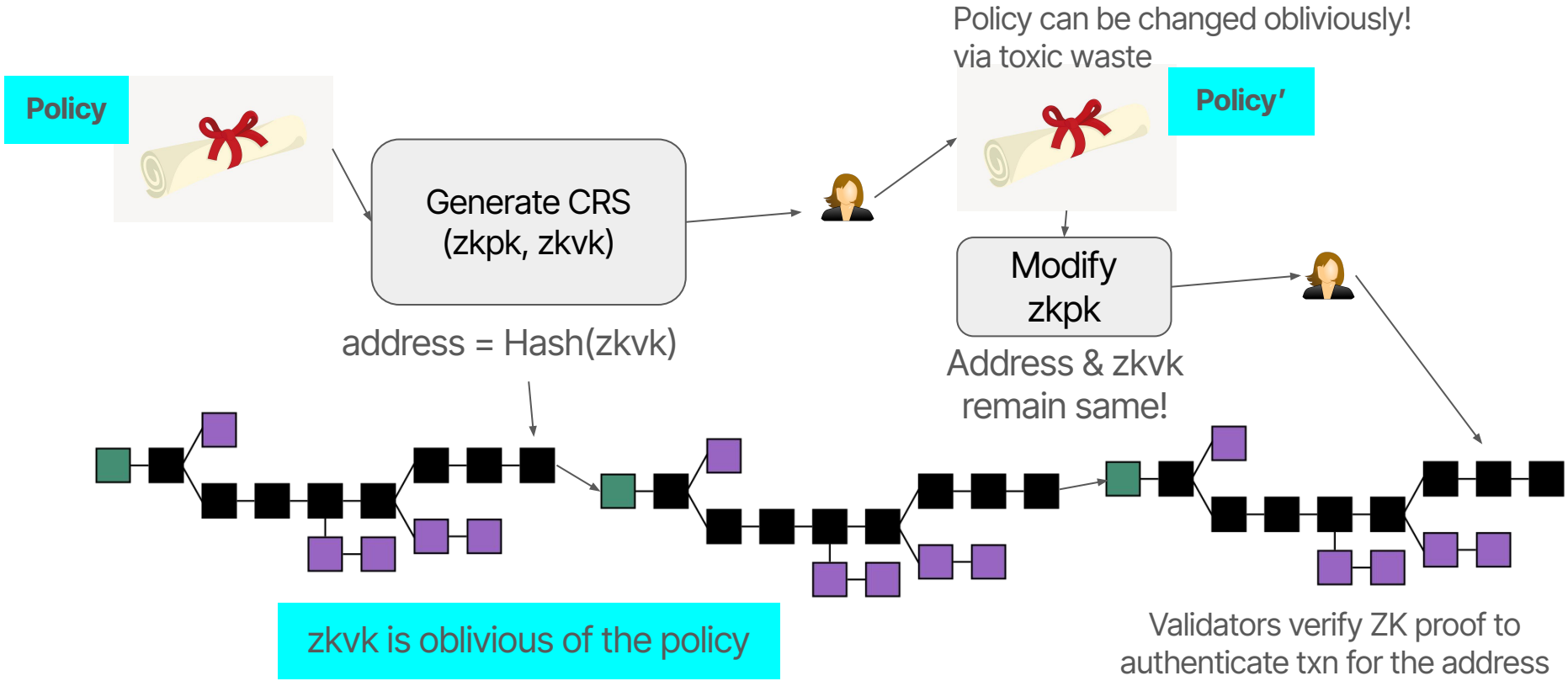
+

trusted ceremony is NOT always required

Policy-Private ZK Authenticator



Dynamic Oblivious Policy-Private ZK Authenticator



asked chatGPT

K

What do you think is the biggest problem for the average user to adopt the blockchain technology?

Blockchain technology, despite its transformative potential, has seen challenges in widespread adoption among average users. Several factors contribute to this:

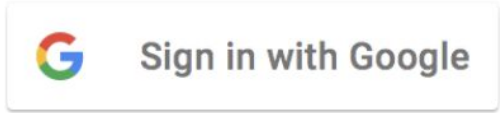
1. **Complexity and Understanding**: Blockchain, as a concept, is not straightforward for everyone. Terms like decentralized ledgers, consensus algorithms, and cryptographic hashing can be overwhelming. The average user often struggles to understand how blockchains work, which makes them hesitant to adopt the technology.
2. **User Interface and Experience**: Many blockchain applications and platforms still don't offer the level of user-friendliness that's typical of mainstream applications. The user interfaces can be clunky, unintuitive, or overly technical.

segment	cabbage	shuffle	mosquito
nephew	furnace	cinnamon	garment
wine	wing	wink	stumble
saddle	nominee	grief	vacuum

zkLogin:
OAuth2 + Zero Knowledge Proof

SNARKing your login w/ Google, Apple, FB

example Google openID 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": "166379188139080602618705289039940387216697996138036016166781555121812732894 77",
  "iat": 1682002642,
  "exp": 1682006242,
  "jti": "a8a0728a3ffd5dc81ecfd0ea81d0d33d803eb830"
}
```


you can ask for email
and other personal info

first trick before SNARKing

inject a fresh / ephemeral pub key to 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": "166379188139080602618705289039940387216697996138036016166781555121812732894 77",
  "iat": 1682002642,
  "exp": 1682006242,
  "jti": "a8a0728a3ffd5dc81ecfd0ea81d0d33d803eb830"
}
```


OpenPubKey

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

ephemeral pub key + expiration

first trick before SNARKing

inject a fresh / ephemeral pub key to 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": "166379188139080602618705289039940387216697996138036016166781555121812732894 77",
  "iat": 1682002642,
  "exp": 1682006242,
  "jti": "a8a0728a3ffd5dc81ecfd0ea81d0d33d803eb830"
}
```


We have a DIGITAL CERT over our fresh key + expiration

OpenPubKey

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

ephemeral pub key + expiration

2nd trick before SNARKing

ID-based (key-less) address


```
{
  "iss": "https://accounts.google.com",
  "azp": "575519204237-msop9ep45u2uo98hapqmngv8d84qdc8k.apps.googleusercontent.com",
  "aud": "575519204237-msop9ep45u2uo98hapqmngv8d84qdc8k.apps.googleusercontent.com",
  "sub": "1104634521            ",
  "nonce": "166379188139080602618705289039940387216697996138036016166781555121812732894 77",
  "iat": 1682002642,
  "exp": 1682006242,
  "jti": "a8a0728a3ffd5dc81ecfd0ea81d0d33d803eb830"
}
```


iss = providerID
aud = walletID
sub = userID
we could ask for email too

nonce =
eph. pubKey
+ expiration

strawman ADDRESS
hash(providerID + walletID + userID)

2nd trick before SNARKing

ID-based (key-less) address


```
{
  "iss": "https://accounts.google.com",
  "azp": "575519204237-msop9ep45u2uo98hapqmngv8d84qdc8k.apps.googleusercontent.com",
  "aud": "575519204237-msop9ep45u2uo98hapqmngv8d84qdc8k.apps.googleusercontent.com",
  "sub": "1104634521            ",
  "nonce": "166379188139080602618705289039940387216697996138036016166781555121812732894 77",
  "iat": 1682002642,
  "exp": 1682006242,
  "jti": "a8a0728a3ffd5dc81ecfd0ea81d0d33d803eb830"
}
```


iss = providerID
aud = walletID
sub = userID
we could ask for email too

nonce =
eph. pubKey
+ expiration

strawman ADDRESS

hash(providerID + walletID + userID)

3rd trick before SNARKing

Randomized ID-based (key-less) address


```
{
  "iss": "https://accounts.google.com",
  "azp": "575519204237-msop9ep45u2uo98hapqmngv8d84qdc8k.apps.googleusercontent.com",
  "aud": "575519204237-msop9ep45u2uo98hapqmngv8d84qdc8k.apps.googleusercontent.com",
  "sub": "1104634521            ",
  "nonce": "166379188139080602618705289039940387216697996138036016166781555121812732894 77",
  "iat": 1682002642,
  "exp": 1682006242,
  "jti": "a8a0728a3ffd5dc81ecfd0ea81d0d33d803eb830"
}
```


iss = providerID
aud = walletID
sub = userID
we could ask for email too

nonce =
eph. pubKey
+ expiration

strawman ADDRESS

hash(providerID + walletID + userID + salt)

But who owns the salt?

3rd trick before SNARKing

Randomized ID-based (key-less) Address


```
{
  "iss": "https://accounts.google.com",
  "azp": "575519204237-msop9ep45u2uo98hapqmngv8d84qdc8k.apps.googleusercontent.com",
  "aud": "575519204237-msop9ep45u2uo98hapqmngv8d84qdc8k.apps.googleusercontent.com",
  "sub": "1104634521            ",
  "nonce": "166379188139080602618705289039940387216697996138036016166781555121812732894 77",
  "iat": 1682002642,
  "exp": 1682006242,
  "jti": "a8a0728a3ffd5dc81ecfd0ea81d0d33d803eb830"
}
```


iss = providerID
aud = walletID
sub = userID
we could ask for email too

nonce =
eph. pubKey
+ expiration

strawman ADDRESS

hash(providerID + walletID + userID + salt)

But who owns the salt?

4th trick before SNARKing

zk-friendly hashing


```
{
  "iss": "https://accounts.google.com",
  "azp": "575519204237-msop9ep45u2uo98hapqmngv8d84qdc8k.apps.googleusercontent.com",
  "aud": "575519204237-msop9ep45u2uo98hapqmngv8d84qdc8k.apps.googleusercontent.com",
  "sub": "1104634521            ",
  "nonce": "166379188139080602618705289039940387216697996138036016166781555121812732894 77",
  "iat": 1682002642,
  "exp": 1682006242,
  "jti": "a8a0728a3ffd5dc81ecfd0ea81d0d33d803eb830"
}
```


iss = providerID
aud = walletID
sub = userID
we could ask for email too

nonce =
eph. pubKey
+ expiration

ADDRESS

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

5th trick: SNARK-it


```
{
  "iss": "https://accounts.google.com",
  "azp": "575519204237-msop9ep45u2uo98hapqmngv8d84qdc8k.apps.googleusercontent.com",
  "aud": "575519204237-msop9ep45u2uo98hapqmngv8d84qdc8k.apps.googleusercontent.com",
  "sub": "1104634521            ",
  "nonce": "166379188139080602618705289039940387216697996138036016166781555121812732894 77",
  "iat": 1682002642,
  "exp": 1682006242,
  "jti": "a8a0728a3ffd5dc81ecfd0ea81d0d33d803eb830"
}
```


iss = providerID
aud = walletID
sub = userID
we could ask for email too

nonce =
eph. pubKey
+ expiration

- Verify JWT correctness
- Verify openID provider's signature (RSA) over base64 of sha256(JWT)
- Verify the public key and expiration are injected into the nonce
- Verify knowledge of address pre-image and that JWT's userID is part of it.

Goal: hide JWT data, but prove you have a valid, non-expired JWT over your userID + you know the salt + you injected the eph. key into JWT.

zkLogin tricks

sample openID JWT token signed by Google / FB

```
{
  "iss": "https://accounts.google.com",
  "azp": "575519204237-msop9ep45u2uo98hapqmngv8d84qdc8k.apps.googleusercontent.com",
  "aud": "575519204237-msop9ep45u2uo98hapqmngv8d84qdc8k.apps.googleusercontent.com",
  "sub": "1104634521            ",
  "nonce": "166379188139080602618705289039940387216697996138036016166781555121812732894 77",
  "iat": 1682002642,
  "exp": 1682006242,
  "jti": "a8a0728a3ffd5dc81ecfd0ea81d0d33d803eb830"
}
```

iss = providerID
aud = walletID
sub = userID
we could ask for email too

nonce =
eph. pubKey
+ expiration

+

+ **ZK proof** =

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

&

verify ZKproof + verify eph key sig

Groth16 Circuit Implementation

- Implemented in circom: 1M constraints (2^20 - 7 to be precise)
  - Avoided going above 1M due to ceremony dependency + proof time
  - https://github.com/sui-foundation/zklogin-circuit/tree/main
- Key operations: SHA-2 (74%), RSA signature verification (15%) and JSON parsing, Poseidon hashing, Base64, extra rules, string slicing (11%)
- Delegated prover to backend is possible
  - The prover doesn't need to be "trusted" for security thanks to the nonce embedding trick!

zkLogin properties

Native auth, cheap

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

Geofencing

Contracts can accept accounts from particular countries or orgs.

Prove your ID

Reveal identity only when the user wants to!

Fully or Partially, i.e., prove this account is from a particular influencer or gov or some@zkproof.com

Discoverability

Send assets and tokens just by typing someone's known ID (e.g. fb_link)

Claimability

Receive assets even before onboarding!

2FA

Can be combined with other auth schemes in 1-of-2 or o 2-of-2 or any other structure. Imagine sign-in with mnemonic and/or zkLogin

Reusable

Every wallet can use it, not a vendor's privilege.

ADDRESS

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

+ ZK proof

Zklogin

single-click accounts w/

- Google
- Facebook
- Twitch
- Slack
- Apple

native authenticator
non-custodial
*discoverable, claimable
invisible wallets
semi-portable, 2FA

ZK for modern authentication

How & Why to S{T}NARK our login

Thanks

Quick Contacts

kostas@mystenlabs.com

deepak@mystenlabs.com

joy@mystenlabs.com

arnab@mystenlabs.com

zkLogin paper: https://arxiv.org/abs/2401.11735