-- draft --

Fermat

A Peer-to-Peer Financial Application Framework

Contributors: Luis Fernando Molina

Advisors:

Reviewers:

Noviembre 2014

www.fermat.org

Abstract

A peer-to-peer financial application framework could allow standalone crypto wallets to evolve into any kind of peer-to-peer financial applications.

Developing peer-to-peer financial applications is challenging. Crypto networks provide part of the solution as a system of electronic cash, but the main benefits are lost if a trusted third party is still required to transport meta-data, synchronize devices, hold wallets files or keys, manage identities, interface crypto networks or the legacy financial system.

We propose a peer-to-peer network for transporting meta-data and inter-connect network clients between each other. A synchronization scheme running on top

of it transform a standalone app into a distributed application across several devices still owned by the same user.

We propose a framework to replace the standalone wallet application. This framework handles the full stack on top of crypto networks up to the user interface. In this way we enable the development of peer-to-peer financial applications that are both crypto-currency and digital-asset-enabled, and that does not require a trusted third party of any sort.

Introduction

Standalone bitcoin wallets were the first generation of trust-less financial applications since they didn't require to trust any third party, inheriting this property from the bitcoin network itself. As the ecosystem evolved, trusted third parties were introduced again and they took over the wallet space because of technical capabilities that are easier to build in a centralized way: communication between wallets, synchronization between devices, interfacing the legacy financial system, securing funds, etc., and they consistently took the biggest share of funding, leaving standalone wallets far behind and at the same time trashing the benefit of bitcoin of not relying on trust, one of its key features. Applications trying to use the blockchain to transport meta-data were considered spammers and standalone wallets were effectively left behind.

What is needed on top of all existing protocols is a layer that faces the end user and that finishes the job bitcoin started respecting its core principles of openness, decentralization and privacy. Using crypto networks for transporting value or as a registry for digital assets and the Fermat Network for transporting the required meta-data at a network client level, allows financial apps to run any

user-level interconnected-functionality without ever going through a trusted third party.

By choosing a plug-in architecture for the Framework we make it possible for any developer to add their own reusable components. We define micro-use-licensing-scheme as the mechanism for plug-in developers to monetize their work. The Framework itself enforces these micro-use-licenses and guarantees developers a revenue stream.

OS dependent GUI components are built on top of the multi-layered plug-in structure to face the end user as wallets or financial applications in general. Apps and wallets with similar functionality are wrapped into what we call *platforms*, each one introducing new plug-ins, to the ever increasing functionality of the whole system.

A built-in *wallet-factory* allows developers to reuse the highest level components and create niche-wallets or niche-financial-apps by combining existing functionality and adding their own code to the combo. A built-in *wallet-editor* allows non-developers to reuse any of these niche-wallets to build new branded-wallets just by changing their look and feel. A built-in *p2p-wallet-store* allows end users to choose which wallets or financial apps to install from the ever growing catalogue.

Framework

The solution we propose begins with a Framework that must be portable into different OS. On a multi-layered format, the bottom most layer is interfacing the OS and must be built with replaceable components implementing the same set of public interfaces in order to build on top a single set of OS-independent

components. At the same time, the upper most layers are again OS-dependent, providing a native GUI on each device.

We identify 3 different kind of components that we arbitrarily call **Add-ons**, **Plug-ins**, and **GUI** components. We define Add-ons as low level components that do not need to identify themselves to consume services from other components. They have broad access to the file system and databases. Plug-ins have their own identity and must identify themselves to other components to use their services which in return restrict the scope of their services based on the caller's identity (for example the filesystem add-on would only give access to the Plug-ins own folder structure, the database system add-on would only give access to the plugins own databases, and so on). In this way we handle the problem of plug-ins accessing the infromation of other plug-ins.

The core framework is in charge of initializing Add-ons and Plug-ins and managing Plug-ins identities. An internal API library defines the public interfaces that each component exposes to the rest of the components within the same device in order to allow them to use their services locally. This provides a strong encapsulation of each components business logic allowing them to freely define their internal structure and data model.

Crypto Networks

A set of Plug-ins is needed for each crypto network to be supported. One for interfacing the network, pushing outgoing transactions and monitoring incoming transactions. Another couple being the digital vaults where the crypto currency value and digital assets are stored.

Wallets are higher level abstractions and have their own set of Plug-ins to keep each kind of accounting. This means that we split the accounting from the handling of the value by having components on different layers to handle each activity.

Fermat Network

The network is intended for two main reasons:

a. For network clients to find other network clients.

b. For network clients to call other network clients.

Every network node has a copy of a distributed geo-localized inventory of all network nodes. They run a protocol that allows them to keep their copy synchronized.

An **actor** is a type of role an end user might play and in this context represents one of the end users identities.

To be able to be found, a network client checks itself and its actors in with the geographically closest node. We call this the *Home Node*. When an actor needs to find another actor, it follows a protocol that requires the approximate location of the actor it's looking to connect to. This is possible because the system deals with end users and usually people know the city, state or country where the people they know live. In this way we avoid using phone numbers, emails or any other possible personal identifier that could compromise privacy.

For scalability reasons, we moved the responsibility of finding actors within the network, from the network itself to the network clients. Each node has a distributed catalog of Actors and is responsible for keeping it updated. Instead

of every node having the full catalog, only a set of nodes nearby the *Home Node* of an actor knows that actor and which node is its home. This allows any actors who know the approximate location of the actors they are searching for to easily find, trace and follow them to their current *Home Node* where they are checked in, allowing them to stablish a connection between themselves.

Incentive

For developers

Plug-in developers declare a *Micro-Use-License* for each plug-in they add to the Framework. Wallet or Financial Apps developers declare a *Micro-Use-License* for their components. end users install the Apps (wallets) of their choice. The license to be paid is the summary of the Apps *Micro-Use-License* plus all the *Micro-Use-Licenses* of the plug-ins used by that App.

The Framework is responsible to charge the end user and distribute the payments to all developers involved.

For network nodes

Network clients establish a *Home Node* where they check themselves and their actors in so as to be found by other network clients. They must pay a subscription fee to their *Home Node* for its services. Finding and calling other clients through other nodes is free for the caller. The nodes income is covered by those network clients for whom they act as their *Home Node*.

Platforms

We define as *Platform* a set of interrelated functionality. *Platforms* may consume services from other *platforms* and their dependencies form a hierarchical stack.

Each *Platform* may introduce new workflows to the system , Add-ons, Plug-ins, GUI components (Apps, wallets) and Actors. This enables the system to target different use cases with different actors involved.

Identities

We handle identities at different levels for multiple reasons. In all cases, identities are represented by private and public keys.

End User Identities

The need to handle multiple logins on the same device brings with it, the first kind of identity which we call *device-user*. This identity lives only at a certain device and not even a public key is exposed to the network.

Besides, the end users can have multiple types of identities (we call this *Actors*), and within each type as many instances as they want. Each type of identity corresponds to a role in real life or an actor in a Use Case. Usually each Platform introduces a set of actors and all the Platforms functionality orbits around all the use cases derived on the interactions between those actors.

The Framework handles a hierarchy of identities. One of them is what we call the *root identity*. At root level end users can set a standard set of information that can be overwritten at any level down the hierarchy, narrowing or expanding

that information as needed. All these identities are exposed to the Fermat Network in a way that from the outside, no one could tell they are related between each other or to a certain end user.

Component Identities

Many components have their identities for a variety of reasons:

a. Plug-ins to identify themselves to Add-ons in order to get access to identity-specific resources, such as Databases or their own share of the File System.

b. *Network Services* to encrypt the communication between each other.

c. Network Clients to encrypt the communication with nodes.

d. Nodes to recognize each other even when their IP location or other profile information changes.

Workflows

We define workflows as high level processes that requires several components to achieve a certain goal. Manny workflows start at a GUI component triggered by the end user and spans through several Plug-ins on the same device, and in some cases jumping into other devices. Other workflows may start at some Plug-ins, triggered by events happening within the same device.

From a workflow point of view, each Plug-in runs a task and is fully responsible for doing its job. Workflows are a chain of tasks that may split into several paths and may span through more than one device.

In some cases workflows interconnect with each other, forming a *workflow chain* that usually spans more than one*Platform*.

Transactions

Transactional Workflows

As the Framework runs on potentially unstable devices such as mobile phones, each Plug-in must be prepared to overcome the difficulties caused by a device shutting down at any moment and it must be able to complete its intended job later and never leave information on an inconsistent state. This is quite challenging but not impossible.

The solution is to make each Plug-in responsible for the workflow while they are handling part of a transaction on a transactional workflow. This responsibility is transferred to each step of the chain using what we call a *Responsibility Transfer Protocol*. This means that the component that is responsible at the moment of a black out is the one that must resume and do its best to get rid of that responsibility moving it further down the chain within the transactional workflow.

Value Transactions

We handle monetary and digital assets transactions dividing the accounting from the value. Usually transactions start on specialized Plug-ins which are in charge of coordinating the whole transaction. These Plug-ins usually interact with wallets-Plug-ins debiting or crediting the accounts involved. The accounting of the currency or digital asset involved are kept by these wallet-Plug-ins. Later the transactional workflow splits between moving the value (usually crypto currency) and moving the meta-data associated to the transaction.

Through two different paths, the value and the meta-data arrives to the recipients device and they are combined together by the remote counter-party transaction

component which in return interacts with the remote wallet-Plug-in to record the accounting as appropriate.

Synchronization

We define a Private Device Network as a network of devices owned by the same end user. Using the Fermat Network, the Framework synchronizes the information on all nodes of the Private Network. By doing so the information and system identities belonging to the end user are available at any device.

Crypto funds are kept into a Multi-Sig vaults and there they are shared, making *Petty-Cash-Vault* accessible from all nodes even when they are offline from this Private Network. An automated process monitors the Petty-Cash-Vault and tops it up when needed. Several nodes must sign the top transaction in order to proceed. This way if a device is lost or stolen, only the Petty-Cash fund is at risk. End users can eject stolen devices from its Private Network and if they act quickly they might be on time to re-create the Petty-Cash fund under the new configuration and be able to save those funds.

User Interface

The Framework handles a stack of layers. Starting from the bottom we have the *OS API level*, then the *Blockchain Level*, the *Communication Level*, *Platform Level* and the *User Interface Level*. With the goal in mind for allowing even non-developers to deploy their own peer-to-peer financial applications, we define several concepts:

Wallet: Any kind of financial application that handles either crypto or digital assets for any purpose.

Reference Wallet: A primitive wallet that is used by a single actor for a handful of use cases.

Niche Wallet: A combination of several *Reference Wallets* into a single product with its own look and feel and possibly extra functionality.

Branded Wallet: A *niche wallet* turned into a new product owned by a different end user. Achieved by a process similar to building a Wordpress site but locally, on the end users device. Usually it involves re-using the business logic of the*niche wallet* it derives from and adding a new look and feel (different skin and navigation structure).

External Wallet: A third party APP running on the same device that uses Fermat as a backend for different reasons. For example to benefit from its infrastructure to interface crypto networks, transporting data through its p2p network, or storing data on the end users *Private Device Network*.

Several tools were designed with the purpose of enabling the development of new wallets, an their distribution.

Wallet Factory: Is a built-in functionality that enables the development of reference and niche wallets.

Wallet Editor: Enables the creation by non-developers of *Branded Wallets* based on any one of the *Niche Wallets*available.

Wallet Store: Is a distributed application which manages a shared wallet catalog and enables the end user to download from peers the different wallets available for the Framework.

Privacy

The proposed system complements the privacy properties of crypto networks, extending them to the full stack needed to run different kind of financial applications. By using its own P2P network with point to point encryption for transporting meta-data both value and information are under a similar privacy standard.

Identities are public keys related to private keys kept by the end user and never shared to anyone in any way.

The collection of system information for visualization and statistics uses hashes of public keys to protect end users privacy and at the same time preserve the relationships between them.

Conclusion

We have proposed a Framework for developing and running peer-to-peer financial applications. The Fermat Framework shows the way of how to keep the end user away from trusted third parties at a higher level. We propose a solution to several problems at the same time. The highlights of our work are:

- How to exchange meta-data in a peer-to-peer way
- How to prevent the loss of private keys (funds and identities)
- How to maximize reusability by building with Plug-ins
- How to enable even non-developers to create and use their own wallets and financial applications.

With this system we allow for a new ecosystem of peer-to-peer financial applications that are both crypto and digital asset enabled.

References

[1] Satoshi Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System", https://bitcoin.org/bitcoin.pdf, 2008