A trust module for the interaction with virtual characters

Luca Boldrin
InfoCert
Rome, Italy

Vanesa Daza
Un. Pompeu Fabra
Barcelona, Spain

Roberto De Prisco
Un. of Salerno and eTuitus
Fisciano, SA, Italy

Sergi Rovira
Un. Pompeu Fabra
Barcelona, USA

Saverio Sivo
eTuitus
Fisciano, SA, Italy

***Abstract*—This paper describes the architecture of a trust module for the interaction between human users and virtual characters. The module is part of a bigger project whose goal is that of creating virtual and truly realistic characters capable of interacting with human users. An example of such a virtual character is a computer rendered museum clerk that interacts with visitors. The role of the security module in the envisioned scenarios is that of providing trust, mainly regarding the identity of the human user, but also about the "identity" of the virtual character. In this paper we design (and describe the implementation) of an architecture in which human users identities are checked through face recognition and handled with verifiable credentials managed through a public blockchain. The project has been funded by the European Union and is under realization.**

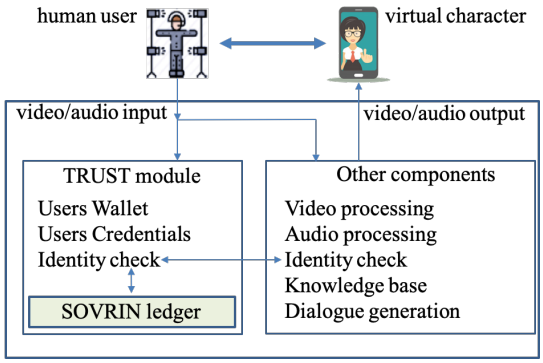

Fig. 1. Trust module within the overall system

I. Introduction

The goal of the overall project is to create sentient agents by combining features of Intelligent Personal Assistants and Embodied Conversational Agents, adding visual photorealism, believable animations, and socially aware and reactive communication to the resulting virtual character.

The virtual characters that are being developed in the project will be both highly realistic and extremely flexible in terms of functionalities. For example, the characters will be able to reproduce a predefined script with high fidelity, incorporating emotions, expressed through human-like facial expressions and gestures. Also, they will be able to interact with the user realistically, making the experience as close to a real one as possible, with current state-of-the-art techniques. Beside the aspect of rendering them very much human like, the virtual characters will be able to interact with human users to complete real actions, such as booking hotel rooms or making purchases.

It is clear that these functionalities are very convenient, but they pose a security threat to the users if they are badly implemented. Therefore, it is of paramount importance to put in place a secure and efficient framework to validate the identity of both the user and the virtual character during their interactions. Achieving this is the main purpose of the security module that we describe in this paper.

Although this paper focuses exclusively on the security module, it is important to keep in mind that it has been designed for the specific overall project we just described and that there are many other components with which the security module has to interact. Figure 1 gives a rough idea of the modules of the project and their interaction with the security module. The picture emphasizes also the fact that the security module is built upon the Sovrin [2] public ledger.

The scenarios envisioned in the overall project call for a highly distributed system built upon existing infrastructures. For example, virtual characters might be created by the management of a hotel chain for the reception of customers in every branch of the chain. Or can be created to offer online services, such as bank services or similar. Human users can thus interact with virtual characters in a variety of ways and the interactions are inherently distributed. We thus assume that there is a network layer (the Internet) that interconnects all the actors of the overall systems.

The problem faced by the security module is that of checking the identity of the human users that access the service and also the "virtual identity" of the virtual characters. At the core of the solution are the concept of verifiable credential and existing, publicly usable infrastructures for their management. In this paper we describe the architecture we have designed to solve the problem and an implementation.

The paper is organized as follows. In Section II we briefly describe the concept of Self-Sovereign Identity systems, verifiable credentials and the facial recognition approach that we use to check the identity of human users. Then, in Section III we provide an overall description of the security module, describing the basic authentication modes. Section IV provides details about the flow of interactions needed to perform the authentication checks. Then in Section V we provide a brief description of the open source technologies available and in

Section VI we provide a description of the implementation. Finally, in Section VII we provide some concluding remarks.

II. Preliminaries

Verifiable credentials and their handling through the Self-Sovereign Identity paradigm are crucial for our security module. Thus in this section we briefly recall them. We also mention the face recognition techniques that we use.

A. Verifiable credentials

Verifiable credentials are the digital analogs of any physical document that can attest the identity of an individual such as identity cards, passports or driving licenses. Beside providing the same functionality as their physical counterparts, verifiable credentials can be more tamper-evident and trustworthy, exploiting, for example, by using digital signatures.

There are three main actors in an ecosystem where verifiable credentials are used:

- Holder: An entity that possesses one or more verifiable credentials.
- Issuer: An entity that asserts claims about one or more subjects and creates a verifiable credential from these claims. It also transmits the credential to the corresponding holder. Governments, corporations and non-profit organizations are some examples of issuers.
- Verifier: An entity that receives one or more verifiable credentials for processing. Examples of verifiers are employers and websites.

There are two other actors that play an important role when implementing verifiable credentials:

- Subject: Any entity about which claims are made. Clearly, in most cases the holder of a verifiable credential is also the subject but there are exceptions. For example, parents and pet owners might hold the credentials of their children or their pet.
- Verifiable data registry: Any system that mediates the creation and verification of relevant information such as revocation registries and issuer public keys which is needed to use verifiable credentials. Examples are government ID databases and distributed ledgers.

In our project, users and virtual characters will have their own verifiable credentials, becoming holders of the system. These credentials will be managed through the Sovrin network, thus, our approach follow the SSI paradigm. Notice that Sovrin will provide the registry infrastructure. Issuers can be external entities. The verifiers will be both the users and virtual characters, that is, the users will use the verifiable credentials of the virtual characters to assert their identity and vice versa. For further information see [4].

B. Self-Sovereign identity systems

Nowadays, we have a digital identity for every task that we need to perform over the internet. These identities are managed by centralized systems and the users do not have any real control over them. The paradigm of Self-Sovereign Identity (SSI) was created to change this. It allows to create digital identities that are not managed by a central authority and that last as long as the user wants. Moreover it allows the user to have complete control over the information that he or she discloses each time that his or her identity is needed. A central component to build an SSI system is that of a verifiable credentials, discussed in the next subsection.

This idea of SSI has been around for decades but its implementation has only become possible after the introduction of the Blockchain technology which started with the "Bitcoin" era [1]. And it is now becoming a central pillar in the EU identity ecosystem [15]. We use the Sovrin network to manage the identities of the users and virtual characters of the system. For further information about Sovrin and SSI see, for example, [2], [3].

C. Biometric recognition

To validate the identity of a user we have chosen to use deep learning based facial recognition. Currently, the best facial recognition results are obtained from Deep Convolutional Neural Networks (DCNN). The software that we use (called Face Matching) uses two of these networks. One network (NN1) is trained to detect faces in a given image and the other (NN2) uses Additive Angular Margin Loss (ArcFace) to obtain highly discriminative features representing each of the detected faces. A detailed explanation of this technique can be found in [6]. Also, the reader can find all the necessary information on deep learning in [5].

We remark that it is possible to use also additional biometric recognitions, such as voice recognition or fingerprint recognition, and they might be used both as alternatives or in conjunction. The functioning of the security module does not change.

III. Security Module: Overall description

The security module implements authentication of the players. The players of the overall project are human users, that access services, and virtual characters, that offer services. Both human users and virtual characters are equipped with digital identities. For a human user, the digital identity is a picture of the user, together with some secret information; for virtual characters, the digital identity will consist only of some secret information. The digital identities are stored in so called (digital) wallets. The digital identity is acquired in a registration phase. Several types of security levels are possible for the registration phase, depending on how strong the authentication needs to be. The security module that we design can handle any level; in other words there is no difference in the identification check. In a real application we expect that the users will obtain digital identity credentials from third parties, e.g. Certification Authorities. For the virtual characters, the digital identity will be created by the security module upon specific requests for the various use-cases being built in the project.

Following the Self-Sovereign paradigm, the digital identity of the user is kept in a private wallet. Hence the users will have to manage the wallet. To this end, the security module

Fig. 2. Single authentication

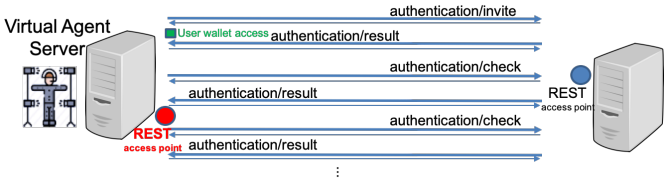

Fig. 3. Single authentication

comprises a component which is a mobile app through which the user can manage the wallet and at the same time can interact with the security module. For the rest of the paper, we will refer to this mobile app as the Mobile Wallet App, or MWA for short. Virtual characters are not envisaged in the Self-Sovereign paradigm. So, we need to somehow handle their "identities". In order to manage the identity of the virtual character in a manner similar to that of human users we will identify each virtual character with a pair of public-secret keys and store the public key in the wallet of the user. The public key will be the "identity" of the virtual character, that only the virtual agent can "validate" using its secret key.

The authentication services offered by the security module can work in a single-check mode or in a continuous mode. In the single-check mode, the authentication check is made only once. In the continuous mode, after the initial authentication, which is the same as the single-check authentication, the virtual characters can continuously make authentication requests to keep checking the identity of the human user currently interacting with them. These two modus operandi correspond to the two general real situations in which a real user needs to be authenticated only once (for example at the entrance of a site) or continuously (for example during a conversation).

The digital identity used for the authentication is the binding of a biometric feature, namely the face of the user, with the real name of the user. The wallet contains a verifiable credential with the face of the user. Such a picture is compared with the pictures, taken by the virtual character, of the person currently interacting with the virtual character. In the continuous authentication mode, the virtual character can keep taking pictures anytime, to repeat the check, for example at fixed time intervals. The first authentication, the one performed in the single-check authentication, involves access to the wallet of the user. This access has to be approved by the user: the app that manages the wallet asks permission to access the data (the data is protected by a password that the user needs to type). Such an access will not be requested for the subsequent checks in the continuous mode.

The interaction between a virtual character and the security module are implemented by means of REST calls. The security module offers a REST access point, whose details are explained in the subsequent section of this document, which allow to request both the single-check authentication and the continuous authentication checks. The virtual agent will also need to offer a REST access point to receive the result of the authentication. Figure 2 summarizes the interaction between a virtual character and the security module, for the single-check authentication.

Similarly, the continuous authentication starts with a single-check authentication, which requires access to the user's wallet, and then continues with authentication checks that do not require access to the user wallet: to avoid such an access we make the check against the latest authenticated picture. Figure 3 summarizes the interaction between a virtual character and the security module, for the continuous authentication.

In the following sections we will provide technical details about the implementation of the security authentication checks described above. These authentication checks require a secure management of verifiable credentials (the digital identity cards) and this is achieved by using existing paradigms, infrastructures and open source projects as detailed in the following.

IV. Authentication process

In this section we proceed to explain in detail the authentication process between a virtual character and a user. The interaction between two virtual characters is out of scope for this paper (and for the overall project).

Before the authentication process can begin, the user and the virtual character need to obtain a verifiable credential. That is, we need a registration step in which such credentials are created. After the credentials are granted, the user and the virtual character can start interacting with each other.

The authentication process starts with what we call initial authentication. At this stage, the virtual character takes a picture of the user and compares it with the front-facing image of the user using the face matching software. If the comparison is correct it is possible to proceed, otherwise, the interaction is terminated.

For the continuous authentication stage, the system, upon request of the virtual character, takes new pictures of the user and compares it with the latest authenticated picture. This is done for different reasons:

- Using the image that is stored in the credential of the user requires acquiring a permission from the user. This is a huge drawback in terms of usability, therefore we use this image once.
- The conditions under which the image of the user was taken could differ greatly from that of the images taken by the agent leading to an increase in the number of false negatives.

For the continuous authentication check we need also to decide when to perform the subsequent checks. Since this decision depends on the application the event that trigger a

check are determined by other components of the system (the ones providing the service) and not by the security module.

We now proceed to explain each step of the authentication process in much more detail.

A. Registration

The registration of users and virtual characters to the system is the first step of the authentication process. What this step achieves is to deliver to each party a verifiable credential that will be the key component used for authentication. Let us detail the registration process for users and for virtual characters.

Users. The credential to prove the identity of a user will contain the following information:

- Personal data: name, surname and date of birth.
- Biometric data: biometric signature of the face of the user (a front-facing image).

All this information will be stored in his or her local wallet. We do not restrict the level of the certification, that is the certification can be self-attested (lowest level) or anything else up to an identity verified by a strong authentication process, such as the one granted by a Certification Authority.

Virtual Characters. The identity of the virtual characters will consist of a pair of public and secret keys generated by the owner of the virtual character. The public key will be associated to the DID of the virtual agent and the secret key will be stored locally and kept hidden permanently by the agent.

B. Single (initial) authentication

Figure 4 shows the flow of interactions needed for the single authentication. This pictures shows 6 columns: the first one represents the human user, the fourth one, the security module and the last one the Sovrin ledger. The other three, are extra components that are needed to integrate the service offered by the Sovrin public ledger. The mediator component is needed to integrate the Mobile Wallet App, the virtual agent is needed to connect the virtual character and the virtual agent is needed to allow the security module to interact with the Sovrin ledger. In the following we will use the expression "virtual agent" to indicate the component that acts on behalf the virtual character. To a rough approximation virtual characters and virtual agents are the same thing, but to a fine distinction the virtual character is the one created to interact with the human user of the services being offered, while the virtual agent is the software that handles the necessary steps needed to implement the authentication check on behalf of the virtual character. Because of this, in the following we may misuse the two expressions considering them synonymous. We remark that the expression "virtual agent" comes from the Sovrin jargon.

When the session starts, the virtual agent initiates the authentication process. We will refer to this step as authentication request. This leads to a proof request where the agent asks the user for the user credentials (as discussed in the previous section). The identity of the virtual agent is added to this request for validation from the user. From his or her side, the user will validate the identity of the virtual agent by using its public key to encrypt some random string and asking the agent to give a correct decryption of the corresponding ciphertext.

All the needed information will be encoded in a QR code that the user will scan with a smartphone. This allows the user to accomplish two actions: validate the identity of the virtual character (which is stored in the virtual agent wallet in the cloud) and give authorization to the virtual agent to deliver his/her biometric credential to the virtual agent (which is stored in the wallet on the smart device of the user).

Once the virtual agent has at its disposal the biometric information of the user (at the moment we use a front-facing image, but alternatives also used in synergy are possible) it can proceed to make an initial assessment of the authenticity of the credentials provided by the user. The virtual agent will use a camera to capture a picture of the user (who will be asked in advance to be facing the camera at all times during the interaction) and will make an API call to the face recognition software to make the matching. If the matching is correct, the initial validation is completed.

C. Continuous authentication

After a first successful check of the identity from both the virtual agent and the user, the continuous authentication phase keeps making authentication checks. In this part of the interaction, the virtual agent will use the image that it took during the previous phase as the user's biometric information and will iterate the previous process until either party interrupts the interaction. More precisely, the virtual agent will take a picture of the user and make a call to the security module to check the match between the two pictures. If the match is successful, the agent will wait a predetermined time until taking the next image. If the match is not successful, the agent will close the interaction. To prevent abrupt service interruption, depending on the application, the termination of the interaction might be delayed to a number of consecutive unsuccessful matches. Figure 5 shows this process.

V. Technologies

As we have said in previous sections, we build on the self-sovereign identity paradigm which is based on distributed ledgers and blockchains. We have identified Sovrin as a useful framework. Hence the Sovrin framework is the base for building the security module. Over the last years, a number of open source projects have provided several implementations of the Sovrin Self-Sovereign identity infrastructure. There have been also efforts to standardize the approach and make the available components able to interoperate. Some of these projects focus more on implementations, some more on higher levels aspects, such as protocols and standardization. We have evaluated the available possibilities and we have focused our attention on the following open source Hyperledger projects: (i) URSA, that will be used for its ZKP-capable W3C verifiable credentials primitives; (ii) INDY, that will be used for its Decentralized Key Management System (DKMS) primitives; (iii) ARIES, which provides other libraries and also standardization.

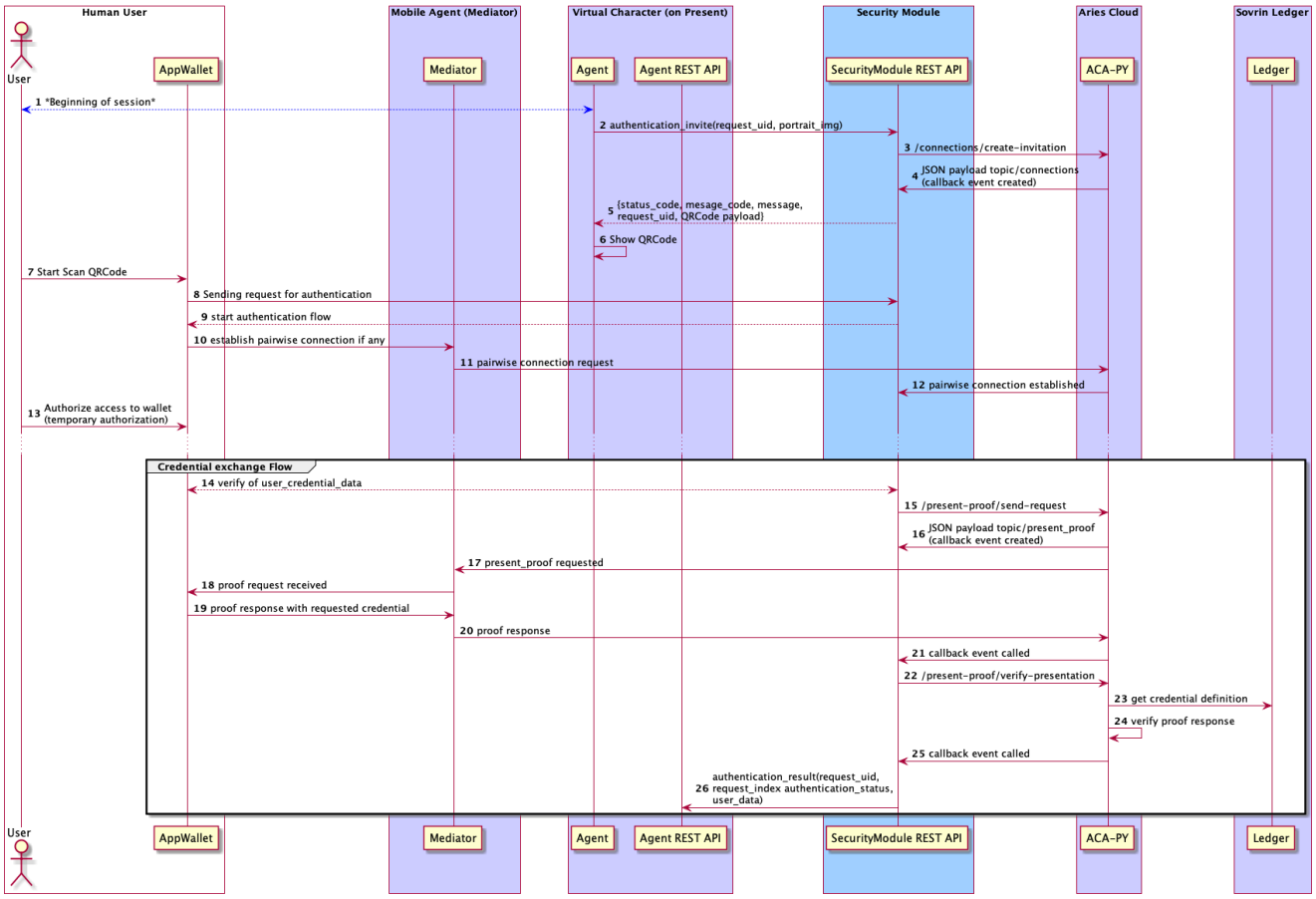

Fig. 4. Single authentication flow

All of them are under the umbrella of Hyperledger [7], which is an open source community that fosters development of open source code to help implement specific applications over the public infrastructure. Hyperledger Aries [8] provides tools to store, transmit and verify digital credentials. In turn, the Aries library, exploits the cryptographic support offered by the Ursa [9] and Indy [10] projects. The Aries infrastructure offers:

- a blockchain interface layer (known as a resolver) for creating and signing blockchain transactions;
- libraries to implement cryptographic wallets for secure storage of cryptographic secrets and other information;
- an encrypted messaging system for off-ledger interactions between clients using multiple transport protocols;
- an implementation of ZKP-capable W3C verifiable credentials using the ZKP primitives found (this is in the Ursa library);
- an implementation of the Decentralized Key Management System (DKMS) specification currently being incubated in Hyperledger;
- a mechanism to build higher-level protocols and API-like use cases based on the secure messaging functionality described earlier.

Our secure module is built upon the Aries infrastructure. In order to exploit the tools offered by the Aries infrastructure we need to develop components able, on one hand, to exploit such tools and, on the other hand, to interact with the other components of the project, implementing all the needed functionalities. Based on the role of each software component we will have an Aries component to integrate. The piece of software for such an integration is called a “controller”. One of the most important components is the Aries agent. An Aries agent is able to establish connections with other Aries agents, exchange messages between connected Aries agents, send notifications about protocol events to a controller and expose an API for responses from the controller with instructions in handling protocol events.

The specific Aries Agent implementation we have chosen is the Aries Cloud Agent, named ACA-Py for short and implemented in Python. The source code is available in a GitHub repository [11]. This agent implementation embeds the Indy Software Development Kit, a SDK already used in many production deployments. The Aries component used to manage/control an Aries Agent is named “Controller”. The Aries Cloud Agent and a controller run together, communicating using asynchronous, stateful applicative level protocols named Aries protocols. For the implementation of our controller we

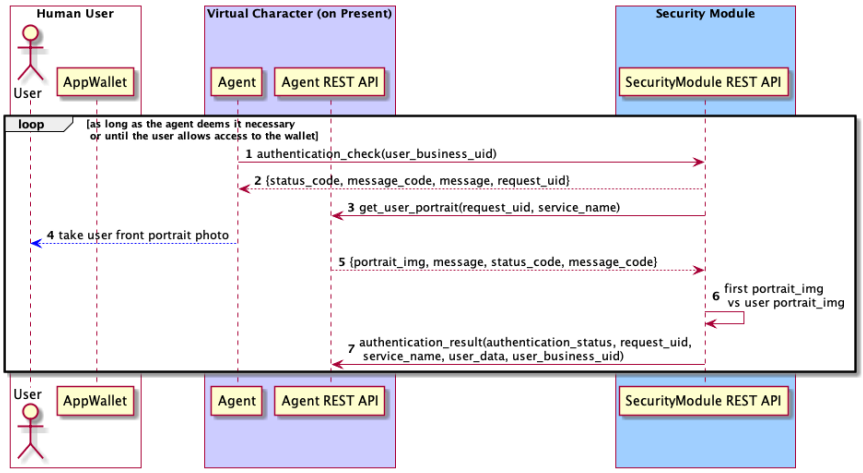

Fig. 5. Continuous authentication

used the Python Framework Django. Beside the controller, another component that we need is the MWA (Mobile Wallet App) which will manage the user's SSI wallet and that acts as an Aries Controller mobile side. The mobile app uses a special Aries agent running on the mobile device, agents running mobile side are called "edge agents", in the Sovrin jargon. To develop the edge agent we used the library VCX, whose source code is available in a GitHub repository [12]. VCX, a wrapper on Libindy, is suitable for mobile development both on Android and IOS OS. To communicate with an edge agent, the Aries architecture requires another special server side agent named mediator. For the mediator agent, we used a Node AriesVCX Agency [13] (absaOSS). The Mobile Wallet Application has been developed using the Flutter [14] framework. Flutter is an open-source framework, created by Google, that allows to develop cross-platform mobile applications. Mobile applications written with Flutter are compiled both for Android and iOS.

VI. Implementation

As we have already said, the foundation of the security module is given by the Sovrin public ledger, and we use as building blocks some open source projects (Aries, Indy, AbsaOSS). These open source projects offer basic functionalities for the management of the credentials: applications, such as our security module, need to implement several pieces of code to have a full functioning service. Figure 6 summarizes the software components of the security module. Some of these components are the ones offered by the open source projects that we have mentioned earlier. Some others are those that need to be developed to implement the security module.

A. Open source components

1) Aries Cloud Agent (ACA-Py): This software component is the interface between the security module and the Sovrin blockchain used to manage the digital identities. It will be connected to the core part of the security module, that is the

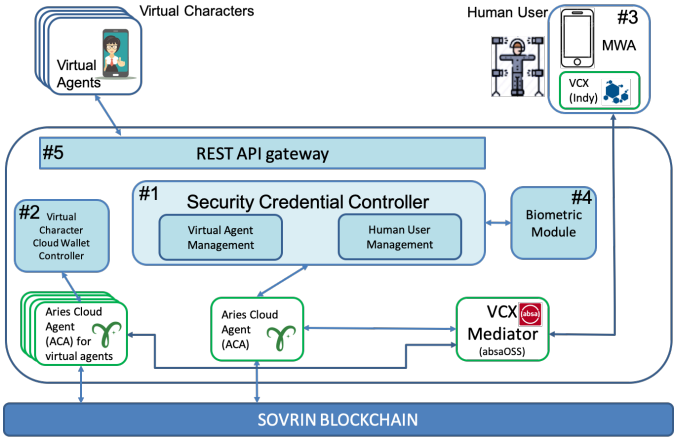

Fig. 6. Software components

Security Credential Controller, but it is needed also for the cloud wallet of the virtual agents.

Indy (and its underlying Ursa cryptography) enables issuers, holders and verifiers to exchange verifiable credentials and presentations by exchanging sequences of messages. All those issuers, holders and verifiers will use agents (software) that can be built by many different organizations — open source tools, commercial agents available for purchase, and custom-built proprietary agents. Since there are other verifiable credential ecosystems being developed, in the Aries community, in parallel to Hyperledger Indy, Aries is also intended to be verifiable credential-agnostic and multiledger — some Aries implementations are becoming able to support different verifiable credential implementations.

The Hyperledger Aries standardizes the operations on verifiable credentials through the Aries Cloud Agents (ACA). So, in order to be compliant with the Aries standard we need to access the functionalities of the infrastructure through the Aries Cloud Agents (ACAs). In the jargon of Aries, the software components that interact with the ACAs, are called

controllers. In our case we need to develop the Security Credential Controller and also the Virtual Agent Controller that will manage the wallet of the virtual agent.

Messaging between Aries agents is defined at several conceptual layers. At the lowest level, it is just the ability for one agent to send a chunk of data, and for another agent to receive that data. In the upper layer, there is the ability for agents to exchange a sequence of messages to accomplish some shared task. That's a protocol. Aries messaging is based on the DIDComm (DID Communications) protocol. As indicated by the name, DIDComm uses the data associated with DIDs, that is public keys and service endpoints, to, respectively, encrypt and route messages. The controller can be built embedding the ACA-Py functionalities as a library. The controller uses the library to drive the agent according to the needs of our application. In particular we are using the ACA-Py implementation available in a public GitHub repository . The ACA-Py implementation is suitable for all non-mobile agent applications, separates the controller and agent into processes communicating using an HTTP REST API. The controller receives web-hook notifications from the agent as events occur (e.g. messages are received from other agents) and uses an HTTP API exposed by the agent to respond to those events or to initiate new actions. With this approach, a web controller is similar to any modern web app, sending requests and receiving messages over HTTP to an agent service. Although the ACA-Py is written in Python, the controller is by design completely independent of the agent and thus can be written in any language. A controller could even be an existing enterprise application, extended to use verifiable credentials by adding API calls to an Aries agent process. Our specific controller will be described in the next section.

The ACA-Py provides all of the core Aries functionality such as interacting with other agents and the ledger, managing secure storage, sending event notifications to, and receiving instructions from the controller. The controller executes the logic of the application and this determines the behavior of the agent, that is how it responds to the events it receives, and when it initiates events.

2) VCX Mediator – based on AbsaOSS: The Mediator component is necessary in order to securely deliver messages from one edge agent to another, because mobile agents do not have an endpoint (a physical address) that other agents can use for sending messages. Thus, it is impossible for mobile agents and enterprise agents to send messages to each other directly. In DIDComm, the Mediator is used to manage the list of agents through which the messages will be routed. If there is no list of agents, the message will go directly to the receiver. For each mediator, the sender, explicitly adds another envelope, another layer of encryption and a "To" address.

In our security module, this software component is responsible for the communication between the user wallet application and the various Aries Cloud Agent deployed.

Mobile wallets are not online at all times, and are not constantly polling to see if they have any incoming messages (that consumes resources, particularly data and battery, on the phone), the mediator provides a queue to hold messages until the mobile agent requests them. The mediator will use the mobile OS (iOS or Android) notification mechanism to let the user know when a message arrives in the queue, triggering a check with the mediator.

The specific Mediator component that we use is the Mediator developed by Absa and called AbsaOSS.

3) VCX Indy – part of the mobile app: The mobile wallet app needs to manage the identity of the users. To this end we use the VCX library. This library is a C-callable library built on top of Libindy that provides a high-level credential exchange protocol. It simplifies creation of agent applications and provides a better agent-2-agent interoperability.

There are two communication methods: proprietary and Aries. The mobile wallet app will use the Aries method in order to interact, using DIDcomm messaging, with other Aries agents of the security module.

The VCX library is written in Rust. It provides many wrappers for many programming languages. Among these there are also the wrappers for the mobile platform that will be used for the Wallet Application development.

The mobile wallet app will communicate with other agents using a VCX mediator.

B. Code developed

In order to integrate the open source code with the specific application we are building, we needed to implement several software components. We have split the needed code into 5 pieces, which are (see Figure 6):

1) *A Security Credential Controller.* This controller is in charge of interacting with the Aries agents both to issue a credential and to verify a credential. In typical SSI jargon, it acts both as "Issuer" and as "Verifier".
2) *A Virtual Agent Identity Controller.* This is needed to manage the (virtual) wallet of the virtual agents; it acts a SSI holder. As we have explained in other parts of the project this wallet should be part of the virtual agent, but to ease the implementation of use cases we incorporated the management of the identities of virtual agents within the security module.
3) *A Human User Identity Controller (Mobile Wallet App).* This is needed to manage the identities of the human users; it acts as an SSI holder. The mobile app includes the management of the wallet of the user. It includes also the code for creating the self-issued credentials.
4) *Biometric Module.* This module is the one that performs the comparison of two faces that is needed for the verification of the users' credentials.
5) *A REST API Gateway.* This will be used for communication with the virtual agent as described in other parts of this document. In later sections, we will delve into more details for each of these components. Before that we review the flow of interactions between the components.

C. Virtual Character Cloud Wallet

The virtual agent wallet will be maintained in the cloud and the agents will access them through specific REST API. The cloud wallet is managed through a controller which is part of the security module (see Figure 6). This component allows the virtual agent to store his own credentials and to communicate with the secure module to create a "credential proof" to prove that he has a specific credential.

D. Mobile Wallet App

Users will be able to manage their identity credentials using a mobile wallet app. With the app, the user will be able to collect one or more identity credentials and will have the opportunity to prove that he has certain credentials by responding to the required proofs. The app is being developed to be compliant to the philosophy of Self Sovereign Identity and respecting the user's privacy. The contents of the credentials will be accessible only with the explicit consent of the user, e.g. the user will have to enter a secret password or scan his fingerprint to allow access to the credentials.

E. Face recognition module

The biometric module is responsible for the face matching of users' pictures. The basic functionality of the module is the comparison between front-facing images of human faces. The result of the comparison is a similarity value. The Biometric AI-Face-Matching solution uses machine learning to identify the faces, analyze the characteristics and finally compare them. The basic light/dark flow allows to find a basic pattern in the face image. The comparison algorithm uses a neural network trained to generate 128 measurements for each face. The algorithm is able to generate these measurements from a face that has never processed in a few milliseconds and turn it into a vector. The reliability of the similarity value returned depends on the context and the quality of the images.

The response contains both a "match" result, which is true for a match and false for a mismatch, and in case of a match it also contains the similarity value, which is a number between 75 and 100. For smaller similarities values we have a mismatch. Notice that, depending on the use case, we can also look at the actual value of the similarity and decide to accept or deny authentication directly on the similarity value. The use of higher thresholds allows for a stronger face recognition but might create more false negatives.

The biometric module also allows the creation of a biometric "signature" that can be used by the security module to create the user's biometric credential. The biometric signature is just an encoding of a picture of the face of the user.

VII. Conclusions

In this paper we have presented the design of a security module whose purpose is that of checking identities of human users and virtual characters. The current implementation is based on face recognition. However the design allows for easy extension to additional identity checks, such as voice recognition: the flow for the checking is the same. This is a possible future extension.

Acknowledgment.

The security module described in this paper is part of a bigger project, currently under realization, funded by H2020 from the European Commission: PRESENT - Photoreal REaltime Sentient ENTity - H2020-ICT-2018-20/H2020-ICT-2018-3. Grant Agreement: 856879.

References

[1] Satoshi Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System", bitcoin.org/bitcoin.pdf.

[2] Sovrin library: https://sovrin.org/library/

[3] A. Satybaldy, M. Nowostawski, J. Ellingsen, "Self-Sovereign Identity Systems". In Proc. of Privacy and Identity Management. Data for Better Living: AI and Privacy. IFI AICT 576, pp. 447– 461, 2019. DOI:10.1007/978-3-030-42504-3_28.

[4] M. Sporny, D. Longley, D. Chadwick, "Verifiable Credentials Data Model 1.0 Expressing verifiable information on the Web". W3C Recommendation 19 November 2019. https://www.w3.org/TR/vc-data-model/.

[5] I. Goodfellow, Y. Bengio, A. Courville, "Deep Learning". MIT Press, 2016.

[6] J. Deng, J. Guo, N. Xue, S. Zafeiriou, "ArcFace: Additive Angular Margin Loss for Deep Face Recognition". In Proc. of IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019. doi:10.1109/cvpr.2019.00482.

[7] https://www.hyperledger.org/

[8] https://www.hyperledger.org/use/aries

[9] https://www.hyperledger.org/use/ursa

[10] https://www.hyperledger.org/use/hyperledger-indy

[11] https://github.com/hyperledger/aries-cloudagent-python

[12] https://github.com/hyperledger/aries-vcx

[13] https://github.com/AbsaOSS/vcxagencynode

[14] https://flutter.dev

[15] https://digital-strategy.ec.europa.eu/en/library/trusted-and-secure-european-e-id-regulation