PayString Protocol

December 9, 2020

Aanchal Malhotra, Austin King, David Schwartz, Michael Zochowski
info+dev@PayString.org

Contents

Abstract

Complicated and hard-to-remember payment addresses inherently lead to a poor user experience resulting in confusion, errors, and potential loss of funds. This presents a major barrier to adoption of blockchain and other payments technology. In this work we present PayString, a standard for human-readable payment addresses that can be resolved to any underlying payment rail, whether cryptocurrency or traditional, in a secure and private manner. PayString protocol is designed to be general, flexible, and extensible. More specifically, we present two extensions that layer functionality on top of ledger transactions. First, functionality that cryptographically correlates on-ledger transactions to third party verifiable proofs of payment and receipts of payment. Second, PayString protocol is extended to provide a messaging standard for regulated ("covered") institutions to exchange information to fulfil their compliance requirements. In its various forms, PayString protocol provides strong guarantees to transacting parties, including strong security, non-deniability by generating third party verifiable signed cryptographic proofs, and privacy from third parties.

Introduction

Cryptocurrency addresses are usually long strings of random alphabets and integers. This leads to substantial confusion, a myriad of errors, and potential loss of funds. To accelerate mainstream adoption of decentralized payment networks in a user-friendly manner, it is imperative to:

a) Create a simple, easy to remember, human-readable payment address system for non-technical users.
b) Standardize a protocol that provides the mapping between these human-readable addresses and the underlying rail addresses in a secure and private way.

In this work, we present PayString, a human-readable payment address standard for *all* payment rails and currencies. The PayString protocol is a simple request/response application-layer protocol built on top of the existing standards HTTP and DNS. In its most basic form, it provides a mapping between human-readable addresses and their corresponding payment addresses. Despite its simplicity, PayString offers several compelling benefits. First, it lowers the barriers to adoption of blockchain technology through an improved user experience. Second, it provides for interoperability of namespaces across payment rails and currencies by allowing parties to transact via any shared rail using a *single* standard address. Third, it fully abstracts underlying payment rail details from end users, thus enabling greater accessibility and improved management of addresses for security, privacy, or enabling complex features.

The PayString protocol is designed to be simple, flexible, secure, and fully compatible with existing namespace systems, with a robust future roadmap for more advanced features. It can be extended to provide secure, private, and streamlined solutions for a variety of payments, identity and compliance use cases across both cryptocurrency and traditional finance.

The most significant set of extensions under development are verifiable PayString - a suite of security and privacy enhancements that add a variety of digital signature fields for linking external digital identities, proving ownership of the payment rail address, and providing a cryptographic trail of non-repudiable messages for the entire communication. It can be used to enable trust-minimized and trust-free security regimes and has applications in both custodial and non-custodial settings. We present two applications of verifiable PayString in this paper.

First, we show an extension to the verifiable PayString protocol that can be used to generate invoice requests, invoice reponses, proofs of payment and receipts of payments that allows a recipient to deploy access control mechanisms and better track transactions based on the sender. Together with PayString protocol’s substantial improvements in user experience, this extension enables a streamlined flow for point-of-sale, ecommerce, and other merchant transactions that are burdensome in the current cryptocurrency paradigm.

Second, we show another extension to the verifiable PayString protocol’s invoicing functionality that provides a simple and secure solution to meet the current and potential future compliance requirements of cryptocurrency service providers. In particular, we present a messaging standard for compliance with the Travel Rule, which requires certain cryptocurrency service providers to exchange information on senders and receivers of transactions in the immediate future. This is a particularly complex task under current cryptocurrency payments flows, where it is challenging to determine both when the Travel Rule applies and how to securely exchange sensitive customer information when it does apply, but verifiable PayString protocol presents a straightforward and elegant solution to this pressing problem.

Additionally, we present a potential integration of verifiable PayString protocol with another Travel Rule information sharing messaging proposal Travel Rule Information Sharing Architecture (TRISA).

Ultimately, PayString is an open standard that is not limited to the applications discussed in this paper. We anticipate PayString to continue to grow to cover additional use cases and networks, and our goal is that PayString provides a truly universal and composable solution for all payments.

PayString protocol design principles

PayString is designed to provide solutions that are broadly appealing, inclusive, and streamlined across both the traditional finance and cryptocurrency spaces. Accordingly, we have emphasized the following principles in the PayString design:

1. Simplicity

Rather than reinventing the wheel, PayString protocol is built on existing web standards and infrastructure. We believe that new tools or infrastructure, particularly those involving a blockchain integration, significantly increase overhead. We've designed PayString protocol so that the barrier to adoption is minimal by building on proven tools and infrastructure. Each institution can participate in the network by deploying or using a single web service. No node management, no consensus; pure utility.

2. Neutrality: currency and network agnostic

Acknowledging that the cryptocurrency community must work collectively to meet the needs of our users and their governments, we designed PayString protocol as a fundamentally neutral protocol. PayString is capable of returning a user's address information for any network that they (or their service) support. This makes PayString a network and currency agnostic protocol, capable of enabling payments in BTC, XRP, ERC-20 tokens, Lightning, ILP, or even fiat networks like ACH.

3. Decentralized & peer-to-peer

Just like email servers, anyone can run their own PayString server or use third-party hosted services. If self-hosted, PayString introduces no new counterparty risk or changes to a service's security or privacy model. Unlike some of the other approaches, PayString *does not* require new, complex, and potentially unreliable peer discovery protocols, instead establishing direct peer-to-peer connections between communicating institutions from the start.

PayString is built on the most successful decentralized network: the web. There is no designated centralized authority, or a risk of a patchwork of different standards in different jurisdictions that make a global solution impossibly complex.

4. Extensibility and improved user experience

PayString itself is highly extensible, and can be used in a variety of other contexts, including improving the UX of sending and receiving to different users. PayString is designed to be an upgradeable and open standard, with a robust roadmap of future improvements and additional features.

5. Service sovereignty

Each service that uses PayString for their users maintains full control of its PayString service and has the ability to incorporate any policy they choose, including privacy, authentication, and security. They also have full sovereignty over users on their domain, just like in email. PayString is highly generalized and does not prescribe any particular solution outside of the standardized communication, which makes it compatible with existing compliance and user management tools and philosophies.

6. Composable with existing standards and namespaces

By design, PayString is highly abstract and generalized. As a result, PayString can easily wrap existing standards or namespaces, such as Ethereum Name Service, Unstoppable Domains, or service-specific identifiers like Cashtags or Coinbase Usernames, and provide each of them far greater reach and user value. For example, a user Bob of DigitalWallet that currently has username @bob could be served by DigitalWallet's PayString Server as bob$digitalwallet.com.

PayString URI scheme and PayString discovery

We define the PayString URI as a standard identifier for payment account information. In the same way that an email address provides an identifier for a mailbox in the email ecosystem, a PayString can be used as an identifier to provide details about the payment addresses. PayString is an email style identifier that separates the user and the host with a ‘$’ sign and resolves to a URL with the HTTPS scheme.

PayString URI Syntax
PayString URI defines the PayString syntax as follows:

```
PayString: user$host
```

The following example URIs illustrate several variations of PayStrings and their common syntax components:

```
PayString: alice$example.net
PayString: john.doe$example.net
PayString: jane-doe$example.net
```

The PayString Discovery protocol can be used to discover information about a 'PayString' URI using standard HTTP methods. The primary use-case of this protocol is to define how to transform a PayString URI into a URL that can be used with other protocols.

The following example illustrates an example PayString URI to URL resolution:

```
PayString URI: alice$example.com
PayString URL: https://example.com/alice
```

PayString Protocol - A protocol for human-readable payment addresses[1]

The asic PayString protocol is a simple application-layer request/response protocol. The primary use-case is to discover network-specific payment addresses along with optional metadata identified by a PayString. These PayStrings are accessible to end users and they fully abstract the underlying payment protocol details, allowing for a far improved user experience, integrations between different services, and an enhanced ability of services to manage their backend.

The protocol is based on HTTP transfer of PayString protocol messages over a secure transport. To support PayString protocol, the PayString client needs to discover a PayString URL corresponding to the PayString. This can be obtained either using mechanisms described in PayString Discovery or could be entered manually. HTTP requests to this endpoint may return payment addresses for different payment-networks and environments associated with a PayString. PayString protocol's web infrastructure — rather than a blockchain-based solution — makes it universally usable across both cryptocurrency and traditional finance, compatible with other namespace solutions, and universally appealing.

Basic PayString Protocol details

Terminology

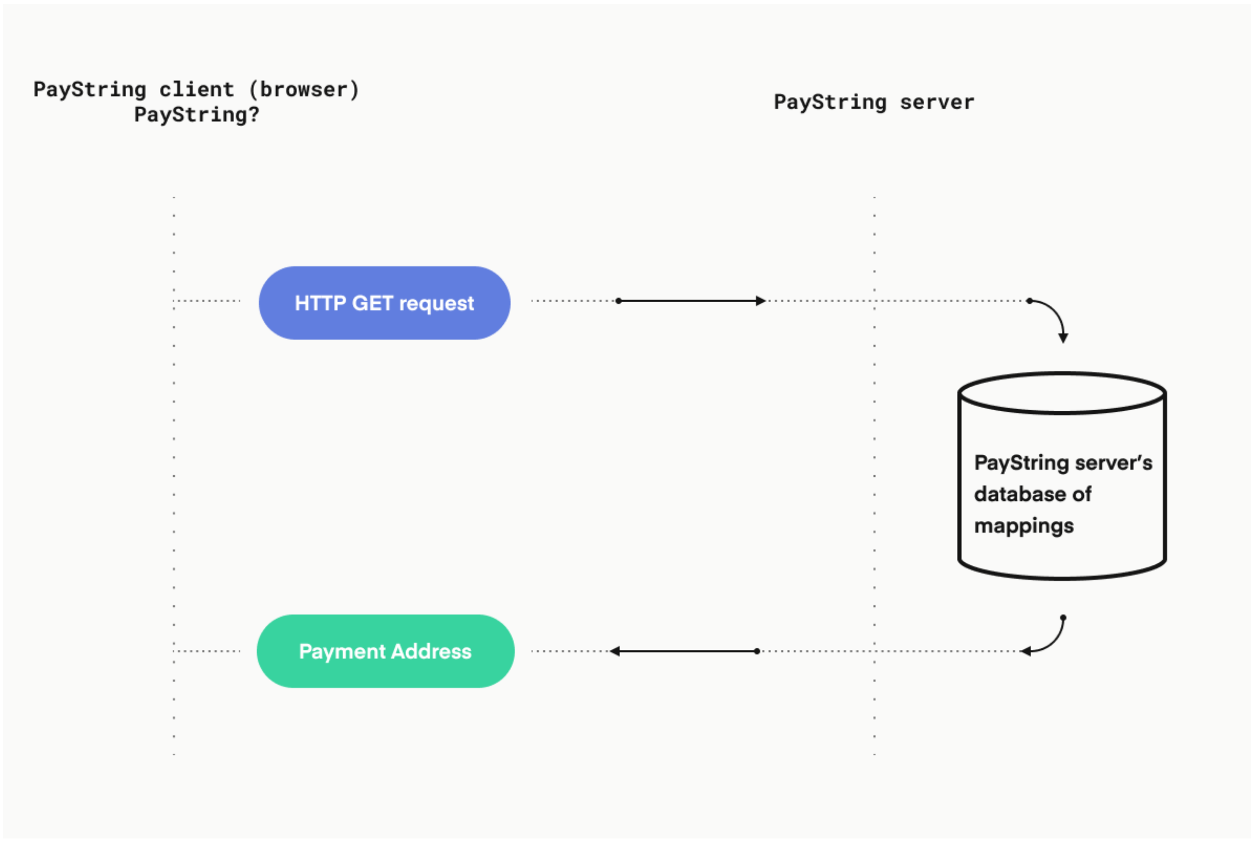
This protocol can be referred to as "Basic PayString Protocol" or "PayString Protocol". The following terminology is used in the following section.

- Endpoint: either the client or the server of the connection.
- Sender: individual or entity originating the transaction.
- PayString client: the endpoint that initiates PayString protocol/sending side of the transaction.
- PayString server: the endpoint that returns payment address information/receiving side of the transaction (custodial or non-custodial wallets, exchanges, etc).
- Receiver/PayString owner: individual or entity receiving the transaction/owner of the PayString.

Basic PayString protocol flow: securely retrieve payment address(es) corresponding to a PayString

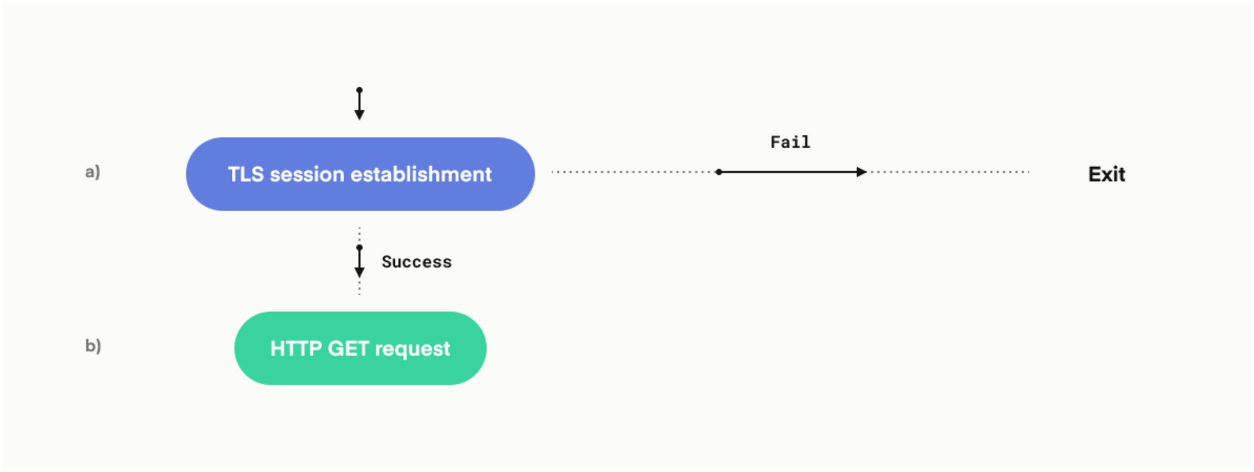
Basic PayString protocol can be used by PayString clients to easily query PayString servers for payment address information corresponding to human-readable PayStrings.

[1] This paper describes the v1 of PayString protocol. For details on versioning see here.


PayString client (browser)
PayString?
PayString server
HTTP GET request
PayString server's database of mappings
Payment Address

The following steps describe how the PayString client retrieves the payment address(es) corresponding to a PayString.[2]

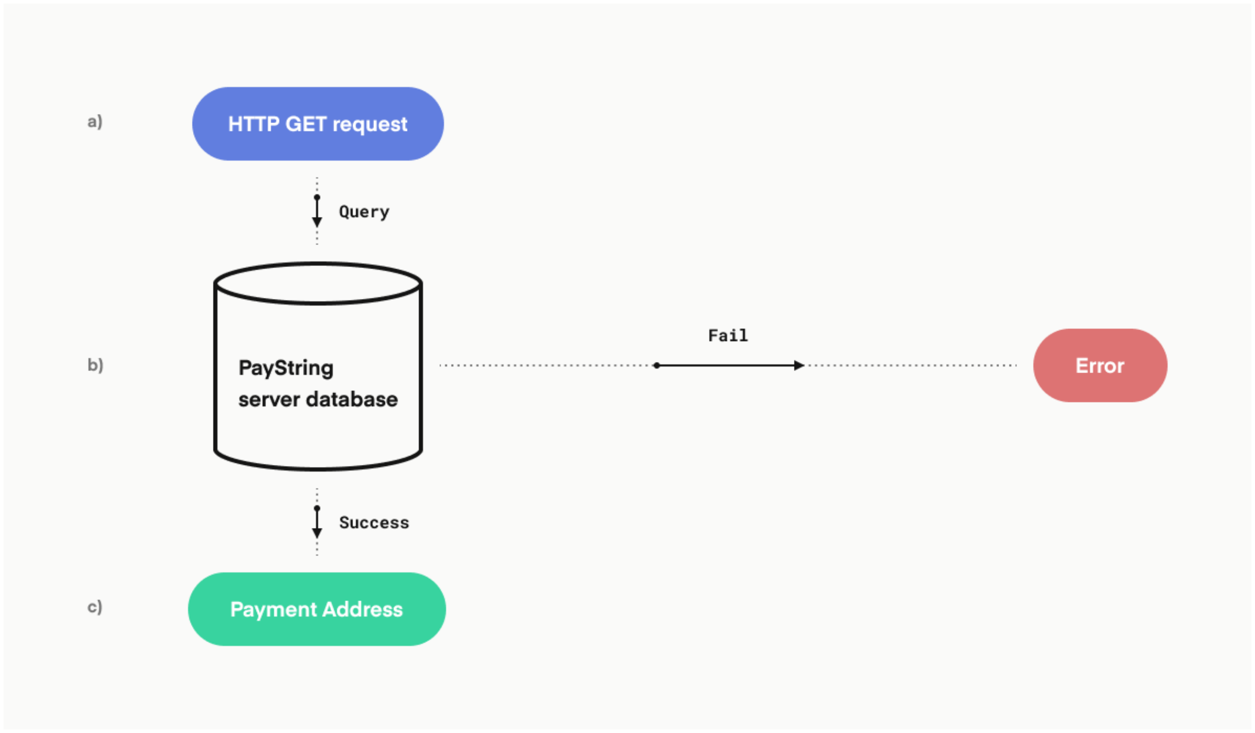
1) **HTTP GET request**: Processing steps by PayString client (typically a browser) to send an HTTP "GET" request:



[2] How a client obtains PayString is out-of-scope of this protocol. Instead of a PayString, the client can also use its corresponding URL directly, as described in the syntax resolution section.

   a) Establish a secure, mutually authenticated TLS 1.3 session with the PayString server as described in the session establishment section. The URL of the PayString server is derived from the PayString as described here.
   b) If the TLS session is successfully established, send an HTTP "GET" request over the established secure channel. PayString client specifies the payment-network and environment they support via the HTTP "Accept" header. They must specify the PayString version via "PayString-version" header. For details on PayString request headers refer to the HTTP Request and Response Headers; otherwise exit.

2) **Payment Address(es) Response**: Processing steps by PayString server to generate the Payment Address response corresponding to the queried PayString:



   a) Receive the "GET" request from PayString client.
   b) Query its database for the queried PayString. If the Payment Address information corresponding to the queried PayString, payment-network and environment exists in the database, the PayString server generates the PaymentInformation response with appropriate response headers as described in the HTTP Request and Response Headers section. Otherwise it generates an Error message. For details on error codes refer to the PayString protocol status communication section.
   c) Send PaymentInformation response or Error message to the PayString client.

Basic PayString protocol security model

The following is considered out-of-scope:

- Communication between the PayString owner and the wallet or exchange (which acts as PayString server) for PayString URI registration, etc.
- Communication between the sender of the transaction and PayString client to transfer information such as PayString URI and other transaction details, etc.
- PayString server URL discovery by PayString client. Implementations using PayString discovery protocol MUST consider the security considerations in the corresponding document.
- PayString server URL resolution by PayString client. Implementations using DNS, DNSSEC, DNS-over-HTTPS, DNS-over-TLS, etc. MUST consider the security considerations of the corresponding documents.

Network attacks

Basic PayString protocol's security model assumes the following network attackers:

- Off-path attacker: An off-path attacker can be anywhere on the network. She can inject and spoof packets but can not observe, or tamper with the legitimate traffic between the PayString client and the server.
- On-path attacker: An on-path attacker can eavesdrop, inject, spoof and replay packets, but can not drop, delay or tamper with the legitimate traffic.
- In-path or Man-in-the-middle (MiTM) attacker: An MiTM is the most powerful network attacker. An MiTM has full access to the communication path between the PayString client and the server. She can observe, modify, delay and drop network packets.

Additionally we assume that the attacker has enough resources to mount an attack but can not break the security guarantees provided by the cryptographic primitives of the underlying secure transport.
The basic PayString protocol runs over HTTPS and thus relies on the security of the underlying transport. Implementations utilizing TLS 1.3 benefit from the TLS security profile defined in RFC 8446 against all the above network attackers.

Denial-of-Service (DoS) attacks

As such cryptography can not defend against DoS attacks because any attacker can stop/interrupt the PayString protocol by:

- Dropping network packets
- Exhaustion of resources either at the network level or at PayString client and/or server.

The PayString servers are recommended to follow general best network configuration practices to defend against such attacks as outlined in RFC 4732.

Implementations are recommended to apply appropriate rate-limiting and other network-access control mechanisms to prevent flooding of requests.

Information integrity

The HTTPS connection provides transport security for the interaction between PayString client and server but does not provide the response integrity of the data provided by PayString server. A PayString client has no way of knowing if data provided in the payment account information resource has been manipulated at the PayString server, either due to malicious behaviour on the part of PayString server administrator or as a result of being compromised by an attacker. As with any information service available on the Internet, PayString clients should be wary of the information received from untrusted sources.

Basic PayString protocol privacy model

All application and user data stays private from passive third-parties. Our protocol ensures application and user data privacy against third-parties by encapsulating all traffic in HTTP-over-TLS.

a) Provides end-to-end encryption of communicating data between parties.
b) Provides mutual authentication between the communicating parties.
c) Provides perfect-forward secrecy, i.e. keys compromised in the future do not compromise the privacy of data encrypted in the past.

The PayString client and server should be aware that placing information on the Internet means that any one can actively access that information. While PayString protocol is an extremely useful tool to discover payment account(s) information corresponding to a PayString URI, PayString owners should also understand the associated privacy risks. The easy access to payment account information via PayString protocol was a design goal of the protocol, not a limitation.

Access control

PayString protocol MUST not be used to provide payment account(s) information corresponding to a PayString URI unless providing that data via PayString protocol by the relevant PayString server was explicitly authorized by the PayString owner. If the PayString owner wishes to limit access to information, PayString servers MAY provide an interface by which PayString owners can select which information is exposed through the PayString server interface. For example, PayString servers MAY allow PayString owners to mark certain data as “public” and then utilize that marking as a means of determining what information to expose via PayString protocol. The PayString servers MAY also allow PayString owners to provide a whitelist of users who are authorized to access the specific information. In such a case, the PayString server MUST authenticate the PayString client.

Payment address rotation

The power of PayString protocol comes from providing a single place where others can find payment account(s) information corresponding to a PayString URI, but PayString owners should be aware of how easily payment account information that one might publish can be used in unintended ways. As one example, one might query a PayString server only to see if a given PayString URI is valid and if so, get the list of associated payment account information. If the PayString server uses the same payment address each time, it becomes easy for third-party to track one's entire payment history. The PayString server SHOULD follow the best practice of payment address rotation for every query to mitigate this privacy concern.

On the wire

PayString protocol over HTTPS encrypts the traffic and requires mutual authentication of the PayString client and the PayString server. This mitigates both passive surveillance (RFC 7258) and the active attacks that attempt to divert PayString protocol queries to rogue servers.

Additionally, the use of the HTTPS default port 443 and the ability to mix PayString protocol traffic with other HTTPS traffic on the same connection can deter unprivileged on-path devices from interfering with PayString operations and make PayString traffic analysis more difficult.

In the PayString server

The Basic PayString protocol data contains no information about the PayString client; however, various transports of PayString queries and responses do provide data that can be used to correlate requests. A Basic PayString protocol implementation is built on IP, TCP, TLS and HTTP. Each layer contains one or more common features that can be used to correlate queries to the same identity.

At the IP level, the PayString client address provides obvious correlation information. This can be mitigated by use of NAT, proxy, VPN, or simple address rotation over time. It may be aggravated by use of a PayString server that can correlate real-time addressing information with other identifiers, such as when PayString server and other services are operated by the same entity.

PayString client implementations that use one TCP connection for multiple PayString requests directly group those requests. Long-lived connections have better performance behaviours than short-lived connections; however they group more requests, which can expose more information to correlation and consolidation. TCP-based solutions may also seek performance through the use of TCP Fast Open (RFC 7413). The cookies used in TCP Fast open may allow PayString servers to correlate TLS connections together.

TCP-based implementations often achieve better handshake performance through the use of some form of session resumption mechanism, such as Section 2.2 of RFC 8446. Session resumption creates a trivial mechanism for a server to correlate TLS connections together.

HTTP's feature set can also be used for identification and tracking in a number of ways. For example, Authentication request header fields explicitly identify profiles in use, and HTTP cookies are designed as an explicit state-tracking mechanism and are often used as an authentication mechanism.

Additionally, the “User-Agent” and “Accept-Language” request header fields often convey specific information about the PayString client version or locale. This allows for content-negotiation and operational work-arounds for implementation bugs. Request header fields that control caching can expose state information about a subset of the client's history. Mixing PayString queries with other HTTP requests on the same connection also provides an opportunity for richer data correlation.

The PayString protocol design allows implementations to fully leverage the HTTP ecosystem, including features that are not enumerated in this document. Utilizing the full set of HTTP features enables PayString to be more than HTTP tunnel, but it is at the cost of opening up implementations to the full set of privacy considerations of HTTP.

Implementations of PayString clients and servers need to consider the benefits and privacy impacts of these features, and their deployment context, when deciding whether or not to enable them. Implementations are advised to expose the minimal set of data needed to achieve the desired feature set.

Determining whether or not PayString client implementation requires HTTP cookie (RFC 6265) support is particularly important because HTTP cookies are the primary state tracking mechanism in HTTP, HTTP cookies SHOULD NOT be accepted by PayString clients unless they are explicitly required by a use case.

Overall, the PayString protocol does not introduce privacy concerns beyond those associated with using the underlying IP, TCP, TLS and HTTP layers.

Verifiable PayString Protocol

Verifiable PayString protocol, an extension to Basic PayString protocol, provides payment address information associated with a PayString while allowing involved parties to exchange “identity” information, proof of ownership of on-ledger keys, and non-repudiable cryptographic proof of the entire exchange. It can be used to enable trust-minimized and trust-free security regimes and has applications in both custodial and non-custodial settings. More specifically, verifiable PayString protocol provides the following enhancements to the Basic PayString protocol:

- Verifiable Custodial PayString service: allows custodial wallets and exchanges to send payment address information and other resources digitally signed with their off-ledger private key.
- Verifiable Non-Custodial PayString service: allows non-custodial wallets and exchanges to send payment address information digitally signed with the off-ledger private key of the PayString owner along with PayString owner's “identity” information.
- Privacy-enhanced PayString service: allows PayString service providers (both custodial and non-custodial) to deploy appropriate access control mechanisms by allowing the PayString clients or senders to transmit their “identity” information for authentication.

Basic PayString protocol protocol relies on the underlying secure transport (TLS 1.3) to ensure message integrity and privacy from network attackers. There are at least two assumptions in the security and privacy model of the basic PayString protocol that are less desirable.

1. Trust requirement between the PayString client and PayString server: As pointed out in the Basic PayString security model section, PayString server has full control over the contents of the response message, and may go rogue or be compromised. The PayString client has no way of knowing if the PayString server behaves maliciously. This implicit trust assumption between the PayString client and server is not ideal in the world where the information provided by the PayString server may be used by the PayString client to transmit money.
2. Privacy: Per Basic PayString protocol, anyone can query the PayString server and retrieve the payment address information corresponding to the queried PayString. The PayString server or PayString owner has no way of deploying access control mechanisms since the
“identity” of the PayString client and the sender is unknown to the PayString server.

The motivation for verifiable PayString protocol is the following:

1. Eliminate the implicit trust assumption between the PayString client and custodial PayString server: While it is not possible for any protocol to prevent custodial PayString server or PayString client from acting maliciously, the best we can do is to allow for mechanisms in the protocol that enables PayString client and server to prove this

misbehaviour to third-parties and potentially hold the other party legally accountable for misbehaving.

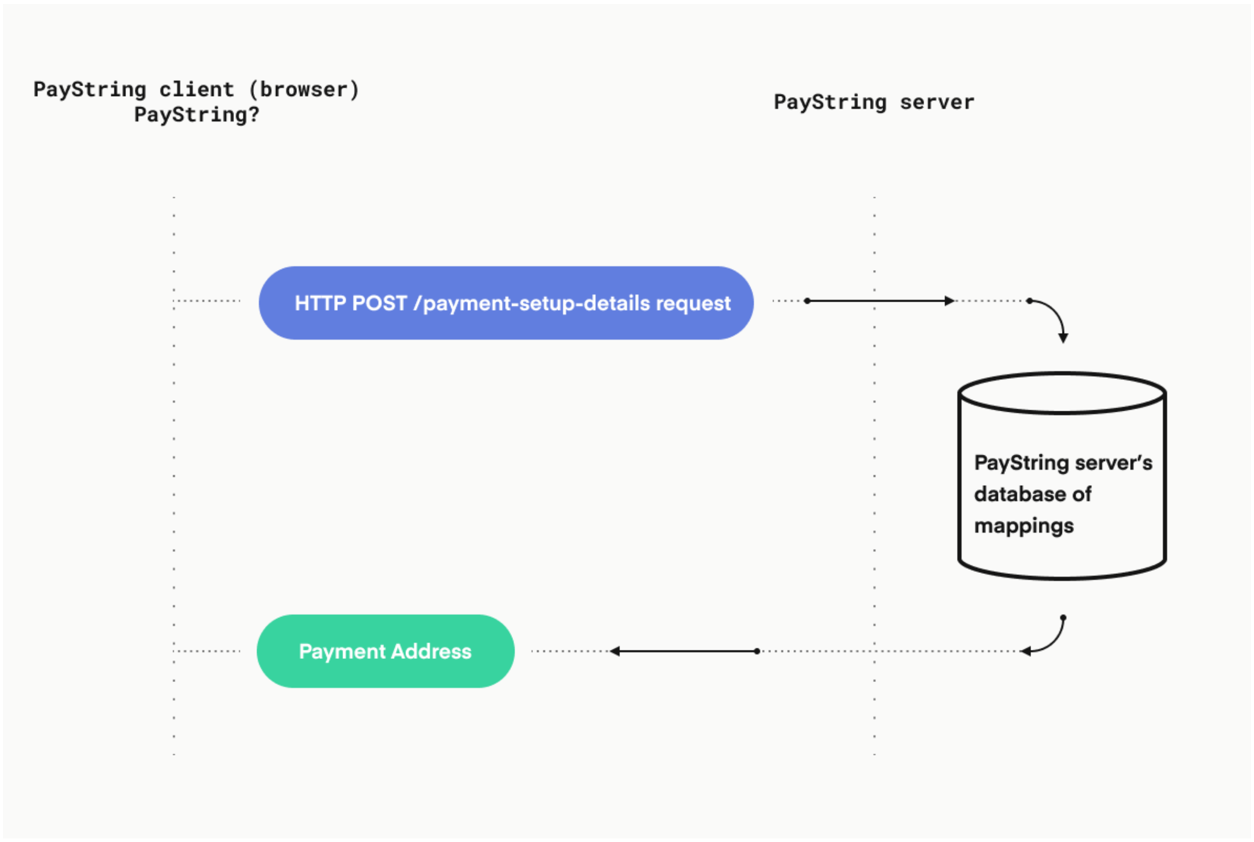
2. Ensure that if the PayString server is compromised, an attacker can not swap payment addresses in the payment account information response and redirect funds to the attacker controlled payment network and address. Allows the custodial PayString server to pre-sign PaymentInformation in a cold/airgapped system offline instead of online on a hot wallet.

3. Allows for non-custodial service providers to run non-custodial PayString service by allowing the PayString owners to digitally sign the PaymentInformation locally on their device with their off-ledger private keys and send PayString owner's “identity” information in the response. This information can then be used by the PayString client and sender to authenticate the PayString owner and decide if they want to proceed with the transaction.

4. Enhance privacy of the PayString protocol by allowing the PayString client to share their and the sender's “identity” information with the request to the PayString server. This information could then be used to:
    a. Give the PayString owner and/or PayString server the ability to decide if they want to share their payment address information and other resources with the PayString client or the sender.
    b. Allow for an open standards based way for endpoints to keep verifiable records of their financial transactions, to better meet the needs of accounting practices or other reporting and regulatory requirements.

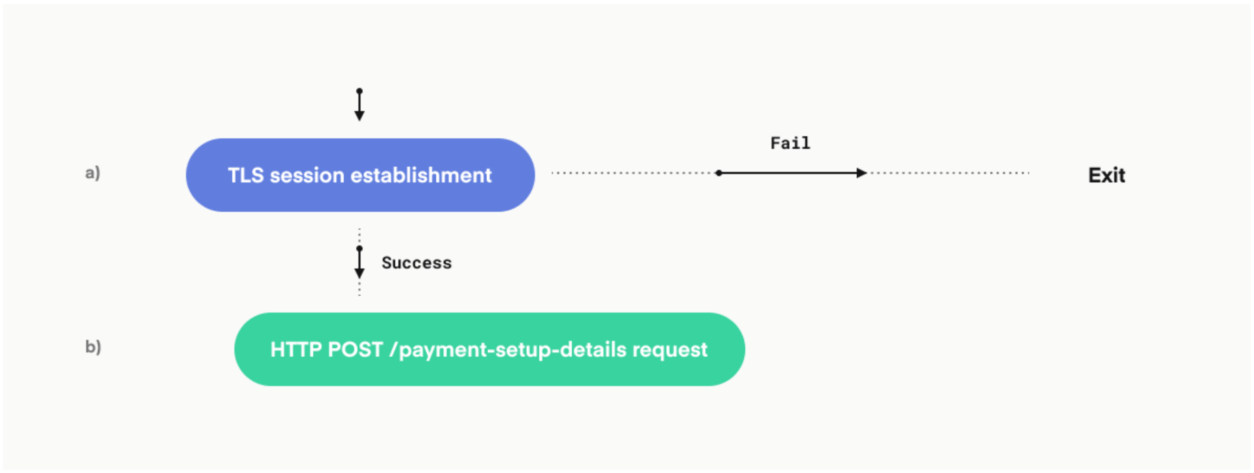
Verifiable PayString protocol flow

Verifiable PayString protocol can be used by PayString clients to easily query PayString servers for retrieving digitally signed payment address information responses and other resources corresponding to human-readable PayStrings.


PayString client (browser)
PayString?
PayString server
HTTP POST /payment-setup-details request
PayString server's database of mappings
Payment Address

The following steps describe how the verifiable PayString client retrieves the signed payment address(es) corresponding to a PayString.[3]

1) **POST /payment-setup-details request**: Processing steps by verifiable PayString client to send an HTTP "POST" request


a)
TLS session establishment
Fail
Exit
Success
b)
HTTP POST /payment-setup-details request

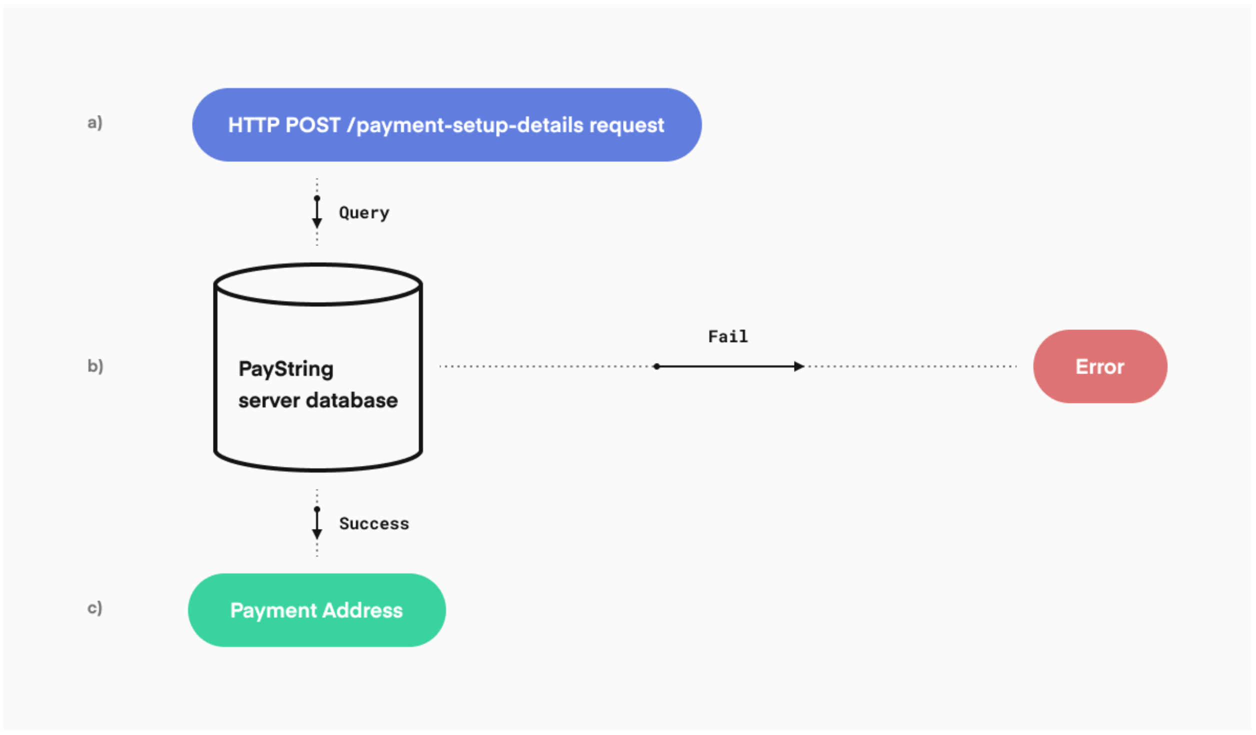
[3] How a client obtains PayString is out-of-scope of this protocol. Instead of a PayString, the client can also use its corresponding URL directly, as described in the syntax resolution section.

a) Establish a secure, mutually authenticated TLS 1.3 session with the PayString server as described in the session establishment section. The URL of the PayString server is derived from the PayString as described here.
b) If the TLS session is successfully established, prepare an InvoiceRequest message with any optional fields. The optional fields in the message body would depend on the use-case. E.g. for the simple use-case of a PayString client meaning to send a transaction to PayString owner, the request body may contain the
    i) "identity": The type/value of the "identity" field is TBD. We anticipate this being a mechanism for the PayString client to transmit their or sender's "identity" information to the PayString server. This information can then be used by the PayString server/PayString owner to:
        - Enhance privacy by exercising access control mechanisms such as authorized access via accept/deny lists, etc. for the PaymentInformation or other resources for a PayString.
        - Record-Keeping

The PayString client MUST specify the payment-network and environment they support via the HTTP "Accept" header. They MUST specify the PayString version via "PayString-version" header. For details on PayString request headers refer to the HTTP Request and Response Headers.

The PayString client sends the InvoiceRequest message using HTTP "POST" method to the PayString URL with path parameter "payment-setup-details" over the established secure channel.

2) Payment Address Response: Processing steps by verifiable PayString server to generate the Payment Address response corresponding to the queried PayString:


a)
HTTP POST /payment-setup-details request
Query
b)
PayString server database
Fail
Error
Success
c)
Payment Address

a) Receive the POST /payment-setup-details request from PayString client with optional body.
b) Query its database for the queried PayString. If the Payment Address information corresponding to the queried payment-network and environment exists in the database, the PayString server generates the PaymentInformation response encapsulated in the SignatureWrapper with appropriate response headers as described in and HTTP Request and Response Headers section. Otherwise it generates an Error message. For details on error codes refer to the PayString protocol status communication section.
c) Send PaymentInformation response or Error message generated in the previous step to the PayString client.

Verifiable PayString protocol as a trustless solution for custodial and non-custodial service providers

We anticipate that the most common use-case for retrieving "PaymentInformation" is to make transactions. We can categorize the providers of such services as follows:

- Custodial wallets and exchanges: Custodial wallets and exchanges hold the private keys of their customers on their servers and essentially hold their funds. There is an implicit trust between the custodial service provider and their customers.
- Non-Custodial wallets and exchanges: Non-custodial wallets and exchanges do not store their customers' keys on their servers. The customers hold their private keys locally on their device. There is a no trust requirement between the non-custodial wallets and exchanges and their customers. Since the customers hold the private keys the wallets are not liable for any consequences coming from the lost, compromised or hacked

private keys of the customers. Nor do they need their customers to trust their servers in case wallet's servers go malicious or are compromised.

Notice that the custodial and non-custodial service providers operate under different trust models. To continue operating under the same trust model, verifiable PayString requires slightly different treatment for the two.

Verifiable PayString protocol preserves these trust models. Consequently, a non-custodial wallet running a PayString server has no liability for providing accurate “PaymentInformation”, i.e. the “PayString --> Payment Address” mappings, for their customers that is signed with the private key of the non-custodial PayString server wallet. Instead, the PayString owners or the customers can generate this signed mapping with their own off-ledger private key locally on their app/device. The PayString client can easily verify this signature based on the trust relationship between the sender of the payment (PayString client wallet’s customer) and the receiver (non-custodial PayString server's wallet). This eliminates any risk of the non-custodial PayString server wallet losing its private keys, going malicious, getting hacked, or becoming otherwise compromised in a way that customers might lose funds.

Distributing identity key

There are two general approaches to associating an identity key with a PayString so that request responses to that PayString can be verified in a trustless manner.

The first approach is to include or point to an identity key in the response itself. This approach preserves the user-friendly readability of the PayString and provides flexibility to include metadata that can help interpret the identity key.

The following table enumerates the possible ways to share the public key of PayString owner using “identity” field.

identity	**Description**
Global Identifier (GiD), Human Universally Unique Identifier (Human UUID), Digital Identifier (DID)	Digital identifier or an attested certificate that associates digital identifier to PayString and public key
URL	URL for secure retrieval of public key of the PayString owner
Public Key	Public key of PayString owner

- Digital identifier/Attested Certificate: A global digital identifier that uniquely associates the “PayString owner's identity” as defined by the identifier (GiD, Human UUID, DID, etc.) to the “PayString” and “public key”. The PayString client can then verify the “public key” using the digital identifier. This could be a direct retrieval of the corresponding “public key” from a digital identity service provider if PayString is a part of that digital identifier or an attested certificate that associates digital identifiers such as GiD, Human UUID, DID, etc. to the “PayString” and “public key”.

- URL: A URL for secure retrieval of “public key” of the PayString owner.
- Public Key: This could be
  - Public key that has been pre-shared between the PayString client and PayString owner. E.g. PayString client and PayString owner could use popular messaging systems such as Signal Messaging App/ WhatsApp and use their long term identity keys that are shared between the PayString client and server using out-of-band public-key fingerprint matching or Trust On First Use (TOFU). The extensive reach and trust in these apps makes them a good potential solution.
  - Certificates verifying the PayString owner’s centralized web PKI keys, decentralized PGP keys or Blockchain-based PKI keys.

The second approach is to embed the public key of PayString owner in specialized identity PayStrings that point to other PayStrings.

This approach entails reserving the hostname “pkh” for “public key hashes” and supporting a PayString format of the form “public_key_hash”$pkh.provider.domain. PayString client implementations would require that any “PaymentInformation” resource that resulted from the PayString of that form be signed with the “private key” corresponding to that “public key hash”, so only a “PaymentInformation” signed by the owner of the PayString is valid.

The caveat is that this PayString format is not human-readable anymore. The solution is simple: the non-custodial wallets and exchanges would provide a non-human-readable PayString of the form `public_key_hash`$pkh.provider.domain, but the customers may get a human-readable PayString from another trusted service providers (say from their email provider) that maps to the non-human-readable PayString they got from their non-custodial service-provider. Non-custodial service-providers could even automate this process by allowing the user to choose a mapping provider.

Verifiable PayString protocol extensions

In this section, we describe two extensions to the verifiable PayString protocol: one for invoices and receipts and one for satisfying compliance requirements for payments.

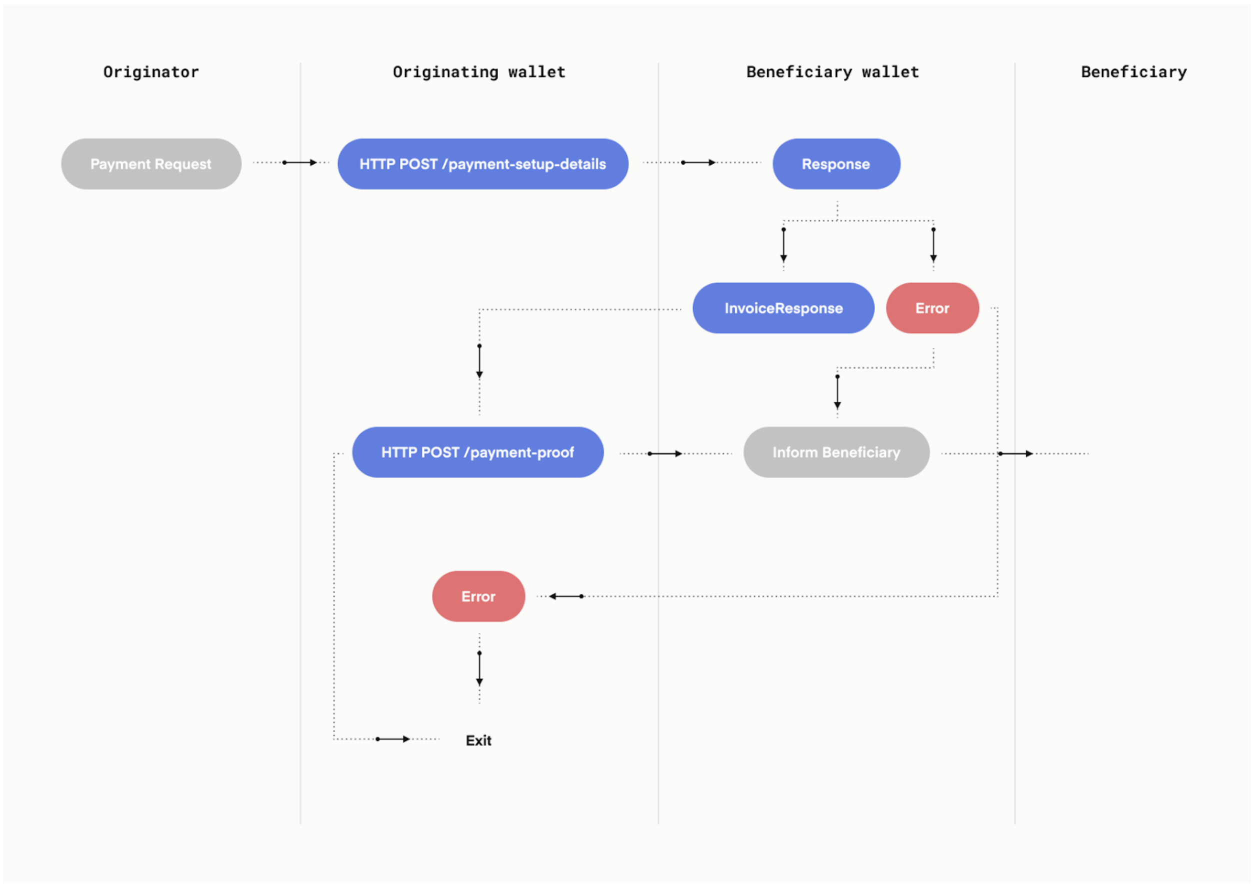
Terminology

Additional terminology used in the following section:

Beneficiary/Receiver/PayString owner: Individual or Entity receiving the transaction/owner of the PayString
Beneficiary wallet/PayString server: Receiving Endpoint; receives transaction on behalf of the Beneficiary (custodial or non-custodial)
Originating wallet/PayString client: Sending Endpoint; initiates transaction on behalf of the Originator (custodial or non-custodial)
Originator/Sender: Individual or entity originating the transaction.

Third-party verifiable cryptographically signed proof-of-payment and receipt-of-payment

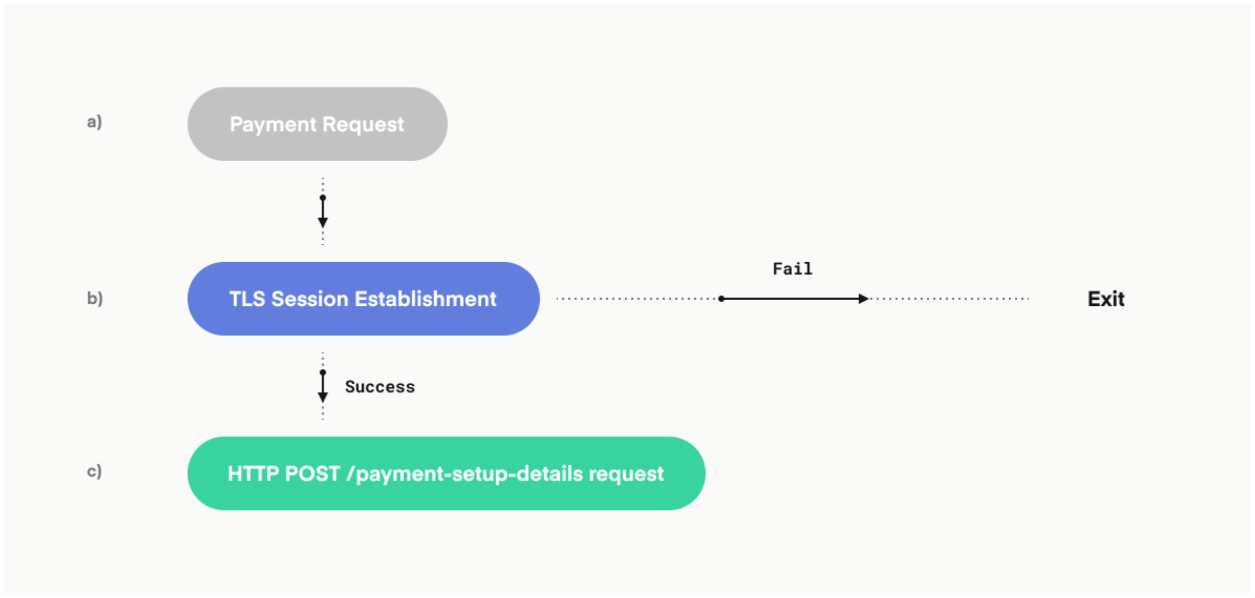
The first extension is for the case of making payments to include cryptographically signed invoice requests, invoice response, proof-of-payment and receipt-of-payment.



In case the Originating wallet wants to make a payment with a signed proof of PaymentInformation from the Beneficiary or the Beneficiary wallet, they would generate an InvoiceRequest message for the Beneficiary wallet [4].

1) **POST /payment-setup-details request**: Processing steps by the Originating wallet to generate InvoiceRequest message:

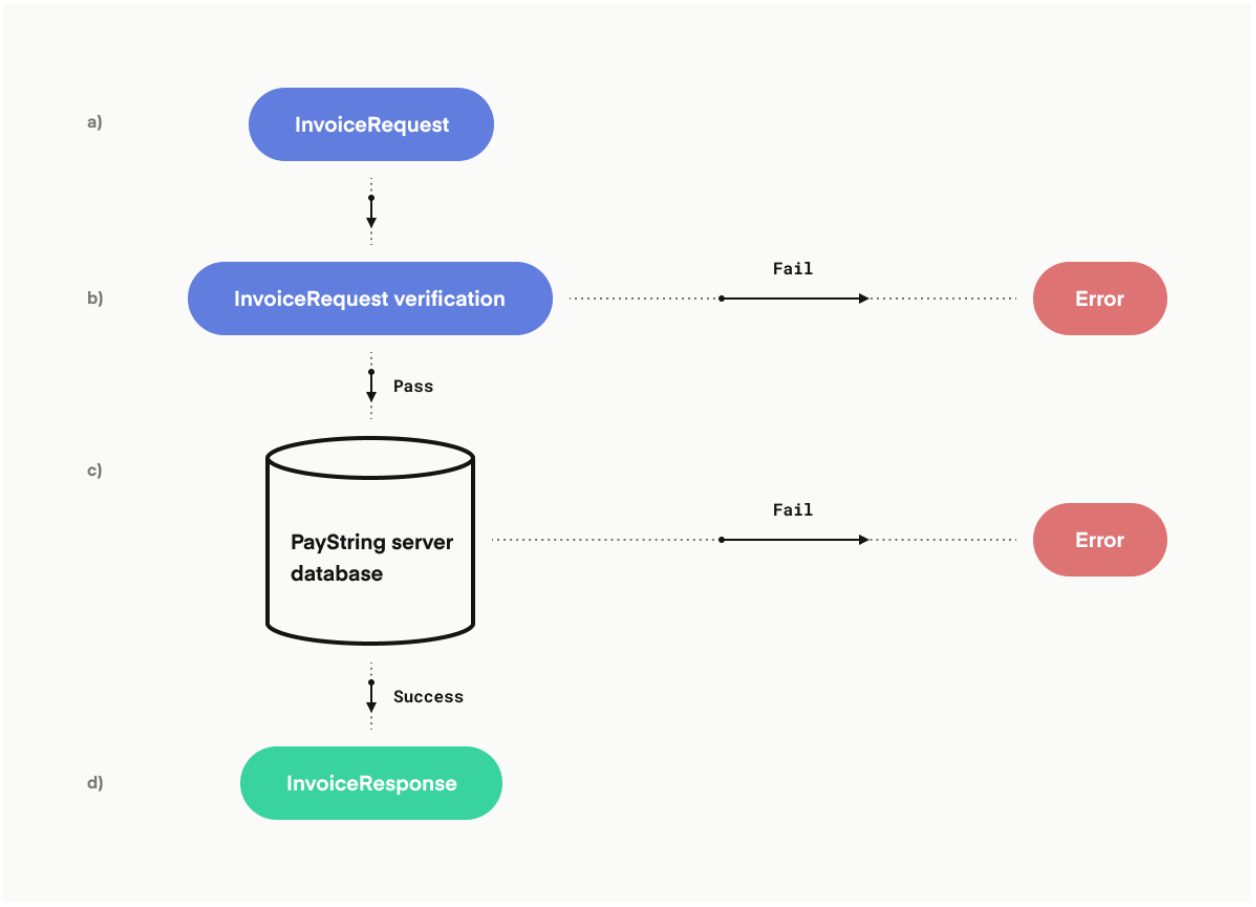
[4] In this specific flow, we assume that both end-points are non-VASP entities


a)
Payment Request
b)
TLS Session Establishment
Fail
Exit
Success
c)
HTTP POST /payment-setup-details request

Prerequisite: Receive the Payment Request from the Originator with Beneficiary's PayString and other relevant information such as amount, etc. (This is not a part of the PayString protocol flow.)

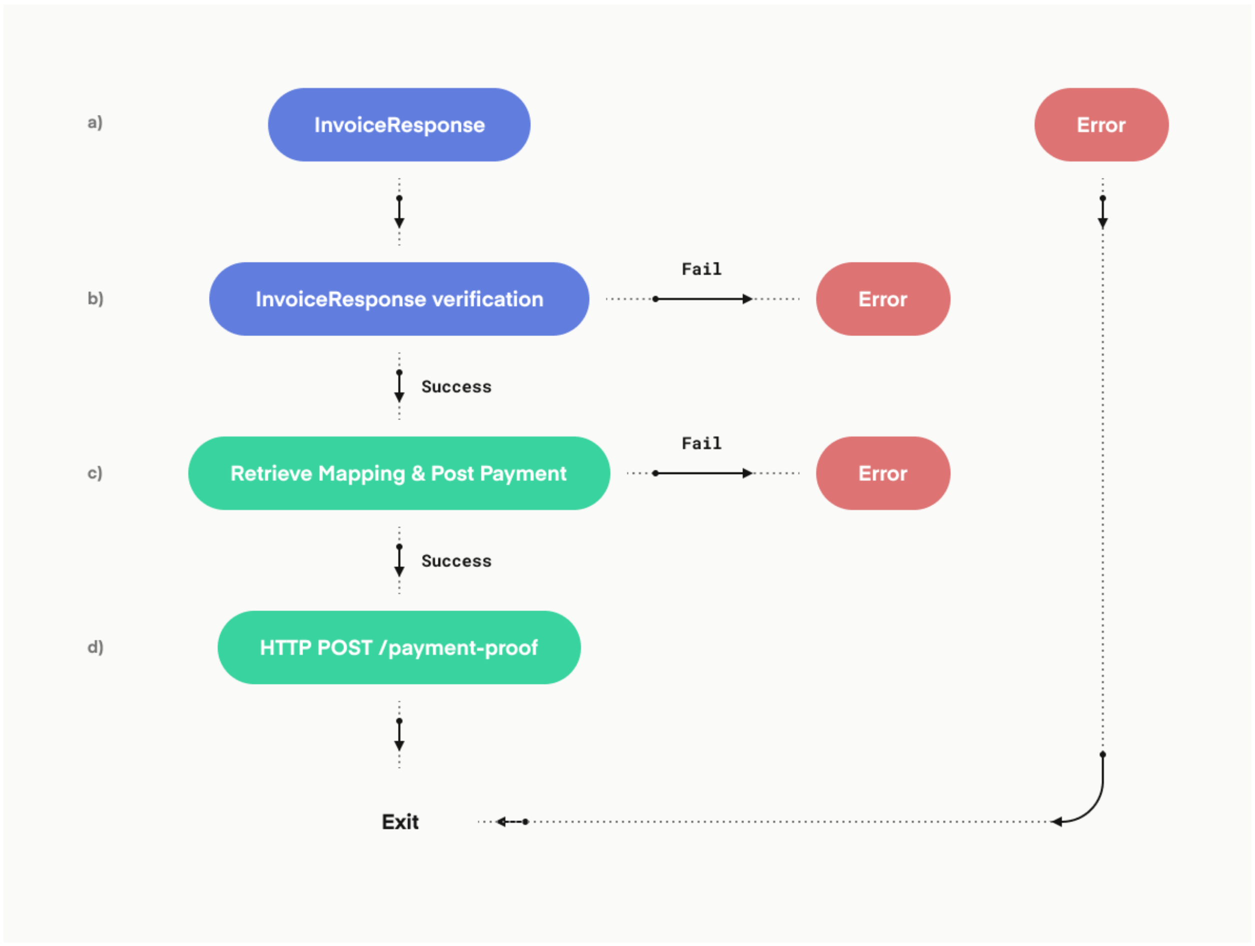
a) Establish a secure and mutually authenticated TLS 1.3 session with the Beneficiary wallet as described in the session establishment section. The URL of the Beneficiary wallet is derived from Beneficiary's PayString as described here.
b) If the TLS session is successfully established, prepare an InvoiceRequest message encapsulated in SignatureWrapper. The Originating wallet must include any optional relevant fields that are required to generate an invoice response.
c) Send HTTP "POST" request with path parameter "payment-setup-details" and InvoiceRequest as message body over the established secure channel.

2) **Response**: Processing steps by Beneficiary wallet to generate a Response corresponding to the InvoiceRequest:



a) Receive the InvoiceRequest message from the Originating wallet.
b) Verify if the incoming message has all the mandatory data in valid format and is correctly signed as described in the verifying InvoiceRequest message**.**
c) If the InvoiceRequest message passes verification, the Beneficiary wallet queries its PayString server database for the cryptographically signed 'beneficiaryPayString → payment address' information corresponding to the queried PayString and payment-network and environment. This database of payment information is generated by the Beneficiary wallet as described in the PaymentInformation section.
d) If the payment information corresponding to the queried PayString exists in the database, then the Beneficiary wallet generates a cryptographically signed InvoiceResponse message encapsulated in SignatureWrapper otherwise it generates an Error message. For details on error codes refer to the PayString protocol status communication section. Send InvoiceResponse or Error message to the Sending Endpoint.

3) **POST /payment-proof [optional]:** Processing steps by the Originating wallet to generate the PaymentProof message as a proof of payment on the payment address provided by the Beneficiary wallet in the InvoiceResponse message:



a) Receive InvoiceResponse or Error message from the Beneficiary wallet
b) Verify if the incoming message has all the mandatory data in valid format and is correctly signed as described in the verifying InvoiceResponse message or verifying Error message section.
    i) If it is an Error message and it verifies, exit. If the Error message does not pass verification, drop the message
    ii) Otherwise, if verification for InvoiceResponse message passes, goto (c).
c) Retrieve the payment address from the *paymentInformation* field of the InvoiceResponse message and post the transaction on the corresponding address.
d) If the payment succeeds, then obtain the corresponding transaction ID and generate a cryptographically signed PaymentProof message encapsulated in SignatureWrapper. Otherwise generate an Error message. For details on error codes refer to the generating PayString protocol status communication section. Send POST /payment-proof or Error message to the Beneficiary wallet.

2) **PaymentReceipt [optional]:** Upon receiving the signed PaymentProof message from the Originating wallet, the Beneficiary wallet may choose to send a PaymentReceipt message as a receipt of payment. Following are the processing steps by Beneficiary wallet to generate PaymentReceipt:

a. Receive PaymentProof message or an Error message from the Originating wallet.
b. Verify if the incoming message has all the mandatory data in valid format and is correctly signed as described in the verifying PaymentProof message or verifying Error message section.
    i. If it is an Error message and it passes verification, exit. Otherwise if the Error message does not pass verification, then drop the message.
    ii. Otherwise if the PaymentProof message passes verification, goto (c).
c. Retrieve the *transactionConfirmation* field from the PaymentProof message and confirm the transaction on the corresponding payment network.
d. If the payment exists, generate a cryptographically signed PaymentReceipt message. Send PaymentReceipt or Error message response to the Originating wallet.

Verifiable PayString protocol with compliance extensions

The verifiable PayString allows us to easily extend cryptographically verifiable invoice requests and responses for financial institutions to use with their array of compliance requirements.

Of particular relevance, increasing regulatory scrutiny has introduced additional compliance and legal issues for the cryptocurrency industry and more of such regulations are anticipated in the future. As a result, there is a pressing need to come up with a messaging standard between the transacting entities that are required to meet such requirements to agree on a mechanism that:

a) Allows the entities to communicate to each other their respective compliance requirements.
b) Securely send (and store) required information/data if the entities indicate that they fall under the umbrella of such requirements.

Accordingly, we present an extension to the verifiable PayString protocol that provides a standard mechanism to meet the current and potential future compliance and legal requirements along with cryptographic signed proofs that can be stored by both entities involved in a transaction as a record of their compliance.

The most salient compliance need facing the cryptocurrency space is the Travel Rule, which requires financial institutions to exchange information on senders and receivers of the covered transactions. In the US, FinCEN has indicated heightened focus on enforcing the Travel Rule, while FATF recommended in June that the Travel Rule be enforced for Virtual Asset Service Providers (“VASPs”)[5] starting in mid-2020.

While relatively straightforward for traditional payment rails such as wire or ACH, Travel Rule compliance is non-trivial for VASPs. When a user asks a service to send to an on-ledger address, it is exceedingly difficult for the VASPs to determine who owns the address, whether Travel Rule applies, and how to contact the owner of the address if it does. The challenge is to

[5] https://www.fatf-gafi.org/glossary/u-z/

come up with a lightweight messaging protocol that is both secure and private from third parties and does not require any trust relationships between the transacting VASPs.

We show how verifiable PayString protocol can easily be extended to accommodate the Travel Rule that allows the participating entities to indicate to each other if they are a VASP or not and to send and store the required Travel Rule information. The protocol requires no trust between the participating entities and is both secure and private from third parties. We provide non-deniable, publicly verifiable cryptographically signed proofs that can be stored by both VASPs involved in a transaction as record of their compliance with the Travel Rule.[6]

Note: In this paper, we describe PayString protocol flow for Travel Rule compliance specifically but our protocol can be extended to exchange information for other compliance requirements with little to no change.

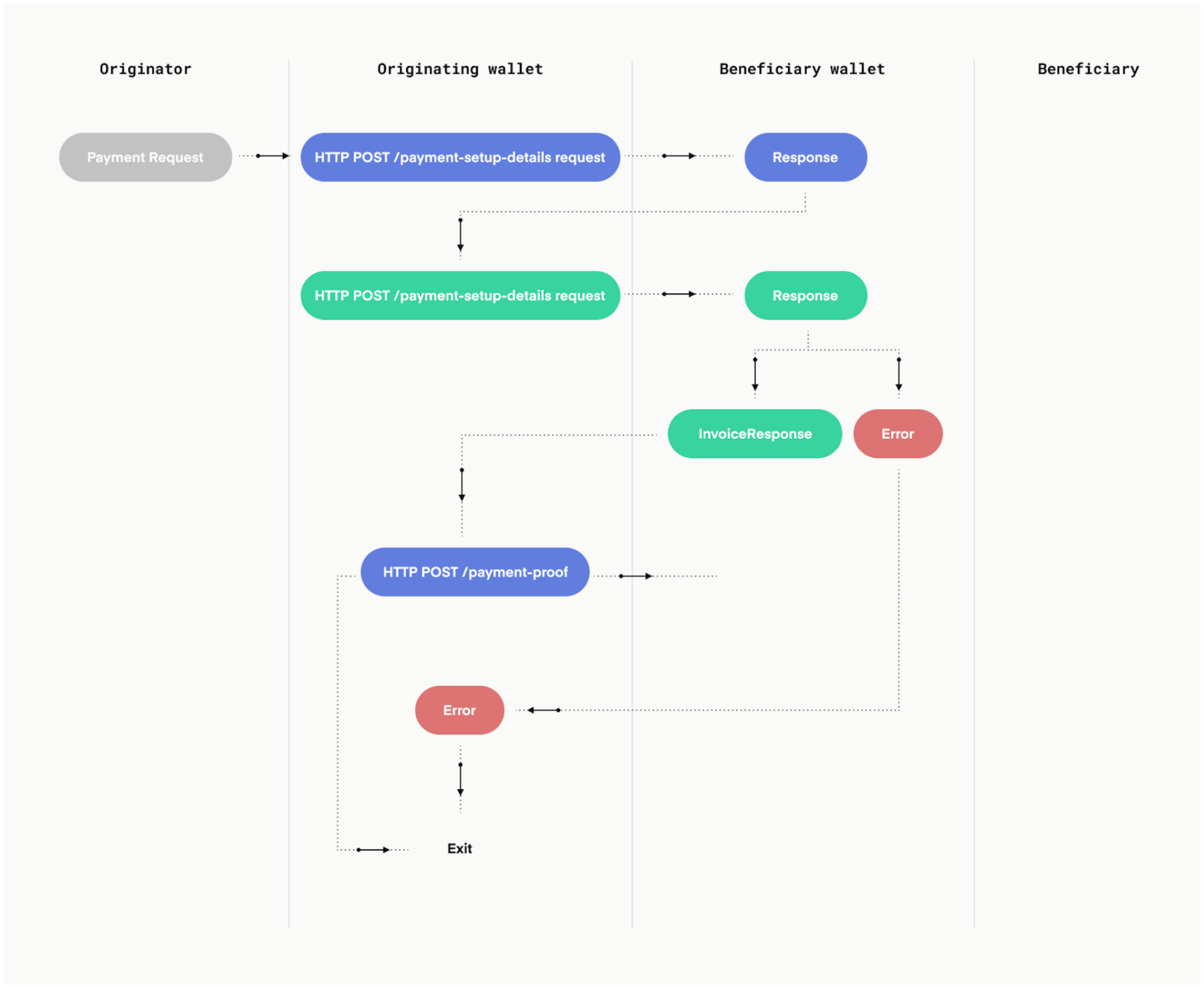
Terminology

Additional terminology in context of Travel rule used in the following sections.

Beneficiary Institution: Receiving Endpoint that is a VASP; receives transaction on behalf of the Beneficiary.
Covered Institution: Entity that must comply with some set of regulatory requirements.
Covered Transaction: Transaction that is subject to compliance with Travel Rule requirements.
Endpoint: either the client or the server of the connection
Sending Endpoint: sending side of the transaction (VASP or non-VASP)
Receiving Endpoint: receiving side of the transaction (VASP or non-VASP)
Originating Institution: Sending Endpoint that is a VASP; initiates transaction on behalf of the Originator.

Prerequisite[7] [8]: Before the PayString protocol flow begins, the Originator sends the Payment Request that MUST include the Beneficiary's PayString and transaction details (amount, etc.) and MUST include any information about the Beneficiary and the Beneficiary Institution (where applicable) to the Originating Institution over a pre-established secure channel (e.g. through the service's web app)[9]

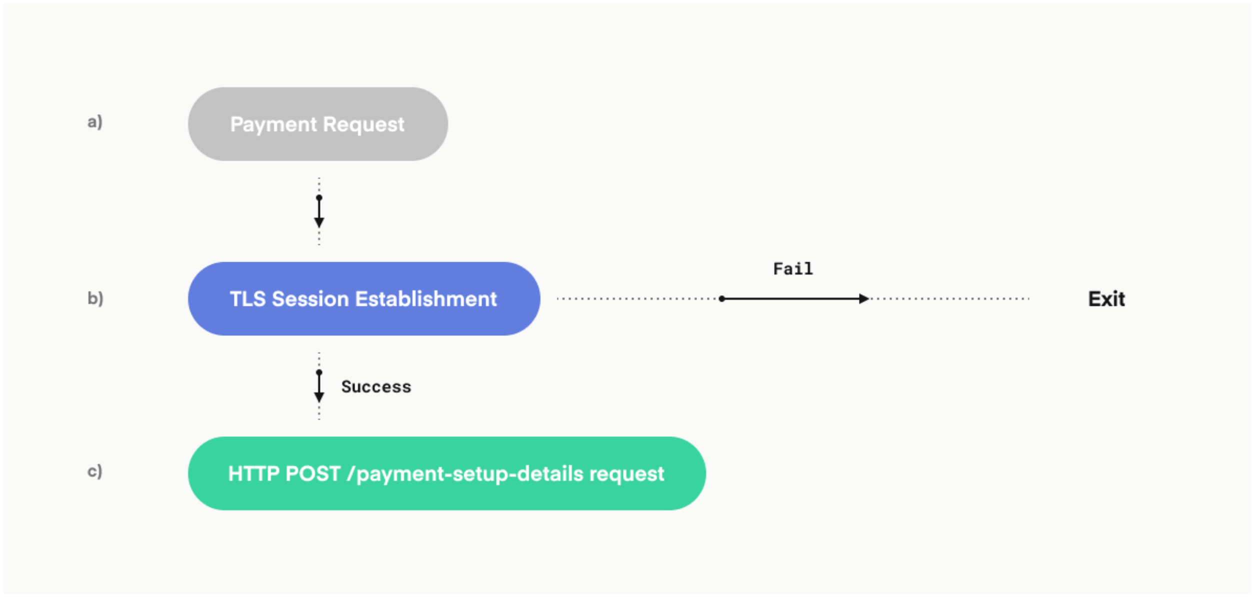
[6] It is up to the user how and for how long they wish to store these proofs and other data artifacts. Per https://www.law.cornell.edu/cfr/text/31/1010.410, they can be required to retain records for up to 5 years in the US.
[7] For this following flow, we assume that both endpoints are covered institutions and the transaction is covered transaction
[8] This is not a part of the PayString protocol flow.
[9] At this stage i.e. before initiating the protocol flow, based on the information provided by the Originator to the Originating Institution about the identity of the Beneficiary and/or Beneficiary Institution, the Originating Institution has the opportunity to run checks such as sanctions screening, blacklists, etc. and decide if they want to proceed with the transaction. If the Originating Institution decides against, they MAY optionally inform the Originator of the reason for the failed transaction.



Verifiable PayString protocol with Travel Rule flow begins here[10]
Travel Rule handshake begins here

1) **POST /payment-setup-details request**: Processing steps by Originating Institution to generate InvoiceRequest message for the Beneficiary Institution:

[10] In this flow, we describe the case when both endpoints are covered institutions and the transaction amount is above a threshold that kicks in Travel Rule, i.e. it's a covered transaction.


a)
Payment Request
b)
TLS Session Establishment
Fail
Exit
Success
c)
HTTP POST /payment-setup-details request

a) Upon receiving the Payment Request from the Originator, establish a TLS session with the Receiving Endpoint as described in the session establishment section. The URL of the Receiving Endpoint is derived from Beneficiary's PayString as described in the syntax resolution section above.
b) If the TLS session is successfully established, generate the InvoiceRequest message encapsulated in the SignatureWrapper. Since the Originating Institution is a VASP, they MUST provide their identity information, amount of transaction and set the *isVASP* field to True in the InvoiceRequest message body. Additionally, they MAY provide any other relevant information in the "memo" field.
c) Send HTTP "POST" request with path parameter "payment-setup-details" and InvoiceRequest as message body to the Receiving Endpoint.

2) **Response**: Processing steps by the Receiving Endpoint:
   a) Receive the InvoiceRequest message from the Originating Institution.
   b) Verify if the incoming message has all the mandatory data in valid format and is correctly signed as described in the verifying InvoiceRequest message**.**
   c) If the InvoiceRequest message fails verification, generate an Error message as described in the generating Error message section.
   d) Otherwise if the Receiving Endpoint is a Beneficiary Institution, the Beneficiary Institution MAY run checks such as sanctions screening, blacklist checks, etc. on the Originating Institution based on any identity information received in the InvoiceRequest and decide if it wants to proceed with the transaction.
      i) If it does want to proceed, the Beneficiary Institution queries its database for the cryptographically signed 'beneficiaryPayString → payment address' information corresponding to the queried PayString and payment-network and environment. This database of payment information is generated by the beneficiary wallet as described in the PaymentInformation section. If the payment information exists in the

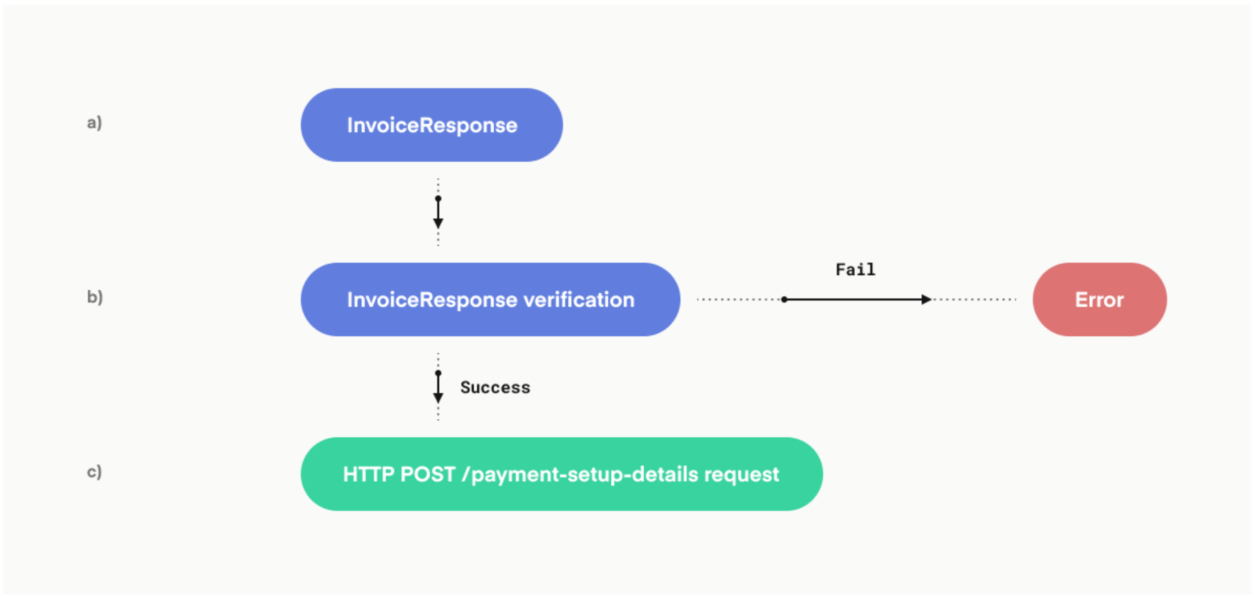
database, then the Beneficiary Institution generates InvoiceResponse encapsulated in SignatureWrapper that includes

(1) compliance requirement for Travel Rule in the list of *complianceRequirements* field.
(2) its identity information to the Originating Institution.
(3) The empty *paymentInformation* field. The Beneficiary Institution MUST NOT send the payment address information yet.

Otherwise if the payment information does not exist in the database it generates an Error message and exit. For details on error codes refer to the generating PayString protocol status communication section. Send InvoiceResponse or Error message to the Originating Institution.

ii) If the Beneficiary Institution decides not to proceed with the transaction, it MAY generate an Error message and exit. For details on error codes refer to the generating PayString protocol status communication section. Send the Error message to the Originating Institution and MAY optionally inform the Beneficiary of the failed transaction.

3) **POST /payment-setup-details request (upgraded)**: Processing steps by the Originating Institution:



a)
InvoiceResponse
b)
InvoiceResponse verification
Fail
Error
Success
c)
HTTP POST /payment-setup-details request

a) Receive InvoiceResponse or Error message from the Beneficiary Institution.
b) Verify if the incoming message has all the mandatory data in valid format and is correctly signed as described in the verifying InvoiceResponse or verifying Error message section.

i) If it is an Error message and it verifies, exit. If the Error message does not verify, drop the message.

ii) Otherwise, if the InvoiceResponse message passes verification, goto (c). If verification for InvoiceResponse fails, generate an Error message exit. For details on error codes refer to the generating PayString protocol status communication section.

c) The Originating Institution MAY run checks such as sanctions screening, blacklist, etc. on the Beneficiary Institution based on any identity information received in the InvoiceResponse and decide if it wants to proceed with the transaction.

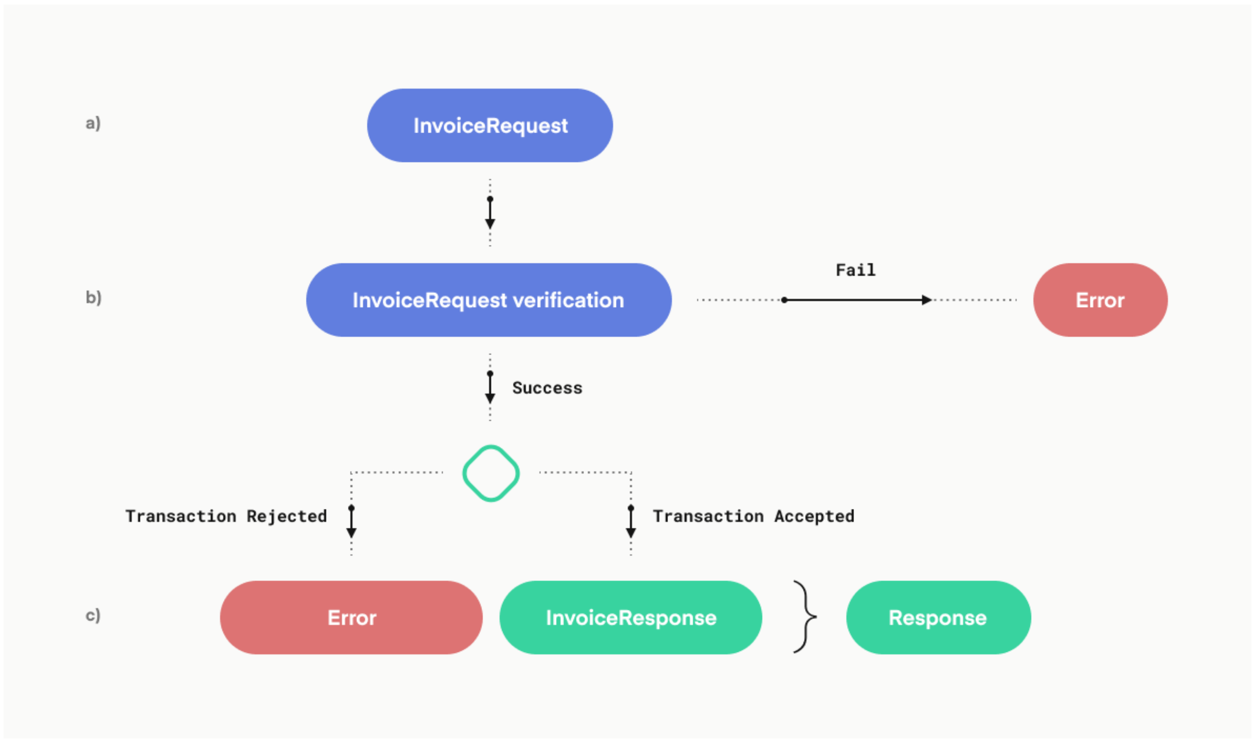
i) If it does want to proceed, the Originating Institution generates a cryptographically signed upgraded invoice request message with a body as described in the ComplianceData and SignatureWrapper sections that among other fields includes:

(1) Travel Rule data payload that contains the required data/information to be transferred as described in the TravelRule section.

(2) Previous InvoiceResponse message received from the Beneficiary Institution in step (2).

Send the upgraded POST /payment-setup-details request message to the Beneficiary Institution.

4) **Response**: Following are the processing steps by the Beneficiary Institution to generate an upgraded InvoiceResponse message in response to the upgraded InvoiceRequest from the Originating Institution that contains the Travel Rule payload:



a) Receive upgraded invoice request (ComplianceData) message or an Error message from the Originating Institution.

b) Verify if the incoming message has all the mandatory data in valid format and is correctly signed as described in the verifying ComplianceData or verifying Error message sections. If it is an error message and it verifies, exit. If it is a ComplianceData message and it fails verification, generate an Error message and exit. For details on error codes refer to the generating PayString protocol status communication section.

c) Otherwise if the ComplianceData message passes verification,

    i) Retrieve the compliance data sent by the Originating Institution in the ComplianceData message which includes the Originator's identity information. The Beneficiary Institution MAY run checks such as sanctions screening, OFAC, etc. on the Originator and decide if it wants to proceed with the transaction.

        (1) If the Beneficiary Institution decides that it wants to proceed with the transaction, then the Beneficiary Institution generates an upgraded InvoiceResponse encapsulated in SignatureWrapper that includes among other fields:

            (a) *paymentInformation* field containing the cryptographically signed 'beneficiaryPayString → payment address' information corresponding to the queried PayString in the Payment

            (b) empty *complianceRequirements* field

            (c) *previousMessage* field containing the previous upgraded invoice request (ComplianceData) message received from Originating Institution

            (d) [optionally] A *proofOfControlSignture* field containing the signature proving control over the destination address[11]

    Otherwise if the Beneficiary Institution decides to not proceed with the transaction, it MAY optionally generate an Error message and send it to Originating Institution and exit. For details on error codes refer to the generating PayString protocol status communication section. Originating Institution MAY also optionally inform the Beneficiary of the failed transaction.

Travel Rule handshake ends here.

Optionally perform Steps 3 and 4 as in Verifiable PayString protocol Extensions section to generate proof of payment and receipt of payment.

Verifiable PayString protocol with Travel Rule flow ends here

[11] This ensures that the Originating Institution knows they are communicating with the VASP that controls the on-ledger address.

Verifiable PayString protocol integration with Travel Rule Information Sharing Architecture (TRISA)[12]

While PayString can provide a standalone framework to satisfy compliance requirements, its flexible structure allows it to complement any other solution by providing for easier set up, enhanced privacy, and an improved user experience.

One PayString extension that is in active development is an integration with TRISA, a Travel Rule solution for VASPs that facilitates transaction identification exchange between transacting counterparties without modifying the core blockchain and cryptocurrency protocols. The goal of the TRISA is to create a separate out-of-band mechanism to augment existing blockchains and cryptocurrencies for compliance purposes.

Verifiable PayString protocol allows for secure and private out-of-band mechanism to retrieve payment addresses corresponding to PayString. Integrating PayString protocol into the TRISA flow enhances the protocol in several aspects:.

1. Determining by Sender if Receiver is a VASP: *The Risk of Sending Private Information to the Wrong Entity*
   PayString accomplishes this in two steps:
   a. Allows the Beneficiary VASP to send the signed on-ledger payment address to the Originating VASP. This proves that the identity who signed this address (i.e. the Beneficiary VASP) provided this payment address.
   b. Allows the Beneficiary VASP to send "proof of control signature" to the Originating VASP to prove the ownership of the private key corresponding to the on-ledger payment address.

   The above two proofs together tie the ownership of a private key for an on-ledger address to the identity of the Beneficiary VASP.

2. Determining by Receiver if Sender is a VASP: The problem as stated in the TRISA paper.
   "*A somewhat more complicated problem is how a receiving VASP, who gets an inbound transaction to one of their addresses, can determine if the inbound transaction is from a VASP or not. For full compliance if the inbound transaction is from a regulated VASP the receiving VASP should not make funds available to the beneficiary until the Travel Rule transaction identity information is received and recorded.*"

   This is a challenging problem when sending to a ledger address since the address is not actively provided by the Beneficiary VASP but rather is by the Originator to the Originating Institution. From the perspective of the Beneficiary VASP, determining

[12] For more details on TRISA, see the TRISA White Paper or visit trisa.io.

ownership of the address and any corresponding compliance implications is cumbersome at best.

When a payment is instead sent to a PayString, the on-ledger payment address to make the transaction is sent by the Beneficiary VASP and signed with their private key (that identifies the VASP) *after* determining if the sending side is a VASP or not.

A corollary benefit of PayString is that it precludes both false positives and false negatives, regardless of what combination of VASPs and non-VASPs are involved in a transaction. That is, a payment to a PayString will definitively determine the counterparty without ambiguity. This is impossible in a payment to an on-ledger address unless every VASP participates in the same compliance system.

3. PayString enhances the compliance screening and privacy of TRISA because the blockchain address to make the payment is only sent by the Beneficiary VASP to the Originating VASP after
    a. Beneficiary VASP and the Originating VASP have verified each other's identity and have decided to proceed with the transaction.
    b. Each side has received the required Travel Rule information about the Originator and the Beneficiary.

Below, we describe integration of PayString with TRISA for VASPs flow. The participating entities i.e. the originating and beneficiary institutions MUST acquire the following three certificates:

1. Identity Certificates for VASPs with Extended Validation
2. Transactions Signing Certificates for VASPs
3. Web PKI certificate (Non-VASP certs)

The integrated protocol flow begins at the originating VASP as a PayString client. The prerequisite is Originator issues a Payment Request that contains the Beneficiary's PayString and the transaction amount along with the other meta-data to the Originating VASP. The Originating VASP resolves the PayString URI to VASP's URL as described in the PayString discovery section.

1. The Originating VASP establishes a secure, mutually authenticated TLS 1.3 connection (non-VASP web PKI certificate) with the receiving endpoint.
2. If the TLS session is successfully established, Originating VASP (PayString client) generates the InvoiceRequest message. The body of the message MUST contain *isVASP* field set to true to indicate to the receiving endpoint that the sending endpoint is a VASP. Then it sends an HTTP POST request with path parameter /payment-setup-details to the receiving endpoint (PayString server)
3. Upon receiving this InvoiceRequest, the receiving endpoint (PayString server) parses the message body for *isVASP* field to check if the sending endpoint is a VASP. The receiving endpoint (PayString server/beneficiary VASP) generates an InvoiceResponse message encapsulated in the SignatureWrapper that includes

    a. “Travel Rule” as a compliance requirement in the list of *complianceRequirements* field.
    b. A redirect URI to redirect PayString client (Originating VASP) to TRISA server to initiate the TRISA flow.
    c. An empty *paymentInformation* field. The Beneficiary VASP MUST NOT send the payment address information yet.

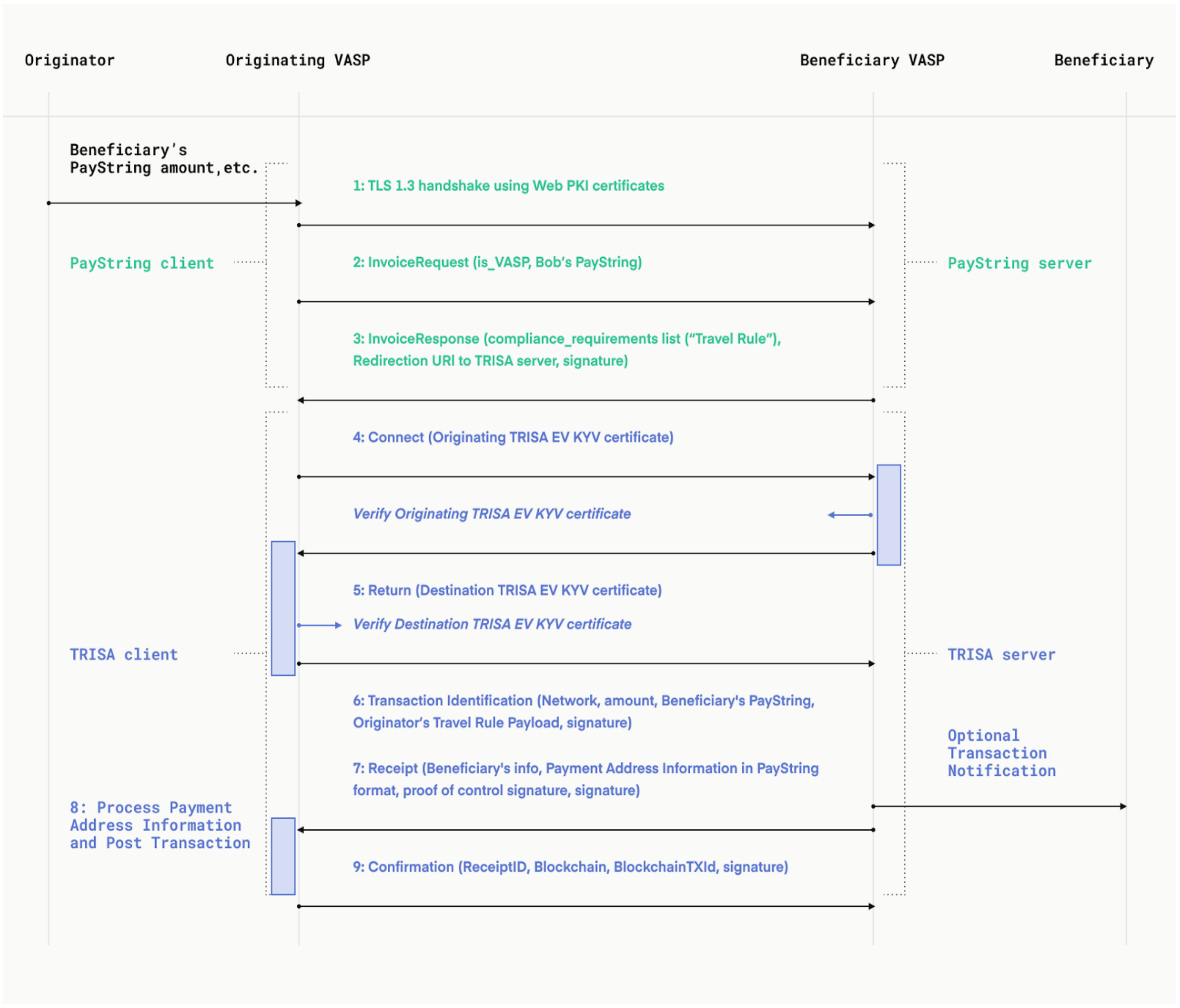
4. Upon receiving this InvoiceResponse message with a redirect URI, the PayString client (Originating VASP) forwards the request to the TRISA client. TRISA client initiates a secure, mutually authenticated TLS connection between VASPs by the Originating VASP to assure privacy of data in transit using TRISA identity certificate for VASPs.
5. Upon receiving Originating VASP’s TRISA identity certificate, the Beneficiary VASP verifies the certificate and decides if they want to proceed with the transaction with the Originating VASP. If they do, they send their TRISA identity certificate to the Originating VASP.
6. Upon receiving Beneficiary VASP’s TRISA identity certificate, the Originating VASP verifies the certificate and decides if they want to proceed with the transaction. If they do, the Originating VASP sends a transaction identification message. The transaction identification message MUST contain the *Blockchain*, *amount* and *Travel Rule information* as described in the TRISA paper [13]**.**

   Note here that there are two changes in the TRISA transaction identification message here:
    a. Originating VASP sends additional *PayString* field.
    b. Originating VASP does not send the “address” field as described in the TRISA flow. This is because this transaction identification message is a query for the Beneficiary VASP for payment address corresponding to the queried *PayString*, and *Blockchain*.
7. Beneficiary VASP sends a signed receipt to the Originating VASP. The receipt MUST contain the *Beneficiary’s information* and signed *PaymentInformation* in PayString format corresponding to the queried *PayString* and *Blockchain*.

   Note here that the response from the Beneficiary VASP includes an additional field *PaymentInformation* which contains the queried payment address corresponding to PayString.
8. Originating VASP extracts the payment address from *PaymentInformation* and posts the transaction and receives a transaction ID.
9. Originating VASP posts the transaction ID to the Beneficiary VASP.

For details on message formats for TRISA flow, refer to the TRISA paper[14]

[13] https://s32708.pcdn.co/wp-content/uploads/2020/06/Travel-Rule-Info-Sharing-ArchitectureV6.pdf
[14] https://s32708.pcdn.co/wp-content/uploads/2020/06/Travel-Rule-Info-Sharing-ArchitectureV6.pdf


Originator
Originating VASP
Beneficiary VASP
Beneficiary
Beneficiary's PayString amount,etc.
1: TLS 1.3 handshake using Web PKI certificates
PayString client
2: InvoiceRequest (is_VASP, Bob's PayString)
PayString server
3: InvoiceResponse (compliance_requirements list ("Travel Rule"), Redirection URl to TRISA server, signature)
4: Connect (Originating TRISA EV KYV certificate)
Verify Originating TRISA EV KYV certificate
5: Return (Destination TRISA EV KYV certificate)
Verify Destination TRISA EV KYV certificate
TRISA client
TRISA server
6: Transaction Identification (Network, amount, Beneficiary's PayString, Originator's Travel Rule Payload, signature)
Optional Transaction Notification
7: Receipt (Beneficiary's info, Payment Address Information in PayString format, proof of control signature, signature)
8: Process Payment Address Information and Post Transaction
9: Confirmation (ReceiptID, Blockchain, BlockchainTXId, signature)

Verifiable PayString protocol security model

The security guarantees of Basic PayString protocol apply to verifiable PayString protocol. In this section we describe additional security guarantees for verifiable PayString protocol.

While the PayString protocol operates between an originating institution and a beneficiary institution, there are actually four parties to any payment. The other two parties are the originator whose funds are being transferred and the beneficiary who the originator wishes to pay.

In the current security model, there is necessarily some existing trust between the originator and the originating institution. The originating wallet is holding the originator's funds before the payment is made. Similarly, there is necessarily some existing trust between the beneficiary and the beneficiary institution since the beneficiary has directed that the beneficiary institution receive their funds.

Verifiable PayString protocol provides stronger security guarantee: The ideal scenario that we strive for is that the originator should be able to hold the originating institution legally accountable if the originating institution *provably* mishandles their funds. Similarly, the beneficiary should be able to hold the beneficiary institution legally accountable if their funds are mishandled. However, this mechanism requires that it be possible for either wallet to establish that it acted properly and that the other wallet acted improperly.

Of course, the preferred outcome of any payment is that nothing goes wrong and both the originator and beneficiary are satisfied that the payment took place as agreed. A less desirable outcome is that the payment cannot take place for some reason and the originator still has their money and understands why the payment cannot take place.

While the protocol cannot possibly prevent the originating institution from sending the funds to the wrong address or the beneficiary wallet from receiving the funds but refusing to release them to the beneficiary, it is vital that the institutions not be able to plausibly blame each other for a failure where the originator has been debited but the beneficiary has not been credited.

Accordingly, the security model permits four acceptable outcomes:

1. The payment succeeds, the originator is debited, and the beneficiary is credited.
2. The payment fails, the originator is not debited, and the beneficiary is not credited.
3. The payment fails, the originator is debited, the beneficiary is not credited, the originator can show that the originating institution did not follow the protocol.
4. The payment fails, the originator is debited, the beneficiary is not credited, the originator can show the beneficiary that the beneficiary institution did not follow the protocol.

Again, the protocol cannot possibly prevent outcomes 3 or 4 because the originating institution can always send the money to the wrong address and the beneficiary institution can always refuse to credit the beneficiary. It is, however, critical that the originating and beneficiary wallets not need to trust each other to ensure that one of these four outcomes occurs and that they cannot point blame at each other.

Fully-malicious adversary model for originating and beneficiary institutions

We assume that the originating and beneficiary institution are fully malicious and can actively try to cheat each other. Our protocol does not require any trust relationship between the originating and beneficiary institution. In other words, the protocol enforces honest behavior by generating non-deniable third-party verifiable cryptographically signed proofs of malfeasance or thereby a lack of it.

Non-repudiation

- First our protocol ensures that the originating and beneficiary institution can not steal funds from the other side, and if they do then the other party is able to provide a verifiable cryptographically signed proof of malfeasance to any third party.

- Second, our protocol provides proof of compliance for the covered parties.

Non-deniable cryptographic proofs for originating institution:

1) Signed InvoiceResponse with the *nonce* field is a proof verifiable by a third party that the beneficiary wallet generated an invoice response corresponding to the specific invoice request message sent by the originating institution with the same nonce value.
2) The *complianceRequirements* field in the signed InvoiceResponse provides a signed confirmation of the list of compliance requirements that the beneficiary institution needs to meet.
3) Signed *paymentInformation* field in the signed InvoiceResponse is a proof verifiable by a third party that the beneficiary institution provided the corresponding payment address for a specific beneficiary.
   The *expirationTime* field in invoice response makes sure that the originating institution can not use an old response as a proof to make a future payment (This protects the beneficiary institution in case there is a change in the payment address)
4) *complianceHashes* field in signed InvoiceResponse (in case the beneficiary institution received compliance data from originating institution) is a proof for originating institution that beneficiary wallet acknowledges that originating institution has sent the required data. (This is useful in cases when the originating institution bears the burden of compliance as in case of Travel Rule.)

Non-repudiable cryptographic proofs for the beneficiary institution:

1) *data* in signed invoice request message (ComplianceData and TravelRule) is a third party verifiable cryptographic proof that binds the data sent by the originating institution corresponding to the participating originator and/or beneficiary to meet the compliance requirements mentioned by the beneficiary wallet.
2) *previousMessage* (that indicates beneficiary institution's signed compliance list) field in the signed PaymentProof is a third party verifiable proof for beneficiary institution that they successfully communicated their requirements to originating institution such that it is now up to originating institution to fulfill them. This is a proof of compliance.
3) *transactionConfirmation* and *previousMessage* fields in the signed PaymentProof is a proof for the beneficiary institution that the originating institution made the payment on the address provided by the beneficiary institution.

Fully compromisable originating and beneficiary wallet servers (hot systems): Adding another layer of security

We assume that the servers can be physically or remotely compromised by an adversary. These are the most attractive attack vectors. There is sufficient evidence that hot/always online systems are more vulnerable.

There are two signing operations that the beneficiary wallet MUST perform to generate cryptographic proofs.

a. Payment Information that maps beneficiary to payment address, and
b. Invoice response generation and secure communication channel establishment.

These two operations have very different security requirements and compromising the cryptographic keys required for these operations have different security implications.

- **High risk impersonation attack to steal funds**: If the beneficiary wallet's cryptographic keys used to cryptographically sign Payment Information are compromised, an attacker may impersonate as the beneficiary wallet and sign malicious mappings ('beneficiary → attacker controlled payment address') to send to the originating wallet. This would lead to indirection of funds by the originating wallet to the attacker controlled address. Therefore, it is extremely important to keep these keys safe offline.
- **Lower-risk impersonation attacks**: An attacker can never steal funds if only cryptographic keys used to establish secure network connection between the originating wallet and beneficiary wallet are compromised. They can, however, maliciously generate cryptographically signed invoice responses impersonating the beneficiary wallet. They can also decrypt the messages sent by the originating wallet. This may lead to privacy violation in case the messages contain sensitive data (e.g. compliance data, etc)

These differing security implications warrant a separation of generating cryptographically signed proofs and storing the cryptographic keys used to perform these two tasks separately. Some observations that inform us on how we can deal with this is that:

a) generating the cryptographic signatures on payment information need not be an online operation. This can be performed offline in a safe cold system with a separate set of keys, and

b) All other cryptographic operations need to be performed online such as signing invoice response messages.

Based on these observations, we propose to maintain two separate systems (hot and cold) and two separate sets of cryptographic keys for the two operations.
We propose that the originating wallet and beneficiary wallet SHOULD follow best practices described for key management to reduce the attack surface and be more robust. Security is a requirement and not just an option or a feature, so we strongly recommend implementing the key management solution described in the next section.

Security model for non-custodial PayString server wallets

In the current security model, non-custodial wallets do not store their customers' keys on their servers. The customers hold their private keys on their device. There is a no trust requirement between the service provided by the non-custodial wallets and the customers of this service. Since the customers hold the private keys:

- the wallets are not liable for any consequences coming from the lost, compromised or hacked private keys of the customers.
- the non-custodial wallets do not require their customers to trust their servers in case wallets servers go malicious or are compromised.

Verifiable PayString protocol preserves this trust model. For the non-custodial PayString server wallets this means that

On the receiving side of the payment (as a PayString server) non-custodial wallets have no liability on their end for providing “PaymentInformation”, i.e. the “PayString --> Payment Address” mappings for their customers that is signed with the private key of the non-custodial PayString server wallet. The PayString owners or the customers can generate this signed mapping with their own off-ledger private key locally on their app/device. The PayString client can easily verify this signature based on the trust relationship between the sender of the payment (PayString client wallet’s customer) and the receiver (non-custodial PayString server's wallet). The non-custodial PayString server wallet has no role whatsoever. This eliminates any risk of the non-custodial PayString server wallet having lost their private keys, going malicious or getting hacked, etc. because if this happens then their customers might lose funds.

Verifiable PayString protocol privacy model

All privacy guarantees in Basic PayString protocol apply to Verifiable PayString protocol and further addresses some of the privacy issues highlighted in Basic PayString protocol.

Access Control

In case, the PayString servers allow PayString owners to provide a allow/deny list of users who are authorized to access the specific information the PayString server MUST authenticate the PayString client. The additional “identity” field in the PayString client query request allows for this.

PayString protocol message types

SignatureWrapper

This message is an encapsulating wrapper for signing PayString protocol messages. It allows for the generation of cryptographically signed third-party verifiable proofs of the contents of the messages exchanged between the participating endpoints. We define *SignatureWrapper* as JSON object with the following name/value pairs.

Field name	Required/ Optional	Type	Description
messageType	required	string	The value of this field describes the type of contents delivered in the message field. E.g. “PaymentInformation”, “InvoiceRequest”
message	required	PaymentInformation || InvoiceRequest || InvoiceResponse || PaymentProof || PaymentReceipt || ComplianceData || Error	The value of this field is the contents of the verifiable PayString protocol message of the type "messageType" to be signed.
publicKeyType	required	string	The value of this field is the Public Key Infrastructure (PKI)/identity system being used to identify the signing endpoint. e.g. "X509+SHA512" means an X.509 certificate as described in RFC5280 and SHA512 hash algorithm used to hash the contents of "message" for signing. This field defaults to empty string.
publicKeyData	required	string[]	The value of this field is the PKI-system/identity data used to identify the signing endpoint who creates digital signatures over the hash of the contents of the “message”. e.g. in the case of X.509 certificates, it may contain one or more X.509 certificates as a list upto the root trust certificate. Defaults to empty.

publicKey	required	string	Contents of the public key. Defaults to empty.
signature	required	string	The value of this field is the digital signature over the hash of the contents of the “message” using the private key corresponding to the public key in “publicKey”. This is a proof that the “message” was signed by the corresponding private key holder.

If this wrapper is present, then it MUST include *all* the required fields.

PaymentInformation

This message MUST be encapsulated in the SignatureWrapper in case of Verifiable PayString protocol and its extensions.
This message MUST be signed using the keys as described in the Key Management section.

Field name	Required/ Optional	Type	Description
addresses	required	Address[]	The value of this field is an array of one or more JSON objects of type addresses
proofOfControlSignature	optional	ProofOfControlSignature	The value of this field is a JSON object as described in ProofOfControlSignature. This is the digital signature proving ownership of the on-ledger address
identity	optional	string	This field may specify any additional identity information about the PayString owner or PayString server. See here.
payId	optional	string	The value of this field is the PayID URI in the client request that identifies the payment address information

memo	optional	string	Specifies additional metadata corresponding to a payment

PayString is optional. If the PaymentInformation message is encapsulated in the SignatureWrapper, *PayString* field MUST be set to the receiver's PayString.

addresses

This is a required field in the PaymentInformation message.

Field name	**Required/ Optional**	**Type**	**Description**
paymentNetwork	required	string	The value of this field is the payment-network as specified in the client request's "Accept" header (e.g. XRPL)
environment	optional	string	The value of environment as specified in the client request's "Accept" header (e.g. TESTNET)
addressDetailsType	required	string	The value of this field is the string "CryptoAddressDetails" or "ACHAddressDetails"
addressDetails	required	CryptoAddressDetails || ACHAddressDetails	The value of this field is the address information necessary to send payment on a specific network as described in addressDetails

addressDetails

This is a field in the PaymentInformation message. addressDetails for each specific ledger MUST be registered at PayString.org.

Address Type	**Field name**	**Required /Optional**	**Type**	**Description**
CryptoAddressDetails	address	required	string	On-ledger address
	tag	optional	string	Tagging mechanism used by

				some cryptocurrencies to distinguish accounts contained within a singular address. E.g XRP
ACHAddressDetails	accountNumber	required	string	ACH account number
	routingNumber	required	string	ACH routing number

ProofOfControlSignature

This is an optional field in the PaymentInformation message.

Field name	Required/ Optional	Type	Description
publicKey	required	string	on-ledger public key of the PayID server
payID	required	string	PayID of the receiver.
hashAlgorithm	required	string	The value of this field is the hash algorithm used to hash the entire contents of the "ProofOfControlSignature" message. E.g. "SHA512"
signature	required	string	The value of this field is the digital signature over the hash of the entire contents of the "ProofOfControlSignature" message using the private key corresponding to the public key in "publicKey". This is a proof that the owner of the private key corresponding to the public key in the "publicKey" used to sign this message is the owner of the on-ledger public key in "publicKey".

InvoiceRequest

This message is sent by the Sending Endpoint. This message contains the required information for the Receiving Endpoint to return the payment setup details. This message is encapsulated in the SignatureWrapper and MUST be signed using the short-term keys as described in the Key Management section.

Field name	Required/ Optional	Type	Description

identity	optional	string	TBD
fullLegalName	optional	string	Full legal name of the Sending Endpoint
postalAddress	optional	string	Principal place of Business Address of the Sending Endpoint
isVASP	optional	boolean	Indicates if the Endpoint is a VASP.
transactionAmount	optional	integer	Amount of intended payment
scale	optional	integer	Orders of magnitude necessary to express one regular unit of the currency e.g. a scale of 3 requires an amount of 100 to equal 1 US dollar
memo	optional	string	Specifies additional metadata

InvoiceResponse

This message is sent by the Receiving Endpoint in response to the InvoiceRequest message sent by the Sending Endpoint. This message is encapsulated in the SignatureWrapper and MUST be signed using the short-term keys as described in the Key Management section.

Field name	Required /Optional	Type	Description
id	required	string	The value of this field is the UUID as described in RFC 4122
fullLegalName	optional	string	Full legal name of the Receiving Endpoint
postalAddress	optional	string	Principal place of Business Address of the Receiving Endpoint
isVASP	optional	boolean	Indicates if the Endpoint is a VASP. Defaults to false
redirectURI	optional	string	A redirect URI for PayString client
transactionAmount	optional	integer	Amount of intended payment
scale	optional	integer	Orders of magnitude necessary to express one regular unit of the currency e.g. a scale of 3 requires an amount of 100 to equal 1 US dollar

expirationTime	required	integer (milliseconds from epoch)	This message is considered void and payments MUST NOT be made on the specified address in the *paymentInformation* field past the specified timestamp
paymentInformation	required	PaymentInformation	Contains details as to how a payment can be made to the Beneficiary. Defaults to empty.
complianceRequirements	required	string[]	List of the regulatory requirements that the Beneficiary must satisfy during the proposed transaction. Allows the client to send relevant compliance data corresponding to the data in this field. e.g TravelRule data in case of Travel rule compliance requirement. Defaults to empty list
previousMessage	required	string	This is the previous InvoiceRequest message received from the Sending Endpoint.
memo	optional	string	Specifies additional metadata to a payment

ComplianceData

This is the upgraded invoice request message sent by the Originating Institution to transmit the required compliance data corresponding to the requirements mentioned in the *complianceRequirements* field of the InvoiceResponse message by the Beneficiary Institution. This message is encapsulated inside the SignatureWrapper.

Field name	**Required/ Optional**	**Type**	**Description**
previousMessage	required	InvoiceResponse	This is the previous InvoiceResponse message corresponding to which this upgraded invoice request (ComplianceData) message is generated
type	required	string	Type of the compilanceData field. e.g. “TraveRule”
data	required	TravelRule	Travel Rule payload
memo	optional	string	Optional data field

TravelRule

This is an example of the kind of data in the *data* field of ComplianceData message. The following payload conforms to the industry standard for messaging ISO 20022 PACS.008 Following is the message format for Travel Rule payload.

Field name	Required /Optional	ISO Data Type	Description
originator	required	PartyIdentification135	Field containing some data about the Originator
userLegalName	required	Max140Text	Legal name of the Originator
userPhysicalAddress	required	PostalAddress24	Details about the physical address of the Originator
institutionName	required	BranchAndFinancialInstitutionIdentification6	Contains Legal title of the Originating Institution
accountId	required	CashAccount38	Contains Account ID within Originating Institution of the Originator
amount	required	ActiveorHistoricCurrencyAndAmount	Amount of transaction
timestamp	required	ISODate	Date of transaction
beneficiary	optional	PartyIdentification135	Field containing known data about the Beneficiary
userLegalName	optional	Max140Text	Legal name of Beneficiary
userPhysicalAddress	optional	PostalAddress24	Contains Physical address of the Beneficiary
identification	optional	CashAccount38	Contains specific identifier of the Beneficiary
institutionName	required	BranchAndFinancialInstitutionIdentification6	Contains legal title of the Beneficiary Institution
accountId	optional	CashAccount38	Contains Account ID within the Originating Institution of the

			Beneficiary

PaymentProof

This message is optionally sent by the Sending Endpoint as a proof of payment on the payment address sent in the InvoiceResponse message by the Beneficiary Institution.
This message is encapsulated in the SignatureWrapper.

Field name	Required /Optional	Type	Description
previousMessage	required	InvoiceResponse	The InvoiceResponse message that this PaymentProof is fulfilling
transactionConfirmation	required	string	Evidence of the submitted transaction on the payment address provided in the InvoiceResponse message. e.g. for cryptocurrencies, this would be the transaction output and for ACH transactions, this would be the trace number.
memo	optional	string	Specifies additional metadata to a payment

PaymentReceipt

This message is optionally sent by the Beneficiary Institution to the Originating Institution as an receipt of payment. This message is encapsulated in the SignatureWrapper and MUST be signed using the short-term keys as described in the Key Management section.

Field name	Required/ Optional	Type	Description
previousMessage	required	PaymentProof	PaymentProof message that this receipt is acknowledging
organizationName	optional	string	Name of the Receiving Endpoint
paidAmount	optional	string	Amount paid/transferred by the originating wallet to the beneficiary wallet
remainingAmount	optional	string	Any remaining amount
transactionStatus	optional	string	The status of transaction. e.g. “complete”, “pending”, “failed”

scale	optional	integer	Orders of magnitude necessary to express one regular unit of the currency e.g. a scale of 3 requires an amount of 100 to equal 1 US dollar
currency	optional	string	Currency in which amount is paid
timestamp	required	integer (milliseconds)	Number of seconds since the Unix epoch to indicate the time when this paymentReceipt is generated
memo	optional	string	Specifies additional metadata to a payment

Error

This message is used to communicate the PayString protocol level errors. We follow the RFC 7807 with the HTTP header Content-type := application/problem+json and the following fields in the Error message body.

Field name	**Required/ Optional**	**Type**	**Description**
type	required	string	As described in RFC 7807. For further details on the URIs see PayString protocol status communication.
title	required	string	As described in RFC 7807. For further information on Title of the error see PayString protocol status communication.
statusMessage	optional	string	As described in RFC 7807. For further information about the status of the PayString protocol see PayString protocol status communication.
statusCode	required	integer	As described in RFC 7807. For further information on relevant HTTP status code for the error generated see PayString protocol status communication.
previousMessage	required	string	This is the message corresponding to which this Error message is generated.

PayString protocol status communication

The following status codes MUST be communicated in the Error message specific to the corresponding error in the PayString protocol message.

Type	Title	status_message	HTTP status_code
"https://PayString.org/invoice/address-not-found"	"Payment address Not Found"	"Payment address does not exist in the database"	404
"https://PayString.org/invoice/PayString-not-found"	"PayString Not Found"	"PayString does not exist in the database"	404
"https://PayString.org/invoice/certificate-required"	"Certificate Required"	"A certificate is required to verify the signature"	400
"https://PayString.org/invoice/certificate-expired"	"Certificate Expired"	"The certificate sent by the sending endpoint has expired"	400
"https://PayString.org/invoice/certificate-revoked"	"Certificate Revoked"	"The certificate sent by the sending endpoint has been revoked"	400
"https://PayString.org/invoice/certificate-invalid"	"Certificate Invalid"	"The certificate sent by the sending endpoint is invalid"	400
"https://PayString.org/invoice/signature-invalid"	"Signature Invalid"	"Could not verify the signature using the provided *publicKey* "	400
"https://PayString.org/invoice/invalid-previous-message"	"Invalid/Missing previousMessage"	"*previousMessage* is invalid or missing"	400
"https://PayString.org/invoice/invalid-compliance-data"	Invalid/missing/incompletedata	"*data* is invalid, missing or incomplete"	400
"https://PayString.org/invoice/invoice-expired"	"Invoice Expired"	"The current system time is past the expiration time on invoice response"	400
"https://PayString.org/	"Missing/Mismatch	"*nonce* is invalid or missing"	400

invoice/invalid-nonce"	nonce”		
"https://PayString.org/invoice/legal-reasons"	“Legal Reasons”	“Endpoint sending the message does not want to proceed with the transaction due to legal reasons”	451
"https://PayString.org/invoice/invalid-proof-of-control-signature"	“Missing/Invalid proof of control signature”	“*proofOfControlSignature* is invalid or missing”	400
"https://PayString.org/invoice/invalid-payment-receipt"	“Invalid payment receipt”	“The payment receipt is invalid”	400
"https://PayString.org/invoice/invalid-payment-proof"	“Invalid payment proof ”	“The payment proof is invalid”	400

Transport layer communication errors

Transport-layer communication errors must be communicated to the party that initiated the communication via the communication layer's existing error messaging facilities. In the case of HTTP-over-TLS, this should be done through standard HTTP Status Code messaging (RFC 7231)

HTTP request and response headers

PayString protocol defines semantics around the following request and response headers. Additional headers MAY be defined, but have no unique semantics defined in the PayString protocol.

Common headers

The following headers are common between the PayString requests and responses.

1. Header Content-Type

PayString requests and responses with a JSON message body MUST have a “Content-Type” header value of `application-json`.

2. Header Content-Length

As defined in RFC 7230, a request or response SHOULD include a “Content-Length” header when the message's length can be determined prior to being transferred. PayString protocol does not add any additional requirements over HTTP for writing Content-Length.

3. Header PayString-version

Versioning enables clients and servers to evolve independently. PayString protocol defines semantics for protocol versioning. PayString requests and responses are versioned according to the PayString-version header.
PayString clients include the PayString-version header in order to specify the maximum acceptable response version. PayString servers respond with the maximum supported version that is less than or equal to the requested `major`

PayString-version: major.minor

The PayString client MUST include the PayString version request header field to specify the version of the PayString protocol used to generate the request.
If present on a request, the PayString server MUST interpret the request according to the rules defined in the specified version of the PayString protocol or fail the request with an appropriate error response code.
If not specified in a request, the PayString server MUST fail the request with an appropriate error code.

Request headers

In addition to common headers, the PayString client MUST specify the following request header.

1. Header Accept

The PayString client's HTTP "GET" and "POST" requests MUST specify the "Accept" request header field with at least one of the registered media types defined in this section. The purpose of this header is to indicate what type of content can be understood in the response. It specifies the "payment-network" and "environment" of the payment address and its representation format for which the PayString client wants to receive information. The representation format is always JSON.

PayString server MUST reject formats that specify unknown or unsupported format parameters.

Accept: application/(payment-network)-(environment)+json

- payment-network is the short string of letters representing the currency. See here for a common list.
- environment should be "mainnet" for live currencies or the appropriate testnet name for test currencies.

Initially, we propose standards with the headers specific to XRP, ACH and ILP payment-networks. We also propose one header that may return ALL addresses across all payment-networks and environments. Other payment networks will be able to establish standard media types for their networks over time at PayString.org.

ALL

Accept-header	Description
application/PayString+json	May return addresses for all payment-networks and environments

XRP

Accept-header	Description
application/xrpl-mainnet+json	Returns XRPL mainnet xAddresses or classic addresses
application/xrpl-testnet+json	Returns XRPL testnet xAddresses or classic addresses
application/xrpl-devnet+json	Returns XRPL devnet xAddresses or classic addresses

ACH

Accept-header	Description
application/ach+json	Returns account and routing number

ILP

Accept-header	Description
application/interledger-mainnet+json	Returns mainnet payment pointer to initiate SPSP request
application/interledger-testnet+json	Returns testnet payment pointer to initiate SPSP request

PayString servers MUST reject formats that specify unknown or unsupported format parameters.

Response headers

In addition to the Common Headers, the PayString server MUST specify the following response header.

1. Header Cache-Control

PayString server MUST include the “Cache-Control” header with the max-age limit of 0. The intermediate hops or PayString client MUST not cache the responses.

Protocol extensibility

1. Payload Extensibility

PayString protocol supports extensibility in the payload, according to the specific format. Regardless of the format, additional content MUST NOT be present if it needs to be understood by the receiver in order to correctly interpret the payload according to the specified PayString-Version header. Thus, clients MUST be prepared to handle or safely ignore any content not specifically defined in the version of the payload specified by the PayString-version header.

2. Header Field Extensibility

PayString protocol defines semantics around certain HTTP request and response headers. Services that support a version of PayString protocol conform to the processing requirements for the headers defined by this specification for that version.

Individual services MAY define custom headers. These headers MUST NOT begin with PayString. Custom headers SHOULD be optional when making requests to the service. A service MUST NOT require the PayString client to understand custom headers to accurately interpret the response.

3. Format Extensibility

A PayString service MUST support JSON format as described above and MAY support additional formats response bodies.

Acknowledgements

The PayString protocol is part of the Open Payments Coalition, a group of leading payments companies across traditional finance, FinTech, and blockchain.

Significant support for and development of PayString has been provided by Ripple and RippleX. We would like to especially recognize the following individuals for their contributions: David Fuelling, Dino Rodriguez, Doug Purdy, Ethan Beard, Hans Bergren, Keefer Taylor, Mayur Bhandary, Stephen Gu, Ted Kalaw, Tyler Longwell, and Tyler Storm, Will Liu.

Appendix A. PayString protocol message verification

Verifying InvoiceRequest message

Upon receiving an InvoiceRequest message from the Sending Endpoint, the Receiving Endpoint performs the following verification steps:

a) Verifies the *publicKey* using the *publicKeyData* and verifies the signature on the message body

If the verification fails, the Beneficiary Institution generates the relevant signed Error message For details on error codes refer to the section PayString protocol status communication

Verifying InvoiceResponse message

Upon receiving an InvoiceResponse message from the Receiving Endpoint, the Sending Endpoint performs the following verification steps:

a) Verifies the *publicKey* using the *publicKeyData* and verifies the signature on the InvoiceResponse message body
b) If this is an upgraded InvoiceResponse message, verifies the *publicKey* using the *publicKeyData* and verifies the signature on *paymentInformation* field (Recall that this field is signed using a different short-term key)
c) Checks if the time in the *expirationTime* field is less than the current system time[15] of the Originating Institution

All the verification steps MUST pass. The Sending Endpoint proceeds to the next step only if the previous step passes, otherwise it generates the relevant Error message. For details on error codes refer to the PayString protocol status communication section.

Verifying ComplianceData message

Upon receiving the upgraded invoice request message i.e. the ComplianceData message from the Originating Institution, the Receiving Institution performs the following verification steps:

a. Verifies the *publicKey* using the *publicKeyData* and verifies the signature on the ComplianceData message body.
b. Verifies if the message in the *previousMessage* field matches the InvoiceResponse message previously sent by the Beneficiary Institution.

All the verification steps MUST pass. The Beneficiary Institution proceeds to the next step only if the previous step passes, otherwise it generates the relevant Error message. For details on error codes refer to the PayString protocol status communication section.

[15] originating wallets can choose the source of truth for the current time. We assume that most systems use their system time obtained from network timing protocols such as Network Time Protocol (NTP) and likes. For details on security issues related to using NTP, etc. refer Attacking the Network Time Protocol

Verifying PaymentProof message

Upon receiving the PaymentProof message from the Sending Endpoint, the Receiving Endpoint performs the following verification steps:

a. Verifies the *publicKey* using the *publicKeyData* and verifies the signature on the message body.
b. Verifies if the message in the *previousMessage* field matches the InvoiceResponse message previously sent by the Receiving Endpoint.

Verifying PaymentReceipt message

Upon receiving the PaymentReceipt message from the Receiving Endpoint, the Receiving Endpoint performs the following verification steps:

a. Verifies the *publicKey* using the *publicKeyData* and verifies the signature on the message body.
b. Verifies if the message in the *previousMessage* field matches the InvoiceResponse message previously sent by the Receiving Endpoint.

Verifying Error message

The Endpoint that receives the Error message performs the following verification steps:

a. Verifies if the message in the *previousMessage* field matches the message previously sent by them.

If the verification step fails, it drops the Error message.

Session establishment

We recommend RFC 8446 for TLS 1.3 session establishment. Each side MUST use ECDHE (Diffie-Hellman over elliptic curve) key exchange mode. Each side should use the short-term key-pair as described in the Key Management section for TLS handshake.

Appendix B. Key management

In this section we describe the key hierarchy for signing PaymentInformation, InvoiceRequest, InvoiceResponse, ComplianceData, PaymentProof, and PaymentReceipt messages. This is a one-time key-generation set-up performed by each institution before the protocol is run. Note that each institution can be the Originating and Beneficiary Institution in different instantiations of the protocol. We lay down the requirements for different roles.

Long-term elliptic-curve(EC) key-pair generation

Each institution generates a long-term Elliptic Curve (EC) public/private key pair MPK/MSK and obtains a corresponding valid X.509 certificate. We call the public and private keys corresponding to Originating and Beneficiary Institutions as MPK_o/MSK_o and MPK_b/MSK_b.

Parameter generation

Each institution chooses a set of domain parameters that include a base field prime p, an elliptic curve E/F_p, and a base point G of order n on E. An elliptic-curve key pair (d, Q) consists of a private key d, which is a randomly selected non-zero integer modulo the group order n, and a public key Q = dG, the d-multiple of the base point G. Thus the point Q is a randomly selected point in the group generated by G.

Short-term EC key-pair generation (Receiving Endpoint/Beneficiary Institution)

1) On a cold system, generate a short-term EC public/private key-pair. We call it $SKP1_b$, where public key = $PK1_b$ and private key = $SK1_b$. Generate an X.509 certificate for $PK1_b$ signed with MSK_b. This key-pair is needed for secure TLS session establishment and for signing InvoiceResponse and PaymentReceipt messages
2) On a cold system, generate another short-term EC public/private key-pair. We call it $SKP2_b$, where public key = $PK2_b$ and private key = $SK2_b$. Generate an X.509 certificate for $PK2_b$ signed with MSK_b. $SK2_b$ is used to sign PaymentInformation mappings.
3) On a hot system, save the following:
    a) database of signed PaymentInformation mappings in PayString server,
    b) X.509 certificate for $PK1_b$,
    c) X.509 certificate for $PK2_b$,
    d) $SK1_b$,
    e) X.509 certificate of MPK_b
4) Store the MSK_b offline in a safe vault.

Short-term EC key-pair generation (Sending Endpoint/Originating Institution)

1) On a cold system, generate short-term EC public/private key-pair. We call it $SKP1_o$, where public key = $PK1_o$ and private key = $SK1_o$. Generate an X.509 signed certificate for $PK1_o$ with MSK_o. This key-pair is needed for secure TLS session establishment and for signing InvoiceRequest, ComplianceData, and PaymentProof messages.
2) On a hot system, save the following:
    a) X.509 signed certificate of $PK1_o$,
    b) $SK1_o$,
    c) X.509 certificate of MPK_o
3) Store the MSK_o offline in a safe vault.

Key Rotation: Short-term keys SHOULD be rotated periodically as a general good key management practice. Public keys required for signature verification must be included in the corresponding signed messages every time the protocol is run. This provides flexibility to periodically rotate the keys without worrying about key-updates and also makes each message self-contained. Our protocol does not require caching of keys or payment addresses, so key-update is not a concern.

Cryptography choices

We recommend using elliptic-curve cryptography because:

a) ECC provides greater security for a given key-size
b) Better performance: The smaller key size also makes possible much more compact implementations for a given level of security. This means faster cryptographic operations. ECC has very fast key generation and signature algorithms.
c) There are good protocols for authenticated key-exchange.
d) Efficient implementations: There are extremely efficient, compact hardware implementations available for ECC exponentiation operations, offering potential reductions in implementation footprint even beyond those due to the smaller key length alone.

Appendix C. Additional security considerations

Warning on X.509 certificates

There are various types of SSL certificates available. We warn the implementations to use certificates that require rigorous validation process for issuance. This is important to leverage the security guarantees provided by the “key separation” security model above. Below we highlight the security scenarios for different kinds of web certificates in case an attacker is able to compromise the online server of either endpoint.

1. Domain Validated (DV) certificates may not provide the same level of security in case the online server of the endpoint is compromised. This is because the validation process to obtain a DV certificate requires the lowest level of authentication to prove domain ownership. If an attacker can break into the server, they may be able to impersonate as domain owner and pass the most commonly deployed validation checks to prove domain ownership by CAs. An attacker can thus easily get a new DV certificate issued for the attacked domain with the new MPK/MSK pair, which they can then use to generate new short-term keys and certificates.
2. Organization Validation (OV) certificate may provide better security in our security model. The validation process to obtain an OV certificate requires vetting of the individual and/or organization by CAs in addition to validating the domain ownership. Thus, obtaining an OV certificate for a domain is relatively harder even if an attacker can get hold of all the resources on the attacked server.
3. Extended-validation (EV) certificates are considered to be most secure as they require even more rigorous validation checks. The validation process to obtain an EV certificate requires much more rigorous identity checks on individuals and organizations in addition to domain ownership validation. Thus, obtaining an EV certificate for a domain is relatively harder because feven if an attacker can get hold of all the resources on the attacked server.

We, therefore, strongly recommend

1. The Endpoints to use high-assurance EV or OV certificates for their long-term keys.
2. The Endpoints validating the certificates to be wary while accepting low-assurance DV certificates.