Google

ZK Credentials from ECDSA

Matteo Frigo
abhi shelat
Google

March 2025

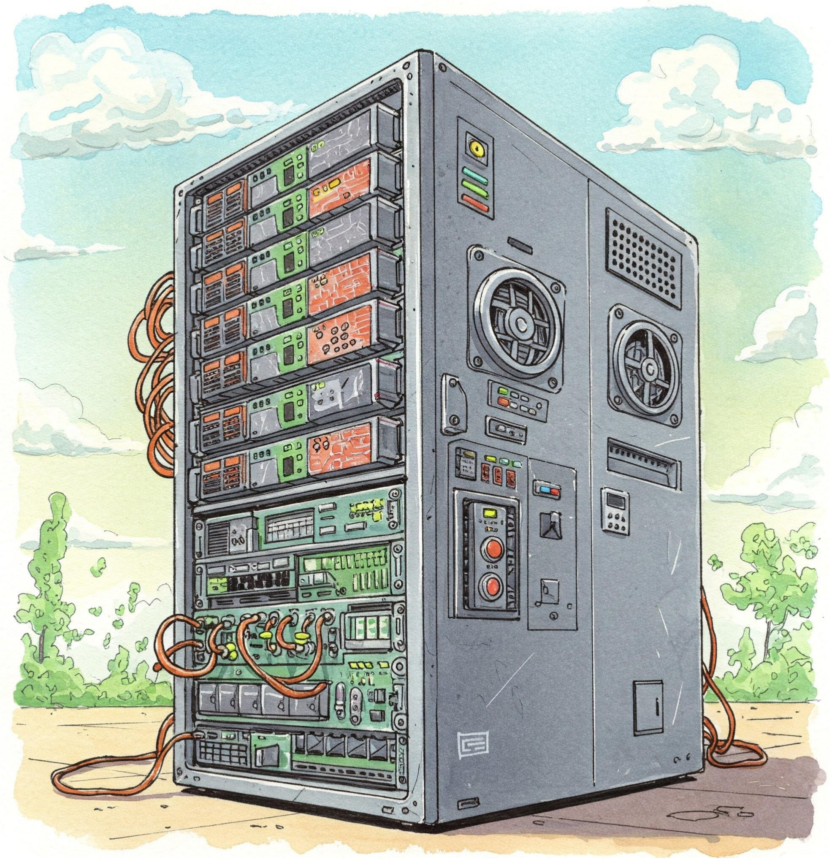
ZK-identity, different problem constraints

Minimize prover resource (cf of verifier time).

- **<u>NARK</u>** instead of ~~S~~NARK
- 1mb code + data, <0.5gb of ram.

Small prover

Eliminating the requirement for "succinctness" is a boon.



Very large Verifier (website)

ZK-identity, different problem constraints

Minimize prover resource (cf of verifier time).

People/standards coordination is expensive.

- Trusted parameters will be difficult/impossible to setup. (No CRS).
- Can't ignore all the completed work on standardizing "data formats", e.g., ISO Mobile DOC, or JSON WEB TOKEN (JWT)

ZK-identity, different problem constraints

Minimize prover resource (cf of verifier time).

People/standards coordination is expensive. (No CRS, Use legacy standards)

Issuer's are limited.

- Hard to deploy new crypto/infrastructure.
- Not highly available, cannot scale to O(internet activity). Should not be involved in online flow, or do work proportional to # of logins.

The right ZK system can solve all of these problems.

2nd oldest recipe for ZK [BGGHKMR88]

Run an IP → Transcript.

Commit to Transcript → Com.

ZK for "Com contains T and IP-verifier(T)=1"

It just remains to pick, IP, Commit, ZK.

Our Choices

IP: Sumcheck (Prover can run in O(C) time!)

Commit/ZK: Ligero

Ligero(Layered-sumcheck) > Ligero

Faster to use ligero to verify the transcript of any IP (layered-sumcheck).

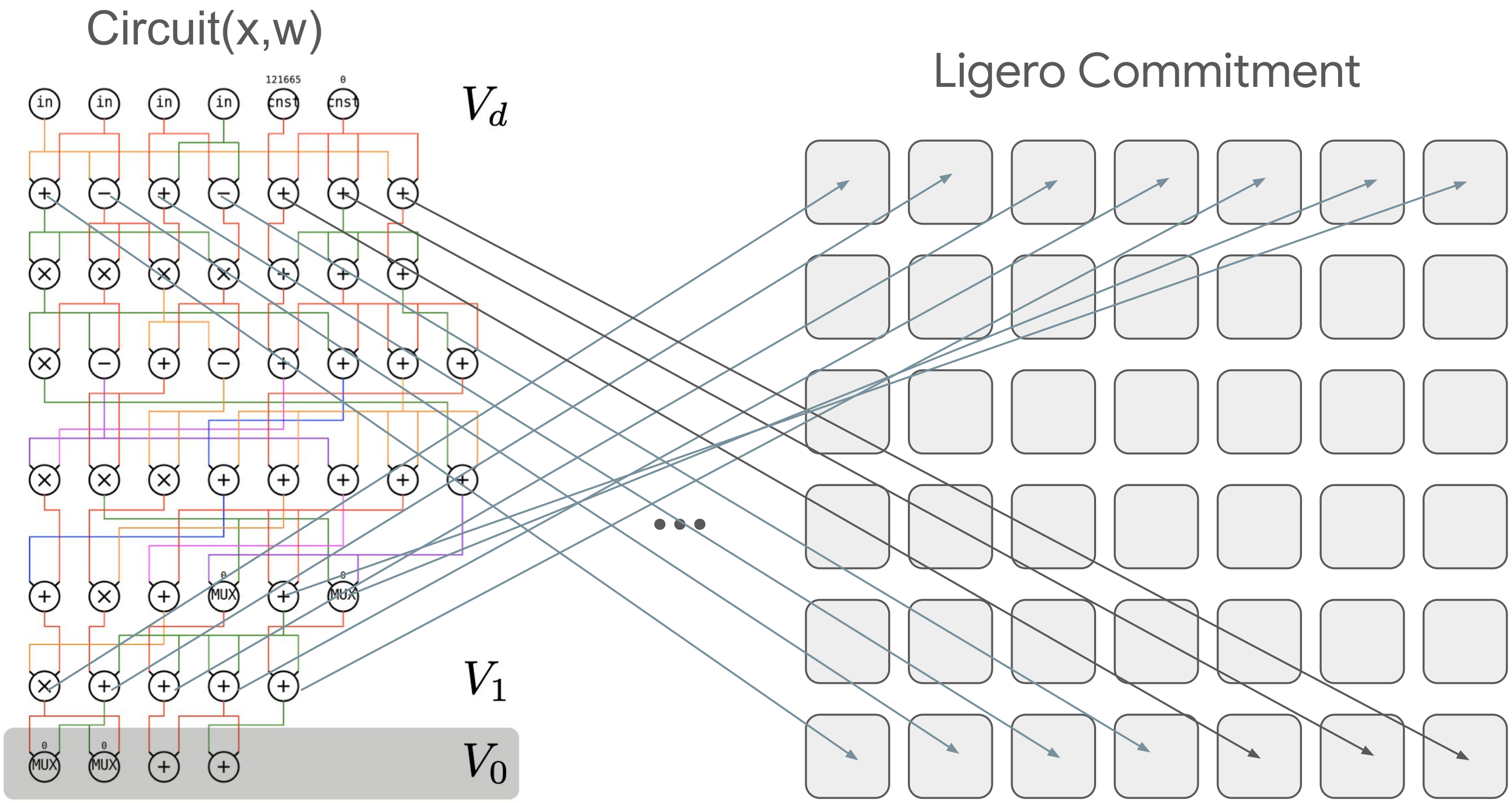
This follows the “Hyrax approach”, but w/ different Com + ZK.

SpartanZK (next 2 talks) also uses “Hyrax-variant over R1CS”, EC commitments.

This work only uses SHA-256

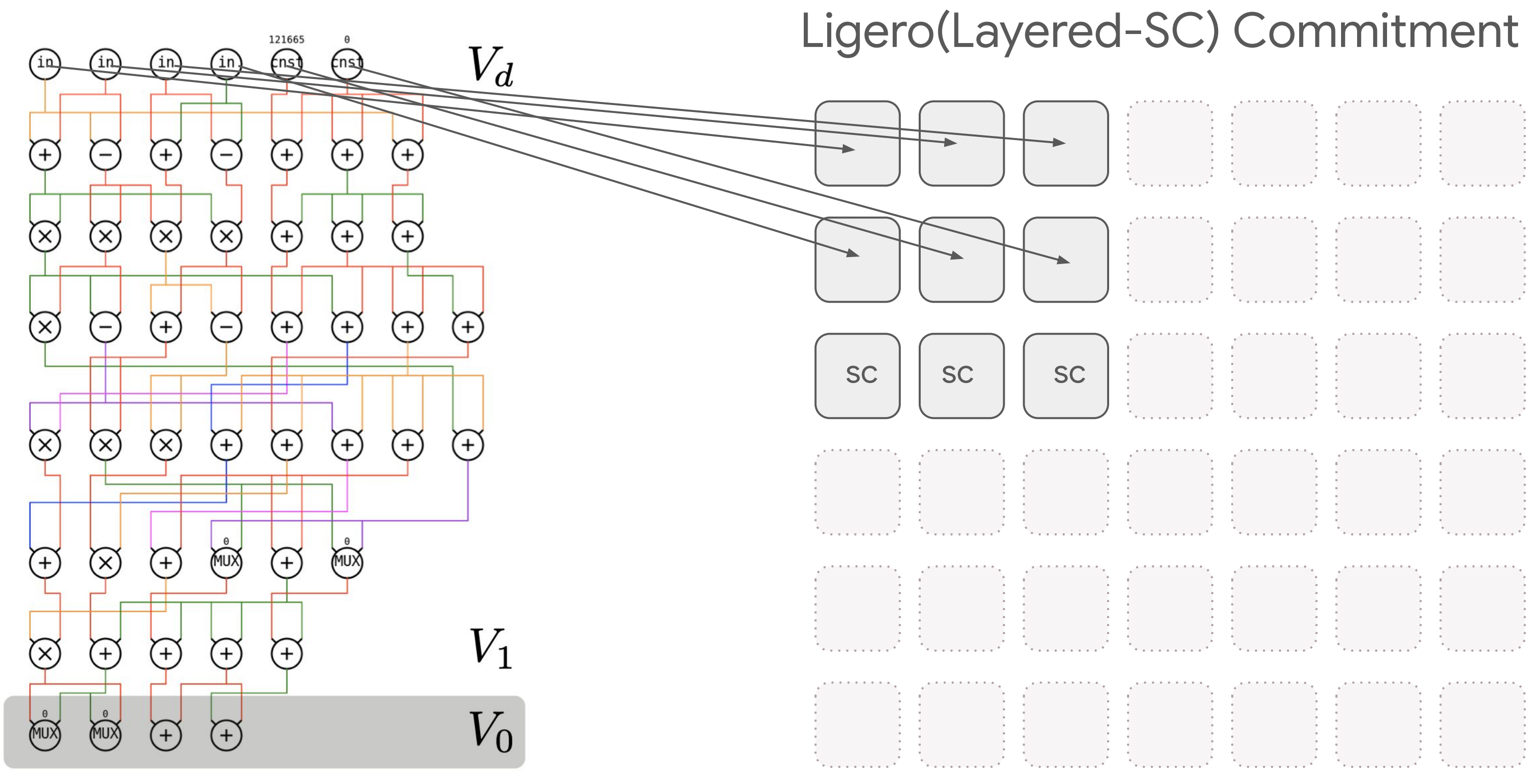
(purportedly pq-safe)

Ligero must commit to every intermediate wire



Ligero(Layered-sumcheck) commits to input+sc proof

and only verifies the sumcheck proof



Ligero(Layered-sumcheck) > Ligero

Efficient for Ligero to verify the transcript of any IP (layered-sumcheck).

Commitment time can be a bottleneck (NTT); our approach reduces the commitment size by 25-100x.

Verifying an ECDSA signature

Algorithm 2 Verification Circuit $V(Q, H(m), r, s, r_y)$

1: Derive integer e from $H(m)$.
2: Verify that $r, s \in [1, q-1]$
3: Verify that $Q, R = (r, r_y) \in E$ and $R \neq id$.
4: Verify $id = G \cdot e + Q \cdot r - R \cdot s$

Prior work (eg circom-ECDSA, Stark,Plonk,...) needs to emulate the field math of Fp, Fq.

Performing NTT on P256

The P256 finite field doesn't have enough roots of unity for an NTT.

P256 - 1 = 2 * 3 * 5^2 * 17 * 257 * 641 * 1531 * 65537 * 490463 * 6700417 * 835945042244614951780389953367877943453916927241

Its quadratic extension does. So we can lift the NTT to F_P256^2.

$P256^2 - 1$ = 2^97 * 3 * 7 * 5^2 * 17 * 257 * 641 * 1531 * 65537 * 274177 * 490463 * 6700417 * 67280421310721 * 11318308927973941931404914103 * 835945042244614951780389953367877943453916927241

ECDSA verification is a small circuit in F_P256

Table 1: Circuit size and depth for ECDSA verification.

	DEPTH	QUADS	TERMS	INPUTS
Multi-exponentiation	7	19,534	37,550	1,038
Range check + rest	12	5,475	10,569	1,038
Total	12	23,453	47,598	1,038

~50x fewer wires
to commit

Finite field used by our circuit is over the base field of the P256 curve. Prior work was expensive b/c it simulated math in F_p in an NTT-friendly F_q.

Possible because we can perform efficient NTTs for F_p256.

zk-ECDSA

PoK(r,s) for (e,pk)

		Time (ms)		
		$n = 1$	2	3
x64 Single thread	Ligero com	38.7	51.0	60.8
	Verify transcript	13.5	26.5	38.6
	Total ZK	$\boxed{58.8}$	87.0	110
	Verifier	$\boxed{6.09}$	11.0	14.5
Pixel Single thread	Ligero com	51.0	67.2	80.0
	Verify transcript	20.3	40.1	58.4
	Total ZK	80.5	120.0	152.0
	Verifier	8.50	16.2	21.2

circom-ECDSA:
<u>140000ms</u> (single core)
973MB trusted parameter

Woo et al (IEEE S&P'25):
<u>900ms</u> (uses Spartan zk system + sidecar sigma prot) [Last talk this session]

Update: 300ms

ZK for SHA-256 pre-image (n block message)

		OurZK Protocol, Time (ms)					
		$n = 1$	2	4	8	16	32
x64 Single thread	Verify	6.07	12.1	24.3	49.8	106	228
	Total ZK	10.7	19.8	35.2	67.9	140	286
Pixel Single thread	Verify	11.3	23.5	48.9	100.1	206	436
	Total ZK	19.2	37.0	68.9	132	257	517

Proof size ~200kb.

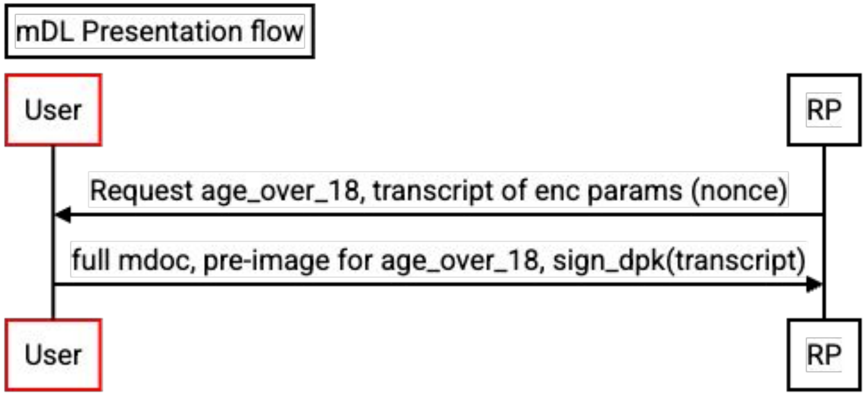
Using field F_2^{128}

Ligero ZK SHA-256 preimage systems

		Time (ms)		
		$n = 1$	2	4
x64	Commit	222	384	736
	Total ZK Prover	273	493	939
	Overhead wrt this work	26x	25x	27x
	[WHV24] Smaller field, multi-threaded	250	450	750
Pixel	Commit	294	536	1022
	Total ZK Prover	380	717	1370
	Overhead wrt this work	20x	19x	20x

Legacy ISO MDOC identity protocol

“The state of Massachusetts has produced a signature on an mDL document stored on my cell phone that includes the attribute ‘age_over_18 = true’.”



1. Verify Signature of mdoc by Massachusetts.
2. Parse mdoc to find DPK.
3. Verify Signature of transcript under DPK.
4. Verify pre-image of the “age_over_18” attribute, verify it is set to True.
5. Verify credential expiry condition.

ZK-MDOC statement to prove

Given the public values (PK_II, “age_over_18”, transcript, time_now),
there exists
a 2231 byte string MDL, a hash e_1, a hash h_2, an index X,
a signature sig_1, …, a 32-b nonce, a pk DPK,
a pair of strings time_start, time_end, and a signature sig_2
such that:

Costly part

```
e_1 = SHA-256(MDL)                                                 and
p256.verify( sig_1, e_1, PK_II ) = true                            and

h_2 = MDL[valueDigests][org.iso.18013.5.1][‘age_over_18’]]         and
h_2 = SHA-256(nonce, ‘age_over_18’, ‘True’)                        and

DPK = MDL[deviceKeyInfo][deviceKey][-2, -3]                        and
p256.verify( sig_2, transcript, DPK )    = true                    and

time_start = MDL[validityInfo][validFrom]                          and
time_end = MDL[validityInfo][validUntil]                           and
time_start < time_now < time_end                                   and
```

Device binding

Prover_MDOC_2.4kb:
1.2s

Verifier: **0.6s**

Credential revocation

- Easy: short validity period, issuer re-issues every 30 days
- Not enough.

Pseudonyms Supported

“The state of Massachusetts has produced a signature on an mDL document stored on my cell phone that includes the attribute ‘age_over_18 = true’.

AND

the attribute ‘pseudo-nym seed’ = H(secret)

AND

PRF(secret || context) = nym.”

JWT

eyJhbGciOiJFUzI1NiIsInR5cC
I6IkpXVCJ9.eyJzdWIiOiIxMjM
0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiYWdlX292ZXJfMTg
iOnRydWUsImlhdCI6MTUxNjIzO
TAyMn0.a8x_0hdedg20oaz0ArQ
H7n59Vl0jNgT23ymUL7ro-UieP
gy7apdJDFwz1MkqeSAIlAXKBW1
wZgGov7CIXOVVhw

{ "alg": "ES256",
"typ": "JWT" }

{ "sub": "1234567890",
"name": "John Doe",
"age_over_18": true,
"iat": 1516239022 }

`(r, s) ECDSASHA256 signature`

ZK-JWT

“PoK(hdr,pay,r,s) s.t (r,s) is a sig on base64(hdr).base64(payload) and payload contains age_over_18=true.

Fiat-Shamir attacks don't apply

Nobody uses sumcheck to "compute F." We use it to *verify* f(x) in very low depth, which renders the recent attacks mute.

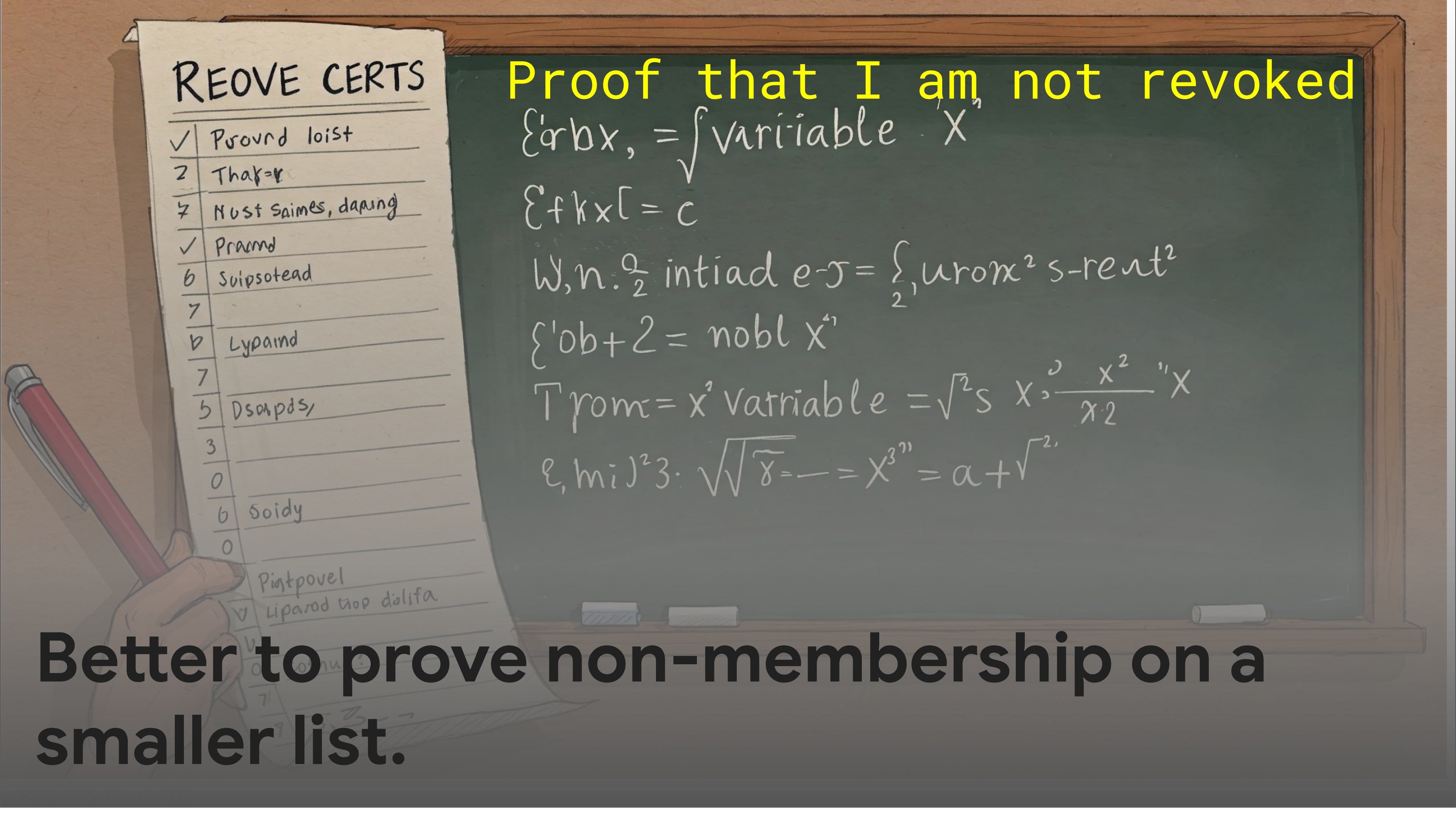
Additionally, we set the RO function to be > complex than f, which also eliminates all attacks in which the protocol computes the RO.

Thank you

Code will be open source soon.

Backup slides

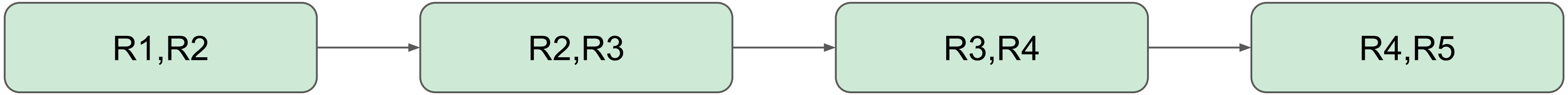
Proving a cert is valid is more total work.



Proof that I am not revoked

Better to prove non-membership on a smaller list.

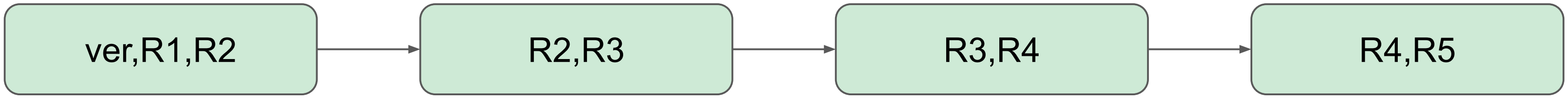
Credential revocation



Issuer signs each node of a sorted linked list, a node is a pair of adjacent, revoked cred identifiers.

“100x better if revocation list is < 1% of users.”

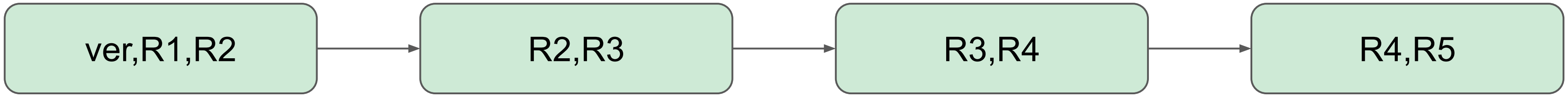
Credential revocation



Issuer signs each node of a sorted linked list, a node is a pair of adjacent, revoked cred identifiers.

Valid cred identifiers exist between R_i, and R_{i+1}.

Credential revocation implemented



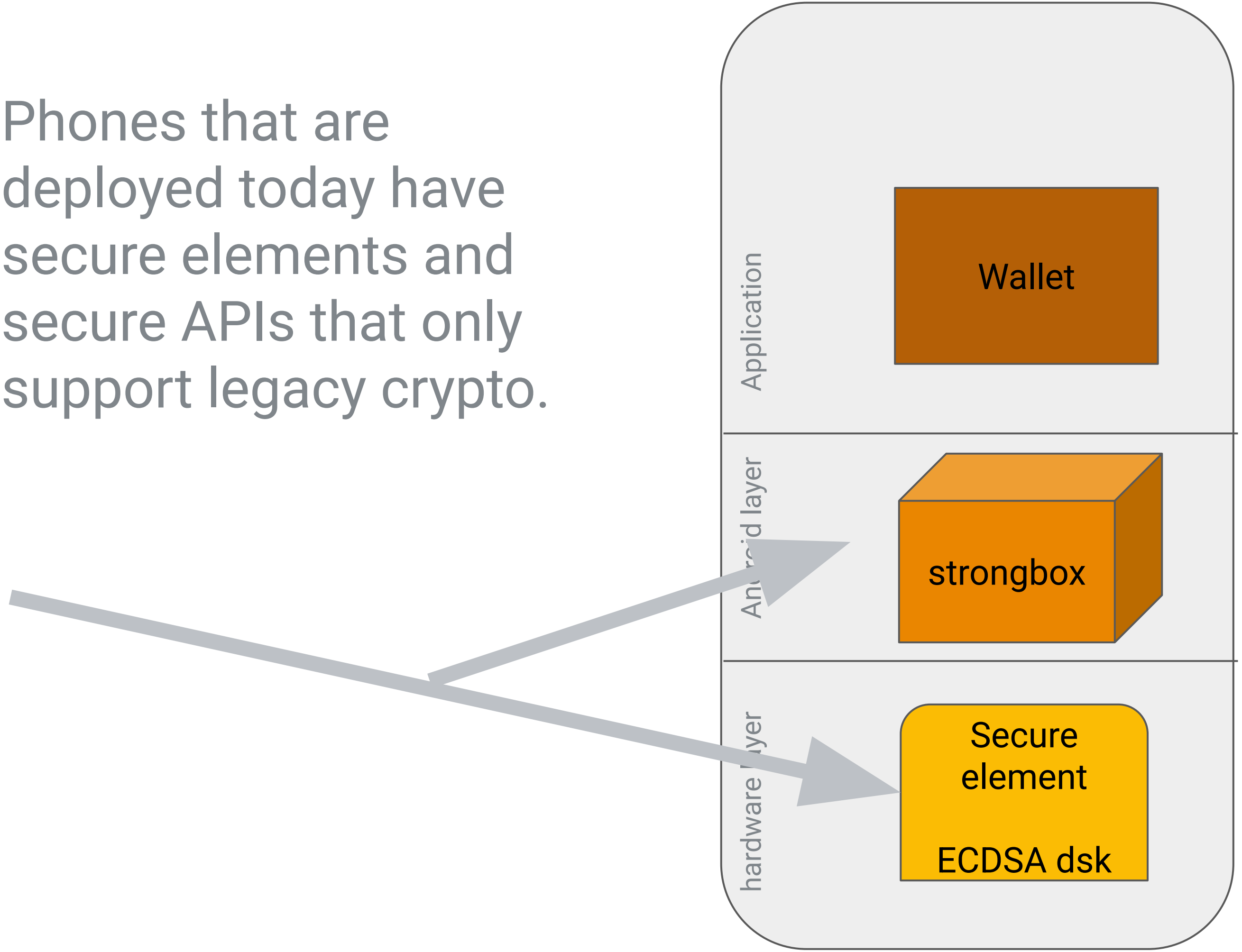
Issuer signs each node of a sorted linked list, a node is a pair of adjacent, revoked cred identifiers.

Valid cred identifiers exist between R_i, and R_{i+1}.

ZK-Prove you are valid by proving knowledge on a signature of (left, right) such that (left < your-id < right).

Phones that are deployed today have secure elements and secure APIs that only support legacy crypto.

5 years.



P convinces V that “there exists w such that $f(\vec{x}, \vec{w}) = 0$ ”

Prover $(f, \vec{x}, \vec{w})$

Verifier $(f, \vec{x})$

Pick a random pad.
Ligero-Commit(w, pad)

com

Run Sumcheck-P(C,x,w)

Run Sumcheck-V(C,x)

When Sumcheck sends message
m_i, send m_i + pad_i.
Record transcript T.

…

Send random challenges.
Record transcript T

Run Ligero-Prover(com, Enc-Sumcheck-Verify(T,x,pad))

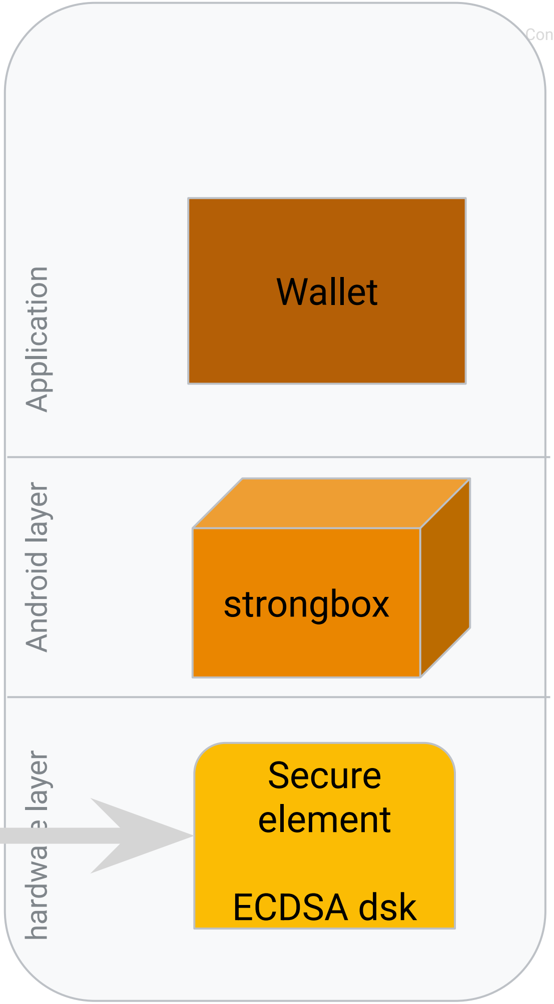
Ligero-Verify(com,
Enc-Sumcheck-Verify,T,x,proof)

Copying credentials

Here, you can install my credential on your phone to get free access to all the services available to students with the U.of X credential.

If it is easy to “root” a device, copy the storage, and extract a credential, then users will trade them, and the ultimate soundness of the system will degrade.

To prevent users from sharing credentials, issuers only create credentials that include a **device bound** public key that is stored here



This device-bound key is required in the presentation protocol (sign a transcript with DPK)

SHA-256 layered circuit arithmetization

Table 2: Circuit size and depth for 1 SHA-256 block over 2 different fields.

	PACKING	DEPTH	QUADS	TERMS	INPUTS
$\mathbb{F}_{P256}$	-	7	37,974	167,348	6,657
	2	9	65,690	215,504	3,585
	3	10	76,287	239,919	2,625
$\mathbb{F}_{2^{128}}$	-	13	53,435	87,642	6,657
	2	14	65,727	150,991	3,585
	3	15	73,818	166,494	2,625

STARK zk SHA-256 preimage

We tested the stone prover.

1 SHA-block needed degree bound 4.1m. **106s (single core)**

2 SHA-blocks fit into the same degree bound, same time.

4-block didn't fit into memory.