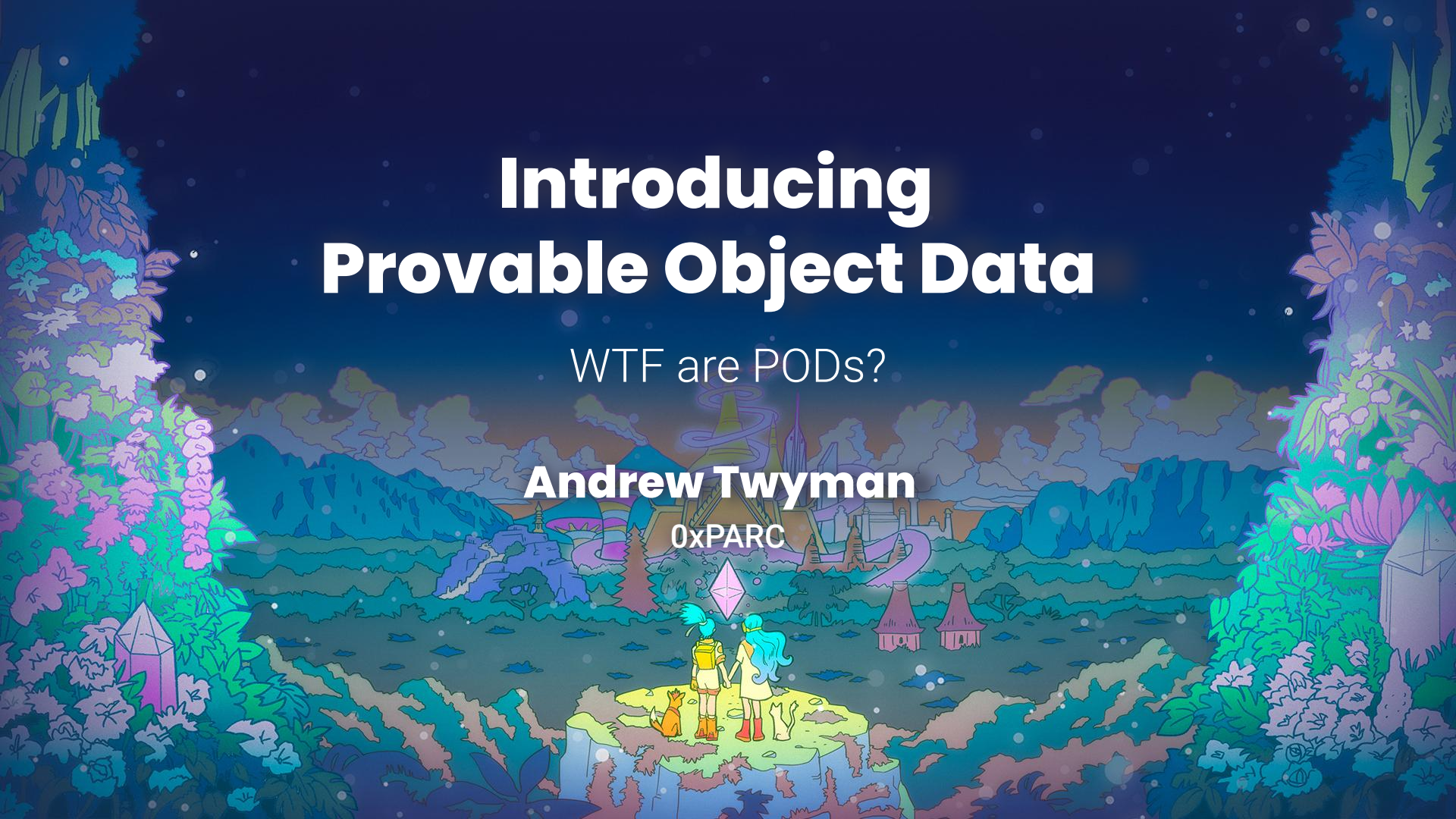
Introducing Provable Object Data

WTF are PODs?

Andrew Twyman

0xPARC



WTF are PODs?

- Your Devcon ticket
- A proof of attendance to this talk
- A cryptographic frog
- A secret message
- Your identity credentials

WTF is POD?

- POD makes it easy for any app to make and verify ZK proofs
- POD is
  - A data format
    - Optimized for efficient proving
  - A standard
    - Exchange data and proofs on any platform
  - A framework
    - With developer SDKs

README.md

@pcd/pod

project PCD | license GPL-3.0 | npm v0.5.0 | downloads 4.3k/month

Developer Site | Tutorial Code | TypeDoc | GitHub

A library for creating and manipulating objects in the Provable Object Data format. For a full introduction, see the Developer Site.

POD libraries enable any app to create zero-knowledge proofs of cryptographic data. A POD could represent your ticket to an event, a secure message, a collectible badge, or an item in a role-playing game. Using PODs, developers can create ZK-enabled apps without the effort and risk of developing custom cryptography.

ZK proofs about PODs use General Purpose Circuits (**GPC**) which can prove many different things about PODs without revealing all details. GPCs use human-readable configuration and pre-compiled circuits so no knowledge of circuit programming is required.

PODs and GPCs can be used in Zupass, or in your own apps without Zupass.

What is a POD?

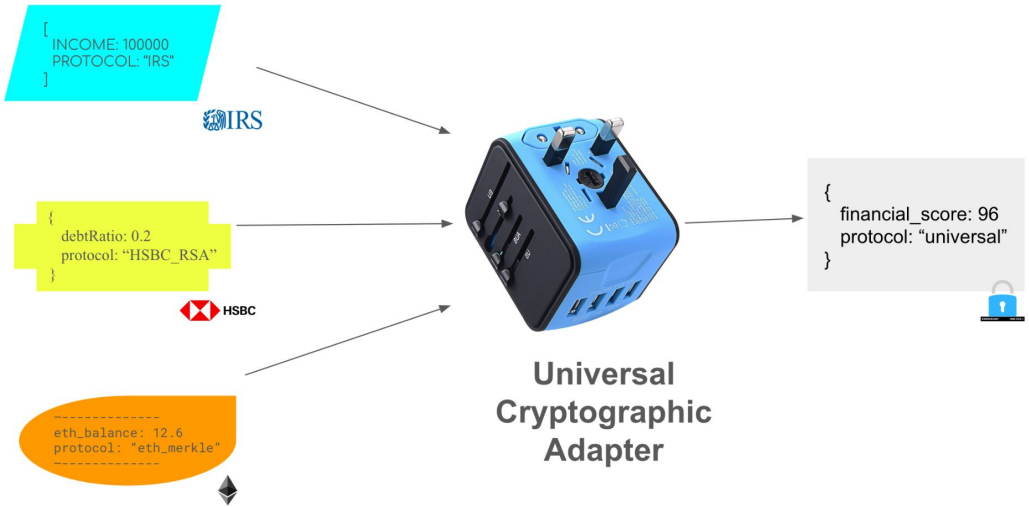
To a user, a POD is a piece of cryptographic data attested by some issuing authority. For a developer, a POD object is a key-value store which can hold any data. The whole POD is signed by an issuer. Apps can verify the signature, to trust the authenticity of the values in the POD.

When a POD is issued, its entries (key-value pairs) are hashed as part of a Merkle tree. This allows GPCs to selectively prove about individual entries without revealing the whole POD.

```
const podSword = POD.sign(
  {
    pod_type: { type: "string", value: "myrpg.item.weapon" },
    attack: { type: "int", value: 7n },
    weaponType: { type: "string", value: "sword" },
    itemSet: { type: "string", value: "celestial" },
    isMagical: { type: "boolean", value: true },
    owner: { type: "eddsa_pubkey", value: purchaser.pubKey }
  } satisfies PODEntries,
  privateKey
);
```

WTF are Zero Knowlege Proofs?

- Prove the anything about private data without revealing
- Trustworthy via math
  - Comes with some complexity (more later)



https://0xparc.org/blog/programmable-cryptography-1

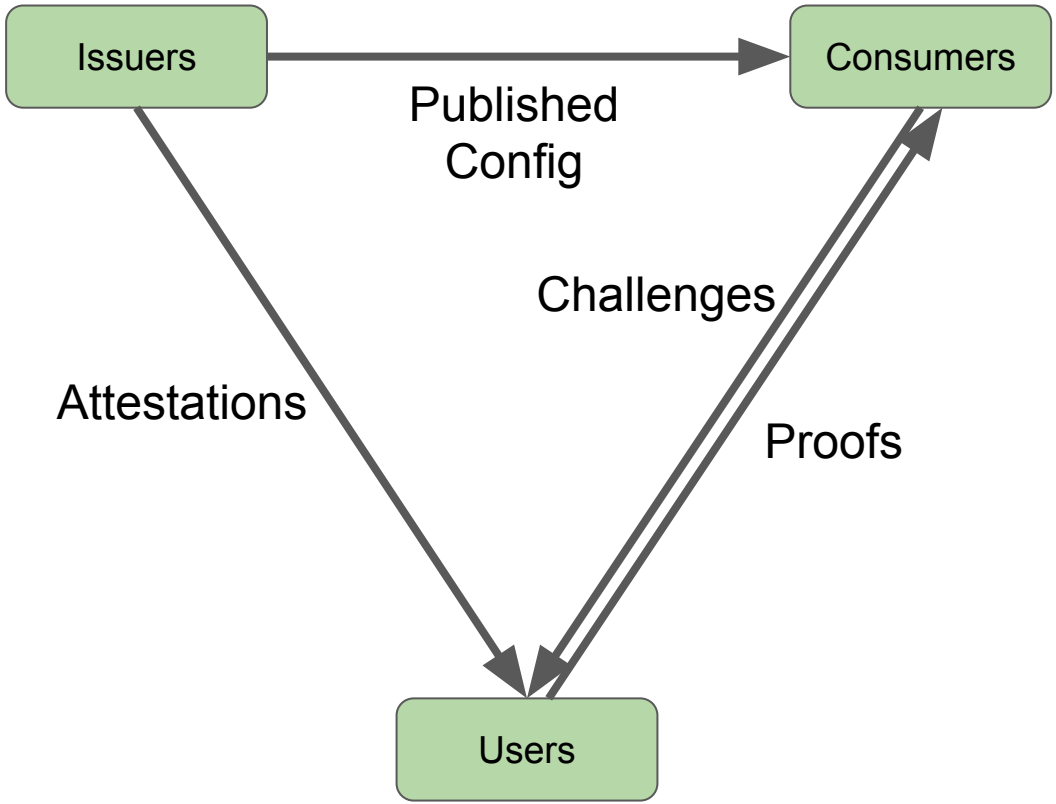
Programmable Cryptography Internet (PARCNET)

- Decentralized
- Self-custodial
- Privacy-preserving
- Permissionless
- Trustworthy

- ZK Proofs are only the beginning
  - … but are today's focus

https://0xparc.org/blog/programmable-cryptography-1

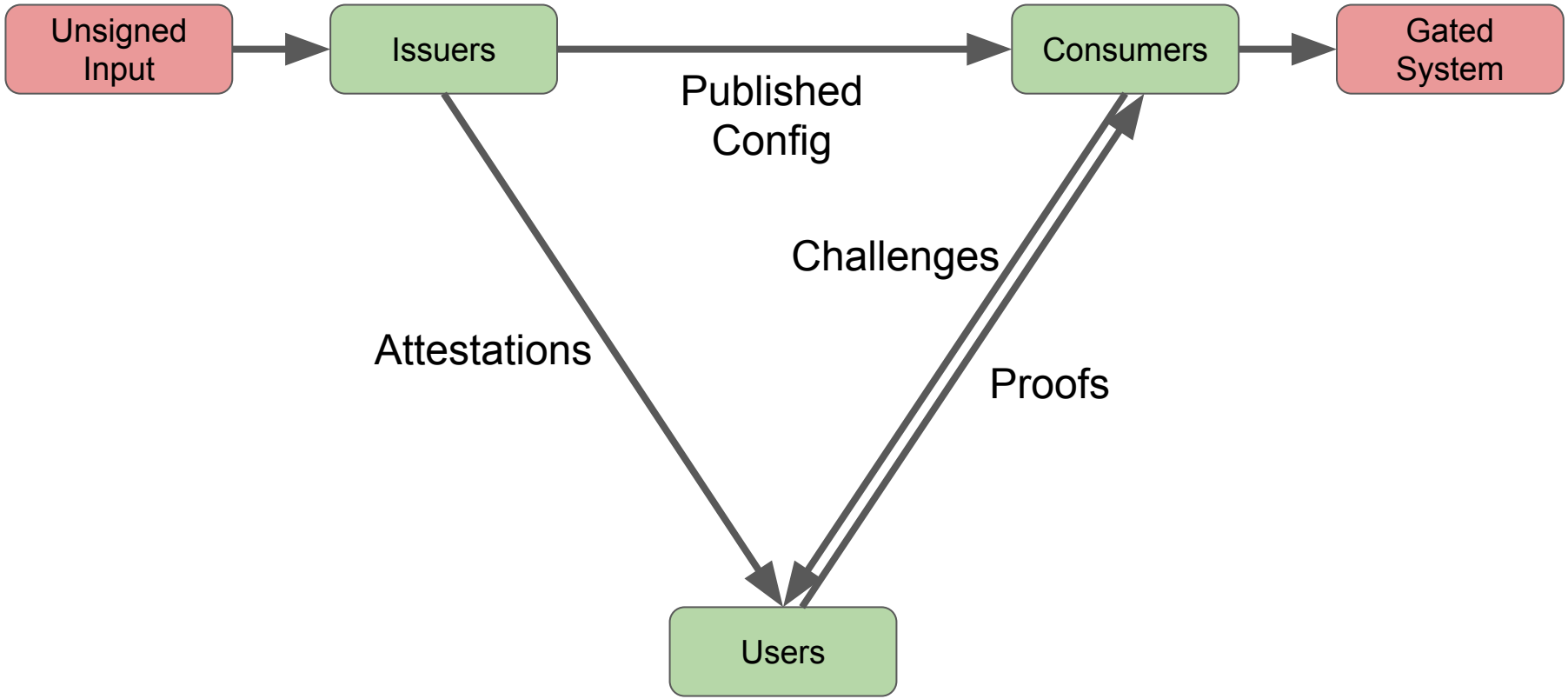
Programmable Cryptography Ecosystem



Zupass = ZK For Everyone!

- The best learning comes from “contact with reality”
- Early adopters are willing to try new tech
- Devcon is where we battle-test
- Onboard users and data by bridging non-ZK systems

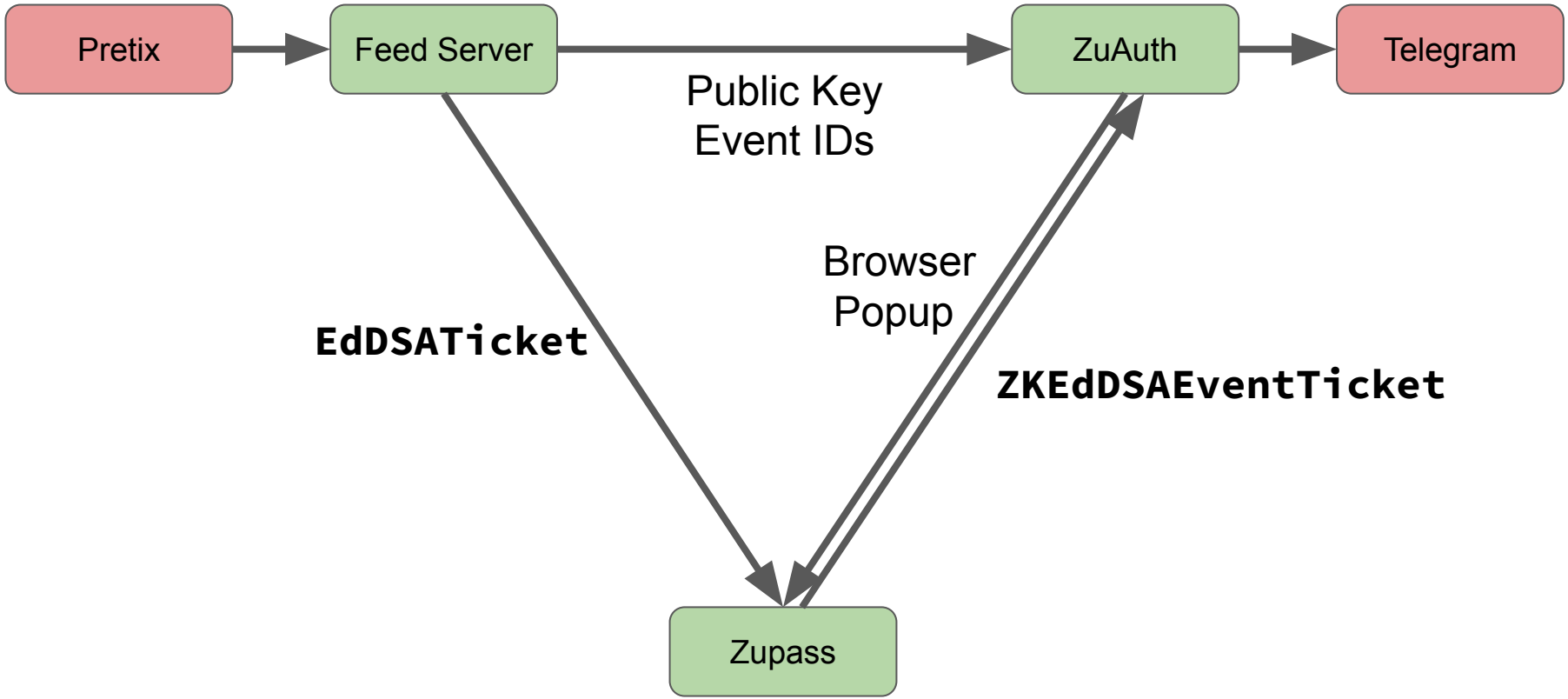
Bridged Ecosystem



Technical Constraints

- Cryptography all users can access: mobile-friendly web app
- Data is portable to any platform, including on-chain verification
- Tried and true technologies: **Circom, Groth16**
- Must be performant **in a browser, on a phone**
    - Even older phones, on bad networks

Zupass Tickets (Devconnect 2023)



ticket pcd #302

Edit <> Code

Merged **ichub** merged 35 commits into `main` from `ivan/ticket-pcd` on Jul 14, 2023

+1,131 −252

Add ZKEdDSAEventTicketPCD #574

Edit <> Code

Merged **artwyman** merged 17 commits into `main` from `artwyman/multi-event-ticket` on Sep 16, 2023

+5,638 −147

FrogCrypto

FROGCRYPTO

get frogs | frogedex

search SWAMP

search JUNGLE

search DESERT

search THE CAPITAL is closed

search CELESTIAL POND is closed

search THE WRITHING VOID is closed

Sort: A Z

#32 Malagasy Rainbow Frog

View as proof-carrying data

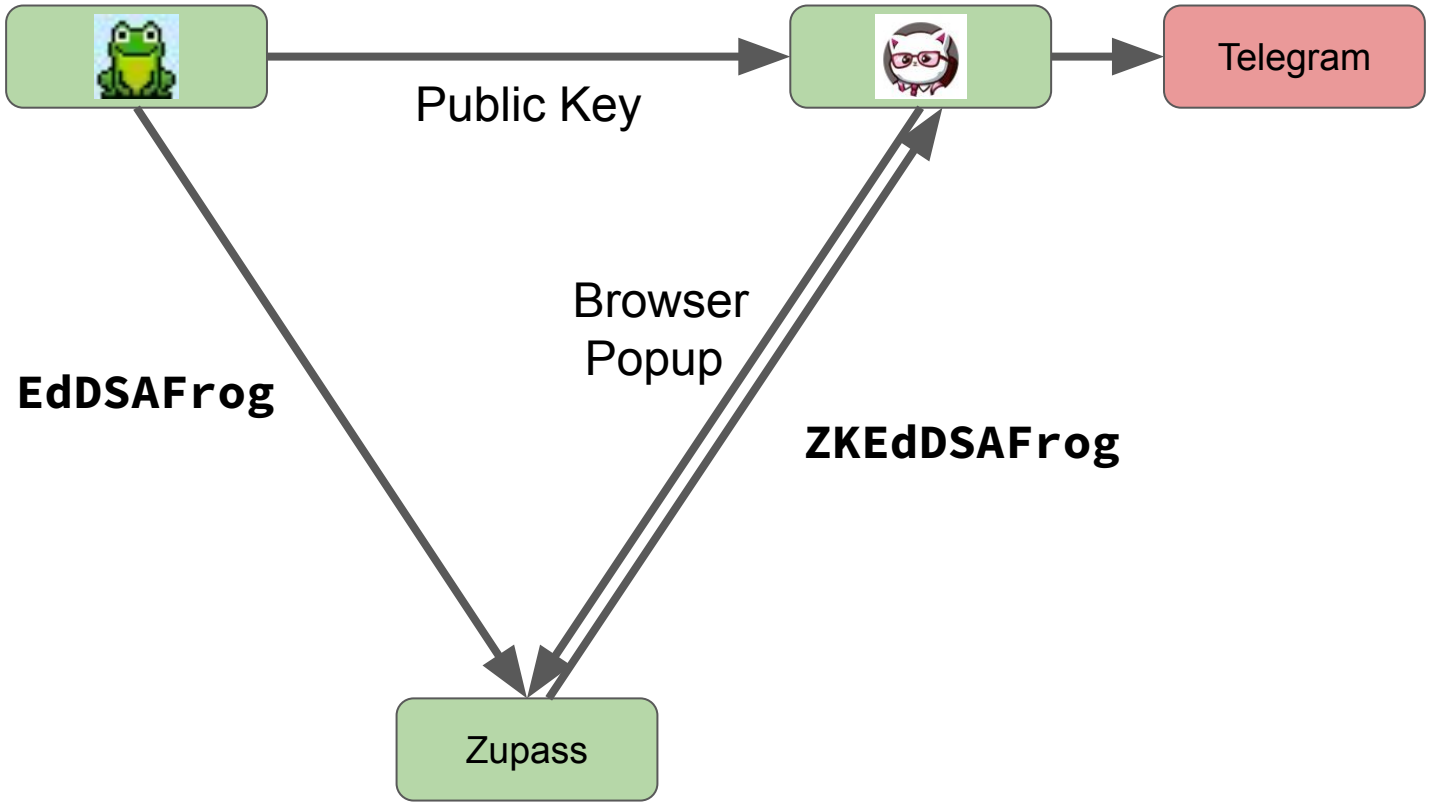
JMP	VIB	SPD	INT	BTY	EXP
04	SADG	07	06	03	01

See more

Remove

#15 Tiger-Legged Monkey Frog

Frogcrypto (Devconnect 2023)



[frogcrypto][1/n] frog pcd package #1058

Edit Code

Merged **ichub** merged 2 commits into `main` from `forestfang--frog-pcd` on Oct 25, 2023

+1,699 −0

Add ZK EdDSA Frog PCD #1162

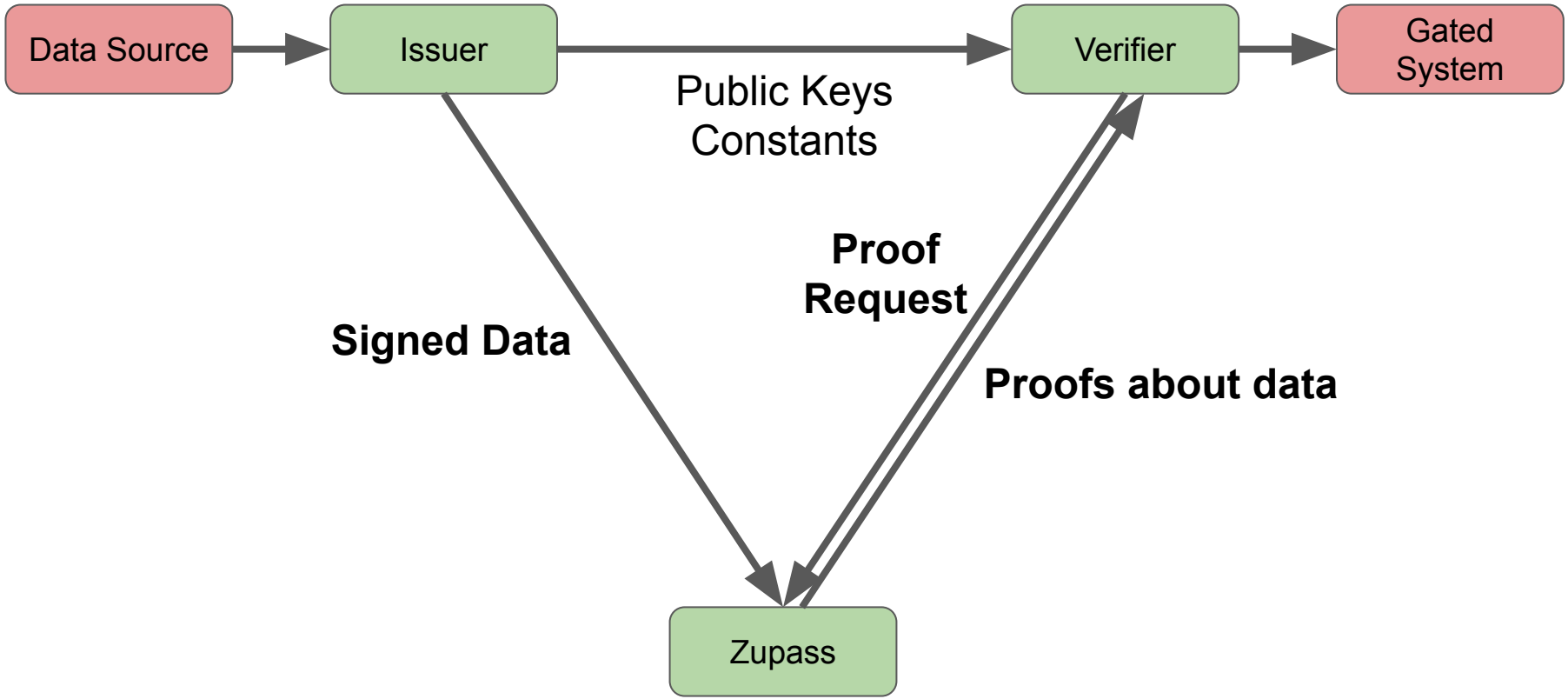
Edit Code

Merged

saurfang merged 8 commits into `proofcarryingdata:main` from `veronicaz41:vzheng/frog-pcd` on Nov 10, 2023

+1,816 −5

A pattern emerges…



Why is this so much effort?

Signed Data

- EdDSAPCD is an array of bigints
- Fixed size hash, max 16 values
- Custom coding for every new datatype

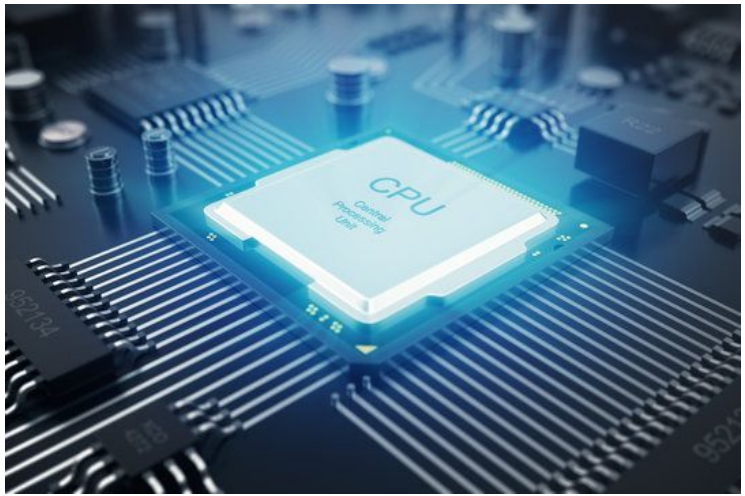
```
00000010 01 E0 00 00 02 80 00 08  à   €
00000018 00 03 00 00 00 00 00 00
00000020 52 8E 38 42 49 4D 04 04 RŽ8BIM
00000028 00 00 00 00 00 07 1C 02
00000030 00 00 02 00 02 00 38 42       8B
00000038 49 4D 04 25 00 00 00 00 IM %
00000040 00 10 46 0C F2 89 26 B8   F ò%&¸
00000048 56 DA B0 9C 01 A1 B0 A7 VÚ°œ ¡°§
00000050 90 77 38 42 49 4D 04 24  w8BIM $
00000058 00 00 00 00 39 15 3C 3F     9 <?
```

Proofs

- ZK Circuits are awkward to program
  - Each variable is a field element (number mod p, 254 bits)
- ZK Circuits are hard to reuse
  - Fixed inputs
  - Fixed logic
- Trusted setup makes updates costly

So we need a ZKVM?

- Run code inside a zkSNARK
- Problem solved!



So we need a ZKVM?

- Yes, but not feasible yet **in a browser, on a phone**

We can engineer for generality

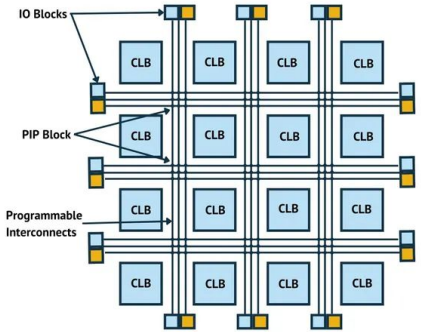
Provable Object Data (POD)

- Arbitrary names and values
- Multiple data types
  - int, string, boolean, date, …
- Cryptographically signed
- Easy to make proofs about

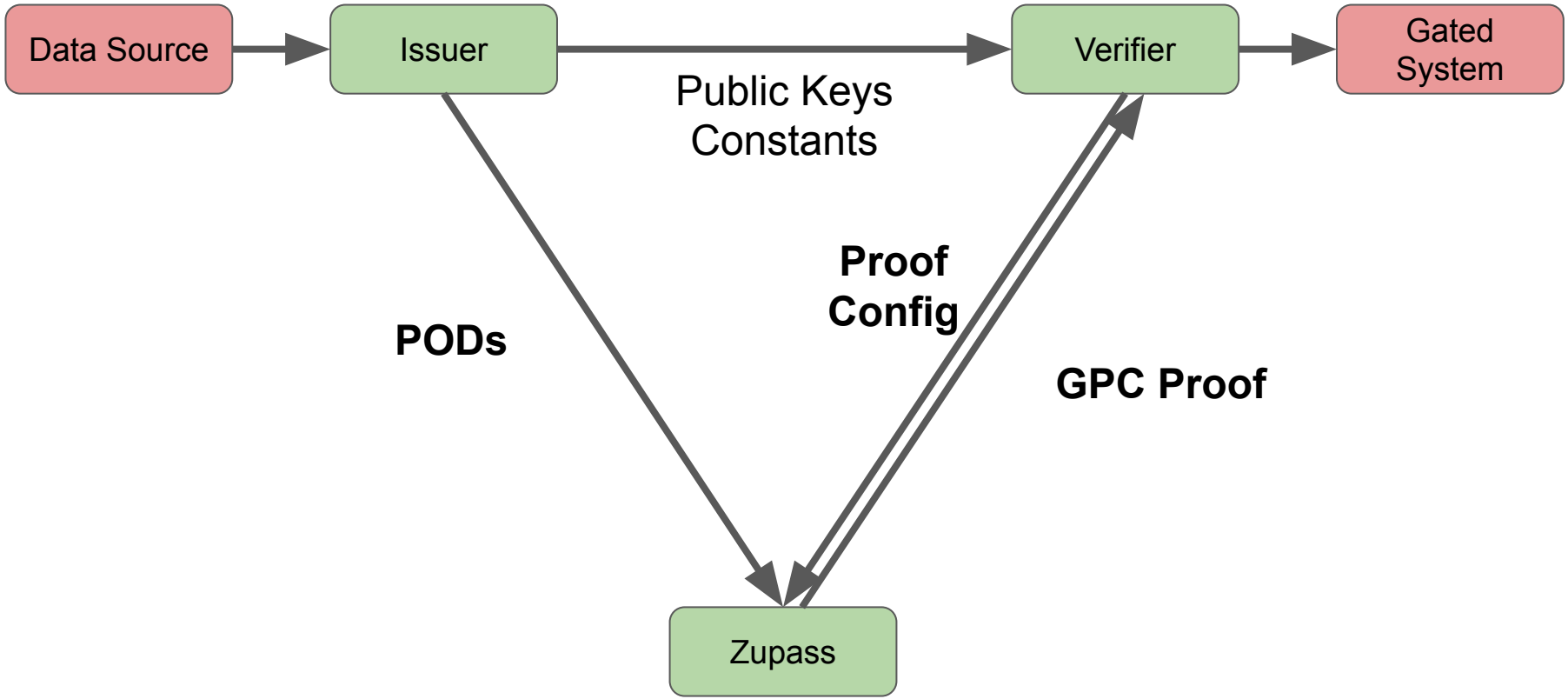
```
{
    "count": 7,
    "items": ["socks", "pants", "shirts", "hats"],
    "manufacturer": {
        "name": "Molly's Seamstress Shop",
        "id": 39233,
        "location": {
            "address": "123 Pickleton Dr.",
            "city": "Tucson",
            "state": "AZ",
            "zip": 85705
        }
    },
    "total_price": "$393.23",
    "purchase_date": "2022-05-30",
    "country": "USA"
}
```

General Purpose Circuit (GPC)

- Reusable circuit blocks (modules)
- Public inputs configure interconnections
- Pick from a family of circuits, each with a mix of modules
- Config compiler → circuit inputs



POD Ecosystem



What is a POD?

- A data format which makes ZK proofs easy

- Key/Value store
- Hashed and signed
- Optimized for efficient ZK proving

POD Entries

Name	**Value**	**Type Hint**
name	Filip Frog	string
date_of_birth	March 20, 1999	date
cardholder	Semaphore ID	eddsa_pubkey
postcode	94107	int
driver	true	boolean
pod_type	`dmv.license`	string


```
{
  "cardholder": {
    "eddsa_pubkey": "c433f7a696b7aa3a5224..."
  },
  "date_of_birth": {
    "date": “1999-03-20T00:00:00.000Z”
  },
  "driver": true,
  "name": “Filip Frog”,
  "pod_type": "dmv.license",
  "postcode": 94107
}
```

Hashed (Merklized) POD

Name	**Value**	**Type Hint**
name	Filip Frog	string
date_of_birth	March 20, 1999	date
cardholder	Semaphore ID	eddsa_pubkey
postcode	94107	int
driver	true	boolean
pod_type	`dmv.license`	string


```
{
  "cardholder": {
    "eddsa_pubkey": "c433f7a696b7aa3a5224..."
  },
  "date_of_birth": {
    "date": “1999-03-20T00:00:00.000Z”
  },
  "driver": true,
  "name": “Filip Frog”,
  "pod_type": "dmv.license",
  "postcode": 94107
}
```


Content ID = 0x2b11d47db364c7e64c...

Signed POD

Name	**Value**	**Type Hint**
name	Filip Frog	string
date_of_birth	March 20, 1999	date
cardholder	Semaphore ID	eddsa_pubkey
postcode	94107	int
driver	true	boolean
pod_type	`dmv.license`	string


```
{
  "entries": {
    "cardholder": {
      "eddsa_pubkey": "c433f7a696b7aa3a5224...
    },
    "date_of_birth": {
      "date": “1999-03-20T00:00:00.000Z”
    },
    "driver": true,
    "name": “Filip Frog”,
    "pod_type": "dmv.license",
    "postcode": 94107
  },

  "signature": "kKt/qddVepEm1Q+hCa34...",

  "signerPublicKey": "NnGAciO/OIz+R5...",
}
```


Content ID = 0x2b11d47db364c7e64c...

What is a GPC?

- A system for proving anything about PODs

- Circuits made of reusable modules
  - Config controls how they are connected
- Pick from a family of pre-compiled circuits
  - Circuit size (cost) depends on complexity of config

Proof Configuration: just reveal

```
{
  "pods": {
    "idcard": {
      "entries": {
        "driver": { "isRevealed": true }
      }
    }
  }
}
```

- Everything else is hidden
- Only the configured entries are proven - any others may not exist

Proof Configuration: remember to check signer

```
{
  "pods": {
    "idcard": {
      "entries": {
        "driver": { "isRevealed": true }
      },
      "$signerPublicKey": { "isRevealed": true }
    }
  }
}
```

- Signer’s public key is revealed by default
  - Verifier must check who they trust
- Can also be constrained like an entry
  - E.g. signer is in a list, or 2 PODs have the same signer

Proof Configuration: check ownership

```
{
  "pods": {
    "idcard": {
      "entries": {
        "driver": { "isRevealed": true },
        "cardholder": { "isRevealed": false, "isOwnerID": "SemaphoreV4" }
      }
    }
  }
}
```

- Ownership checks the prover has the private key corresponding to the pubkey in the POD

Proof Configuration: Range Check

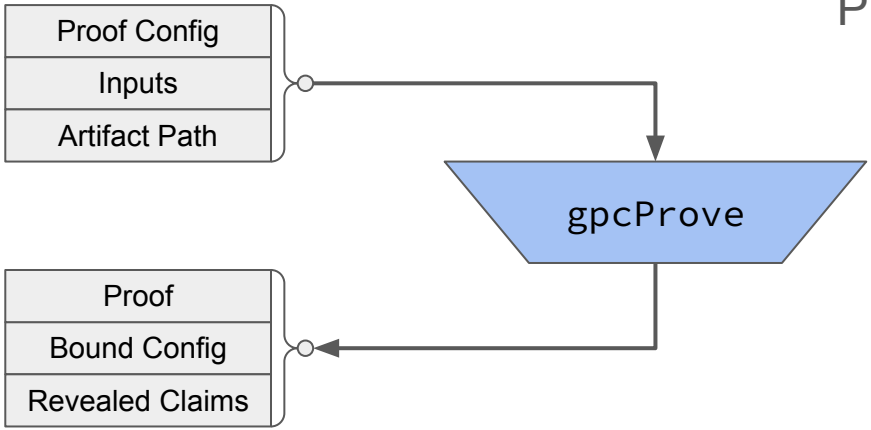
```
{
  "pods": {
    "idcard": {
      "entries": {
        "date_of_birth": {
          "isRevealed": false,
          "inRange": { "min": 0, "max": 1068104558283 }
        },
        "cardholder": { "isRevealed": false, "isOwnerID": "SemaphoreV4" }
      }
    }
  }
}
```

- Date range to prove the holder is over 21

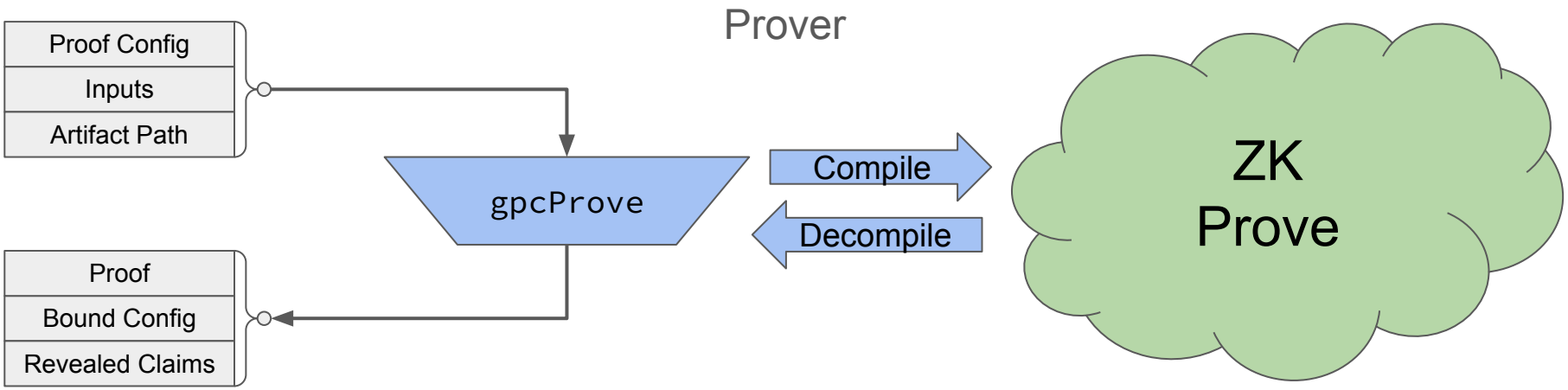
Proof Configuration: Multiple PODs

```
{
  "pods": {
    "idcard": {
      "entries": {
        "date_of_birth": {
          "isRevealed": false,
          "inRange": { "min": 0, "max": 1068104558283 }
        },
        "cardholder": { "isRevealed": false, "isOwnerID": "SemaphoreV4" },
        "name": { "equalsEntry": "ticket.attendee" }
      }
    },
    "ticket": {
      "entries": {
        "attendee": { "isRevealed": false },
        "eventID": { "isRevealed": false, isMember: "devconEvents" },
        "owner": { "isRevealed": false, "isOwnerID": "SemaphoreV4" }
      }
    }
```

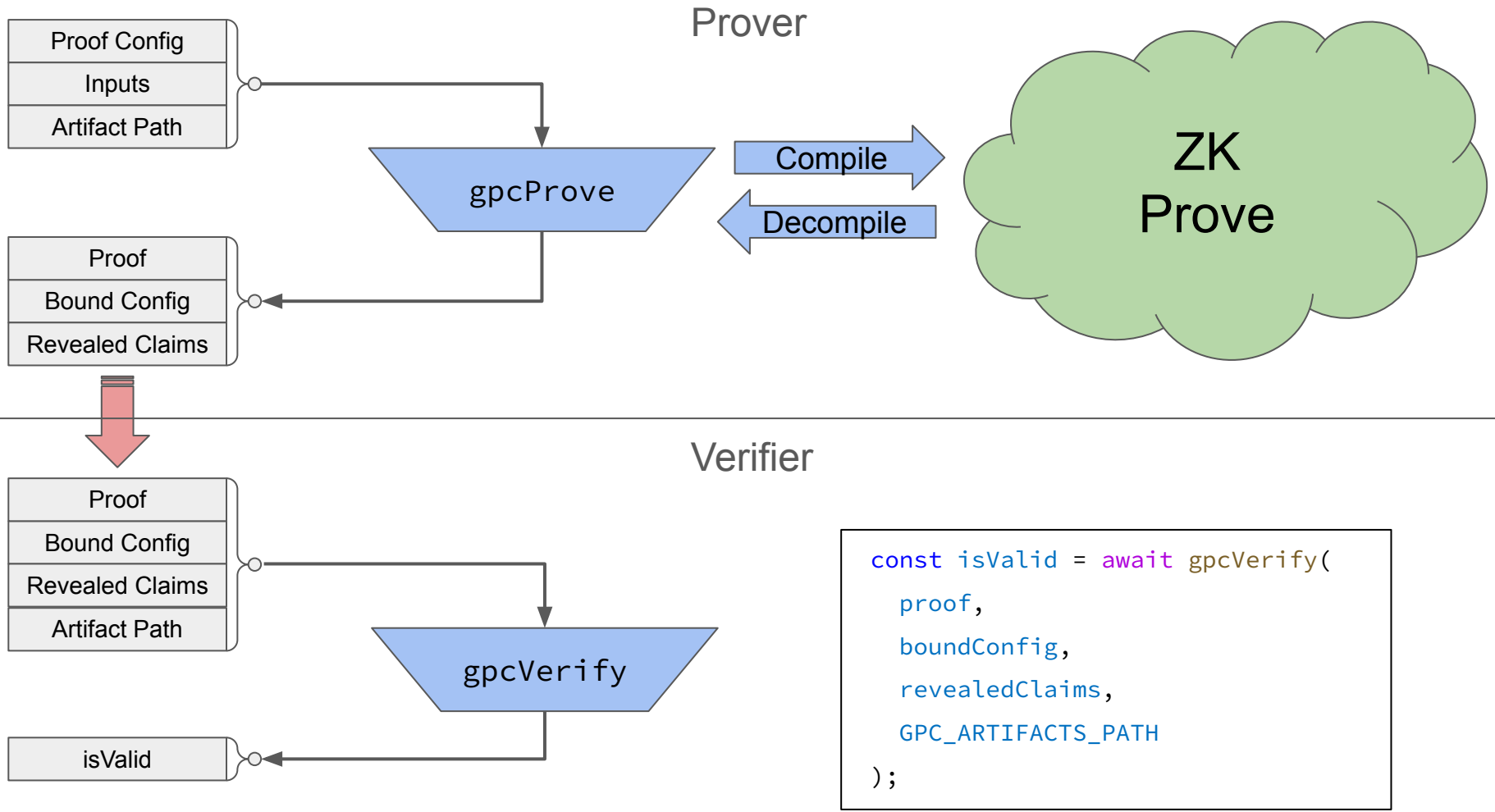
Prover



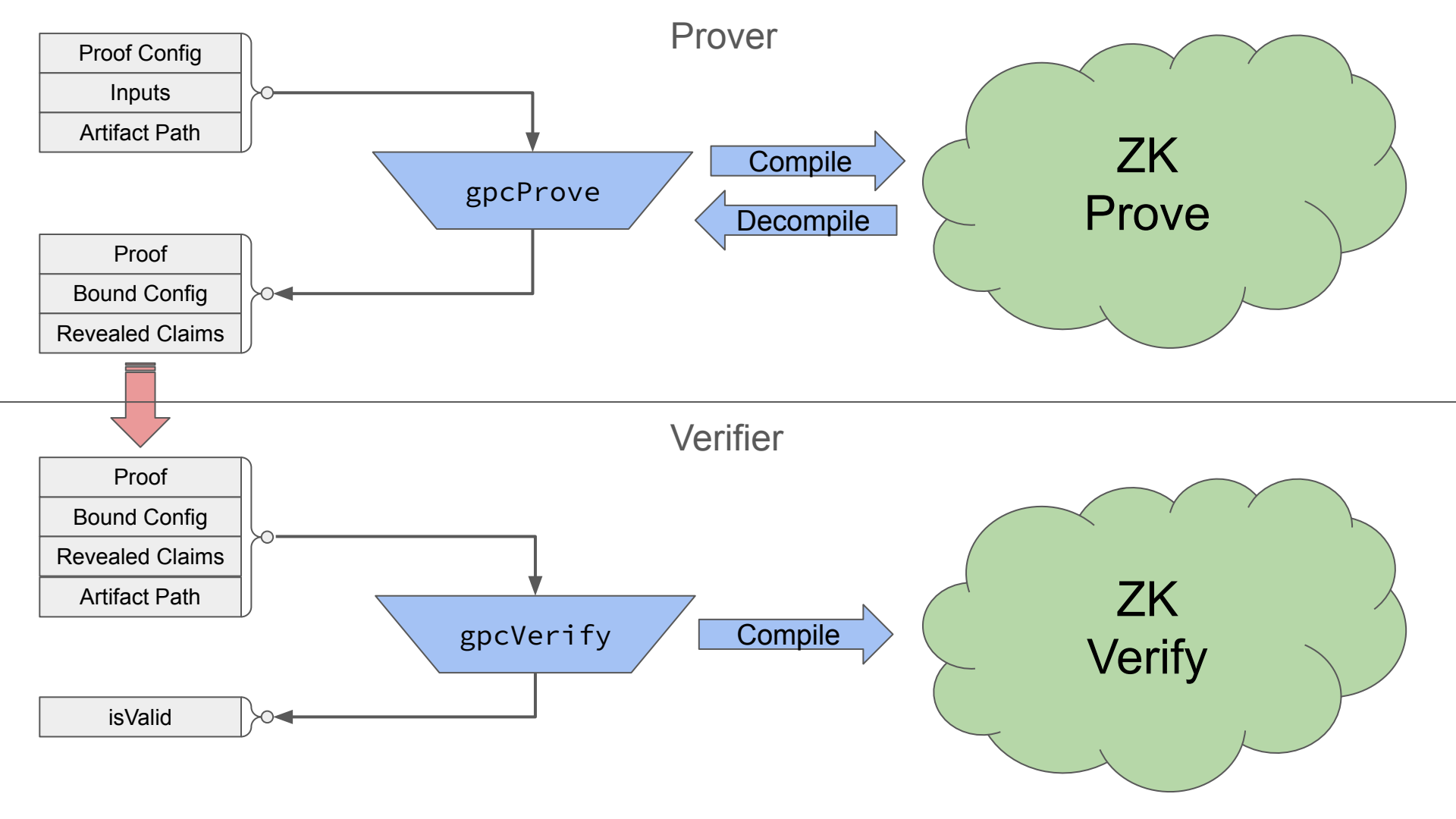

```
const { proof, boundConfig, revealedClaims } = await gpcProve(
  proofConfig,
  proofInputs,
  GPC_ARTIFACTS_PATH
);
```




```
const { proof, boundConfig, revealedClaims } = await gpcProve(
  proofConfig,
  proofInputs,
  GPC_ARTIFACTS_PATH
);
```




```
const isValid = await gpcVerify(
  proof,
  boundConfig,
  revealedClaims,
  GPC_ARTIFACTS_PATH
);
```


Prover
Proof Config
Inputs
Artifact Path
gpcProve
Compile
Decompile
ZK
Prove
Proof
Bound Config
Revealed Claims
Verifier
Proof
Bound Config
Revealed Claims
Artifact Path
gpcVerify
Compile
ZK
Verify
isValid

Takeaways

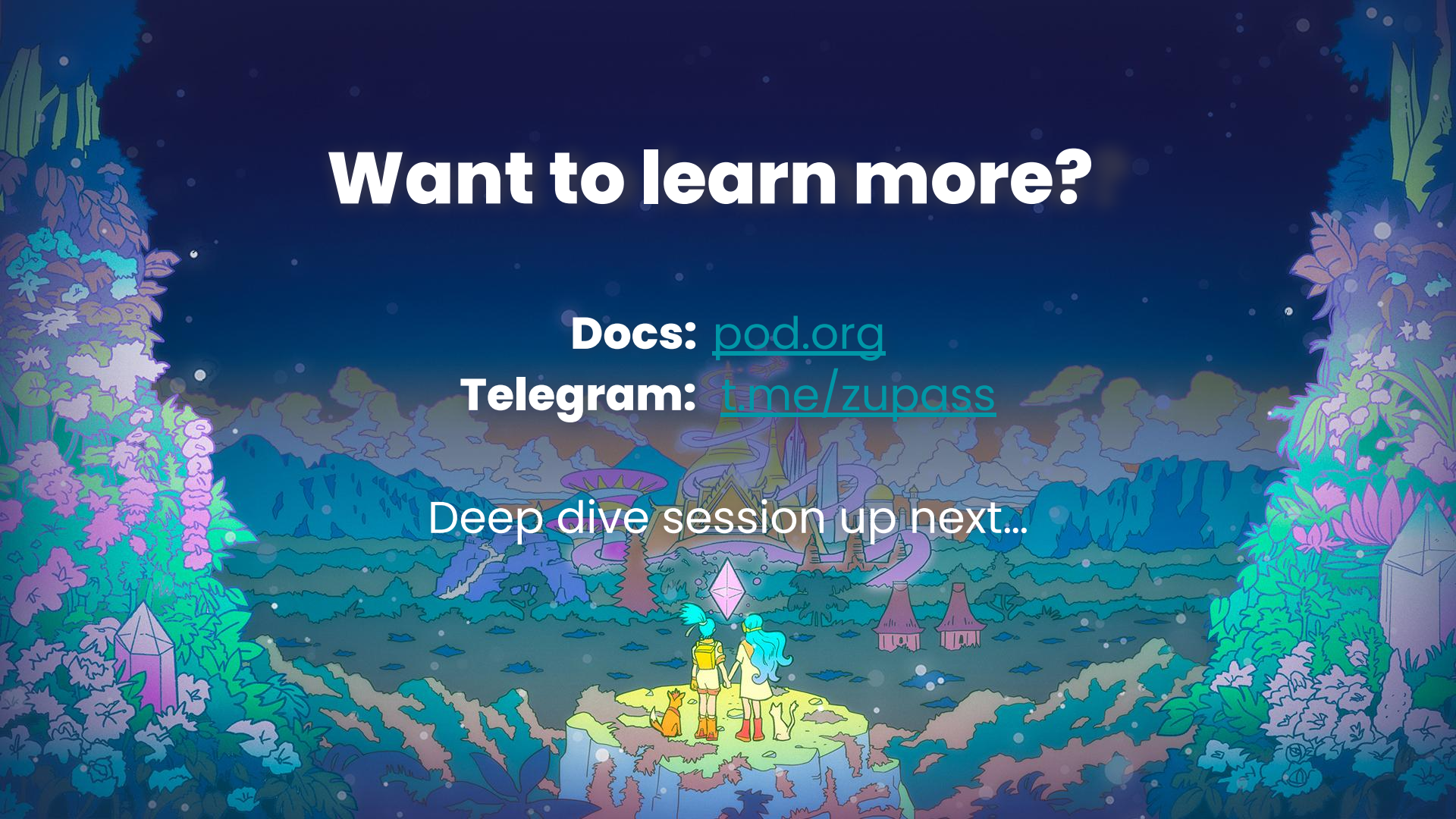
- PODs are data designed for proving
  - A signed attestation
- GPCs allow flexible proving
  - Modular circuits can be configured
  - Auto-select a circuit to fit your config
- Apps decide what to trust
  - Trusted signers
  - Published schemas/constants

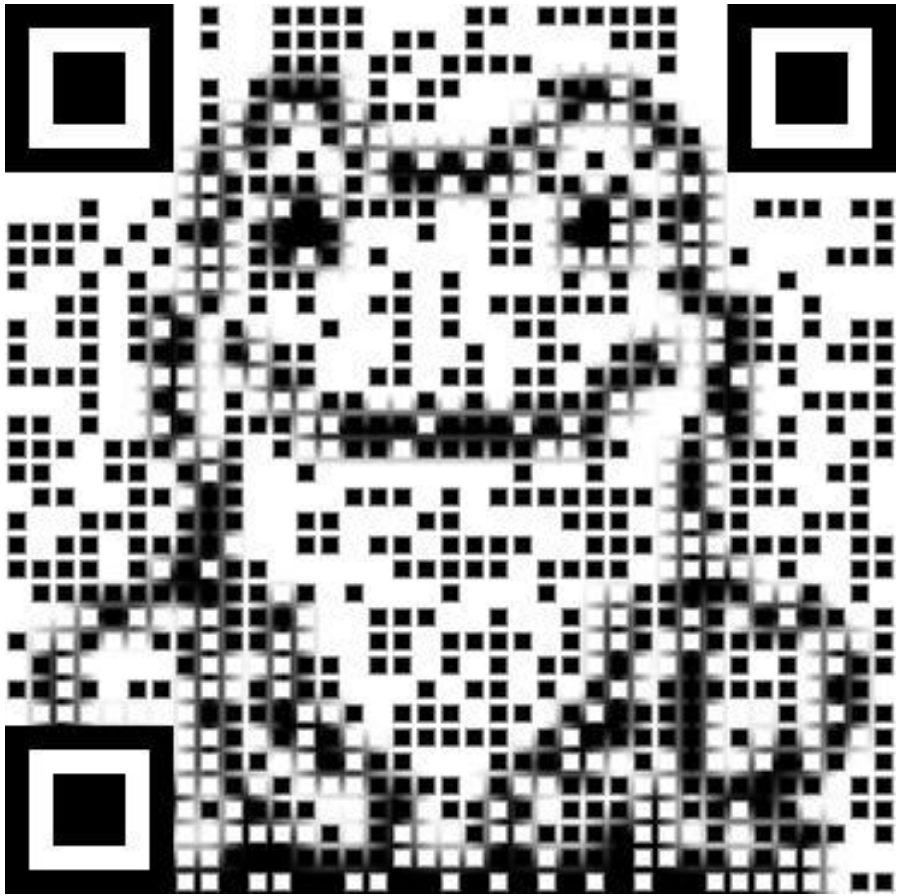
Want to learn more?

Docs: pod.org

Telegram: t.me/zupass

Deep dive session up next...





https://dc7.getfrogs.xyz/scanner/FrogPOD