

Design, Implementation, and Assessment of an Improved Revocation Mechanism for Verifiable Credentials

Researcher: Srajit Sakhujā
PhD Supervisor: Mr. Felix Hoops

12th February 2024

Chair of Software Engineering for Business Information Systems (sebis)
Department of Computer Science
School of Computation, Information and Technology (CIT)
Technical University of Munich (TUM)
www.matthes.in.tum.de

Presentation Outline

A Quick Recap

- Core Concepts
 - Verifiable Credentials (VCs), Status List 2021
- Problem Statement
 - Design Goals for the Revocation Data Structure
 - A Suitable Data Structure, The AMQFilter Interface, Benchmarks

(Further) Solving the Problem

- Absence of an E2E Architecture
- Architectural Extensions of GX Credentials
 - GX Credentials: Architectural Recap, Drawbacks
 - Design Principles for the Architectural Overhaul
 - Company Membership Service (CMS)
 - Revocation, Suspension, Verification

A Brief Demo

Design, Implementation, and Assessment of an Improved Revocation Mechanism for Verifiable Credentials

Verifiable Credentials (VCs)

Claims + Digital Signatures

VCs are a foolproof way for

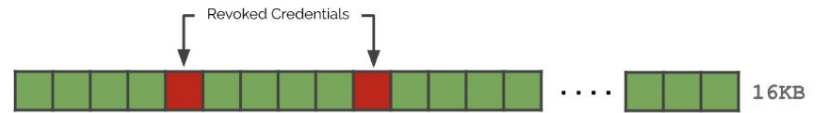
- **Real World Entities** to stake **claims** about themselves
- **Issuers** to ensure that the **claims** are **tamper-proof**
- **Verifiers** to ensure that the **claims** are **not falsehoods**

Revocation Mechanism - Status List 2021

VCs = Cryptographically-ensured **Indefinite** ownership over claims

An appendage to enable revocation / temporary suspension of VCs

Bitstring with 1s indicating revoked VCs



Status List with 2 revocations;
16 KB = 131,072 VCs

Three Design Considerations for any Revocation Mechanism - Privacy, Scalability, Minimum Propagation Delay



Privacy: (a) Only the issuer, the holder, and the verifier should know that the VC has been revoked, (b) The issuer should not know who the verifier is



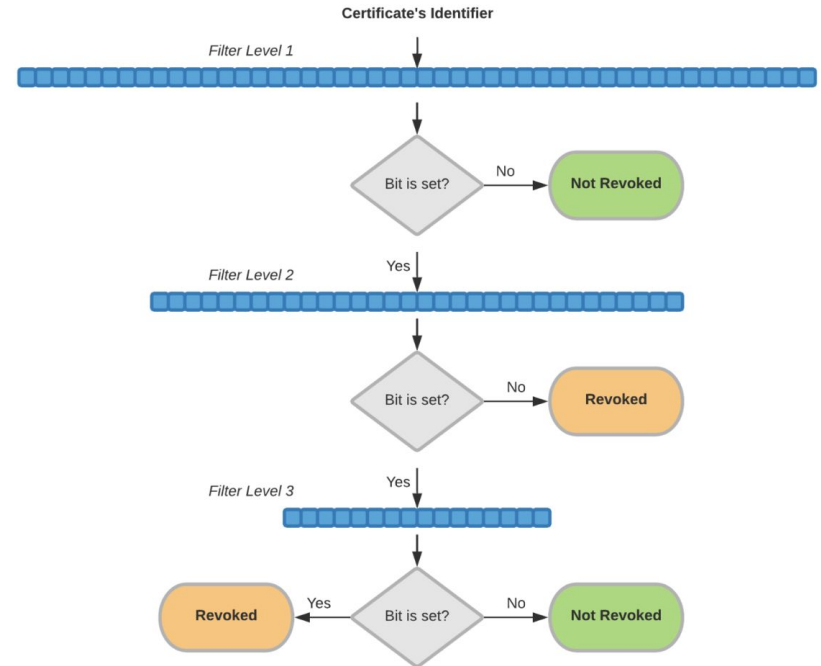
Scalability: Designed for planet-scale - universities issuing degrees over the course of a century, the American DMV



Minimal Propagation Delay: minimal delay to minimize the misuse of an effectively invalid VCs

Drawing Inspiration from TLS Certificates: CRLite and the concept of Cascading Filters

- Research from 2020 by **Mozilla's Security Group**
- Introduced as a mechanism for **revoking SSL/TLS certificates**
- The underlying data structure is a **Bloom Filter**
- A Bloom Filter is an **AMQ - Approximate Membership Query** data structure
 - (potentially) returns **False positives**
 - No means No | Yes means maybe Yes



Drawing Inspiration from TLS Certificates: CRLite and the concept of Cascading Filters

The Novel
Idea in CRLite

AMQFilter Interface

Cascading

Deterministic Set
Membership
Data Structure

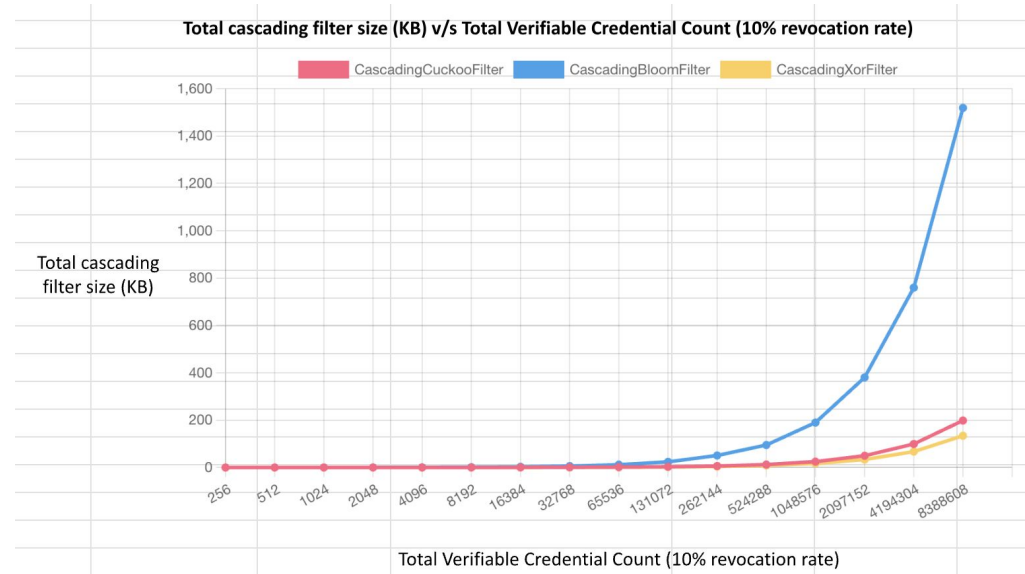
prone to false positives -
probabilistic in answers to
membership queries

```
void insert(E element)  
boolean maybeContains(E  
element)
```

no false positives/negatives -
deterministic in answers to
membership queries

Overview of the Filter Benchmarking Setup and results

- Data generation: A randomly generated set of integer values; size $\in [2^8, 2^{23}]$; revocation rate 10%
- A interface-based benchmarking code where the underlying AMQFilter can rapidly be changed
 - Bloom Filters
 - Cuckoo Filters
 - **XOR Filters**
- Results (raw data on next slide):
 - Computation time ranging from 60 ms for 100k VCs to 4000 ms for 8M VCs
 - Filter size (see adjacent graph)



Achieving Scale and Privacy

Total VC Count	Filter Type	Revoked VC count	# layers	Layer-wise size (bits)	Total cascading filter size (bits)	Total cascading filter size (bytes)	Total cascading filter size (KB)	Layer-wise FP count	Computation time (ms)
65536	CascadingCuckooFilter	6554	3	[12288 384 12]	12684	1585.5	1.59	[227 6 0]	39
	CascadingBloomFilter		3	[94232 536 104]	94872	11859	11.86	[37 7 0]	34
	CascadingXorFilter		4	[8115 351 87 57]	8610	1076.25	1.08	[242 27 2 0]	42
131072	CascadingCuckooFilter	13108	4	[24576 192 48 6]	24822	3102.75	3.11	[106 25 1 0]	64
	CascadingBloomFilter		3	[188464 1168 144]	189776	23722	23.73	[81 10 0]	63
	CascadingXorFilter		4	[16176 621 117 57]	16971	2121.375	2.13	[461 51 1 0]	62
262144	CascadingCuckooFilter	26215	3	[49152 384 96]	49632	6204	6.21	[172 46 0]	102
	CascadingBloomFilter		3	[376912 2048 26144]	405104	50638	50.64	[142 1818 0]	112
	CascadingXorFilter		4	[32298 1206 183 63]	33750	4218.75	4.22	[936 104 6 0]	127
524288	CascadingCuckooFilter	52429	4	[98304 768 96 6]	99174	12396.75	12.4	[433 56 1 0]	226
	CascadingBloomFilter		3	[753808 4504 848]	759160	94895	94.9	[313 59 0]	294
	CascadingXorFilter		5	[64542 2271 318 60 57]	67248	8406	8.41	[1802 215 4 2 0]	262
1048576	CascadingCuckooFilter	104858	4	[196608 1536 384 6]	198534	24816.75	24.82	[713 138 1 0]	454
	CascadingBloomFilter		3	[1507616 8504 1584]	1517704	189713	189.72	[591 110 0]	658
	CascadingXorFilter		5	[129030 4719 576 75 57]	134457	16807.125	16.81	[3793 424 16 2 0]	439
2097152	CascadingCuckooFilter	209716	4	[393216 3072 768 6]	397062	49632.75	49.64	[1530 476 3 0]	1089
	CascadingBloomFilter		3	[3015224 24304 3240]	3042768	380346	380.35	[1690 225 0]	1048
	CascadingXorFilter		6	[258006 9030 1029 96 57 57]	268275	33534.375	33.54	[7297 792 34 1 1]	1128
4194304	CascadingCuckooFilter	419431	4	[786432 6144 768 12]	793356	99169.5	99.17	[3039 459 4 0]	1942
	CascadingBloomFilter		3	[6030432 38192 7104]	6075728	759466	759.47	[2656 494 0]	2159
	CascadingXorFilter		7	[515955 18228 2109 117 63 57 57]	536586	67073.25	67.08	[14774 1671 50 6]	2328
8388608	CascadingCuckooFilter	838861	5	[1572864 12288 1536 24 6]	1586718	198339.75	198.34	[6060 983 11 3 0]	5390
	CascadingBloomFilter		5	[12060840 75272 12440 88 32]	12148672	1518584	1518.59	[5235 865 6 2 0]	4482
	CascadingXorFilter		5	[1031853 36480 4026 201 78]	1072638	134079.75	134.08	[29615 3230 119]	4319

How does it all fit together?

Where will this Data Structure be stored?

Can it also support other use cases
e.g., suspensions?

How does it all fit together?

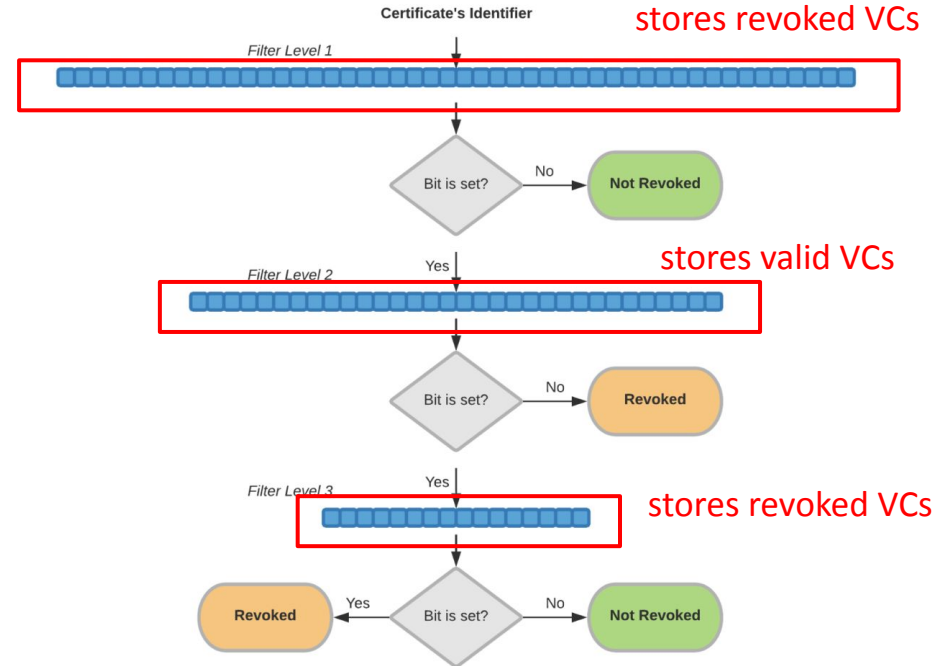
Who will generate the Cascading
Filter?

How will a verifier obtain it?

Drawing Inspiration from TLS Certificates: CRLite and the concept of Cascading Filters

Insertion algorithm:

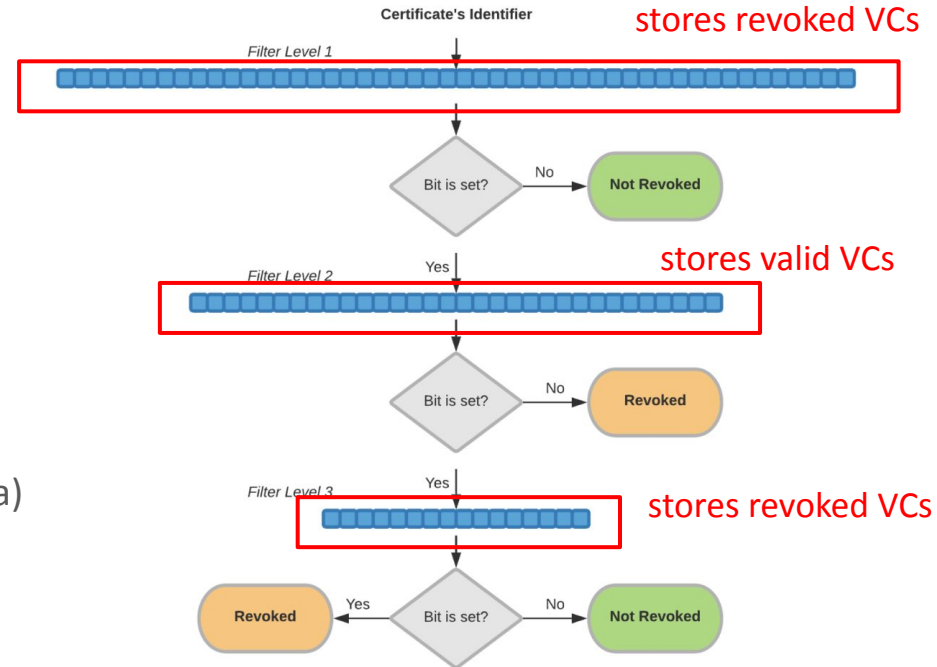
1. Add all **revoked VCs** to the **filter level 1**
2. Iterate over all **valid VCs** to find **level 1's FP set**
3. Add all the **VCs in the FP set** to **filter level 2**
4. Iterate over all **revoked VCs** to find **level 2's FP set**
5. Repeat until **FP set is empty** for a level
(Termination criteria)

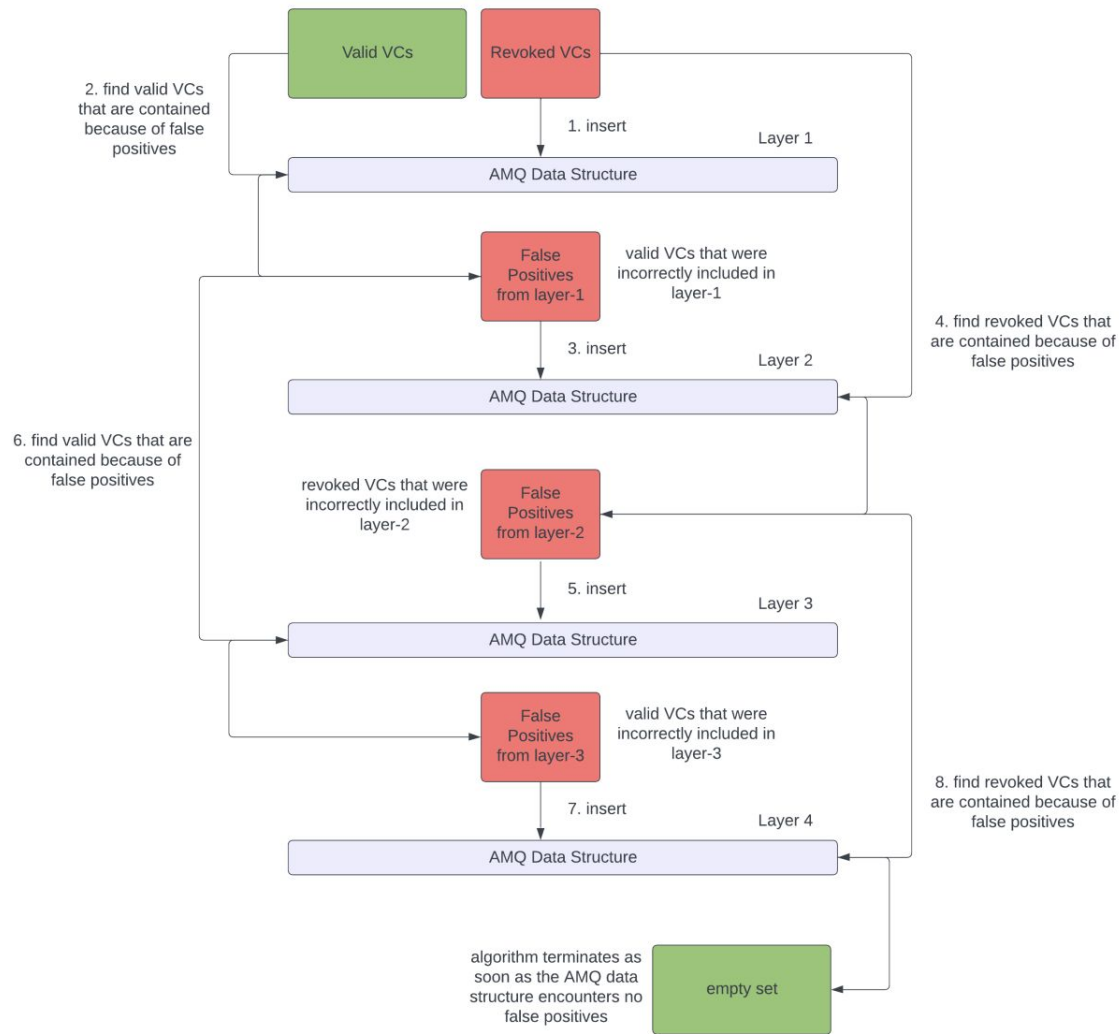


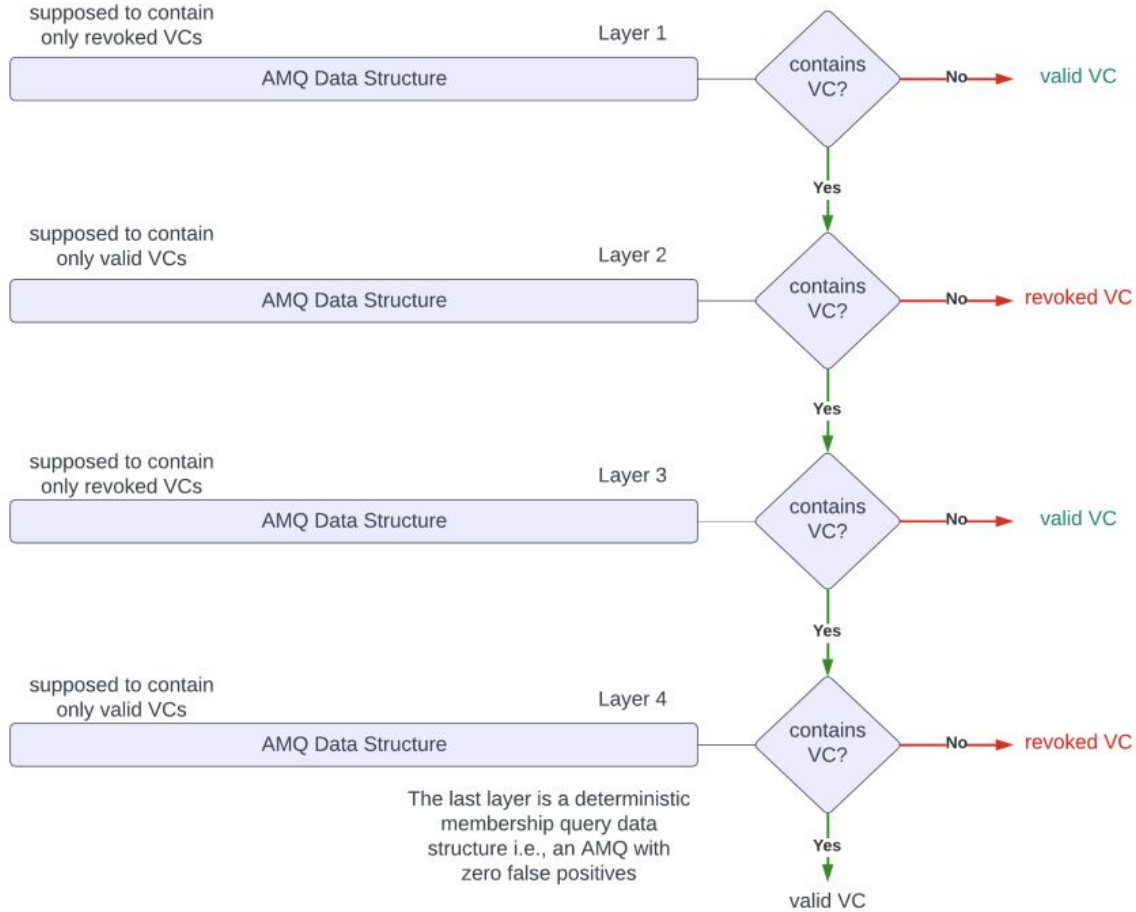
Drawing Inspiration from TLS Certificates: CRLite and the concept of Cascading Filters

isContained algorithm:

- The only **conclusive** answer is **“No”**
(recall - No means No | Yes means maybe Yes)
- No at an **odd level** => **not revoked**
- No at an **even level** => **revoked**
- The **last level** has **no FPs** (recall: termination criteria)
=> No means No | Yes means ~~maybe~~ Yes

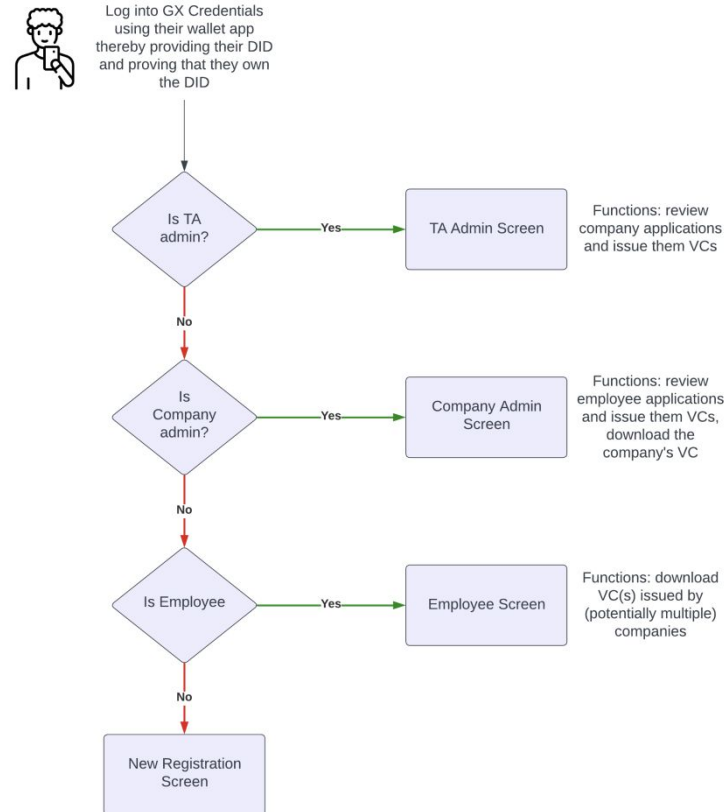






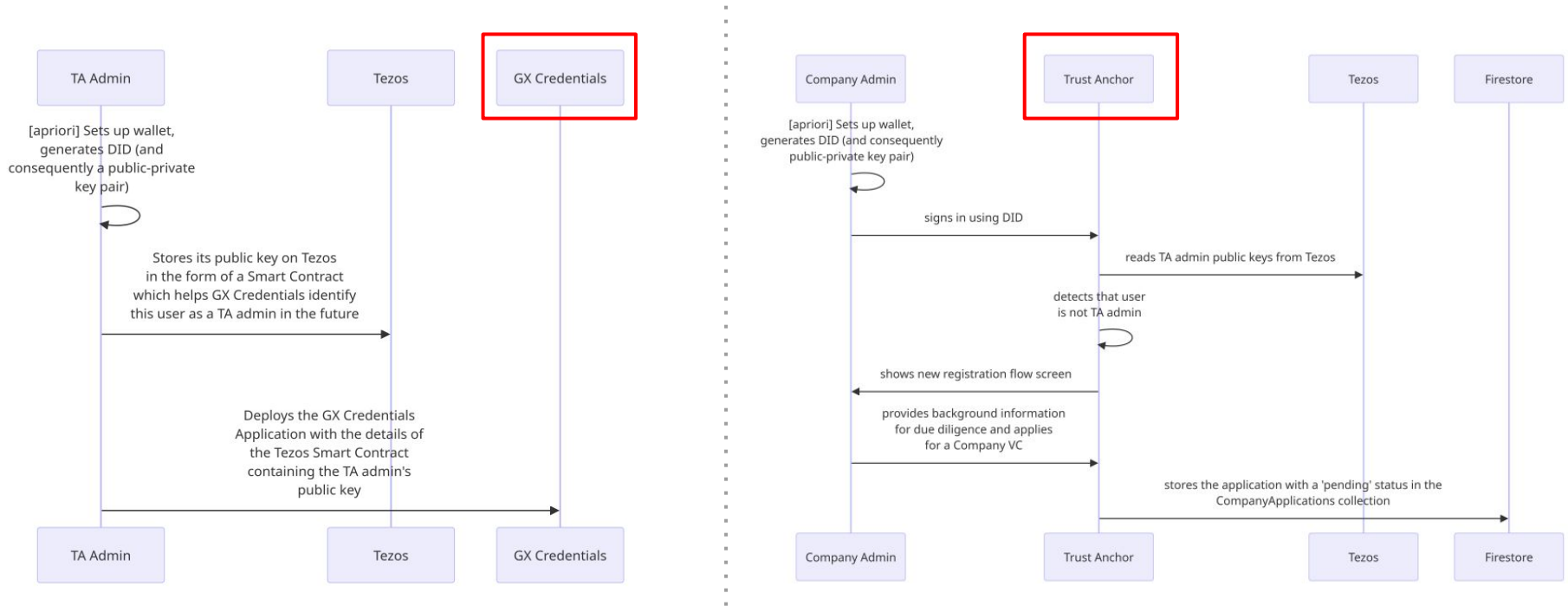
GX Credentials: Architectural Recap

- Depending on the type of the user they see a different welcome screen



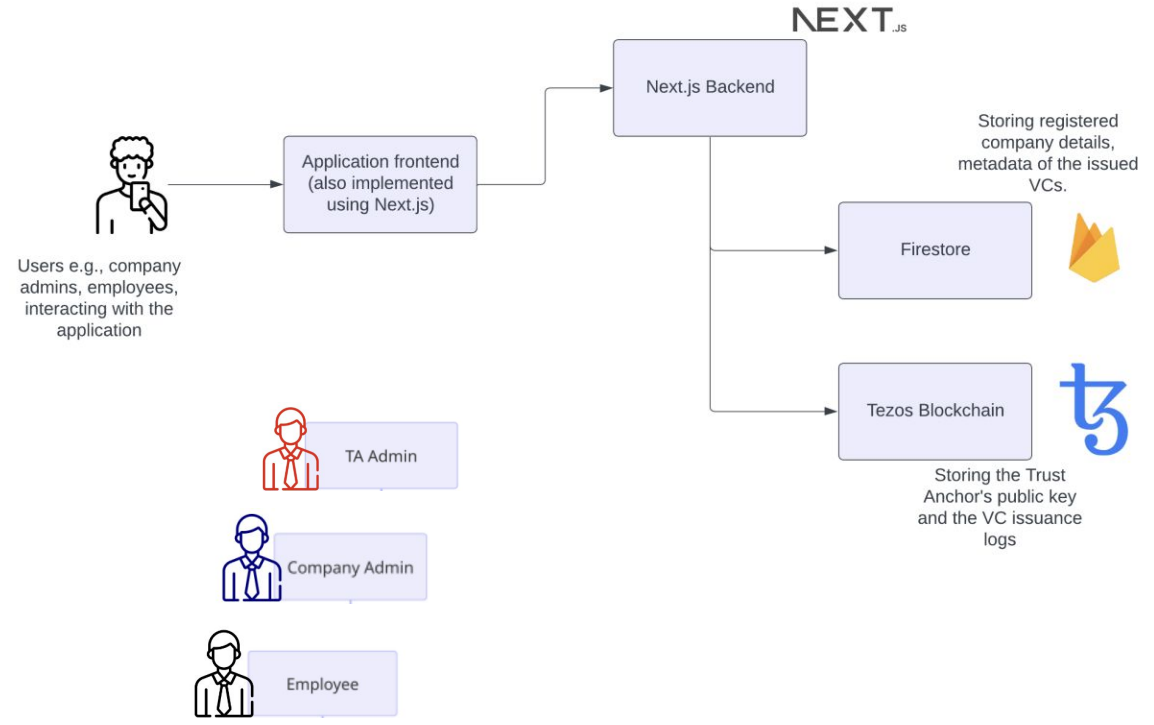
GX Credentials: Architectural Recap

- Different setup/registration experience for TA admins and Company Admins

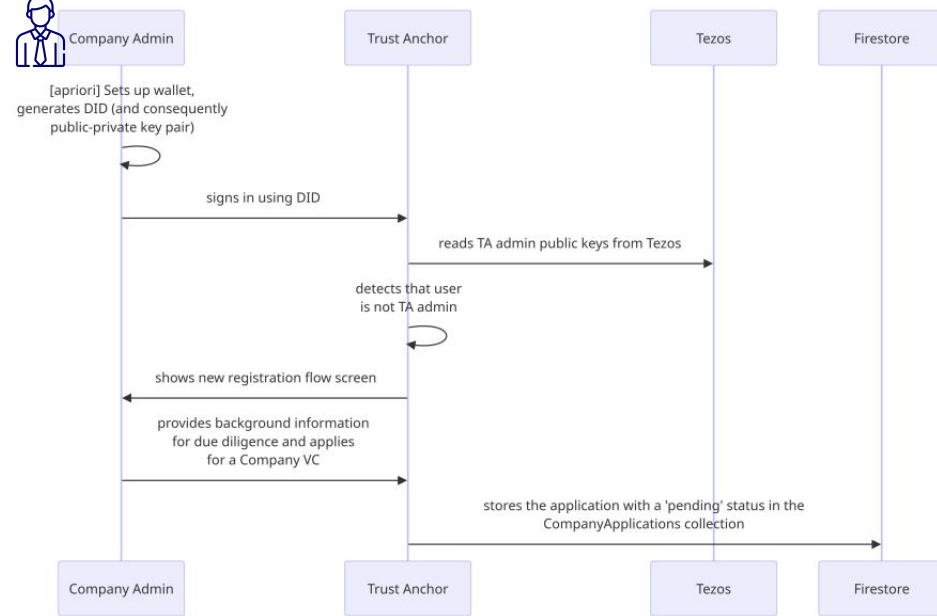
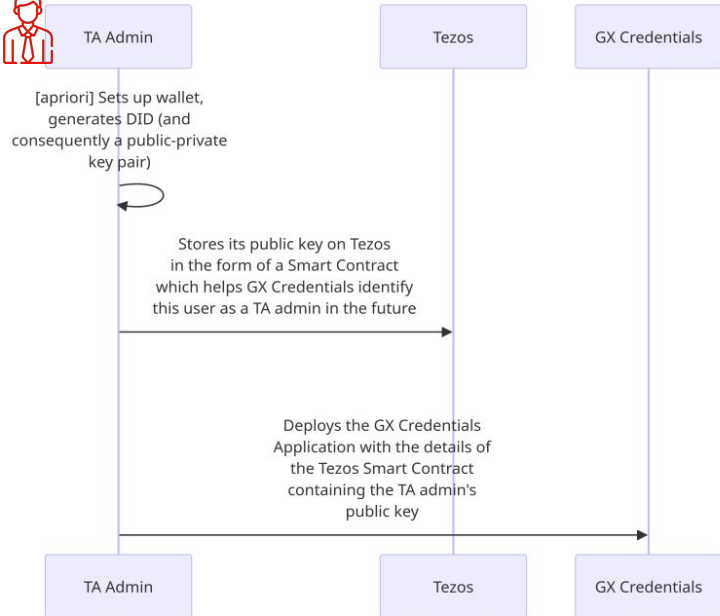


Tech Stack and Actors

- Next.js based frontend and backend
- Collections stored in Firestore
- Public Key and Issuance Logs stored on the Tezos Blockchain
- Three actors interacting with the application

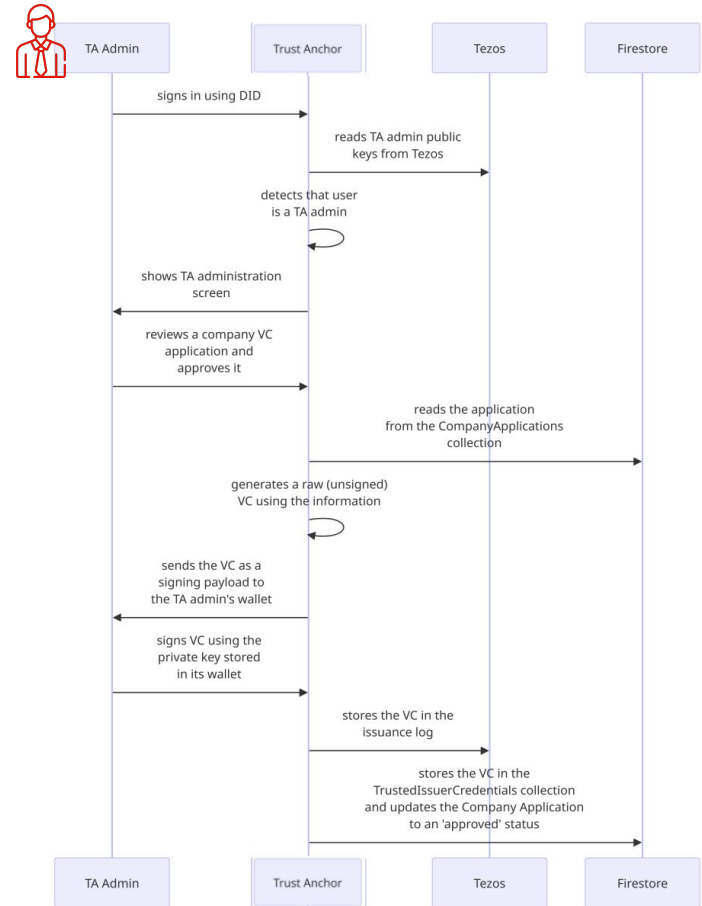


Different setup/registration experience for TA admins and Company Admins



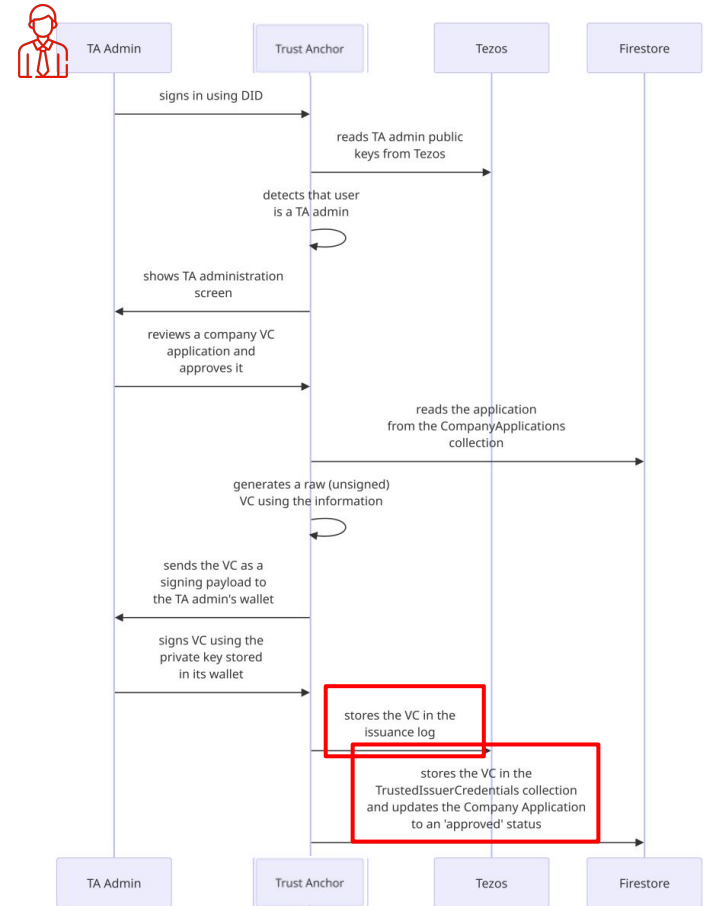
The Company VC Request Approval Flow

- involves the Trust Anchor reading from and storing to the Firestore Db, Issuance Log



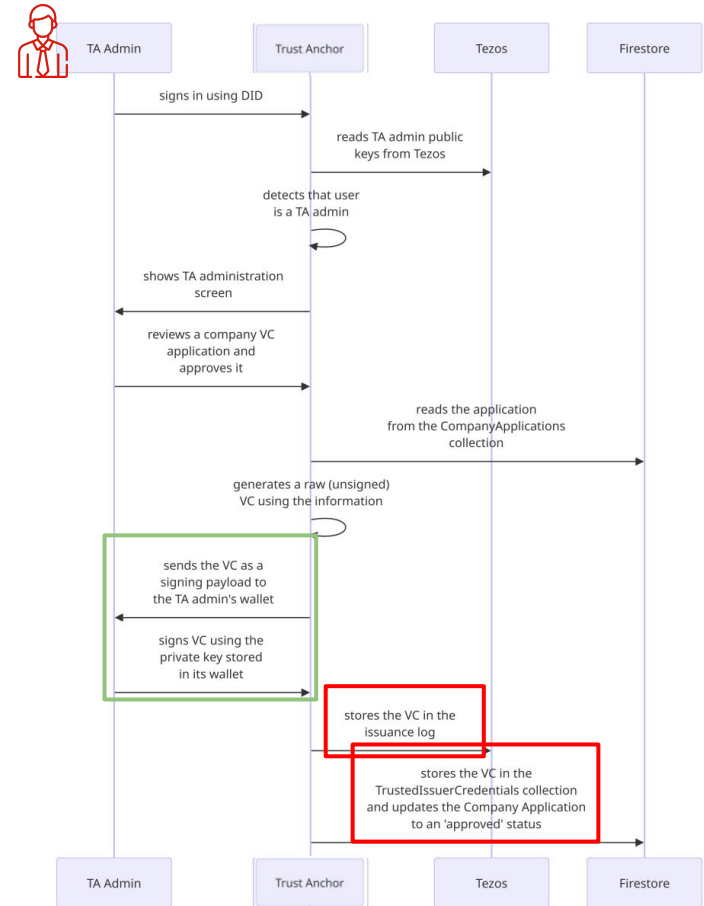
The Company VC Request Approval Flow

- involves the Trust Anchor reading from and storing to the Firestore Db, Issuance Log



The Company VC Request Approval Flow

- involves the Trust Anchor reading from and storing to the Firestore Db, Issuance Log
- Keys are privately owned by each actor in their respective wallets



Drawbacks of the Current Architecture

- Information Silos must exist (even though we have a 'Trust' Anchor)
 - Issuer must have sovereignty over their VCs
 - TA becomes a single point of failure
 - Storage, Data Replication, Security costs for the TA
 - The issuance log is a fatal flaw (publicly readable)
 - There exists an information silo -> private keys => need to extend this concept to the VCs
- Limited Support for revoking/suspending VCs -
`revokeIssuance` endpoint in the Tezos Smart Contract
- No Verifier Implementation



Design Principles for the Architectural Overhaul

1. Information Silos: Ensuring that systems within GX Credentials have access to information on a **need-to-know basis**.
2. Revocation: Enabling revocation/suspension of Verifiable Credentials.
3. Verification: Checking the authenticity and validity of Verifiable Credentials

Design Principles for the Architectural Overhaul

1. Information Silos: Ensuring that systems within GX Credentials have access to information on a **need-to-know basis**.

2. Revocation: Enabling revocation/suspension of Verifiable Credentials.

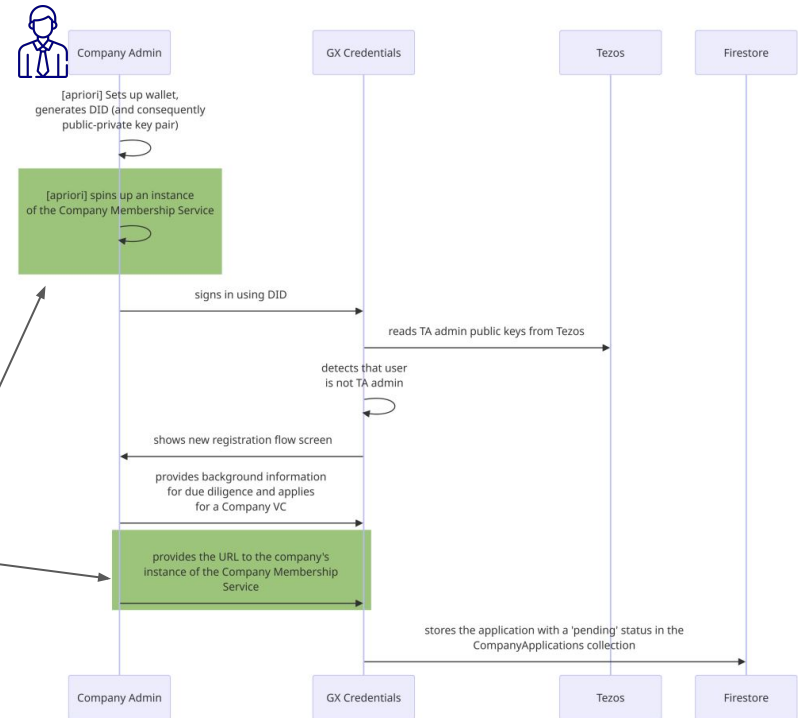
3. Verification: Checking the authenticity and validity of Verifiable Credentials

Information Type	System storing the information GX Credentials	Actors Managing the System
Trust Anchor Public Key	Tezos Blockchain	Trust Anchor Admins
Issued Employee Credentials	Firestore Database	Trust Anchor Admins
Issued Company Credentials	Firestore Database	Trust Anchor Admins
Company Applications	Firestore Database	Trust Anchor Admins
Employee Applications	Firestore Database	Trust Anchor Admins
Issuance Logs (for Companies and Employees)	Tezos Blockchain	Trust Anchor Admins

Information Type	Status in proposed architecture
Trust Anchor Public Key	unchanged
Issued Employee Credentials	not stored by TA
Issued Company Credentials	unchanged
Company Applications	unchanged
Employee Applications	not stored by TA
Issuance Logs (for Companies and Employees)	deprecated

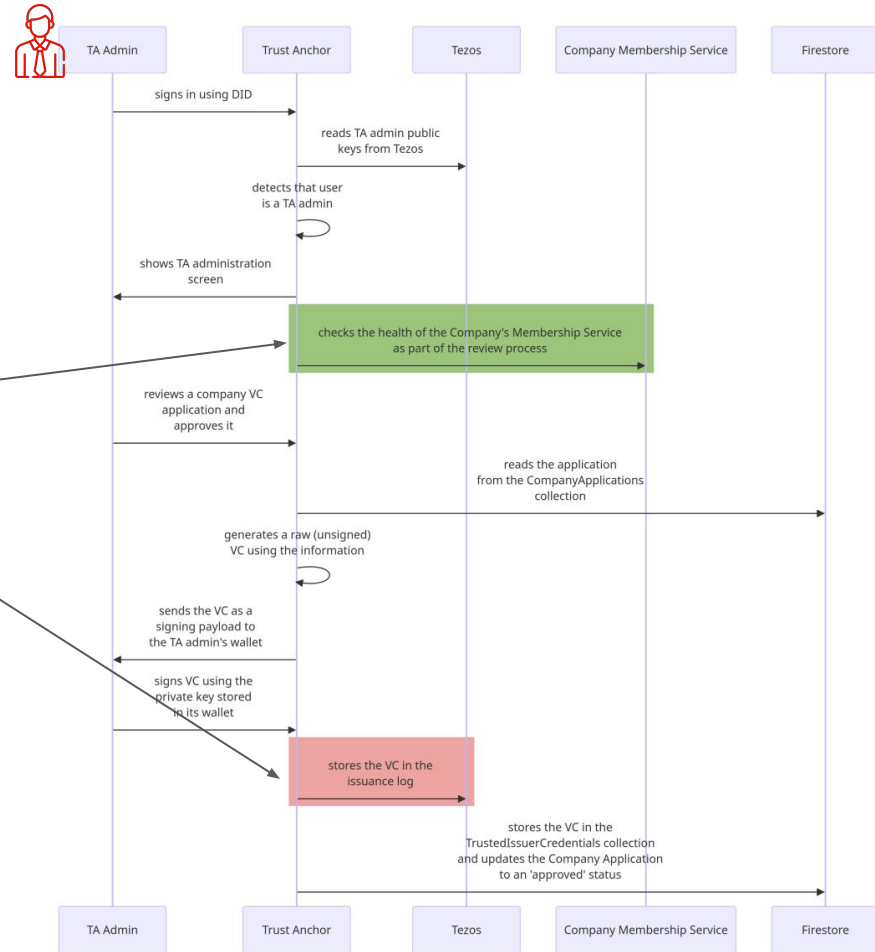
Company Membership Service (CMS)

- GX Credentials should be seen as a collection of open source libraries
- A new library in the GX Credentials code base which will be deployed and run by the Companies themselves
- CMS is a microservice responsible for storing, querying, and managing the lifecycle of the VCs issued by a Company and their metadata
- The Company admins must set up an instance of the CMS before applying for a VC, provide its URL during the VC application process



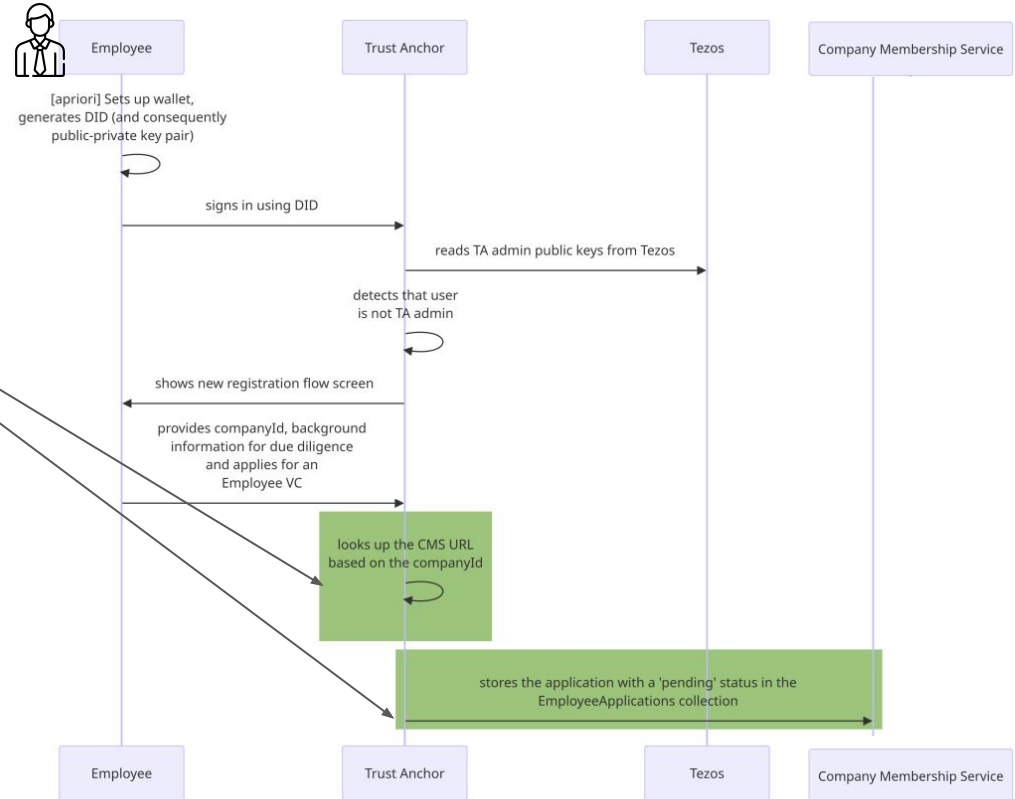
Adding CMS to the Company Registration Flow

- Additional step for the TA Admin to check the health of the CMS
- Issuance Logging is eliminated



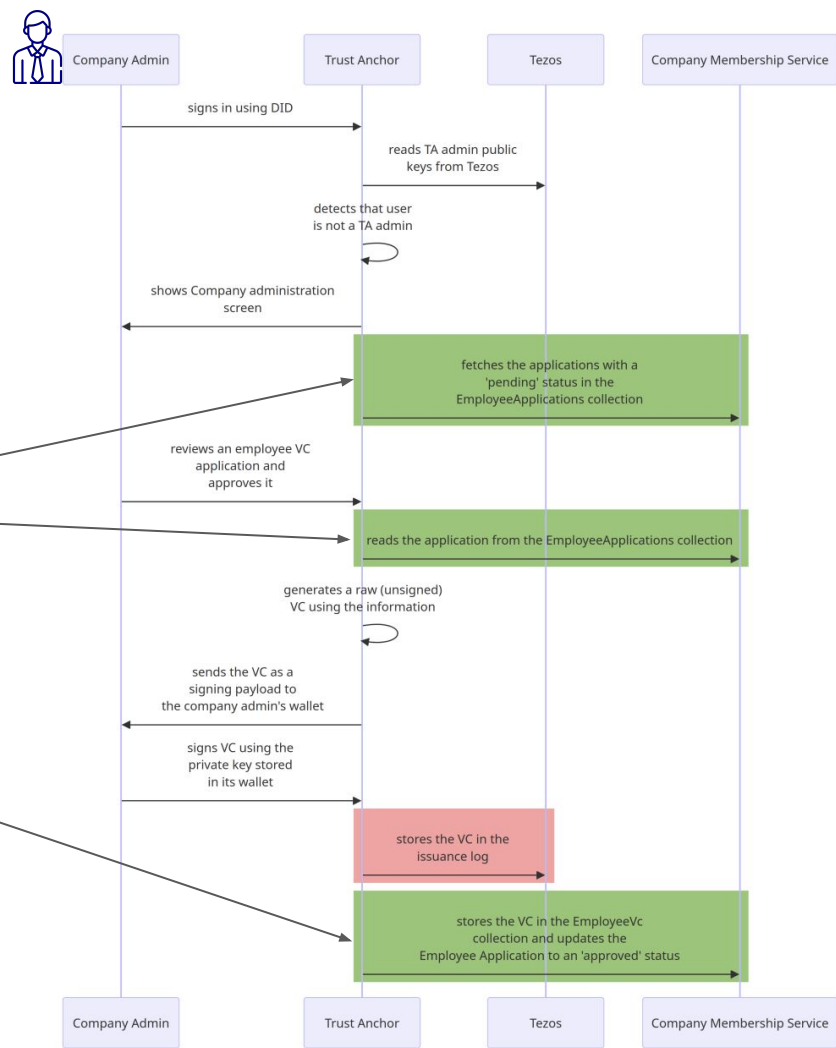
Adding CMS to the Employee Registration Flow

- The Trust Anchor no longer stores the Employee applications, CMS does



Adding CMS to the Employee VC Approval Flow

- The Trust Anchor no longer stores the Employee applications, CMS does
- The Trust Anchor no longer stores the Employee VCs, CMS does



CMS Introduces a set of REST endpoints for interacting with it

- /listPendingApplications,
/createApplication,
/readApplication,
/storeVc - already used
- Using VCs similar to JWTs - Trust established between two inherently untrustworthy systems on the internet

Endpoint	Short Description
/listPendingApplications	returns all the employee applications that have status == 'PENDING'. Called by Trust Anchor to display the Company Administration screen.
/createApplication	adds an Employee's request for a VC as an application and sets its status to 'PENDING'
/readApplication	fetch the employee application for a given applicationId.
/storeVc	store a signed VC issued to an employee.
/revokeVc	revoke an issued VC.
/suspendVc	suspend an issued VC until a specified suspension expiry time.
/undoRevokeVc	un-revoke a previously revoked VC.
/undoSuspendVc	un-suspend a previously suspended VC.
/getVc	fetch the VC for a given VcId.

CMS Introduces a set of REST endpoints for interacting with it

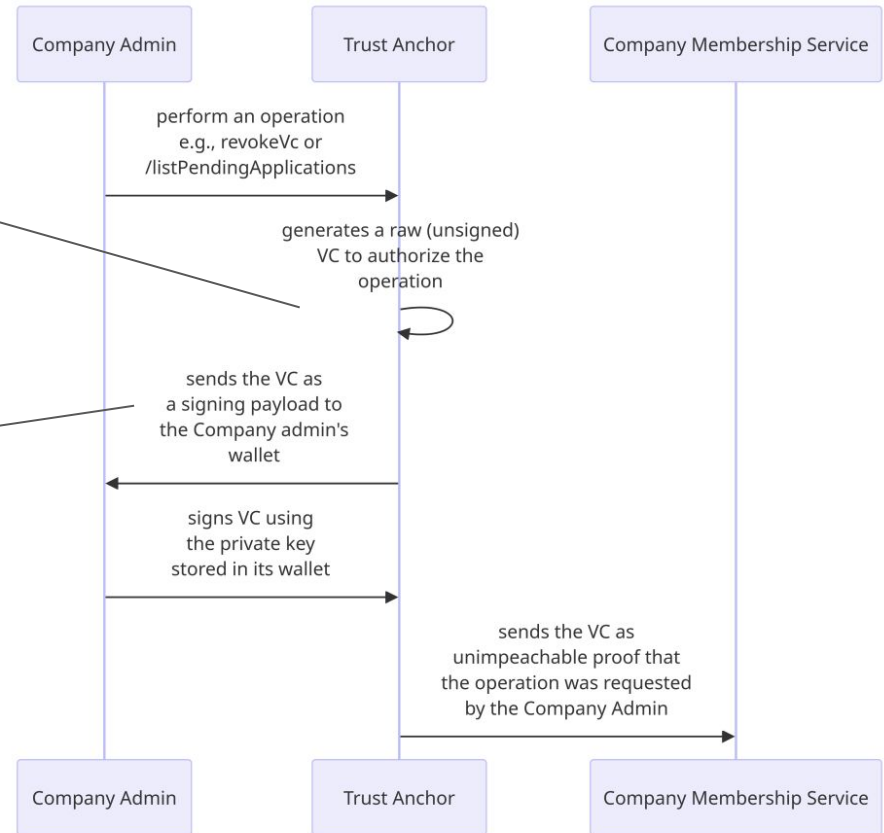
- /listPendingApplications,
/createApplication,
/readApplication,
/storeVc - already used
- Using VCs similar to JWTs - Trust established between two inherently untrustworthy systems on the internet

Endpoint	Short Description
/listPendingApplications	returns all the employee applications that have status == 'PENDING'. Called by Trust Anchor to display the Company Administration screen.
/createApplication	adds an Employee's request for a VC as an application and sets its status to 'PENDING'
/readApplication	fetch the employee application for a given applicationId.
/storeVc	store a signed VC issued to an employee.
/revokeVc	revoke an issued VC.
/suspendVc	suspend an issued VC until a specified suspension expiry time.
/undoRevokeVc	un-revoke a previously revoked VC.
/undoSuspendVc	un-suspend a previously suspended VC.
/getVc	fetch the VC for a given VcId.

The Company Admin needs a VC for all interactions with the CMS

```

{
  "credentialSubject": {
    "ox:issuerCompanyName": "Sraait Inc.",
    "gx:operation": "REVOCATION",
    "gx:employeeVcId": "b1226336-8e7d-11ee-b9d1-0242ac120002",
    "id": "did:pkh:tz:tz1MRoRc5DgRmXHiyfxtGm3EB8mqeLpBfrUA",
  },
  "type": [
    "VerifiableCredential",
    "Operation Credential"
  ],
  "issuanceDate": "2023-10-07T16:41:56.044Z",
  "id": "urn:uuid:20e34a3d-fec5-46d7-949f-7ebf3947642d",
  "proof": {
    "proofValue": "edsigtajEzCi1DApxSue15AC5uw...sUxkyn9mtjyo648",
    "created": "2023-10-07T16:41:56.082Z",
  }
}
  
```

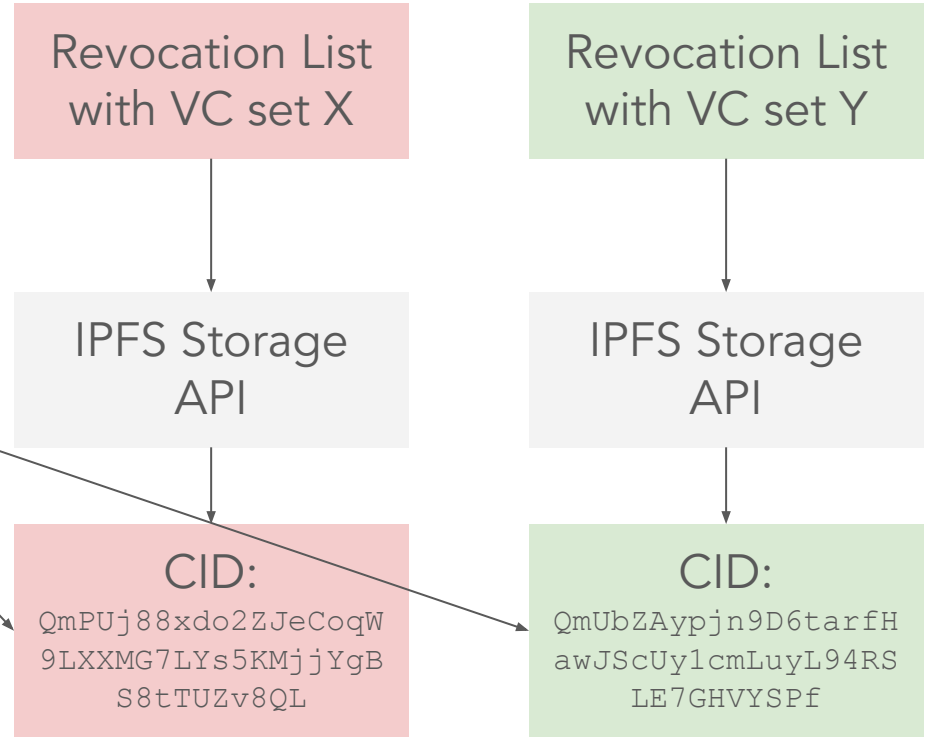


The Architecture uses the Interplanetary File System for Storage

- Design Features:
 - distributed and therefore, has no single point of failure
 - high throughput
 - content-addressed and self-certifying namespace
- IPNS - Interplanetary Name Space addresses the challenge of a constantly mutating address (CID)

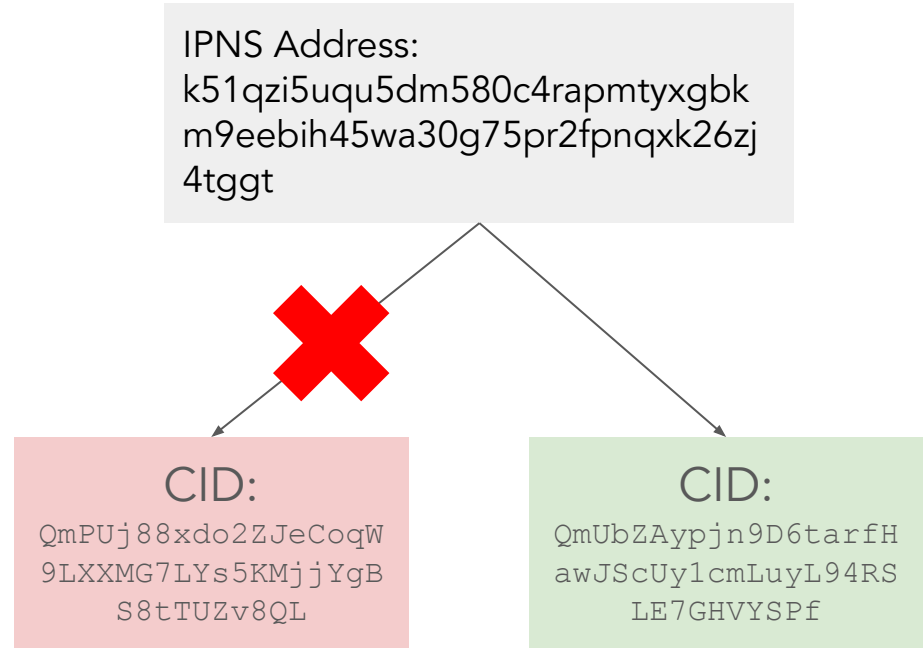
The Architecture uses the Interplanetary File System for Storage

- Design Features:
 - distributed and therefore, has no single point of failure
 - high throughput
 - content-addressed and self-certifying namespace
- IPNS - Interplanetary Name Space addresses the challenge of a constantly mutating address (CID)



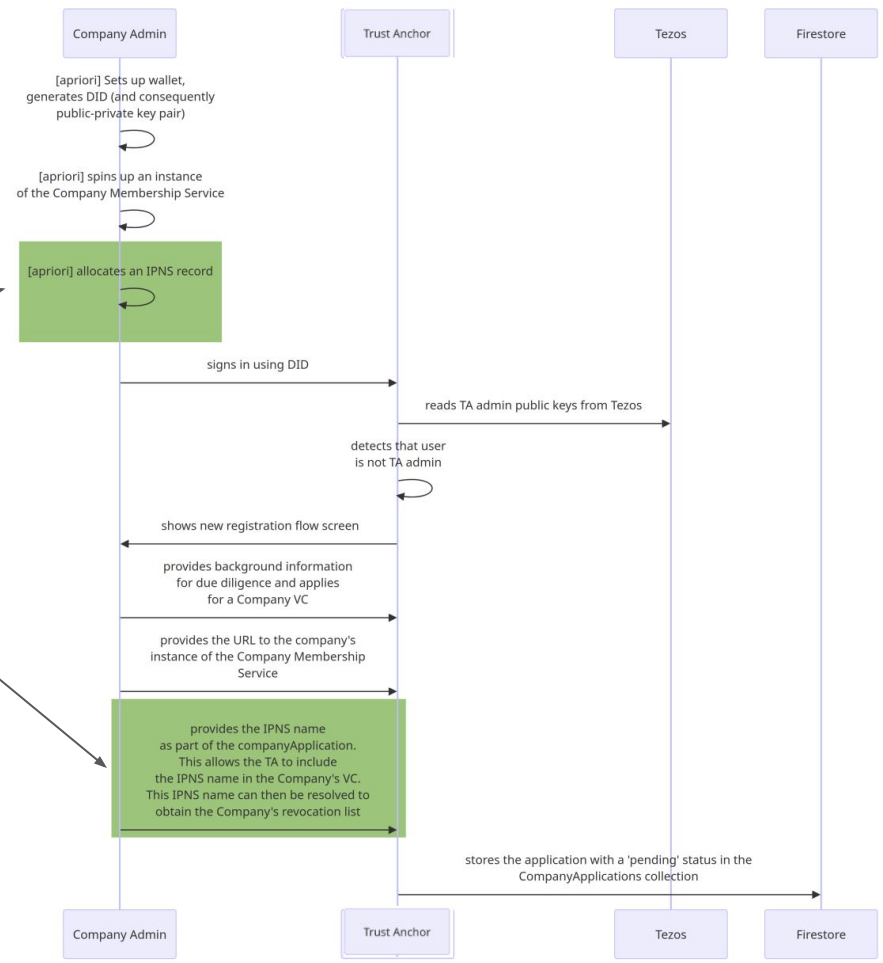
The Architecture uses the Interplanetary File System for Storage

- Design Features:
 - distributed and therefore, has no single point of failure
 - high throughput
 - content-addressed and self-certifying namespace
- IPNS - Interplanetary Name Space addresses the challenge of a constantly mutating address (CID)



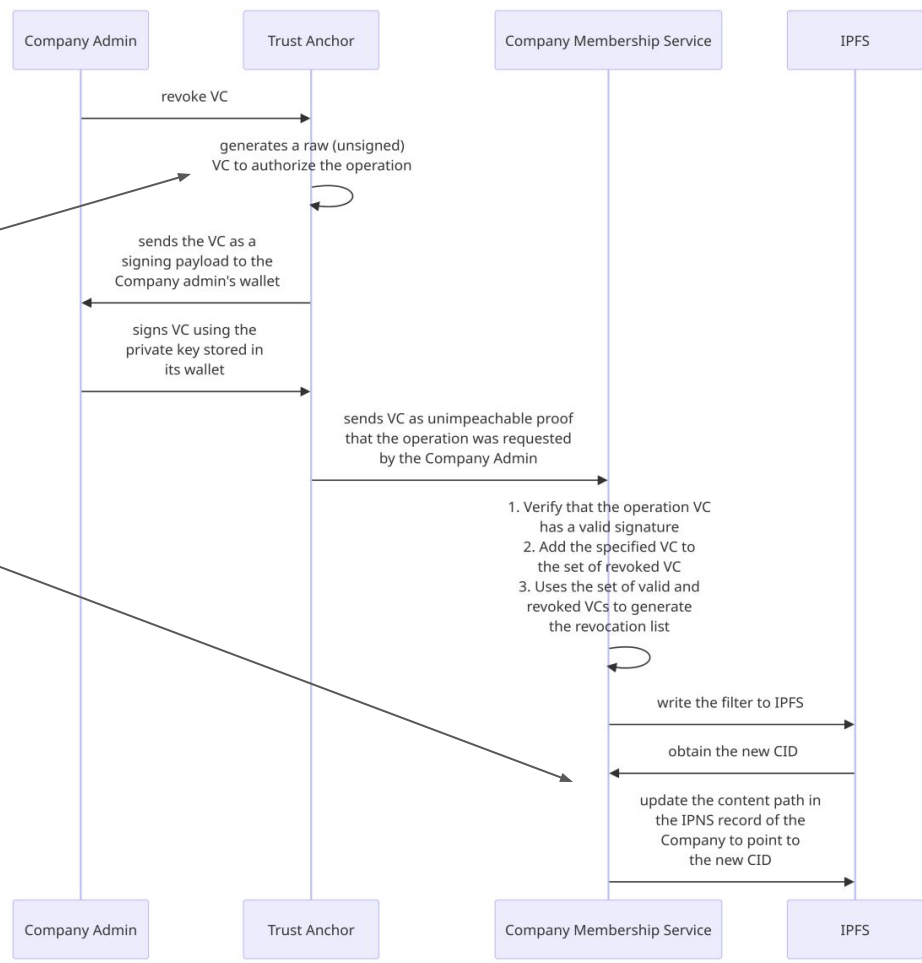
Changes to the Company Registration Flow

- Similar to wallet setup, CMS instantiation, Company Admin also allocates space on IPNS
- IPNS name is provided to the Trust Anchor which bakes it into the VCs



Handling a VC Revocation Request

- Special revocation VC issued using the TA
- CMS also computes the revocation data structure and publishes it in the preordained IPNS record



Minor Extensions: VC Suspension

- Similar to the REVOCATION Operation Credential
- Additional `gx:suspensionDuration` field
- Cron job to check the Database for VCs that have served their suspension periods

```
{
  "credentialSubject": {
    "gx:issuerCompanyName": "Srajit Inc.",
    "gx:operation": "SUSPENSION",
    "gx:suspensionDuration": "P1Y2M3DT1H30M10S",
    "gx:employeeVcId": "b1226336-8e7d-11ee-b9d1-0242ac120002",
    "id": "did:pkh:tz:tz1MRoRc5DgRmXHiyfxtGm3EB8mqeLpBfrUA",
  },
  "type": [
    "VerifiableCredential",
    "Operation Credential"
  ],
  "issuanceDate": "2023-10-07T16:41:56.044Z",
  "id": "urn:uuid:20e34a3d-fec5-46d7-949f-7ebf3947642d",
  "proof": {
    "proofValue": "edsigtajEzCi1DApxSue15AC5uw...sUxkyn9mtjyo648",
    ...
  },
  "created": "2023-10-07T16:41:56.082Z",
}
```

Minor Extensions: Membership Checking Service

- Standalone service that can be used by Verifiers
- Includes support for concepts introduced in this thesis - IPFS, Cascading Filters, Tezos

1. Check the validity of the Employee VC's signature using the Company's public key.
2. Check the validity of the Company VC's signature using the TA's public key.
3. Find the IPNS address for the Company's revocation list in the Company's VC.
4. Read the Company's revocation list from IPFS and check that Employee VC is not included in it.
5. Read the IPNS address for the TA's revocation list from Tezos.
6. Read the TA's revocation list from IPFS and check that Employee VC is not included in it.

A vertical white bar on the left side of the slide.

Q & A



Prof. Dr.

Florian Matthes

Technical University of Munich (TUM)
TUM School of CIT
Department of Computer Science (CS)
Chair of Software Engineering for
Business Information Systems (sebis)

Boltzmannstraße 3
85748 Garching bei München

+49.89.289.17132
matthes@in.tum.de
www.matthes.in.tum.de



Backup slides after this point

Future Roadmap

Research Question	Timeline
Identifying requirements and design goals for revocation/suspension mechanisms for VCs Identifying key metric dimensions to be considered for these requirements/design goals Identifying use cases and categorising them on these key metric dimensions	W1 - W3
Researching the state of the art for data structures that support exclusion operations Identifying parallels between revocation schemes used by TLS and the VC ecosystem Identifying the differences in requirements between TLS and the VC ecosystem	W4 - W8
Delineating possible options for an alternative Revocation/Suspension mechanism Categorizing them based on the use cases defined above Benchmarking/evaluating candidate data structures	W9 - W14
Building/Designing the E2E revocation infrastructure Experiments with off-chain data storage systems e.g. IPFS Benchmarking with other systems e.g. Status List 2021	W15 - W20
Collating the research into document form, preparing for the thesis defence	W21 - W23

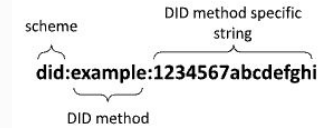
Zooming out to the Research Roadmap

- Identifying use cases and benchmarking them on the basis of their needs to optimise scale v/s minimum propagation delay
- Building/Designing the E2E revocation infrastructure
 - Proposed changes to the VC standard
 - Publishing the revocation list for different use cases to a DLT system
 - Experiments with off-chain data storage systems e.g. IPFS

Decentralized Identifiers (DIDs) - route to the Issuer's Metadata



```
{
  "@context": [
    "https://www.w3.org/2018/credentials/v1",
    "https://w3id.org/vc/status-list/2021/v1"
  ],
  "id": "https://example.com/credentials/23894672394",
  "type": ["VerifiableCredential"],
  "issuer": "did:example:12345",
  "issued": "2021-04-05T14:27:42Z",
  "credentialStatus": {
    "id": "https://example.com/credentials/status/3#94567"
    "type": "StatusList2021Entry",
    "statusPurpose": "revocation",
    "statusListIndex": "94567",
    "statusListCredential": "https://example.com/credentials/status/3"
  },
  "credentialSubject": {
    "id": "did:example:6789",
    "type": "Person"
  },
  "proof": { ... }
}
```



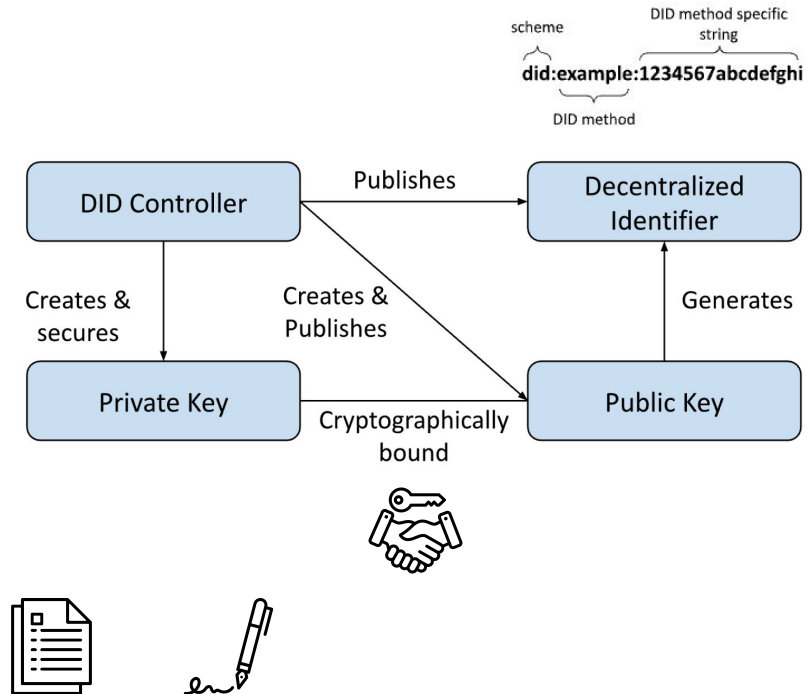
did:**btcr**:xz35-jzv2-qqs2-9wjt

did:**ethr**:0xE6Fe788d8ca214A080b0f6ac7F48480b2AEfa9a6

DIDs, VCs, and Asymmetric Encryption - a seamless certification system

DID are:

- Resolvable to DID document -> used to obtain metadata about the issuing party.
- Cryptographically verifiable - asymmetric cryptography is used to ensure this
- Decentralized - a central authority is not required to allocate DIDs
 - No single point of failure
 - No unilateral revocation by an authoritarian central server



AMQFilter Interface

1. void insert(E element)
2. boolean maybeContains(E element)
3. [optional] void delete(E element)

Algorithm 1: Insert (x)

```
 $f = \text{fingerprint}(x);$ 
 $i_1 = \text{hash}(x);$ 
 $i_2 = i_1 \oplus \text{hash}(f);$ 
if bucket[ $i_1$ ] or bucket[ $i_2$ ] has an empty entry then
  add  $f$  to that bucket;
  return Done;

// must relocate existing items;
 $i =$  randomly pick  $i_1$  or  $i_2$ ;
for  $n = 0; n < \text{MaxNumKicks}; n++$  do
  randomly select an entry  $e$  from bucket[ $i$ ];
  swap  $f$  and the fingerprint stored in entry  $e$ ;
   $i = i \oplus \text{hash}(f);$ 
  if bucket[ $i$ ] has an empty entry then
    add  $f$  to bucket[ $i$ ];
    return Done;

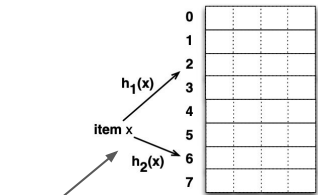
// Hashtable is considered full;
return Failure;
```

Any Hash function

“partial-key cuckoo hashing to derive an item’s alternate location based on its fingerprint”

$i_1 = \text{hash}(x)$
 $i_2 = i_1 \wedge \text{hash}(f)$
 $i_1 = i_2 \wedge \text{hash}(f)$
 $= i_1 \wedge \text{hash}(f) \wedge \text{hash}(f)$
 $= i_1 \wedge 0$
 $= i_1$

8 buckets with 4 entries per bucket



(c) A cuckoo filter, two hash per item and functions and four entries per bucket

AMQFilter Interface

1. void insert(E element)
2. boolean maybeContains(E element);
3. [optional] void delete(E element)

Algorithm 2: Lookup (x)

```
f = fingerprint(x);  
i1 = hash(x);  
i2 = i1 ⊕ hash(f);  
if bucket[i1] or bucket[i2] has f then  
    return True;  
return False;
```

Trivial

Algorithm 3: Delete (x)

```
f = fingerprint(x);  
i1 = hash(x);  
i2 = i1 ⊕ hash(f);  
if bucket[i1] or bucket[i2] has f then  
    remove a copy of f from this bucket;  
    return True;  
return False;
```

Trivial

Algorithm 1: Insert (x)

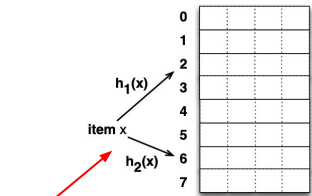
```
f = fingerprint(x);  
i1 = hash(x);  
i2 = i1 ⊕ hash(f);  
if bucket[i1] or bucket[i2] has an empty entry then  
    add f to that bucket;  
    return Done;  
// must relocate existing items;  
i = randomly pick i1 or i2;  
for n = 0; n < MaxNumKicks; n++ do  
    randomly select an entry e from bucket[i];  
    swap f and the fingerprint stored in entry e;  
    i = i ⊕ hash(f);  
if bucket[i] has an empty entry then  
    add f to bucket[i];  
    return Done;  
// Hashtable is considered full;  
return Failure;
```

Any Hash function

“partial-key cuckoo hashing to derive an item’s alternate location based on its fingerprint”

```
i1 = hash(x)  
i2 = i1 ^ hash(f)  
i1 = i2 ^ hash(f)  
    = i1 ^ hash(f) ^ hash(f)  
    = i1 ^ 0  
    = i1
```

8 buckets with 4 entries per bucket

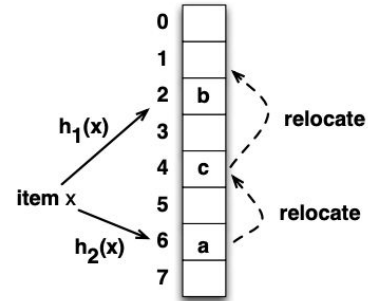


(c) A cuckoo filter, two hash per item and functions and four entries per bucket

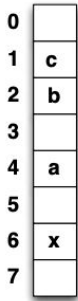
Cuckoo Filters

Quick Overview

- AMQ data structures like Bloom filters don't need to store 'anything' related to the original element - only bits
- DSM data structures like Sets, Maps need to store the original element because there is a possibility of rehashing -> when we have more elements than we allocated space for, hash collisions, etc.
- Cuckoo Filters take ideas from both
 - eviction and relocation of elements needs to store 'something'
 - Fingerprints - unique proxy for the element, upper bound on the size of what is being stored
 - [+ve side effect] privacy - we don't want raw VCs to be stored in the AMQ data structure



(a) before inserting item x



(b) after item x inserted

Research Questions

1. What requirements exist for revocation/suspension mechanisms for VCs?
 - a. What general requirements can we identify?
 - b. What are key metric dimensions we need to consider?
 - c. Can we look at some example use cases and place them on the key metric dimensions to validate?
2. What is the state of the art for data structures that support exclusion operations?
 - a. What work exists towards VC revocation/suspension specifically?
 - b. What can we learn from other (experimental) revocation schemes for TLS certificates?
Some examples include Bloom Filters, Mozilla CRLite.
 - c. Compared to TLS, what significant differences in requirements/environment can we identify for a VC ecosystem?
3. Choosing or designing the Revocation/Suspension data structure
 - a. Delineating possible options
 - b. Categorizing options based on use case requirements (see 1c)
 - c. Benchmarking/evaluating candidate data structures
4. Building/Designing the E2E revocation infrastructure i.e., in tandem with a DLT
 - a. How can we minimize cost?
 - b. Do we need to keep data off-chain and if so, what data and where?
 - c. What key metrics of the system can we measure and benchmark with other systems?

Core Concepts: Verifiable Credentials (VCs)

- Foolproof way for real-world entities to stake claims - claim + signature
- Issuing Party
- Holder
- Verifying Party

VERIFIABLE CREDENTIALS DATA MODEL V1.1 PUBLICATION HISTORY

See also the history of the other specifications of the series: [vc-data-model-2.0](#)

2022-03-03	Verifiable Credentials Data Model v1.1 Recommendation
2021-11-09	Verifiable Credentials Data Model v1.1 Recommendation
2019-11-19	Verifiable Credentials Data Model 1.0 Recommendation
2019-09-05	Proposed Recommendation
2019-07-25	Candidate Recommendation Snapshot
2019-03-28	Candidate Recommendation Snapshot
2019-02-08	Working Draft
2017-08-03	First Public Working Draft

W3C Standard that has seen mature development since 2019;
vc-data-model-2.0 is in the works

Core Concepts: Decentralised Identifiers (DIDs)

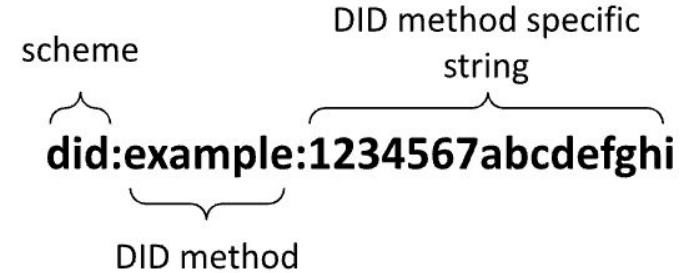
- Uniform Resource Locator (URI) leading to the DID document
- Multiple specifications

`did:btcr:xz35-jzv2-qqs2-9wjt`

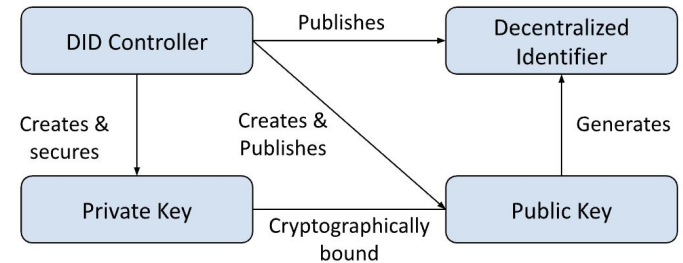
`did:ethr:0xE6Fe788d8ca214A080b0f6ac7F48480b2AEfa9a6`

- Design Properties

- Resolvable
- Cryptographically Verifiable
- Decentralised (self-certifying identifiers)
- Permanent (even with Key Rotation)



Syntax of Decentralized Identifiers (DIDs)



The DID Trust Triangle