Credentials as a Service

Providing Self Sovereign Identity as a Cloud Service Using Trusted Execution Environments

Hira Siddiqui*, Mujtaba Idrees*, Ivan Gudymenko*, Do Le Quoc†, Christof Fetzer†
*T-Systems Multimedia Solutions GmbH {Hira.Siddiqui, Mujtaba.Idrees, Ivan.Gudymenko }@t-systems.com,
†Scontain {Do, Christof.Fetzer}@scontain.com

***Abstract*—With increasing digitization, more and more people use their identification credentials for accessing online services; which increases concern for data privacy. To ensure user's privacy, alternate credential management schemes must be adopted. Self-Sovereign Identity (SSI) is a form of credential management where users are in charge of their credentials. Privacy-critical data is stored only at the user's end and they can choose to do selective disclosure of minimal required information to access services. Currently, SSI solutions are not being widely adopted by service providers and the ecosystem is fragmented. One of the reasons for the lack of adoption is the need for maintaining private infrastructure for credential issuance, as critical user information is to be handled during credential issuance. To cater to this, we present a solution that enables the service providers to run their credential issuers on public cloud - a so-called Credentials as a Service (CaaS). CaaS issuers run inside Trusted Execution Environments (TEE) enabling credential issuers to ensure user's privacy while enjoying the flexibility of the pay-per-use cloud model. CaaS can pave the way for making SSI credentials ubiquitous in identity management solutions.**

***Index Terms*—self sovereign identity, verifiable credentials, zero knowledge proofs, hyperledger indy, trusted execution environments, blockchain, privacy protection, SCONE**

I. Introduction

Nowadays, data privacy is an increasing concern. People share significant pieces of Personally Identifiable Information (PII) online like name, credit card numbers etc. Misuse of such data is a privacy concern and can lead to adverse consequences. Corporate giants like Facebook, PayPal, Government organizations etc. store data of millions of people and analyze them in privacy-invasive manner. Similarly, centralized systems for credentials verification like WES [1] also store sensitive identity information. In the past, there have been data breaches of Facebook [2], Yahoo [3] and Equifax [4] which affected millions of users. Hence, storage of privacy-critical data by trustworthy corporates makes them a honeypot for malicious entities. To protect the misuse of user's data either due to malicious intent or gross negligence, a law was created in Europe known as General Data Protection Regulation (GDPR) [5]. This law imposes large fines on companies in case of a data breach, putting huge responsibility on such corporates. An alternative way to sharing information and providing services online is by creating a Self Sovereign Identity ecosystem in which the services:

- do not store any personal identifiable information
- can verify that the data provided to them is correct
- can verify that the data is issued by a valid authority

In an SSI based system, the user is in control of his/her identity data and can choose to disclose parts of their identity credentials (only as much as required). Proof that data is provided by a valid authority without disclosing entire credential is also possible. Moreover, the credential issuing authorities and the services which need identity data from customers both can choose not to store user's data, while ensuring complete functionality of the system. Therefore, there is lesser responsibility on authorities and services for customer data handling and management.

Prior work has been done in the domain of SSI [6] [7] [8] [9] but the focus has never been on the requirements of service provider. The SSI ecosystem cannot evolve until enough service providers adopt SSI ecosystem, thus creating a chain of trust. The goal of this research is to bridge this gap by catering to the requirements of both the service providers and users. If such a system is created as a cloud service, it would lead to increased adoption due to the traditional cloud benefits like pay-per-use, relief from maintaining in-house infrastructure, on-demand scaling etc.

In a classic SSI ecosystem like Hyperledger Indy [10], the authorities acting as issuers of credentials maintain on-premise infrastructure, as outsourcing user's data and its processing to a public cloud provider would make the data vulnerable to various cloud related attacks for example root admin attack. This problem can be mitigated by creating a cloud-based system which provides holistic data protection (at rest, in transit and in runtime) along with confidential execution guarantees. To provide such protection, Trusted Execution Environments (TEE) [11] were used. TEEs enable the process execution inside a sealed area in CPU, thus providing hardware protection to the execution. TEEs provide confidentiality guarantees against malicious root admin etc. Therefore, if the authorities issue credentials using a cloud service over TEEs, they have security guarantees that confidentiality and integrity of client's data is preserved.

In this work an SSI solution named as Credentials as a Service (CaaS) is created which provides a credential issuance microservice running inside public cloud over Intel SGX, which is currently the most mature TEE implementation in the confidential computing paradigm. Hyperledger Indy [10] cloud agent has been made to run inside TEEs over Intel SGX using a code wrapper SCONE [12] [13]. CaaS can save legal (e.g. GDPR compliance) costs for cloud providers by alleviating the need to store sensitive data. Moreover, managing TEE hardware can open new business opportunities for cloud providers by allowing them to run applications with sensitive data e.g. medical records in their systems and overall enhancing the trust in public cloud.

II. Credentials as a Service: a general system setup

To show how the system works, an application was created which shows a university graduate who needs to apply for a job. The job application requires verifiable information from Identity Card and University Transcript. User will first receive his/her ID card credential and Transcript credential from the respective authorities and then add the required information in the job application.

The main actors considered in the target use case essentially represent the core entities of a digital credential system, namely an issuer, a holder and a verifier.

Government and University will run their CaaS issuers in the public cloud inside TEEs to provide credentials to the users. A user will first take the credentials from issuers and then present them to the verifier. Figure 1. shows the high level application flow.

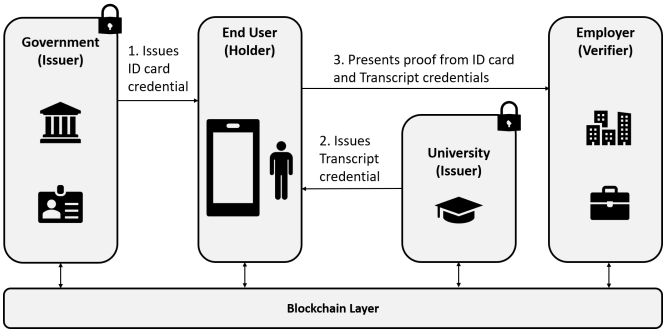

Fig. 1. The credential issuance and proving process in the example application

The identity card credential is issued by the government issuer and follows the schema *[Name, Age, Gender, Place of Birth]*. The transcript credential is issued by the university issuer and follows the schema *[Name, Degree, Grade, Enrollment Date]*.

Now, in traditional setting, user would get credentials and present them to the employer in whole, disclosing essentially in an all-or-nothing fashion. However, in this scenario, user would selectively disclose the minimum required information as shown in Table I. The employer would not know extraneous

TABLE I
Selective and ZKP-based disclosure of attributes

Attribute	Disclosure Type
Name	Plaintext
Degree	Plaintext
Proof that age >= 18	Zero Knowledge Proof
Proof that grade <= 3	Zero Knowledge Proof

information from shared credentials i.e. gender, place of birth, enrollment date. The employer would also not know the exact value of attributes presented as zero knowledge proofs [14] i.e. age and grade, it would just know that they meet the required criteria.

A. Design Goals

The goal of this work is to ensure that organizations can issue verifiable credentials to their clients in a confidential and integrity-protected manner even when they run their issuer services on third-party cloud providers. In the entire lifecycle of this system, it must be ensured that:

- The client's information must not be visible to unintended parties
- The client's credential information must only be stored at the client's side e.g. in user's app
- The issuer's functionality cannot be changed i.e. the code and configuration's confidentiality and operating system files's integrity is ensured.
- Proof presentation can be done in a peer-to-peer fashion i.e. the client must be able to create composite proofs and do selective disclosure with zero knowledge functionality to third-parties (verifiers) utilizing the previously acquired credentials without involving issuers
- The availability and non-repudiation of public information of the issuers is ensured

B. Threat Model and Protection Mechanisms

Our threat model comprises of an omnipotent adversary that can attack everywhere: on the cloud as the root admin, on the memory and OS libraries, on the code and config files of issuers and on the data over the network. The only threat surface we do not consider is when the data is in user's wallet. We protect against the threats as follows:

- HTTPS connections are used to protect the data packets over the network
- Access to the memory or execution information of issuer is restricted by using secure enclaves
- Changes in code, configuration and dynamic libraries is inhibited using SCONE's file system protection file (fspf) [15]. All the metadata required for checking the consistency of files is stored in fspf and changes to selected files and libraries can be tracked.
- The secrets necessary to decrypt the files are supplied to the issuer after verification that they have correct software stack (MrEnclave [16]).

III. System implementation leveraging Hyperledger Indy and Scone Trusted Execution Environments

A. Technical Background

Trusted Execution Environments (TEE) and Secure Container Environment (SCONE)

Trusted Execution Environments (TEE) is an environment for secure execution on untrusted systems in a confidential and integrity preserving manner. The wrapper framework of TEE that is used in this system is Secure Containers Environment (SCONE) [12] [13]. SCONE was chosen because it allows to build and run secure containers on top of Intel SGX hardware without having to re-write applications. The applications only needs to be recompiled with SCONE cross compilers to run it in a containerized environment. Moreover, it also offers features like transparent attestation of system state, file system protection [15] i.e. protection of data-at-rest and secrets management.

SSI Framework and Hyperledger Indy

SSI frameworks provide the functionality to issue, verify and store decentralized verifiable credentials. In CaaS, we use Hyperledger Indy [10] as the SSI framework. Indy facilitates creation of Decentralized Identifiers (DIDs) and Verifiable Credentials supporting selective disclosure and Zero Knowledge Proofs (ZKP). It supports a consortium of member organizations and uses Plenum blockchain as the distributed ledger. However, in CaaS, we use public Ethereum blockchain instead of plenum.

B. System Components

Figure 2 shows the system components used in CaaS. The system can easily be extended to include as many issuers, verifiers and users as needed.

Issuers: Issuers are microservices setup by organizations that

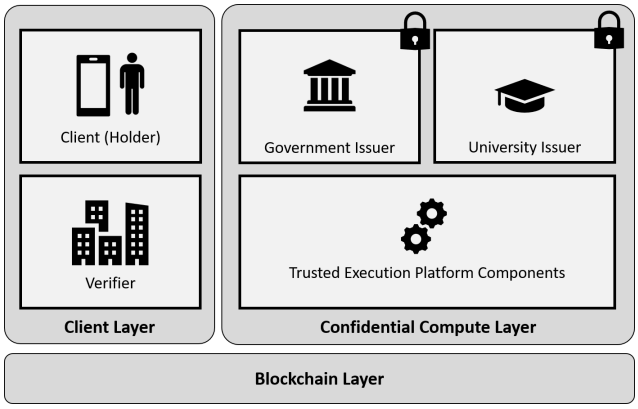

Fig. 2. System components

issue verifiable credentials. These issuer services run as secure containers (enclaves) on public cloud.

Verifiers: Verifiers are services that verify user's credentials. They are able to check whether the provided proof is correct and is issued by a valid issuer using underlying SSI framework libraries. In CaaS, the verifier does not need to contact the issuer for proof verification (see design goals II-A). It only needs public issuer and credential information from blockchain. This approach has two benefits:

- Load on issuers is reduced by eliminating the requests made to issuer during verification
- Blockchain provides high availability due to its inherent design utilizing redundancy, so instead of asking information from issuer, it can be fetched from blockchain.

Holder: Holders are entities who wish to leverage self-sovereign identity. The holders get their credentials from issuers, create their proofs using selective disclosure and zero knowledge proofs and provide these proofs to the verifier. Their personal credential information is only stored in their own wallet.

Distributed Trusted Registry: Hyperledger Indy uses plenum consortial blockchain. However, one of the core novelties of CaaS is that it serves as a public SSI ecosystem. To do this, CaaS was developed to use Ethereum smart contract as a publicly-accessible distributed trusted registry that stores issuer's public information and the credential schema they are issuing. Ethereum was chosen because of its mature smart contract platform and large public adoption. In this way, CaaS converts a consortial SSI ecosystem to a public one.

Trusted Execution Platform Components: In our solution, we used SCONE Configuration and Attestation Service (CAS) and SCONE Local Attestation Service (LAS) as the trusted platform components. These services (also running as enclaves on third-party cloud provider) are provided by SCONE that perform local and remote attestation. Together they attest that the enclaves registered with them are running in a valid Intel SGX hardware and with the correct system state (MrEnclave) i.e. they ensure integrity and authenticity of enclaves. Any enclave can be registered with SCONE CAS by using a "session". A session holds secrets. SCONE CAS does safe-keeping of these secrets and only allow access by legitimate enclaves.

Our issuer enclaves are also registered with SCONE CAS using sessions. When SCONE CAS verifies that the issuers are running inside enclaves with correct system state (MrEnclave), only then it lets them access the secrets.

C. Workflow

At first, our system needs to setup in a trusted manner and after that it can securely issue and verify credentials in a trustless setup.

Trusted Setup

1) **CAS and LAS Setup:** Firstly, SCONE CAS and SCONE LAS should be setup on the cloud machine.
2) **Smart Contract Setup:** A public smart contract is published. It stores the public issuer information and the details of the credentials that the issuers will provide.
3) **Issuer Bootstrap:** An issuer needs to be bootstrapped before it can start issuing credentials.

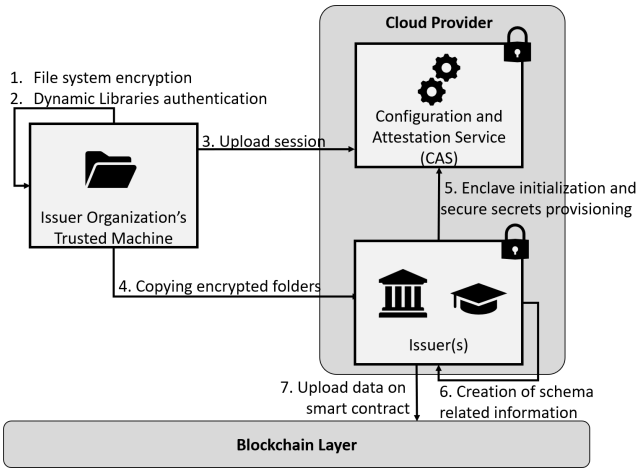

Fig. 3. Issuer Bootstrapping

Issuer bootstrap has seven steps as shown in (Figure 3) and explained below:

a) File System Encryption: Issuer's organization would encrypt all of the issuer code, configuration files and the wallet files on its trusted machine before transmitting it to untrusted cloud provider. SCONE FSPF [15] mechanism is used for encryption.

b) Dynamic Libraries Authentication: Our issuer compilation includes dynamic libraries. These libraries are added at runtime and therefore can be used by an attacker to inject unwanted functionality. Therefore, to protect from such an attack, dynamic libraries are authenticated using SCONE FSPF before starting the issuer.

c) Uploading of Session on CAS: Issuer organization first attests SCONE CAS using Intel Attestation Service (IAS) [17] [18]. Issuers are then registered with the CAS by uploading their session. The session contains secrets like file system decryption keys, hashes to authenticate dynamic libraries and TLS certificate for setting up secure endpoint. The secrets can only be accessed by a service with same MrEnclave as specified in the uploaded session.

d) Copying Encrypted Folders: All the encrypted files, folders and configurations necessary for the functioning of issuer are then uploaded to the cloud provider's machine.

Trustless Setup

e) Enclave Initialization: At this point, the enclave needs to be initialized on the public cloud in a trustless manner. Issuer's session is invoked on CAS. If the machine or container invoking the session has the same MrEnclave as in the uploaded session, CAS would give access to the session's secrets and run the issuer with the specified environment i.e. keys to decrypt the folders and TLS certificates to create secure connections with clients.

f) Creation of credential information and storage in SSI wallet: The issuer then creates a schema and credential definition with the required attributes by calling the SSI framework functions. It stores this information in its local SSI wallet.

g) Publishing of credential and issuer public information on blockchain: Once the schema and credential definition are created, the issuer publishes the information on the blockchain through smart contract. All the information required to securely connect and independently verify an issuer gets published including the session name, schema and credential definitions. The session name is published so that the clients know which session's public certificate is needed to securely connect to the required issuer. The schema and credential information are needed by the verifier to verify the proofs submitted by client. The schema is also needed by the clients to know which credential is being offered by the issuer. At this point, the issuer becomes ready to issue credentials and waits for incoming client requests over the secure TLS channel.

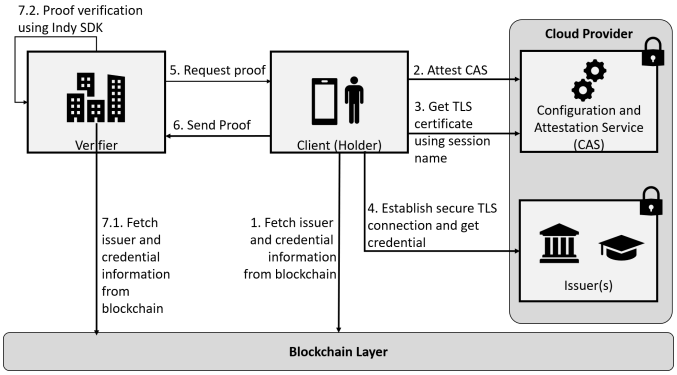

Fig. 4. Credentials acquisition and verification workflow

Clients can now use CaaS to get credentials and prove identity to verifiers. In this paper, we assume that the client (student) and the issuer(s) know each other already. However, to authenticate a client, a pairwise DID between client and issuer is created which is used to issue credentials. The generation of this pairwise DID can be done after manually verifying the student or national ID card or it can be a result of another SSI verification where the issuer would first verify the ID card before issuing credentials. To get the credential, client follows the steps shown in Figure 4 and explained below:

1) **Get data from smart contract:** Issuers advertise their names publicly which clients use to get the corresponding issuer and their credential public information from blockchain. Technically, the issuer authentication can also be done via centralized data stores. However, blockchains provide additional value as in terms of high availability, decentralization and immutability of the information.
2) **SCONE CAS Attestation:** Client would attest SCONE CAS which would ensure client that CAS is a non-malicious service running on a valid Intel SGX hardware with the expected software state.
3) **Get TLS certificate from SCONE CAS:** The client uses the session name that it got from blockchain to get issuer's TLS certificate from CAS. Client uses this certificate to create a secure connection with the issuer.
4) **Get credential from issuer:** The client sends its information to the issuer over the secure TLS connection. The issuer creates a credential for the client and sends it back over the same secure channel. The client then stores it in its wallet.
5) **Verifier requests proof from client:** The verifier service is setup before the proof verification takes place.
   The verifier sends a proof request to the client which contains (1) attributes needed as plaintext i.e. selective disclosure (2) predicates that need to be satisfied i.e. zero knowledge proofs.
6) **Client sends proof to verifier:** Client uses the credentials already stored in its wallet to create the proof adhering to the proof requirements. This proof is then sent to the verifier over a secure communication channel.
7) **Proof verification:** After receiving the proof, verifier

checks the credentials used to construct the proof. It fetches the corresponding issuer and credentials information from the blockchain. The verifier checks:

- The proof is mathematically correct and all the predicates are satisfied
- It has been constructed from credentials that were required by the verifier
- The credentials that were used to create the proof were issued by a valid and trusted issuer

These checks are performed using the Hyperledger Indy SDK functions and then by verifying information published on the blockchain.

IV. Evaluation

A. Evaluation Hardware and Configuration

CaaS was evaluated on an Intel SGX-enabled server (SGX version 1), a Xeon E3-1280v6 @ 3.90 GHz. The Enclave Page Cache (EPC) was 128MB [19]. The server had ubuntu 18.04.3 LTS as its operating system, a RAM of 62 GB and a total of 8 processors. JVM Heap memory was fixed to 256 MB in all tests. Our SCONE HEAP was set to 8GB. Only one instance of issuer(implemented in Java) was tested.

B. Evaluation Modes

It was essential to know the performance impacts of running issuers with secure enclaves v/s without the enclaves. Therefore, the performance of issuer was evaluated in three modes:
1) SCONE Hardware: Hardware mode is when the issuer is running inside a secure enclave.
2) SCONE Simulation: Simulation mode is used to run SCONE containers on machines that do not have Intel SGX support. Simulation mode is used for debugging or development.
3) Native: Native execution is when the issuer is running as a native java based microservice without the environment of SCONE/Intel SGX.

C. Evaluation Results

Sequential and parallel requests were sent in bulk to the issuer and results were evaluated in terms of latency, CPU usage and JVM Heap usage.
1) Experiment: Sequential execution with hardware mode
In hardware mode, a sequential request took 0.13s to create a credential on average, CPU utilization was <=0.1% and the JVM heap usage was 20-30%.
2) Experiment: Comparison of increasing parallel requests in hardware mode
The issuer running in hardware mode could not handle more than 300 parallel requests. The average latency of issuer starts from 0.13s for a single request and reaches to 64.48 seconds in 300 parallel requests as shown in Figure 5. With more than 300 parallel requests, the issuer stops responding. The bottleneck was JVM heap usage. CPU consumption remains under 1% for all workloads. However, JVM heap consumption, reaches to 80% on 300 parallel requests as shown in Figure 6 at which point it stops responding.

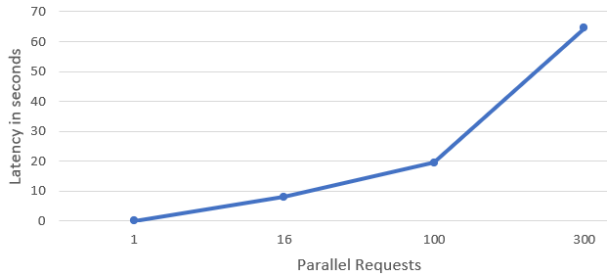

Fig. 5. Issuer latency in hardware mode

3) Experiment: Comparison of executions in hardware, simulation and native mode
The previous experiment was repeated with issuers running in different modes and it was found that native executions have the best performance. Simulation mode has performance close to native, and hardware mode has the worst performance.
While hardware mode can only process 300 parallel requests, simulation mode is capable of handling 15000 parallel requests and native mode is capable of handling 30000 parallel requests as shown in Figure 6 Once again JVM heap was found to

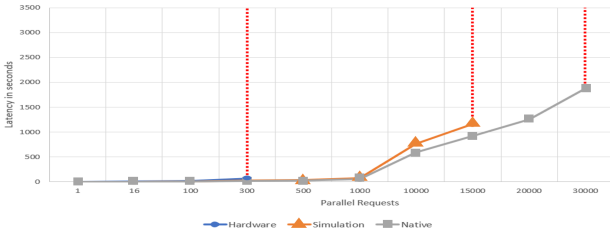

Fig. 6. Issuer latency in different modes

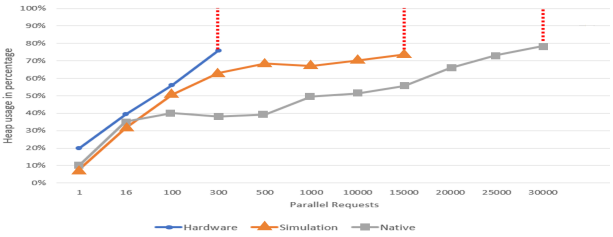

Fig. 7. Issuer heap usage in different modes

be the bottleneck and in each of the three modes, whenever JVM heap reaches 80%, the issuer stopped responding. This trend can be seen in Figure 7. The cpu usage remains low in all cases. In the future, issuers can be implemented in lightweight languages like Rust to improve performance. On average, an overhead of 3.3 times was observed when running the issuer in hardware mode and 1.16 times in simulation mode.
4) Experiment: Enclave Bootup Time
On average, enclave bootup time was 5 minutes and 47 seconds. Currently, we evaluate our prototype at the SGXv1, which provides slow boot up time. However, with SGXv2, the boot up time would be improved significantly. The new Intel Icelake hardware also don't have the limitation about the

EPC size [19] which would also improve performance during runtime.

V. Related Work

Improving the adoption of fragmented SSI landscape has been a long-standing research objective. Some initial works [20] describe that non-usage of established access management protocols are one of the reasons for minimal SSI adoption by service providers. Therefore, integration architectures were proposed that allows service providers to utilize SSI functionality in their web applications whilst using their existing authentication protocols. However, these architecture did not cover the aspect of credentials issuance in the cloud setting whilst maintaining privacy of client's data.

Furthering this research to improve the SSI adoption, PRIME Core [21], an anonymous credentials systems for web applications based on IDEMix [22] has also been developed. It discusses that the SSI systems have become like a chicken and egg problem. Users cannot use SSI because there are not enough SSI-based service providers and service providers do not adopt the SSI infrastructure as the landscape is very fragmented for now. Therefore, an easy to use web based anonymous credentials systems can help in boosting SSI adoption. However, in PRIME Core, the credential issuance happens within the trusted boundary of the organization to preserve the privacy of user's data and does not allow generating credentials on a third-party cloud provider. Our solution CaaS, however, allows this.

Before SSI based credentials, a lot of research was done in field of blockchain based credentials [23] [24] where blockchain was used as a root of trust for verifying whether the credentials are valid or not. These solutions naturally did not support functionalities of selective disclosure and zero knowledge proofs like SSI. Therefore, SSI is now the next step forward in user-centric identity. Furthering this arena of research, another aspect in SSI adoption being discussed in recent works [6] is forming the technical backbone of SSI ecosystems over blockchain. Such works explain that SSI can now be made ubiquitous by laying its foundations on top of blockchain as many of its properties coincide with the desired properties of self-sovereign identity for example availability, immutability, distributed data control and transparency. Such research contributions gives credibility to CaaS where Blockchain has been utilized to provide an available, transparent and distributed registry for storing the information related to issuers and the credentials they offer.

Similarly, SSI-based use cases are also being explored in the industry to find practical steps on adopting the SSI ecosystem, furthering the importance of CaaS. T-Systems Multimedia Solutions GmbH and BMW explored the use case of car-mobility services based on SSI credentials [25] using Hyperledger Indy and Zokrates. It discusses a proof-of-concept application built for car-sharing where users take SSI credentials from organizations and later present them to a verifier using selective disclosure to get access to a car from mobility pool. This paper did not focus on the requirements of provider and the possibility of creating credentials in a third-party cloud provider whilst maintaining privacy. However, it served as a precursor to our research in CaaS.

VI. Future Work

In the future, CaaS can be shifted to new generation SSI platform known as Hyperledger Aries [26] [27], hardware can be shifted to Intel SGXv2 for improved performance and other TEE offerings like Graphene [28] can be explored. Currently, we have designed CaaS in a manner that it can easily scale with the increasing number of requests i.e. we can spawn new issuers and the attestation will be performed automatically using CAS [29]. However, detailed scalability tests need to be done.

VII. Conclusion

Self Sovereign Identity (SSI) is a modern approach to provide and consume services which require certain aspects of personal information to be processed in order to ensure service provision. It allows users to be in control of their identity attributes (such as age, name etc.) by giving them the freedom to allow what information to reveal and to whom. Multiple SSI solutions are available but the ecosystem is fragmented and the SSI adoption is low, largely because the needs of service providers and credential issuers in SSI domain have been ignored. There exists a need for a system that makes it convenient for all parties to adopt SSI based credentials. One way to do this is by enabling providers to run their issuers on third-party cloud machines while providing them guarantees that their client's privacy will be preserved using Trusted Execution Environments (TEEs).

This paper presents a neoteric solution to identity known as Credentials as a Service (CaaS). CaaS is an SSI ecosystem where credential issuers can create SSI credentials for their users on a third-party cloud machine using secure enclaves while service providers (verifiers) can verify proofs created from these SSI credentials. The system provides guarantees that the user's data would not be compromised in the presence of omnipotent adversary as the credential issuance is happening inside trusted execution environments and user's data is only stored at the user's end. To find the required issuers, clients use an Ethereum smart contract that serves as a distributed trusted registry. CaaS is an empowering system that can have huge social impacts by aiding in creating workflows that inhibit discrimination and can pave the way to a society based on the principles of equality. It is built on top of Hyperledger Indy SDK, SCONE Trusted Execution Environment and Ethereum Blockchain. Using CaaS, organizations with minimal resources can become a part of self-sovereign identity ecosystem and applications can be created based on existing SSI credentials. The issuers used in our system can issue around 460 verifiable credentials per minute and can serve up to 300 concurrent credential issuance requests. However, there is no free lunch. Our TEE based issuers are up to 3 times slower than non-TEE issuers.

REFERENCES

[1] WESTeam. World education services. [Online]. Available: https://www.wes.org/

[2] E. Boldyreva, "Cambridge analytica: Ethics and online manipulation with decision-making process," 12 2018, pp. 91–102.

[3] L. J. Trautman and P. C. Ormerod. Corporate director's and officer's cybersecurity standard of care: the yahoo data breach. [Online]. Available: https://ssrn.com/abstract=2883607

[4] J. Thomas, "A case study analysis of the equifax data breach 1 a case study analysis of the equifax data breach," 12 2019.

[5] D. Savić and M. Veinović, "Challenges of general data protection regulation (gdpr)," 01 2018, pp. 23–30.

[6] M. S. Ferdous, F. Chowdhury, and M. O. Alassafi, "In search of self-sovereign identity leveraging blockchain technology," *IEEE Access*, vol. 7, pp. 103 059–103 079, 2019.

[7] A. Mühle, A. Grüner, T. Gayvoronskaya, and C. Meinel, "A survey on essential components of a self-sovereign identity." [Online]. Available: https://arxiv.org/pdf/1807.06346.pdf

[8] D. van Bokkem, R. Hageman, G. Koning, T. L. Nguyen, and N. Zarin, "Self-sovereign identity solutions: The necessity of blockchain technology," 04 2019.

[9] A. Tobin and D. Reed, "The inevitable rise of self-sovereign identity." [Online]. Available: https://sovrin.org/wp-content/uploads/2018/03/The-Inevitable-Rise-of-Self-Sovereign-Identity.pdf

[10] H. I. Community.

[11] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted execution environment: What it is, and what it is not." [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7345265

[12] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'Keeffe, M. L. Stillwell, D. Goltzsche, D. Eyers, R. Kapitza, P. Pietzuch, and C. Fetzer, "SCONE: Secure linux containers with intel SGX," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, 2016, pp. 689–703. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/arnautov

[13] SconeTeam. Scone. [Online]. Available: https://sconedocs.github.io/

[14] J. Hasan, "Overview andapplications of zero knowledge proof (zkp)," vol. 8, p. 5, 10 2019.

[15] S. Team. Scone file shield. [Online]. Available: https://sconedocs.github.io/SCONE_Fileshield/

[16] ——. Scone. [Online]. Available: https://sconedocs.github.io/MrEnclave/

[17] V. Costan and S. Devadas, "Intel sgx explained," *IACR Cryptol. ePrint Arch.*, vol. 2016, p. 86, 2016.

[18] T. Knauth, M. Steiner, S. Chakrabarti, L. Lei, C. Xing, and M. Vij, "Integrating remote attestation with transport layer security," 01 2018.

[19] T. Dinh Ngoc, B. Bui, S. Bitchebe, A. Tchana, V. Schiavoni, P. Felber, and D. Hagimont, "Everything you should know about intel sgx performance on virtualized systems," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 3, no. 1, Mar. 2019. [Online]. Available: https://doi.org/10.1145/3322205.3311076

[20] A. Grüner, A. Mühle, and C. Meinel, "An integration architecture to enable service providers for self-sovereign identity," in *2019 IEEE 18th International Symposium on Network Computing and Applications (NCA)*, 2019, pp. 1–5.

[21] B. Kellermann and I. Scholz, "Anonymous credentials in web applications – a child's play with the prime core." [Online]. Available: http://dud.inf.tu-dresden.de/~ben/kellermann_scholz09_anonymous_credentials_in_web_applications.pdf

[22] J. Camenisch and E. Herreweghen, "Design and implementation of the idemix anonymous credential system," *Proceedings of the ACM Conference on Computer and Communications Security*, 05 2003.

[23] C. Bapat, "Blockchain for academic credentials," 2020. [Online]. Available: https://arxiv.org/abs/2006.12665v1

[24] M. Jirgensons and J. Kapenieks, "Blockchain and the future of digital learning credential assessment and management." [Online]. Available: https://files.eric.ed.gov/fulltext/EJ1218203.pdf

[25] I. Gudymenko, A. Khalid, H. Siddiqui, M. Idrees, S. Clauß, A. Luckow, M. Bolsinger, and D. Miehle, "Privacy-preserving blockchain-based systems for car sharing leveraging zero-knowledge protocols," pp. 114–119, Aug 2020.

[26] H. A. Community. Hyperledger aries. [Online]. Available: https://github.com/hyperledger/aries

[27] ——. Hyperledger aries rfcs. [Online]. Available: https://github.com/hyperledger/aries-rfcs

[28] C.-C. Tsai, D. E. Porter, and M. Vij, "Graphene-sgx: A practical library os for unmodified applications on sgx," in *Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference*, ser. USENIX ATC '17. USA: USENIX Association, 2017, p. 645–658.

[29] F. Gregor, W. Ozga, S. Vaucher, R. Pires, D. Le, S. Arnautov, A. Martin, V. Schiavoni, P. Felber, and C. Fetzer, "Trust management as a service: Enabling trusted execution in the face of byzantine stakeholders," 06 2020, pp. 502–514.