SYRE

A Transaction Surety Solution

Syre Whitepaper Version 1.0
May 2019
Robert Kornacki <robk@syre.io>
Syre.io

Abstract:
Syre is a tokenless invoice based protocol designed to make sending cryptocurrencies easy and worry-free. It has neither token nor blockchain of it's own, but is instead an extension to current systems in place. Syre only requires the ability to append data to transactions put together with signing & encrypting of data via asymmetric cryptography. As such it is a blockchain agnostic protocol that is easy to implement while providing users with peace of mind for every transaction. Furthermore it opens up a new realm of possibilities for functionality and UX in the standard transaction experience through Syre Extensions.

1. Introduction

Syre was developed due to the realization that the Transaction Surety problem still hasn't been addressed in the blockchain space even after all of these years.

The Transaction Surety problem is something we've all experienced before but have never put a name to. It is exemplified every time we send a transaction, whether it be a payment to a friend or depositing coins into an exchange. In a nutshell, it is the fact that we, the sender of funds, have absolutely no guarantee(surety) that the money will actually arrive at the correct wallet we expect.

With the systems in place currently for cryptocurrencies, there are a plethora of ways the whole interaction can go wrong:

1. *You messed up copying the entire address from a webpage/email*
2. *You hit a key along the way while pasting the address and didn't notice*
3. *You have a clipboard editing malware which changed the address unbeknownst to you*
4. *The entity which provided you with the address had an issue of their own and provided you with a malformed or incorrect address*
5. *Etc*

And yet with such an error-prone process we still somehow gleefully exclaim to new users that blockchains are immutable and thus there is no recourse when things go wrong. Is it any surprise that there are many skeptics when the user experience is of such low quality?

Furthermore, something that is never even talked about, yet felt all too astutely whenever we send transactions ourselves, is the reality that the sender of the money has 100% of the responsibility with 100% of the funds. What other system on earth expects flawed and erring humans to bear all of the responsibility with everything at risk? Now factor into the equation that what is at risk is money, and one begins to wonder how anyone could ever consider this "good enough" let alone expect mass-adoption to be right around the corner.

A recent survey conducted by FIO exemplifies this problem. They found that for cryptocurrency users who have been in the ecosystem for at least 3 years, only up to 40% of these individuals actually felt comfortable & confident while sending transactions. This number tumbled all the way down to 21% when newer cryptocurrency users were polled. This means that in total, 75% of cryptocurrency users range from slightly worried all the way to extremely nervous whenever they send funds. The fact that this hasn't been addressed shows just how immature the blockchain space still is.

To finally bring Transaction Surety to cryptocurrency users, there are three key points that need to be tackled:

1. **Proof of address ownership**
2. **Offsetting and minimizing risk to near zero**
3. **Removing user input error as much as possible**

2. Solving Transaction Surety

In order to address the points above, and thus solve the Transaction Surety problem, we need to start from the foundations of how cryptocurrencies are sent.

Currently blockchains run on a purely push-based system for sending value. The simplicity of this is great, yet we run into all of the issues we've touched on earlier, which impairs it from being a good system as long as humans are involved.

Instead of a push-based system then, what if we use a request-based (invoice) system and change the dynamics entirely from the ground up?

2.1 Proof Of Address Ownership

One of most important aspects of providing Transaction Surety to users, is proving address ownership. It is the most obvious solution to the issue at hand. Simply provide proof that the address the payer is sending money to is owned by the expected receiver.

In a push-based system, this isn't doable unless there was already a previous exchange of information beforehand, and evenso there are no set standards for this today.

With an invoice based system, this can be done very simply. If an invoice is sent on-chain, attached as appended data to a transaction, then the very fact of the invoice arriving proves that the sender of the transaction owns his/her address. There is no way someone other than the owner of the private key for that address sent the transaction.

2.2 Offsetting And Minimizing Risk To Near Zero

Next we must face how to offset and minimize risk for the payer.

As mentioned previously, in the classic push-based model the payer takes on 100% of the responsibility with 100% of the funds. However with an invoice based model, we flip this entirely.

Instead of all of the risk being on the payer, it is severely mitigated by moving it over to the invoice creator. The invoice creator has to take the risk of making sure the invoice arrives at the

payer's wallet while the payer simply leverages the invoice's proof of address ownership to make the risk zero on his end.

However, this also completely changes the amount of funds at risk. From 100% of the payment, it shrinks all the way down to a mere transaction fee for on-chain invoices, and no money at all for off-chain invoices.

This is the kind of risk profile which we can sensibly provide a new cryptocurrency user without scaring them off instantly.

2.3 Removing User Input Error As Much As Possible

Lastly we need to address user input error. With push-based systems we run into the problem that everything needs to be perfect on the first go.

With an invoice based system, one of the beauties is that both sides of transaction have the ability to double-check all of the inputted data before the actual fulfillment payment is sent. The invoice creator first checks before sending, and then the payer also checks before fulfilling the invoice.

Furthermore in many scenarios, b2c especially, user input error will be taken out of the equation entirely. When an online merchant's server is the creator of the invoice, the data will be inputted automatically and sent to the customer. The customer's wallet reads the invoice, provides UI for the customer to *Fulfill* or *Deny* the invoice, and nothing more. Thus the customer has the ability to double check that the invoice is indeed correct, all the while not having to worry about properly filling out the expected amount or any of the other data. Therefore we improve the ability for both sides of the transaction to spot user error, decrease the chance it will happen, and in many cases remove it entirely.

Thus by proving address ownership, offsetting and minimizing risk to near zero, and by removing user input error, it becomes clear why an invoice based system is far superior to our current push based model.

3. Why A New Invoice Protocol?

Now you may be asking yourself, why bother with a new invoice protocol when there have been a few projects that exist already? Wouldn't it be best to leverage off of their work rather than starting from zero?

That is a good point indeed, however no other project currently available created their system specifically targeting Transaction Surety. What this means is that these systems made numerous design decisions which while useful in their context are the equivalent of excess bloat in ours.

For a good invoice system to truly impact the blockchain space it needs a few things:

1. Be free of smart contracts
2. Be blockchain agnostic
3. Have no token or blockchain of its own

These are basic requirements that are required to ensure the protocol as useful as possible in every context.

Without smart contracts we keep as much of the computation as possible off-chain thus lowering fees, removing complexity, and making it an order of magnitude easier for wallet developers to implement.

By being blockchain agnostic an ecosystem can grow and thrive around one core standard so all of the tooling which is built around it can be reused and shared no matter the blockchain.

Having no token or blockchain of its own means that we can actually provide a top notch user experience without having to reinvent the wheel and hide all of the complexity in a multi-layered solution.

If such a system were already in use then many of the common worries and woes users have would be put to rest. Alas such is not the case, and thus you are here today reading this whitepaper to see what this *Syre* protocol is all about.

4. Syre Invoices

Syre is intended to be as lean as possible with no excess fluff. What this also means is that it is quite easier to understand at its core.

At their simplest, invoices are just two pieces of data:

Fulfillment Address	“`Ae2tdPwUPEZDfaziwNdt83thqfPKfMNSKGwQWWK4CL2o6MJDec9TrHrCbCv`”
Requested amount	5000 ada

The fulfillment address being where the money is to be sent back to, and the requested amount is how much money is to be sent.

On-chain we would append this data to a transaction and send it, while off-chain we would deliver this data through an alternate means such as sms, email, web browser api, etc. In an ideal world, the payer would receive the invoice, check it over, fulfill it, and life would go on swimmingly.

Unfortunately, that is not the reality we live in. If that was all there was to the protocol then it would be open to countless attacks by bad actors. Hearing about users fulfilling the scam invoices would be a daily occurrence. Instead of bringing Transaction Surety to users it would likely scare off numerous while causing chaos at the same time.

From this simple setup we need to end up with a protocol that can keep invoice data private, stop man-in-the-middle attacks, guarantee address ownership, and prevent foretelling attacks. The dynamics for on-chain and off-chain invoices are slightly different, but below we will go into how achieving all of this is possible.

4.1 On-chain Syre Invoices

On-chain Syre invoices have a few key distinctions from our simplistic model.

Before going in depth into the rationale behind the design decisions, let's look at what exactly our Syre on-chain invoice data looks like.

Data Identifier	"Invoice"
Requested Amount	5000 ada
Secret Message	"Ab35sd79dr4" or "Yard Work"
Invoice Fulfillment Address	"`Ae2tdPwUPEZDfaziwNdt83thqfPKfMNSKGwQWWK4CL2o6MJDec9TrHrCbCv`"

We only added two simple fields, being the data identifier and the secret message. The former is merely to provide context to the receiving wallet for what the data at hand is. The latter is important for preventing foretelling attacks, while also providing a field to add in extra information about the invoice.

All of this invoice data is also encrypted with the payer's public key before being sent.

Let's go through the rationale of how this new invoice design addresses the issues we mentioned earlier.

Proving Address Ownership
One of the benefits of on-chain invoices is the fact that if an invoice arrives at a user's wallet, and the fulfillment address matches the sending address, then address ownership is verified.

There are no further checks required, as we leverage the blockchain at hand to do the verification for us.

Man-in-the-Middle Attacks
On-chain the MitM attack that we need to specifically tackle is the Replay attack.

If we simply append unencrypted invoice data to a transaction, it is trivial for a bad actor to scan the mempool, find our pending invoice, and then replay it to the payer themselves. If the payer does not specifically know the invoice sender's address (which will be common especially in UTXO-based blockchains) then depending on which pending tx confirms first, the payer may fulfill the wrong invoice and lose their money.

To prevent this the data that is appended to a transaction is encrypted with the payer's public key. What this means is that no other entity, other than the private key owner (which in the majority of cases will be solely the payer), is able to see any of the invoice data. This both provides privacy and acts as the first addition to our initial protocol to aid in preventing attacks of all kinds.

However, this doesn't fix everything. Bad actors may still be able to figure out which pending transactions are Syre invoices based on their size and the fact that encrypted data is appended onto them.

On the plus side, now that the invoice data is encrypted the adversary has no clue what the value of the invoice is. This means they will be left shooting in the dark trying to replicate it. However this still leaves open the possibility for successful attacks if the bad actor can foretell invoice values and times sending them appropriately making it non-trivial for the payer to figure out which invoice is indeed the correct one.

Foretelling Attacks

Thus we need a solution for scenarios where an adversary can foretell future invoice values.

An example of this could be any system/service which uses on-chain invoices along with predetermined fees/costs. It could be quite easy to extrapolate when/how much a given user would be asked to pay based on actions they had taken previously due to the fact it is all visible publicly on the blockchain.

To solve this we also add a secret message in the invoice. This secret message is a string which acts as a piece of data that both the invoice sender and the payer would recognize as common knowledge, yet that a third party could never figure out & reproduce. The invoice sender fills out the secret message which the payer verifies in his wallet once the invoice arrives.

It could be a random alphanumeric string which the two parties agree upon beforehand or a message which the receiver knows only the sender could possibly think of. An example of the prior would be "432ma02s" while an example of the latter is "Yard Work April 25 2019 3pm". For many p2p transactions using an instructive secret message is very useful for both parties. This can also be reasonably secure as long as the same secret message/format does not get reused repeatedly in the future.

The potential of an adversary figuring out the correct timing, specific invoice value, and the exact secret used for a specific invoice is nigh impossible, especially if the sender/receiver are vigilant.

4.2 Off-chain Syre Invoices

Off-chain invoices are invoices which are delivered via a medium that isn't a Blockchain.

Possibilities include:

- Email
- Text
- Web browser
- Intercommunication between mobile apps
- USB Drive
- Etc.

This opens the door to a whole new world of possibilities, where an invoice can be delivered to a user at 0 cost through one's preferred messaging medium. As the invoice is in essence an encrypted blob of data it does not matter how it arrives at the payer's wallet but just that it does.

This can be leveraged in a mobile app or web environment to provide instantaneous and seamless invoice delivery straight to a user's wallet. At the mere click of a button, an invoice will be crafted and sent off-chain to the web or mobile wallet via relevant API calls, thus streamlining the process immensely. This is optimal for many scenarios, one example being online ecommerce scenarios.

Off-chain invoices can also be user-generated, exported, and sent via their favorite chat application, sms, email, etc to people they know.

These invoices provide all of the benefits of invoice transactions, but leverage alternate transfer channels to cut down on transaction fees and remove confirmation times.

Let's go into what is unique about off-chain Syre invoices.

Data Identifier	"Invoice"
Requested amount	5000 ada
Secret Message	"Ab35sd79dr4" or "Yard Work"
Invoice Fulfillment Address	"`Ae2tdPwUPEZDfaziwNdt83thqfPKfMNSKGwQWWK4CL2o6MJDec9TrHrCbCv`"
Signature	"…"

Proving Address Ownership
As off-chain invoices do not use transactions, there is no inherent proof that the fulfillment address is owned/controlled by the sender. Therefore we must bring in an extra piece of data to fill in this role, a signature.

The invoice sender must sign the rest of the invoice data, and then this signature is added at the end with the other data before it is all encrypted with the payer's public key.

This signature will be checkable by the invoice receiver's wallet to see if it was signed by the fulfillment address, and if so, it means that the invoice sender does indeed have control of said address. Thus we have proved address ownership.

Man-in-the-Middle Attacks
Given that we are not typically broadcasting off-chain invoices publicly, we have a smaller attack surface to worry about for MitM attacks.

Since the invoice data is encrypted with the payer's public key before being sent, the vast majority of MitM attacks should be prevented.

Furthermore, given that the invoice sender signs the rest of the data in the invoice, this also means that there is no way for an adversary to change the invoice data after the signature is already made. It would require changing the data and then requesting the new data to be signed by the sender once again. Unless the bad actor has total control, this will not happen.

Between the signature assuring no invoice data modifications and the encryption ensuring privacy, Syre off-chain invoices have a solid defense against MitM attacks.

Foretelling Attacks
Though the chance that an attacker will have access/the ability to send off-chain invoices is slim due to the more private nature of the interaction, it is still potentially possible.

As such the secret message is still included and used to guarantee that no foretelling attacks are possible.

4.3 On-Chain Vs. Off-Chain Invoices Chart

	Tx Invoices (On-Chain)	Off-chain Invoices
Tx Feeless		
Instantaneous		Depends On Delivery Mechanism
Universal Interoperability		
Self-User Created Invoices		
No Additional Delivery Mech.		

5. Understanding UX & Wallet UI

Let’s get into what the wallet UI and general user experience would be like.

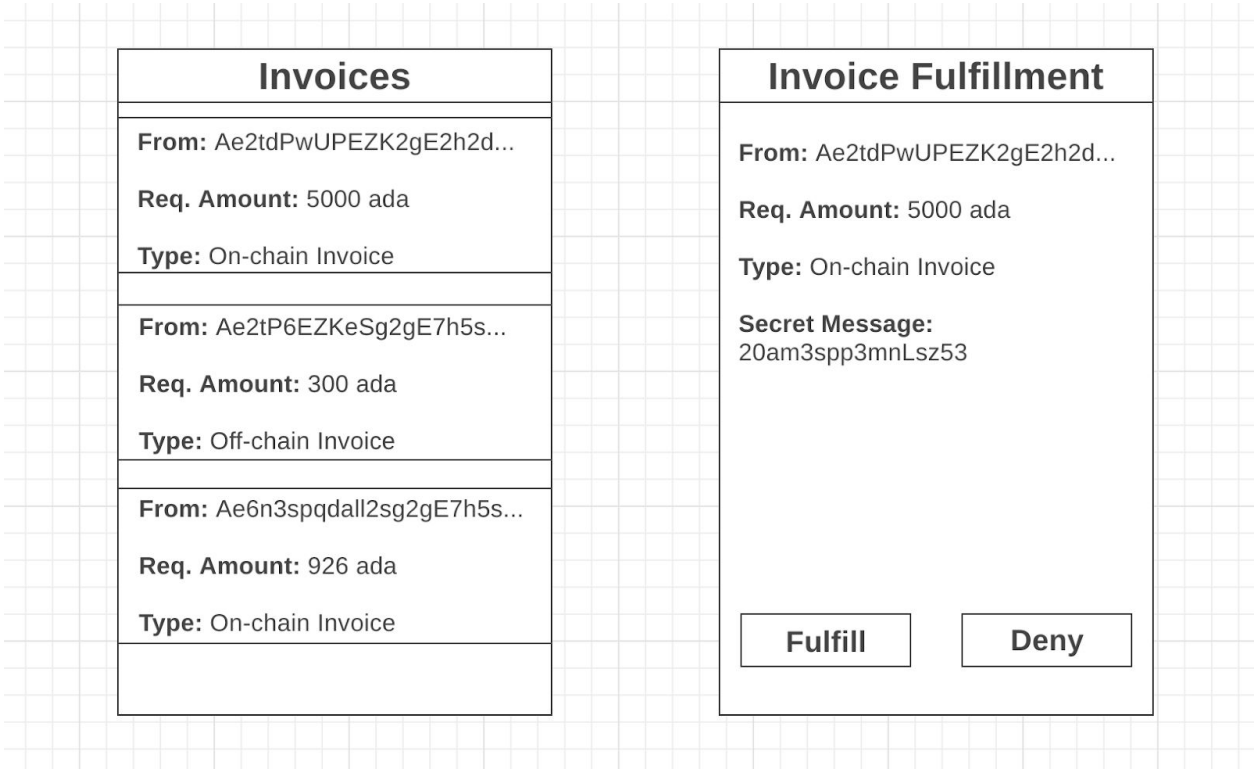
5.1 Invoice Fulfillment UI

A typical wallet interface would provide an “Invoices” section with a list of invoices received. While listed, invoices show a blurb of what is inside, with the ability to click to expand and show all of the specific invoice data.

Once expanded the invoice displays the:

- Fulfillment address
- Requested amount
- Invoice Type (On-chain vs Off-chain)
- Secret Message

From there the user has the ability to *Fulfill* the invoice, or *Deny*. The very simplistic wireframe below provides a visual example for how this UI could be potentially implemented.



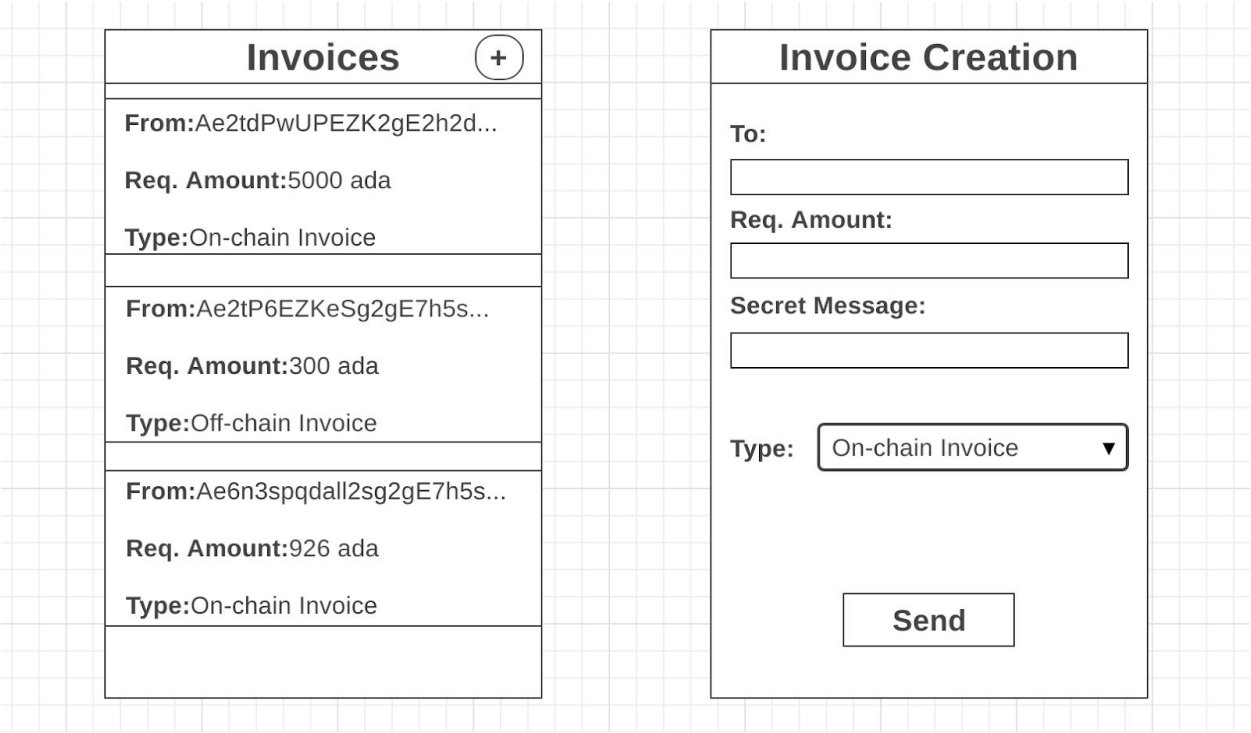
5.2 Invoice Creation UI

Just as in our previous example, we start off with our list of invoices, however this time we also add a (+) button which provides the user with the ability to create their own invoice.

The wallet creation interface asks the user for:

- Address To Send Invoice To
- Amount To Request
- Secret Message
- Invoice Type

Once the user has input the data, they are able to click *Send*. The simplistic wireframe below provides an example of how this UI could be implemented.



One thing to note is that for off-chain invoices the wallet should provide an easy way for the user to take the encrypted invoice data and send it via their favorite transfer medium. Good integration into email, sms, chat applications, and the like make the experience that much more smooth and enjoyable for the user.

5.3 On-Chain Syre Invoice Example

In this example we will cover a very simple peer to peer situation that leverages on-chain Syre invoices. We will go into what is required of each party, as well as what the wallet automatically does to get an idea of how this is supposed to work.

Alice performed some yard work for Bob for which they agreed upon a price of 500 ada. Alice now wishes to claim her pay.

1. Alice acquires Bob's address, and fills out an on-chain invoice transaction to him with the following data:
    a. 500 ada (Requested amount)
    b. Bob's address (Payer's address)
    c. "Yard Work April 25 2019 3pm" (Secret Message)
2. Alice verifies what she input is correct, and her wallet automatically appends the following data before encrypting it all with Bob's public key:
    a. Alice's address (Fulfillment Address)
    b. "Invoice" (The Data Identifier)
3. The invoice is then sent to Bob via the blockchain.
4. Bob's wallet receives the invoice and decrypts the data with Bob's private key.
5. Bob's wallet then verifies that the fulfillment address is equal to the address the transaction was sent from. If not, it is likely a replay attack, and thus the wallet throws that invoice away.
6. If they were equal, then Bob's wallet displays the invoice in the "Invoices" section of the UI.
7. Bob finds the invoice in his wallet software, checks that everything is in order & that he recognizes the secret message.
8. Bob presses "Fulfill" and his wallet automatically generates a payment back to Alice using the invoice data.
9. Bob confirms the auto-generated invoice fulfillment transaction back to Alice.
10. Alice receives her money.

5.4 Off-Chain Syre Invoice Example

A customer wishes to make a purchase on an online store using ada and their web-wallet.

1. The customer adds items to their cart and goes to checkout.
2. On the checkout page the merchant's server generates a random secret message and displays it.
3. The customer clicks on a button which requests an invoice (background API call get the customer's address/public key from their web-wallet)

4. The merchant's server fills out an off-chain invoice with total cost, fulfillment address, secret message, and a signature of the rest of the off-chain invoice data.
5. All of this off-chain invoice data is then encrypted with the customer's public key.
6. The encrypted off-chain invoice is sent to the customer's web-wallet via API.
7. The customer's wallet decrypts the off-chain invoice and uses the signature/rest of the data to verify that the fulfillment address is equal to the address which provided the signature.
8. The customer is then shown the invoice along with the secret message.
9. The customer checks that the secret message on the webpage is equal to the secret message in the invoice.
10. The customer fulfills the invoice.
11. The merchant receives the money finishing the checkout process.

5.5 Whitelists

Since we are already on the topic of user experience, it is also worth thinking about how to improve the process for common but not universal situations. The one we will tackle right here is how to streamline & provide greater security when constantly receiving invoices from the same person/entity.

As we mentioned earlier foretelling attacks are an issue especially when you have recurring invoices sent on-chain at specific intervals. However this is only an issue if you do not know where the invoice is coming from. Typically we use a secret message to act as a form of common knowledge. In essence it acts as an authorization mechanism.

But what if instead of the human acting as the checker, we offload this checking process to the software? Whitelists precisely do this.

By adding an address book into the wallet which acts as a whitelist, it is possible to use it as a filter to only receive invoices from trusted individuals. Optimally filtering out all non-whitelisted addresses would be a setting which one could turn on/off in the wallet depending on one's usecase & desire for security. For account-based systems this removes the need for relying on the secret message entirely.

For UTXO-based systems which constantly generate new addresses however, the whitelist must rely on the secret messages rather than the addresses themselves. Thus one would have a *"Secret Message Whitelist"* within their wallet that they append unique strings to. The invoice sender then is provided with and saves one of the whitelisted secret messages to send trusted invoices to the payer from whatever address they wish.

Thus we can improve the experience for both account and UTXO-based systems using whitelists.

6. Syre Extensions

While the Syre protocol provides great functionality from the get-go, it was designed to be as lean as possible to limit transaction fees and to make implementation easy for wallet developers. What this also means is that there is further functionality possible, which while not required for the core protocol to work, may indeed greatly enhance the user experience. Some of these may even open doors to entire new experiences which were never possible before.

Syre Extensions are just that. They are additions to the minimum fields required in Syre which allows for unique types of data to be included. These are not required for the protocol to work, but may greatly enhance the experience for users.

It is up to the wallet to support a given extension and provide the needed extra UI to take advantage of it. Ideally most general use extensions should only include data that is informative. This means that they should not cause wildly different circumstances to arise in a wallet that does not support said field extension compared to one that does. This may not be the case for very niche extensions for specific use cases where everyone is guaranteed to be using a wallet supporting said niche extension, but otherwise it is a good call.

If an extension is not supported by a wallet, the best default action a wallet should take is to let the user know that the invoice is using some unsupported field extensions, and then provide the ability to examine the extra fields/data. Providing an extra warning/confirmation screen to the user when fulfilling an invoice that has unrecognized extensions is recommended.

Extensions are optimal for off-chain invoices due to the fact that most messaging mechanisms are essentially free. This means that even if your invoice multiplies 10 times in size compared to a vanilla Syre invoice, there really isn't anything to worry about. However for on-chain invoices limiting the number of extensions to as close to as 0 is ideal. Every extra bit of data will increase transaction fee costs.

Since there are a lot of possibilities for extensions, and some focused on very narrow use cases, over time standards will naturally develop for the most useful day-to-day extensions that every wallet should expect to support. Listed below are some of the most bare-bones basic extensions that will find great use.

6.1 Basic Extensions:

Expiration Date/Time: A string which acts as an expiration date/time to let the payer know if the fulfillment is time-sensitive. This can be useful in a variety of scenarios, and wallets can even automate removing invoices from being shown after they have expired.

Currency: A string that defines which cryptocurrency is being requested. This is primarily useful for off-chain invoices which may be imported into a multi-currency wallet. In such cases it can be crucial to identify if a person is requesting ETC instead of ETH for example.

Description: A string which provides information about the invoice and whatever else the invoice creator intends to convey.

6.2 Syre Extension Usecase Example

Using the *Expiration Date/Time* extension completely transforms the current online ecommerce experience when paying with cryptocurrencies.

At the moment, the user is provided with an address and is shown a ticking timer typically ranging from 15-30 minutes. The user is expected to submit their payment transaction within this window, and it is on them if they are too slow and send it too late.

This is horrible user experience where the systems in place feel like they are haphazardly glued together without any real integration. The user is expected to handle all the minute details, which is absurd as the technology at hand is better suited to do this with an order of magnitude more reliability as well.

With the *Expiration Date/Time* extension, the countdown timer actually transfers with the invoice into the user's wallet and is integrated into the payment experience. The invoice UI itself can show a ticking timer (in the case that the invoice expires in less than 1 hour) while also deleting the invoice + stopping the user from fulfilling the invoice if it has already expired.

Thus instead of relying on the human to manage all of this, we use good design and engineering to provide enhanced safety while streamlining the process significantly.

7. Conclusion

We are at a good point in the cryptocurrency space at this moment to make serious improvements to the core infrastructure and fix the obvious usability issues. We've been diving deep into building the most complex systems which aim at changing the world, yet neglect to lay the needed foundations required for mass adoption to be a reality.

Syre isn't a revolutionary idea which will take down all the banks, replace all current-day infrastructure, or take you to the moon. What Syre is on the other hand is a genuine solution to a problem we've all experienced yet never wanted to admit aloud. Sending cryptocurrencies today, to put it bluntly, sucks. Unless we improve it we will endlessly spin our wheels with little to show for it. Syre is one decent-sized step to remedy the situation at hand and enable the space to move forward.

If you wish to stay up to date with the latest news about Syre, visit Syre.io and sign up to the newsletter.

If you are interested in implementing Syre in your wallet, have questions, or just wish to get in contact, feel free to send an email to robk@syre.io.

Syre was created through the dLab Emurgo Fellowship Program, and as such I would like to thank them for their support and dedication to investing in projects that they truly believe can make a serious difference to the space at large.