A Decentralized IIoT Identity Framework based on Self-Sovereign Identity using Blockchain

Akanksha Dixit*, Max Smith-Creasey† and Muttukrishnan Rajarajan*
*School of Mathematics and Computer Science Engineering, City, University of London, UK
Email: {akanksha.dixit, r.muttukrishnan}@city.ac.uk
†Applied Research, British Telecommunications plc, Adastral Park, Ipswich, IP5 3RE, UK
Email: max.smith-creasey@bt.com

***Abstract*—The fundamental requirement for interaction between digital entities is a secure and privacy-preserving digital identity infrastructure. Traditional approaches rely heavily on centralized architectural components such as Certificate Authorities (CAs) and credential storage databases that have drawbacks like a single point of failure, attack prone honeypot databases and poor scalability. Self-Sovereign Identity (SSI) is a novel decentralized digital identity model that uses Decentralized Identifiers (DIDs) and Verifiable Credentials (VCs). In this work, we propose a novel decentralized identity framework for Industrial Internet-of-Things (IIoT) based on SSI model. The proposed framework is implemented on two blockchain platforms namely Ethereum and Hyperledger Indy to study the underlying overheads.**

***Index Terms*—Industry 4.0, Internet of Things, Self-Sovereign Identity, Blockchain.**

I. Introduction

With increase in automation, machine-to-machine (M2M) communication has become pervasive in various sectors. However, in complex Industrial Internet-of-Things (IIoT) settings, interactions are not limited to intra-device or intra-user communication but extend in heterogeneous forms. Generally, IoT devices are assigned with default weak passwords which lead to large-scale attacks like Mirai botnet [5]. Since devices are largely unattended in IoT scenarios, password based manual approaches are unsuitable for inter-device communication. In fact, around 80% of IoT devices fail to implement passwords securely [4]. Hence, automated solution like Public-Key Certificates (PKC), which binds identity of an entity to their public key became popular among IoT vendors. However, standards like PGP and X.509 were not designed with privacy in mind, as it is evident with the use of subject names in their certificates.

A recent development in identity management models is "Self-Sovereign Identity" (SSI). The primary idea of SSI is that subjects can fully create, build and control their identity and associated credentials rather than relying on a third party to manage them. In this paper, we describe a decentralized digital identity framework using the *principles of SSI*. The proposed model facilitates faster on-boarding of a large fleet of devices in a secure, scalable, privacy-preserving and easily verifiable manner. The key contribution of this work are:

- We propose a novel proof-of-concept solution to IoT devices bootstrapping that supports digital identity creation as DIDs, issuance and verification of VCs all in a decentralized manner by leveraging a innovative combination of blockchain and a decentralized data storage layer.
- The proposed model presents an efficient way of applying DIDs and VCs to IoT by demonstrating the usability of such a system as a use-case i.e. to establish secure remote connections for M2M communication.
- We implement a proof-of-concept to test the feasibility of such a solution. We perform the testing on two different blockchain platforms for a realistic comparison. These platforms are Ethereum and Hyperledger Indy (henceforth Indy). We leverage Inter Planatery File System (IPFS) as the decentralized off-chain data storage layer.

II. Current Practices and State-of-the-art Identity Management Models

Over the years identity management models have evolved. The first model comprised of centralized authorities acting as issuers and authenticators of digital identity. Organizations like ICANN determined validity of domain names and later certificate authorities (CAs) created and stored identities for entities. This led to excessive authority vested in a few corporations with little control left to the users. The second model known as federated identity allowed multiple service providers forming a federation with one of the identity provider, allowing the user to use same credential across these platforms. This led to great control and usage data tracking by technology giants [1], [7]. The third model known as user-centric identity focused on two major elements: user consent and interoperability. Standards like OpenID (2005), OAuth (2010), FIDO (2013), OpenID Connect (2015) and CTAP 2.0 (2018) were introduced for creating user/device authentication. The most recent model called SSI gives complete autonomy of its identity to the subject itself.

SSI is being widely studies by scholars and research agencies. Soltani et al. [10] proposes a client on-boarding and Know Your Customer (KYC) process using SSI and distributed ledger technology. They make use of Hyperledger Indy to propose a KYC framework and evaluate their framework against SSI and GDPR principles. Othman et al. [9] presents a *Horcrux* protocol that combines DID with Biometric Open Protocol Standard (BOPS) that enables the subjects to control the process of accessing their identities by giving the consent via a biometric authentication. Several use-cases for SSI in IoT have been proposed by researchers for instance facilitating

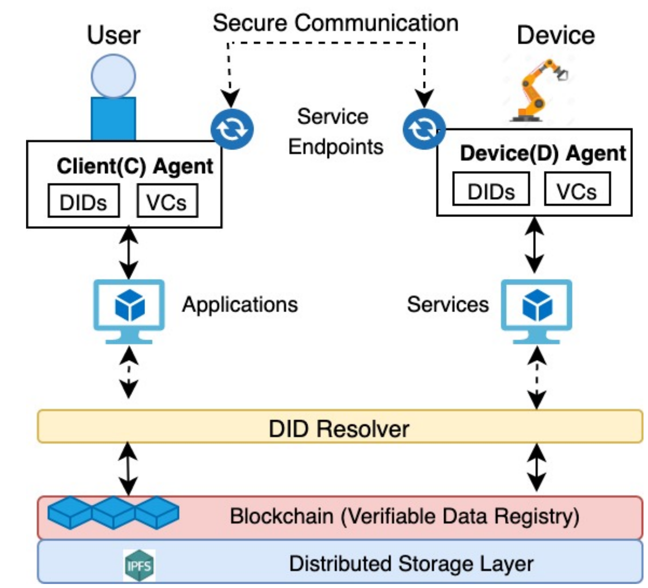


Figure 1: Reference Architecture for M2M communication carried out by IoT device agents

interaction between users/devices, automated authentication and authorization between users and devices, part life-cycle management etc.[2]. Lagutin et al. [8] proposes use of DIDs in an IoT scenario through a use-case of a trusted printing service in a university, that needs to be accessed by authorised users without leaking any confidential information. In [6], the authors present a comparative study for using SSI for IoT environments against standards such as PGP (Pretty-Good Privacy) Keys and X.509 certificates. In [3], the authors propose an SSI based access control system for educational document verification.

```
{
  "@context": "https://www.w3.org/ns/did/v1",
  "id": "did:example:123456789abcd",
  "authentication": [{      //to authenticate
    "id": "did:example:123456789abcd#keys-1",
    "type": "Ed25519VerificationKey2018",
    "controller": "did:example:123456789abcd",
    "publicKeyBase58": "
    H3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqPV"
  }],
  "service": [{        //to retrieve VCs
    "id":"did:example:123456789abcdefghi#vcs",
    "type": "VerifiableCredentialService",
    "serviceEndpoint": "https://example.com/vc/"
  }]
}
```

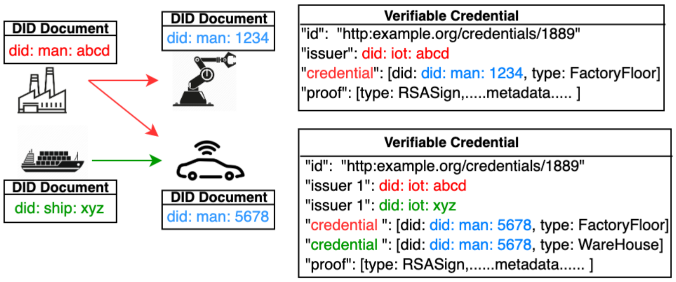
Listing 1: DID Document Syntax

III. Proposed Model

A. System Actors and Components

Fig 1 illustrates proposed architecture. The primary actors and components in the system are as follows:

Decentralized Identifier (DIDs) are new type of digital identifiers proposed by W3C that uniquely identifies a subject (person, thing, abstract entity etc.). The DID has the following syntax: *"did:methodName:method-specific-id"*. DID Methods are a set of specifications by which a DID and its associated DID Documents are created, resolved, updated and deactivated using verifiable data registry (VDR). A unique identity for that network is defined by *method-specific-id*. DIDs are resolved to DID Documents as shown in Listing 1. DID Doc contain set of public keys, authentication methods and service-endpoints.



```
Verifiable Credential
"id":  "http:example.org/credentials/1889"
"issuer": did: iot: abcd
"credential": [did: did: man: 1234, type: FactoryFloor]
"proof": [type: RSASign,.....metadata..... ]
```

```
Verifiable Credential
"id":  "http:example.org/credentials/1889"
"issuer 1": did: iot: abcd
"issuer 1": did: iot: xyz
"credential ": [did: did: man: 5678, type: FactoryFloor]
"credential ": [did: did: man: 5678, type: WareHouse]
"proof": [type: RSASign,......metadata...... ]
```

Figure 2: VCs issued to two types of IoT devices (a stationary robotic arm and a mobile autonomous bot) by two separate issuers.

Verifiable Credentials (VC) VCs are portable, tamper-proof and cryptographically signed digital claims made about a subject. A credential is made of a set of individually verifiable claims. The format of a VC is shown in Fig 2. By leveraging VC, a subject can present the claim to a verifier when requested.

Subject/Holder (S) can be a machine, a user or an enterprise that holds a unique identity in the network. They want to connect in order to exchange data, gain authorization to access restricted services, remotely log-in to applications/devices.

Issuers (I) are actor that issues VCs to the *subjects*. Issuer can be a government agency, a manufacturing unit, etc. and each issue statement is logged in verifiable data registry (VDR).

Verifier (V) are actors that form the interface of a resource and verify the VCs presented by S to grant them access. It can be a cloud or edge application.

Agent An *agent* is a software process that acts on behalf of the *subjects*. It offers persistent internet addressable service endpoints It processes *subject's* request to the VDR.

Verifiable Data Registry (VDR) stores references to the DIDs and VCs issued by I to S. Depending on VDR implementation, it can include some or all of the following data: DID, credential schema, credential definition and a revocation registry. In our work, we use two different VDRs for a comparative study. One is Ethereum blockchain and other is Hyperledger Indy blockchain platform.

Smart Contracts are automated scripts embedded in blockchain that can execute, control and legally automate events defined in them. In Ethereum we use the *Ethereum DID Registry* contract, Listing 2 that acts as a registry for key and attribute management for identities. The *Verifiable Claims Registry* contract, Listing 3 is a secure place to hold issued VCs. Indy does not host smart contracts. Rather than storing data on the ledger and providing access to that data using smart contracts, Indy enables subjects to own the data in their wallets.

Distributed Storage Layer is a layer of peer-to-peer connected nodes that form a decentralized system for storing, linking and transporting data. In the proposed model this layer using IPFS.

```
contract EthereumDIDRegistry{
  % add owner of the DID holder
  function addOwner(address owner, bytes32 ipfsHash)
  % returns owner of the identity holder
  function identityOwner(address identity)
  % change the owner of DID
  function changeOwner(address identity, address
    newOwner)
  % check signature of identity holder
  function checkSignature(address identity, bytes32
    signature, bytes32 hash)
}
```

Listing 2: Ethereum DID Registry

```
contract VerifiableClaimsRegistry{
  % add claim to subject, key = type of claim, its
    value
  function setClaim(address subject, bytes32 key,
    bytes32 value)
  % retrieve claim for a subject
  function getClaim(address issuer, address subject,
     bytes32 key)
  % remove claims for a subject
  function removeClaim(address issues, address
    subject, bytes32 key)
}
```

Listing 3: Verifiable Claims Registry

B. Framework Explained

At the time of provisioning, $subjects$ creates DID through their $agents$, creating an immutable record of the operation on the blockchain. The respective DID Doc for each DID is created and stored on IPFS. In Ethereum, IoT devices and users (or any *subject*) create their $DIDs$ in the form of Ethereum Externally Owned Accounts (EOA) on blockchain with the help of $agents$. The created $DIDs$ are registered on $EthereumDIDRegistry$ with the hash of DID Doc content returned by IPFS using *addOwner* function. In Indy, $subject$ creates a new DID record in the wallet and then send an NYM transaction to the blockchain to make a record of this identity creation. NYM transactions in Indy are used for creation of new DIDs, setting/rotation of verification key, setting/changing of roles. Once, the $subjects$ in the network are equipped with $DIDs$, VCs can be issued to the devices and applications attesting to it's legitimacy, access privileges and roles. Claims are issued by an $issuer$ signed with their private keys, to a $subjects$ with a key that indicates type of claim. In Ethereum, the claims are registered on $VerifiableClaimsRegistry$ contract using *setClaim* function. Whereas in Indy, the VCs are issued by $TrustAnchors$ or trusted $issuers$ in the form of NYM transactions recorded on the ledger.

Once the device is deployed in a remote location, it's an essential part of device management to remotely communicate with the device for tasks like data access, device parameter tuning, firmware upgrades etc. The $subject$ first resolves the DID of IoT device it wants to remotely communicate to. The DID Doc of IoT device is returned to the $subject$ by the IPFS. The $subject$ sends a request to the IoT device for establishing a secure connection on the endpoint described in DID Doc of the IoT device under “Service Endpoint” list. The IoT device resolves the received DID of the $subject$ and verifies its identity. Until now, the device only knows that this $subject$ is a part of operator network as they share a similar *DID method*. The IoT device and subject requests each other to share their VCs for mutual authorization. The VCs are shared and both the subject and IoT device verify them through the VDR. The IoT device verifies if the $subject$ possesses authority to make any changes to device settings and $subject$ verifies if the device is what it claims to be. If claims are legitimate i.e. signed by the operator, the IoT device sends another endpoint to the $subject$ on which it can communicate securely with the device.

Table I: Architectural difference Between Blockchains

Features	**Ethereum**	**Hyperledger Indy**
Access Type	Public	Public
Validation Type	Permissionless	Permissioned
Smart Contract	Applicable	Not-applicable
Mining	Applicable	Not-applicable
Consensus Mechanism	PoW	RBFT
Wallet/Agent Support	No	Yes

IV. Simulation and Results

The experiment was setup on a Linux machine installed with the following hardware specifications: Intel Core i5-7200U CPU@2.50GHz and 8 GiB RAM. For Ethereum, a private Ethereum blockchain was used to deploy smart contracts, perform transactions and run tests. The smart contract was developed in Solidity v0.5.16. For Indy, a test network of Indy Pool Nodes was deployed that has three $TrustAnchor$ that act as $issuer$ to issue claims and act as $verifier$ to verify each other's claims.

Ethereum: We computed three parameters to understand the performance of the framework namely, $GasCost$, $ConfirmationTime$ and $StorageCost$. $GasCost$, refers to the fee required to successfully execute transactions and smart contracts on the Ethereum virtual machine (EVM) and is calculated as $GasCost(Gwei) = GasUsed * GasPrice$ and its unit is denoted as $Gwei$, $(1Ether = 10^9 Gwei)$. The gas is used to allocate resources of the EVM so that smart contracts can be executed in a secured manner. $GasPrice$ is maximum price the user is willing to pay in the network. We have calculated the *gas cost* with an average *gas price* of 15 $Gwei$. $ConfirmationTime$ shows the time required for a transaction or smart contract to be mined in the network, only after which it will be valid in the network. $StorageCost$ will represent overhead incurred for contract storage and IPFS for DID Doc storage. Table II and Table III shows the results.

Note that Identity creation has no associated cost and takes no time as it only involves creating an EOA account on the

Table II: Gas Used, Gas Cost and Confirmation Time

Txn/Contract Deploy	**Gas Used**	**Gas Cost(Gwei)**	**Time(sec)**
Identity Creation	0	0	0
DID Doc creation	32918	493770	31
EthereumDIDRegistry	1923852	28857780	172
VerifiableClaimsRegistry	410148	6152220	121
DID Registration	42800	642000	31
Issue Claim	44765	671475	31

Table III: Storage Overhead for Smart Contracts and DIDDocs

Txn/Smart Contract	**Storage Cost (KB)**	**Stored At**
EthereumDIDRegistry Contract	950	EVM
VerifiableClaimsRegistry Contract	350	EVM
DID Doc for each subject (approx.)	10-20	IPFS

Table IV: Framework Performance on Hyperledger Indy

Transaction	**Time(sec)**
Network Node Pool Setup	1.983
Steward Setup	2.607
Create wallet and DID for OEM	2.668
Create wallet and DID for Organization 1, $O1$	3.086
Create wallet and DID for Organization 2, $O2$	3.0197
Creating wallet and DID for $D1$	3.015
Creating wallet and DID for $D1$	3.392
OEM creates 2 schema definition on VDR	6.085
O1 to D1 Credential Issue	0.299
O2 to D2 Credential Issue	0.252
Time taken by D2 to verify D1 credential	0.2039
Time taken by D1 to verify D2 credential	0.331

blockchain and the keypair. DID Doc creation is a relatively less computationally expensive task. The contract deployment takes the maximum computational resource on EVM and hence have greater $GasCost$ and respective confirmation time. However, these are one time cost and therefore practically feasible. DID registration and issuing claims are also less resource and time consuming transactions in the network. As far as storage cost is concerned, the overhead is relatively low.

Hyperledger Indy: For Indy setup we deployed a pool of 4 nodes in the network that form the VDR. They store the transactions related to DID creation and VC issued by $TrustAnchors$. First a $Steward$ is created, these entities have authority to write transaction to the ledger. Later, steward on-boards three $TrustAnchors$, an OEM manufacturer and two organizations participating in business. Then we on-board two IoT devices and issue them DIDs and VCs. Device $D1$ was issued a credential with 5 claims and $D2$ with 7 claims. When these devices try to communicate with each other, they exchange a verifiable presentation of these credentials to authenticate each other. The time taken for each of these transactions are shown in Table IV. It's important to note that while one time transactions like network setup, $Steward$ setup, wallet and DID creation for $TrustAnchors$ take practically feasible time, the time taken to verify VCs are practically negligible, as it only involves verifier querying the ledger.

V. Conclusion and Future Work

This paper proposes a novel distributed-ledger based M2M digital identity framework for implementing autonomous remote device-to-device communication. Experiments conducted indicates feasibility of such an architecture with lightweight smart contract development and distributed storage services. In the extended version of this paper, a detailed analysis of our proposed architecture and comparison with current certificates infrastructure will be conducted.

References

[1] *"Cybersecurity incident - information for UK consumers"*. Available: https://www.equifax.co.uk/incident.html. [Accessed: 21-Dec-2021].

[2] P. C. Bartolomeu, E. Vieira, S. M. Hosseini, and J. Ferreira. "Self-Sovereign Identity: Use-cases, Technologies, and Challenges for Industrial IoT". In: *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1173–1180.

[3] R. Belchior, B. Putz, G. Pernul, M. Correia, A. Vasconcelos, and S. Guerreiro. "SSIBAC: Self-Sovereign Identity Based Access Control". In: *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. 2020.

[4] E. Bertino. "Data privacy for IoT systems: Concepts, approaches, and research directions". In: *2016 IEEE International Conference on Big Data (Big Data)*. 2016.

[5] *DDoS attack that disrupted internet was largest of its kind in history, experts say*. [Available]: https://www.theguardian.com/technology/2016/oct/26/ddos-attack-dyn-mirai-botnet. [Accessed: 14-Oct-2020].

[6] G. Fedrecheski, J. M. Rabaey, L. Costa, P. C. Calcina Ccori, W. T. Pereira, and M. K. Zuffo. "Self-Sovereign Identity for IoT environments: A Perspective". In: *2020 Global Internet of Things Summit (GIoTS)*.

[7] D. Khosrowshahi. *"2016 Data Security Incident"*. [Online]. Available: https://www.uber.com/newsroom/2016-data-incident/. [Accessed: 21-Dec-2021].

[8] D. Lagutin, Y. Kortesniemi, and N. Fotiou. "Enabling decentralised identifiers and verifiable credentials for constrained Internet-of-Things devices using OAuth-based delegation". In: *Workshop on Decentralized IoT Systems and Security, San Diego, CA*. 2019.

[9] A. Othman and J. Callahan. "The Horcrux Protocol: A Method for Decentralized Biometric-based Self-sovereign Identity". In: *2018 International Joint Conference on Neural Networks (IJCNN)*. 2018, pp. 1–7.

[10] R. Soltani, U.T. Nguyen, and A. An. "A New Approach to Client Onboarding Using Self-Sovereign Identity and Distributed Ledger". In: *2018 IEEE International Conference on Internet of Things (iThings)*. 2018.