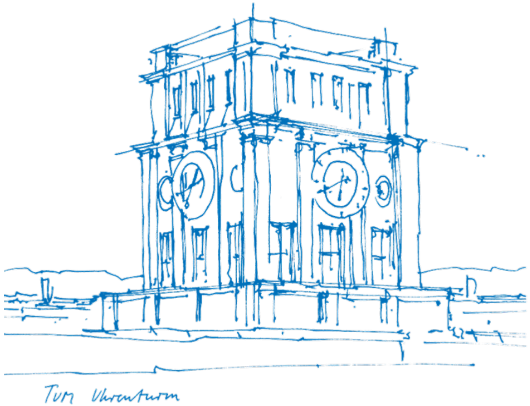
IEEE ICBC
IEEE International Conference on Blockchain and Cryptocurrency
3-6 May 2021 // Virtual Conference

A-PoA: Anonymous Proof of Authorization for Decentralized Identity Management

Jan Lauinger, Jens Ernstberger, Emanuel Regnath,
Mohammad Hamad, Sebastian Steinhorst

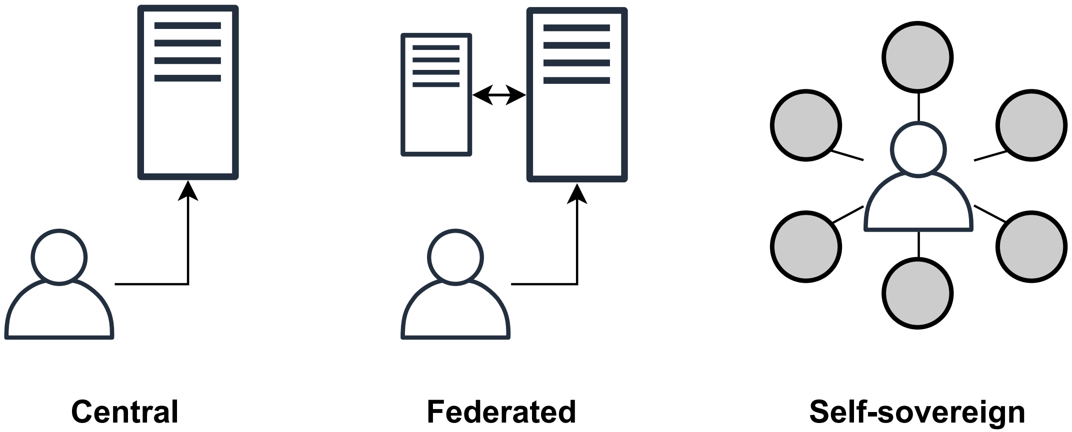
Technical University of Munich
TUM Department of Electrical and Computer Engineering
Associate Professorship of Embedded Systems and Internet of Things
October 13, 2024



nIoVe

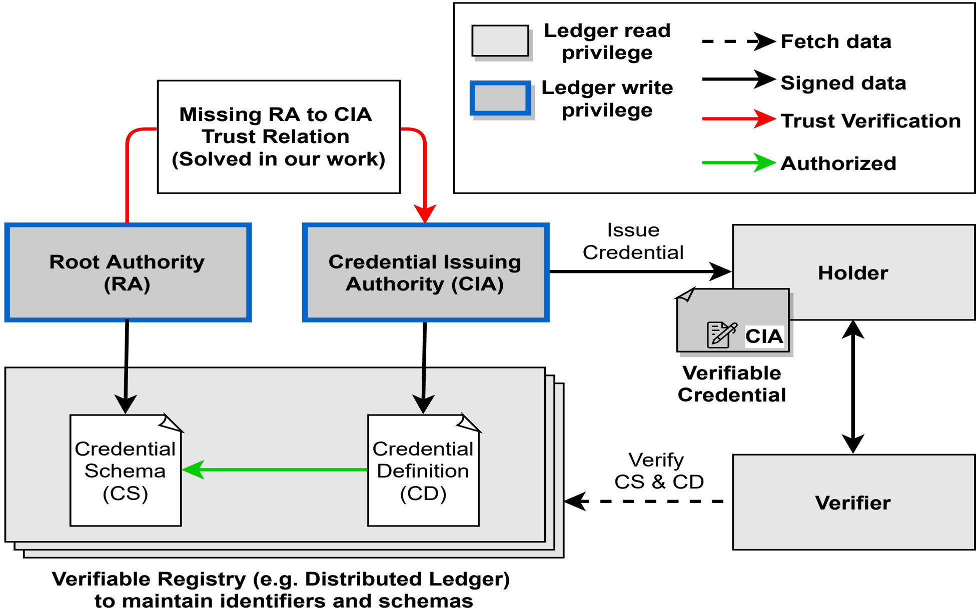
Motivation

Context: Self-sovereign Identity Management (SSIM)



Problem: Missing Authorization

Distributed Ledger-based Identity Management (IdM) based on Verifiable Credentials



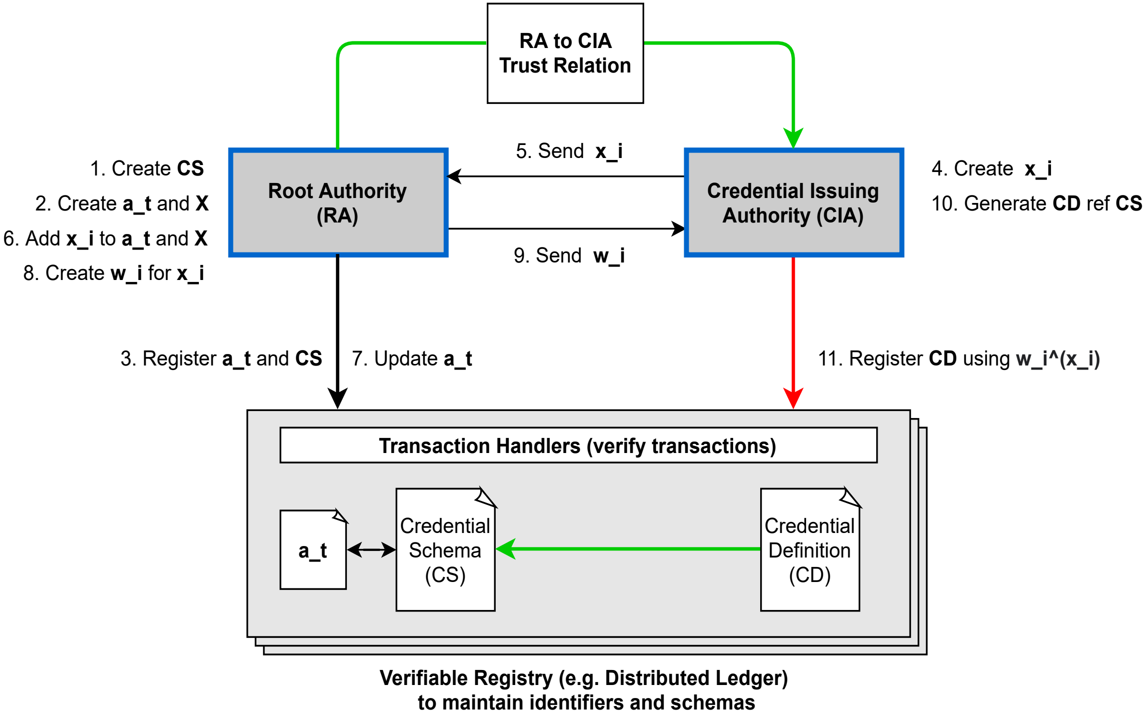
Dynamic Cryptographic Accumulator

RSA-Accumulator (Camenisch and Lysyanskaya [1])

- Initialize accumulator: $a_t = (g_{\mathrm{acc}})^2 \mod n$
- Add element to the accumulator: $a_{t+1} = a_t^{x_i} \mod n$
- Calculate element witness pair (x_i, w_i): $w_i = a_t^{X_t \setminus \{x_i\}} \mod n$
- Verify accumulator membership of x_i: $w_t^{x_i} \mod n == a_{t+1}$
- Revoke accumulator element x_i: $a_{t+1} = a_t^{x_i^{-1} \mod \phi(n)} \mod n$, with $\phi(n) = (p-1) \cdot (q-1)$
- Requirement to update all witness values after addition or revocation.

Cryptographic Accumulator in the Context of Hyperledger Indy

Privacy Issue

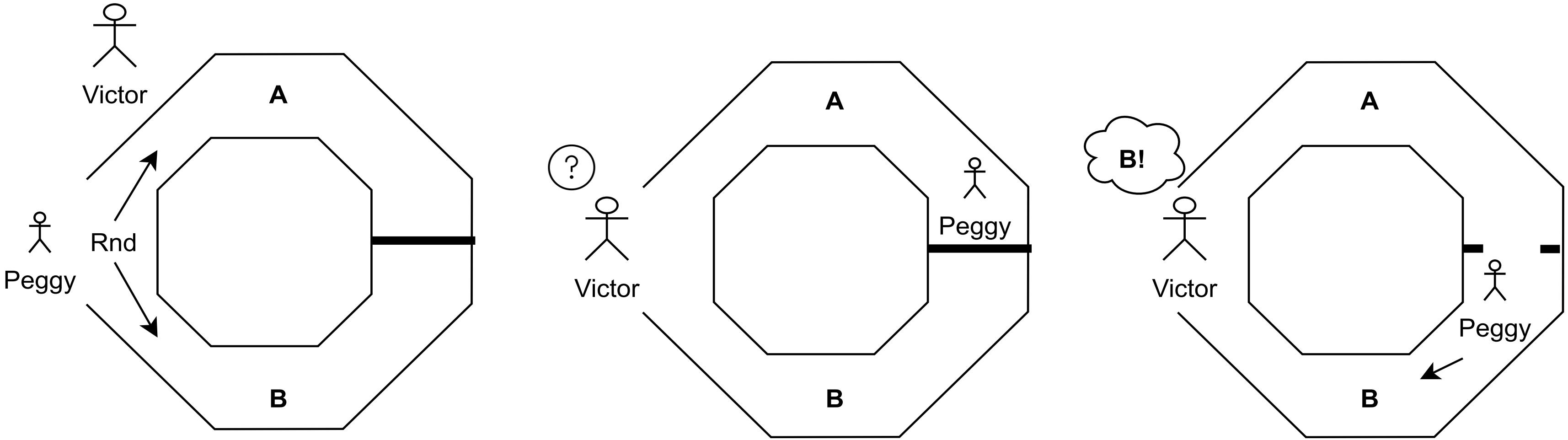


Zero Knowledge Proof

- **Completeness**
  - Prover *P*, **knowing** a solution to a problem, can successfully convince a verifier *V*.
- **Soundness**
  - Prover *P*, **not** knowing a solution to a problem, will fail to convince a verifier *V*.
- **Zero Knowledge**
  - A **zero knowledge** Proof of Knowledge (PoK) scheme requires the verifier *V* to learn nothing but the validity of a convincing assertion from prover *P*.

Intuitive Zero Knowledge Proof: Alibaba's Cave Analogy

- Peggy P the prover challenges Victor V the verifier.
- After several rounds, V is convinced that P knows a secret.



Schnorr Proof of Knowledge of Discrete Log

Prover		Verifier
$\alpha \in \mathbb{Z}_q$		$(U, A = U^\alpha) \in \mathbb{G}$
$r \leftarrow \mathbb{Z}_q$	$\xrightarrow{R = g^r \in \mathbb{G}}$	
	$\xleftarrow{c \in C}$	$c \leftarrow C = \{1, \dots, 2^\lambda\} \subseteq \mathbb{Z}_q$
$z = c \cdot \alpha + r \in \mathbb{Z}$	$\xrightarrow{z}$	accept iff $U^z = A^c \cdot R$

Figure: The Schnorr Proof of Knowledge protocol [2], as shown in [3].

1. Completeness Commitment scheme (opening, challenge, response)

$$U^z = g^z = g^{c \cdot \alpha + r} = g^r \cdot (g^\alpha)^c = R \cdot (U^\alpha)^c = R \cdot A^c$$

2. Soundness *Extractor* concept with the ability to extract secret knowledge from P convinces V of the existence of a satisfying solution.

2. Soundness Assumption: V is able to receive two accepting conversations (R, c, z) and (R, c', z').

$$\text{With } \; U^z = A^c \cdot R \quad \text{and} \quad U^{z'} = A^{c'} \cdot R$$
$$\implies \qquad U^{z - z'} = A^{c - c'}$$
$$\implies \qquad \alpha = \frac{z - z'}{c - c'} = \log_g(U)$$

3. HVZK *Simulator* concept with simulated transcript T_{sim} and real transcript T_{real} of interactive protocol [4].

3. Honest Verifier Zero Knowledge

Select $z, c \leftarrow \mathbb{Z}_q$

Calculate $\alpha = \frac{g^z}{U^c}$

Output $T_{sim} = (\alpha, c, z)$

4. Zero Knowledge (Fiat-Shamir Heuristic)

$$c = H(R) \implies c = H(T)$$

Towards Non-interactive Zero Knowledge Proof of Exponent

Problem with Schnorr: $\mathbb{G}$ is a finite cyclic group of prime order q. $\Rightarrow$ We need a ZK proof of discrete log in a group of unknown order for RSA accumulator.

$\Rightarrow$ Boneh, Bünz, and Fisch [3] construct the NI-ZKPoKE protocol, which is sound and secure under the adaptive root problem.

Extraction based on Chinese Remainder Theorem [5] with recovery of $(\alpha \mod l)$ for many l and simulation for HVZK leverages the Pedersen commitment [6].

Boneh et al. NI-ZKPoKE [3]

GenProof $(\, w_x, x, a_t \,)$:

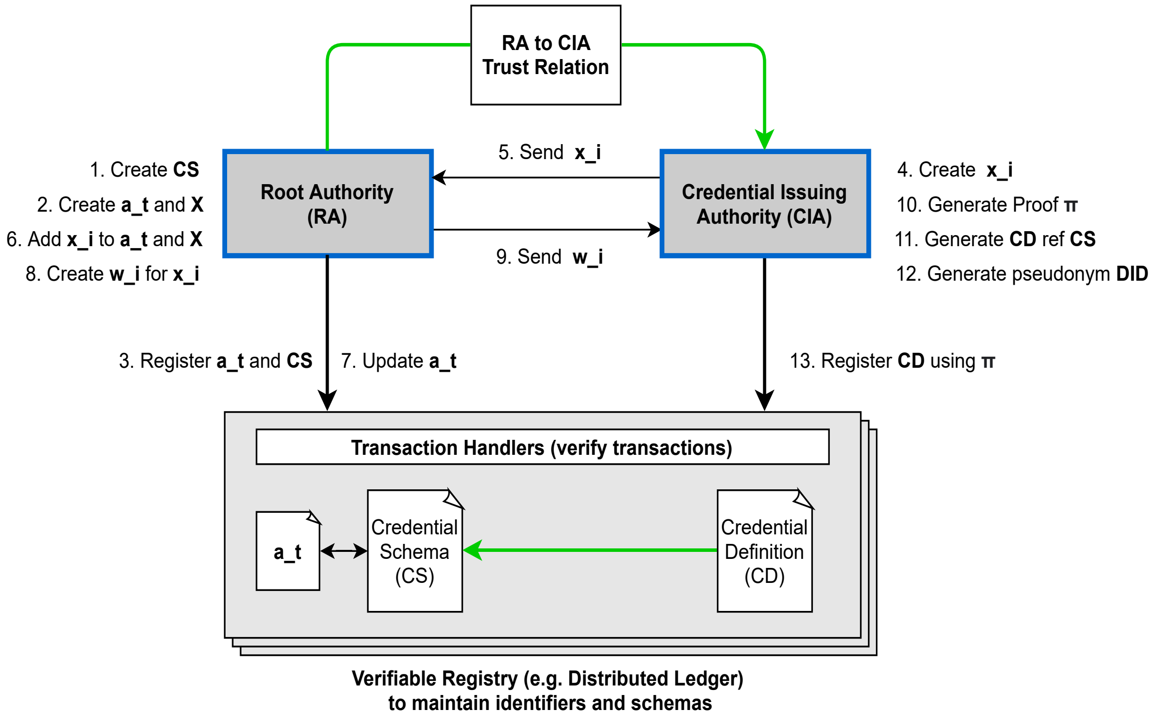
$$
\begin{array}{ll}
k, \rho_x, \rho_k \xleftarrow{R} [-B, B]\,; & z = g^x h^{\rho_x} \\
A_g = g^k h^{\rho_k}\,; & A_{w_x} = w_x^k \\
l \leftarrow H_{\text{prime}}(w_x, a_t, z, A_g, A_{w_x})\,; & c \leftarrow H(l) \\
q_x \leftarrow \lfloor (k + c \cdot x)/l \rfloor\,; & q_\rho \leftarrow \lfloor (\rho_k + c \cdot \rho_x)/l \rfloor \\
r_x \leftarrow (k + c \cdot x) \mod l\,; & r_\rho \leftarrow (\rho_k + c \cdot \rho_x) \mod l \\
\pi \leftarrow \{l, z, g^{q_x} h^{q_\rho}, w_x^{q_x}, r_x, r_\rho\} & \\
\text{return}: \;\; \pi &
\end{array}
$$

VerifyProof $(\, w_x, a_t, \pi \,)$:

$$
\begin{array}{ll}
\{l, z, Q_g, Q_{w_x}, r_x, r_\rho\} \leftarrow \pi\,; & c = H(l) \\
A_g \leftarrow Q_g^l g^{r_x} h^{r_\rho} z^{-c}\,; & A_w \leftarrow Q_{w_x}^l w_x^{r_x} a_t^{-c} \\
\mathit{Verify} \;\; r_x, r_\rho \in [l]\,; & l = H_{\text{prime}}(w_x, a_t, z, A_g, A_w) \\
\text{return}: \;\; \{0,1\} &
\end{array}
$$

High-Level Overview of A-PoA

Cryptographic Accumulator and Zero Knowledge Proof of Knowledge of Exponent



Security

Adversary Model

- $\mathscr{A}_1$ *(Network Eavesdropper)*: Suppose a hostile network participant, acting as $\mathscr{A}_1$, intends to eavesdrop and modify or decrypt all messages m exchanged throughout the introduced protocols.
  $\Rightarrow$ Authenticated Encryption for DID communication relying on security of e.g. asymmetric cryptography.
- $\mathscr{A}_2$ *(Unforgeability)*: Suppose $\mathscr{A}_2$ is a malicious adversary, trying to **forge** a valid proof of an invalid identity. $\mathscr{A}_2$'s efforts can be based on previously seen witness pairs (x, w) (only w is known by $\mathscr{A}_2$) and accumulator values a.
  $\Rightarrow$ Accumulator collision ($a'^{x'} = g^{x_1,\dots,x_n} \mod n$) protected by *strong* RSA assumption. + Revocation of x' causes adversary with collision to authenticate.
- $\mathscr{A}_3$ *(Cheating Verifier)*: Suppose $\mathscr{A}_3$ is a **malicious** Verifier V that verifies the authentication proofs of a prover P. Then, $\mathscr{A}_3$ does not learn anything else than the validity of the statement proven by P.
  $\Rightarrow$ Computational indistinguishable transcripts T_{sim} and T_{real} + *Fiat Shamir* heuristic in NI-ZKPOKE [3].

Evaluation

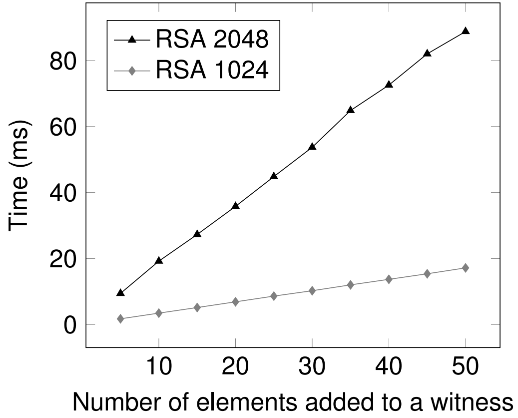


Figure: Duration (ms) of adding elements x_i to an already existing witness w_t for a single holder witness update (numbers averaged by 100 repetitions).

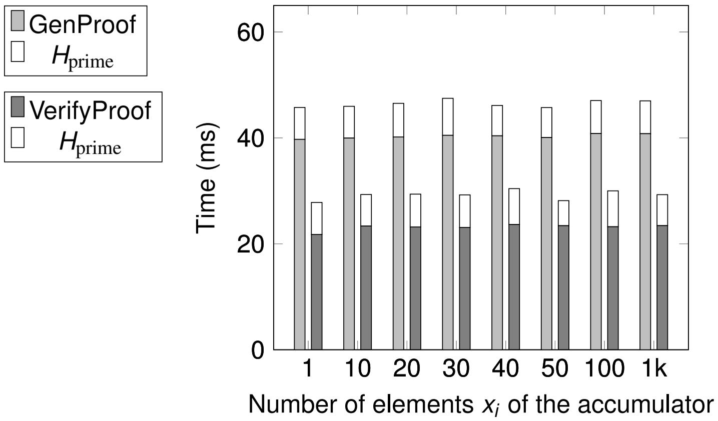


Figure: GenProof (left, lightgray) and VerifyProof (right, gray) execution times (ms) of the NI-ZKPoKE protocol with 128-Bit polynomial time H_{prime} hash function and the RSA-accumulator (2048-Bit).

Evaluation

Table: Mean execution times (ms) of A-PoA with a 2048-Bit RSA- accumulator ($\lambda = 128$), k=50 elements, and 128-Bit hashes.

Protocol	**Function**	**Time (ms)**	**COM**	**Big $\mathcal{O}$**
Authorization	Prime Gen.	309.81	k	$\mathcal{O}(n)$
	Acc. Gen.	88.80	1	$\mathcal{O}(1)$
	Wit. Gen.	4383.60	k	$\mathcal{O}(n)$
Authentication	GenProof	40.06	1	$\mathcal{O}(1)$
	VerifyProof	23.42	0	
Revocation	Acc. Revoke	2244.85	(0-k)	$\mathcal{O}(n)$
$\sum$**Authorization**	N/A	4782.21	$2k+1$	$\mathcal{O}(n)$
$\sum$**Authentication**	N/A	63.48	1	$\mathcal{O}(1)$
$\sum$**Revocation**	N/A	2244.85	(0-(k-1))	$\mathcal{O}(n)$

Related Work

- Asynchronous accumulators with backwards compatibility to build a distributed Public Key Infrastructure (PKI) [7]. Authorized key pairs certify services and remain verifiable with accumulator membership via the ledger.
  $\Rightarrow$ Authorization by membership verification but missing anonymous setup.
- Disposable dynamic accumulator in the context of a pseudonym-based signature scheme to establish privacy-preserving electronic IDs [8]. One time token to authenticate which preserves anonymity, unlinkability, and backward unlinkability.
  $\Rightarrow$ One time tokens require generation while our scheme keeps witnesses for authentication. Our schema requires pseudonym DIDs with new verifiers.

Acknowledgements

- This work has received funding by the European Union's Horizon 2020 Research and Innovation Programme through the nIoVe project (https://www.niove.eu/) under grant agreement no. 833742.
- With the support of the Technische Universität München - Institute for Advanced Study, funded by the German Excellence Initiative and the European Union Seventh Framework Programme under grant agreement no. 291763.

References I

- J. Camenisch and A. Lysyanskaya, “Dynamic accumulators and application to efficient revocation of anonymous credentials,” in *Annual International Cryptology Conference*. Springer, 2002, pp. 61–76.
- C.-P. Schnorr, “Efficient identification and signatures for smart cards,” in *Conference on the Theory and Application of Cryptology*. Springer, 1989, pp. 239–252.
- D. Boneh, B. Bünz, and B. Fisch, “Batching techniques for accumulators with applications to iops and stateless blockchains,” in *Annual International Cryptology Conference*. Springer, 2019, pp. 561–586.
- M. Fischlin, “Trapdoor commitment schemes and their applications.” Ph.D. dissertation, Citeseer, 2001.
- D. Pei, A. Salomaa, and C. Ding, *Chinese remainder theorem: applications in computing, coding, cryptography*. World Scientific, 1996.
- L. Chen and T. P. Pedersen, “New group signature schemes,” in *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 1994, pp. 171–181.
- L. Reyzin and S. Yakoubov, “Efficient asynchronous accumulators for distributed pki,” in *International Conference on Security and Cryptography for Networks*. Springer, 2016, pp. 292–309.
- M. Hölzl, M. Roland, O. Mir, and R. Mayrhofer, “Disposable dynamic accumulators: toward practical privacy-preserving mobile eids with scalable revocation,” *International Journal of Information Security*, vol. 19, no. 4, pp. 401–417, 2020.

References II

- J. Camenisch and A. Lysyanskaya, "Signature schemes and anonymous credentials from bilinear maps," in *Annual international cryptology conference*. Springer, 2004, pp. 56–72.
- J. Camenisch, M. Kohlweiss, and C. Soriente, "An accumulator based on bilinear maps and efficient revocation for anonymous credentials," in *International workshop on public key cryptography*. Springer, 2009, pp. 481–500.
- W. W. W. Consortium *et al.*, "Verifiable credentials data model 1.0: Expressing verifiable information on the web," *https://www. w3. org/TR/vc-data-model/?# core-data-model*, 2019.
- D. Reed, M. Sporny, D. Longley, C. Allen, R. Grant, M. Sabadello, and J. Holt, "Decentralized identifiers (dids) v1. 0," *Draft Community Group Report*, 2020.
- C. Adams, S. Farrell, T. Kause, and T. Mononen, "Internet x. 509 public key infrastructure certificate management protocol (cmp)," RFC 4210 (Proposed Standard), Tech. Rep., 2005.
- M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams, "X. 509 internet public key infrastructure online certificate status protocol-ocsp," 1999.
- D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, W. T. Polk *et al.*, "Internet x. 509 public key infrastructure certificate and certificate revocation list (crl) profile." *RFC*, vol. 5280, pp. 1–151, 2008.
- D. Boneh and V. Shoup, "A graduate course in applied cryptography," *Recuperado de https://crypto. stanford. edu/˜ dabo/cryptobook/BonehShoup_0_4. pdf*, 2017.

Questions?

Thank You for listening.

Contact

- Jan Lauinger: **jan.lauinger@tum.de**
- Jens Ernstberger: **jens.ernstberger@tum.de**
- Emanuel Regnath: **emanuel.regnath@tum.de**
- Mohammad Hamad: **mohammad.hamad@tum.de**
- Sebastian Steinhorst: **sebastian.steinhorst@tum.de**

Backup Slides

Boneh et al. NI-ZKPoKE [3]

Completeness

Accept if: $\Rightarrow Q_g^l \cdot g^{r_x} \cdot h^{r_\rho} = (g^{q_x} \cdot h^{q_\rho})^l \cdot g^{r_x} \cdot h^{r_\rho} = g^{q_x \cdot l + r_x} \cdot h^{q_\rho \cdot l + r_\rho} = g^{s_x} \cdot h^{s_\rho} = g^{k + c \cdot x} \cdot h^{\rho_k + c \cdot \rho_x} = g^k \cdot h^{\rho_k} \cdot g^{x \cdot c} \cdot h^{\rho_x \cdot c} = A_g \cdot z^c$
$\Rightarrow Q_u^l \cdot u^{r_x} = (u^{q_x})^l \cdot u^{r_x} = u^{q_x \cdot l + r_x} = u^{s_x} = u^{k + c \cdot x} = u^k \cdot u^{x \cdot c} = A_u \cdot w^c$

Soundness (Extractor)

1. Extractors to extract x, ρ such that $z = g^x \cdot h^\rho$ and $g^{s_1} \cdot h^{s_2} = A_g \cdot z^c$.
2. Set $R \leftarrow \{\}$ and sample $s_1, s_2 \overset{R}{\leftarrow} [0, 2^\lambda]$.
3. Sample $l \overset{R}{\leftarrow} \mathrm{Primes}(\lambda)$, $c \overset{R}{\leftarrow} [0, 2^\lambda]$ and send s_1, s_2, l, c to $\mathscr{A}_1$.
4. Obtain output Q_g, Q_u, r_1, r_2 from $\mathscr{A}_0$. If transcript is accepting ($Q_g^l \cdot g^{r_1} \cdot h^{r_2} = A_g \cdot z^c$ and $Q_u^l \cdot u^{r_1} = A_u \cdot w^c$) then update $R \leftarrow R \cup \{(r_1, r_2, l, c)\}$. Otherwise return to step 2.
5. Use CRT to compute $s_1 = r_1^{(i)} \mod l^{(i)}$ and $s_2 = r_2^{(i)} \mod l^{(i)}$, for each $(r_1^{(i)}, r_2^{(i)}, l^{(i)}) \in R$. If $u^{s_1} = A_u \cdot w^c$ then output s_1, otherwise return to step 2.
6. Repeat for s_1', s_2', c', so that $x = \Delta s_1 / \Delta c = (s_1 - s_1')/(c - c')$ and $\rho = \Delta s_2 / \Delta c = (s_2 - s_2')/(c - c')$, with extraction based on $u^{s_1} = A_u \cdot w^c$ and $u^{s_1'} = A_u \cdot w^{c'}$, thus $(u^x)^{\Delta c} = w^{\Delta c} \Rightarrow u^x = w$.

Boneh et al. NI-ZKPoKE [3]

Zero Knowledge (Simulator)

1. $\widetilde{c} \xleftarrow{R} [0, 2^{\lambda}]$, $\widetilde{l} \xleftarrow{R} \mathrm{Primes}(\lambda)$
2. $\widetilde{z} \leftarrow h^{\widetilde{\rho}}$, $\rho \xleftarrow{R} [B]$
3. $\widetilde{q}_x, \widetilde{q}_r \xleftarrow{R} [B]^2$
4. $\widetilde{r}_x, \widetilde{r}_\rho \in [l]^2$
5. $\widetilde{Q}_g \leftarrow g^{\widetilde{q}_x} \cdot h^{\widetilde{q}_\rho}$, $\widetilde{Q}_u \leftarrow u^{\widetilde{q}_x}$
6. $\widetilde{A}_g \leftarrow \widetilde{Q}_g^l \cdot g^{\widetilde{r}_x} \cdot h^{\widetilde{r}_\rho} \cdot z^{-\widetilde{c}}$, $\widetilde{A}_u \leftarrow \widetilde{Q}_u^l \cdot u^{\widetilde{r}_x} \cdot w^{-\widetilde{c}}$

$\Rightarrow$ $(\widetilde{z}, \widetilde{A}_g, \widetilde{A}_u, \widetilde{c}, \widetilde{l}, \widetilde{Q}_g, \widetilde{Q}_w, \widetilde{r}_x, \widetilde{r}_\rho)$ is statistically indistinguishable from $(z, A_g, A_u, c, l, Q_g, Q_w, r_x, r_\rho)$

Motivation

Anonymous Credentials

Group Signature Scheme [9]

1. Key generation procedure (Key generation for revocation management [10])
2. Join protocol between member and group manager (Holder obtains signature of group manager on committed values)
3. Sign algorithm for member to sign messages (Proving Knowledge of a signature)
4. Algorithm to verify group signatures (manager/member) for validity using the group's public key
5. Opening algorithm for group manager to identify member for revocation

Involved roles → Issuer, Prover/Holder, Verifier

Hyperledger Indy[1] implementation of anonymous credentials

- Based on W3C standard of Verifiable Credentials [11] and Decentralized Identifiers [12]
- Endorser Role (Ledger write privilege) to register *Credential Schema* (*CS*) → *CS* defines attributes of a credential
- Endorser Role (Ledger write privilege) to register *Credential Definition* (*CD*) → *CD* defines public cryptographic data required for credential validation (attributes/validity, revocation) and references *CS*

→ *Indy* anonymous credential protocol[2] supports anonymous credentials from various issuers to multiple holders

[1] https://www.hyperledger.org/use/hyperledger-indy

[2] https://github.com/hyperledger-archives/indy-crypto/blob/master/libindy-crypto/docs/AnonCred.pdf

Code: 2048-Bit RSA Accumulator

```
class RSA2048_Accumulator():
    def __init__(self, elements):
        self.p, self.q = 113670..., 1472657... # 1024 primes p and q
        self.modulus, self.phi = 1673972..., 16739728... # 2048 bit N as RSA modulus p*q = N, PHI (p-1)(q-1)
        self.g_acc, self.value = pow(3,2), 9 # g=3 works, look at the script pick_g-2.py, # squaring g mod N
        self.elements = elements # generate x elements, hash to primes
    def add(self, element):
        if element not in self.elements:
            old_value = self.value
            self.value = pow(self.value, element, self.modulus)
            return old_value
    def gen_witness(self, element):
        wit = self.g_acc
        for x in self.elements:
            if x == element: continue
            prime_item = self._items.as_prime(item)
            wit = pow(wit, x, self.modulus)
        return wit
    def verify_membership(self, wit_t, x_t):
        return pow(wit_t, x_t, self.modulus) == self.value
    def remove(self, x):
        if x in self.tails:
            old_value = self.value
            self.tails.remove(x)
            base = modinv(x, self.phi)
            invers = pow(base, 1, self.phi)
            self.value = pow(self.value, invers, self.modulus)
            return self.value
```

Code: NI-ZKPoKE

```python
def Com(g, x, h, r, q):
    return (pow(g,x,q) * pow(h,r,q)) % q

def gen_proof(security, acc, u, x):
    s = number.getRandomRange(2, acc.modulus-1)
    g = 2
    h = pow(g, s, acc.modulus)
    w = pow(u, x, acc.modulus) # u^x = w

    B = pow(2, 2*security, acc.modulus)*acc.bit_size+1
    k = number.getRandomRange(3, B) # random numbers max 64-bit size if security lambda of 128 possible
    ro_x = number.getRandomRange(3, B)
    ro_k = number.getRandomRange(3, B)

    z = Com(g, x, h, ro_x, acc.modulus)
    A_g = Com(g, k, h, ro_k, acc.modulus)
    A_u = pow(u, k, acc.modulus)
    l = Hash2Prime(u, w, z, A_g, A_u, "md5")
    hf = hashlib.md5(str(l).encode("utf-8"))
    c = int(hf.hexdigest(), 16)

    q_x = (k + c * x) // l # proof values:
    r_x = (k + c * x) % l
    q_ro = (ro_k + c * ro_x) // l
    r_ro = (ro_k + c * ro_x) % l
    Q_g = Com(g, q_x, h, q_ro, acc.modulus)
    Q_u = pow(u, q_x, acc.modulus)
    return [l, z, Q_g, Q_u, r_x, r_ro]
```

Code: NI-ZKPoKE

```python
def verify_proof(pi, u, acc_value):
    # value extraction of proof params
    hf = hashlib.md5(str(l).encode("utf-8"))
    c = int(hf.hexdigest(), 16)

    base = modinv(z, n)
    base2 = modinv(w, n)
    A_g_ver = ( pow(Q_g, l, n)*pow(g, r_x, n)*pow(h, r_ro, n)*pow(base, c, n)  ) % n
    A_u_ver = ( pow(Q_u, l, n)*pow(u, r_x, n)*pow(base2, c, n) ) % n
    l_ver = Hash2Prime(u, w, z, A_g_ver, A_u_ver, "md5")

    compare1 = pow(Q_u, l_ver, n)*pow(u, r_x, n) % n == A_u_ver*pow(w, c, n) % n
    compare2 = pow(Q_g, l_ver, n)*Com(g, r_x, h, r_ro, n) % n == A_g_ver*pow(z, c, n) %n
    return compare1 and compare2

def Hash2Prime(u, w, z, A_g, A_u, hashtype):
    chal = str(u)+str(w)+str(z)+str(A_g)+str(A_u)
    h = hashlib.md5() # or hashlib.sha256()
    h.update(chal.encode("utf-8"))
    nonce, temp = 0, 0
    while True:
        h.update(str(nonce).encode("utf-8"))
        nonce += 1
        c = int(h.hexdigest(), 16)
        if number.isPrime(c):
            return c
```

Motivation

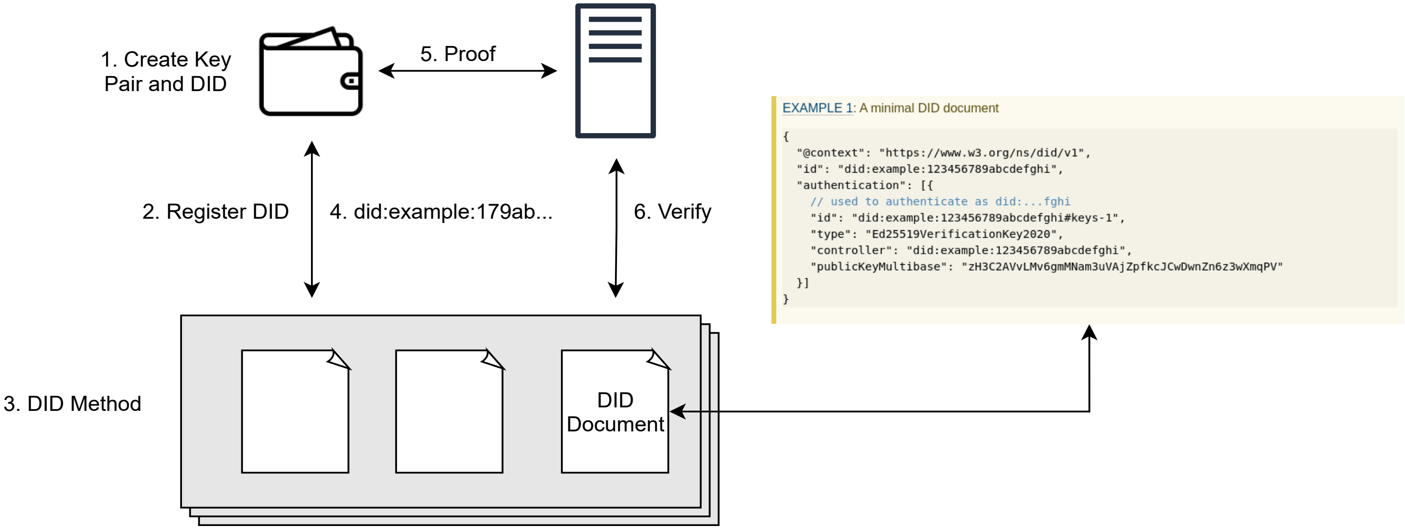
Credential Schema and Credential Definition Test Code Example[3]

```
let mut credential_schema_builder = Issuer::new_credential_schema_builder().unwrap();
credential_schema_builder.add_attr("name").unwrap();
credential_schema_builder.add_attr("age").unwrap();
let credential_schema = credential_schema_builder.finalize().unwrap();

let mut non_credential_builder = NonCredentialSchemaBuilder::new().unwrap();
non_credential_builder.add_attr("master_secret").unwrap();
let non_credential_schema = non_credential_builder.finalize().unwrap();

let (cred_pub_key, cred_priv_key, cred_key_correctness_proof) =
                Issuer::new_credential_def(&credential_schema, &non_credential_schema, true).unwrap();

let mut credential_values_builder = CredentialValuesBuilder::new().unwrap();
credential_values_builder.add_value_hidden("master_secret", &prover_mocks::master_secret().value().unwrap()).unwrap();
credential_values_builder.add_value_known("name", &string_to_bignumber("indy-crypto")).unwrap();
credential_values_builder.add_dec_known("age", "25").unwrap();

let cred_values = credential_values_builder.finalize().unwrap();

let credential_nonce = new_nonce().unwrap();

let (blinded_credential_secrets, credential_secrets_blinding_factors, blinded_credential_secrets_correctness_proof) =
                Prover::blind_credential_secrets(&cred_pub_key, &cred_key_correctness_proof, &cred_values, &credential_nonce).unwrap();
```

[3] https://github.com/hyperledger-archives/indy-crypto/blob/master/libindy-crypto/src/cl/issuer.rs

Motivation

Decentralized Identifiers (DIDs)

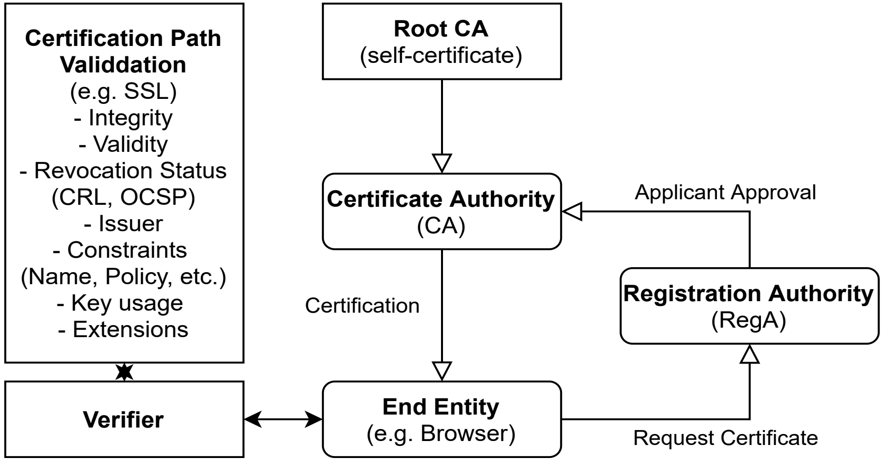


Verifiable Registry (e.g. Distributed Ledger -> Ethereum, Sovrin, Cardano, etc.)

Motivation

X.509 Public Key Infrastructure (PKI)

- Wikipedia: PKI is set of roles, policies, hardware, software and procedures to create, manage, distribute, use, store and revoke digital certificates
- Certificate Management Protocol (CMP) [13]
- Online Certificate Status Protocol (OCSP) [14]
- Certification Path Validation [15]



Notations Overview

Symbol	Definition
t	Discrete time / operation counter
X_t	Tails file at time t
X_0	Initial tails file at $t = 0$
x_i	i-th element of the tails file
w_i	Witness value associated with i-th element
a_t	Accumulator value at time t
p, q, p', q'	Large λ-bit prime numbers
$\mathbb{Z}_n$	$n \in \mathbb{N}$, $\mathbb{Z}_n = \{1, 2, \ldots, n\}$ = ring of integers mod n
$\mathbb{Z}_n^*$	Set of invertible elements in $\mathbb{Z}_n$
$\mathbb{G}_?$	Generic group of unknown order $\{(\mathbb{Z}_n)^*/\{\pm 1\}\}$
[-B, B]	Range of integers such that $\|\mathbb{G}\|/B$ is negligible
$\mathbb{QR}_n$	Subgroup of quadratic residues of $\mathbb{G}_?$, contains $x \in \mathbb{Z}_n^*$, if $\exists\ y \in \mathbb{Z}_n^*$, with $y^2 \equiv x \pmod{n}$
$\phi(n)$	Number of elements in $\mathbb{Z}_n^*$, if $p \cdot q = n$ then $\phi(n) = (p-1) \cdot (q-1)$
g, h	Generator of a Group $\mathbb{G}_?$

Notations of Groups used in Cryptography

Using values out of different types of groups allow to calculate different equations. Some calculations are believed to be hard to solve, e.g. (taken from [16]):

1. Let g be a generator of $\mathbb{Z}_p^*$. Given $x \in \mathbb{Z}_p^*$ find an r such that $x = g^r \mod p$. This is known as the *discrete log* problem.
2. Let g be a generator of $\mathbb{Z}_p^*$. Given $x, y \in \mathbb{Z}_p^*$ where $x = g^{r1}$ and $y = g^{r2}$. Find $z = g^{r1 \cdot r2}$. This is known as the *Diffie-Hellman* problem.

With finite cyclic group $\mathbb{G}$, $\mathbb{G}^*$ represents the set of generators of $\mathbb{G}$.

Cyclic Group $\mathbb{Z}_p^*$, if $g \in \mathbb{Z}_p^*$ exist with property $\mathbb{Z}_p^* = \{1, g, g^2, g^3, \ldots, g^{p-2}\}$, then g is called generator. Elements in $\mathbb{Z}_p^*$ are invertible $(x, a, \in \mathbb{Z}_p^*$ with $x \cdot a = 1 \mod p)$. Inverse of x is denoted as x^{-1}.

Example: Select $g = 3$ in $\mathbb{Z}_7^*$, $\Longrightarrow \langle 3 \rangle = \{1, 3, 3^2, 3^3, 3^4, 3^5, 3^6\} \equiv \{1, 3, 2, 6, 4, 5\} (\mod 7) = \mathbb{Z}_7^*$.

An element $x \in \mathbb{Z}_p^*$ is called a Quadratic Residue (QR) if it has a square root in $\mathbb{Z}_p$. Let g be a generator of $\mathbb{Z}_p^*$. Let $x = g^r$ for some integer r. Then x is a QR in $\mathbb{Z}_p$ if and only if r is even. Since $x = g^r$ is a QR if and only if r is even, it follows that exactly half the elements of $\mathbb{Z}_p$ are QR's. Testing if an element is a QR in $\mathbb{Z}_n$ is believed to be hard if factorization of n unknown [16].

Notations of Groups used in Cryptography

RSA-Accumulator Value

Requires generator $g_{\mathrm{acc}} \stackrel{R}{\leftarrow} \mathbb{Z}_n^*$, with $n = p \cdot q$, $p = 2 \cdot p' + 1$, and $q = 2 \cdot q' + 1$, where p' and q' are *Sophie Germain* primes (p such that $2p+1$ is prime).

Finding *Sophie Germain* primes, calculate safe prime of $k =$512, 768, 1024, 2048, 4096 bits, repeatedly try to find a random prime p of k-1 bits, until $2p+1$ is prime (or repeatedly find a random k-bit prime p, until $(p\text{-}1)/2$ is prime). *Miller-Rabin* primality test to find primes faster.

[1] requires $a_t \in \mathbb{QR}_n, a_t \neq 1$, so a_t is QR in $\mathbb{Z}_n$.
$\Rightarrow$ Construction in [3] requires $\mathbb{G}_?$ as generic group of unknown order $\{(\mathbb{Z}_n)^*/\{\pm 1\}\}$.

Create random values in $\mathbb{QR}_n$: pick a random number r relatively prime to n, and compute $r^2 \mod n$; that's a random value in $\mathbb{QR}_n$; $\mathtt{size}(\mathbb{QR}_n) = (p-1)(q-1)/4 \approx N/4$

$\Rightarrow a_t = (g_{\mathrm{acc}})^2 \mod n$, with $g_{\mathrm{acc}} \stackrel{R}{\leftarrow} \mathbb{G}_?$

Generation of Accumulator Elements

RSA-Accumulator Elements

[1] requires accumulator elements $x \in primes$, with $x \neq p', q'$ and $A \leq x \leq B$, where A, B can be chosen with arbitrary polynomial dependence (Linear Independence of Polynomials, arbitrary constant coefficients, distinct positive integers as grades), respecting the security parameter λ, as long as $2 < A$ and $B < A^2$.

Generation of $x \leftarrow H_{\text{prime}}(x', \lambda)$ using H_{prime} to achieve collision resistance of accumulator elements.

Tails file X_t = $\{x_1, x_2, ..., x_i\}$, with $i = \{1, 2, \ldots, N\}$ contains accumulator elements.

Notations Overview

Symbol	Definition
t	Discrete time / operation counter
X_t	Tails file at time t
X_0	Initial tails file at $t = 0$
x_i	i-th element of the tails file
w_i	Witness value associated with i-th element
a_t	Accumulator value at time t
p, q, p', q'	Large λ-bit prime numbers
$\mathbb{Z}_n$	$n \in \mathbb{N}$, $\mathbb{Z}_n = \{1, 2, \ldots, n\}$ = ring of integers mod n
$\mathbb{Z}_n^*$	Set of invertible elements in $\mathbb{Z}_n$
$\mathbb{G}_?$	Generic group of unknown order $\{(\mathbb{Z}_n)^* / \{\pm 1\}\}$
[-B, B]	Range of integers such that $\lvert\mathbb{G}\rvert / B$ is negligible
$\mathbb{QR}_n$	Subgroup of quadratic residues of $\mathbb{G}_?$, contains $x \in \mathbb{Z}_n^*$, if $\exists\ y \in \mathbb{Z}_n^*$, with $y^2 \equiv x \pmod n$
$\phi(n)$	Number of elements in $\mathbb{Z}_n^*$, if $p \cdot q = n$ then $\phi(n) = (p-1) \cdot (q-1)$
g, h	Generator of a Group $\mathbb{G}_?$