Anon-Aadhaar Protocol using Halo2 and Noir

Proof of Identity of Indian Citizens using ZKPs

Hridam Basu
Cryptography Research Engineer
PSE, Ethereum Foundation

Plan Ahead

- Introduction
- Problem Statement
- Anon-Aadhaar
- How does it work?
  - Secure QR Code
  - Circuit
  - Features
- Halo2 Implementation
- Noir Implementation
- Applications

Introduction

- Anon-Aadhaar is a zero-knowledge protocol which allows Indian citizens with an Aadhaar card to prove their identity in a privacy-preserving manner without revealing sensitive information
- Provides a set of tools for generating and verifying proofs, authenticate users and verify proofs on-chain

Previous Proof-of-Identity Solutions

- Need to reveal full identity for KYC
- Vulnerable to Database Hacks and Privacy Leaks (eg. Aadhaar)
- Sensitivity of On-Chain Verification

On-Chain Identity Solutions

- Biometrics - utilizing unique biological traits like fingerprints, iris scan, etc. to create secure immutable identity records on the blockchain

- Aggregation - Combining activity data from various sources to generate a comprehensive trust score potentially including social graph analysis

- Leveraging Document Signatures - Using cryptographic signatures of government issued identity documents to verify and authenticate on-chain

Anon-Aadhaar

- Anon Aadhaar conducts verification by checking both the **SHA-256** hash and the **RSA** signature of an Aadhaar identity.
- This process ensures that users can authenticate their uniqueness as individuals.
- ZK protocol, originally built with **Circom Groth16**, developed by PSE
- Served with:
  - **Typescript SDK** [@anon-aadhaar/core]
  - **Solidity Lib** [@anon-aadhaar/contracts]
  - **React Lib** [@anon-aadhaar/react]

Problem Statement

Governments:
sign and issue identities.

"Your Id is 0x01"

Citizen

"Here is my ID"

Prover:
Construct a proof from the ID that hides the inputs.

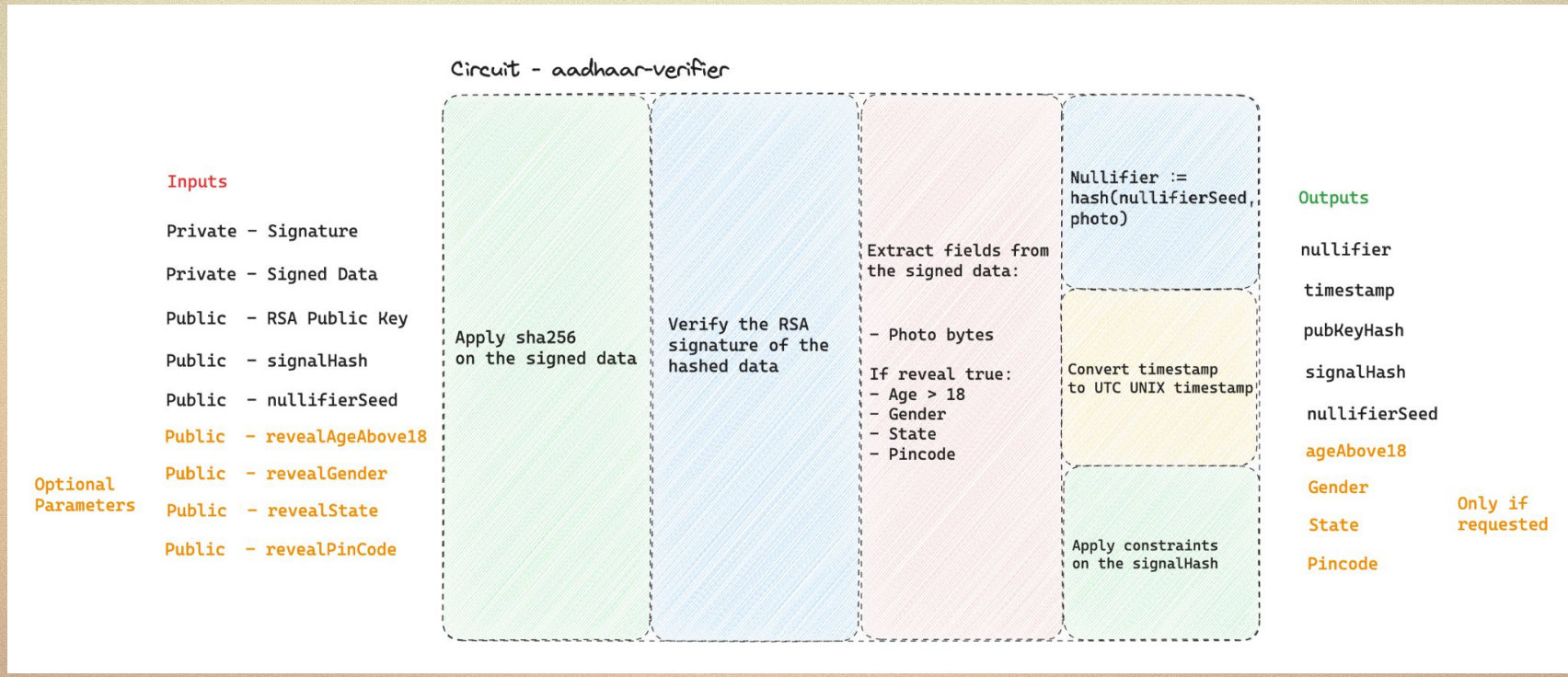
"He's a citizen"

Verifier:
Verify the proof of identity from the prover.

ZK-SNARKs

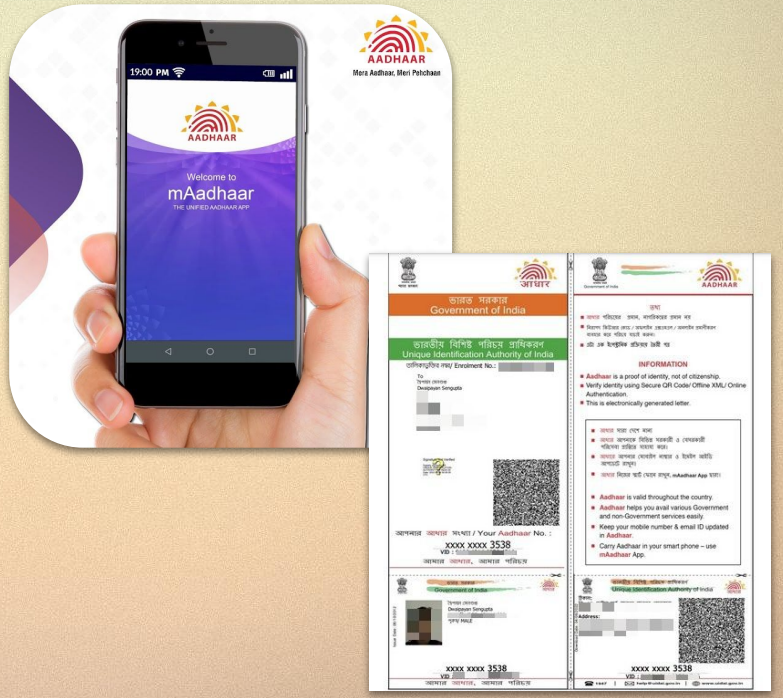
- Zero-Knowledge Proofs - Zk-SNARKs prove validity of a statement without revealing the secrets (, ie, private inputs)
- Succinctness - Circom Groth16 offers short proof sizes and short verification time, ideal for on-chain verification
- Correctness - Ensures proof validity
- Soundness - prevents forgery

Circuit


Circuit - aadhaar-verifier
Inputs
Private - Signature
Private - Signed Data
Public - RSA Public Key
Public - signalHash
Public - nullifierSeed
Public - revealAgeAbove18
Public - revealGender
Public - revealState
Public - revealPinCode
Optional Parameters
Apply sha256 on the signed data
Verify the RSA signature of the hashed data
Extract fields from the signed data:
- Photo bytes
If reveal true:
- Age > 18
- Gender
- State
- Pincode
Nullifier := hash(nullifierSeed, photo)
Convert timestamp to UTC UNIX timestamp
Apply constraints on the signalHash
Outputs
nullifier
timestamp
pubKeyHash
signalHash
nullifierSeed
ageAbove18
Gender
State
Pincode
Only if requested

Aadhaar Secure QR code

- QR code representing an Aadhaar Identity
- Signed data is hashed with Sha256
- Hash is signed with RSA
- The Signed data contains:
  - Ref ID (last 4 digits, timestamp)
  - Name
  - Gender
  - DOB
  - Mobile
  - Email
  - Address
  - Photograph
  - Signature



Features

- **userNullifier:** Prevents proof double spending and enables the revocation of users' access or interaction with your applications, enhancing privacy and control;
- **timestamp:** UNIX UTC timestamp at the moment the QR code is signed, functioning as a TOTP system (e.g. show me a proof signed less than 1 hour ago);
- **pubkeyHash:** Facilitates the validation of the signer's public key, ensuring it matches the official public key registered with UIDAI;
- **signalHash:** Empowers users to transmit a unique signal alongside their Aadhaar identity. It can be used to prevent front-running or to sign transactions with an ERC-4337 wallet;

Challenges and Caveats

- Generating Anonymous Identifiers - Creating new unique identifiers to maintain anonymity and privacy
- No control over signed data - Inability to choose what data the government signs, requiring adaptation to available signed data
- Lack of Signed documents - Not all identity documents are signed by the issuer, posing a challenge for verification (5 billion in 2024)

Halo2

- ZK library based on Rust
- Originally developed by Zcash
- Uses an evolved version of PLONKish proof system
- Maintained by PSE
- Recursive Proofs support
- Steep learning curve
- Discord server and halo2-lib telegram group

Implementation Details - Halo2

- Library Used:
  - PSE version of halo2
  - Halo2_base::halo2_proofs, poseidon
  - BN254 curves
  - Proof System: PLONKish with Recursive Proofs

- Separate Circuits
  - RSA-SHA256: halo2-rsa
  - Nullifier with Poseidon Hash function
  - Timestamp
  - Signal
  - Conditional Disclosure of Secrets

- Combine them into 1 circuit and test with real data generated for the Circom circuit

Results in Halo2

Part of the Circuit	Proving Time	Verification Time
RSA-SHA256	14.442124258s	11.461932341s
Nullifier	322.659513ms	96.948µs
Conditional Secrets	17.916018ms	307.496281ms
Timestamp	9.821774ms	1.990614ms
Signal	12.089368ms	78.350583ms

On-Chain Verification Cost in Halo2

- Halo2-Solidity-Verifier by PSE converts the circuit into a Solidity Verifier
- Deployed on Remix IDE
- Total Gas Cost: 6520332 gas
- Transaction Cost: 5669850 gas
- Execution Cost: 4955130 gas

Noir

- zk-DSL by Aztec
- Low entry barrier, Rust like syntax
- Write custom circuits and specify the constraints
- Prove using a backend like Barrentenberg CLI
- Great docs, one of the foremost zk proving systems today
- Noir Office hours

Implementation Details - Noir

- Library Used:
  - Poseidon
  - Nargo version: 0.32.0
  - Barrentenberg version: bb v0.46.1
  - Elliptic Curves: BN254
  - Proof System: Groth16
- Separate Circuits
  - Extractor
  - RSA-SHA256: noir_rsa
  - Nullifier with Poseidon Hash function
  - Timestamp
  - Signal
  - Conditional Disclosure of Secrets
- Combine them into 1 circuit and extract real data and test it

Results in Noir

Part of the Circuit	Proving Time	Verification Time
RSA-SHA256	0.502s	0.064
Nullifier	0.611s	0.066s
Conditional Secrets	0.102s	0.061s
Timestamp	0.401s	0.057s
Signal	0.092s	0.065s

On-Chain Verification Cost in Noir

- Barrentenberg backend API generates solidity contract
- Deployed on Remix IDE
- Total Gas Cost: 2904342 gas
- Transaction Cost: 2525514 gas
- Execution Cost: 2251848 gas

Open Source

- Code is entirely open-source
- Halo Implementation: https://github.com/anon-aadhaar/anon-aadhaar-halo2
- Noir Implementation: https://github.com/anon-aadhaar/anon-aadhaar-noir

Summary

- Noir is faster than Halo2 in terms of proving and verification time
- Bottleneck: the RSA-SHA256 component
- Noir on-chain verification gas cost is cheaper than Halo2
- Circom beats both Noir and Halo2 in terms of on-chain verification cost
- Noir ultrahonk backend might be more efficient

Applications

- Aadhaar Quadratic Funding / Voting
- GitcoinPassport-AnonAadhaar
- Anon Check-in (ETH India)
- Micro-Loan Approval Platform
- Payment Channel
- HeyIndia (Twitter for Aadhaar residents)
- SSO Server
- ERC-4337 wallet
- On-chain voting/polling
- Privado ID
- Semaphore (Hacker House Goa)

References

- Anon-Aadhaar: https://pse.dev/en/projects/anon-aadhaar
- Halo2 Book: https://zcash.github.io/halo2/
- Noir Docs: https://noir-lang.org/
- Awesome Halo2: https://github.com/adria0/awesome-halo2
- Awesome Noir: https://github.com/noir-lang/awesome-noir
- Halo2-Solidity-Verifier: https://github.com/privacy-scaling-explorations/halo2-solidity-verifier

Thank you!

Questions?

Email: hridam.basu@gmail.com
Linkedin: linkedin.com/in/hridambasu
X: @hridambasu
Telegram: @hridambasu