zk-creds

Flexible Anonymous Credentials from zkSNARKs and Existing Identity Infrastructure

Michael Rosenberg
University of Maryland
micro@umd.edu

Jacob White
Purdue University
white570@purdue.edu

Christina Garman
Purdue University
clg@purdue.edu

Ian Miers
University of Maryland
imiers@umd.edu

https://ia.cr/2022/878

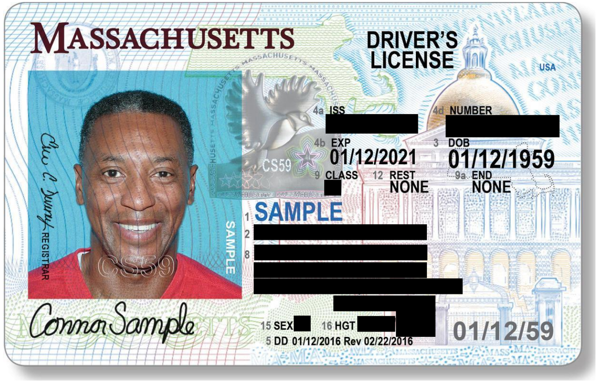
https://github.com/rozbb/zkcreds-rs

(Adapted from IEEE S&P 2023)

Anonymous Credentials

1. Prove something about yourself ("I'm over 18")

2. ... without revealing anything else (name, address, etc.)



Anonymous Credentials

Example 1

Serving **age-restricted videos** using photo ID or credit card (Utah, Louisiana, EU laws)

Example 2

Preventing spam / DoS using PrivacyPass-like tokens

Anonymous Credentials

Many anonymous credential schemes exist
[Cha85, CL01, CL03, CL04, CHK+06, BCKL08, CG08, BL13, GGM14, CDHK15, SAB+19]

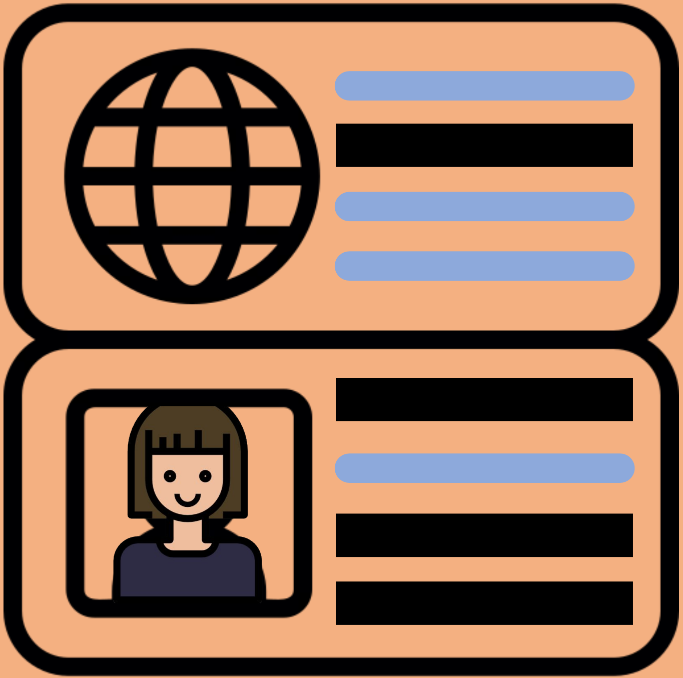
But each:

- Requires the govt. to issue exotic new credentials digitally
- Requires cryptographers to design a custom protocol for each new use case

zk-creds

A practical system must:

1. Support existing identity documents
2. Not require new trusted parties for issuance
3. Be easily programmable for new use cases



zk-creds uses SNARKs to get all of these

Background: zkSNARKs

Zero-Knowledge Succinct Non-interactive ARgument of Knowledge:

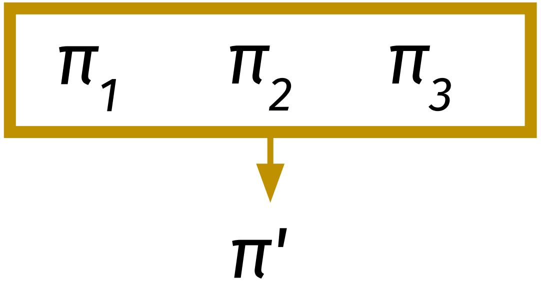
- **ZK.** Can prove "I know *x* such that *P*(*x*, aux)" where aux is public
- **Succinct.** That proof π is the same size no matter how large or complex *P* is
- **Non-interactive.** Verify(π, aux) is constant time*

Example: Unlinkable signatures

"I know σ such that $\text{SigVerif}_{pk}(\sigma, m)$"

New

LinkG16 extends Groth16 zkSNARKs, letting us **package together and reuse** multiple proofs:



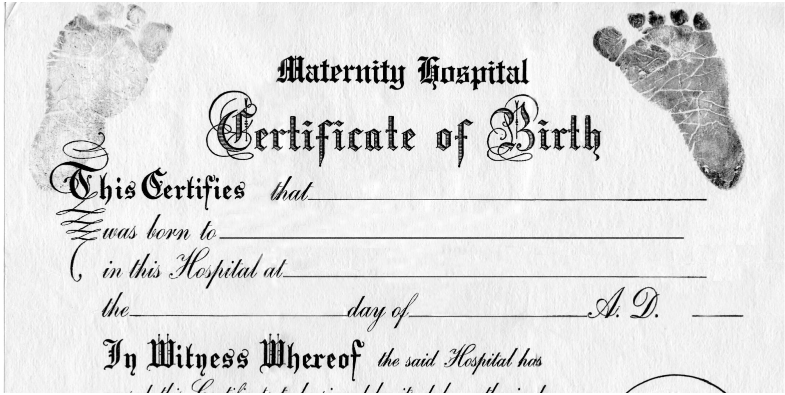
Supporting existing identity documents

Showing (e.g.) age requires govt. ID or other source of identity.

Existing creds schemes assume that the govt. or other trusted third party will issue your cred.

Observation: some govt. IDs have **non-anonymous** digital IDs inside an RFID chip.

Furthermore, these IDs are **signed** by the govt. itself



Maternity Hospital

Certificate of Birth

This Certifies that ______

was born to ______

in this Hospital at ______

the ______ day of ______ A. D. ______

In Witness Whereof the said Hospital has

Idea

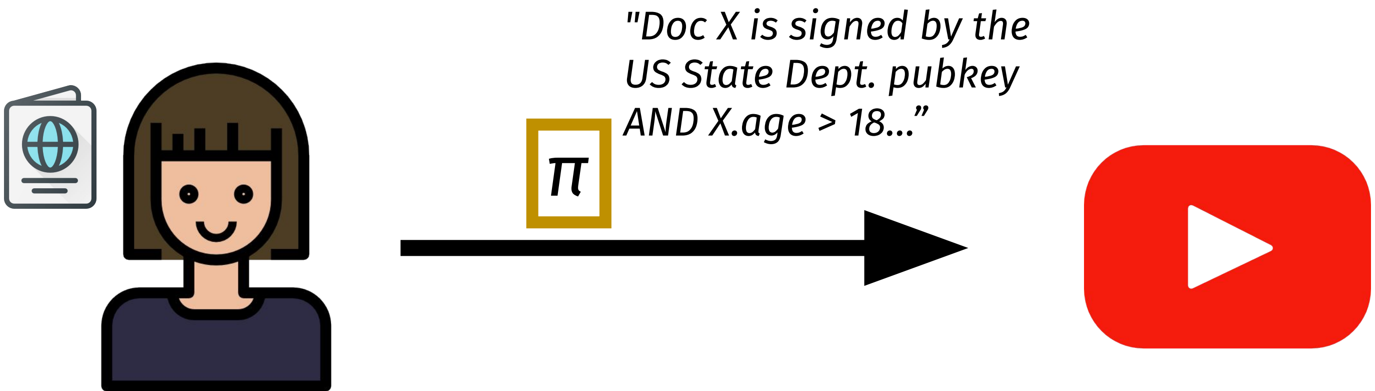
Let's bootstrap an anonymous system on top of this non-anonymous one!

Supporting existing identity documents

Use zero-knowledge proofs (zkSNARKs) to:

- Prove an ID is signed by the govt.
- Prove other details to access service

Privacy ✅ Authenticity ✅



"Doc X is signed by the US State Dept. pubkey AND X.age > 18…"

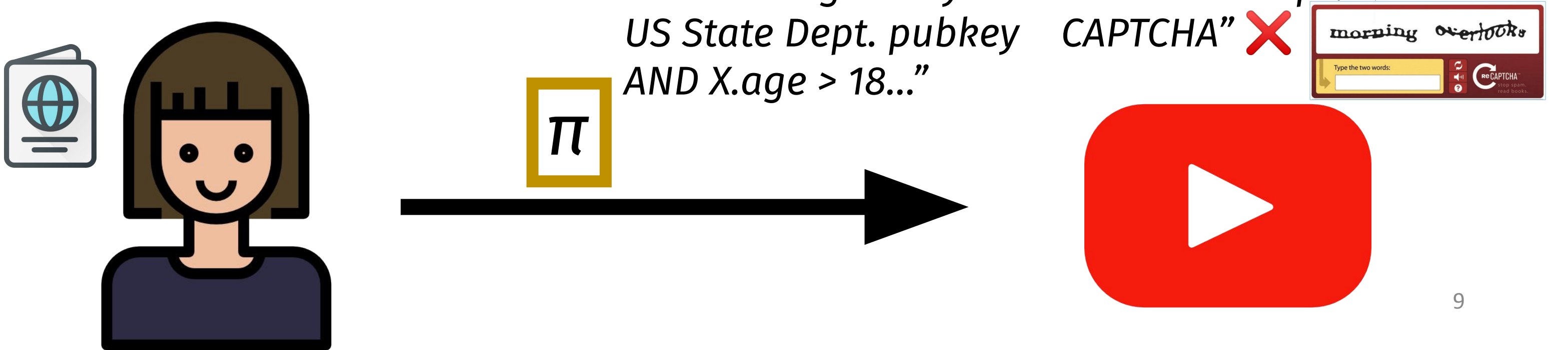
Supporting existing identity documents

Use zero-knowledge proofs (zkSNARKs) to:

- Prove an ID is signed by the govt.
- Prove other details to access service

Privacy ✅ Authenticity ✅

Problem
Credentials often need more info than a single document

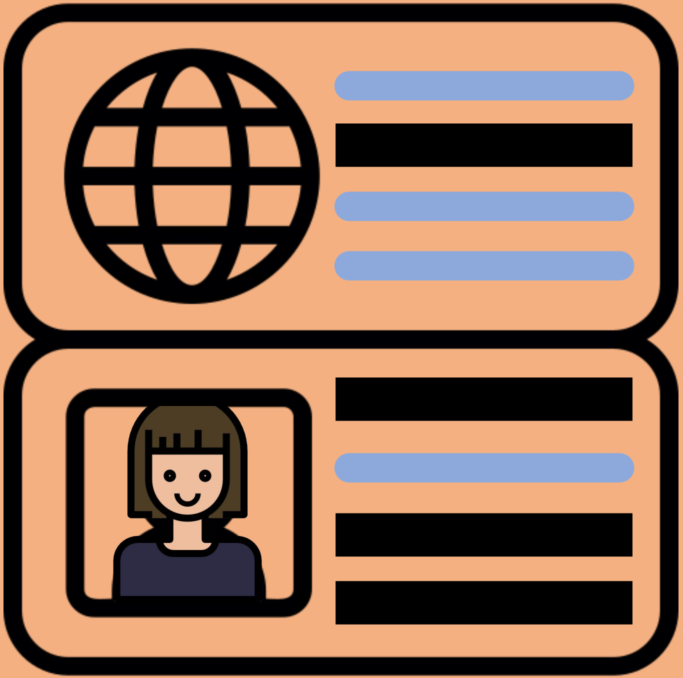


"Doc X is signed by the US State Dept. pubkey AND X.age > 18…"

"… AND I completed a CAPTCHA" ❌

zk-creds

A practical system must:



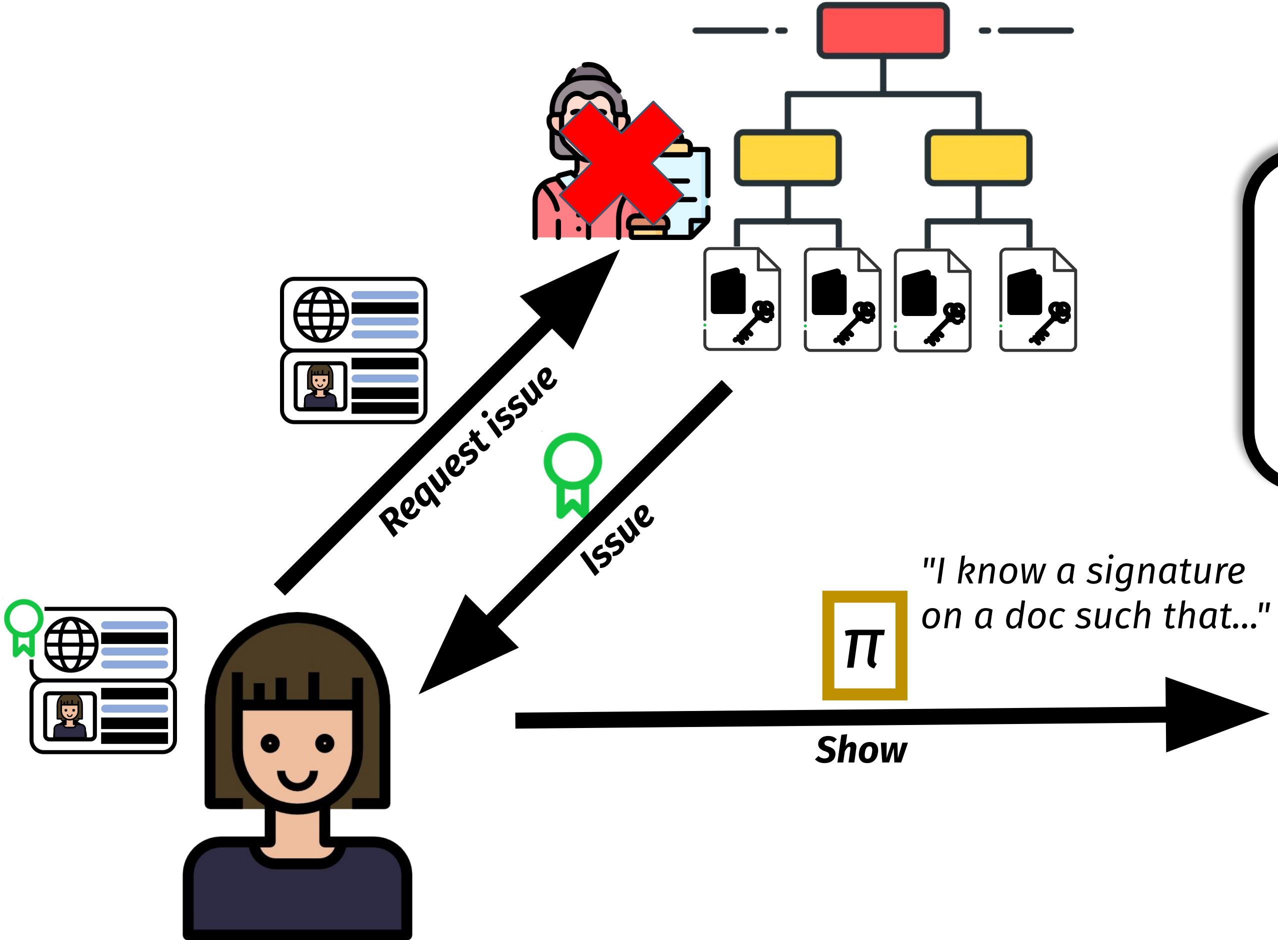
1. Support existing identity documents

2. Not require new trusted parties for issuance

3. Be easily programmable for new use cases

Anonymous credentials

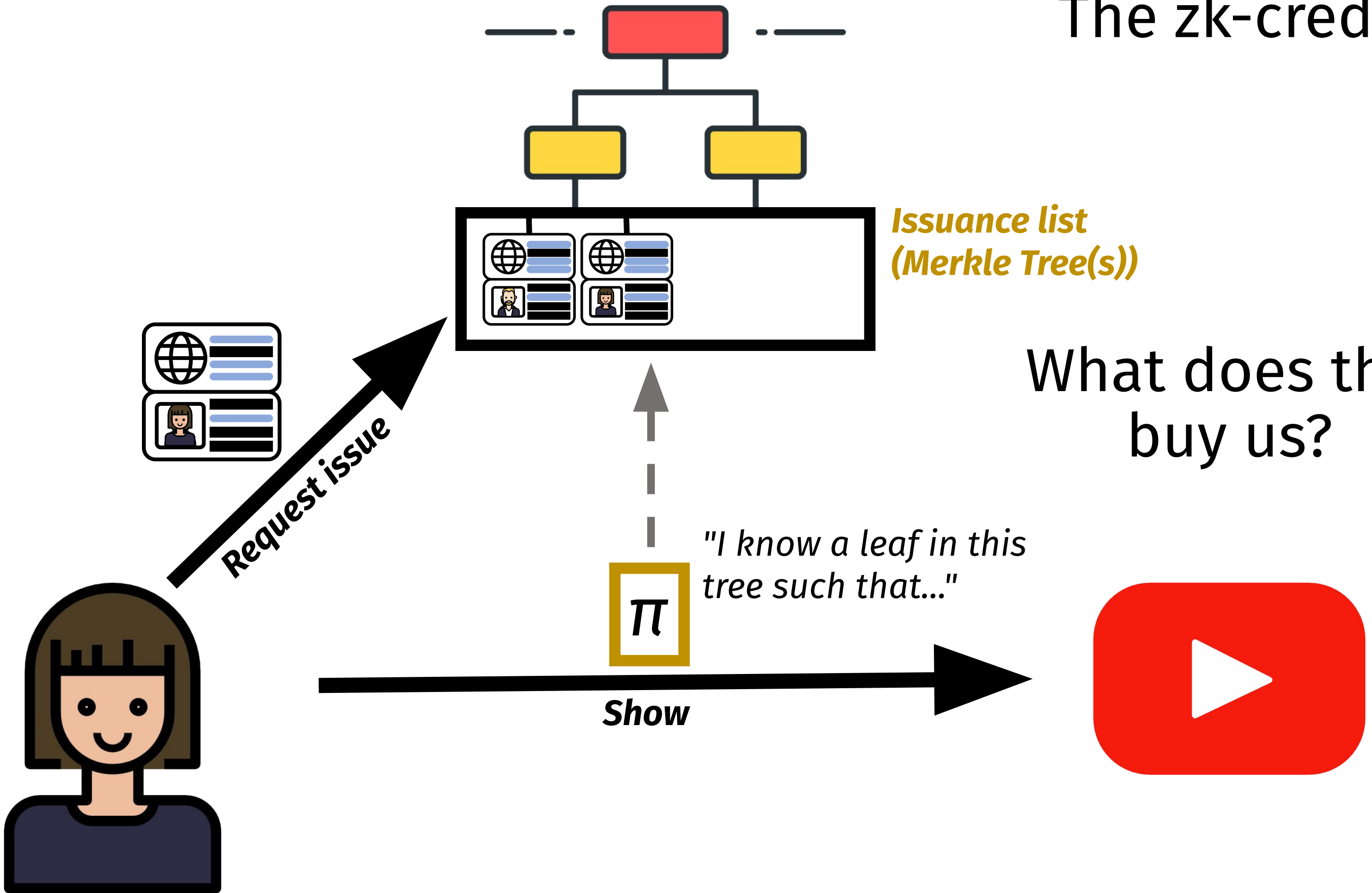
The usual flow



Problem
We want *fewer*, not more, trusted third parties

Anonymous credentials

The zk-creds flow

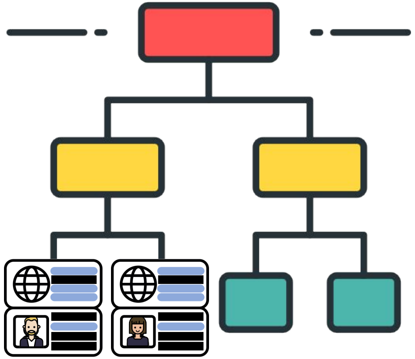


What does this buy us?

zk-creds

When to issue, when to reject

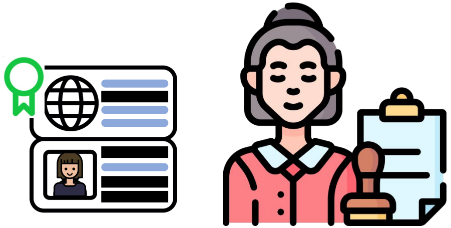
To get an issued credential, a user might need to give extra information to the issuer. We call this **zk-supporting documentation**



To request issuance in zk-creds, the user provides:

1. A credential
2. zk-supporting docs
3. A proof of correctness

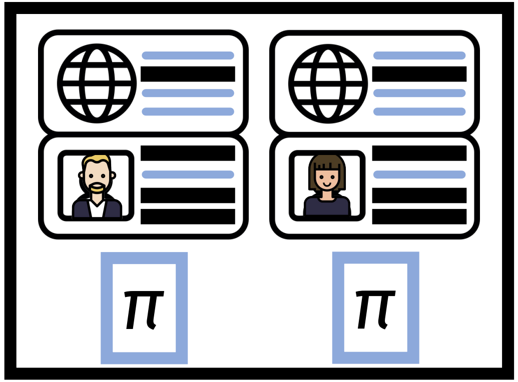
Transparent issuance



Previously: issuer could privately sign whatever they wanted

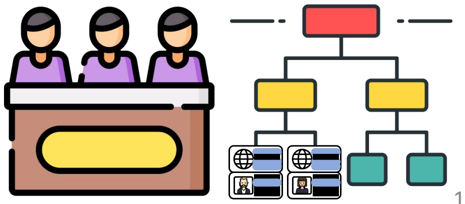
With a public list: we can now see what's on the list, and use **zk-supporting documents** to justify *why* it was issued

Issuance list



Issuance is now publicly auditable

Bonus: more ways to issue (threshold permissions, Byzantine consensus, blockchain, etc.)

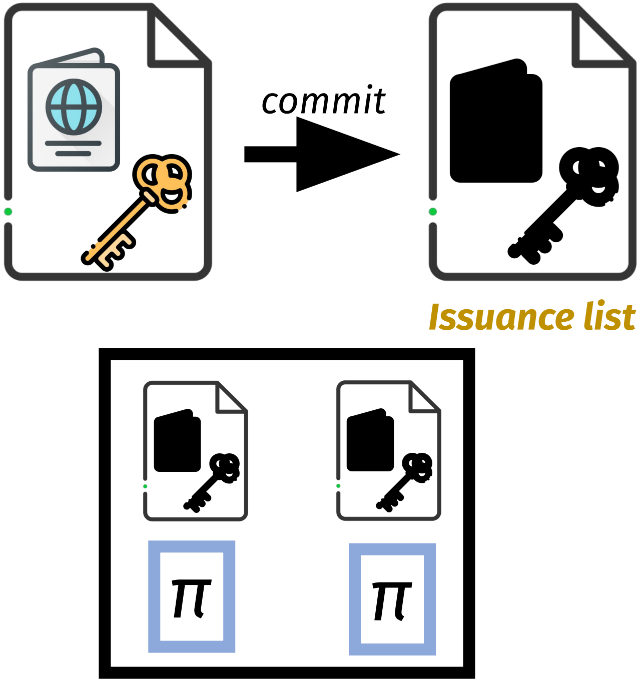


Transparent issuance

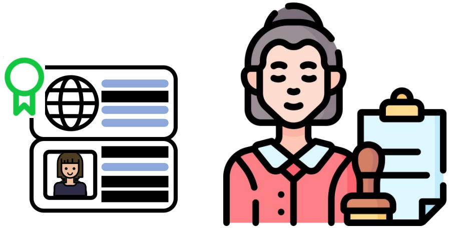
Credential’s attributes are still private

Issuance happens on **commitments** to ID, not ID themselves:

Zero-knowledge proofs during issuance and show means nothing extra is revealed across proofs

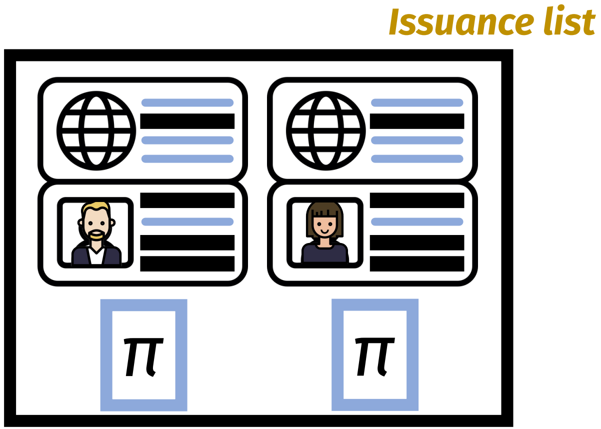


Flexible issuance



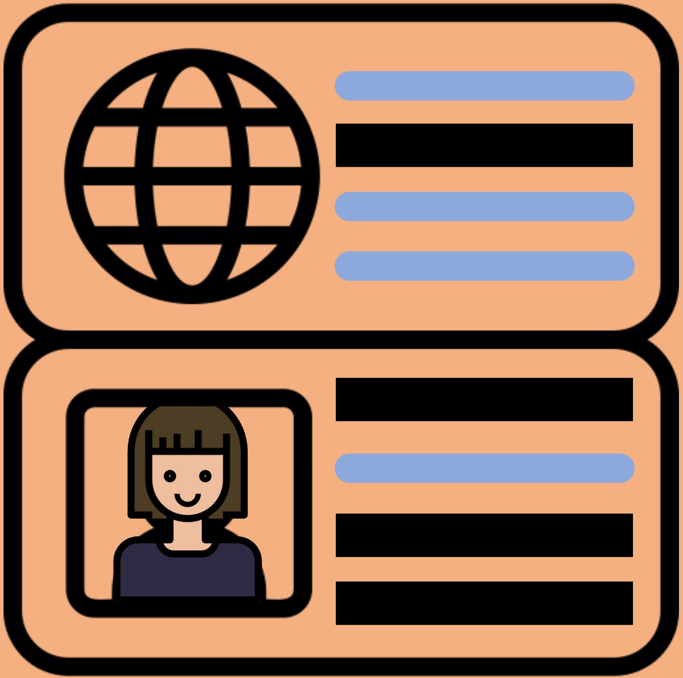
Another bonus: the proofs can be anything! No longer need to sign attributes in a bespoke manner

Can combine proof over cred and other information to argue for issuance



zk-creds

A practical system must:



1. Support existing identity documents
2. Not require new trusted parties for issuance
3. Be easily programmable for new use cases

Extending flexibility

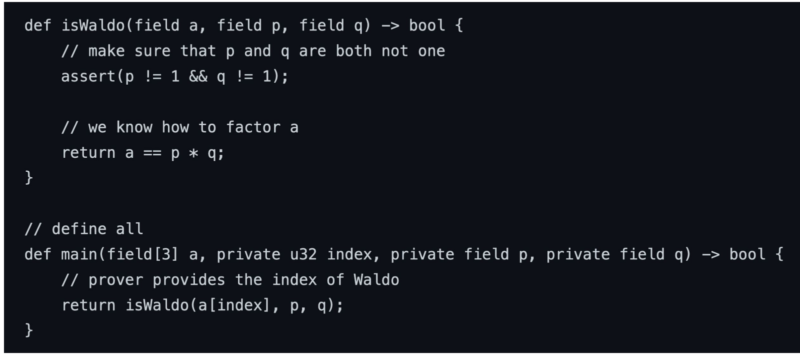
$\boxed{\pi}$ can be an **arbitrary** statement

Show

Huge ecosystem for SNARKs – many libraries for writing R1CS circuits, PLONK circuits, etc. for many SNARK protocols

Allows developers to write complex statements without being experts in cryptography

zokrates

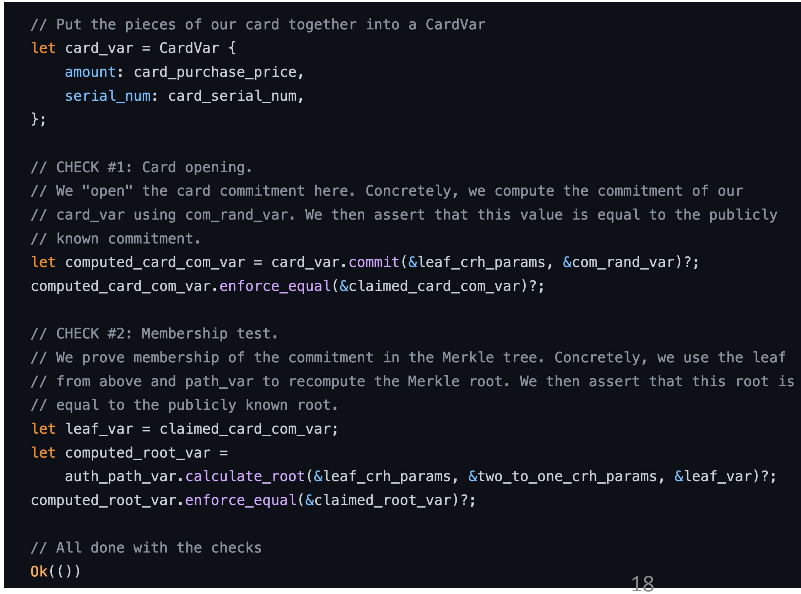


```
def isWaldo(field a, field p, field q) -> bool {
    // make sure that p and q are both not one
    assert(p != 1 && q != 1);

    // we know how to factor a
    return a == p * q;
}

// define all
def main(field[3] a, private u32 index, private field p, private field q) -> bool {
    // prover provides the index of Waldo
    return isWaldo(a[index], p, q);
}
```

arkworks



```
// Put the pieces of our card together into a CardVar
let card_var = CardVar {
    amount: card_purchase_price,
    serial_num: card_serial_num,
};

// CHECK #1: Card opening.
// We "open" the card commitment here. Concretely, we compute the commitment of our
// card_var using com_rand_var. We then assert that this value is equal to the publicly
// known commitment.
let computed_card_com_var = card_var.commit(&leaf_crh_params, &com_rand_var)?;
computed_card_com_var.enforce_equal(&claimed_card_com_var)?;

// CHECK #2: Membership test.
// We prove membership of the commitment in the Merkle tree. Concretely, we use the leaf
// from above and path_var to recompute the Merkle root. We then assert that this root is
// equal to the publicly known root.
let leaf_var = claimed_card_com_var;
let computed_root_var =
    auth_path_var.calculate_root(&leaf_crh_params, &two_to_one_crh_params, &leaf_var)?;
computed_root_var.enforce_equal(&claimed_root_var)?;

// All done with the checks
Ok(())
```

Extending flexibility

Gadgets:

Expiry The credential hasn't expired

Linkable Show I'm the same person as before

Rate limiting I haven't used my credential too many times

Clone resistance If I reused my credential too many times, you can deanonymize me

We build all these with just a few lines of arkworks code.

Optimization
A new cryptographic technique that lets you reuse and link gadgets together

π_{bul} π_{gadg1}
π_{gadg2} π_{gadg3}

Extending flexibility

Gadgets:

Expiry: The credential hasn't expired

Linkable Show: I'm the same person as before

Rate limiting: I haven't used my credential too many times

Clone resistance: If I reused my credential too many times, you can deanonymize me

We build all these with just a few lines of arkworks code.

Extending flexibility

Expiry: The credential hasn't expired

Proof (gadget) π:

1. takes date as public input, *today*
2. opens credential's commitment to expiry date attribute, e
3. checks that $e > today$

Extending flexibility

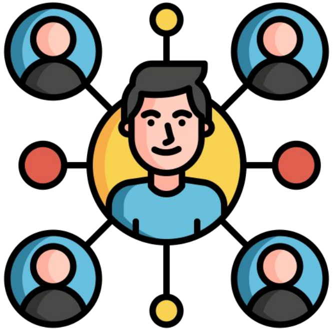
Linkable Show: I'm the same person as before

Proof (gadget) π:

1. takes context of persistent interaction as public input, *ctx*

2. opens credential's commitment to pseudonym key, *nk*

3. generates $PRF_{nk}(ctx)$ and checks against expected pseudonym



Extending flexibility

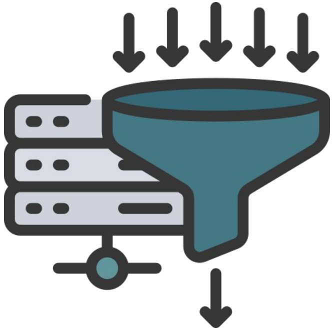
Rate limiting: I haven't used my credential too many times

Proof (gadget) π:

1. takes rate limit *(N, epoch)* and rate count *ctr* as public inputs

2. opens credential’s commitment to rate key, *rk*

3. generates token $PRF_{rk}(epoch \,||\, ctr)$ and checks that the token is unique wrt epoch and $ctr < N$



Extending flexibility

Clone resistance: If I reused my credential too many times, you can deanonymize me (Camenisch, Hohenberger et al., CCS 2006)

Proof (gadget) π is same as rate-limit gadget, with two differences:

1. verifier sends unique *nonce* with each credential show
2. generates *two* tokens:
   $tok_1 = PRF_{rk}(epoch \,||\, ctr)$
   $tok_2 = id + H(nonce) * PRF_{rk}(epoch \,||\, ctr)$

Reusing *ctr* with new *nonce* makes tok_1 repeat: solve for *id*

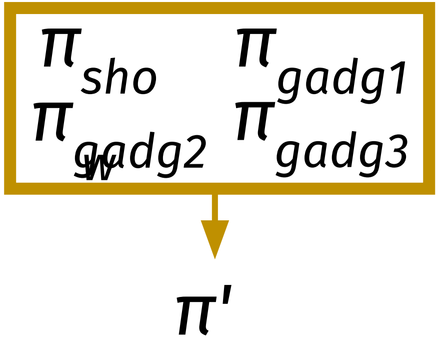
Extending flexibility

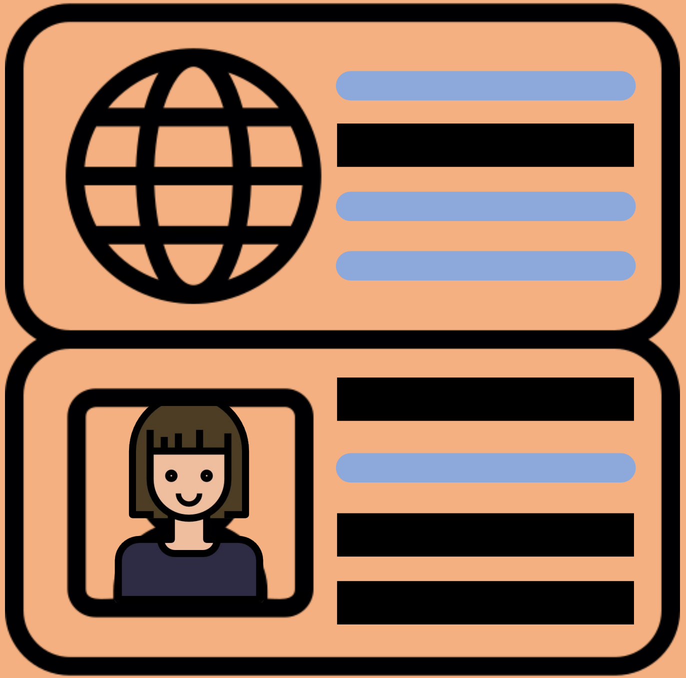
Proofs need to be re-computed when its *private* inputs change, but not *public* inputs.

Many useful credential proof gadgets only change public inputs across shows, even for a different show statement.

By binding shared public inputs across Groth16 proofs, we can **link** reused gadget proofs into a single LinkG16 show proof.

Optimization
A new cryptographic technique (**LinkG16**) lets you reuse and link gadgets together





zk-creds

A practical system must:

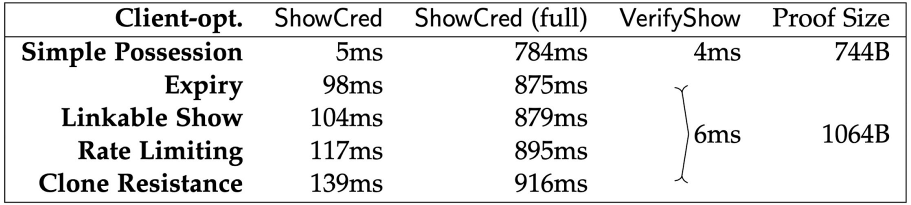
1. Support existing identity documents

2. Not require new trusted parties for issuance

3. Be easily programmable for new use cases

Experiments

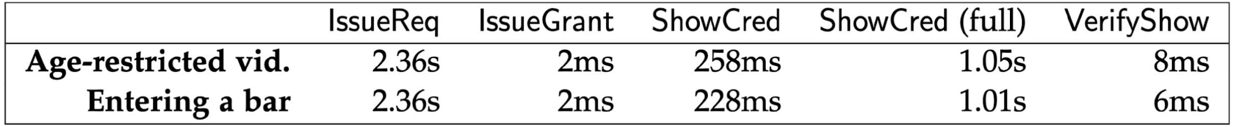
Microbenchmarks Benchmarked list membership + 1 gadget



Client-opt.	ShowCred	ShowCred (full)	VerifyShow	Proof Size
Simple Possession	5ms	784ms	4ms	744B
Expiry	98ms	875ms	6ms	1064B
Linkable Show	104ms	879ms		
Rate Limiting	117ms	895ms		
Clone Resistance	139ms	916ms		

Case study Wrote an **Android app** that dumps passports. Wrote a SNARK for US passport validity. Benchmarked proofs.

Show < 300ms
Proofs ~1KiB
Verify < 10ms



	IssueReq	IssueGrant	ShowCred	ShowCred (full)	VerifyShow
Age-restricted vid.	2.36s	2ms	258ms	1.05s	8ms
Entering a bar	2.36s	2ms	228ms	1.01s	6ms

Takeaway

Fast enough to run on your phone in the real world!

Extensions & future work

More identity sources. No limits on what we can use. DKIM to prove email ownership. DECO/TLSNotary to prove web account ownership.

More show statements. Prove you live in a specific voting district. Even if expensive, you only prove once.

Faster primitives. New ZKP-based crypto is coming out all the time!

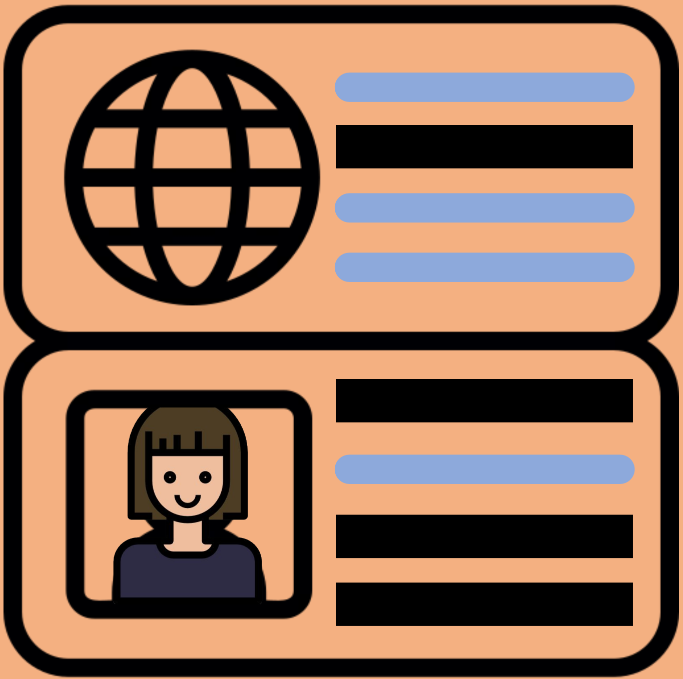
Conclusion

Show < 300ms
Proofs ~1KiB
Verify < 10ms

We built a **fast, flexible** anonymous credentials scheme.

Any part can be **swapped out** (hash, proof system, issuance list/signatures)

This is all possible due to general purpose **zkSNARKs**



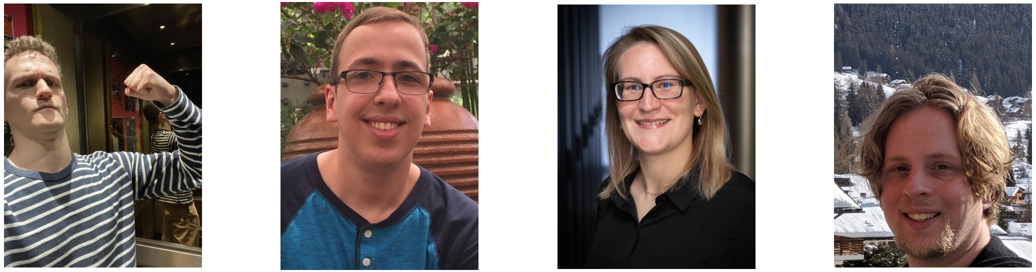
Images from flaticon.com

Q&A

zk-creds

Flexible Anonymous Credentials from zkSNARKs and Existing Identity Infrastructure

- Support for existing identity documents
- No new trusted parties issuing credentials
- Customizable w/o needing cryptographers



https://ia.cr/2022/878

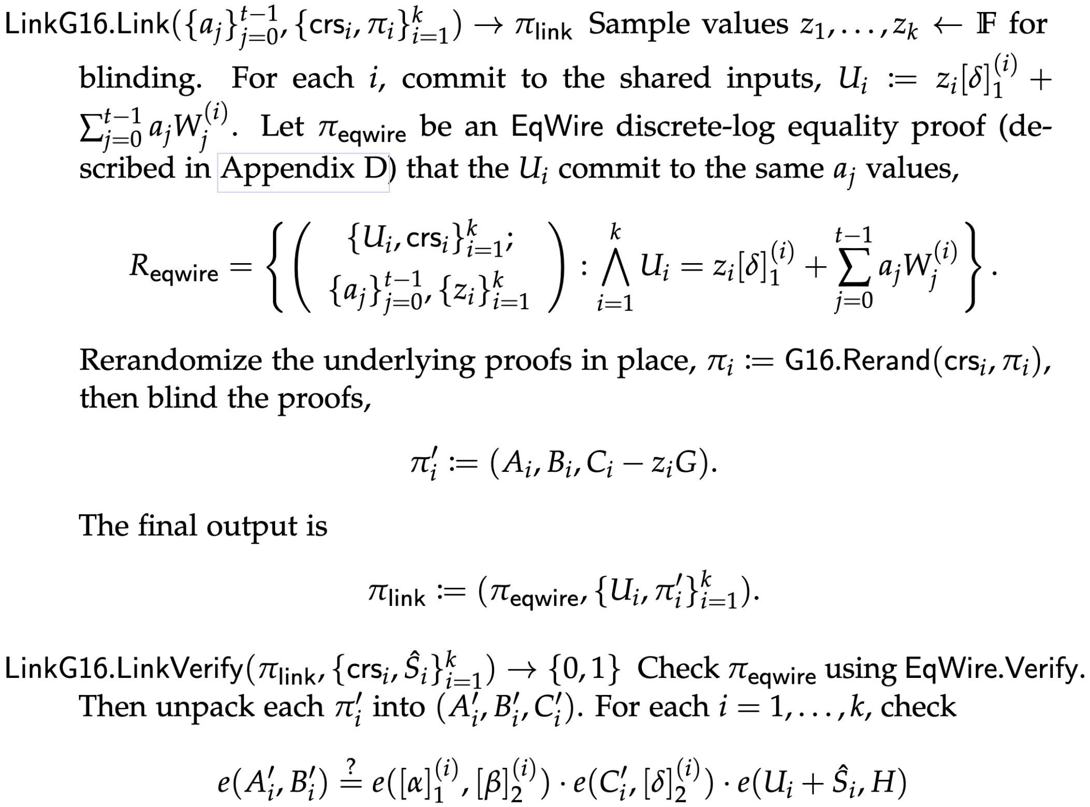
https://github.com/rozbb/zkcreds-rs

Jacob White
white570@purdue.edu

Backups

LinkG16

$$R_{\mathsf{linkg16}} = \left\{ \begin{pmatrix} \{\mathsf{crs}_i, \hat{S}_i\}_{i=1}^{k}; \\ \{a_j\}_{j=0}^{t-1}, \{\pi_i\}_{i=1}^{k} \end{pmatrix} : \bigwedge_{i=1}^{k} \mathsf{G16.Verify}(\mathsf{crs}_i, \pi_i, \hat{S}_i + \sum_{j=0}^{t-1} a_j W_j^{(i)}) \right\}$$



$\mathsf{LinkG16.Link}(\{a_j\}_{j=0}^{t-1}, \{\mathsf{crs}_i, \pi_i\}_{i=1}^{k}) \to \pi_{\mathsf{link}}$ Sample values $z_1, \ldots, z_k \leftarrow \mathbb{F}$ for blinding. For each i, commit to the shared inputs, $U_i := z_i[\delta]_1^{(i)} + \sum_{j=0}^{t-1} a_j W_j^{(i)}$. Let π_{eqwire} be an EqWire discrete-log equality proof (described in Appendix D) that the U_i commit to the same a_j values,

$$R_{\mathsf{eqwire}} = \left\{ \begin{pmatrix} \{U_i, \mathsf{crs}_i\}_{i=1}^{k}; \\ \{a_j\}_{j=0}^{t-1}, \{z_i\}_{i=1}^{k} \end{pmatrix} : \bigwedge_{i=1}^{k} U_i = z_i[\delta]_1^{(i)} + \sum_{j=0}^{t-1} a_j W_j^{(i)} \right\}.$$

Rerandomize the underlying proofs in place, $\pi_i := \mathsf{G16.Rerand}(\mathsf{crs}_i, \pi_i)$, then blind the proofs,

$$\pi_i' := (A_i, B_i, C_i - z_i G).$$

The final output is

$$\pi_{\mathsf{link}} := (\pi_{\mathsf{eqwire}}, \{U_i, \pi_i'\}_{i=1}^{k}).$$

$\mathsf{LinkG16.LinkVerify}(\pi_{\mathsf{link}}, \{\mathsf{crs}_i, \hat{S}_i\}_{i=1}^{k}) \to \{0,1\}$ Check π_{eqwire} using $\mathsf{EqWire.Verify}$. Then unpack each π_i' into (A_i', B_i', C_i'). For each $i = 1, \ldots, k$, check

$$e(A_i', B_i') \stackrel{?}{=} e([\alpha]_1^{(i)}, [\beta]_2^{(i)}) \cdot e(C_i', [\delta]_2^{(i)}) \cdot e(U_i + \hat{S}_i, H)$$

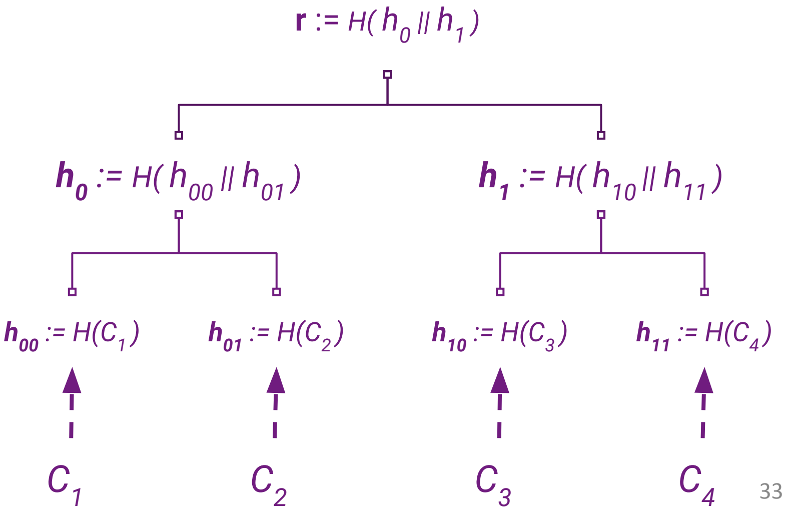
Background: Merkle Trees

An **accumulator** data structure that recursively applies a cryptographic hash function, *H*, to a list of values. The tree's **root** summarizes the state of the list. Membership proofs in a Merkle tree are the nodes to re-compute the root:

Example. Merkle path proving membership of C_2 in MT:

x := $(h_{01} = H(C_2), h_{00}, h_1)$, $aux := r$

Verify: $r = H(H(h_{00}||h_{01})||h_1)$



$\mathbf{r} := H(h_0 || h_1)$
$\mathbf{h_0} := H(h_{00} || h_{01})$
$\mathbf{h_1} := H(h_{10} || h_{11})$
$\mathbf{h_{00}} := H(C_1)$
$\mathbf{h_{01}} := H(C_2)$
$\mathbf{h_{10}} := H(C_3)$
$\mathbf{h_{11}} := H(C_4)$
C_1
C_2
C_3
C_4

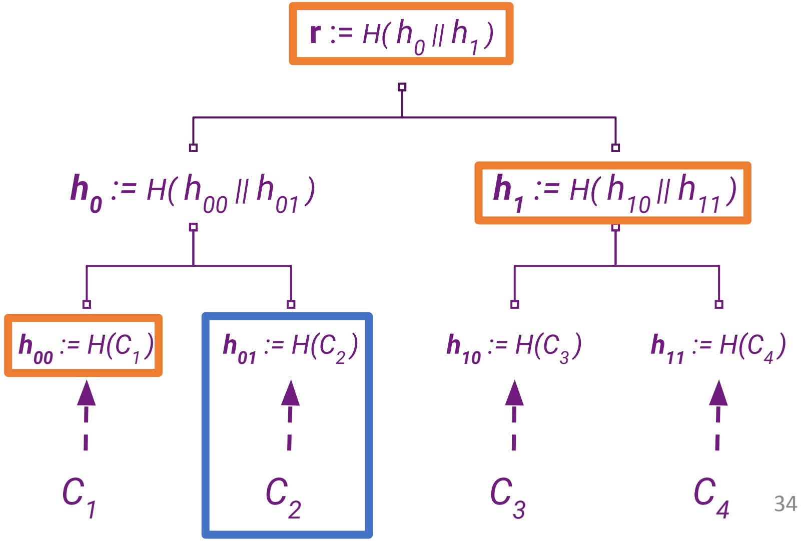
Background: Merkle Trees

Zero-Knowledge proofs of membership in the list simply hide the leaf being verified wrt the current state of the list (root):

Example. Merkle path proving membership of C_2 in MT:

$x := (h_{01} = H(C_2), h_{00}, h_1), aux := r$

Verify: $r = H(H(h_{00}||h_{01})||h_1)$



$\mathbf{r} := H(h_0 \,||\, h_1)$
$\mathbf{h_0} := H(h_{00} \,||\, h_{01})$
$\mathbf{h_1} := H(h_{10} \,||\, h_{11})$
$\mathbf{h_{00}} := H(C_1)$
$\mathbf{h_{01}} := H(C_2)$
$\mathbf{h_{10}} := H(C_3)$
$\mathbf{h_{11}} := H(C_4)$
C_1
C_2
C_3
C_4

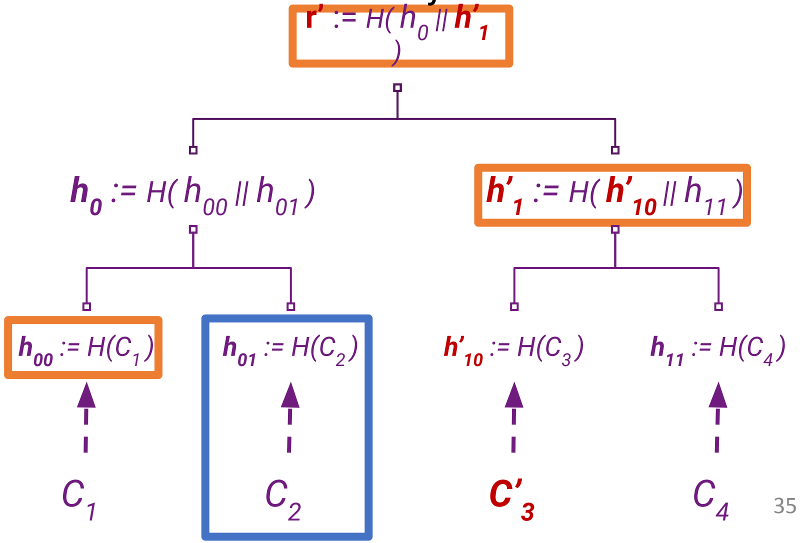
Background: Merkle Trees

Problem. Any credential in the Merkle tree, say C_3 , that changes (e.g. new cred or in-place revocation) will change the straight-line path from C_3 to *r*, → Merkle path up to r + the corresponding ZK proof for *all* other credentials $C_j \neq C_3$:

Example. Merkle path proving membership of C_2 in MT is **now**:

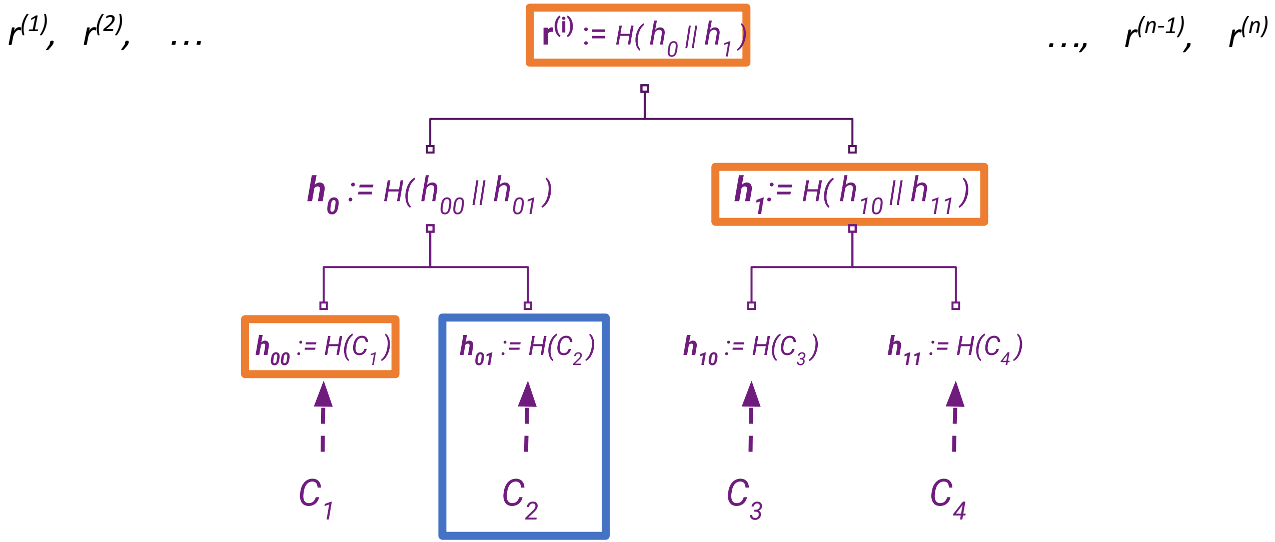
$x := (h_{01} = H(C_2), h_{00}, \mathbf{h'_1}), aux := \mathbf{r'}$

Verify: $\mathbf{r'} = H(H(h_{00}||h_{01})||\mathbf{h'_1})$



Contribution: Merkle Forests

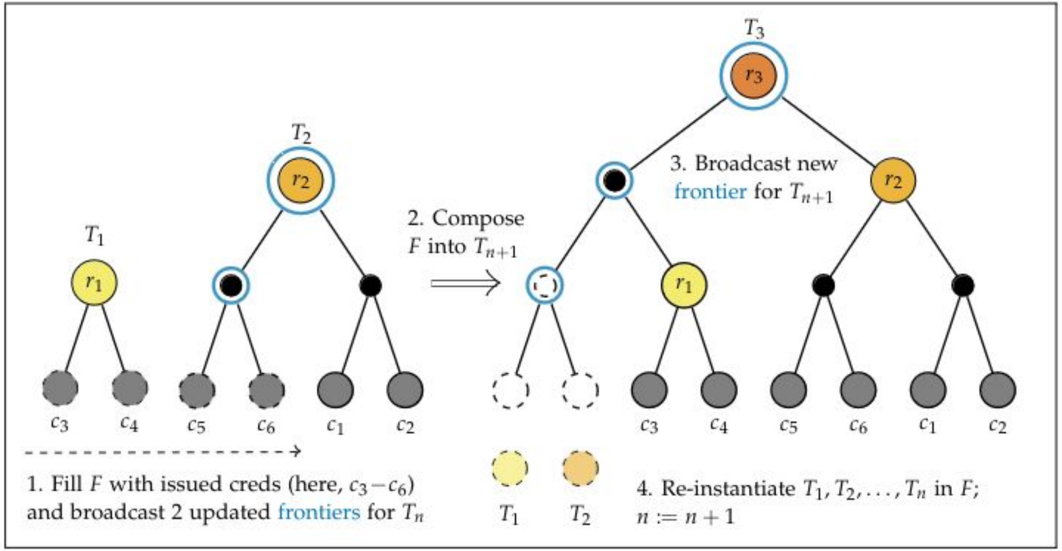
Solution. Have each zk-creds "issuer" manage their own individual Merkle trees to reduce the rate of Merkle path changes:



$r^{(1)}$, $r^{(2)}$, …
$\mathbf{r^{(i)}} := H(h_0 \| h_1)$
…, $r^{(n-1)}$, $r^{(n)}$
$\boldsymbol{h_0} := H(h_{00} \| h_{01})$
$\boldsymbol{h_1} := H(h_{10} \| h_{11})$
$\boldsymbol{h_{00}} := H(C_1)$
$\boldsymbol{h_{01}} := H(C_2)$
$\boldsymbol{h_{10}} := H(C_3)$
$\boldsymbol{h_{11}} := H(C_4)$
C_1
C_2
C_3
C_4

Contribution: Merkle Forests

Solution. Novel optimizations to:
1) **reduce rate of change to Merkle path** / membership proof;
2) make it easier for users & ver. to use **frontiers to sync MT**;
3) **eliminate leaks** about the credential updating its proof.
Assuming credentials added left-to-right and unchanging...



T_1
r_1
T_2
r_2
T_3
r_3
r_1
r_2
c_3
c_4
c_5
c_6
c_1
c_2
c_3
c_4
c_5
c_6
c_1
c_2
1. Fill F with issued creds (here, $c_3 - c_6$) and broadcast 2 updated frontiers for T_n
2. Compose F into T_{n+1}
3. Broadcast new frontier for T_{n+1}
T_1
T_2
4. Re-instantiate $T_1, T_2, \ldots, T_n$ in F; $n := n + 1$

Terminology

Credential

User

Service Provider

Issuer