Delegated Anonymous Credentials with Revocation Capability for IoT Service Chains (DISC)

Sandeep Kiran Pinjala[1,2] and Krishna M. Sivalingam[1]
[1]*Dept. of Computer Science and Engineering, Indian Institute of Technology Madras, Chennai, India*
[2]*HCL Technologies, Chennai, India*
Email: sandeepkiranp@gmail.com, cs16s001@smail.iitm.ac.in, skrishnam@iitm.ac.in, krishna.sivalingam@gmail.com

***Abstract*—With growing influence of Internet Of Things on our day-to-day lives, ensuring privacy of the users or devices who act on behalf of the users has been a challenging task. One because the these IoT devices are very constrained in resources that they cannot themselves implement highly secure logic and two because these devices are mostly placed in the open and are easily susceptible to physical attacks. IoT services normally do not act in silos. They interact with each other to provide a complete package of services. In this paper we discuss the problem of ensuring user and device privacy when a chain of IoT services are invoked on an event. We discuss the system architecture and components in detail and provide various options for implementing the solution on less constrained and very constrained services. Finally, we capture the results of our implementation and show how the scheme can easily scale-up in the IoT world.**

***Index Terms*—**

I. Introduction

Internet of Things (IoT) enables physical objects also called *Things* to communicate with each other and to their human operators over the internet. This opens up a myriad of use cases such as smart homes, smart factories, smart cities, smart healthcare, smart grids etc [1]. It is also expected that such connected devices could reach upto 50 billion by 2020 [2]. The IoT devices (for example, a smart bulb or a temperature sensor) are very constrained in terms of memory, processing power, storage and most often are battery powered. Unlike the traditional computers these devices cannot perform computationally intensive tasks and are intended for minor operations of sensing and actuation. Most of these devices are out in the open without any physical supervision making them easily susceptible to physical attacks.

Owing to the resource constraints and physical openness, IoT devices have been targets of various attacks [3] at physical, network and application layers. IoT devices also collect lot of personal information like user's location, eating habits, medical history etc of its users because of which there has been a growing concern among consumers of such services. Unlike normal computers, these devices cannot provide an interface where the user can look up what personal information is being shared and with whom. In [4] the authors define privacy in IoT as a guarantee for the subject

a) To be aware of the privacy risks imposed by smart things.
b) Control over collection and processing of personal information
c) Control over subject's personal information being disseminated outside of his control sphere.

They then categorize privacy threats and challenges of IoT into a) Identification b) Localization and tracking c) Profiling d) Privacy-violating interaction and presentation e) Inventory and life cycle tracking and f) linkage.

IoT services do not act in silos. They interact with each other and with external entities to provide a complete package of services to the user. For example, in Home Automation, based on the user who is entering the house (say Owner vs Guest), a completely different set of services may get invoked. The service interactions and invocations depend on the roles and capabilities of the user invoking them. We call the sequence of services that get invoked as *IoT Service Chain*. In this paper we look at providing security and privacy to users and IoT devices that invoke IoT service chains. The rest of the paper is organized as follows. Section II describes the motivation, problem statement and our contribution. Section. III talks about the preliminaries on which the system is built. Section IV talks about the anonymous credential mechanism that we will use in our scheme. Section V talks about the architecture and the components. In Section VI we evaluate the security aspect of the proposal and in Section VII we talk about the implementation aspects and the results.

II. Motivation and related work

1) IoT Service Chains: We introduced *IoT Service Chains* in Section I to refer to the chain of services invoked when an event occurs. Individual services in the chain interact with each other, on-behalf of the initiator towards a common goal. Initiator could either be a human or an IoT device. Each service in the chain would in-turn validate the user/IoT device's credentials for the desired service functionality. IoT devices can either act on behalf of their operator or can be independent of it. For example, if a smart Heart Monitoring System (HMS) detects a low pulse rate for a patient, it immediately needs to initiate the Advanced Cardiac Life Support (ACLS) by injecting an IV of an antidote, reading and interpreting the ECG, starting CPR, inform the doctor etc. The HMS in this case acts as an independent device and does not impersonate the patient. But in case of a smart fridge, which keeps track

978-1-5386-4980-0/19/$31.00 ©2019 IEEE

of the stock of milk, it automatically places an order for replenishment by using the owners credit card details. In this case, the smart fridge acts on behalf of the owner.

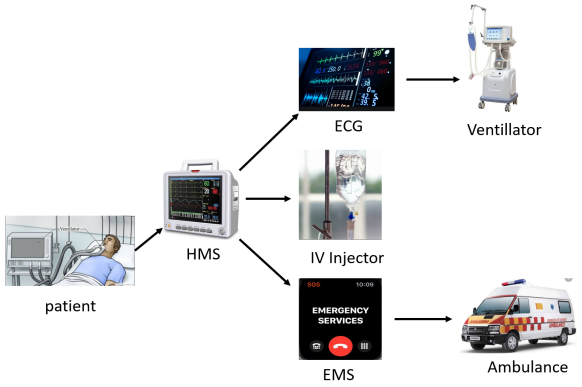

Fig. 1. IoT Service Chain

As can be seen in Fig. 1, the HMS continuously monitors the patients pulse rate. If the pulse goes below a threshold value, it immediately invokes the ECG service, the IV injector service and registers with the EMS system. The EMS may in turn call the ambulance. The ECG service detects any abnormalities in the heart rate and if so, may immediately put the ventilator into service.

One thing to notice from the above figure is that while the service chain is being invoked, the user/IoT device has to be online so that it supplies the necessary credentials for authentication and authorization. This isn't a huge problem if the initiating entity is a user (having a smart phone or a tablet). But for a constrained IoT device generating authentication and authorization information for each service in the chain would quickly result in draining of its resources. Also, most of the IoT devices are duty cycled and may not remain active till the chain completes. This again results in loss of packets or in retransmission. It will be even more challenging if the user/device's privacy needs to be protected through out the chain. In this paper we focus on some of the options on how user or device privacy can be ensured when invoking the service chain without pushing the device to its resource limits.

2) Attribute Based Anonymous Credentials: In traditional credential based system, a central authority grants credentials to users and systems. These credentials can be password or token (for example, certificate) based. When the user wants to access a resource he presents the credential to it which inturn validates the credential. The resource (also called service) trusts the central authority and thereby the credentials issued by it. Once the credential verification is done, the service checks whether the user has got the right access level to access the resource. One of the primary concerns with such a system is that the service gets to know the complete details of the user presenting the credential. For example, when user's Digital Certificates are presented for access, the service will know how long the credential (certificate in this case) is valid, the country, state, organization, email address etc of the user, who are all the intermediate chain of issuers etc. The only information that matters to the service is whether the certificate is valid and is issued by the central authority and whether the user possesses the required access rights. But in this case lot more user details are available to it. With so much personal information available to the service, there is always a chance of misuse of data if it falls in wrong hands. According to European Union General Data Protection Regulation (GDPR) Data Minimization principle, entities should only process adequate, relevant and limited personal data that is necessary in relation to the purposes for which they are processed. As mentioned in [4], the privacy problem becomes manifold in IoT world. One, because there are huge number of constrained IoT devices without proper security measures in place. And two, there is very little control over what personal information is disseminated from these devices to the outside world.

Anonymous Credentials introduced in [5] provide a way in which a user can prove that he holds a credential without revealing any information about the user. The verifier cannot also forge the user's credential. In Attribute Based Anonymous Credentials, a user can selectively prove that he holds the set of attributes needed by the verifier and not reveal all of his attributes, thereby maintaining privacy. The user creates a zero-knowledge proof of possession of the credential which the verifier verifies using the public key of the central authority. Some of the most popular Anonymous Credential Systems (ACS) are [6] and [7].

A Delegatable Anonymous Credential (DAC) System, introduced in [8] not only allows the users to generate anonymous credentials, but also allows them to anonymously delegate their credentials to other entities. For example, in a hierarchical setup, the root issuer may delegate its issuing authority to region wise issuers who in turn may delegate to division wise issuers and so on. Although there can be multiple levels of delegation, the anonymous credential generated by the prover at any level can only be verified by the public key of the root issuer. One of the primary advantages of DAC is that it alleviates the burden on the root issuer to issue credentials but still maintains anonymity of all the issuers in the chain.

3) Our Contribution: In section II-1 we talked about IoT service chains and the difficulty of ensuring user/IoT device's privacy during the chain propagation. We address this problem with our proposal on ***Delegated Anonymous Credentials in IoT Service Chains*** **called DISC**. We describe a mechanism in which the IoT device can delegate its credentials to a Controller which in turn generates an anonymous credential based on the attributes needed by the service. Our scheme is based on the DAC system developed by Camenisch *et al.* [9]. The authors developed a scheme where credentials are not delegated anonymously but the presenter anonymously proves that the entire chain of delegation is valid. The verifier verifies the anonymous credential just with the root issuer's public key.

The following are major contributions in this work.

a) Discuss in detail the problem of ensuring privacy to users and IoT devices in IoT service chains.
b) Implementation of the DAC scheme outlined in [9]. We used the Pairing Based Cryptography Library from [10] for the implementation. We implemented the full L-level credential Delegation, Presentation and Verification.

c) Credential revocation has been mentioned as "future work" in [9]. We extended [9] to incorporate credential revocation.
d) Once the verification of the anonymous credential token completes, the next logical step would be to communicate with the prover to exchange data. In order to do that securely, a common session key needs to be established between the two parties. We demonstrate how the session key can be established.
e) We then used the above DAC implementation to realize DISC. We outline the framework consisting of Root issuer, Controller, User/IoT device and Service as the principal components. We describe the messages exchanged between these components and outline how privacy of users can be ensured.
f) We implemented DISC and discuss the various possibilities of how token verification and policy implementation can happen for constrained IoT services. Various metrics like time taken, number and size of messages exchanged, memory, CPU etc are evaluated for the various models that we discuss.

4) Related Work: Subsubsection text here.

III. Preliminaries

In this section we introduce some of the concepts related to DAC building blocks.

1) Bilinear map: A Bilinear map is a Pairing-Based Cryptography construct that allows us to build and analyse cryptographic systems. Let G_1, G_2 and G_T be multiplicative groups of order q, then a bilinear map $e : G_1 X G_2 \rightarrow G_T$ satisfies the following properties

Bilinearity

$$\forall P \in G_1, \forall Q \in G_2 : e(P^a, Q^b) = e(P, Q)^{ab}$$

Non-Degeneracy

$$A \neq 0 \Rightarrow e(A, A) \neq 1$$

Computability

e is efficiently computable

2) Zero Knowledge Proofs: Zero knowledge Proofs (ZKP), introduced in [11] allow the prover to prove that he knows certain information without revealing the information (or anything related to it) to the verifier. A ZKP should be complete, sound and should not reveal any information from the proof. A zero-knowledge proof of knowledge is a special case where the prover possesses a secret information and not just the knowledge of it. Interactive ZKP require interaction between the prover and the verifier to prove that the prover has the secret information. For example, the Schnorr ZKP is outlined in [12]. The Non-Interactive ZKP (NIZKP) proofs do not require any kind of interaction for the proof to proceed. The Schnorr NIZKP proof is obtained through the Fiat-Shamir transformation [13].

3) Signature Schemes: A signature scheme consists of the following algorithms.

a) Setup : Given the security parameter, the algorithm outputs public parameters.
b) Key Generation : This algorithm generates a public key and the corresponding private key based on the public parameters.
c) Signature Generation : This algorithm takes the private key and the message and outputs the signature. The generated signature should not be forge-able.
d) Signature Verification : This algorithm takes the signature, public key and the message and outputs whether the verification is successful or not.

[9] uses Groth's structure preserving signature scheme [14] where the public keys, messages to be signed and the signatures are all elements of group $G1$ or $G2$.

IV. DAC By Camenisch *et al.* [9] and our extensions

The authors in [9] built a practical attribute based DAC system of L-level hierarchy where the delegation is not anonymous. The delegator reveals its identity and the identity of the entire chain to the delegatee during the delegation process. It is only during the presentation of the anonymous credential, the prover hides the identity of all the intermediate delegators and himself. The prover also selectively reveals attributes at each level of delegation. The verifier in turn uses the root issuer's public key to verify the anonymous credential token. The authors also describe the instantiation of their DAC system based on Groth [14] and Schnorr [12] schemes. The authors introduce a new signature scheme called *sibling signatures* which allows the users with a single key pair to use two different signature schemes. This single key pair can be used both during delegation and token presentation.

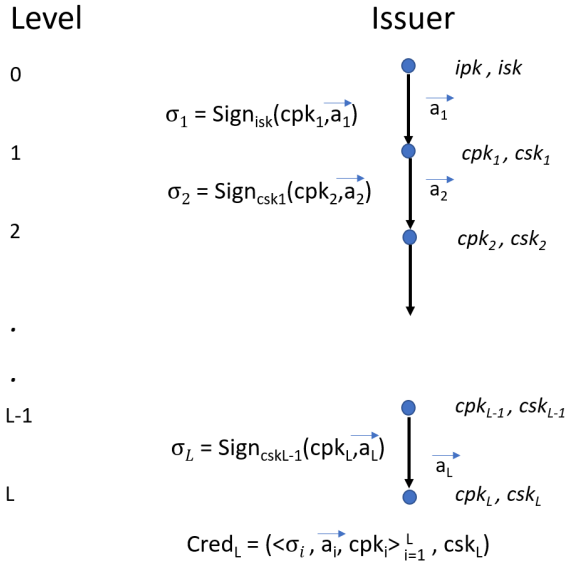

Fig. 2. Generic Construction of Camenisch *et al.* [9]

The author's generic construction scheme is depicted in Fig. 2. At Level-0 is the root issuer with public, private key pair as (ipk, isk). The root issuer delegates certain attributes $\vec{a_1}$ to

$user1$ at Level-1 with key pair as (cpk_1, csk_1). It generates a signature σ_1 on the combination of $(cpk_1, \overrightarrow{a_1})$ using isk. It passes on the signature and attributes to $user1$. Similarly, if $user1$ at Level-1 wants to delegate certain attributes $\overrightarrow{a_2}$ to $user2$ with key pair (cpk_2, csk_2), it generates a signature σ_2 using $csk1$ over $(cpk_2, \overrightarrow{a_2})$. $user2$ then forwards the σ_1, $\overrightarrow{a_1}$ and cpk_1 generated at Level-1 as well as σ_2, $\overrightarrow{a_2}$ generated at Level-2 to $user2$. This can go on till any Level-l credential. So a Level-l credential is a combination of all signatures, attributes and public keys of all $l-1$ levels and Level-l private key.

In order to anonymously present a credential at any level, the prover generates a NIZKP proof proving that he holds the entire credential chain (signatures, attributes and public keys at each level). He can selectively reveal attributes at each level. The public keys (and thereby the identity of users) at each level will remain hidden. The verifier verifies the token using just the root issuers public key ipk. The authors provided an instantiation of their construction scheme using Bilinear Pairings and Groth-Schnorr sibling signatures. They use Groth1 to denote Groth signatures on messages in $G1$ with public key in $G2$ (called SibGS1) and Groth2 to denote Groth signatures on messages in $G2$ with public key in $G1$ (called SibGS2). The final attribute token would then look like

$$
\begin{aligned}
at \leftarrow NIZK\{(\sigma_1, ..., \sigma_L, cpk_1, ..., cpk_L, < a_{i,j} >_{i \notin D}, tag) : \\
\bigwedge_{i=1,3,..} 1 = SibGS1.Verify_1(cpk_{i-1}, \sigma_i, cpk_i, a_{i,1}, ...a_{i,n_i}) \\
\bigwedge_{1=2,4,..} 1 = SibGS1.Verify_1(cpk_{i-1}, \sigma_i, cpk_i, a_{i,1}, ...a_{i,n_i}) \\
\wedge 1 = SibGSb.Verify_2(cpk_L, tag, m)\} \quad (1)
\end{aligned}
$$

In order to prove that the prover holds the private key corresponding to the public key at Level-L(cpk_L), the NIZK proof also includes the proof for a message m signed with the private key at Level-L (csk_L). In the concrete instantiation, this proof of private key possession is later turned into Schnorr signature proof of knowledge using Fiat-Shamir heuristic.

1) Credential Revocation: One of the important functionalities of any credential based system is to account for revoked credentials. Revocation allows the administrator to stop rouge elements from entering into the system. The authors in [9] left out credential revocation as a future item. In this section we discuss how credential revocation can be achieved by extending the author's concrete instantiation.

As mentioned in the previous section, if a user at level K-1 wants to delegate certain attributes $(\overrightarrow{a_K})$ to another user, he signs the message $(cpk_K, \overrightarrow{a_K})$ with csk_{K-1}. We propose to extend the signature message to include a Hash of $(cpk_K, \overrightarrow{a_K})$. We call this hash as the credential hash at level K. Therefore,

$$h_K = Hash(cpk_K, \overrightarrow{a_K})$$

$$\sigma_K = Sign_{csk_{K-1}}(cpk_K, h_K, \overrightarrow{a_K})$$

Hash in this case can be one of the *Secure Hash Algorithms* in the SHA-2 family. The credential hash is a unique representation of the public key and the delegated attributes at any level. Since the signature includes the credential hash, signature verification would fail if the hash is tampered. The delegator forwards $\sigma_K, h_K, \overrightarrow{a_K}$ to the delegatee. h_K forms a part of credential at every level. The attribute token generated for presentation would now look like (changes in bold)

$$
\begin{aligned}
at \leftarrow NIZK\{(\sigma_1, ..., \sigma_L, cpk_1, ..., cpk_L, < a_{i,j} >_{i \notin D}, tag) : \\
\bigwedge_{i=1,3,..} 1 = SibGS1.Verify_1(cpk_{i-1}, \boldsymbol{h_i}, \sigma_i, cpk_i, a_{i,1}, ...a_{i,n_i}) \\
\bigwedge_{1=2,4,..} 1 = SibGS1.Verify_1(cpk_{i-1}, \boldsymbol{h_i}, \sigma_i, cpk_i, a_{i,1}, ...a_{i,n_i}) \\
\wedge 1 = SibGSb.Verify_2(cpk_L, tag, m)\} \quad (2)
\end{aligned}
$$

When generating the attribute token, unlike other attributes, the credential hash value is always revealed to the verifier. There is no danger of someone modifying the hash as that would result in signature verification failure. If the root issuer or any delegator in the chain needs to revoke a credential, he would publish the credential hash in a Black List (BL) of credential hashes. The verifier has to check the credential hash for each level against the BL and if there is a match, verification would fail.

The concret instantiation of the NIZK for the attribute token would now look like (changes in bold),

$$
\begin{aligned}
SPK\{(< s'_i, t'_{i,j} >_{i=1,..L, j=1,..n_i}, cpk_1, ..., cpk_L, < a_{i,j} >_{i \notin D}, tag) : \\
\bigwedge_{i=1,3,..} (e(y_{1,1}, g_2)[e(g_1, ipk)]_{i=1} = e(s'_i, r'_i)[e(g_1^{-1}, cpk_{i-1})]_{i \neq 1} \wedge \\
1_{G_t}[e(y_{1,1}, ipk)]_{i=1} = e(t'_{i,1}, r'_i)[e(cpk_i, g_2^{-1})]_{i \neq L}[e(g_1, g_2^{-1})^{csk_i}]_{i=L}[e(y_{1,1}^{-1}, cpk_{i-1})]_{i \neq 1} \wedge \\
\boldsymbol{1_{G_t}[e(y_{1,2}, ipk)]_{i=1} = e(t'_{i,2}, r'_i)e(h_i, g_2^{-1})[e(y_{1,2}^{-1}, cpk_{i-1})]_{i \neq 1}} \wedge \\
\bigwedge_{j:(i,j) \in D} e(a_{i,j}, g_2)[e(y_{1,j+2}, ipk)]_{i=1} = e(t'_{i,j+2}, r'_i)[e(y_{1,j+2}^{-1}, cpk_{i-1})]_{i \neq 1} \wedge \\
\bigwedge_{j:(i,j) \notin D} 1_{G_t}[e(y_{1,j+2}, ipk)]_{i=1} = e(t'_{i,j+2}, r'_i)e(a_{i,j}, g_2^{-1})[e(y_{1,j+2}^{-1}, cpk_{i-1})]_{i \neq 1}) \wedge
\end{aligned}
$$

$$
\begin{aligned}
\bigwedge_{i=2,4,..} (e(g_1, y_{2,1}) = e(r'_i, s'_i)e(cpk_{i-1}, g_2^{-1}) \wedge 1_{G_t} = e(r'_i, t'_{i,1})[e(g_1^{-1}, cpk_i)]_{i \neq L}[e(g_1^{-1}, g_2)^{csk_i}]_{i=L}e(cpk_{i-1}, y_{2,1}^{-1}) \wedge \\
\boldsymbol{1_{G_t} = e(r'_i, t'_{i,2})e(g_1^{-1}, h_i)e(cpk_{i-1}, y_{2,2}^{-1})} \wedge \\
\bigwedge_{j:(i,j) \in D} e(g_1, a_{i,j}) = e(r'_i, t'_{i,j+2})e(cpk_{i-1}, y_{2,j+2}^{-1}) \wedge \bigwedge_{j:(i,j) \notin D} 1_{G_t} = e(r'_i, t'_{i,j+2})e(g_1^{-1}, a_{i,j})e(cpk_{i-1}, y_{2,j+2}^{-1}))\}(sp, r'_1, ..., r'_L, m)
\end{aligned}
$$

2) Session Key Establishment: In the IoT scenario, once the token generated by the IoT device/User is verified by the IoT service, the next logical step would be to exchange data. In order to securely perform data exchange, both the endpoints should establish a common secret key. In the earlier section, we talked about how the authors use signature proof of knowledge over message m to prove the possession of the Level-L private key. The proof then acts as a Digital Signature over m. Instead of any random message, the prover can generate an ephemeral public, private key pair (pk, sk) and use the public key pk as the message over which the signature proof is performed. This public key is also sent along with the attribute token to the verifier. Once the token verification succeeds, the verifier generates a random secret key and uses the prover's ephemeral public key to encrypt it. It then sends the encrypted secret key over to the prover which then decrypts it using the private key sk. Any attempt by a Man-In-The-Middle to modify the public key sent to the verifier would result in the token verification failure. The scheme assumes that the response from the verifier is not tampered. Additional measures will have to be taken if verifier's identity and its response have to be validated at the prover.

V. The Proposal

In this section we discuss in detail the architectural components of our proposal and the messages exchanged between them. One of the primary requirements of the system was to provide a light weight solution for IoT devices and users to anonymously access IoT services.

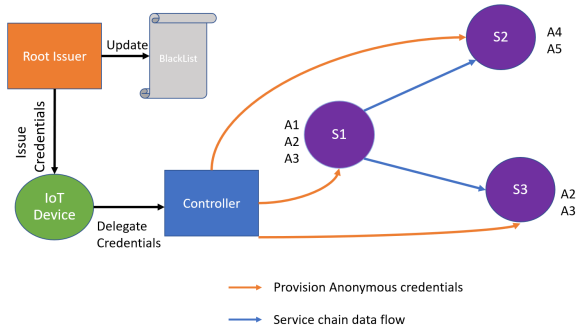

Fig. 3. System Architecture

Fig. 3 illustrates the system architecture that has the Root Issuer issuing credentials to IoT devices and users. The device delegates its credentials to the Controller when it detects an event. The Controller facilitates the generation and propagation of the anonymous credential when ever a Service requests for it. Here S1, S2, S3 etc are the various services that get invoked as part of the service chain with A1,A2,A3 etc being the attributes required by each service for its invocation. The following sections discuss each component in detail.

1) The Root Issuer: As the name implies, the Root Issuer (RI) issues Level-1 credentials to users/IoT devices. The RI service maintains a public, private key pair (ipk, isk) using which it signs the Level-1 credentials. The public key ipk is distributed across the system so that services can validate credential tokens presented from any delegation level just by using ipk.

The RI generates the system public parameters $G1, G2, g1, g2$ and the bilinear map e when it is initialized for the very first time.

All users/IoT devices are assumed to hold one long term public, private key pair. Any user/IoT device which wishes to obtain credentials from RI, submits the credential request containing their public key and the attributes that they wish to possess. RI then validates the entity's request and generates the credential. As mentioned in section IV-1, RI signs the combination of user/device's public key, the credential hash and the attributes, with its private key isk. The credential hash is the hash of the user's publick key and all the attributes that the RI wishes to assign to the entity. The entity's credential would then be a combination of the signature, the credential hash, the attributes and its public, private key pair.

If the root issuer wishes to revoke an entity's credential, it would publish the credential hash for that entity in a BlackList (BL) that is available for everyone in the system to view. Only the RI can add, delete or modify the entries in the BL. Since the credential hash is hashed over public key and attributes, no one can infer any useful information about the entity just by looking at the hash.

2) The Controller: The Controller forms the core of the system. One of the primary reasons for choosing a centralized controller based approach is to allow the flexibility for both constrained devices as well as resourceful endpoints to anonymously authenticate and authorize against services. Even on the services end, constrained IoT services for example a smart bulb do not have the resources to validate an anonymous credential. With our approach, we allow both less constrained and very constrained endpoints to participate in the system. Some of the core functionalities of the Controller are

a) Accepts Delegated Credentials from Users/IoT devices and generates Anonymous Credentials to be presented to services in the IoT service chain.
b) Maintains the list of system generated events and the corresponding services to be invoked.
c) Maintains session information like services that were invoked, what attributes are needed for each service and so on for the entire service chain.
d) Validates the User/Device credential against BlackListed credentials.

On startup, the Controller loads the system parameters, generates public and private keys if this is the first time it is starting or loads the stored public and private keys. The Controller maintains a list of all the attributes required by each service in the system. This allows the Controller to generate anonymous credentials on behalf of the user/device by selectively revealing only the attributes required by the service and hiding the remaining ones.

IoT devices/Users generate events when there is a change in normal behaviour. For example, the HMS detecting a low pulse rate for the patient, an intruder trying to enter the house, component breakdown in a shop floor etc. All these activities generate system events which are processed by the Controller. The Controller maintains a list of the services that need to

be invoked for each event. For example, if a Guest enters the Smart Home, access to only the Guest room needs to be allowed, the Guest preferred lighting and temperature needs to be set and so on. Once an event is detected, the Controller invokes the corresponding services maintained in its list. Each service may in turn invoke other services in a chained manner. Each service in the chain validates if the user/device that generated the event has the required attributes to fulfil the service request. This continues till all the services in the chain are invoked. The controller also maintains a list of invoked services in the chain so as to avoid repeated invocations of the same service.

To elaborate more on the technical details, when a User/IoT device detects an event, it informs the Controller of the event. As part of the signalling mechanism, the IoT device delegates its credentials to the Controller. The device delegates all its attributes to the Controller. See section V-3. On receiving the delegated credential and the event, the Controller first verifies the delegated credential using the Root Issuer's public key ipk. It also checks if the user's credentials are black listed or not. If the delegated credential is fine, it would then retrieve the list of services that need to be invoked for the event. For each such service, the Controller generates an anonymous credential based on the attributes that were delegated to the Controller by the user/device. The Controller selectively reveals only those attributes that are required for the service to process the request and hides the remaining attributes. The generated attribute token as mentioned in section IV-1 will then be presented to the service for verification.

Here we have certain design options on the Controller based on whether the invoked service is constrained or not. A very constrained service, for example a Temperature Sensor would not have the processing capability to verify the attribute token generated by the Controller. Where as a Smart TV or a Smart Fridge may be able to verify the attribute token. In addition to the token verification, the service may choose to implement a policy evaluation based on the user/device attributes received in the token. See section V-4. Even here, constrained services may not be able to evaluate any policies based on the attributes in the token. The following are the design options.

a) The Controller identifies the service as not a constrained one and would let the service perform the attribute token verification and policy evaluation.
b) The Controller identifies the service as constrained and would in turn evaluate if the user/device delegated credential has the required attributes to fulfil the service request. Here it skips the token generation. The Controller also evaluates the policies, if any, defined for the service. The result of policy evaluation could in turn invoke the next set of services. See section. V-4. If so, the controller directly invokes the next service in the chain.

We evaluate the performance of the above options in section. VII.

The Controller generates a unique SessionID for every event generated by the system. The SessionID remains the same for all services in the chain. As mentioned above, after the Service performs the token verification and policy evaluation, it signals the next service in the chain. It passes the SessionID as part of the signalling mechanism. The next service will then reach out to the Controller for the anonymous credential using the SessionID. The Controller will then look for the desired attributes of the new service and generates an anonymous credential and passes it over to the service. This process continues till all the services in the chain are invoked.

So far we talked about each service in the chain requesting the Controller for the anonymous credential using the sessionID. Instead, the Controller can also push the anonymous credential to each service before the chain starts. This way, each service would have the credential that it has to verify and evaluate the policies. It need not wait for the chain propagation to reach it. All the services therefore can process the token in parallel thereby saving time. The number of messages that fly around are also reduced as now the service does not have to make an explicit request for anonymous credential. The drawback of such an approach is that the Controller would have to know the complete chain of services that need to be invoked for the event. For this, it needs to evaluate the policies at each and every service in the chain to know what the next service would be. Once it forms the complete chain, it starts distributing the anonymous credentials for each service using the SessionID.

3) The User/IoT device: Users or IoT devices generate system events. These events in turn need a series of actions to be performed by various entities in the system. The entities (also referred to as services) interact among themselves towards achieving a common goal intended for satisfying the event. Every entity that is invoked needs to check if the user/IoT device that generated the event has the necessary privileges to invoke it. It is also important from a privacy perspective that the service only knows as much information about user/device as is needed for its invocation.

Very constrained devices may not be able to authenticate/authorize themselves anonymously to each service in the chain. Less constrained devices may have the compute power to do so. In which case, they can directly interact with each service in the chain and provide the necessary credentials. In our design, we consider events generated from very constrained devices. Along with the event, the device generates a delegated credential to be issued to the Controller. The Controller is assumed to be a trusted service in the system and the device delegates all its attributes to the Controller. One of the primary reason for choosing [9] as an implementation base is that it provides a flexibility where the delegation process is not anonymous. The delegator reveals its identity (the public key) and all its attributes during delegation. It is only during presentation the prover will hide the public keys of each of the entities in the delegation hierarchy. The prover also selectively reveals only those attributes that are needed for the Verifier. This works well with our model as IoT devices cannot perform the costly anonymous credential delegation. Moreover all devices and users trust the Controller service and so the privacy aspect is not lost during delegation.

Every User/Device holds a public, private key pair and is issued a credential by the RI based on the attributes it possesses. That's the Level-1 credential as per [9]. When the device detects an event, it generates a Level-2 delegated

credential with the Controller's public key and all its attributes and passes it over to Controller. The Controller will then trigger the service chain using this delegated credential.

4) The Service: Services are the resource providers in the system. There could be very constrained services like a temperature sensor that reads and outputs the surrounding temperature and also less constrained services like a smart fridge that smartly manages the contents inside the refrigerator. An IoT device can also act as an IoT service. For example a Heart Monitoring System (HMS) provides the pulse rate of the patient as a service and it can also generate an event if the pulse goes beyond normal values. Any IoT service can have two basic functionalities, a sensing capability and an actuating capability. Anyone wanting to access the service may look to read (sense) or write (actuate) data from the service.

The service must ensure that anyone wanting to read or write to it has the necessary privileges for the operation. These privileges are granted by the system administrator to users/device within the system. In attribute based system, privileges are granted in terms of user/device attributes. For example, only devices with attribute "coolant" can invoke the refrigeration service. Rather than granting complete access, services can also implement policies that dictate the kind of service that is granted. For example, in home automation, a user having an attribute as "$age > 18yrs$" has access to all the channels in a television. Users who do not possess this attribute are allowed access to only a limited set of channels. Each service can in turn communicate with other services for getting the work done. Service policies may also dictate the set of subsequent services that should be invoked based on the attributes that the user/device possesses. With privacy aware services, the Controller or User only reveals those attributes that are needed by the service and not all of them.

On start-up, each service loads the system parameters including the RI's public key ipk. It also loads the service policies and waits for incoming requests. When there is a service request made (from the Controller or from a user/device directly), the endpoint passes an anonymous credential along with a unique SessionID. The service has no way to know the identity of the presenter from the anonymous credential. The presenter reveals attributes that are common to itself and the service. Rest of the presenter attributes, including its public key are hidden. The service verifies the anonymous credential using the RIs public key ipk. It also makes sure the credential hash from the credential is not present in the BlackList published by the RI. Once that is done, it evaluates the policy based on the attributes the user/device has revealed. Policy evaluation may result in invoking other services as well. If so, the service invokes the next service in the chain passing along the received SessionID. The next service will in turn query the original endpoint (Controller or User) using the same SessionID. The endpoint will then generate a new anonymous credential for this new service and pass it along. This continues till the complete chain is invoked.

For very constrained services which do not possess the compute power to verify an anonymous credential and evaluate policies, the Controller can perform both these operations at its end and pass on the results to the service. See section V-2.

5) Message Model: Figure 4 shows the messages that flow between Controller and the Services in Hybrid Mode. When an IoT device or User reports an event (1), Controller invokes a set of corresponding high level services by generating an attribute token and sending a SERVICE_REQUEST (2) for each of the services. The service verifies the token and evaluates its policy. Based on the access policy it replies back to Controller with either SUCCESS or FAILURE message (3). As a result of the policy evaluation, the service may in turn invoke other services by sending a SERVICE_CHAIN_REQUEST (4). When a service receives such a request, it relays the request back to the Controller (5). If the service is constrained, Controller skips token generation and evaluates the corresponding service policy. The result of policy evaluation is then conveyed back to the service via CONSTRAINED_SERVICE_RESPONSE (6). If the chain request is coming from a resourceful service, Controller generates an attribute token and makes a SERVICE_REQUEST (2). This continues till the entire chain completes execution.

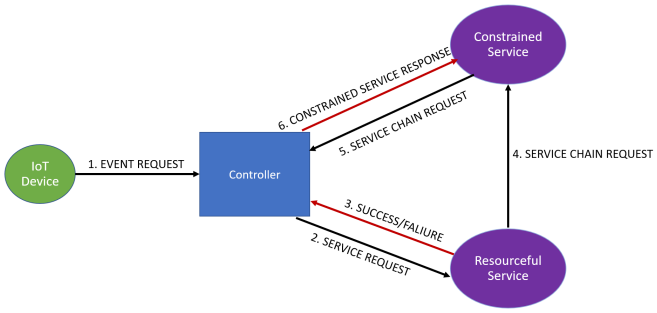

Fig. 4. DISC Messages

VI. Security Evaluation

Some of the security-related properties of the scheme are discussed below.

1) Nonlinkability: One of the primary requirement for implementing a privacy preserving scheme is to ensure that an attacker does not gain any information from two different events where the events maybe related or unrelated. Once the user/device delegates all its attributes, the Controller generates anonymous credentials for every service in the chain. Each service looks for a set of attributes in the anonymous credential. So the Controller has to reveal those attributes (if the user possess them) to the service. As mentioned in the V-4 section, once a service completes the token verification and policy evaluation, it invokes the next service in the chain by passing the SessionID that it received from Controller. the next service goes to the Controller with this SessionID to get the new anonymous credential. This continues for all the services in the chain.

The anonymous credential for each service reveals only the required attributes about the user/device. If the communication between Controller and Services happens in clear, a passive attacker can look at all the credentials for a specific SessionID

and link them to a common user/device, thereby compromising the user/device's privacy. One way to thwart such an attack is to establish a secure channel over which the anonymous credentials are passed. With this, none of the attributes are revealed to a passive on-looker. A secure channel may not be feasible sometimes especially for constrained service. As another approach, the Controller and the Service can establish a session key as defined in Section IV-2. Controller and Service can secretly exchange a random next-SessionID using the established session key. The service then uses the next-SessionID when invoking the next service in the chain. Thereby when the subsequent service reaches out to the Controller using next-SessionID, Controller knows which credential to use. If the messages flying through the system are dense, an attacker cannot correlate the SessionID and next-SessionID as belonging to the same user/device.

An active attacker who can mount physical attacks on services can easily correlate the user attributes gleaned from one or more services in the chain.

2) Device/User Privacy: One of the primary goal of DISC is to ensure the privacy of users and devices is maintained. The attribute based system allows devices to prove that they hold certain attributes (issued by the RI) without revealing their complete identity. The device can reveal only a subset of attributes that it holds to the service. And the service has no way of linking the attributes to a specific user/device as those attributes can be possessed and presented by any entity in the system.

3) Credential Revocation:

4) Centralized Controller: It can be a single point of failure

VII. Implementation and Simulation Results

In this section we evaluate the performance of DISC under two different use cases, one of Home Automation and the other an Industrial Thermal Power plant. Home Automation System typically has few tens of IoT devices with applications ranging from Lighting Control, HVAC, Home Security, Power/Water Metering, entertainment etc. Service chains on any event, say a Guest entering the home would typically be less than thirty in length and are managed in a single administrative domain. Ensuring privacy of the users or devices which act on behalf of the users is very important to control any misuse of personal information. On the other hand a shop floor in a manufacturing industry or a nuclear/thermal power plant will have hundreds of sensors and actuators monitoring various critical parameters. These include monitoring the Generator parameters, Oil/Water cooling system, Power Distribution Systems, Fire Fighting Systems and so on. When a critical event happens, for example, the turbine bearing temperature going high, there would be hundreds of services that are put into operation. These include turning off the generator and turbine, gradually increasing the coolant flow, keeping the emergency services in standby etc. A fire in a shop floor is another example which invokes a large number of services. The service chains in such scenarios easily run into hundreds services. Note that these services may not all be managed in a single domain. The services coming under turbine control will be managed differently than those under generator control. The manufacturers of the sensors and actuators may also be from different vendors.

The simulation environment consists of a Controller running on a Intel(R) Xeon(R) CPU E5-2640 v2 @ 2.00GHz with 32 cores and 128GB RAM. The Home automation set-up consists of 100 services running on three Intel(R) Xeon(R) CPU E5-2420 0 @ 1.90GHz machines with 24 cores and 32GB RAM as docker containers. Docker commands allows us to specify the hard and soft limits on CPU and RAM within which the container operates. We manage these limits to simulate constrained and less constrained services. All machines are running The power plant set-up consists of running 500 services again distributed across three different machines and running as docker containers. The Root Issuer (RI) service is running on a 1GB RAM,1 CPU 2.4 Ghz Intel(R) Xeon(R) CPU processor. All machines are running Ubuntu 18.04.x LTS.

All components are implemented in C language. We used the Pairing Based Cryptography (PBC) Library from [10] for the pairing-based operations. PBC library is built on GMP Library [15] for mathematical operations. We use the Type A curve from PBC [16].

1) Home Automation: As discussed in section V-2, Controller can operate in the following modes.

a) **Decentralized Mode** is where the Controller only generates the anonymous credentials. Services in turn verify the credential token that they receive from Controller. Service policy evaluation also happens at the service. This mode becomes relevant when most or all the services in the domain are resourceful and have sufficient processing power for token verification and policy evaluation.

b) **Centralized Mode** is where both user/device credential validation and service policy evaluation happen at the Controller. This mode is useful when the services in the domain are very constrained in resources.

c) **Hybrid Mode** is for a mixed scenario where there are both resourceful and constrained services in the domain. The Controller has a list of what services fall under which category and therefore makes a decision whether to generate a token or validate the device credential by itself. If the service is a constrained one, the Controller evaluates the service policy and returns to the service a list of next services to be invoked.

Figure. 5 shows the total time taken for each of the Controller modes for a service chain of 50 services. This is the typically the maximum length of chain for Home Automation. The horizontal axis shows the number of attributes the user/device possesses. As can be seen from the figure, the more the number of attributes the device holds, the greater will be the time taken for chain completion. This is because the token generation and verification depend on the number of attributes the device holds. Also note that the Decentralized mode of operation takes the maximum time to complete as the Controller would generate the attribute token for each service (based on the attributes that the service requires and the attributes that the user possesses)in the chain. In the Centralized mode, there is no token generation and propagation. Policy evaluation also happens at Controller. It

took around 50 milliseconds for the chain to complete. If the number of service is too huge, this mode becomes a bottle neck for the system. Time taken for Hybrid mode as the figure shows, falls in between Centralized and Decentralized values.

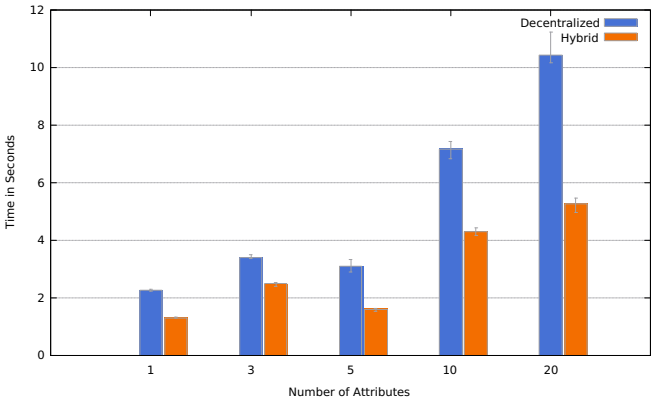

Fig. 5. Controller Modes of Operation - Chain of 50 services

2) Nuclear/Thermal Power Plant: A power generation plant will typically have thousands of constrained and less constrained IoT devices and services running. These devices monitor, report and control various critical parameters of the plant. In case of a critical event, for example bearing temperature going high, a series of services have to be deployed within no time. The chain of services that get invoked can run into hundreds in some cases. For such critical events, latency and reliability in terms of service invocations are crucial parameters which the system has to ensure. We ran simulations to capture the time and resources it takes to invoke a service chain of length 400 that we typically see in power plants. The Centralized mode took around 100 milliseconds to complete and is not shown in the graph.

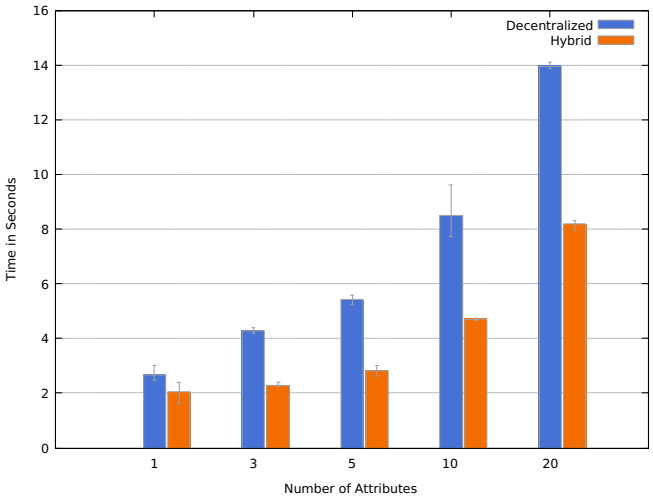

Fig. 6. Controller Modes of Operation - Chain of 400 services

As can be seen in figure 6, the time it takes to complete the chain is relatively longer when compared to the Home Automation case. As the number of services increase, chain propagation time also increases. There are more services for which the Controller has to generate anonymous credentials and the number of service verifications has also increased. Centralized mode continues to take least time of all the three modes with Hybrid mode performing better than Decentralized mode.

We further monitor the time it takes for each of the DISC operations. Credentials are issued by the Root Issuer (RI) to users and devices. Each user/device is assigned certain attributes based on who the user is and to what group it belongs. Credential Delegation happens when a constrained device detects an event and informs the Controller of the event. The device delegates the credentials that it received from the RI. Controller then initiates the service chain by presenting an anonymous credential token to each service in the chain. The service in turn verifies the token forwarded by the Controller.

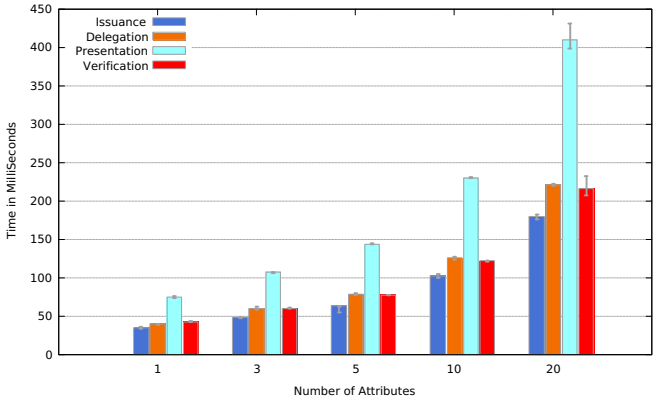

Fig. 7. DISC operations

As can be seen in figure 7, as the number of attributes the user/device possesses increases, the time taken by each of the DISC operations also increases.

The time it takes for attribute token verification depends on the resources that the service possesses. IoT services such as a smart bulb or a temperature sensor are be quite constrained in terms of resources. Whereas a smart TV or a Smart Fridge has sufficient resources for verifying the token. In Figure. 8 we determine the time it takes for a service to verify an anonymous attribute token generated by the Controller. The attribute token was generated for a device with 20 attributes.

In our environment we run services within Docker containers. We use the official Docker Ubuntu image [17] for running DISC services. Docker commands allow us to specify the maximum limits on CPU and Memory that the processes within the container can use [18]. By varying these limits, we were able to simulate various types of constrained and resourceful services. As we can see in Figure. 8, the more constrained the service is, the more time it takes to verify the attribute token. CPUS in the figure indicate how much of the CPU resources a container can use. With CPUS=0.2, the container is guaranteed at most one-fifth of a single CPU.

The RI periodically publishes a list of all BlackListed (BL) Credentials. Services should verify if a credential is BlackListed before granting the service. Entries in the BL reveal nothing about the user whose credential is black listed. An entry in BL is a hash of the credential attributes issued to

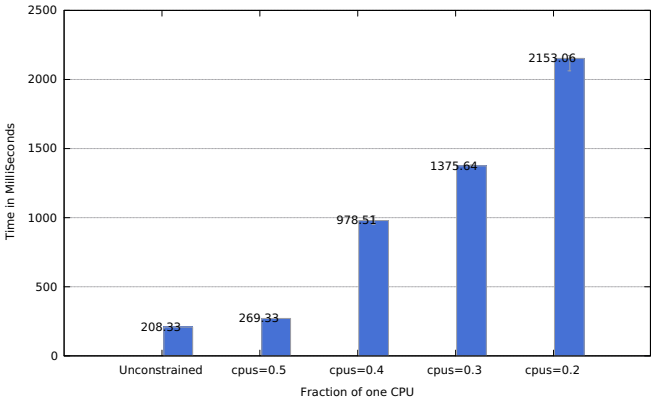

Fig. 8. Token Verification

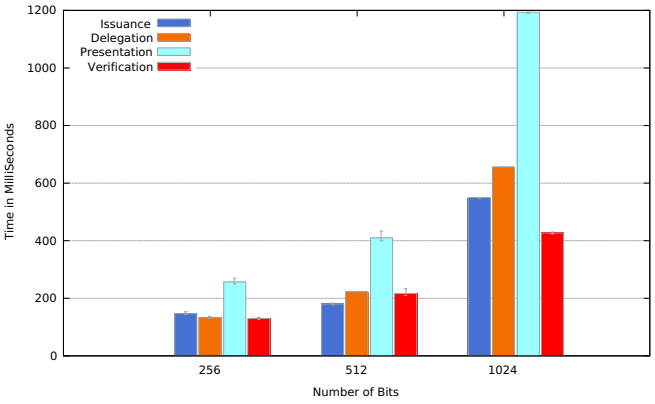

Fig. 10. Time taken for different Base Field sizes

the user. The hash is then encode in Base64 format and stored in BL. Number of entries in the BL can be huge for domains consisting of huge number of devices and users, for example a shop floor in a manufacturing industry.

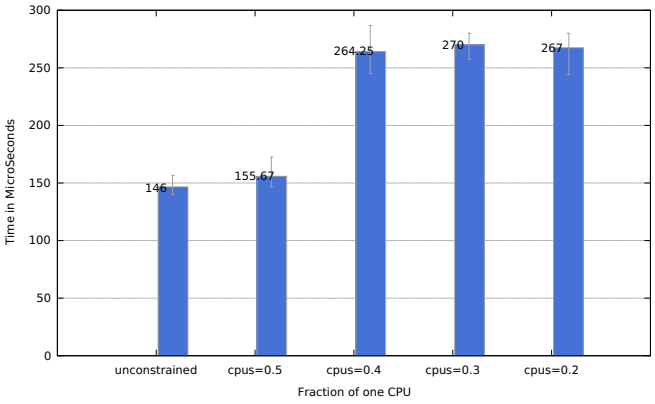

Fig. 9. BlackList Verification

Figure. 9 shows the time taken by a service to validate an attribute token against the BL. Although the number of attributes in an attribute token doesn't matter (credential hash is compared against the BL), the token used for experiment had 20 attributes. The experiment was conducted with a BL containing 100,000 entries. A point to note is that, the RI sorts the entries in the BL whenever it updates the BL. The service would use the Binary Search algorithm to quickly find out if the credential hash is present in the list or not. This saves time and resources when compared to a linear search. As the results show, BL validation for an unconstrained service takes much lesser time compared to a constrained service.

The Type A Curve [16] has a base field size of 512 bits with a Discrete log security of 1024 bits therefore providing an overall security of 80 bits. In Figure. 10 we determine the time taken by each of the DISC operations for a device with 20 attributes with varying Base field sizes. As the field size increases, resulting in increased security, the underlying cryptographic operations take more time to complete.

VIII. Conclusion

The conclusion goes here.

References

[1] Ala I. Al-Fuqaha, Mohsen Guizani, Mahdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys and Tutorials*, 17:2347–2376, 2015.

[2] D. Evans. The internet of things-how the next evolution of the internet is changing everything. April 2011. http://www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf3, last accessed on 01/01/20.

[3] Ioannis Andrea, Chrysostomos Chrysostomou, and George Hadjichristofi. Internet of things: Security vulnerabilities and challenges. pages 180–187, 07 2015.

[4] Jan Henrik Ziegeldorf, Óscar García-Morchón, and Klaus Wehrle. Privacy in the internet of things: threats and challenges. *Security and Communication Networks*, 7:2728–2742, 2014.

[5] David Chaum. Blind signatures for untraceable payments. In D. Chaum, R.L. Rivest, and A.T. Sherman, editors, *Advances in Cryptology Proceedings of Crypto 82*, pages 199–203, 1983.

[6] Jan Camenisch and Els Van Herreweghen. Design and implementation of the idemix anonymous credential system. In Vijayalakshmi Atluri, editor, *ACM Conference on Computer and Communications Security*, pages 21–30. ACM, 2002.

[7] Greg Zaverucha Christian Paquin. U-prove cryptographic specification v1.1, revision 3. December 2013. https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/U-Prove20Cryptographic20Specification20V1.1.pdf, last accessed on 01/01/20.

[8] Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. Randomizable proofs and delegatable anonymous credentials. In Shai Halevi, editor, *Advances in Cryptology - CRYPTO 2009*, pages 108–125, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[9] Jan Camenisch, Manu Drijvers, and Maria Dubovitskaya. Practical uc-secure delegatable credentials with attributes and their application to blockchain. In *Proceedings of*

the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017, pages 683–699, 2017.

[10] Ben Lynn. The Pairing-Based Cryptography Library. https://crypto.stanford.edu/pbc/.

[11] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[12] Ed F. Hao. Schnorr non-interactive zero-knowledge proof. RFC 8235, RFC Editor, 09 2017.

[13] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, 1986.

[14] Jens Groth. Efficient fully structure-preserving signatures for large messages. *IACR Cryptology ePrint Archive*, 2015:824, 2015.

[15] The GNU Multiple Precision Arithmetic Library. https://gmplib.org/.

[16] PBC Type A ECC Curve. https://crypto.stanford.edu/pbc/manual/ch08s03.html. Accessed: 2020-03-17.

[17] Docker ubuntu image. https://hub.docker.com/_/ubuntu.

[18] Docker runtime options with memory, cpus, and gpus. https://docs.docker.com/config/containers/resource_constraints/. Accessed: 2020-03-08.