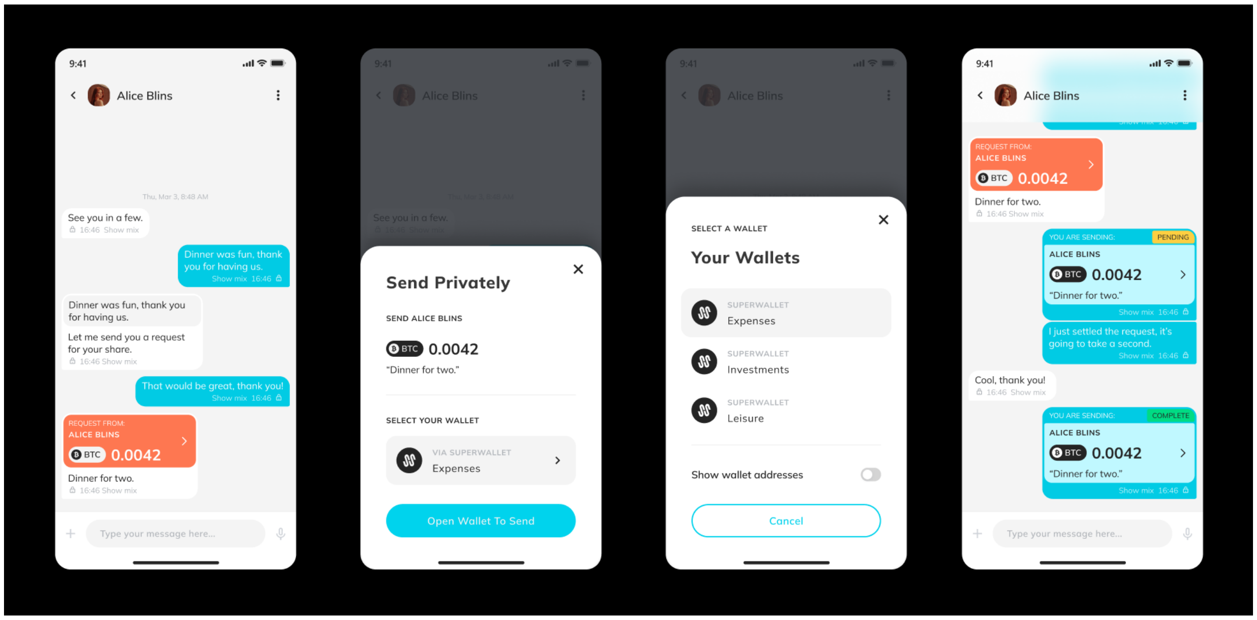
Payments Mechanism for xx messenger

Overview

The xx messenger will have payments integration, allowing a variety of wallets and currencies to be sent through the platform.

The xx messenger is planned to operate as a transport medium for transaction data, adding metadata protection to any blockchain transaction. It is not planned to have wallet signing ability at this time. Instead, it will have an API and UI which allows other wallet apps to register themselves with the messenger and be used to process payments. An example of this UI can be found below.



See you in a few.

Dinner was fun, thank you for having us.

Dinner was fun, thank you for having us.

Let me send you a request for your share.

That would be great, thank you!

Dinner for two.

"Dinner for two."

I just settled the request, it's going to take a second.

Cool, thank you!

This is a product mockup and is subject to change.

After selecting a wallet, it will open to allow the user to approve and sign the transaction.

Key points:

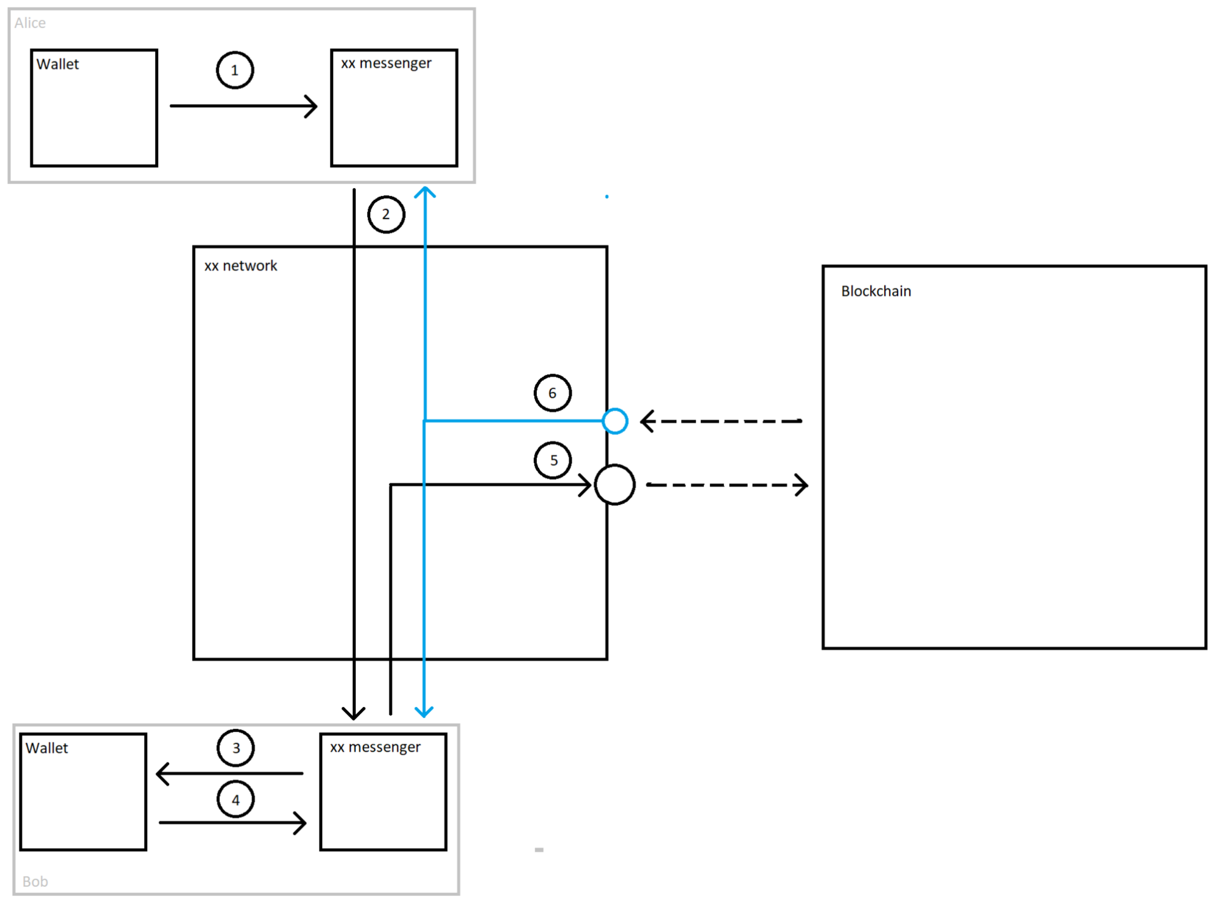
1. You can register and unregister any wallet (ready for integration by the wallet developer) to your xx messenger.
2. You can request payment anonymously through xx messenger using any of your registered wallets.

3. You can pay anonymously (see privacy section at the end of the document) through the xx messenger using any of your registered wallets.
4. You can track the payment status, whether it's pending or complete, without leaving your xx messenger.

On Android, this will use the intents system for cross-app communication. On iOS, we will use x-callback-url.

Architecture

In general, the system will allow clients to send requests to each other, submit them to arbitrary blockchains, and then listen on the network for the result.



General Approach

In this architecture, Alice is asking Bob to be paid. She sends a request to Bob (1 & 2). Bob uses his wallet to create the signature (3 & 4), which is then submitted to the blockchain that handles the payment (5) over the xx network to hide Bob & Alice's IP. This is outputted to a predefined xx network address or a trusted processing bot. This can be picked and delivered to the destination blockchain. It is expected that many will run these shuttle bots as a service. Along with the signed transaction, a Receipt Channel destination address and public key on the xx network will be provided to send receipts showing the status of the transaction to both Bob and Alice.

There are two variants of this architecture—networks that provide independently validatable proofs of transaction acceptance and those that do not.

Networks With Proofs

Some networks can create receipts that prove transactions have been accepted into a block of varying veracity. For example, within the Bitcoin network, a proof consisting of a Merkle proof that a transaction is in a block alongside the winning hashes for that block as a series of preceding and/or following blocks, provides a reasonable proof that the transaction occurred on the Bitcoin network because the computational complexity of generating the winning hashes would be difficult to replicate in the period between the submission of the transaction and the reception of the receipt. In general, this is possible with all proof of work networks, but how convincing the proof is will be proportional to the hashpower mining the network.

Other networks may provide zero-knowledge proofs or other technologies to create similar self-proving receipts in such networks.

Such receipts can be provided over an unsecured channel, allowing any party to provide the receipt. This will be done by predefining a network address for the blockchain network. For example, for Bitcoin the address `BITCOIN////////////////////////////////////D` could be used. This address acts as a public bulletin board, allowing any party to shuttle the payloads to the Bitcoin network and provide receipts.

Code will need to be added to the messenger to evaluate these proofs on a per-network basis.

Networks Without Proofs

Networks that cannot provide receipts with proofs (most proof of work networks) generally assume the user has a validator they trust or are running themselves. In these networks, the bot they are communicating with to process the transaction through the xx network must be trusted and will sign the receipt so it cannot be forged. In such a scenario, the wallet for the requestor will provide the bot information.

There will be a standard signature scheme used to sign these receipts that the xx messenger will be able to evaluate.

Technical Details

① Creating the Request

A payment request needs to be created in order to send to Bob. It needs to contain a destination address and an amount

```
request = destinationAddress, Amount, Data, xxNetworkBlockchainDestination
```

This can either be generated by a registered wallet or by the messenger itself for a registered wallet.

`xxNetworkBlockchainDestination` will contain just the public address for networks that provide proofs. For networks without proofs, it will provide a unique identity along with an RSA public key the bot will use for signing, which the wallet provider is designating as trustworthy.

The `data` field will be an arbitrary field that the wallet returns to allow for flexible protocols.

② Transmitting the Request

Alice transmits the request to Bob over the xx network under end-to-end encryption. The request must contain a Receipt Channel that both Alice and Bob will use to pick up notifications about the transaction's history.

```
xxNetworkRequest =
      destinationAddress, amount, Data, xxNetworkBlockchainDestination
```

This request will be encrypted with the messenger's end-to-end encryption before being sent over the network. Details can be found in The xx network End To End Cryptographic Protocol.

```
encryptedXxNetworkRequest = E(xxNetworkRequest)
```

The request is made under end-to-end encryption, which will use a unique key for encrypting the message. Both parties will use that key as the input to the KDF, which will generate an RSA private key and a salt that will define a channel identity.

```
Receipt Channel = ID = H(RSAPub, salt)
```

The `data` field will be an arbitrary field that the wallet returns to allow for flexible protocols.

③ Sending the request to a Wallet

When a wallet is registered with the xx messenger, it will have to open its own interprocess communication (either intent or x-callback-url) to accept requests from the messenger. When the wallet is selected to process a payment, it will receive the request on this communication. The request will include:

```
walletRequest = destinationAddress,amount,data
```

The `data` field will be an arbitrary field which is returned by the wallet to allow for flexible protocols.

④ Returning the Signed Transaction to xx messenger

Once the wallet has signed the transaction, the signed transaction and any ancillary data will be returned to the xx messenger, identified by the transaction ID.

```
signedTransaction, data, xxNetworkBlockchainDestination
```

The `data` field will be an arbitrary field that the wallet returns to allow for flexible protocols.

⑤ Transmitting the transaction

Bob's xx messenger will send the transaction to the provided network address, whether a public address or a bot. This payload does not need to be encrypted.

```
privateTransaction = signedTransaction,data
```

The recipient will submit it to the designated blockchain.

⑥ Receiving the receipts

Be it a trusted bot or a public access point, the submitter will provide receipts over the provided channel, which both Alice and Bob can read.

Receipts will be encrypted with the channel's Public RSA Key so that they can only be decrypted with the private key, ensuring only Alice and Bob can read the payloads.

Two receipts will be sent. The first will be upon successful submission of the transaction to the blockchain. The second will be when the transaction is accepted or rejected.

For systems without proofs, the receipts will generally be Merkle proofs proving the transaction is in a block, along with a signature from the bot on the block hash.

API

There will be a messenger API that allows this to function. On Android, this will use the intents system for cross-app communication. On iOS, this will use x-callback-url.

The API is likely to have slight differences due to architecture-specific requirements, but generally, they should follow the following proposal. This API is subject to change during implementation.

`RegisterWallet`

Registered a specific wallet address from a wallet application which can be used to send or receive payments

Input	Type	Description
`Address`	`String (Optional)`	Address of the wallet that is registered. It can be left blank to defer selection to the wallet application. It must be Unique.
`Address Name`	`String (Optional)`	Name of the address of the wallet that is registered (i.e., Leisure wallet) Can be left blank to defer selection to the wallet application.
`Wallet Name`	`string`	Name of the wallet application that holds the registered address. If the Address is blank, this must be unique to all others with blank addresses.
`Currency`	`string`	A tag describing what currency this address holds. This tag will be used in requests if the messenger doesn't recognize the currency to use proper icons.
`Wallet Icon`	`image (Optional)`	Icon displayed when describing the address from this wallet.
`Request Command`	`string (uri)`	Link that the messenger uses to return to the Wallet Application with this specific address to make a payment request to another party. String interpolation in the URI is TBD.
`Approve Command`	`string (uri)`	Link that the messenger uses to return to the Wallet Application with this specific address to process a payment request from another party. String interpolation in the URI is TBD.
`View Command`	`string (uri)`	Link that the messenger uses to return to the Wallet Application with this specific address to view the status of the address and/or a specific payment. String interpolation in the URI is TBD.

Return	Type	Description
`Success`	`boolean`	`True` if the operation was successful, `false` if it was not
`Error Code`	`int`	Error code integer. Error Return on failure, Platform Specific. Error code 0 means no error.
`Error String`	`string`	Error description. Error Return on failure, Platform Specific.

`UnregisterWallet`

Unregister a specific wallet address from a wallet application that has already been registered with the messenger.

Input	**Type**	**Description**
`key`	`string`	Address of the wallet that is registered or Wallet Name
`type`	`boolean`	0 - the key is an address, 1 - the key is a wallet name.

Return	**Type**	**Description**
`Success`	`boolean`	`True` if the operation was successful, `false` if it was not
`Error Code`	`int`	Error code integer. Error Return on failure, Platform Specific. Error code 0 means no error.
`Error String`	`string`	Error description. Error Return on failure, Platform Specific.

Commands

The commands which are passed are what the xx messenger will use in order to call to the wallet when the payment is to be made.

The commands should be structured as follows:

`Request`

Command that the xx messenger calls when the user wants to request a payment to another party

Input	**Type**	**Description**
`Address`	`string`	The address that the user selected to make the request from. If blank, the wallet app is expected to allow the user to select an address.
`Amount`	`string (hex)`	Amount of the given currency which is to be sent. Encoded in hex with the format “`0xXXXXX`”. The number should be denominated in the smallest value possible in the currency. 1 satoshi in bitcoin, for example.
`Request description`	`string`	A description of the payment. It was entered by the user in the app.

Return	Type	Description
`Success`	`boolean`	`True` if the operation was successful, `false` if it was not (for example, the user canceled the transaction)
`Data`	`bytes`	Arbitrary data provided by the wallet to allow for flexibility
`xxNetworkBlockchainDestination`	`string`	The destination processor for the request. If it is a network that provides self-proving receipts, this will just be a public address. If it is for a network that does not, this must include an RSA Public key which will be used to evaluate the signature from the bot.
`transactionTag`	`string`	A tag that can be used to return to this transaction at a later time using the `View Accessor`. Optional.
`Error Code`	`int`	Error code integer. Error Return on failure, Platform Specific. Error code 0 means no error.
`Error String`	`string`	Error description. Error Return on failure, Platform Specific.

Approve

Command that the xx messenger calls when the user wants to approve or send a payment to another party

Input	Type	Description
`Address`	`string`	The address which the user selected to make the request from. If blank, the wallet app is expected to allow the user to select an address.
`Amount`	`string (hex)`	Amount of the given currency which is to be sent. Encoded in hex with the format “0xXXXXX”. The number should be denominated in the smallest value possible in the currency. 1 satoshi in bitcoin, for example.
`Request description`	`string`	A description of the payment. It was entered by the user or the requestor.
`xxNetworkBlockchainDestination`	`string (Optional)`	The destination processor for the request. If it is a network that provides self-proving receipts, this will just be a public address. If it is for a network that does not,

		this must include an RSA Public key which will be used to evaluate the signature from the bot. If blank, the Accessor should provide one. If the selected destination is incorrect or blacklisted, an error should be returned.
`Data`	`bytes`	Arbitrary data provided by the wallet to allow for flexibility

Return	**Type**	**Description**
`Success`	`boolean`	`True` if the operation was successful, `false` if it was not (for example, the user canceled the transaction)
`Data`	`bytes`	Arbitrary data provided by the wallet to allow for flexibility
`signedTransaction`	`string`	The signed transaction and any other data that must be communicated to the blockchain network in order to handle the transaction.
`xxNetworkBlockchainDestination`	`string`	If the event that no `xxNetworkBlockchainDestination` was provided, provide one here. Do not provide one if one already was provided on the input.
`transactionTag`	`string`	A tag that can be used to return to this transaction at a later time using the `View Accessor`. Optional.
`Error Code`	`int`	Error code integer. Error Return on failure, Platform Specific. Error code 0 means no error.
`Error String`	`string`	Error description. Error Return on failure, Platform Specific.

View

Command that the xx messenger calls when the user wants to view data in the wallet

Input	**Type**	**Description**

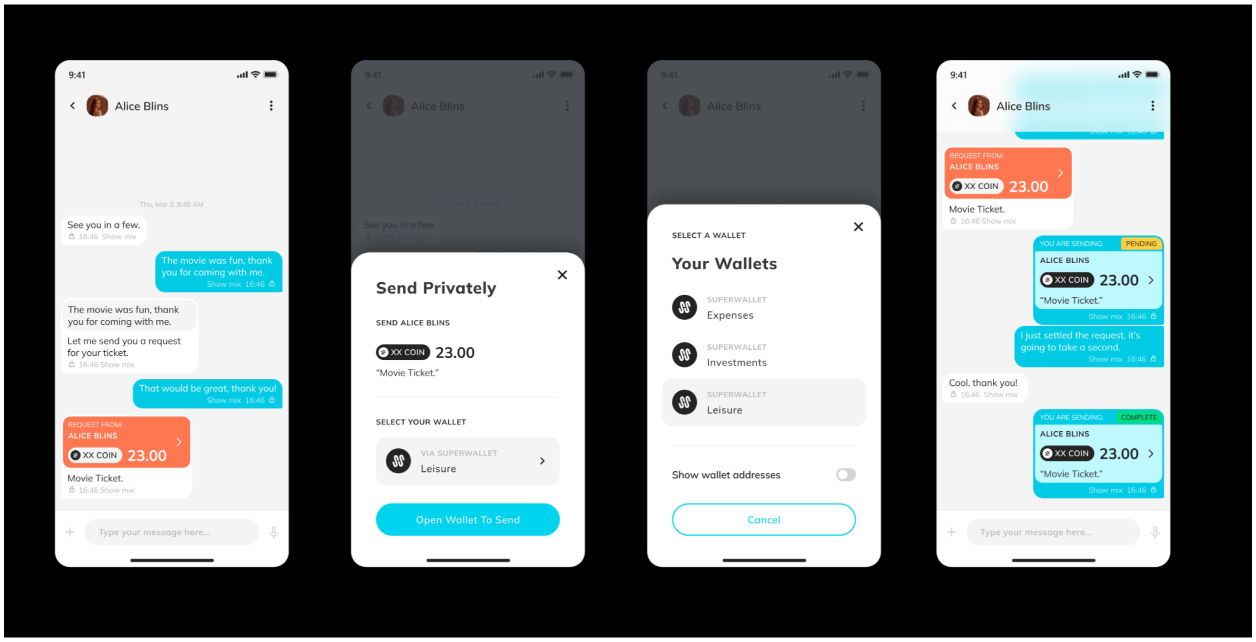
`Address`	`string`	The address which the user selected to make the request from. If blank, the wallet should show all registered address
`transactionTag`	`String (optional)`	If the user is attempting to view a specific transaction, this tag will be provided. If blank, just show the address

Return	Type	Description
`Error Code`	`int`	Error code integer. Error Return on failure, Platform Specific. Error code 0 means no error.
`Error String`	`string`	Error description. Error Return on failure, Platform Specific.

Further discussion

xx coin

Being native to the xx platform, a custom backend will be needed for xx coin. However, because the messenger is already polling the xx network, it is already capable of knowing recent blocks and does not need a signature in order to validate a Merkle proof of the presence of a transaction in a block. There will be RFPs in order to integrate xx coins with wallets, but the underlying submission code will be built internally.



See you in a five. 16:46 Show mix

The movie was fun, thank you for coming with me. Show mix 16:46

The movie was fun, thank you for coming with me.

Let me send you a request for your ticket. 16:46 Show mix

That would be great, thank you! Show mix 16:46

Movie Ticket. 16:46 Show mix

"Movie Ticket."

I just settled the request, it's going to take a second. Show mix 16:46

Cool, thank you! 16:46 Show mix

This is a product mockup and is subject to change.

Privacy

This payment system protects the IP and sender information of payment to any blockchain, but it does not break any on-chain links between accounts. Another service or on-chain support would be needed for that. Furthermore, if the wallet used makes non-private lookups to the target blockchain, it can break any privacy provided.

Front Running

This system is designed to enable decentralized and private consumer access to cryptocurrency payments. Because the payments are posted publicly before making it to the chain, they are vulnerable to front running. This means an attacker can see which transactions are pending and submit an alternate transaction with a higher weight in order to take advantage of market changes due to the transaction. This implementation is intended for consumer applications, not investment, so this issue is not particularly significant here.

Smart Contract support

This system is designed for payments primarily, not smart contracts. But by the usage of the data field, as well as custom currency tags, it may be possible to route smart contracts through the system.