University of Twente

Department of Electrical Engineering, Mathematics and Computer Science (EEMCS)
Services, Cybersecurity & Safety (SCS)

Master Thesis

Cryptographic Implementation of Issuer Policy for Self Sovereign Identity Systems

Naveenaa Anaigoundanpudur Karthikeyan

Committee Chair Prof.Dr. Andreas Peter
Faculty of EEMCS and Services, Cybersecurity & Safety (SCS)
University of Twente

Supervisor Dr.Ing.Florian Hahn
Faculty of EEMCS and Services, Cybersecurity & Safety (SCS)
University of Twente

Committee Member (External) Dr. Ralph Holz
Faculty of EEMCS and Design and Analysis of Communication Systems (DACS)
University of Twente

Supervisors at TNO Rieks Joosten and Sterre Breeijen

October 19, 2021

UNIVERSITY OF TWENTE.

TNO

Cryptographic Implementation of Issuer Policy for Self Sovereign Identity Systems

Naveenaa Anaigoundanpudur Karthikeyan
Faculty of Electrical Engineering, Mathematics and Computer Science
University of Twente
Enschede, Netherlands
n.a.karthikeyan@student.utwente.nl

***Abstract*—In Self-Sovereign Identity (SSI) there are three entities involved, namely issuer (issues the credentials), holder (for whom the credentials are issued), and verifier (the one who needs to view the credentials to provide a service or commodity in exchange). The problem here is that the verifier might request more than the required credentials from the holder. The holder is put into a difficult situation where the holder must give all the requested credentials in order to avail of the service offered by the verifier. To stop this from happening policies must be put into place and these policies must be cryptographically enforced. Various potential solutions are suggested and from those solutions, Ciphertext Policy Attribute-Based Encryption (CPABE) is used to address this problem. Implementation is provided in the form of a demo and the performance of the implemented solution is measured.**

***Index Terms*—Self-Sovereign Identity (SSI), Issuer Policy, Cipher-text Attribute Based Encryption (CPABE)**

I. Introduction

As the world is rapidly digitising, so are our online identities. Since there is no globally agreed standard identification for identities on the internet, each digital entity has decided to have its own way of recognising individuals and organisations using their own custom services. The entities then started providing their own username and password to identify individuals and organisation. This has led to the same individual having multiple different online personas individual and organisation. Maintaining and accounting for these multiple online personas has become a problem for both service providers and users alike. A new concept called Self-Sovereign Identity (SSI) has emerged that can solve the problem of multiple digital identities.

Before looking at what SSI is let us understand how SSI came into being and the evolution of digital identity. As described by Christopher Allen in his comprehensive article "The Path to Self-Sovereign Identity[4]" there are four stages in the evolution of digital identity namely, Centralised identity, Federated/Multiple centralised identity, User-centric identity and Self-Sovereign identity[4]. Initially in the internet both the issuer and verifier of digital identities were the same entity, thus creating a centralised architecture along with a bit of hierarchy in identity management. This was followed by authorities of multiple and federated identity management. The two identity systems were not user centric, instead the authority lied with the issuer and verifier thus leaving the user vulnerable to the decisions made by the before party. This is when user-centric identities came into the picture. User-centric identities were created with the intention of the user being in control of their own identity and for the system to be more decentralised. This allowed people to provide their information at their will for other services online, for example, for services such as OpenID[36], OpenID Connect[18] and OAuth[13]. User-centric identity is once again centralised due to the fact that the identities generated by certain authority is not transportable as of when and where required. SSI aims to provide this transportable identity along with other notable features such as the user being in control of their own identity.

The internet has a wide range of stands on what SSI is and how SSI should be created and used. The standard of SSI that shall be considered in this research is as follows. There will three entities namely issuer, holder, and verifier. These three notations are roles that can be played interchangeably by organisation and individuals depending on the situation, meaning that the entity that acts as issuer for a given scenario can act as the holder or verifier for a different scenario. The issuer will present credentials to the holder. The holder will present their credentials to the verifier. The verifier views the credential of the holder in exchange for service or information provided it provides when the verifier requests for the same. Since the holder is required to provide it's own credentials to the verifier in exchange for service or information from the verifier, the verifier might misuse their stand and request for more than required information from the holder. In other cases the holder might not want to disclose too much information about themselves to the verifier or the issuer of credentials will only want certain verifiers to view the credentials they have issued for privacy reasons. To make sure such actions do not take place policies have to be enforced. Till now within the SSI systems issuing of policies over the SSI credentials have not been thought of. This paper explores how the policies can be cryptographically enforced for SSI systems. In this paper the policies for SSI credentials will be issued by the issuer of SSI credentials and henceforth be called as issuer policy. This is so as to make the working of the research easy and straight

forward. The policy can be in the future be enforced by the holder if required. With the issuer policies in place the requests made by the verifier to view the holder's SSI credentials will be cross checked with set of issuer polices. If the verifier satisfies the issuer policy then the holder's SSI credentials can be viewed by the verifier. If the verifier does not satisfy the issuer policy then the holder's SSI credentials can not be viewed by the verifier. The issuer policy will be implemented using Ciphertext-Attribute Based Encryption (CP-ABE).

This paper is structured as follows. Section II describes about the related work and Self Sovereign Identity(SSI) in depth and states the research problem for this thesis. Section III gives a summary of the potential solutions proposed for this problem. Section IV describes the issuer policy in detail and explains which scenarios or attributes can or can not be implemented. Section V describes how Ciphertext-Attribute Based Encryption (CP-ABE) could be used in SSI and how it can be implemented with a use case scenario. Section VI explains the environmental setup in which the codes were coded and the analysis was done. It also explains the results that were observed based on the implementation of the code depending on various parameters. Section VII describes the future works and section VIII is the conclusion.

II. Background

There are different Self Sovereign Identity (SSI) systems out there at the moment and some of them are discussed in subsection related work. Most of the SSI systems follow the W3C Verifiable claims[30] standard format and their identifiers are of the Decentralised Identifiers (DIDs) [8]. General structure of Self Soverign Identity Systems are summarised in subsection Self Sovereign Identity System.

A. Related Work

The digital identities have evolved over time as described by Christopher Allen in "The Path to Self-Sovereign Identity[4]". He describes that any identity system should have the balance of transparency, fairness, and support of the commons with protection for the individual[4]. [4] describes how SSI came into being and what is SSI and the 10 principles of SSI.

There are various SSI systems namely Sovrin, IRMA, uPort, ShoCard, Blockstack and many more. [20] elaborates on Sovrin, IRMA, uPort, ShoCard, Blockstack and evaluates and compares each of these systems one by one and mentions that all these systems have not been able to truly achieve all the parameters needed for in an SSI. In [17] IRMA and Sovrin are compared in detail. Sovrin, uPort, ShoCard, Blockstack work on blockchain technology. In Sovrin [29] anyone can be the issue credentials and anyone can be verify the credentials. It is built on Hyperledger Indy Project[14] and is an open source. uPort is is also an open source and works entirely dependent on Ethereum blockchain. ShoCard [25] works on Bitcoin blockchain. The identities are stored in the blockchain in the format of signed cryptographic hashes [20]. Blockstack [31] is a decentralised internet secured with the technology of blockchain, is not just used for identities but also for other services such as discoveries and storage [3]. IRMA [35] implements the Idemix attribute-based credential scheme and supports attribute-based signatures [33]. IRMA makes use of the concept of Zero-Knowledge Proof(ZKP) [11] to prove that a number satisfies a certain property without giving away what the number actually is[39].

These systems have tried to address the core values of SSI to an extent. The problem of the holder being vulnerable to give away too much information about oneself to the verifier have not been discussed or thought of so far in the current SSI systems. This problem will be addressed in this paper in the form of issuer policy over SSI credentials and implemented cryptographically using CP-ABE.

B. Self Sovereign Identity System

SSI as a new emerging technology will help in solving the modern-day digital identity problems faced due to the advancement of internet and network infrastructure. For example SSI will protect the privacy of the individual according to the GDPR regulations, Web-shops will no longer have to save critical personal information about its users. Other use cases include the elimination of passwords and falling victim to phishing attacks. SSI will be able to enforce trust on entities and make digital life more convenient for all of its users[22].

The common structure present in the systems are the 3 major roles namely the issuer, holder, and verifier. Entities participating in SSI could take up one of these roles at any given point in time and work interchangeably for different scenarios. For a particular scenario the roles are not interchangeable. The entity playing the roles of the issuer is the one issuing the credentials. Holder is the one for whom the credentials are issued for. Verifier is the one who requires the credentials to provide information/service. For example University of Twente will be an issuer when issuing the degree certificate to it's students. In another scenario when a prospect student applies to the university, the university now becomes the verifier for the prospect student's credentials. This way the entities take up these 3 roles alternatively depending on the situation.

The model of SSI flow presented in Figure 1 is based on the contents provided in Verifiable Credentials Flavors explained[38].

Step 1: The process is done within the issuer side. The issuer issues the verifiable credential (VC).

Step 2: The holder makes a connection with the issuer in order for the issuer to share the credentials with the holder. The connection will be made with the help of identifiers.

Step 3: The issuer now prepares the VC asked by the holder.

Step 4: The issuer and the holder connect with each other via the help of identifiers to pass the VC from the issuer to the holder. The VC will be now present in the holder's digital wallet.

Step 5: The verifier requests to the holder on what type of credential is required. This is known as presentation exchange. At the time of writing this paper this exchange method was still in development stage.

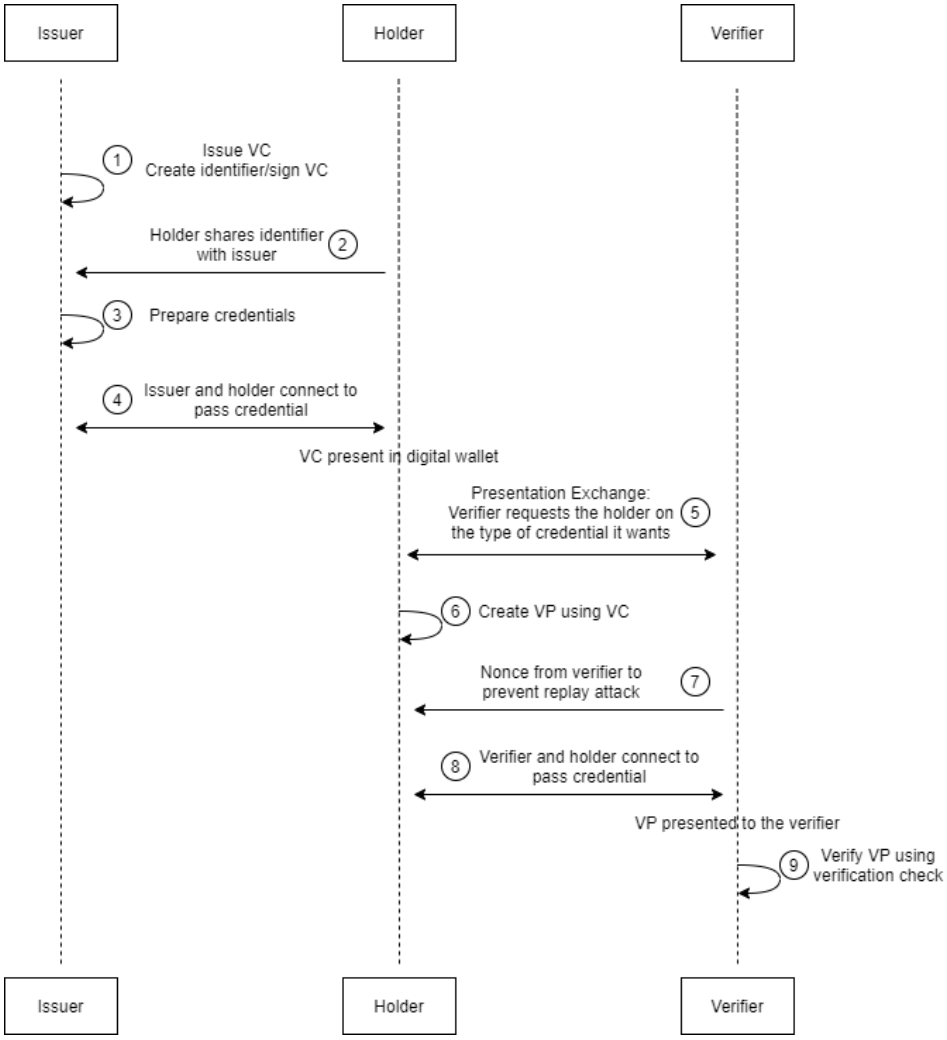

Fig. 1. System Flow

Step 6: Depending on what was asked by the verifier it will be decided whether VP needs to be made from the VC and made if needed to.

Step 7: The verifier produces number used only once (nonce) and sends it to the holder. This is done in-order to prevent any replay attack for the same credentials in the future.

Step 8: As done in step 4 the verifier and holder connect with the help of identifiers and then transfer the required VP/VC. The VP/VC is not present at the verifier side.

Step 9: The verifier verifies the VP/VC using verification check. This process involves the issuer of the VC.

The current system of SSI mentioned above has a major problem. The entity that plays the role of the verifier can exploit it's own power and demand for more than the required credentials and claims from the holder in return for the services offered by it. The digital entity holders may not want to disclose too much information than required due to privacy reason. There is a possibility of abuse of power in SSI: when the holder wants to access the verifier's services or information then the verifier might push the holder to present more than required credential/claims about themselves. This way the verifier will have to pass the criterion present in the policy if not the verifier will not get the desired credentials, thus helping the holder not to be forced into providing more than required information. The issuer policy is a set of instructions in the form of attributes that must be satisfied by the verifier in order for the verifier to view the credentials issued by the issuer to the holder. The issuer policy will be generated by the issuer to encrypt the SSI credentials of the holder. The encrypted credentials of the holder will only be able to be decrypted by the decryption key that satisfies the conditions mentioned in the issuer policy. This decryption key will be obtained by the verifier from the SSI key smith by presenting the verifier's attributes.

To solve the above mentioned problem about the verifier misusing their power, a policy should be made to over look which specific credentials or claims can be accessed by whom and for what reasons. This policy will be created by the issuer. This is so because the issuer will know to whom the generated credentials will be of more value and whom

will want to exploit it. With such information the issuer will be able to create policies accordingly to protect the holder's credentials/claims. To go about implementing this policy the first option is to look into the possible cryptographic techniques/schemes for the same. This policy enforcement can in the future be extended to be done by the holder depending on the scenario faced.

The attacker in this case will be an entity who wants to access the holder's credentials without passing the policy issued by the issuer. Two entities, whom individually can not pass the policy will try to collude together to satisfy the policy and to get the credential. For example entity X, Y wants to get H's credential. H here is the holder. The policy states that in order for an entity to see H's credential, the entity must have the attributes A and B. Entity X has attribute A and entity has attribute B. Individually X and Y can not access H's credential. But combing both their attributes X and Y have the required attributes A and B to access the H's credential. This collusion between entities is not possible and will be explained in *Security analysis and Attacker model for CP-ABE* of section III.

III. Potential Solution

To cryptographically implement the issuer policy for SSI systems various techniques and solutions were considered. Below are a short descriptions of the potential solutions and a short description of why they were not used in the implementation. A detailed explanation of each solution and how they could be used for issuer policy in SSI is provided in the appendix.

- *Shamir Secret Sharing:* In Shamir Secret Sharing the credential issued will be taken as a secret and divided into parts, in our case divided into exactly two parts. In order to read the credential the two parts must be put together, hence one part will be with the issuer and the second with the holder. This way the issuer can verify the verifier when the verifier requests for the credential. This leaves the issuer with lots of power and enables the issuer to know about the usage of the credentials by the holder.
- *Trusted Third party:* The party will over look the whole process from generating the credentials to distributing it. This meant all the communications between the entities including transfer of credentials will go through this third party. This meant it is a single point of failure or compromise.
- *Smart Contract:* Smart contract are digital contracts present on top of the blockchain, that will get executed when certain actions are fulfilled as stated by the code written. The holders credential will be inside the smart contract, the code will be written such that the verifier has to match the conditions of the issuer policy. If the verifier satisfies the issuer policy then they will be able to get access to the credentials of the holder. The disadvantage here is that once the contract is made it is permanent and hence can not be changed. The electric power and cost for running the blockchain is also high.
- *Attribute Based Credentials:* Attribute Based Credentials (ABC) uses the principle of Zero-Knowledge proof which aids in an entity to reveal information about oneself in the form of attributes. An entity called the Semi Trusted Third Party (STTP) generates the attributes for the verifier if the verifier passes the issuer policy. These verifier attributes will be checked by the holder's wallet and if the attributes are correct the requested credentials will be passed to the verifier. The disadvantage of this system is the communication overhead related to the generation of attributes for verifier every time request for credential is made.
- *Attribute Based Encryption:* In Attribute Based Encryption(ABE) the SSI credentials will be encrypted using a set of attribute values. The decryption can be done by an entity who posses the attribute values mentioned during encryption.
- *Two Layer Encryption:* The credential will be encrypted twice. The first encryption can be decrypted by the verifier and the second layer can be decrypted by the holder. The encryption and key generation will be done by the issuer. In this case the issuer could collude with the verifier or could be compromised.
- *Hybrid Solution:* A solution where two or more of the above mentioned solutions are combined to overcome the short comings each solution had.

The solution that was used for implementation is Attribute Based Encryption which is described below.

A. Chosen Solution: Attribute Based Encryption

Traditionally encryption of a message was done either using symmetric or asymmetric keys, where the message will be encrypted using the receiver's public key and the receiver would be able to decrypt it using their own private key. Through out this subsection reference to attributes means both attribute field and attribute values unless stated specifically as either attribute field or attribute values. With Attribute Based Encryption (ABE) the encryption and decryption of the message is done through the receiver's attributes. The base line of this development in encryption and decryption method was called Identity Based Encryption (IBE)[24]. The concept was first introduced by Amit Sahai and Brent Waters in the paper "Fuzzy Identity-Based Encryption[19]". ABE allows messages to be encrypted in such a way that entities with certain attributes will only be able to decrypt the message[19]. This was achieved by using one public key for encryption and many private keys for decryption. Creation of multiple private keys is possible by splitting up the master private key using Shamir's secret sharing. This use of Shamir's secret sharing makes ABE error tolerant and is resistant against collision attacks [24]. Collision resistance means two different receiver's with two different private keys combine them to decrypt a message that is not intended for them.

There are two types of ABE namely Key-Policy Attribute Based Encryption (KP-ABE) [12] and Ciphertext-Policy Attribute Based Encryption (CP-ABE) [6]. In CP-ABE the ciphertext is associated with access structure/policy and the private keys are linked with attribute values of the receiver. The access structure/policy states which private keys can decrypt the ciphertext.

Overall there are four different roles played on ABE specific to SSI which are the encrypter, the decrypter, the user and the key-issuer. The encrypter is the one that encrypts the plaintext message into ciphertext using the public key and the access structure/set of attributes in case of CP-ABE/KP-ABE respectively. The decryptor takes the ciphertext and decrypts it using the public key and private key. The user is the one for whom the message is intended to be delivered. The key-issuer is the one who issues the public key, master secret key and the private keys.

For the purpose of implementing the issuer policy CP-ABE seems to be a better option to implement than KP-ABE. The reason being that in CP-ABE the the encrypter decides the policy about who can decrypt the ciphertext generated. The user's attributes are used to generate the private keys and these attributes are linked to their credentials. While in KP-ABE the encrypted data were the ones about whom the attributes where described and policies where built into the private keys. This means the encrypter will have to trust the key-issuer to ensure the correct execution of the issuer policy. In case of CP-ABE the control of issuing policy and executing it stays with the issuer and not an external party, thus making CP-ABE the apt choice for implementing the issuer policy problem of SSI.

In CP-ABE there are four main algorithms namely the setup, encrypt, key-generation and decrypt based on [6]. In setup algorithm outputs the public key (PK) and master key (MK) taking the security parameters as the input. The encrypt algorithm takes the PK, message and access structure as input and outputs the ciphertext encrypting the message. The key-generation algorithm takes the MK and set of attributes that describe the key and generate the SK and give it as the output. The decrypt algorithm takes PK, SK and the ciphertext as the input and outputs the message. The decryption happens only if SK contains the required attributes stated in the access structure of the ciphertext.

a) Security analysis and Attacker model for CP-ABE: The security model for the cryptographic implementation of issuer policy for SSI systems is similar to that described in [6]. The adversary can query for any private key. This private key queried by the adversary can not be used to decrypt the ciphertext that will be used to challenge the adversary. The adversary will be challenged on the encryption to the access structure of its own choice and can ask for any decryption key such that the decryption key does not satisfy the policy stated in the encryption. The formal security game is as follows based on [6]: (The SSI Key Smith is the challenger)

- **Setup:** The SSI Key Smith runs the setup algorithm. The public parameters (PK) is given to the adversary.
- **Phase 1:**The adversary generated decryption key in correspondence to the set of attributes $S_1, \ldots, S_{q1}$
- **Challenge:** The adversary given two equal length SSI credentials to the SSI Key Smith namely SC_0 and SC_1. The adversary also gives an access structure A*. The A* does not satisfy the $S_1, \ldots, S_{q1}$ from Phase 1. The SSI Key Smith now randomly flips a coin b and encrypts SC_b under A*. The corresponding ciphertext CT* is given to the adversary.
- **Phase 2:** Phase 2 is the repetition of Phase 1, with a restriction that the access structure corresponding to the challenge must not be satisfied by the set of attributes $S_{q1+1}, \ldots, S_{q1}$.
- **Guess:** A guess, b‘ of b is outputted by the adversary.

The advantage of an adversary A in this game is defined as Pr[b‘ = b] -1/2, as stated in [6] . The model can easily be extended to handle chosen-ciphertext attacks by allowing for decryption queries in Phase 1 and Phase 2, as stated in [6].

The implementation of the issuer policy is collusion resistance, because the underlying cryptogrpahic technique CP-ABE is collusion resistance. Meaning two different entities can not combine their attributes to decrypt the holder's credential. This is because CP-ABE is collusion resistance due to randomisation of each key generated for decryption purpose [6].

IV. Issuer Policy

This section explains about what is an issuer policy and about what can and can not be cryptographically implemented as an issuer policy using ABE. Hence forth in this paper if the word attributes are mentioned it is referred to the attributes that are used to identify the verifier and not the attributes values the SSI system issues via the issuer, unless specifically mentioned otherwise.

As discussed in Section II issuer policy is a set of instructions in the form of attributes that the verifier must satisfy in order for the verifier to view the credentials issued by the issuer to the holder. The attributes in the issuer policy can be of two types. One type is where the attribute is of string type where the whole string value will be compared with the verifier's attributes and it should be a exact match. This type of attribute must be mentioned inside quotes for example:"$attributeDescrption - attributeValue$". The second type is of integer type where the arithmetic function $<, >, =$ can be used. This means the given attribute values can be compared to a given integer value and can be in between certain values mentioned according to the policy. This feature enables the issuer to define parameters specific to certain age group or management levels in an organisation or institution. This type of attribute need should not be within quotes but rather in the format for example: $attributeDescription < atributeValue$. All attribute values mentioned in the issuer policy are connected to the next attributed mentioned in the same policy with either the "or" or "and" logical operator. Below are a few examples of how the logical operators "or"

or "and" can be used in accordance to how the issuer policy needs to be formulated.

- "(age>18) and (("nationality-netherlands") or ("nationality-german"))". This issuer policy states that the verifier must be above 18 years of age and should be either a Netherlands citizen or a German citizen.
- "((department-hr) and ("designation-seniormanager")) or (("designation-manager") and ("department-it"))". This issuer policy states that the verifier must be either the senior manager of the HR department or the IT manager.

In the case where a certain integer value alone should be neglected from the policy then the attribute can be written as $(age < 17) and (age > 19)$. Here the the person with the age as 18 will not be included. This is how one can perform negation of a certain integer value. Negation of string values is not straight forward. For example if all the EU nationalities except Netherlands can view the encrypted credential, the issuer policy must include all the nationalities in the format as follows "$("nationality - austria")or("nationality - belgium")or("nationality - bulgaria")or("nationality - croatia")or("nationality - republicofcyprus")or("nationality - czechrepublic")or("nationality - denmark")or("nationality - estonia")or("nationality - finland")or("nationality - france")or("nationality - germany")or("nationality - greece")or("nationality - hungary")or("nationality - ireland")or("nationality - italy")or("nationality - latvia")or("nationality - lithuania")or("nationality - luxembourg")or("nationality - malta")or("nationality - poland")or("nationality - portugal")or("nationality - romania")or("nationality - slovakia")or("nationality - slovenia")or("nationality - spain")or("nationality - sweden")$", and not mentioning Netherlands.

The cases where issuing issuer policy over the SSI attributes are not possible are when the same set of SSI attributed must be dynamically changed depending on the situation. For now SSI attributes that are static and will not change drastically can be encrypted using the issuer policy. When the condition to be stated in the issuer policy should be static and not dynamic in nature. The attributes must be in the format as stated above in paragraph 3.

V. Implementation

This section describes how CP-ABE can be used to implement the issuer policy in SSI systems.

Figure 2 describes how ABE can be used for the SSI systems and the entities. The four algorithms of CP-ABE as mentioned in Section III are used by the entities mentioned in figure 2 to implement the issuer policy. The entities are the issuer, the holder, the verifier and the SSI Key Smith. The SSI Key Smith here is a new additional entity to the SSI system compared to that mentioned in Figure 1. SSI Key Smith is similar to that of Semi-Trusted Third Party in Attribute Based Credentials, with the difference of name to match it's main role of the generation of the master and secret keys to be used in the system. SSI key smith used the setup algorithm to generate the keys needed for encryption and decryption processes. The issuer used the encrypt algorithm to encrypt the SSI credentials which is to be issued to the holder. The verifier uses the decrypt algorithm to decrypt the encrypted SSI credentials of the holder.

The description of the Figure 2 is as follows. In step 1 the SSI Key Smith generates the master key (MK) and the public key (PK). The MK is transferred to the issuer in step 2. The issuer then prepares the issuer policy in step 3 and the verifier will now ask for the secret key by presenting it's own attributes in step 4. In step 5 the SSI Key Smith generates the SK using the MK that was generated in step 1 and is linked with the attribute values of the verifier for whom the SK is generated. Now the SK is transferred to the verifier in step 6. In step 7 the holder requests for its credentials. The issuer now prepares the credentials and encrypts it using PK provided by the SSI Key Smith and with the access structure that states which attributes need to be present in the SK for decryption in step 8. These encrypted credentials is then transferred to the holder in step 9. The verifier requests for the credentials of the holder in step 10. The verifier transfers the encrypted credentials to the verifier in step 11. In step 12 the verifier will be able to decrypt the encrypted credentials if the SK of the verifier matches with attributes present in the access structure of the encrypted credentials.

Figure 3 describes a scenario where issuer policy can be used in a office setting. All the entities namely the issuer, holder, verifier, SSI Key Smith belong to the same organisation. In step 1 the SSI Key Smith generates the master key (MK) and the public key (PK). The PK is transferred to the issuer in step 2. In step 3 the holder requests to the SSI Key Smith for a decryption key by presenting it's own attribute values. In step 4 the SSI Key Smith generates the SK using the attributes provided by the holder and transfers the SK to the holder in step 5. The holder now requests for SSI credentials to the issuer by providing it's E_No (unique to each individual in the organisation) as an identifier in step 6. The holder sends the E_No as identifier to the issuer because the holder wants to see their own encrypted credentials issued to them. In step 7 the issuer will now prepare the issuer policy for the SSI credential to be encrypted. The issuer policy will also include the E_No at the end of the policy with an "or" logical operator. This way the holder can also request for their own decryption key (as done in step 3) and be able to decrypt the encrypted credentials. In step 8 the issuer encrypts the SSI credentials using the SSI credential, PK and an access structure that contains the issuer policy. The encrypted SSI credentials is transferred to the holder in step 9. In step 10 the verifier requests for its decryption key (sk) to the SSI Key Smith by specifying it's own attributes. The SSI Key Smith generates the SK for the verifier in step 11 and transfers the SK to the verifier in step 12. In step 13 the verifier requests the holder for the holder's SSI credential. The holder transfers the encrypted SSI credential to the verifier in step 14. The verifier

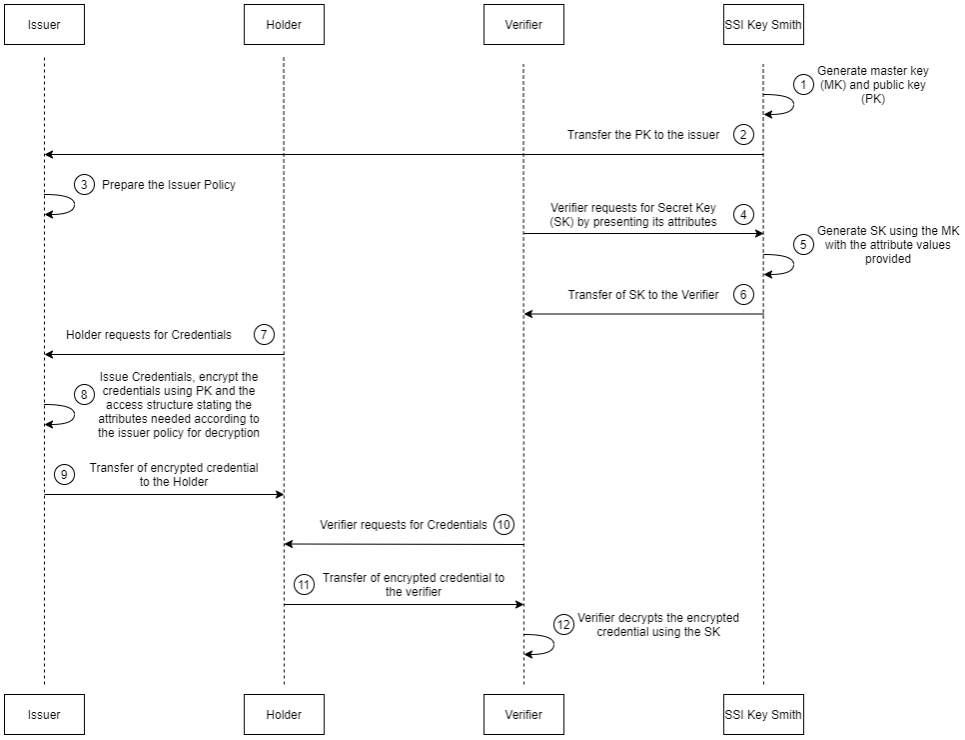

Fig. 2. System flow of ABE

decrypts the encrypted SSI credential of the holder using the encrypted SSI credential, SK and PK.

Each PK will be associated with certain set of fixed attribute list. This is way the issuer can decide to use all or some of the attributes in the list. The verifier when requesting for the decryption key (SK) from the SSI Key Smith, will present all the attribute values in the list of that particular PK used by the issuer. This is done so that if needed the issuer can keep it confidential which attributes are required from the verifier. The issuer policy stated in step 7 of figure 3 has two attributes Designation and E_No having * as the value. This is because the issuer in this case did not need the attributes Designation and E_No to create the policy so * indicates that the verifier can be of any Designation and E_No to be able to satisfy that part of the issuer policy. Another use of the * for attribute value in the policy can be when the policy wants to state the encrypted SSI credential is for everyone in the organisation then the department attribute value will be *. The attribute having the * value denotes that particular attribute in the issuer policy will be a wildcard and the verifier when trying to satisfy the issuer policy can have any value for that particular attribute. This feature has not yet been implemented due to time constraints. The possible way to implement the wildcard feature is explained in section VII

The roles of the issuer, holder, verifier will be played interchangeably by the organisation employees or the departments in the organisation, this will depend on the credentials that are shared. The role of the SSI Key Smith will be played by a separate group of individuals from the organisation who will over look the process of key generation and key distribution within the organisation. In case the individual or department in charge of the key distribution will be playing any of the roles of issuer, holder, verifier then the key distribution for that particular scenario will have to be taken care by another individual or department. This is to ensure there is no colluding between parties and to ensure fair play.

VI. Experiment and result

The implementation was done using Ubuntu in an Oracle VM VirtualBox Manager. The base memory of the system was 5691 MB. The code for implementation was done in python using builtin libraries. The charm library's `Pabe_bsw07`, `HybridABEnc` and `PairingGroup` functions were mainly use for the key generation, encryption and decryption process. All the generated credentials and variables were converted to json and stored in a json file. This was done because most SSI systems communicate within themselves by transferring data in JSON format [38]. The web interface demo explaining the sequence of actions that will take place in a real world setting with issuer policy in place was developed using Flask.

In order to understand how including the issuer policy to the systems of SSI will influence the speed of generation of keys or encryption or decryption of messages, there were a number of benchmarks made to the demo code. The number of benchmarks were added to see what will happen if the system is deployed in a real world scenario where more than just a couple of attributes for the issuer policy or the verifier were used. The benchmarks include:

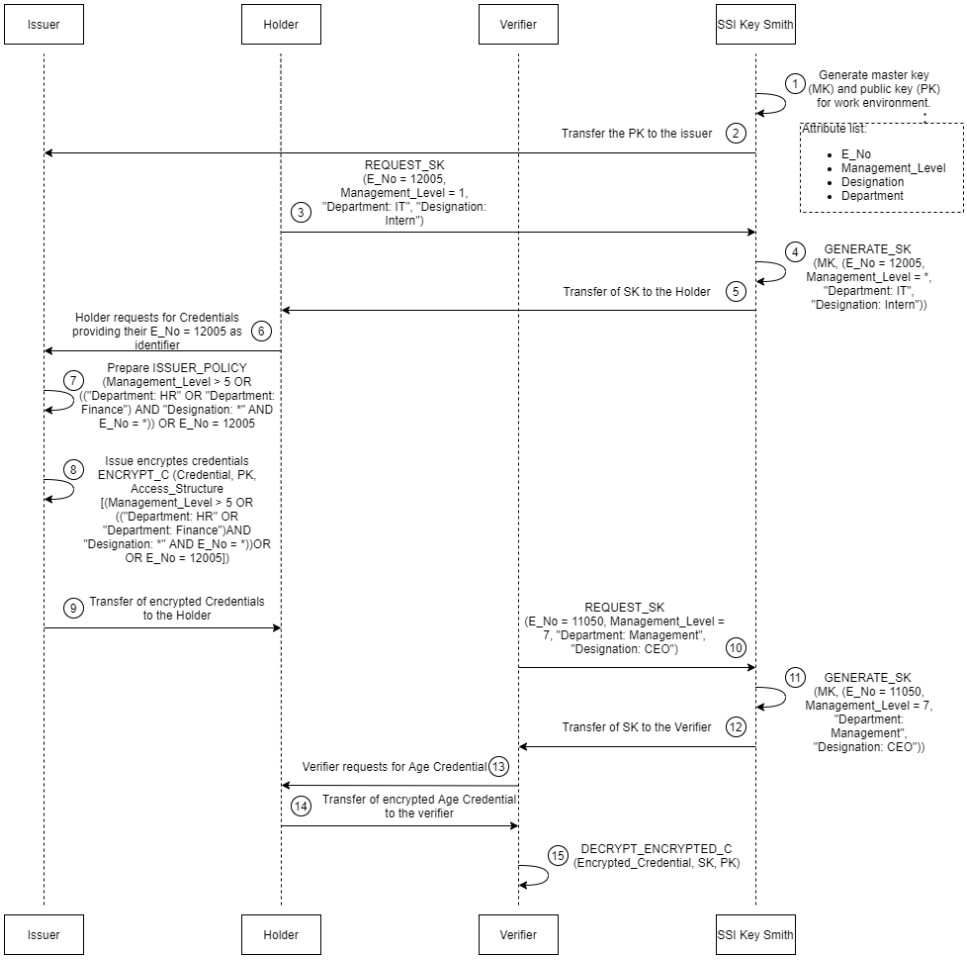

Fig. 3. CP-ABE Scenario

- generating up to 100000 number of attributes for the verifier's key generation.
- generating issuer policy with attributes up to 100000 numbers

The table I displays the result of running the demo code with 1, 50, 100, 500, 1000, 5000, 10000 verifier's attributes and checking the time it took for the generation of verifier's key, encryption and decryption with the respective number of verifier's attributes. The table II below displays the time taken for generating the verifier's key, encryption and decryption but with respect to the number of issuer policy attributes. Each operation for the generation of verifier's key, encryption and decryption was run 100 times and each time it was run the results were stored in a excel file. In order to report stable values it was decided to present the results in the format of the 25^{th} percentile, 50^{th} percentile(the median) and the 75^{th} percentile. The 25^{th}, 50^{th}, 75^{th} percentile is the 25^{th}, 50^{th}, 75^{th} value respectively in the *sorted* table containing all 100 values. Figures 4, 5, 6 are the graphical representation of the data present in the table I. Figures 7, 8, 9 are the graphical representation of the data present in the table II.

The graphical representations are of logarithmic scale and not liner in the x-axis. The increase in the attributes for issuer policy and verifier are done in multiples of 10 alternatively and the middle value in between is half that of the preceding value. The size of Versifier's key and the encrypted SSI credential size always remained the same. The size was of 248 Bytes unchanged even with the increase in the number of verifier's attributes or the issuer policy attributes. This is because the generation of keys and the encryption because of the underlying cryptogrpahic functions used and the output from those functions should remain same for Confidentiality, integrity and availability concept.

a) Increase in the number of Verifier's attributes: As seen in the table I and in the Figure 4 with the increase in the number of verifier's attributes used, the verifier's key generation time will also increase. The encryption time remains the same even if there is a big increase in the number of attributes. This is because the attribute increase is for the verifier's attribute and this will not be used for the encryption process. The graph in Figure 5 shows fluctuation in the encryption time from when the number of attributes was from 1 to 100000. It can also be noted that the time is between 0.01 to 0.02 seconds. The fluctuation is within this limit all the times the

No.of Verifier's attributes	Verifier Key Generation Time (seconds)			Encryption Time (seconds)			Decryption Time (seconds)		
	25^{th} percentile	50^{th} percentile	75^{th} percentile	25^{th} percentile	50^{th} percentile	75^{th} percentile	25^{th} percentile	50^{th} percentile	75^{th} percentile
1	0.005882006	0.006526022	0.007219986	0.016898067	0.018134586	0.020474505	0.004602718	0.004848961	0.005098016
50	0.219068768	0.244886369	0.295062453	0.015087816	0.016456093	0.019169742	0.00460665	0.005014573	0.005497464
100	0.401992285	0.425789023	0.478297425	0.015568035	0.016125024	0.016948232	0.004434496	0.004572559	0.004747808
500	2.016466364	2.038748405	2.079696979	0.015163625	0.015594727	0.016162428	0.004996375	0.005190461	0.005541626
1000	4.066891448	4.130286042	4.329489236	0.016049709	0.016751673	0.01743158	0.005741721	0.006056564	0.006475072
5000	20.42549143	20.61074902	20.81979582	0.014899917	0.015330052	0.016132367	0.008755519	0.009035359	0.009362737
10000	31.62823555	32.69538592	37.74589325	0.011648187	0.011913117	0.012229255	0.00994278	0.010102258	0.010364182

TABLE I

VARYING NUMBER OF VERIFIER'S ATTRIBUTES AND THE CORRESPONDING TIME TAKEN FOR VERIFIER'S KEY GENERATION, ENCRYPTION, AND DECRYPTION

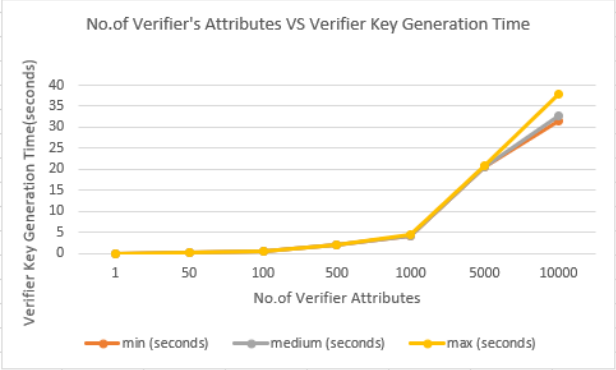

Fig. 4. No.of Verifier's Attributes VS Key Generation Time

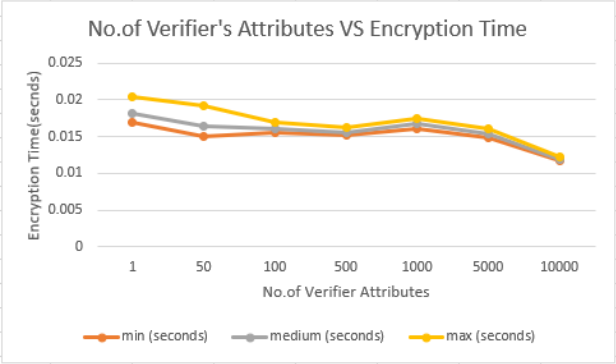

Fig. 5. No.of Verifier's Attributes VS Encryption Time

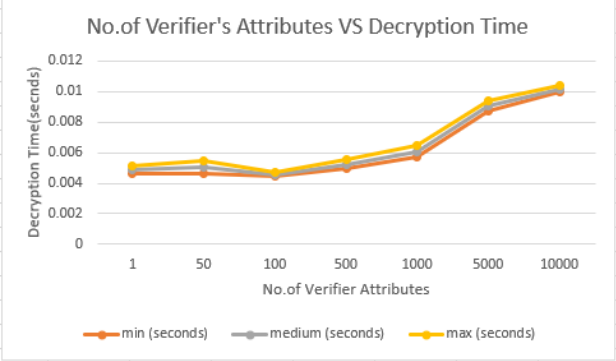

Fig. 6. No.of Verifier's Attributes VS Decryption Time

code was run. All the 100 time values were within the limit of 0.01 to 0.02 seconds. Hence the encryption time remains same and does not increase with the increase in the number of attributes. In decryption the verifier attributes are used hence with the increase in the number of attributes the decryption time will also increase. The increase in decryption time can be seen in the table I and the graph in Figure 6.

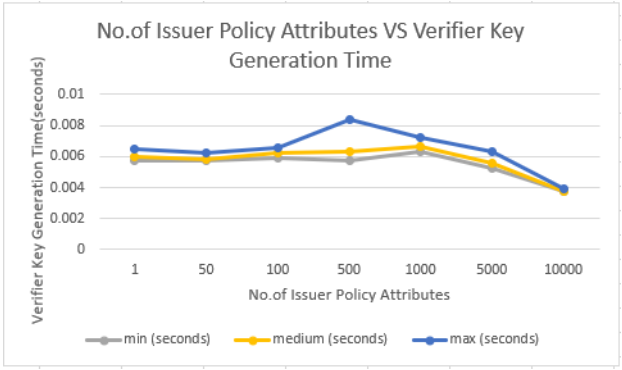

Fig. 7. No.of Issuer Policy Attributes VS Key Generation Time

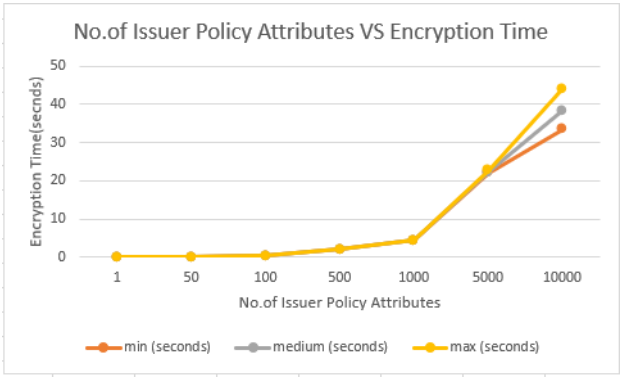

Fig. 8. No.of Issuer Policy Attributes VS Encryption Time

b) Increase in the number of issuer policy attributes: The key generation time for verifier remains same even if there is a big increase in the number of issuer policy attributes. This is because the issuer policy attribute is not directly linked to the production of verifier decryption key. The graph in Figure 7 shows fluctuation in the time from when the number of attributes was from 1 to 100000. It can also be noted that the time is between 0.004 to 0.01 seconds. The fluctuation is within this limit all the times the code was run. All the 100

No.of Issuer Policy attributes	Verifier Key Generation Time (seconds)			Encryption Time (seconds)			Decryption Time (seconds)		
	25^{th} percentile	50^{th} percentile	75^{th} percentile	25^{th} percentile	50^{th} percentile	75^{th} percentile	25^{th} percentile	50^{th} percentile	75^{th} percentile
1	0.005772739	0.006009826	0.00644919	0.006756932	0.007130283	0.008073292	0.004031721	0.004192069	0.004553541
50	0.005731352	0.005842947	0.006228989	0.209232538	0.220608331	0.279577014	0.014465203	0.016077292	0.019417157
100	0.005914765	0.006197408	0.006585778	0.406153804	0.416938659	0.44920792	0.02438119	0.02563117	0.030377412
500	0.005773694	0.006344997	0.008388364	2.059610863	2.088977897	2.131039651	0.105734926	0.11234979	0.11549398
1000	0.00631281	0.006648805	0.007189324	4.37616397	4.428854528	4.504344116	0.230829796	0.235023488	0.242491432
5000	0.005236771	0.00561015	0.006306851	22.07240124	22.48996826	22.78393184	0.793044545	0.813155369	1.059188805
10000	0.003714354	0.003767243	0.003936868	33.504725	38.41387986	44.17917548	1.668248514	1.949586747	2.5513674

TABLE II
VARYING NUMBER OF VERIFIER'S ATTRIBUTES AND THE CORRESPONDING TIME TAKEN FOR VERIFIER'S KEY GENERATION, ENCRYPTION, AND DECRYPTION

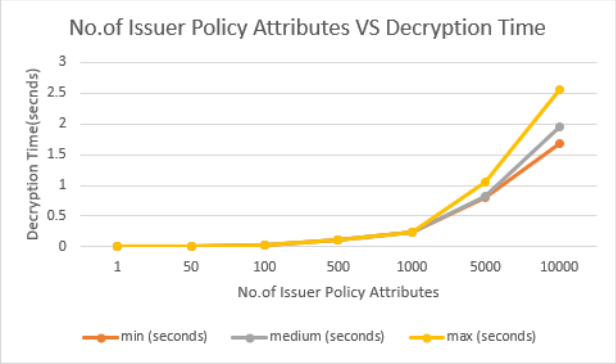

Fig. 9. No.of Issuer Policy Attributes VS Decryption Time

time values were within the limit of 0.004 to 0.01 seconds. Hence the key generation time remains same and does not increase with the increase in the number of issuer policy attributes. With the increase in the number of issuer policy attributes used, the encryption time also increases. This can be clearly seen in the table II and in the Figure 8. The reason being is that the issuer policy attributes are used during the encryption process with the increase in number it should result in higher generation time. In decryption process the issuer policy attributes are used hence with the increase in the number of issuer policy attributes the decryption time will also increase. The increase in decryption time can be seen in the table II and the graph in Figure 9.

It can be inferred from the tables II and I that the decryption time increases linearly with respect to the increase in issuer policy attributes and verifiers attributes. Given that there are 50 holders with 100 issuer policy attributes each who want to get service/information from the verifier. The verifier will be able to decrypt the encrypted SSI credentials of the holders less that 1 seconds. The same action for decryption for 50 holders with 100 verifier attributes will take about 0.01 seconds. In the same way the time taken for encrypting 50 holder's SSI credentials with 100 issuer policy attributes is about 22 seconds. Compared to the decryption time the time taken for encrypting the SSI credentials is higher. Given the scenario that the encryption for a certain SSI credential will be done once and stored with the holder whereas the decryption process will take place many times with various verifiers it is needed that the decryption time is as minimum as possible, which in our case is so. From these observations it can be concluded that the implementation can be used in real time.

VII. FUTURE WORKS

This section explores the future development that can be added to the current model. These include the following concepts:

- Certain scenarios might require the issued credential to be revoked or to be considered invalid. For this to happen a time constraint can be added to the policies being issued. During the generation of SSI attributes and encrypting them with the issuer policy the policy should include a time bound parameter that specifies till when the underlying attributes of SSI are valid. Another way to have time constraint is to enact the policies via blockchain. Executing SSI attributes in a blockchain with the help of smart contracts. The smart contracts can state that after the defined time period the underlying SSI attributes should not be revealed to entity or made use by any entity.
- The wildcard feature mentioned in section V has yet to be implemented. The way to implement it is to include an extra functionality in the built-in library of charm. [] could be done using `import re` python function. which displays all the values of after * for example if words with like as the prefix then words such as likeable, likeliness will all be considered this can be used for attributes. Example for using `re` library for issuer policy where all the departments in the institution can view the encrypted credential, the issuer policy will be written as $("department-.+")$. The $.+$ followed after $department-$ signifies the functionality of wild-card *. Meaning all the attributes which begin with $department-$ can get access to the encrypted credential.
- The concept of using negation when describing a policy was seen in Section IV, with the example of age for integer type and nationality for string type. The negation for integer is simple to implement but where as the string type is rather long and not so convenient in cases where the values for certain attributes are rather large in number. Having a straight forward approach to the negation concept in issuer policy needs to be explored.
- The SSI credentials issued to the holder is in an encrypted form that can be presented to the verifier whom upon getting the verifier's key can decrypt the credential given that the verifier's attribute satisfies the issuer's policy. After decryption the SSI credential will be in plain text and readable by the verifier. In scenarios where the

verifier might be only required to verify that the holder has the credentials and not see the credentials zero knowledge proof(ZKP) [21] could be used. For example if the verifier wants to know if the holder is above 18 years old the verifier can get that information when CPABE is combined with ZKP rather than knowing the holder's age or date of birth. This will ensure better privacy for certain credentials. Combining ZKP with CPABE needs to be explored.

VIII. Conclusion

This paper provides a solution to cryptographically implementing issuer policy for SSI system. The implementation was done using CP-ABE. CP-ABE is resistant against collusion attacks and secure against chosen ciphertext attack [6]. The implementation is such that the SSI credentials will be encrypted with a policy. The policy is defined set of attributed combined together with logical operators "or" and "and". The verifier will be able to decrypt the encrypted SSI credentials if the verifier's attributes satisfy the issuer policy of the encrypted credential. The policy enforcement for now is performed by the issuer but depending on the scenario the holder can also enforce policy on the SSI credentials provided to them. This however is not explored in this paper and left for future scope along with exploring the possibility of combining ZKP with CP-ABE issuer policy implementation for SSI systems. This research has shown that the solution provided is practical to be implemented in the real-life situation with the help of the demo and the performance analysis of the code.

References

[1] Carlisle Adams. "Trusted Third Party". In: *Encyclopedia of Cryptography and Security*. Ed. by Henk C. A. van Tilborg and Sushil Jajodia. Boston, MA: Springer US, 2011, pp. 1335–1335. ISBN: 978-1-4419-5906-5. DOI: 10.1007/978-1-4419-5906-5_98. URL: https://doi.org/10.1007/978-1-4419-5906-5_98.

[2] Maher Alharby and Aad van Moorsel. "Blockchain Based Smart Contracts : A Systematic Mapping Study". In: *Computer Science Information Technology (CS IT)* (Aug. 2017). DOI: 10.5121/csit.2017.71011. URL: https://arxiv.org/abs/1710.06372 (visited on 04/04/2021).

[3] Muneeb Ali et al. *Blockstack Technical Whitepaper*. 2017. URL: https://pdos.csail.mit.edu/6.824/papers/blockstack-2017.pdf (visited on 09/30/2021).

[4] Christopher Allen. *The Path to Self-Sovereign Identity*. Lifewithalacrity.com, Apr. 2016. URL: http://www.lifewithalacrity.com/2016/04/the-path-to-self-soverereign-identity.html (visited on 03/18/2021).

[5] *Attribute Based Credentials - Privacy Patterns*. privacypatterns.org. URL: https://privacypatterns.org/patterns/Attribute-based-credentials#:~:text=Attribute%5C%20Based%5C%20Credentials%5C%20(ABC)%5C%20are (visited on 04/09/2021).

[6] John Bethencourt, Amit Sahai, and Brent Waters. "Ciphertext-Policy Attribute-Based Encryption". In: *2007 IEEE Symposium on Security and Privacy (SP '07)*. 2007, pp. 321–334. DOI: 10.1109/SP.2007.11.

[7] Jan Camenisch and Els Van Herreweghen. "Design and Implementation of the Idemix Anonymous Credential System". In: *Proceedings of the 9th ACM Conference on Computer and Communications Security*. CCS '02. Washington, DC, USA: Association for Computing Machinery, 2002, pp. 21–30. ISBN: 1581136129. DOI: 10.1145/586110.586114. URL: https://doi.org/10.1145/586110.586114.

[8] *Decentralized Identifiers (DIDs) v1.0*. w3c.github.io. URL: https://w3c.github.io/did-core/ (visited on 03/12/2021).

[9] *eSSIF-Lab Glossary — eSSIF-Lab*. essif-lab.pages.grnet.gr. URL: https://essif-lab.pages.grnet.gr/framework/docs/essifLab-glossary (visited on 03/25/2021).

[10] J. M. de Fuentes et al. "Assessment of attribute-based credentials for privacy-preserving road traffic services in smart cities". In: *Personal and Ubiquitous Computing* 21 (Oct. 2017), pp. 869–891. DOI: 10.1007/s00779-017-1057-6. URL: https://link.springer.com/article/10.1007%5C%2Fs00779-017-1057-6 (visited on 11/24/2020).

[11] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. "The Knowledge Complexity of Interactive Proof Systems". In: *SIAM Journal on Computing* 18 (Feb. 1989), pp. 186–208. DOI: 10.1137/0218012. (Visited on 04/15/2021).

[12] Vipul Goyal et al. "Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data". In: *Proceedings of the 13th ACM Conference on Computer and Communications Security*. CCS '06. Alexandria, Virginia, USA: Association for Computing Machinery, 2006, pp. 89–98. ISBN: 1595935185. DOI: 10.1145/1180405.1180418. URL: https://doi.org/10.1145/1180405.1180418.

[13] Dick Hardt. *The OAuth 2.0 Authorization Framework*. RFC 6749. Oct. 2012. DOI: 10.17487/RFC6749. URL: https://rfc-editor.org/rfc/rfc6749.txt.

[14] *Hyperledger Indy*. Hyperledger. URL: https://www.hyperledger.org/use/hyperledger-indy (visited on 09/28/2021).

[15] V. Morabito. "Business Innovation Through Blockchain The B³ Perspective". In: *www.academia.edu* (), pp. 106–107. URL: https://www.academia.edu/35863002/Business_Innovation_Through_Blockchain_The_B_Perspective (visited on 01/25/2021).

[16] *Multiple encryption*. Wikipedia, Mar. 2021. URL: https://en.wikipedia.org/wiki/Multiple_encryption (visited on 04/14/2021).

[17] Jelle Nauta and Rieks Joosten. *Self-Sovereign Identity: A Comparison of IRMA and Sovrin*. July 2019. DOI: 10.13140/RG.2.2.19755.18721.

[18] *OpenID Connect — OpenID*. OpenID Connect. URL: https://openid.net/connect/ (visited on 09/25/2021).

[19] Amit Sahai and Brent Waters. *Fuzzy Identity Based Encryption*. Cryptology ePrint Archive, Report 2004/086. https://eprint.iacr.org/2004/086. 2004.

[20] Abylay Satybaldy, Mariusz Nowostawski, and Jørgen Ellingsen. *Self-Sovereign Identity Systems Evaluation framework*. URL: https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/2731400/SSI+systems+evaluation+framework.pdf?sequence=2 (visited on 09/27/2021).

[21] Berry Schoenmakers. "Zero-Knowledge". In: *Encyclopedia of Cryptography and Security*. Ed. by Henk C. A. van Tilborg and Sushil Jajodia. Boston, MA: Springer US, 2011, pp. 1401–1403. ISBN: 978-1-4419-5906-5. DOI: 10.1007/978-1-4419-5906-5_16. URL: https://doi.org/10.1007/978-1-4419-5906-5_16.

[22] *Self-Sovereign Identity*. TNO. URL: https://www.tno.nl/en/focus-areas/information-communication-technology/roadmaps/data-sharing/ssi/ (visited on 03/31/2021).

[23] Adi Shamir. "How to Share a Secret". In: *Commun. ACM* 22.11 (Nov. 1979), pp. 612–613. ISSN: 0001-0782. DOI: 10.1145/359168.359176. URL: https://doi.org/10.1145/359168.359176.

[24] Adi Shamir. "Identity-Based Cryptosystems and Signature Schemes". In: *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*. Vol. 196. Lecture Notes in Computer Science. Springer, 1984, pp. 47–53. DOI: 10.1007/3-540-39568-7_5.

[25] *ShoCard*. www.shocard.com. URL: https://www.shocard.com/en.html (visited on 09/29/2021).

[26] Gustavus J. Simmons. "Symmetric and Asymmetric Encryption". In: *ACM Comput. Surv.* 11.4 (Dec. 1979), pp. 305–330. ISSN: 0360-0300. DOI: 10.1145/356789.356793. URL: https://doi.org/10.1145/356789.356793.

[27] Paul Snow et al. *Factom Business Processes Secured by Immutable Audit Trails on the Blockchain*. Apr. 2018. URL: https://4454jm4bovib1sa6vrtflbew-wpengine.netdna-ssl.com/assets/docs/Factom_Whitepaper_v1.2.pdf (visited on 05/07/2021).

[28] Kevin Solorio, Randall Kanna, and David H. Hoover. *Hands-On Smart Contract Development with Solidity and Ethereum: From Fundamentals to Deployment*. "O'Reilly Media, Inc.", Nov. 2019, pp. 70–76. URL: https://www.google.com/books/edition/Hands_On_Smart_Contract_Development_with/thbADwAAQBAJ?hl=en&gbpv=1&kptab=getbook (visited on 04/05/2021).

[29] *Sovrin ™ : A Protocol and Token for Self- Sovereign Identity and Decentralized Trust A White Paper from the Sovrin Foundation*. 2018. URL: https://sovrin.org/wp-content/uploads/2018/03/Sovrin-Protocol-and-Token-White-Paper.pdf (visited on 04/15/2021).

[30] Manu Sporny, Dave Longley, and David Chadwick. *Verifiable Credentials Data Model 1.0*. W3.org, Nov. 2019. URL: https://www.w3.org/TR/vc-data-model/ (visited on 04/28/2021).

[31] *Stacks*. www.stacks.co. URL: https://www.stacks.co/ (visited on 09/30/2021).

[32] Nick Szabo. "Formalizing and Securing Relationships on Public Networks". In: *First Monday* 2.9 (Sept. 1997). DOI: 10.5210/fm.v2i9.548. URL: https://firstmonday.org/ojs/index.php/fm/article/view/548.

[33] *Technical overview · IRMA docs*. irma.app. URL: https://irma.app/docs/overview/#attribute-based-signatures (visited on 09/28/2021).

[34] *U-Prove*. Microsoft Research. URL: https://www.microsoft.com/en-us/research/project/u-prove/ (visited on 04/13/2021).

[35] *What is IRMA? · IRMA docs*. irma.app. URL: https://irma.app/docs/what-is-irma/ (visited on 09/28/2021).

[36] *What is OpenID? — OpenID*. Oct. 2007. URL: https://openid.net/what-is-openid/ (visited on 09/25/2021).

[37] Jiani Wu and Nguyen Khoi Tran. "Application of Blockchain Technology in Sustainable Energy Systems: An Overview". In: *Sustainability* 10.9 (2018). ISSN: 2071-1050. DOI: 10.3390/su10093067. URL: https://www.mdpi.com/2071-1050/10/9/3067.

[38] Kaliya Young. *Verifiable Credentials Flavors Explained*. URL: https://www.lfph.io/wp-content/uploads/2021/02/Verifiable-Credentials-Flavors-Explained.pdf (visited on 03/19/2021).

[39] *Zero-knowledge proofs · IRMA docs*. irma.app. URL: https://irma.app/docs/zkp/ (visited on 09/28/2021).

[40] Zibin Zheng et al. "An overview on smart contracts: Challenges, advances and platforms". In: *Future Generation Computer Systems* 105 (2020), pp. 475–491. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2019.12.019. URL: https://www.sciencedirect.com/science/article/pii/S0167739X19316280.

APPENDIX

A. Key Terms:

Most of the key terms used in this report will follow the definition stated for them in the eSSIF-Lab Glossary[9].

B. Potential Solutions:

This section explains the concepts of some of the possible solutions to the problem statement and the research questions. How these concepts will be used to provide solution to the problem and be implemented in the SSI systems will also be explained.

1) ***Shamir Secret Sharing****:* The Shamir Secret Sharing scheme was first introduced by Adi Shamir in his paper "How to share a secret [23]" in the year 1979. This scheme has been popularly used for various use cases since then. A given piece of data or information or secret will be divided into n number of parts. If p number of pieces from the n number of divided parts is brought together then the underlying data

TERMS	DEFINITION / EXPLANATION
ACTION*	Something that is actually done/executed - by a single actor (on behalf of a given party), as a single operation in a specific context.
ACTOR*	Entity that can act (do things), e.g. people, machines, but not organizations.
BLOCKCHAIN	Blockchain is like a chained data structure in which data blocks are connected in a time sequence, and cryptography is used to guarantee the non-defective modification and unforgeability of the distributed ledger [37].
CLAIMS	An assertion made about a subject[30].
CREDENTIAL*	Data, representing a set of assertions (claims, statements), authored and signed by, or on behalf of, a specific party.
ENTITY*	Something that is known to exist.
HOLDER*	The capability to handle presentation requests from a peer agent, produce the requested data (a presentation) according to its principal's holder-policy, and send that in response to the request.
ISSUER*	The capability to construct credentials from data objects, according to the content of its principal's issuer-Policy (specifically regarding the way in which the credential is to be digitally signed), and pass it to the wallet-component of its principal allowing it to be issued.
LINKED DATA (LD)	The information present on the internet is present in human readable form, not machine understandable. Linked data is connecting the data present in the web to be machine readable in the form of property and value mechanism (key:value). The web now becomes a Global Information Repository with different information linked to one another.
ORGANIZATION*	A group of people that work to realize one or more objectives.
OWNERSHIP*	The rights and duties, as defined and enforced in that jurisdiction, of that entity to enjoy, dispose of, and control the other entity.
PARTY*	An entity that has objectives, knowledge about what exists, rules that (should) apply, and some capability that allows it to reason, make decisions, generate and maintain knowledge etc. in a self-Sovereign fashion; humans and organizations are the typical examples.
POLICY*	A (set of) rules, working-instructions, preferences and other guidance for the execution of one or more kinds of actions, that agents (a) have access to, (b) can interpret as intended by their principal (i.e. policy owner) and (c) must use when executing such actions.
VERIFIABLE CREDENTIAL (VC)*	Credential that comes with assurances regarding its provenance (the party that issued it) and its integrity (the property that the credential data has not been tampered with in transit, i.e. is the same as when issued).
VERIFIABLE PRESENTATION (VP)	The holder after receiving its VCs from the issuer will want to share some of the claims or information present in the VCs to the verifier. These claims or information can be combined and cryptographically signed to be shared with the verifier. VPs are signed by the holder and presented to the verifier.
VERIFIER*	The capability to request peer agents to present (provide) data from credentials (of a specified kind, issued by specified parties), and to verify such responses (check structure, signatures, dates), according to its principal's verifier policy.
WALLET*	Wallet (functional component): the capability to securely store data as requested by Colleague Agents, and to provide stored data to Colleague Agents or Peer Agents, all in compliance with the rules of its Principal's Wallet Policy.
ZERO-KNOWLEDGE PROOF (ZKP)	Zero-knowledge is a property attributed to interactive proofs, interactive arguments, and non interactive proofs. The soundness property protects the interest of the verifier, the zero-knowledge property protects the interest of the prover. By means of a zero-knowledge proof, the prover is able to convince the verifier of the validity of a given statement, without releasing any knowledge beyond the validity of the statement [21, 11].

*: definition from [9]. **Bold**: definition rewritten on own-terms from other sources

or information or secret will be revealed. The number of *p* should be less than or equal to the *n* number of divided parts. The scheme works in such a way that even if *n-1* number of pieces are put together the underlying data or information or secret would not be able to be reconstructed as well as no information will be leaked at all [23].

This scheme is designed based on polynomial interpolation. The technical explanation of Shamir Secret Sharing as stated in [23] is as follows:

1) Notations:
   - Let D denote data.
   - Let p denote the number of points in a 2-dimensional plane $(x_1, y_1), \cdots, (x_p, y_p)$;
   - Let us assume that there are unique x_i such that $q(x)$ is a polynomial with degree $p-1$, with $q(x_i) = y_i$ for all i.
2) Setting up/Construction:
   - Dividing D into D_i parts: $q(x)$ is a polynomial of degree $p-1$ where the co-efficients k_i for $i > 0$ is chosen randomly, such that $q(x) = k_0 + k_1 x + \cdots k_{p-1} x^{p-1}$ where $k_0 = D$ and $D_1 = q(1), \cdots, D_i = q(i), \cdots, D_n = q(n)$.
3) Reconstruction:
   - Putting back D_i parts to D: the co-efficient $a_0, \cdots, a_{p-1}$ of the polynomial $q(x)$ can be found using interpolation given any subset of p of D_i values along with their identify indices and by the evaluation of $D = q(0)$.

Value of D can only be calculated with all the p values in place, given $p-1$ values will also not reveal any information about what the value of D is.

In step 3 of Figure 1 when the issuer prepares the credentials the issuer will have to apply Shamir secret sharing technique on the credential. Here the data D will be the credential that

is to be issued to the holder. The p value will be 2, one for the issuer and one for the holder. When the verifier asks for the credentials to the holder in step 5 of Figure 1, the holder presents its part of the secret value. The verifier now will have to ask the issuer for the remaining part of the secret value. This is so because for reconstruction both the values present in the issuer and holder must be combined or brought together in-order to obtain the underlying credential value. The issuer will now verify if the verifier matches with the issuer policy. In-case the verifier passes the criteria present in issuer policy the issuer's secret will be given allowing the verifier to combine it with the holder's secret to get the required credential. In-case the verifier does not pass the criterion present in issuer policy it will not be presented with any secret. The verifier will have the secret parts from the holder but not that of the issuers, since all the required are not present the credential needed by the verifier can not be reconstructed.

*2) **Trusted Third party**:* Another solution for the problem is to use a Trusted Third Party (TTP) to look over the whole procedure of exchange of credentials and claims. A TTP works in such a way that all the parties involved will trust the TTP for interactions and exchange of information or goods. In some architectures, the TTP is required to store and protect long-term secrets, a compromise of the secret will result in reveal of the future as well as past communications until the new long-term secrets are established [1]. Examples of TTP are certificate authority (CA) in public key infrastructure (PKI), key distribution centre (KDC) in Kerberos [1].

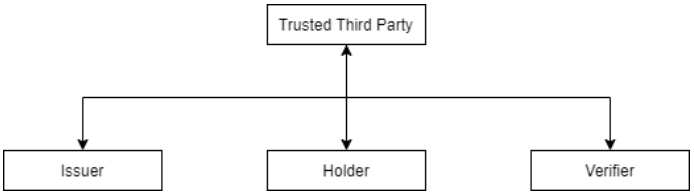

Fig. 10. Trusted Third Party

In the SSI systems the entities playing the roles of the issuer, holder and verifier are interconnected with one another and communicate with themselves on their own. When a trusted third party is involved in the SSI structure then the communication between the issuer, holder and verifier happens via the TTP rather then among themselves. The TTP acts as a regulator for communication and transfer of data/information such as VC/VP. The issuer will create the policies and send it to the TTP. The TTP will now take the responsibility to implement the policies and keep the verifier in check. When the verifier requests for VC from the holder then the TTP will ask for the holder to provide the information only if the verifier abides by the issuer policies, else the request will be disregarded. All the requests will be made to the TTP and the TTP will implement the required actions. The TTP will hold record of all the requests made and which were approved and which were not, basically everything that happens within the SSI infra-structure. The overhead is that TTP could act as the single point of failure, and too much power and responsibility lies with one entity. This structure is more or less like the traditional set-up where a single authority acts as the major power and other entities trust it to act just and right.

*3) **Smart Contract**:* Smart Contract was initially the idea of Nick Szabo. He has mentioned about the concept of using smart contracts in his paper "Formalising and Securing Relationships on Public Networks [32]", later sparked popularity with blockchain. Contracts in the real world is an agreement made between two or more entities to legally bind by the actions stated in the contract. Smart contracts are the same but are executed digitally. When the conditions predefined in the smart contract are met then the conditional clauses set to trigger further actions will be executed [40]. Smart contracts also overcomes the shortcomings such as sophisticated, incentive compatible (rational) breach [32]. The life cycle of a smart contract are creation, deployment, execution, and completion [40]. The issues with smart contracts such as codifying issues, security issues, privacy issues and performance issues and their solution are stated in [2], which is obtained from various other papers about smart contract.

Smart contract works on top of blockchain. Meaning that when a block is being created the contract is embedded in the block. Depending on the type of smart contract whether it is deterministic or not [15] the actions that trigger the execution of the smart contract will change. Initial deployment of smart contract happens like how a transfer of cryptocurrency occurs in blockchain. Transactions from the wallet to the blockchain marks the deployment of the smart contract [28]. This transaction from the wallet to the blockchain includes the details of the smart contract such as the compilation code and the addresses of the receiver when the actions of the smart contract are executed [28]. Like how a crypto transaction must be included in the block of the blockchain to be considered executed or done in the same way the transaction of the smart contract must be included in the block of the blockchain. After which the code present in the smart contract will be executed to mark the initial state of the smart contract [28]. Once the deployment of smart contract goes to the block it can not be updated since blockchains are immutable in nature. Communication to the smart contract now happens via transactions to the blocks in the blockchain. Such communications can result in the execution of the commands in the smart contract or other transnational exchanges.

There are two possible ways to go about to implement the issuer policy using smart contract. One is to use the already existing blockchain networks like Ethereum and plant the smart contracts on existing running networks. This will come with a few draw backs like the whole system will works under the security maintained by the blockchain community, the additional cost to be paid to the miners of the blockchain to have the smart contracts in block. Another way is to create a blockchain specifically for SSI and set up a third party to maintain the nodes and run the blockchain. This is similar to how some blockchains are created for specific purposes such as Factom, who have implemented blockchain to store digital healthcare records securely[27]. In both the cases the

issuer will issue the policy and then policy will be converted from natural language to machine readable form by software specialists. This conversion process is out of the scope of this research and left for future development. Once the policy is converted to code and deployed in the block further changes that issuer policy can not be made since blockchains are immutable in nature. The code will be written in such a way that when the verifier requests for the holders credential the contract will be invoked and the verifier will be checked for authority over the requested credentials. In case the verifier satisfies the issuer policy the smart contract will be executed and the requested holder's credential will be presented to the verifier, if not then the smart contract will not be executed and the verifier will not get the holder's credentials.

*4) **Attribute Based Credentials**:* Attribute based credentials (ABC) makes use of the cryptography policy zero-knowledge proof. ABC enables an entity to reveal information about oneself in the form of attributes without having to reveal much further information about oneself with the help of zero knowledge proof. It is a form of an authentication mechanism, where the attributes can be used as an authentication method. Simply put, credentials obtained from the attributes of an entity are defined as attribute based credentials [5]. Example of an ABC implemented system is IRMA [35], Idemix [7], U-prove [34]. ABC must provides three properties when implemented properly namely confidentiality, security and unlikability of the attributes/data used.

ABC systems have different entities playing the roles of issuer, users, verifier, revocation authorities and inspector. A trusted authority is used to generate the public and private parameters used by the other entities in the system. The issuer is the one issuing the credentials and ensuring the information in the credential is accurate and correct. The credentials are issued in an anonymous way. Users are the entities for whom the credentials are created for.The verifier are the entities that protect resources, information and services. Revocation authority is the entity that revokes the issued credentials. Inspector is another entity that de-anonymize the user under specific situation [10]. Both the revocation authority and the inspector are not mandatory part of the ABC system. The phases of an ABC system is as follows as stated in [10]:

- Set-up: The process which is performed only once by each entity in the system. The trusted authority generates all the public and private parameters that is to be used by the entities of the system. After this phase the issuer is ready to issue credentials and the verifier is ready to validate the credentials.
- Issuance: The process in which the issuer can issue credentials to the user without having any relevance to the previously owned credentials by the user.
- Presentation: The process in which the user present the credentials to the verifier upon request by the verifier.
- Revocation: The process in which the revocation authority revokes the credentials given to the user and also makes the updated revocation information available.
- Inspection: The process that is done to perform de-anonymization of credentials from the user. This process is usually carried out by multiple entities.

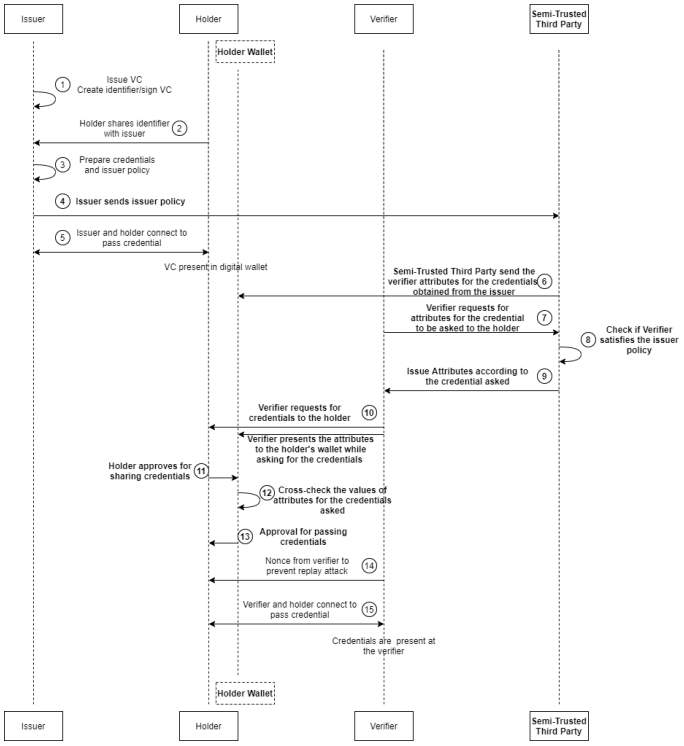

Fig. 11. System Flow with Attribute based Credential solution in place

Figure 11 describes the flow of SSI systems with the use of attributes to enable the working of issuer policies. The steps in bold in Figure 11 indicated that these steps are different from the procedures/steps followed in Figure 1. Semi-trusted third party(STTP) is an additional entity in the SSI system. Reason for using STTP is that no single entity in the system will gain more power over another and mutual trust is enabled. The STTP entity is similar to the TTP but then this third entity is not fully trusted by the other entities. STTP is used for the purpose of keeping track of the issuer policy and issuing attributes for the verifier. The STTP is also used in the system to aid with checking if the verifier checks the issuer policy stated. Since the check is done by the STTP the issuer will not gain more than needed information on the activities of the holder. The attributes are issued according to the issuer policy, the credentials that need to be accessed and the functionalities performed by the verifier with the credentials of the holder. The purpose of holder wallet is to check if the attributes presented to access the credentials of the holder are provided by the verifier. The holder's wallet acts as a barrier from which the verifier can not forcefully ask the holder to handover the credentials unless the required attributes are presented by the verifier.

Like Figure 1 the procedure of issuing the credentials to the holder is carried out by the issuer. Additionally in step 3 the issuer creates the issuer policy and sends it to the STTP, before passing the credentials to the holder's digital wallet in

Figure 11. After receiving the issuer policy the STTP now sends the verifier attributes to the holder for the credentials received by the holder from the issuer in step 6. After this in step 7 the verifier will now request the STTP for its attributes for the credentials it will be asking from the holder. In step 8 the STTP checks if the verifier satisfies the issuer policy, if yes then the attributes are sent to the verifier in step 9. With these attributes the verifier now requested for credentials from the holder in step 10. The verifier's attributes are presented to the holder's wallet. In step 11 the holder will have to approve sharing of the request credentials with the verifier then the holder's wallet will cross-check the attributes presented to it by the verifier with the one provided to it by the STTP before in step 12. If they match the holder's wallet approves the transfer of credentials in step 13. Else the transfer is discarded. In step 14 the verifier sends nonce value to the holder to ensure replay attacks do not happen. After which the holder and the verifier connect to pass credentials in step 15. The requested credentials are present with the verifier.

*5) **Two Layer Encryption**:* As the name suggested the data is encrypted twice. The data referred here is the credentials issued by the issuer to the holder. As stated in Figure 1 the whole process will remain as it is but when the credentials are prepared in step 3 it will be encrypted twice by the issuer. The first layer of encryption will be decoded only by the verifier. The second layer of encryption will be decoded by the holder. In step 4 of Figure 1 the issuer will give the double encrypted credential to the holder. The second layer of encryption is to ensure that no one other than the intended holder can get the credentials and be able to decrypt it and send it to the verifier. When the verifier requests for the credentials in step 5 the holder hands over the single layer encrypted credential in step 8. Now the verifier will have to request to the issuer to issue the decryption key. The issuer will now be able to check if the verifier matches with the issuer policy generated for the credential to be viewed.

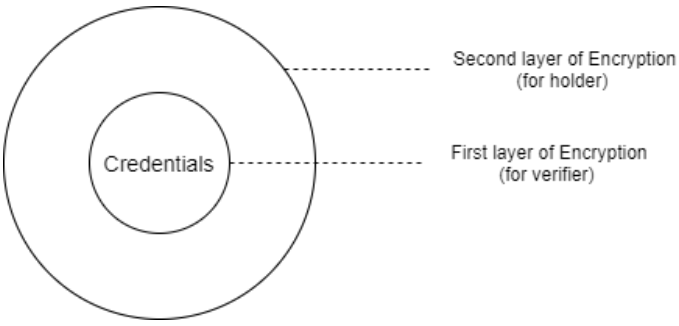

Fig. 12. Two Layer Encryption

Asymmetric cryptography[26] will be used for the encryption and decryption process. The generation of the keys will be done by the issuer. In this solution we assume the issuer to be a trusted entity. The idea of two layer encryption was inspired from multiple encryption concept[16].

*6) **Hybrid Solution**:* Hybrid solution is where two or more of the above mentioned solution will be combined together to attain better results and output for implementing the issuer policy. This is done so as to overcome the shortcomings present in the possible solutions. For example combining Shamir Secret Sharing with Smart Contract. When using Shamir Secret Sharing as the solution the entity playing the role of the issuer will be able to know a lot of information about the activities of the holder. Since the verifier has to ask every time to the issuer to view the requested credentials of the holder, the actions of the holder can be easily known to the issuer. There is also the possibility of the issuer to collude with the verifier and give the issuers part of the secret even though the verifier does not satisfy the issuer policy. To over come these shortcoming smart contract could be used. When creating the contract the issuer part of the secret can be put inside the block. The verifier will get that part of the secret when the condition if the issuer policy in the smart contract is satisfied and the code gets executed.