PayID Protocol

March 24, 2020
Aanchal Malhotra, Austin King, David Schwartz, Michael Zochowski
contactxpring@ripple.com

Abstract

Complicated and hard-to-remember crypto 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 PayID-a standard for human-readable payment addresses that can be resolved to any underlying payment rail, whether crypto or traditional. PayID is designed to be general, flexible, and extensible. We present two specific extensions that layer functionality on top of ledger transactions. First, functionality that cryptographically correlates on-ledger transactions to third party verifiable proofs of invoice and receipt. Second, PayID is extended to provide a messaging standard for regulated ("covered") institutions to exchange information to fulfil their compliance requirements. In its various forms, PayID 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

Crypto wallet and payment 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 payment networks in a user-friendly manner, we believe it is imperative to:

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

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

PayID 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 streamlined solutions for a variety of payments and identity use cases across both crypto and traditional finance, which we present two of in this paper.

First, we extend PayID to provide secure invoicing and receipts, which generates cryptographic records of the entire lifecycle of a PayID transaction and allows a recipient to better track transactions based on the sender. Together with PayID's substantial improvements in user experience, cryptographic receipts enable a streamlined flow for point-of-sale, ecommerce, and other merchant transactions that are burdensome in the current crypto paradigm.

Second, we show an extension to PayID that provides a simple and secure solution to meet the current and potential future compliance and legal requirements of cryptocurrency service providers. In particular, we present a 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 crypto payments flows, where it is challenging to determine both when Travel Rule applies and how to securely exchange sensitive customer information when it does apply, but PayID presents a straightforward and elegant solution to this pressing problem.

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

Solution Overview
PayID - A protocol for human-readable payment addresses

Basic PayID

PayID is a simple application-layer request/response protocol that uses HTTP request methods, error codes, and headers to retrieve network-specific payment addresses corresponding to human-readable addresses (PayIDs). These PayIDs are analogous to email addresses — they 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.

PayID is built on top of an existing standard called Payment Pointers[1], which provides a scheme for converting human-readable addresses to URLs. HTTP requests to these endpoints return details on the user's address for a particular payment rail. The desired network for a particular transaction is specified in the accept-type header of the request, and the response provides the required addressing and metadata to complete the transaction on that network, thus allowing PayID to be used for any payment rail. PayID's web infrastructure — rather than a blockchain-based solution—makes it universally usable across both crypto and traditional finance, compatible with other namespace solutions, and universally appealing.

Implemented alone, PayID provides for interoperability of namespaces across payment rails. It is intentionally designed to be low complexity, lightweight, and built using existing tools. At the same time, it is designed to be flexible, composable, and extensible for a variety of more advanced applications.

Verifiable PayID with Invoices and Receipts

On top of simply retrieving payment addresses, PayID allows for the generation of invoice requests, invoice reponses, and receipts of payments. This enables sophisticated payment routing based on the sender, which is useful in point-of-sale and other transaction contexts.

It also provides non-repudiable cryptographic proof of a PayID transaction life cycle, which allows for improved record keeping. These proofs, which entail the invoices and receipts that are signed by the server key, eliminate all counterparty trust by ensuring that neither party can falsely claim that the payment was sent to the wrong address or the wrong amount was sent.

Combined with PayID's improved user experience, cryptographic invoicing enables user-friendly, secure transactions that are suitable for all commercial transaction types.

Verifiable PayID with Compliance (Travel Rule) extensions

The invoice functionality of PayID allows us to easily extend cryptographically verifiable receipts 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 crypto industry and more of such regulations are anticipated in the future. As a result, there is a pressing need to come up with 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.

[1] https://paymentpointers.org/

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 basic PayID 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 crypto 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")[2] 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 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 PayID 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.[3]

Note: In this paper, we describe PayID 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.

Protocol design principles

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

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

[3] 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.

1. Simplicity

Rather than reinventing the wheel, PayID 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 PayID 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 PayID protocol as a fundamentally neutral protocol. PayID is capable of returning a user's address information for any network that they (or their service) support. This makes PayID a network and currency agnostic protocol, capable of sending payments in BTC, XRP, ERC-20 tokens, Lightning, ILP, or even fiat networks like ACH.

3. No Central Authority

PayID 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. Decentralized & Peer-to-Peer

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

5. Extensibility and Improved User Experience

PayID 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. PayID is designed to be an upgradeable and open standard, with a robust roadmap of future improvements and additional features.

6. Service Sovereignty

Each operator maintains full control of its PayID 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. PayID 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.

7. Composable with Existing Standards and Namespaces

By design, PayID is highly abstract and generalized. As a result, PayID can easily wrap existing standards or namespaces, such as ENS, Unstoppable Domains, or service-specific identifiers like Cashtags or Coinbase Usernames, and provide each of them far greater reach and user value.

Protocol Flow

The PayID protocol provides a set of HTTP requests and responses to facilitate structured communication between the sender and the receiver.

Building Blocks: HTTP Methods and Retrieval

GET is used to retrieve the payment address at a given endpoint.

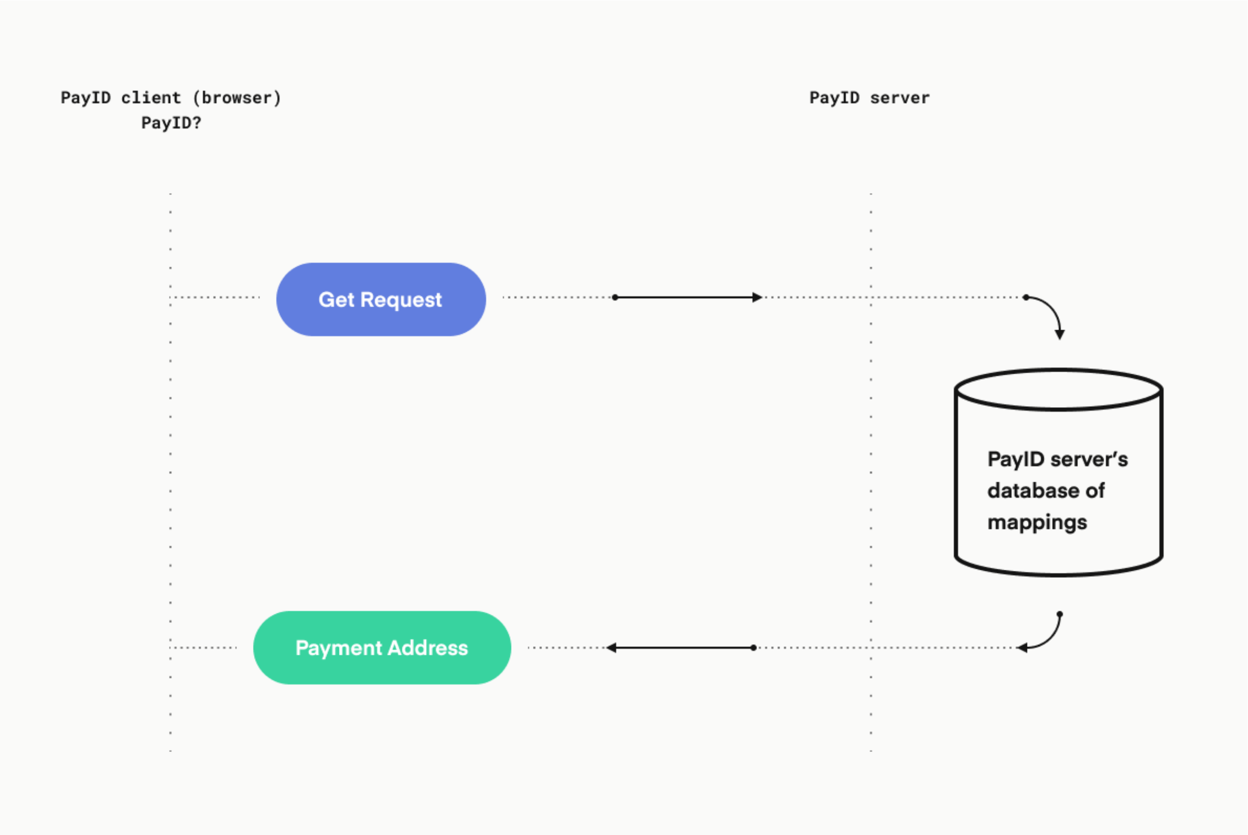
POST is used to

1. retrieve a cryptographically signed ‘beneficiary → payment address’ mapping in the invoice response,
2. send compliance data
3. send the payment receipt as a proof of payment on the provided address

This protocol only covers the GET and POST requests and a few error codes from HTTP. However, the participating entities should be prepared to handle other error codes supported by HTTP.

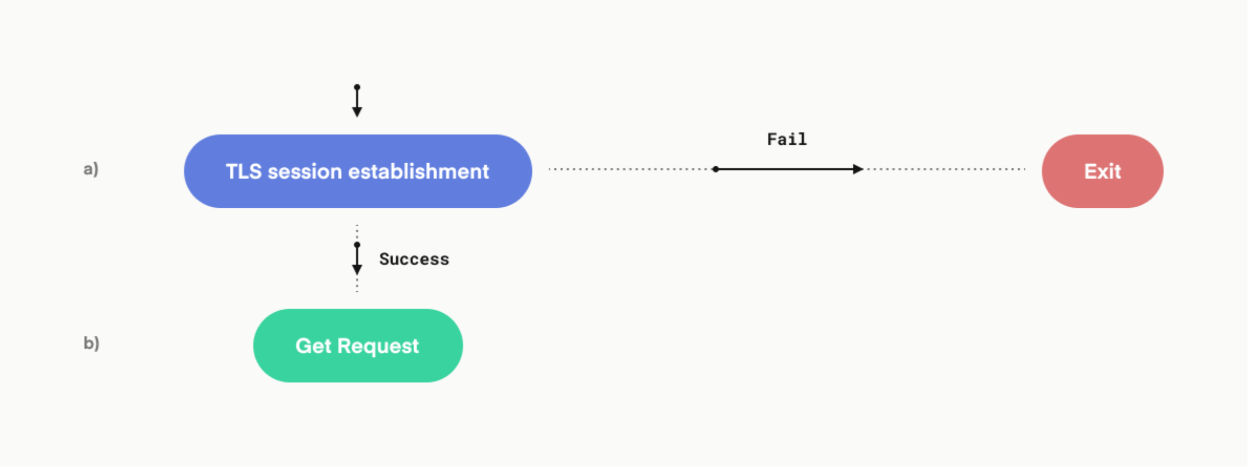
I. Basic PayID protocol flow: Securely retrieve payment rail addresses corresponding to a PayID

Basic PayID protocol is useful for PayID clients (such as developers) to easily query PayID servers for payment addresses corresponding to human-readable PayIDs.



The following steps describe how the PayID client retrieves the payment address corresponding to a PayID.[4]

1) **GET request**: Processing steps by PayID client (typically a browser)

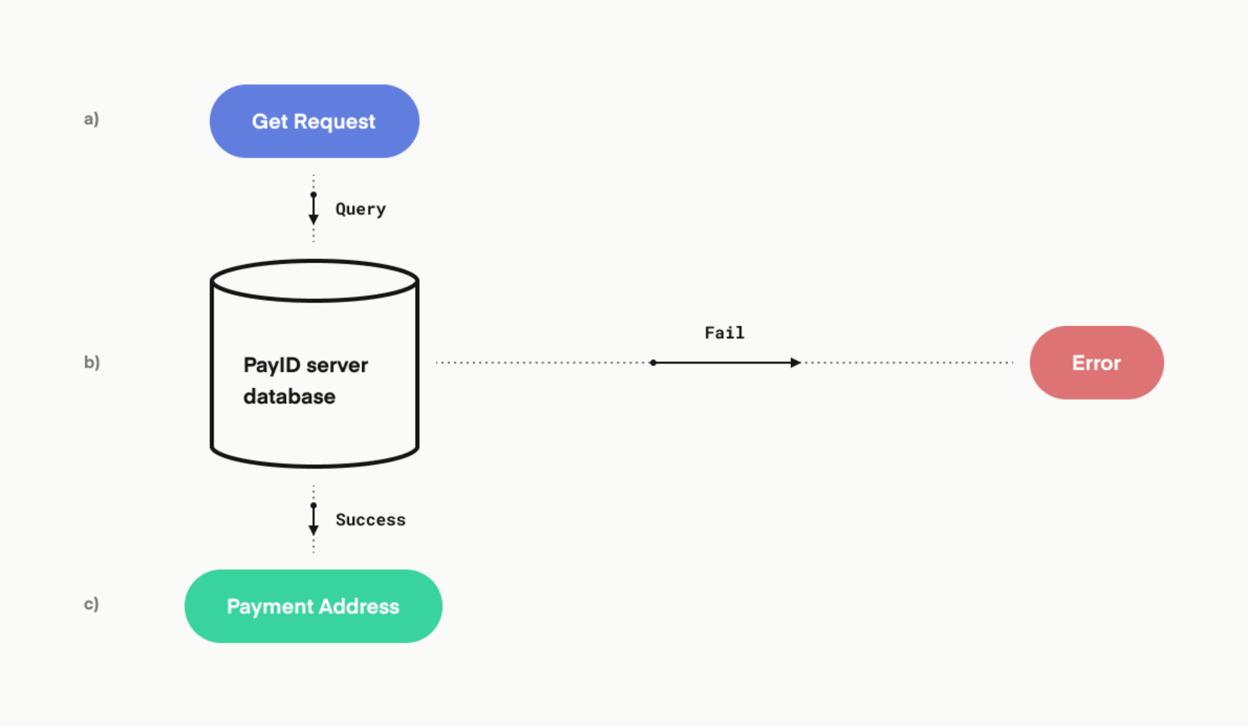


[4] How a client obtains PayID is out-of-scope of this protocol. Instead of a PayID, the client can also use its corresponding URL directly, as described in the Payment Pointer standards.

a) Establish a secure TLS session with the PayID server as described in the session establishment section. The URL of the PayID server is derived from the PayID as described in paymentpointers.org/syntax-resolution/

b) If the TLS session is successfully established, send an HTTP GET request over the established secure channel. PayID client specifies the payment network they support via the HTTP Accept-type header. For details on HTTP Accept-type headers refer to the GET Headers and Payment address response section; otherwise exit.

2) **Payment address response**: Processing steps by PayID server

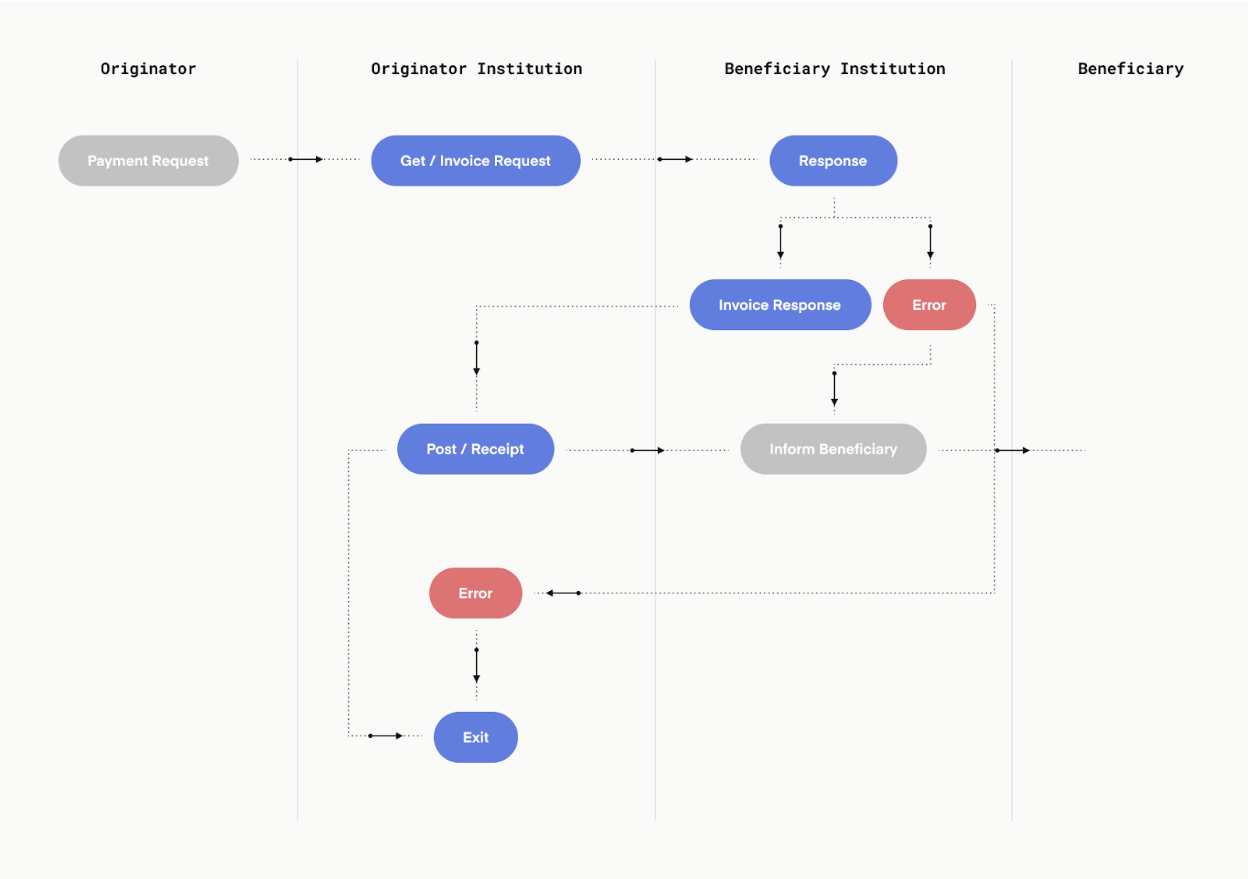


a) Receive the GET request from PayID client.
b) Query its database for the payment address corresponding to PayID in the format described in PaymentInformation. If the payment information exists in the database, the PayID server generates the HTTP payment address response; otherwise it generates an error message. For details on payment address response refer to the GET Headers and Payment address response section. For details on error codes refer to the PayID protocol status communication section**.**
c) Send payment address response or error message to the PayID client.

II. Verifiable PayID protocol flow: extends basic PayID to include third party verifiable cryptographically signed invoices and receipts.

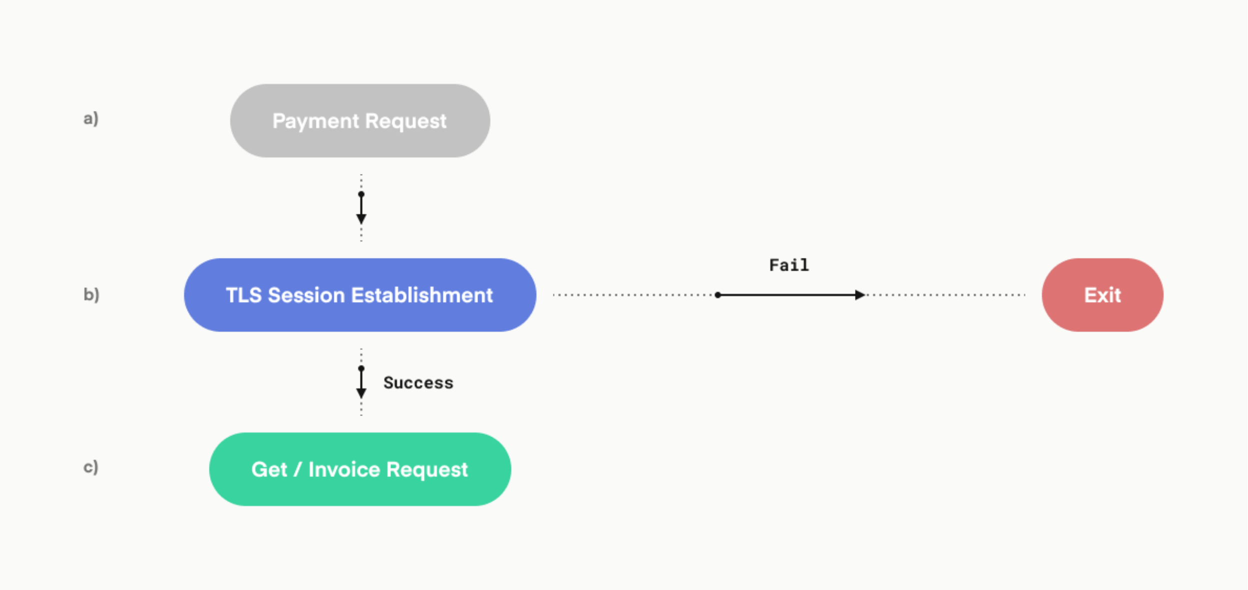
Terminology:

Originating Institution: the service sending a transaction. This may be a user run software or a Virtual Asset Service Provider (VASP)/Financial Institution (FI)
Beneficiary Institution: the service receiving a transaction. This may be a user run software or a VASP/FI
Covered Institution: entity that must comply with some set of regulatory requirements
Originator: Customer of originating institution
Beneficiary: Customer of beneficiary institution



In case, the originating institution wants to make a payment with a signed proof of Payment information from the beneficiary institution, they would generate a request for an invoice from the beneficiary institution.

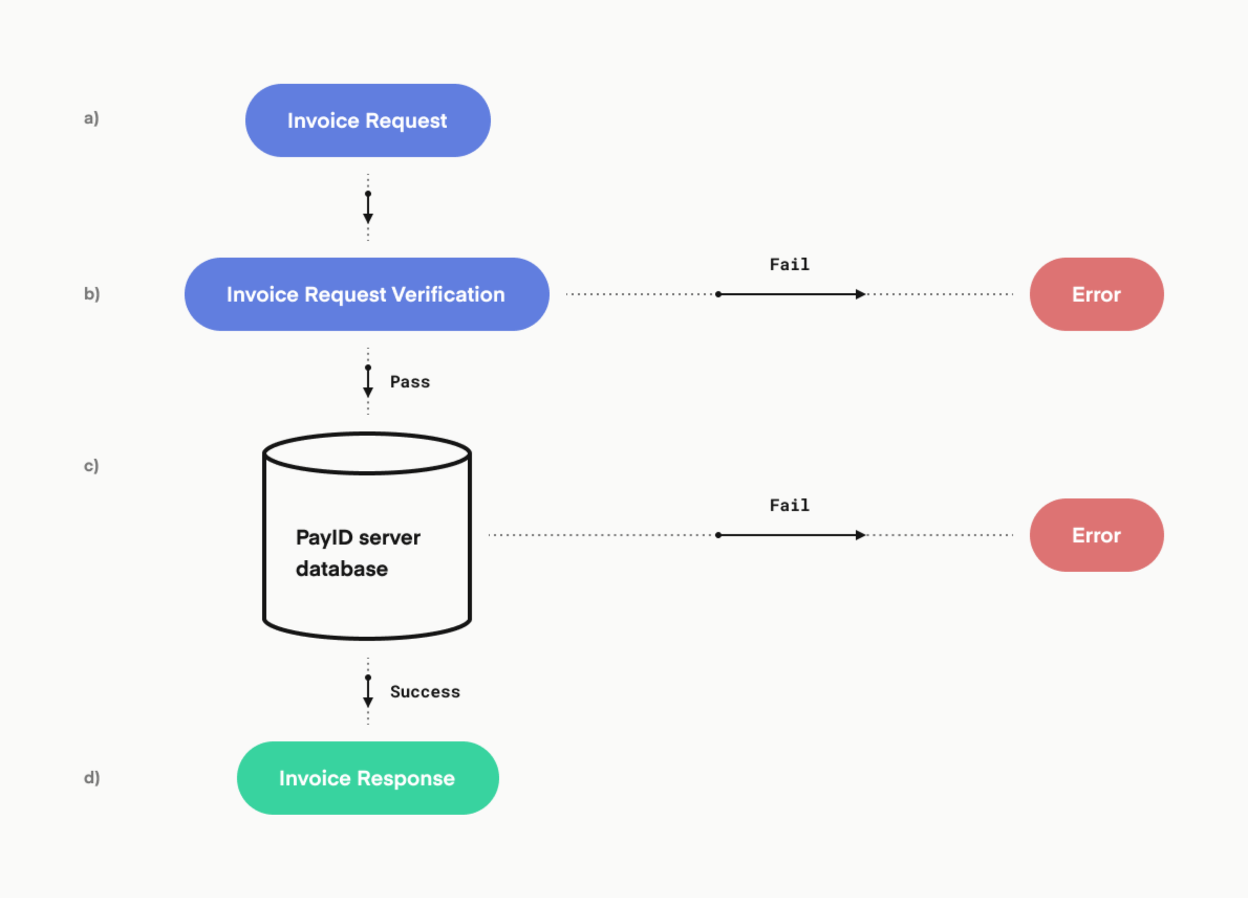
1) **GET /invoice request**: Processing steps by the originating institution to generate invoice request:



a) Receive the payment request from the originator with an amount and beneficiary's PayID (This is not a part of the PayID protocol flow.)
b) Establish a secure TLS session with the beneficiary institution as described in the session establishment section. The URL of the beneficiary institution is derived from PayID as described in paymentpointers.org/syntax-resolution/
c) If the TLS session is successfully established, send an HTTP GET /invoice request with a nonce parameter, otherwise exit.[5]

2) **Invoice response**: Processing steps by beneficiary institution to generate invoice response:

[5] Originating institution or beneficiary institution need not send an explicit PayID protocol error message in this case. This is a transport-layer communication error and will be handled by HTTP errors.



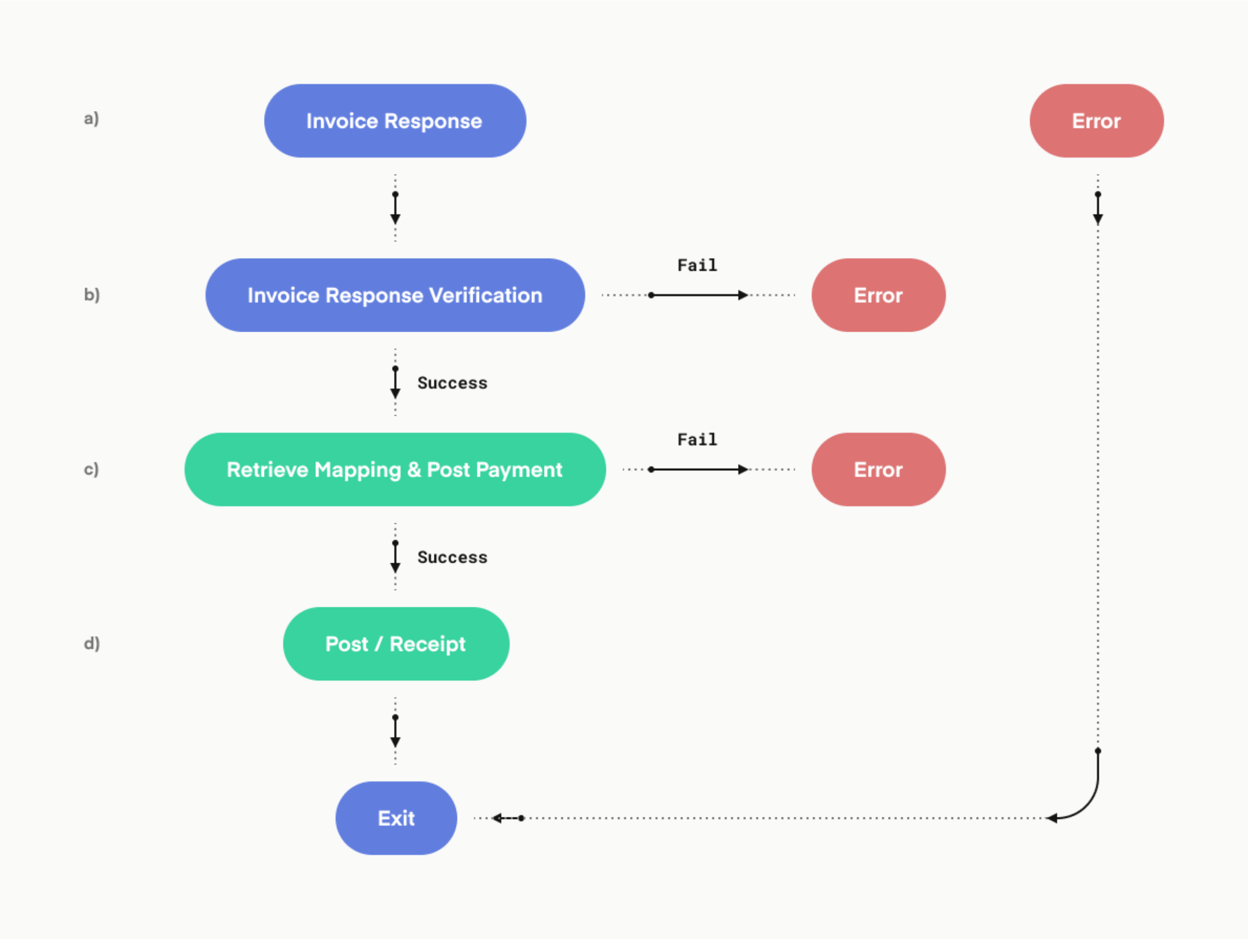
a) Receive the invoice request or error message response from the originating institution.

b) Verify if the incoming message has all the mandatory data in valid format and is correctly signed as described in the verifying invoice request or verifying error message section. If it is an error message and it verifies, exit.

c) If the invoice request message passes verification, query its database for the cryptographically signed ‘beneficiary → payment address’ information corresponding to PayID. This database of payment information is generated by the beneficiary institution as described in the PaymentInformation section. Otherwise if the invoice request message fails verification, it generates an error message. For details on error codes refer to the PayID protocol status communication section.

d) If the payment information exists in the database, then the beneficiary institution generates a cryptographically signed invoice response message as described in the generating InvoiceResponse and SignatureWrapper sections, otherwise it generates an error message. For details on error codes refer to the generating PayID protocol status communication section. Send invoice response or error message to the originating institution.

3) **POST /receipt:** Processing steps by originating institution to generate the receipt of payment on the provided payment address.



a) Receive invoice response 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 invoice response message or verifying error message section. If it is an error message and it verifies, exit.

c) Otherwise, if verification for invoice response passes, retrieve the payment address from the paymentInformation field of the invoiceResponse message and post the transaction on the corresponding address.
Otherwise generate an error response message as described in the generating error messages section. For details on error codes refer to the generating PayID protocol status communication section.

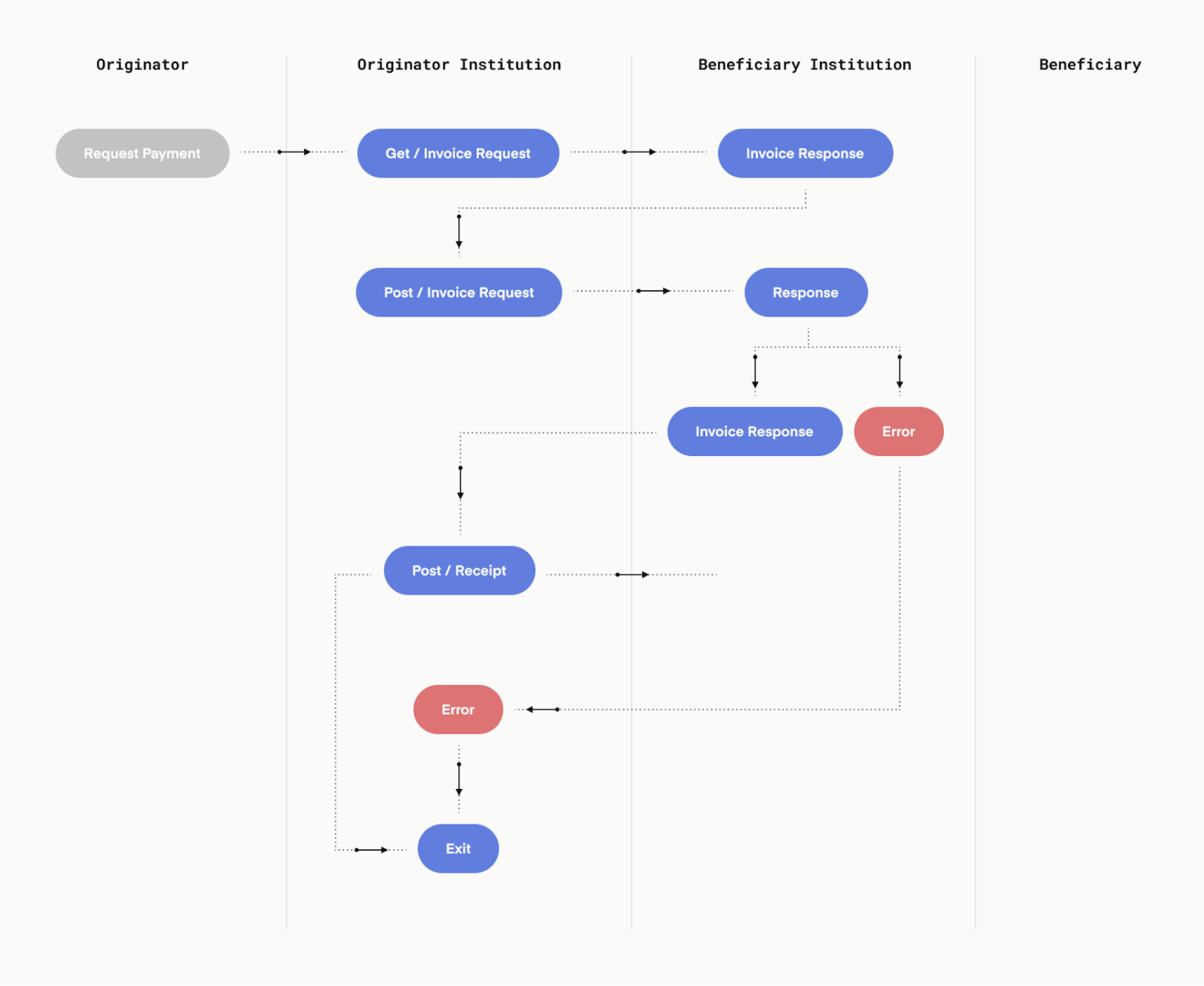
d) If the payment succeeds, then obtain the corresponding transaction information and generate a cryptographically signed receipt message as a proof of payment as described in the generating Receipt and SignatureWrapper sections.
Otherwise generate an error response message as described in the generating error messages section. For details on error codes refer to the generating PayID

protocol status communication section. Send POST /receipt or error response message to the beneficiary institution and exit.

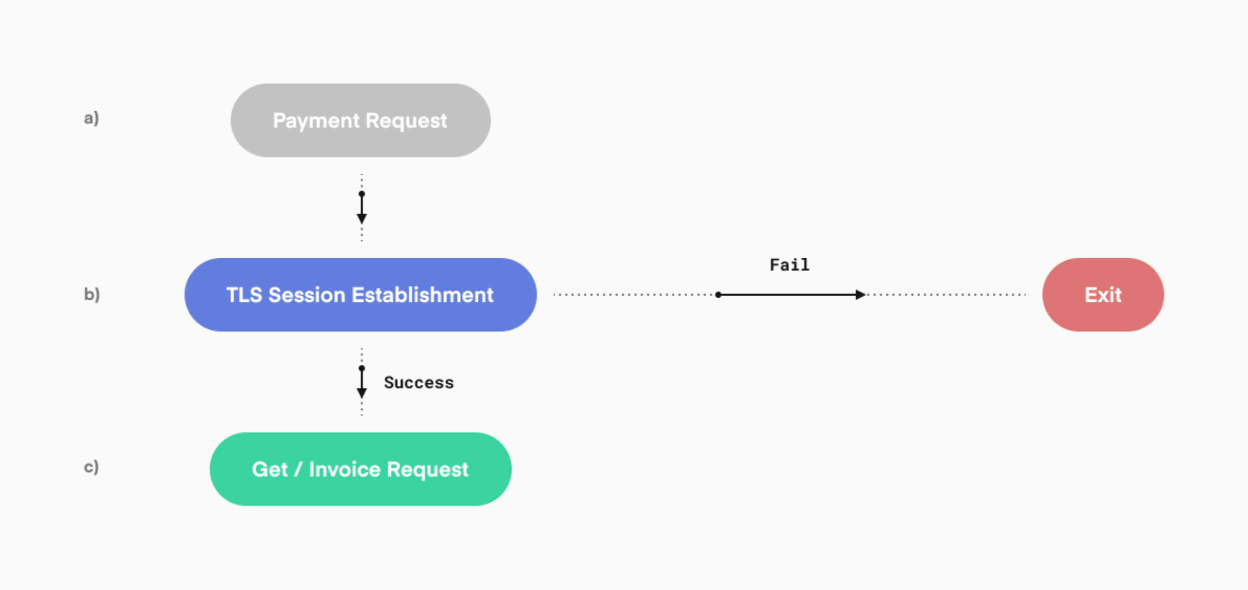
Upon receiving the signed receipt or the error response message, the beneficiary institution may choose to inform the beneficiary of the status of the payment (out-of-scope for the PayID protocol.)

III. Verifiable PayID protocol flow with compliance extensions: Allows for retrieving payment addresses, making payments and fulfilling compliance requirements such as Travel Rule in a secure and non-repudiable way with cryptographically signed proofs.

Moving a step ahead, we now present the verifiable PayID protocol flow with compliance-an extension to the verifiable PayID protocol that enables transacting parties to communicate and fulfil their compliance needs when either one or both the originating and the beneficiary institution are subject to regulatory or compliance requirements (“covered Institutions”) in a secure and non-repudiable way with cryptographically signed third-party verifiable proofs of meeting their compliance requirements.



1) **GET /invoice request**: Processing steps by originating institution

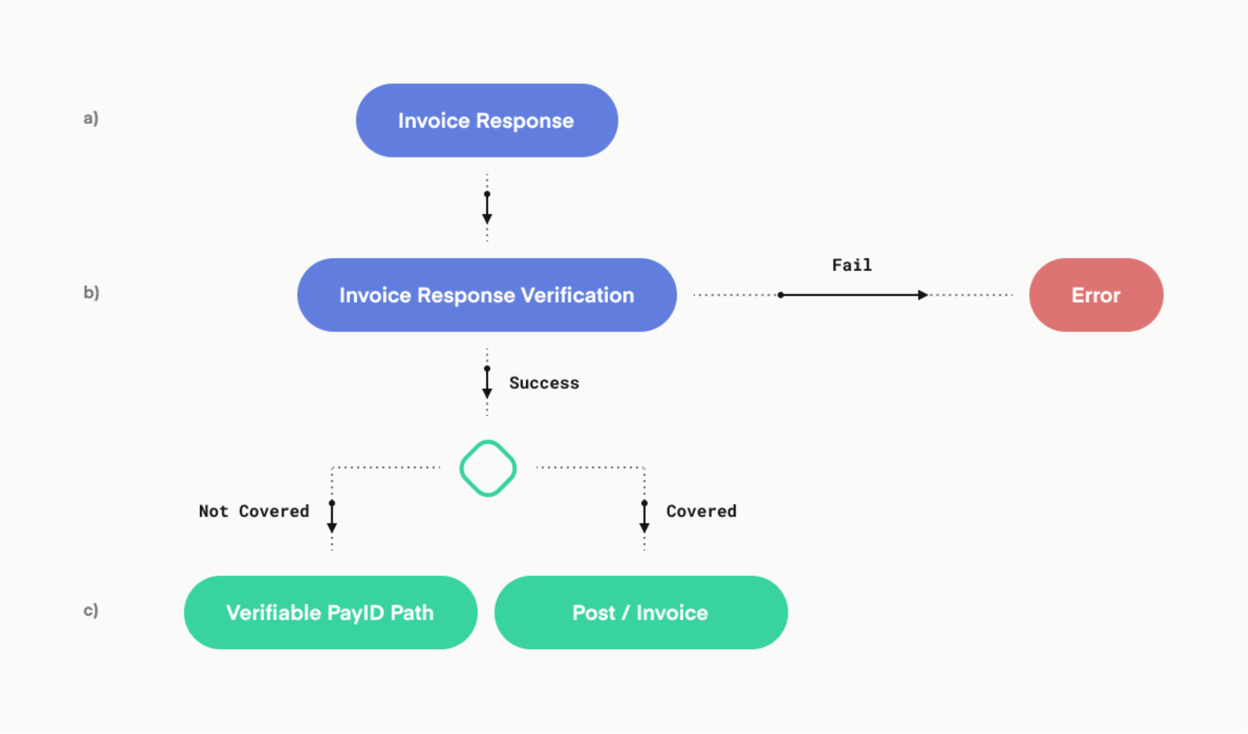


a) Receive the payment request from the originator with an amount and beneficiary's PayID (Again this is not a part of the PayID protocol flow.)
b) Establish a TLS session with the beneficiary institution as described in the session establishment section. The URL of the beneficiary institution is derived from PayID as described in paymentpointers.org
c) If the TLS session is successfully established send a GET /invoice request; otherwise exit.

2) **Invoice response**: Processing steps by the beneficiary institution
   a) Receive the invoice request 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 invoice request or verifying error message section. If it is an error message and it verifies, exit.
   c) Otherwise, if the invoice request message passes verification, generate invoice response message as described in generating invoice response and SignatureWrapper sections with the complianceRequirements field containing a list of all the compliance requirements that the beneficiary institution needs to meet.

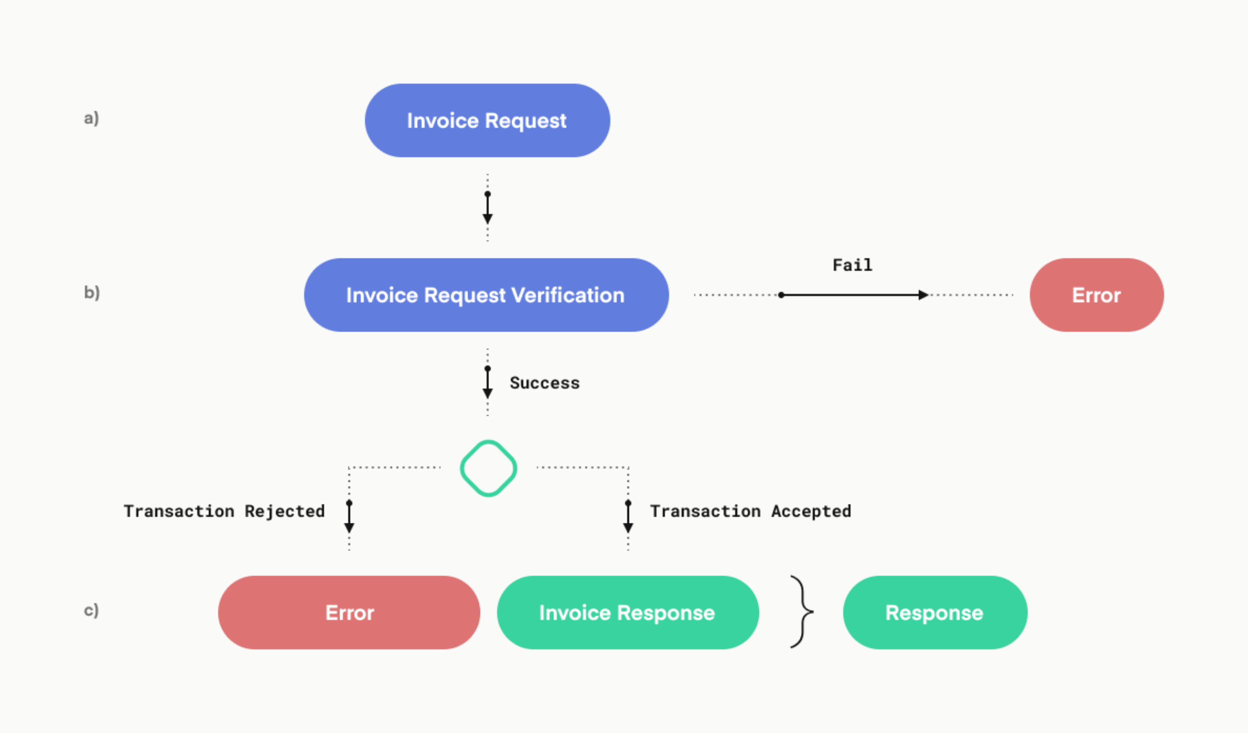
   Otherwise it generates an error message as described in the generating error message section and exit. For details on error codes refer to the generating PayID protocol status communication section. Send invoice response or error message to the originating institution.

3) **POST invoice request**: Processing steps by the originating institution



a) Receive invoice response or error message from 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 invoice response or verifying error message section. If it is an error message and it verifies, exit.
c) Otherwise if the invoice response message passes verification and if the originating institution is a covered institution, then generate a cryptographically signed invoice request message as described in the Compliance and SignatureWrapper sections that additionally contains the required data/information to be transferred as required by the listed compliance requirement (e.g. see TravelRule) in the ComplianceRequirements field of the invoice response message. Send POST /invoice request message to the beneficiary institution.
Otherwise (if the originating institution does not fall under compliance requirements i.e. it is not a covered institution), then the originating institution follows the verifiable PayID protocol path i.e. steps 3 (c-d) of the previous section.
Otherwise, if the invoice response message fails verification, generate an error message with appropriate error code as described in the generating error message section and exit. For details on error codes refer to the generating PayID protocol status communication section.Send error response message to the beneficiary institution.

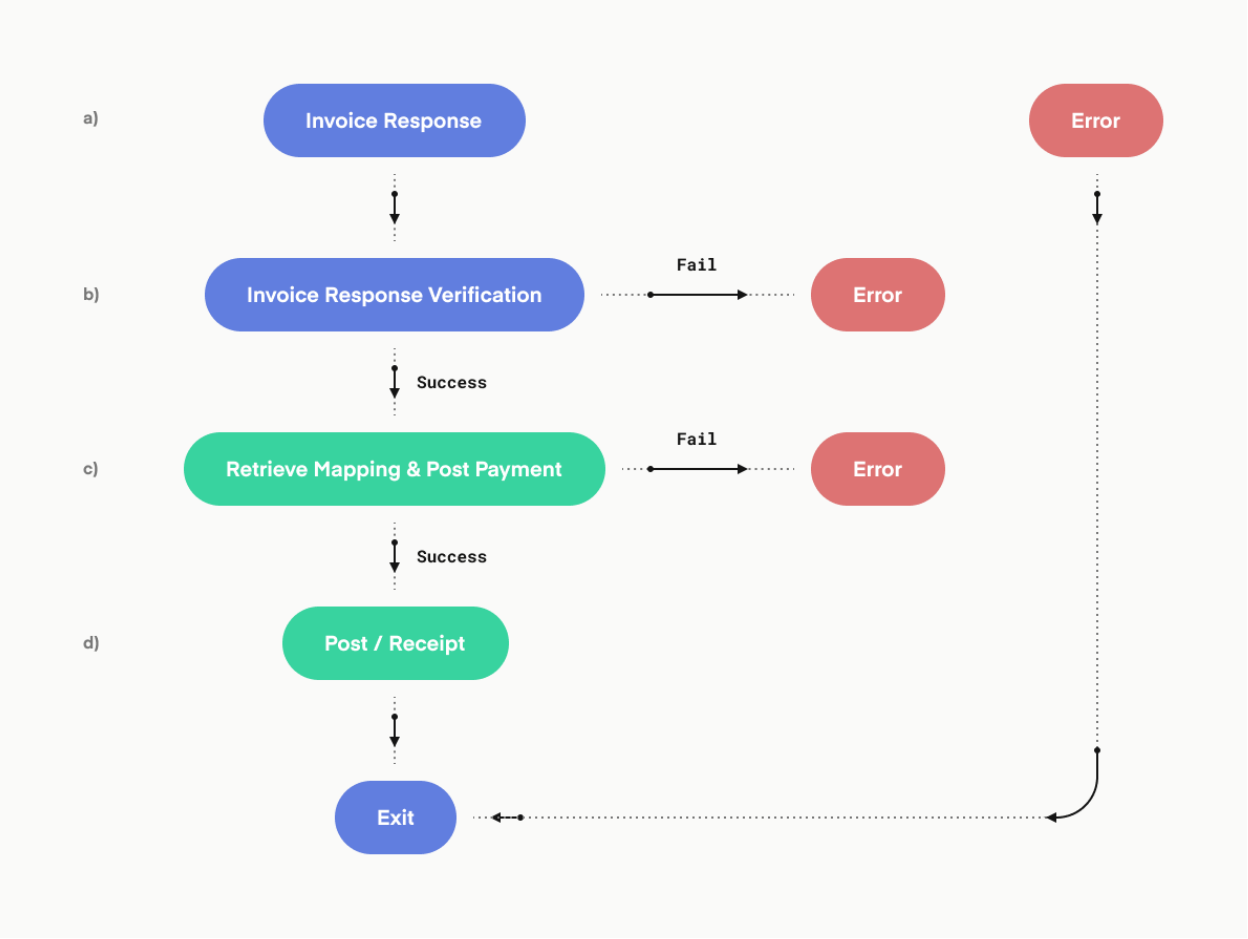
4) **Response**: Following are the processing steps by the beneficiary institution



a) Receive invoice request or a receipt 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 invoice request or verifying error message section. If it is an error message and it verifies, exit.
c) If the invoice request message verifies, retrieve the compliance data sent by the originating institution in the invoice request message and perform any checks. Otherwise if the invoice request fails verification, generate an error message as described in the generating error message section and exit. For details on error codes refer to the generating PayID protocol status communication section.
d) If beneficiary institution decides that it wants to proceed with the transaction with the given originating institution, then beneficiary institution generates cryptographically signed invoice response with an empty complianceRequirements fields as described in the generating invoice response and SignatureWrapper sections and sends it to originating institution

   Otherwise, i.e. after retrieving the compliance data and performing any checks, if the beneficiary institution decides that it does not want to proceed with the transaction with the given originating institution, generate an error message with appropriate error code as described in section generating error message section and send it to originating institution and exit. For details on error codes refer to the generating PayID protocol status communication section.

5) **POST receipt**: Following are the processing steps by the originating institution



a. Receive Response message from 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 invoice response or verifying error message section. If the Response message is an error response and it verifies, then exit (optionally inform the originator.)
c. Otherwise if the Response message is an invoice response message and it passes the verification, retrieve the payment address from the paymentInformation field of the invoice response message and post the transaction on the corresponding address.
d. If the payment succeeds, then obtain the corresponding transaction information and generate a receipt message as a proof of payment as described in the generating Receipt and SignatureWrapper section. Otherwise generate an error response message as described in the generating error message section. For details on error codes refer to the generating PayID protocol status communication section. Send POST /receipt or error message to the beneficiary institution and exit.

Security & Privacy Model

While the PayID 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.

There is necessarily some existing trust between the originator and the originating institution. The originating institution 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.

In realistic cases, the originator may be able to hold the originating institution legally accountable if the institution *provably* mishandles their funds. Similarly, the beneficiary may be able to hold the beneficiary institution legally accountable if their funds are mishandled. However, this mechanism requires that it be possible for either institution to establish that it acted properly and that the other institution 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 institution 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 institutions 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 institutions are fully malicious and can actively try to cheat each other. Our protocol does not require any trust relationship between the originating and beneficiary institutions. In other words, the protocol enforces honest behaviour 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 institutions 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 invoice response with the *nonce* field is a proof verifiable by a third party that the beneficiary institution generated 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 invoice response provides a signed confirmation of the list of compliance requirements that the beneficiary institution needs to meet.
3) Signed *paymentInformation* field in the signed invoice response 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 invoice response (in case the beneficiary institution received compliance data from originating institution) is a proof for originating institution that beneficiary institution 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-deniable cryptographic proofs for the beneficiary institution:

1) *complianceData* in signed invoice request message (Compliance 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 institution.
2) *invoiceHash* (that indicates beneficiary institution's signed compliance list) field in the signed receipt 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 fulfil them. This is a proof of compliance.

3) *transactionConfirmation* and *invoiceHash* fields in the signed receipt is a proof for the beneficiary institution that originating institution made the payment on the address provided by the beneficiary institution.

Fully compromisable originating and beneficiary institution 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 operations that the beneficiary institution MUST perform to generate cryptographically signed proofs.

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

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 institution's cryptographic keys used to cryptographically sign PaymentInformation are compromised, an attacker may impersonate as the beneficiary institution and sign malicious mappings ('beneficiary → attacker controlled payment address') to send to the originating institution. This would lead to indirection of funds by the originating institution 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 institution and beneficiary institution are compromised. They can, however, maliciously generate cryptographically signed invoice responses impersonating the beneficiary institution. They can also decrypt the messages sent by the originating institution. 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 institution and beneficiary institution SHOULD follow best practices described below 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.

Privacy

All application and user data stays private from 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.

PayID protocol message generation

SignatureWrapper

This message is an encapsulating wrapper for signing any PayID protocol messages. It allows for the generation of cryptographically signed third-party verifiable proofs of the contents of the messages exchanged between the participating originating and beneficiary institutions.

Field name	Required/ Optional	Type	Description
messageType	required	string	Type of contents delivered in message field
message	required	PaymentInformation || Invoice || Receipt || Compliance || error	Message body
publicKeyType	required	string	Public Key Infrastructure (PKI) system being used to identify the signing institution. e.g. “X509+SHA512” means an X.509 certificate as described in RFC5280 and SHA512 hash algorithm used to hash the entire contents of the SignatureWrapper message for signing.

publicKeyData	required	string[]	PKI-system data used to identify the signing institution used to create digital signatures over the message hash e.g. in the case of X.509 certs, contains one or more X.509 certs as a list upto but not including the root trust certificate.
publicKey	required	string	Contents of the public key
signature	required	string	Digital signature over the hash of the entire contents of the SignatureWrapper message using the private key corresponding to the public key in publicKey. This is a proof that the message was created by the publicKey holder.

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

PaymentInformation

This message MAY be wrapped inside the SignatureWrapper (in case of Verifiable PayID protocol and Verifiable PayID protocol with compliance extension flows.)

Field name	Required/ Optional	Type	Description
addressDetailsType	required	string	Specifies addressDetails payload
addressDetails	required	CryptoAddressDetails || ACHAdressDetails	Information necessary to send payment
proofOfControlSignature	optional	string	Signature proving ownership of the given address
paymentPointer	optional	string	The payment pointer of the original request

proofOfControlSignature is optional. This is a signature proving the ownership of the payment address by the beneficiary. If present, this is the digitally signed hash of the beneficiary's

payment pointer signed with the private key corresponding to the public key[6] of the beneficiary on the underlying network.
paymentPointer is optional. If wrapped in the SignatureWrapper, this field MUST be set to the beneficiary's Payment pointer or PayID to generate a signed proof of payment information by the beneficiary institution i.e. 'beneficiary → payment address' mapping.

addressDetails

This message is a field in the PaymentInformation message. addressDetails for each specific ledger MUST be registered at payid.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
ACHAddressDetails	accountNumber	required	string	ACH account number
	routingNumber	required	string	ACH routing number

Error

This message is wrapped inside the SignatureWrapper. It is used to communicate the PayID 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 PayID protocol status communication.
title	required	string	As described in RFC 7807, Title of the error as described in PayID protocol status communication.

[6] Note that this is the on-ledger key. All other keys used in the PayID protocol for secure communication and/or generating cryptographic proofs are separate PKI keys and are not related to this on-ledger key in any way.

statusMessage	optional	string	As described in RFC 7807. For further information about the status of the PayID protocol see PayID protocol status communication.
statusCode	required	integer	As described in RFC 7807. For further information on relevant HTTP status code for the error generated see PayID protocol status communication.
messageHash	required	string	This is the hash of the message corresponding to which this error is generated.

InvoiceResponse

This message is wrapped inside the SignatureWrapper. This message is sent by the beneficiary institution in response to the invoice request message.

Field name	Required/ Optional	Type	Description
nonce	required	string	Value copied from invoice request URL parameters.
expirationTime	required	integer (milliseconds from epoch)	This invoice is considered void and payments MUST NOT be made past the specified timestamp
paymentInformation	required	PaymentInformation	Contains details as to how a payment can be made to the beneficiary
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. E.g TravelRule data in case of Travel rule compliance requirement. Defaults to empty list
memo	optional	string	Field for adding additional metadata to a payment
complianceHashes	required	string[]	Hash of compliance data received in invoice request. Used to cryptographically correlate

			previously submitted compliance data messages with an upgraded invoice. Defaults to empty list

Receipt

This message is wrapped inside the SignatureWrapper. This message is sent by the originating institution as a proof of payment to the corresponding payment address sent in the invoice response message by the beneficiary institution.

Field name	Required/ Optional	Type	Description
invoiceHash	required	string	Hash of the invoice response message that this receipt is fulfilling
transactionConfirmation	required	string	Evidence of the submitted transaction on the payment address provided in the invoice response message . e.g. for cryptocurrencies, this would be the transaction output and for ACH transactions, this would be the trace number

Compliance

This message is wrapped inside the SignatureWrapper. This message is used by the originating institution to communicate the required compliance data mentioned in the complianceRequirements of the invoice response message by the beneficiary institution.

Field name	Required/Optional	Type	Description
messageHash	required	string	This is the hash of the previous invoice response message corresponding to which this compliance data is generated
complianceType	required	String	Type of the complanceData field. e.g. “TraveRule”
complianceData	required	TravelRule	Contents of compliance message

TravelRule

This message is an example of the kind of data in the complianceData field of Compliance message.

Field name	Required /Optional	Type	Description
originator	required	object	Top level field containing data about the originator (contents highlighted below)
userLegalName	required	string	Legal name of originator
accountId	required	string	Account ID within originating institution of the originator
userPhysicalAddress	required	string	Physical address of the originator
institutionName	required	string	Legal title of the originating institution
value	required	object	Contains fields shaded in red below
amount	required	string	Units of value
scale	required	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
timestamp	required	integer	Number of seconds since the Unix epoch
beneficiary	required	object	Top level field containing known data about the beneficiary (contents highlighted below)
institutionName	required	string	Legal title of the beneficiary institution
userLegalName	optional	string	Legal name of the beneficiary
userPhysicalAddress	optional	string	Physical address of the beneficiary
accountId	optional	string	Account ID within the originating institution of the beneficiary

PayID Protocol Message Verification

1. Verifying invoice request message

Upon receiving an invoice request message (that contains message body i.e. POST message) from the originating institution, the beneficiary institution performs the following verification steps:

a) Validates the *publicKey* of the sender using the *publicKeyData* and verifies the signature on the compliance message
b) Verifies if data in the *complianceData* field meets their compliance requirements.

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 signed error message as described in the generating error message section. For details on error codes refer to the section PayID protocol status communication.

2. Verifying invoice response message

Upon receiving an invoice response message from the beneficiary institution, the originating institution performs the following verification steps.

a) Validates the *publicKey* of the sender using the *publicKeyData* and verifies the signature on the invoice response message
b) Validates the *publicKey* of the sender 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 value in the *nonce* field matches the nonce value in the corresponding invoice request message previously sent by the originating institution
d) Checks if the time in the *expirationTime* field is less than the current system time[7] of the originating institution
e) In case the originating institution sent compliance data in the previous invoice request, the originating institution generates the hash of the sent compliance data in the previous invoice request and checks if it matches with the contents of the *ComplianceHash* field in the received invoice response message.

All the verification steps MUST pass. The originating institution proceeds to the next step only if the previous step passes, otherwise it generates the relevant signed error message as described in the generating error message section. For details on error codes refer to the PayID protocol status communication.

[7] Originating institutions 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

3. Verifying receipt message

Upon receiving a receipt message from the originating institution, the beneficiary institution performs the following verification steps:

a) Validates the *publicKey* of the sender using the *publicKeyData* and verifies signature on the receipt message

b) Generates hash of the invoice response message that the beneficiary institution previously sent to the originating institution corresponding to the received receipt message and verifies if it matches the *invoiceHash* field in the received receipt message.

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 signed error message as described in the generating error message section. For details on error codes refer to the PayID protocol status communication.

4. Verifying error message

Upon receiving an error message from either institution, the receiving institution performs the following verification steps:

a) Validates the *publicKey* of the sender using the *publicKeyData* and verifies signature on the error message.

b) Generates the hash of the previous message sent by the verifying institution corresponding to which this error is generated by the sending institution and checks if the hash matches the *messageHash* field of the received error message.

All the verification steps MUST pass. The verifying institution proceeds to the next step only if the previous step passes, otherwise it drops the error message.

Session establishment

We recommend RFC 8446 for TLS1.3 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.

PayID protocol status communication

The following status codes must be communicated in the error message specific to the error in the corresponding PayID protocol message.

type	title	status_message	HTTP status_code
"https://payid.org/invoice/not-found"	“Data Not Found”	“PayID does not exist in the database”	404
"https://payid.org/invoice/	“Certificate	“A certificate is required to	400

certificate-required"	Required”	verify the signature”	
"https://payid.org/invoice/certificate-expired"	“Certificate Expired”	“The certificate sent by the sending entity has expired”	400
"https://payid.org/invoice/certificate-revoked"	“Certificate Revoked”	“The certificate sent by the sending entity has been revoked”	400
"https://payid.org/invoice/certificate-invalid"	“Certificate Invalid”	“The certificate sent by the sending entity is invalid”	400
"https://payid.org/invoice/signature-invalid"	“Signature Invalid”	“Could not verify the signature using the provided *publicKey* ”	400
"https://payid.org/invoice/invalid-invoice-hash"	“Invalid/Missing invoiceHash”	“*invoiceHash* is invalid or missing”	400
"https://payid.org/invoice/invalid-compliance-hash"	“Invalid/missing complianceHash”	“*complianceHash* is invalid or missing”	400
"https://payid.org/invoice/invalid-compliance-data"	Invalid/missing/incomplete complianceData	“*complianceData* is invalid, missing or incomplete”	400
"https://payid.org/invoice/invoice-expired"	“Invoice Expired”	“The current system time is past the expiration time on invoice”	400
"https://payid.org/invoice/invalid-nonce"	“Missing/Mismatch nonce”	“*nonce* is invalid or missing”	400
"https://payid.org/invoice/legal-reasons"	“Legal Reasons”	“Entity sending the message does not want to proceed with the transaction due to legal reasons”	451
"https://payid.org/invoice/invalid-proof-of-control-signature"	“Missing/Invalid proofOfControlSignature”	“*proofOfControlSignature* is invalid or missing”	400
"https://payid.org/invoice/invalid-receipt"	“Invalid receipt”	“The receipt 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)

GET Headers and Payment address response

The GET request and POST /invoice request MUST be transmitted using the appropriate Accept-header as defined here. Other payment networks will be able to establish standard headers for their networks over time at payid.org. Initially, we propose standards with the headers specific to XRP, ACH and ILP payment networks.

XRP

Accept-header	**Description**
application/xrpl-mainnet+json	Returns XRPL mainnet xAddresses
application/xrpl-testnet+json	Returns XRPL testnet xAddresses
application/xrpl-devnet+json	Returns XRPL devnet xAddresses

ACH

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

ILP

Accept-header	**Description**
application/spsp4+json	Returns destination address and shared secret

Key Management

Building blocks:

1) **Web Public Key Infrastructure (PKI):** Web PKI is the root of trust in this system. We choose to rely on the existing web PKI (despite its several pitfalls) because first, most, if not all, e-commerce and online banking is secured by web PKI and each of those alone

account for trillions of dollars of economic activity per year [8] and second, because web PKI is relatively easier to set-up and has universal browser support. In our protocol, the originating and beneficiary institution must each have a long-term public/private key pair (with X.509 certificate (hopefully) obtained from a trusted Certificate Authority.)

2) **Cold system (very low probability of compromise)**: Cold system here refers to an offline system /computer which is securely airgapped. In the context of PayID protocol, this system is used to generate the database of signed 'beneficiary → payment address' payment information mappings in the format described in the PaymentInformation section. The short-term private key, which is signed by the long-term private key mentioned above, used to sign the payment information is never online. This short-term private key MUST be kept offline in a safe place. This database of payment information must be kept in the hot system (described below) along with the corresponding short-term public key.

3) **Hot system (higher probability of compromise)**: Hot system here refers to the always online system that is connected to the Internet. The originating institution and beneficiary institution each should use a different short-term key pair signed by the long term private key mentioned above for:
    a) TLS handshake as described in the session establishment section.
    b) Generating cryptographic signature for invoice request, invoice reponse, receipts and error messages.

   This short-term key pair is kept online in the hot system.

Key Hierarchy

In this section we describe the key hierarchy for signing PaymentInformation, InvoiceResponse, Compliance, Receipt and Error 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 originating and beneficiary institutions 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.

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.

[8] https://www.shopify.com/enterprise/global-ecommerce-statistics#1

Short-term EC key-pair generation (***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 Error 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 PayID 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 (***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 a signed certificate for $PK1_o$ with MSK_o. This key-pair is needed for secure TLS session establishment and for signing Compliance, Receipt and Error messages.
2) On a hot system, save the following:
    a) signed $PK1_o$,
    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.

Acknowledgements

We thank Xpring and Ripple leadership for providing resources and support for this work. We would especially like to extend our gratitude to David Fuelling, Doug Purdy, Ethan Beard and Will Liu for providing constructive feedback throughout the process of writing this protocol. We are also thankful to our wonderful team of engineers — Dino Rodriguez, Hans Bergen, Keefer Taylor, Stephen Gu, Ted Kalaw, Tyler Longwell, and Tyler Storm — their insights and suggestions.