D5.7 Agent-to-agent interaction protocol

Grant Agreement nr	856879
Project acronym	PRESENT
Project start date (duration)	September 1st 2019 (36 months)
Document due:	28/02/2022
Actual delivery date	28/02/2022
Leader	INFOCERT
Reply to	luca.boldrin@infocert.it
Document status	Submission version

Project funded by H2020 from the European Commission

Project ref. no.	856879
Project acronym	PRESENT
Project full title	**P**hotoreal **RE**altime **S**entient **ENT**ity
Document name	Agent-to-agent interaction protocol
Security (distribution level)	PU
Contractual date of delivery	28/02/2022
Actual date of delivery	28/02/2022
Deliverable name	D5.7 Agent-to-agent interaction protocol
Type	Report
Status & version	Submission Version
Number of pages	18
WP / Task responsible	InfoCert
Other contributors	eTuitus
Author(s)	Luca Boldrin, Romualdo Carbone, Roberto De Prisco
EC Project Officer	Ms. Diana MJASCHKOVA-PASCUAL Diana.MJASCHKOVA-PASCUAL@ec.europa.eu
Abstract	A formal study on the models and technology for the interaction between software agents.
Keywords	Decentralized identity, virtual agent identity, self sovereign identity, trusted communication protocols, machine-to-machine trust
Sent to peer reviewer	Yes
Peer review completed	Yes
Circulated to partners	No
Read by partners	No
Mgt. Board approval	No

Document History

Version and date	Reason for Change
Initial draft 13-05-2020	Outline, initial draft
Initial version 30-09-2021	Initial version to be read by others
Complete draft 10-11-2021	First complete draft
Final version 10-02-2022	Complete version including all content
Reviewed version 25-02-2022	Integration of reviewers' comments

Table of Contents

1. Abbreviations

API	Application Programming Interface
BLE	Bluetooth Low Energy
B2B	Business To Business
B2C	Business to Consumer
CA	Certification Authority
CRL	Certificate Revocation List
PKI	Pubic Key Infrastructure
RA	Registration Authority
SSI	Self-Sovereign Identity
TLS	Transport Layer Security

2. Abstract

This deliverable provides an analysis of the models and the technologies which are required for the trusted interaction between a virtual agent and a user, as described in D5.2 Trust and security infrastructure design.

Trusted digital interactions are not something new. They are already widely implemented on mainstream protocols (https, wss, ssmtp, etc.), which are mostly based on TLS supported by a PKI.

This deliverable illustrated the limitations of the current mainstream approaches with respect to PRESENT requirements. It also explains why PRESENT trust architecture builds on a different model which leverages on the concept of "agent" developed in the decentralized identity community.

In this community, **"agents" are active software entities which proxy either "real humans" or "virtual humans"** in the digital realm. However, since the term "agent" is already used within PRESENT to mean something different, i.e., a "virtual human", to avoid any confusion, we will use the term "trust agent" for our purpose.

This model adopted in PRESENT requires different protocols, on whose specification and standardization the community is actively working.

This deliverable analyses the ongoing work and identifies the appropriate matching to PRESENT trust requirements.

Additionally, this study also identifies available stacks and open-source implementations which are provided by the community.

3. Context

PRESENT communication involves humans and "virtual humans", i.e., machines. The general interaction happens according to the schema presented in D2.3 revised report on modular architecture, protocols, and APIs.

However, in order to this analysis, we may adopt the simplified representation of fig. 1. The interaction is layered, and includes a "real world", some "edge components", some "cloud components" and eventually a ledger offering some registry and notarization services.

It appears from the picture that the human (Me) may interact with a different real-world entity (e.g., a university) through a virtual human in different modes:

- Via an interface running on someone else's device (e.g., a totem representing a virtual clerk at the university hall)
- Via an interface running on my own device (e.g., a mobile phone), which represents my own agent (e.g., a general purpose app which I can use for interacting with many different parties)
- Via an interface running on my own device (e.g., a mobile phone), representing a different party (e.g an app provided by the university, which offers me some service)

Agents at the "edge" layer may interact between themselves (e.g., via BLE or other proximity protocols), but in typical scenarios they leverage on a corresponding service in cloud which hosts the real "trust agent".

Independently of the specific interaction scenario, our focus here in on the machine-machine interface which normally happens between trust agents in the cloud.

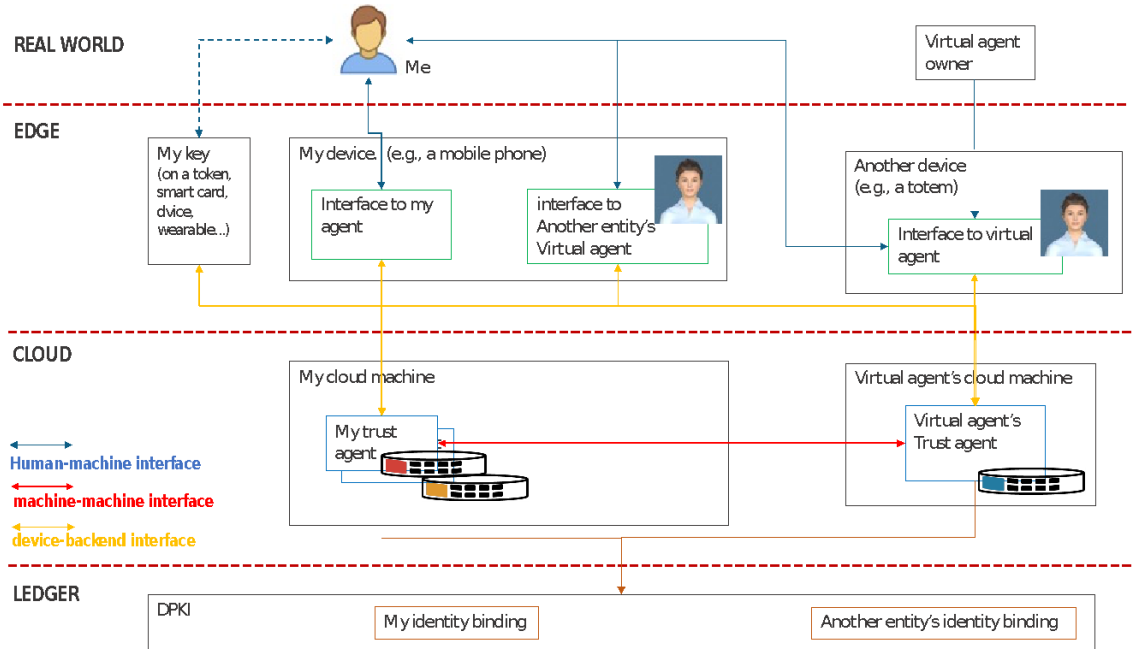

Figure 1: layered interaction schema

The concept of trust agent (normally referred to simply as "agent") is widely adopted in the Self-sovereign identity (SSI) community, one typical representation being illustrated in fig [1].

[1] From: https://manningbooks.medium.com/the-basic-building-blocks-of-ssi-381871e3f362

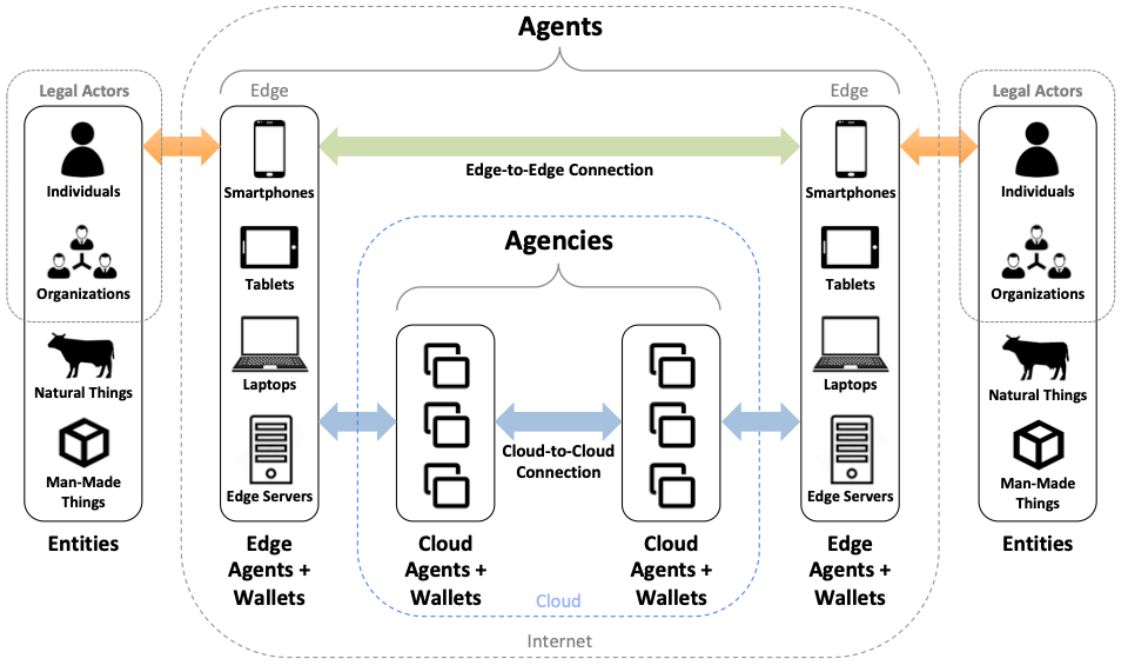

Figure 2: Agents in self-sovereign identity model

Since we are therefore dealing with machine-machine trust, we must note that this is not a novelty: since many years, trusted digital interactions are implemented on mainstream protocols (https, wss, ssmtp, etc.). These protocols, which are mostly based on TLS supported by a PKI, are essentially aimed at providing some confidence on the identity of the real-world parties which are interacting by means of some digital proxy.

Chapter 3 will present an overview of these traditional forms of trust establishment. Chapter 4 will provide some reasons on why to extend this model, and will introduce the requirements of trust in a decentralized setting. Chapter 5 will present the main initiatives aimed at fulfilling these new requirements, while Chapter 6 will illustrate some concrete implementations (some of which still in their infancy).

4. PKI based trusted interactions

Achieving trust and security when operating on the web is fundamental. The problem is not trivial due to the complexity of the interactions involved in the operations and the vulnerabilities that an attacker can exploit. Vulnerabilities arise at different levels of the overall system. Cryptographic tools and techniques help in offering web services security.

In this section, we provide a high level description of the problem and describe the basic tools and approaches used to achieve trust and security, with particular attention to PKIs (Public Key Infrastructures), since public key cryptography is the basis for authenticating parties over the web.

4.1. Web Services

The World Wide Web has been designed with the goal of sharing information. Over time, however, its functionalities grew a lot: today it is not just a distributed database of information, but rather it offers a great variety of services. Today we can buy goods, reserve a hotel room, buy and sell stocks, get an airplane ticket and much more. Each of these "functionalities" is often offered as a web service, since it is implemented using the HTTP protocol as the base of the communication between the server offering the service and the client accessing it.

We can differentiate between a B2C (Business-to-Consumer) service, in which the service is designed to be exploited by a real user (a person) and B2B (Business-to-Business) services, in which the services are designed to be accessed by an application.

Web services are mainly designed for B2B automatic interactions. This is important to offer functionalities that require interaction with several web services. For example, consider the case of a web service of a travel agency that offers a complete package including flights, hotels, and restaurants. The client accesses the web service of the travel agency which, to finalize the operation needs in turn to contact the web services of all the relevant flight companies (or perhaps an intermediate web service that will contact them), the web services of all the useful hotels (or again a single web service that will contact them), and similarly for the restaurants. There is a lot of interaction involved among all of these services while the customer only interacts with the travel agency. Protocols and standards have been designed to ease these interactions.

Due to the complex and heterogeneous nature of the systems that need to interact, web services involve a variety of standards and protocols, such as XML, SOAP, SAML, OAUTH, etc. It is, of course, important to achieve security for the services which, in general, means ensuring data privacy, data integrity, non-repudiation and authentication. For this reason, the standard and protocols involved make use of cryptographic techniques. Encryption and digital signatures are the basis for achieving web services security.

TLS and HTTPS, thanks to encryption techniques, ensure that the communication is secure ensuring data privacy and integrity. Public key cryptography helps for the authentication of parties.

4.2. TLS

The TLS (Transport Layer Security) protocol provides privacy and data security for internet communication. The protocol is implemented on top of a reliable transport protocol, typically the TCP protocol and offers a private and secure connection. The connection is private in the sense that all communication is encrypted, with a symmetric encryption algorithm, and reliable in the sense that the message is protected by an integrity check that uses a keyed Message Authentication Code. A new key for the symmetric encryption is generated for each connection and is exchanged with a TLS Handshake Protocol. The Handshake protocol allows server and client to authenticate to each other, negotiate an encryption algorithm and secret keys before the communication takes place.

4.3. HTTPS

The HTTP protocol was originally designed for clear (unencrypted) communications. The use of HTTP for sensitive applications has led to the need of secure communications. HTTPS is just HTTP over TLS.

So, while HTTP communications directly use TCP, HTTPS communicates using TLS. To do so, the HTTPS client needs to act also as the TLS client, initiate the TLS connection and perform the TLS handshake. At this point the HTTPS client can behave as an HTTPS client and proceed over the secure communication channel.

Web server using the HTTPS protocol are identified by a URI starting with “https://” instead of “http://”. The TLS handshake requires a digital certificate that proves the identity of the parties. Although TLS allows both ends to authenticate themselves to the other party, in a typical web server interaction is only the web server that possesses a digital certificate and thus only the client can check the identity of the server while the server cannot check the identity of client. For most web applications, this is enough (the visitor needs to be sure that the website is the intended one).

4.4. PKI

A public key infrastructure is generally defined as the set of hardware and software, together with everything else needed to make them work, necessary to manage digital certificates based on asymmetric cryptography. The fundamental problem with asymmetric cryptography is that of safely acquiring the public key of other parties. The main purpose of a PKI is that of offering a way to acquire authenticated public keys.

A PKI binds an end user to its public key: if we get the public key from a PKI, then we can trust the key and use it to secure interactions with that user (as long as we trust the PKI that is validating the key). A basic component of a PKI is a certification authority (CA), together with a registration authority (RA).

A Registration Authority is a trusted entity that ``registers'' users checking their identities. A Certification Authority, for each registered user, creates a pair of secret-public keys, with the secret key given to the user and the public key stored in a digital certificate. The digital certificate contains the identity of the user and optionally additional information and is digitally signed by the CA.

It is common that the RA and the CA are the same entity, although the RA can be a separated entity trusted by the CA. The CA also manages a repository of digital certificates, whose purpose is that of providing information about the status of the digital certificates.

Together with the database of certificates the CA must manage also a Certificate Revocation List (CRL) to keep track of certificates that have been revoked. A PKI may consist of several CAs. Finally, PKI users are individuals or organizations that use the services of the PKI, obtaining and checking digital certificates.

Digital certificates are stored using the X.509 standard format. The certificate is protected by a digital signature of the issuer, that is of the CA. Thanks to this signature, users of the system know that the contents have not been tampered with. Clearly one must be able to verify the signature of the CA. The certificates contain 6 mandatory fields: a serial number, the indication of the algorithm used to sign the certificate, the name of the issuer, the validity period (start and end dates) and, clearly, the name of the user and its public key.

A PKI can support many web and e-commerce applications, allowing to secure the exchange of messages, web access and custom applications.

5. Trusted interactions in decentralized settings

As the previous chapter highlights, PKI-based trust is that it depends on the existence of a recognized hierarchy of trusted authorities (CAs), which can certify people's and organizations' identities via registration authorities.

This model has been challenged many times, the most significant alternative being the web-of-trust model initially devised by Phil Zimmermann in 1992 in conjunction with the creation of its technical implementation, PGP (Pretty Good Privacy). The idea presented by Zimmerman was:

> *As time goes on, you will accumulate keys from other people that you may want to designate as trusted introducers. Everyone else will each choose their own trusted introducers. And everyone will gradually accumulate and distribute with their key a collection of certifying signatures from other people, with the expectation that anyone receiving it will trust at least one or two of the signatures. This will cause the emergence of a decentralized fault-tolerant web of confidence for all public keys.*

The model, which relies on binding a public key and its owner via multiple vouching, is an alternative to the centralized trust model of a PKI, which relies exclusively on certificate authorities. As with

computer networks, there are many independent webs of trust, and any user (through their public key certificate) can be a part of, and a link between, multiple webs. In order for these "webs" to emerge, users would meet in public events named "key signing parties".

The scheme is more flexible than public key infrastructures and leaves several trust decisions to the users. It is not perfect and requires both caution and intelligent supervision by users. Conversely, CA-managed PKIs are more consistent with the requirements of legal value and liability which are required in some scenarios – including those which we are considered as use cases for PRESENT.

Both PKI and web-of-trust as implemented in PGP however **only partly meet the expressivity and flexibility requirements which are demanded by real world scenario:** in many cases, besides information on the identity of the user (name, family name, national id number, etc.), relying parties might need to get additional information (e.g., age, education, allergies, financial solvability…) which can only be provided by specific authoritative organizations (national health case system, universities, banks…). These organizations are normally referred to as Attribute Authorities.

When it comes to the machine-machine exchange of more than "core identity data", public key certificates both under the PKI and the web-of-trust flavour are normally insufficient. Specific data structures with extended data are used, and specific protocols are introduced for the exchange of such data structures. The table in fig. 3 represents the match between the most commonly adopted data structures and exchange protocols. For instance, Verifiable Credentials in JSON/JSON-LD format are normally transported on OpenID Connect or DIDComm protocol:

Protocol / **Data structure**		SAML	OIDC	DIDComm	ISO 23220	ad hoc
SAML token		X				
Generic JWT			X			
Verifiable Credential	JSON/JSON-LD		X	X		
	JWT		X	X	X	
X.509 attribute certificate						X

Figure 3: Protocol/data structure mapping

For details on the data structures we refer to:

- Verifiable credentials / presentations [1] – note that more flavors are available, including JSON, JSON-LD and JWT
- SAML token [2][3]
- JWT [4]
- X.509 attribute certificate [8]

As for the protocols, some of them are specifically built to transport their own data structure, even if they often allow encapsulation of different structures. A comparative study on how data structures can be used to convey personal data is presently ongoing in [10], while some work is in progress in [11] to identify the appropriate protocols in the context of personal identity wallets.

SAML is normally considered a heavyweight protocol, which is mostly applied in legacy applications and does not fit well with mobile applications stacks.

ISO 23220 in part 3 is in the process of specifying protocols for the transport of verifiable credentials (JWT flavour). Being inspired from previous work on Mobile Driving licence [14], the focus is on proximity transactions, so QR-code encoding, and Bluetooth Low Energy (BLE) are the main targets [13].

X.509 attribute certificates does not enjoy a standard transport protocol, so different ad-hoc protocols can be met in real world implementations.

The most interesting candidates for agent-to-agent communications according to the requirements for PRESENT today appear to be DIDComm [9] and OIDC [3] transporting a Verifiable Credential [1] data structure in one of its different flavours. They are better detailed in the following.

5.1. OIDC for Verifiable Credentials

According to [12] OIDC can nicely meet the requirements of the SSI personal data exchange schema as follows.

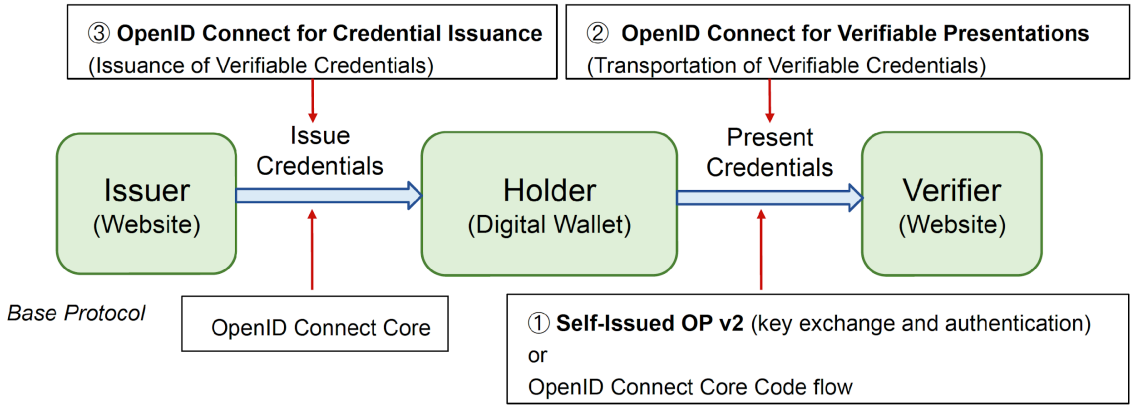

Figure 4: Use of OIDC for credential/presentation exchange

In this model there is a neat distinction between the phases of "credential issuing" (where a user gets the certified information from an attribute authority or similar) and "credential presentation" (where the user transmits this information to a relying party). Both of them are relevant to PRESENT:

- The Registration Authority use case is a scenario for credential issuing
- The Virtual Clerk use case is a scenario for credential presentation (used for user authentication)

The credential issuing phase in this model leverages on a traditional OpenID Connect protocol [5], which would be normally preceded by a discovery [17] and a registration phase [16]. The model is represented below.

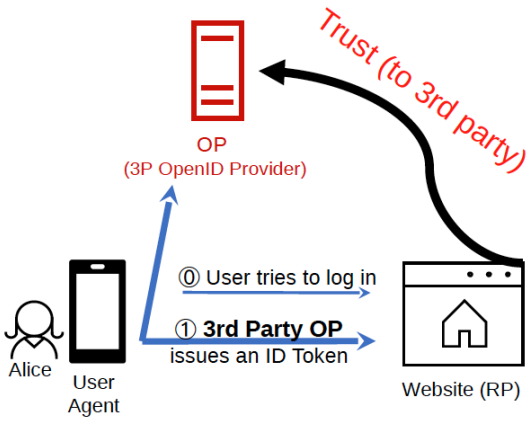

Figure 5: openID connect standard model

Differently, the credential presentation side is based on the Self-issued OpenID Connect protocol [17], which is represented below.

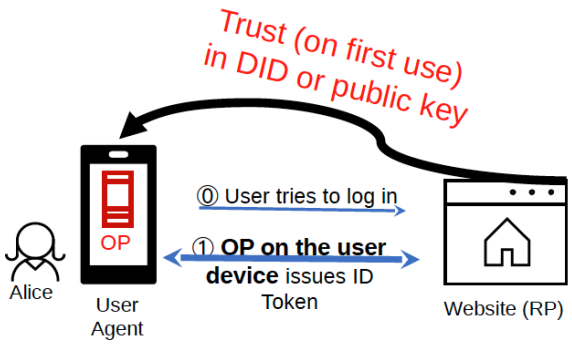

Figure 6: Self-issued openID connect model

The specific profile for transporting verifiable credentials (more precisely, verifiable presentations which are an aggregation of credentials) is provided in [6].

5.2. DIDComm

DIDComm is a novel personal data exchange protocol explicitely conceived for self-sovereign identity frameworks [9].

DIDComm has an initial version, v1, which was born under the umbrella of Hyperledger Aries protocol, and was better known as ARIES RFC 0044 [19]. Version 2, patrocinated by Decentralized Identity Foundation, included major changes [18]:

> *"DIDComm was first developed within an international and collaborative open source co-development project called Aries (hosted at Hyperledger) and under a IPR regime designed to cover software but not specifications. Aries was spun out of the earlier Hyperledger Indy project, to both iterate, expand, and make more blockchain-agnostic the codebase and tooling created earlier for the identity blockchain Indy" … "At some point, however, it became clear that further interoperability would be best served not by writing specifications based on existing Aries implementations, but by a more "green-field", specification-first design process with interlocutors from further afield. This work came to be co-sponsored by DIF and Hyperledger partly to engage these outside interlocutors, and partly because the IPR* protections *of DIF were more appropriate to a*

specification-first open-standards process." and finally that *"This effort has the direct support and participation of many core members of the Aries community. It is within the plans of the Aries community to migrate to DIDComm V2 as it becomes ready for use and tested. In both existing DIDComms (many of which are already in production), protocols relying on a DIDComm-encrypted channel for authentication or communications functions will be moved over."*

In short, major improvements include:

DIDComm v1	DIDComm v2
Born within Hyperledger Aires	Active DIF Working group
Active production across multiple vendors & codebases; full backwards compatibility with Aries & Indy solutions	In development with a broad community of implementers beyond just Aires implementations
Loosely based on JSON Web Encryption standard	Pursuing full JOSE standards (JWM, ECDH-1PU, etc.)
Special Case handling of Peer DIDs	Uniform DID Method support
Dedicated connection handshaking	Streamlined connection handshaking

Figure 7: DIDComm V1 vs DIDcomm V2

Somehow confusingly, DIDComm introduces the concept of "DIDComm protocols", which are specific applicative level stateful exchange patterns built on top of the DIDComm protocol primitives. These patterns include:

- The agent based architecture, where an agent is a sw that acts for a user
- Secure agent to agent communication on different protocols
- W3c standards support
- Interactions with IOT systems or even directly with specific, authenticated devices
- Secure user messaging or even chat-like instant messaging
- Issuing a credential
- Presenting a verifiable presentation including an identity proof
- Payment coordination

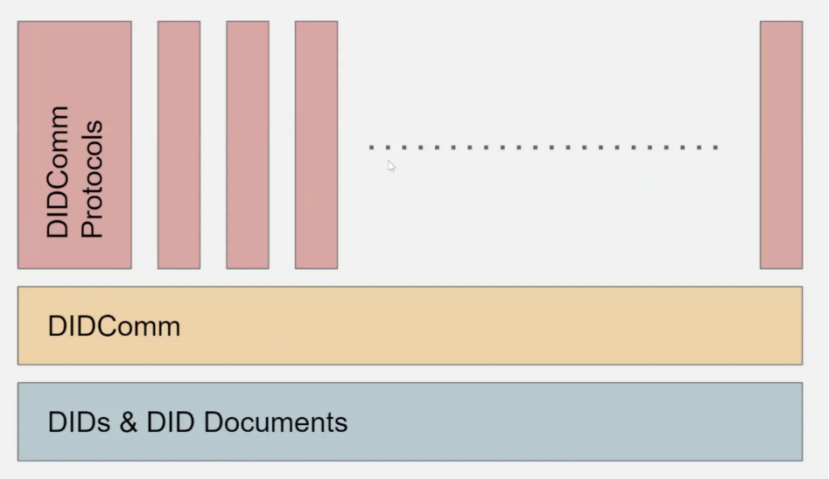

Figure 8: "DIDComm Protocols" vs protocol "DID Communication" (DIDComm)

On its turn, DIDComm is founded on the concepts of Distributed Identifier (DID) and DID Document [20], an object in Json format containing info about the DID, generally saved on a decentralized ledger. The DID Resolution process is the mechanism used to obtain a DID Document from a DID.

DIDComm is appropriate both for the "credential issuing" and the "credential presentation" flows.

6. Implementations

6.1. OIDC implementations

The choice of the OIDC protocol described in section 4.1 for the supply phase of the Verifiable Credential and specifically of the Self-issued OpenID Connect Protocol (SIOPv2) for the credential presentation phase is chosen due to the need to use a well-known open standard for the transport phase between agents in the SSI model previously described. Another peculiarity of the implementations of this specific scenario is its mobile oriented nature. It isn't a coincidence that the most significant open source implementations currently are represented by SSIKit [33] written entirely in Kotlin and therefore compile for mobile oriented environments (e.g. Android).

The EU Blockchain Service Infrastructure (EBSI) [35] in its implementation of the EU Self-Sovereign Identity Framework (ESSIF) refer to OIDC/SIOPv2 for those specific development of its SSI Agents infrastructure and in this context the SSIKit expose specific ESSIF API for the management of VCs using OIDC/SIOPv2:

- **Issuing a Verifiable Credential:**

A natural person obtains a Verifiable Credential from a Trusted Issuer (legal entity). SSIKit defines a profile for OIDC for Credential Issuance. Credential issuance flow is an extension of the OIDC Authorization Code Flow [31].

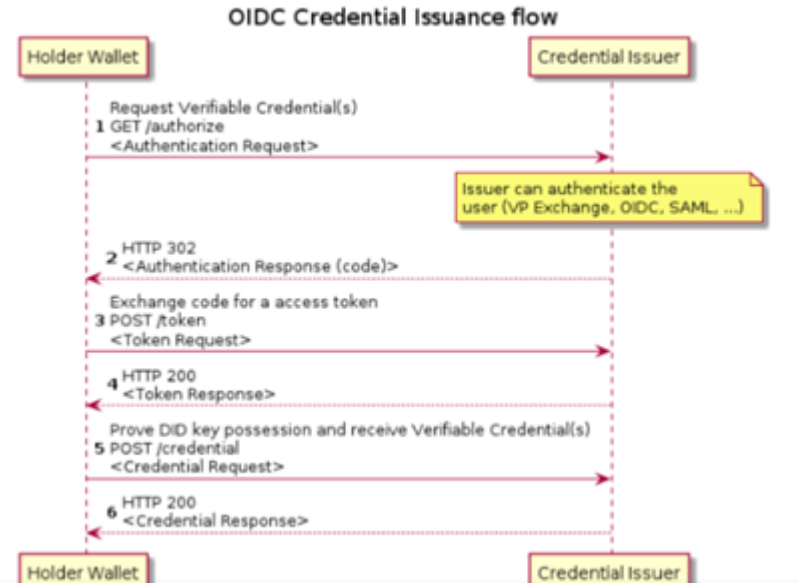

Figure 9: OIDC-based user authorization according to EBSI

- **Exchange of Verifiable Credentials:**

Presentation of a Verifiable Presentation to a Verifier / Relying Party (incl. OIDC-SIOPv2 based data exchange, open interface for third party verifier component). More information and references can be found in [34].

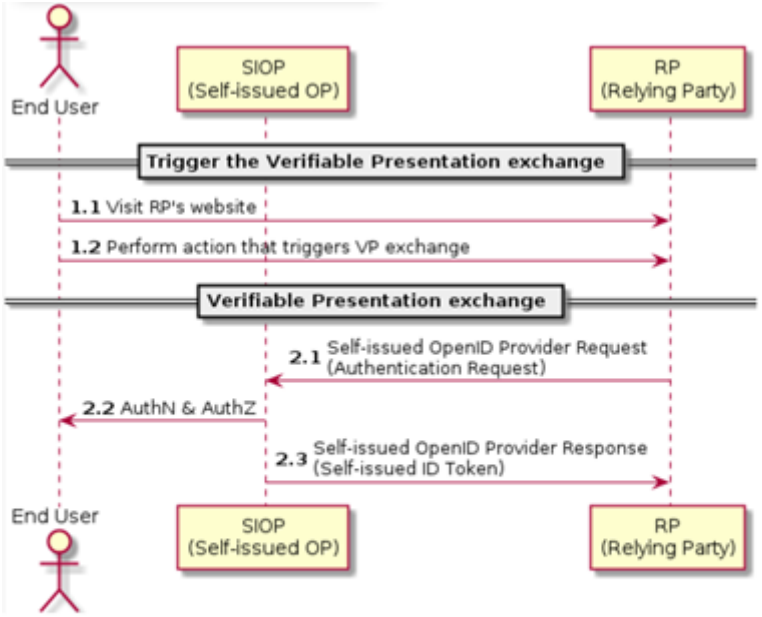

Figure 10: OIDC-based user Verifiable Credential presentation according to EBSI

6.2. DIDComm in Hyperledger Indy

The most significant activities related to the implementation of DIDComm live within Hyperledger project [21].

The initial implementation for DIDComm v1 happened in the context of Hyperledger Indy [22]. Hyperledger Indy is a Decentralized Ledger Technology that provides also an interfacing component, the Indy SDK used in the last years to product SSI identity solutions. According to [23] Indy SDK is a C-callable library that provides the basic building blocks for the creation of applications on the top of Hyperledger Indy ledger. It is available for most popular desktop, mobile and server platforms.

6.3. DIDComm in Hyperledger Aries

Hyperledger Aries [24] builds on top of Hyperledger Indy, and uses Indy ledger as just one of the possible underlying Decentralized Ledgers (in fact, as of today, interfacing non-Indy ledgers is ongoing in the Aries community). The Aries community [25] itself declares that "Aries grew out of the work in Hyperledger Indy to create technologies for managing decentralized identity. Indy provides a specific blockchain purpose-built for identity. The tools created in Indy for building agents are being migrated to Aries and extended to be blockchain-agnostic. Functionality related to the Indy ledger will remain in Indy."

In fact, Aries is a large initiative which includes several working groups and produced more implementations of the same functional component, including:

- **aries-cloudagent-python** (Aca-py) [27]: the Python implementation is based on DIDComm-V1 and uses Indy as a Ledger through Indy SDK. The agent is suitable for all

non-mobile agent applications and has production deployments. Many SSI implementations are already in place based on this component, including [28] and [29].

- **aries-staticagent-python**: a configurable server-side agent that does not use persistent storage. To use it, keys are pre-configured and loaded into the agent at initialization time.
- **aries-framework-dotnet**: it can be used for building mobile (via Xamarin) and server-side agents and has production deployments. The framework uses Indy through the indy-sdk
- **aries-framework-go**: An innovative and technically interesting pure go language (golang) framework that does not currently embed the Indy SDK and works on supporting a golang-based verifiable credentials implementation. The development is in progress and some features are based on still "experimental projects" like go-mobile. The most recent version of the framework [26] provides 3 different implementations on the same common communication libraries:
    - **aries-agent-rest**: a Command Line Interface (CLI) implementation of a server-side framework. Following the framework-controller paradigm, it exposes an HTTP interface for its companion controller. Differently from other frameworks, it does not rely on Indy SDK. A golang-based verifiable credentials implementation is in progress.
    - **aries-agent-mobile**: uses the experimental go-mobile tool [30] to produce SW libraries used for Android and Ios.
    - **aries-js-worker**: a javascript interface to Aries Framework Go, for use in node.js or directly in the browser.

7. Conclusions

The world of trusted agent-to-agent communication is a rapidly evolving one. While there is a well-established model based on traditional TLS technology which nowadays dominates, there are many initiatives aimed at expanding the reach to cover a larger set of trust requirements, especially when there is need to transport custom user attributed (possibly, attested by some party).

This moving context provides for new specifications and prototypical implementations, which are far to be stable yet. PRESENT had to make a pragmatic choice, based on an evaluation of the maturity of the models and the technology. The outcome was the adoption do DIDComm v2, in its python flavour, which was evaluated to be a mature and usable communication protocol provided by the community. Given this positive circumstance, the agent-to-agent trusted communication was therefore adopted out-of-the-box from Aries-Pyton [27], and embedded in PRESENT trust architecture as documented in D5.2. The choice proved to be fully appropriate to the architecture, since functional testing provided sound results and no custom adjustments to the transport protocol were necessary.

8. References

[1] W3C Recommendation (9 November 2021): "Verifiable Credentials Data Model v1.1". Available at https://www.w3.org/TR/vc-data-model/.

[2] OASIS Standard SAML 2.0: "Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0", 15 March 2005. Available at http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf.

[3] OASIS Approved Errata SAML 2.0: "SAML Version 2.0 Errata 05", 01 May 2012. Available at http://docs.oasis-open.org/security/saml/v2.0/errata05/os/saml-v2.0-errata05-os.pdf .

[4] IETF RFC 7519: "JSON Web Token (JWT)", May 2015. Available at https://datatracker.ietf.org/doc/html/rfc7519.

[5] OpenID Core 1.0: "OpenID Connect Core 1.0 incorporating errata set 1". Available at https://openid.net/specs/openid-connect-core-1_0.html.

[6] OpenID Verifiable Presentations: "OpenID Connect for Verifiable Presentations". Internet-Draft December 2021. Available at https://openid.net/specs/openid-connect-4-verifiable-presentations-1_0-07.html -name-vp_token.

[7] OpenID Presentation Exchange: " Presentation Exchange 2.0.0". Working Group Draft. Available from: https://identity.foundation/presentation-exchange

[8] ITU-T Recommendation X.509 (2005) | ISO/IEC 9594-8:2005, Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks.

[9] DIDComm 2.0 Messaging guide. Available from https://identity.foundation/didcomm-messaging/spec/

[10] ETSI TS 119 472 :" Electronic Signatures and Infrastructures (ESI); Profiles for attribute attestations". In progress.

[11] ETSI TS 119 462:" Electronic Signatures and Infrastructures (ESI); Wallet interfaces for trust services and signings". In progress

[12] ISO/IEC DIS 23220-1, Cards and security devices for personal identification. Building blocks for identity management via mobile devices — Part 1: Generic system architectures of mobile eID-Systems

[13] ISO/IEC DIS 23220-3, Cards and security devices for personal identification. Building blocks for identity management via mobile devices — Part 3: Protocols and services for installation and issuing phase. In progress

[14] ISO/IEC 18013-5 Personal identification — ISO-compliant driving licence — Part 5:Mobile driving licence (mDL) application

[15] Sakimura, N., Bradley, J., and M. B. Jones, "OpenID Connect Dynamic Client Registration 1.0 incorporating errata set 1", 8 November 2014, <https://openid.net/specs/openid-connect-registration-1_0.html>.

[16] Sakimura, N., Bradley, J., de Medeiros, B., and E. Jay, "OpenID Connect Discovery 1.0 incorporating errata set 1", 8 November 2014, <https://openid.net/specs/openid-connect-discovery-1_0.html>.

[17] Microsoft, Jones, M. B., and T. Looker, "Self-Issued OpenID Provider V2", 20 July 2021, <https://openid.bitbucket.io/connect/openid-connect-self-issued-v2-1_0.html>.

[18] https://medium.com/decentralized-identity/understanding-didcomm-14da547ca36b

[19] https://github.com/hyperledger/aries-rfcs/blob/main/features/0044-didcomm-file-and-mime-types/README.md

[20] RFC 5280 Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. Available from https://datatracker.ietf.org/doc/html/rfc5280

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

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

[23] https://github.com/hyperledger/indy-sdk

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

[25] https://github.com/hyperledger/aries

[26] https://github.com/hyperledger/aries-framework-go

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

[28] https://dizme.io/

[29] https://www.ontario.ca/page/ontarios-digital-id-technology-and-standards

[30] https://github.com/golang/mobile

[31] https://github.com/decentralized-identity/did-siop

[32] https://ec.europa.eu/digital-building-blocks/wikis/display/EBSIDOC/EBSI+Credential+Issuance+Guidelines

[33] https://docs.walt.id/ssikit/ssikit-for-europe.html#essif-use-cases-flow-diagrams

[34] https://ec.europa.eu/digital-building-blocks/wikis/display/EBSIDOC/EBSI+Verifiable+Presentation+Exchange+Guidelines

[35] https://ec.europa.eu/digital-building-blocks/wikis/display/CEFDIGITAL/EBSI