Privacy-Preserving PayString Service

Flaviene Scheidt de Cristo*, Wazen M. Shbair*, Lucian Trestioreanu*, Aanchal Malhotra† and Radu State*
* University of Luxembourg, SnT, 29, Avenue J.F Kennedy, L-1855 Luxembourg
Email:{flaviene.scheidt, wazen.shbair, lucian.trestioreanu, radu.state}@uni.lu
† Ripple, 315 Montgomery Street, San Francisco, CA 94104, United States
Email: amalhotra@ripple.com

***Abstract*—PayString is an initiative to make payment identifiers global and human-readable, facilitating the exchange of payment information. However, the reference implementation lacks privacy and security features, making it possible for anyone to access the payment information as long as the PayString identifier is known. We propose an innovative solution, named PayStringSecure, for this issue by integrating a privacy layer based on Self-Sovereign Identity (SSI), Decentralized Identifier (DID) and Verifiable Credential (VC) to the PayString protocol. A working prototype has been developed to enrich the protocol with the new features.**

***Index Terms*—PayString, DID, Verifiable Credential, Self-Sovereign Identity**

I. Introduction

International payment systems are siloed and disconnected; fiat transfers for example often take days to be fulfilled, without direct feedback from the receiver to the sender, and oftentimes incurring high fees. Although several *Distributed Ledger Technologies* (DLT)[1] seek to address aspects of interconnectivity, payment speed, fees and feedback. To our knowledge the same is not true concerning the ease and freedom of usage for the end-user, when dealing with complex payment end-point descriptors, and especially when the banking systems and the bank account formats are different.

PayString [1] - a new service from Ripple - is a web-based protocol designed to facilitate the exchange of payment information. The aim is to close the gap by replacing complicated bank account numbers and cryptocurrency wallet addresses with an easy to memorize identifier. As such, users can use a single address linked to several bank accounts or wallet addresses. However, there is still room for improvement concerning the security and privacy of user's data. Our solution aims to solve this issue by adding a layer over the usual protocol implementation. This layer comprises technologies such as Self-Sovereign Identity (SSI), Descentralized Identifier (DID) and Verifiable Credentials (VC) to achieve the desired outcome.

Identity management and authentication have been an issue since the beginning of the Internet, however only recently the idea of giving the user ownership of its digital identity aired with strength; the surge of DLTs, such as Blockchains, gave this movement an engine on which to work over. Bokkem [2] dissects some SSI based solutions present on the market until 2019 and some problems they try to solve. CanDID [3], for example, tackles the problem of the lost key - in which if one does lose its private keys, one can't prove his identity anymore and consequently lose access to the system - and tries to solve it by using oracles.

In a more practical sphere, we can take a look at the work of [4], an initiative between the Delft University of Technology and the Dutch Government that aims to create a system based on SSI and biometric data for the issuance of paperless passports. Also important to mention is uPort [5], a platform that provides tools, libraries and protocols for developers aiming to create user-centric solutions based on SSI. uPort also uses the concepts of DIDs and VCs, but it does not have its blockchain, being built over Ethereum.

The operation of PayString is quite simple; Alice wishes to transfer money to Bob. Bob already registered his payment information on the PayString Server, so he only needs to send his easy-to-remember PayString address (*bob$example.com*) to Alice. She will request the payment information to the server. The server will then send the payment information: a pointer to Bob's wallet. Alice will use this pointer to transfer money from her wallet to Bob's wallet.

There is some risk attached to the PayString server, being it an always-online system: an attacker could hijack or impersonate the server, sending different payment information for a given pointer to deviate the funds to the attacker's wallet. The impersonation attack is considered to be of high risk when the keys used by the PayString server to sign the payment information are compromised. There is less risk if only the keys used for establishing secure channels are endangered.

Besides the server, an attacker may try to impersonate an entity, registering different payment information in the name of an entity that still did not register its true information. The attacker may also change the information of an already existing entity's pointer if the keys got compromised. To close some of those gaps, we propose *PayString Secure*, which is detailed on the next section.

II. PayString Secure

PayString is a solution for simplifying payments in nowadays scenarios, where users need to deal with multiple payment methods, currencies, and identifiers. However, it does not integrate privacy by design concepts. With this purpose in

[1]Like, for example, *Ripple*, *Quorum* (JP Morgan), *Ethereum*, *Hyperledger Fabric* and projects like *Interledger* (Ripple), *Stella* (EU and Japan) and more

978-0-7381-1420-0/21/$31.00 ©2021 IEEE

mind we enriched the protocol with novel privacy and security features.

The first issue we tackled was the fact that anyone could access the payment information of a given user by knowing their PayString identifier. Let's say Bob needs to receive some payment from his friend, Alice. Bob then sends his PayString to Alice. Somehow an anonymous and malicious third party guesses Bob's payment information. This third party now knows the bank where Bob holds his account and sends several phishing e-mails to capture Bob's bank account credentials.

The immediate answer to this problem is to implement an *Access Control List* (ACL), allowing Bob to grant and deny access to his payment information. Considering that Bob wishes to receive only payments on fiat currency from Alice, there is no need to give her access to his XRP wallet address.

With the ACL integration to the PayString server, we solve the primary privacy issue encountered on the protocol. But how do we verify the identity of the user that requests access to another user's payment information? We can verify identities using asymmetric cryptography. So Alice needs to send her public key to Bob, which will include this key in his ACL. When requesting the payment information, Alice needs to sign the request using her private key.

This solution works well but makes us fall again into the issue we are trying to avoid: the non-human-readable identifiers. To address this problem we decided to use SSI in the form of DID and VC.

SSI turns the user in the sole provider and authenticator of his/her own identity and the *trust providers* are the entities that provide trustfulness for these identities [6]. DIDs [7] are unique identifiers for decentralized verifiable identities. And lastly, VCs [8] are credentials whose authenticity can be verified by trustable authorities.

By using DIDs and VCs, the Blockchain serves as a single source of truth, enabling transparency, security and trust. Alice is now able to prove her identity to the PayString Server using a DID instead of a long sequence of random characters facilitating the entire process of payment.

As the foundation layer on which PayString Secure is built, we used Hyperledger Indy [9], a distributed ledger that provides tools and libraries for supporting the development and integration of solutions based on distributed identities. On top of that, we have extended the reference implementation of PayString server with two new modules, the ACL - explained above - and the *Credential Manager*, which stores and verifies the credentials. We also built the *PayString Digital Notary*, which is the entity that issues the VCs.

Figure1 shows an example of the interaction between the modules and entities using PayString Secure, to safely exchange payment information between two users: Alice and Bob. Alice knows Bob's PayString identifier, (*bob$example.com*), and wishes to make a payment to Bob using fiat currency. (1) Alice asks the Notary for a new VC; for that, she needs to provide her PayString and DID identifiers; (2) The Notary issues the VC and send it to Alice's wallet, where the VC will be stored. Possessing the VC, Alice makes

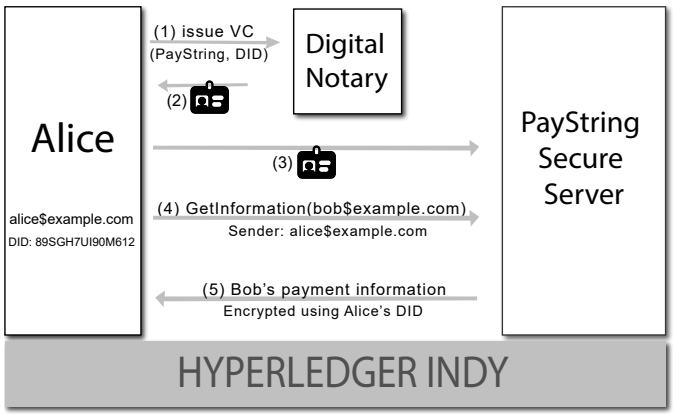

Fig. 1: The workflow involving the Notary, PayString Secure server and Alice

a request; (3) Alice presents her VC to the PayString server, which uses the Credential Manager module to store them locally for future requests; (4) Alice sends a new request for Bob's payment information; (5) The PayString server validates the VC and then check the ACL to see if Alice is authorized to see Bob's information. If she is, the server encrypts Bob's payment information using Alice's DID and sends her the encrypted message that she shall be able to decrypt using her private key.

A prototype of PayString Secure was built for the PayString Block-Sprint Hackathon, winning the grand prize[2] and showing that the implementation of the security and privacy layer is feasible. More than that, it shows how important does features are for the community.

III. Conclusion and Future Work

We propose a security and privacy extension for the PayString protocol. Our security and privacy extension concerns the access control mechanism of PayString, so we propose both a formal model and a way to integrate it with DID to use VCs.

Future work concerns making a full analyses addressing how much of an overhead does the security and privacy layer add to the protocol. Would be also interesting to investigate the usage of *Distributed Hash Tables* (DHT) to improve the service reliability; *integrity of information* is also a potential challenge for those who will run a server because while the data will be accessible to clients, the service providers will know nothing about what is happening inside the server.

Acknowledgments

We thankfully acknowledge the support from the RIPPLE University Blockchain Research Initiative (UBRI) for our research. In addition, this work is partially supported by the Luxembourg National Research Fund through grant PRIDE15/10621687/SPsquared.

[2]Demo: https://youtu.be/nrej87tb7zQ

REFERENCES

[1] A. Malhotra and D. Schwartz, "Verifiable payid protocol internet draft," https://github.com/PayString/rfcs/blob/master/dist/spec/verifiable-payid-protocol.txt accessed on 14/12/2020, Ripple, Tech. Rep., 2020.

[2] D. van Bokkem, R. Hageman, G. Koning, L. Nguyen, and N. Zarin, "Self-sovereign identity solutions: The necessity of blockchain technology," *arXiv preprint arXiv:1904.12816*, 2019.

[3] D. Maram, H. Malvai, F. Zhang, N. Jean-Louis, A. Frolov, T. Kell, T. Lobban, C. Moy, A. Juels, and A. Miller, "Candid: Can-do decentralized identity with legacy compatibility, sybil-resistance, and accountability," *IACR Cryptol ePrint Arch*, 2020.

[4] Q. Stokkink and J. Pouwelse, "Deployment of a blockchain-based self-sovereign identity," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, 2018, pp. 1336–1342.

[5] "Helping you build user centric apps on blockchains," https://developer.uport.me/overview/index accessed on 24/11/2020, uPort, 2020.

[6] A. Tobin and D. Reed, "The inevitable rise of self-sovereign identity," https://sovrin.org/wp-content/uploads/2018/03/The-Inevitable-Rise-of-Self-Sovereign-Identity.pdf accessed on 27/10/2020, Sovrin Foundation, Tech. Rep., 2016.

[7] D. Reed, M. Sporny, D. Longley, C. Allen, R. Grant, M. Sabadello, and J. Holt, "Decentralized identifiers (dids) v1.0," https://www.w3.org/TR/2020/WD-did-core-20201108/ accessed on 20/11/2020, W3C, Tech. Rep., 2020.

[8] M. Sporny, D. Longley, and D. Chadwick, "Verifiable credentials data model 1.0," https://www.w3.org/TR/vc-data-model/ accessed on 20/11/2020, W3C, Tech. Rep., 2020.

[9] "Hyperledger indy," https://indy.readthedocs.io/en/latest/ accessed on 20/11/2020, 2020.