Fireblock in depth review

version **1.9.0**

Laurent MALLET, Régis PIETRASZEWSKI, dev@fireblock.io

- Official Website
- Github repository

Trust is a scarcity in our global economy.

Day to day we use and exchange huge quantities of files of all kind, PDF, contracts, images   but also packages downloaded from the internet. We call them digital assets as they carry out decent amount of work from their authors.

Lack of trust leads organizations and individuals to allocate lots of resources to audit and verify records, integrity and ownership of these assets. Even if lots of automation can be used there, it's still very difficult to verify because computational proofs can be easily tampered. Some would use certificate authorities, but it's still delegating the burden of the proof to fragmented independent third-parties and it comes at a high cost.

A user friendly tool had to be designed to certify and verify basic informations.

Fireblock helps distribute digital assets trustfully by offering an easy way to prove integrity and ownership.

We designed it free of charge for everyone.

It goes through a certification process where the author links his social identities to his assets. This process is tamper-proof and cryptographically signed by the author. The certificate is stored in the blockchain.

The technical background is a decentralized public key infrastructure implemented in an Ethereum smart contract.

We provide a complete SDK.

In this paper, we describe how we designed Fireblock.

What is Fireblock?

Fireblock allows distributed files to be trusted in their integrity and ownership by any of their users.

In order to do this, we need:

- To allow any author to create a trusted checkable certificate of ownership.
- To allow any author to provide his users an easy tool to verify origin and ownership of his files i.e his certificate.

A file's certificate of ownership contains:

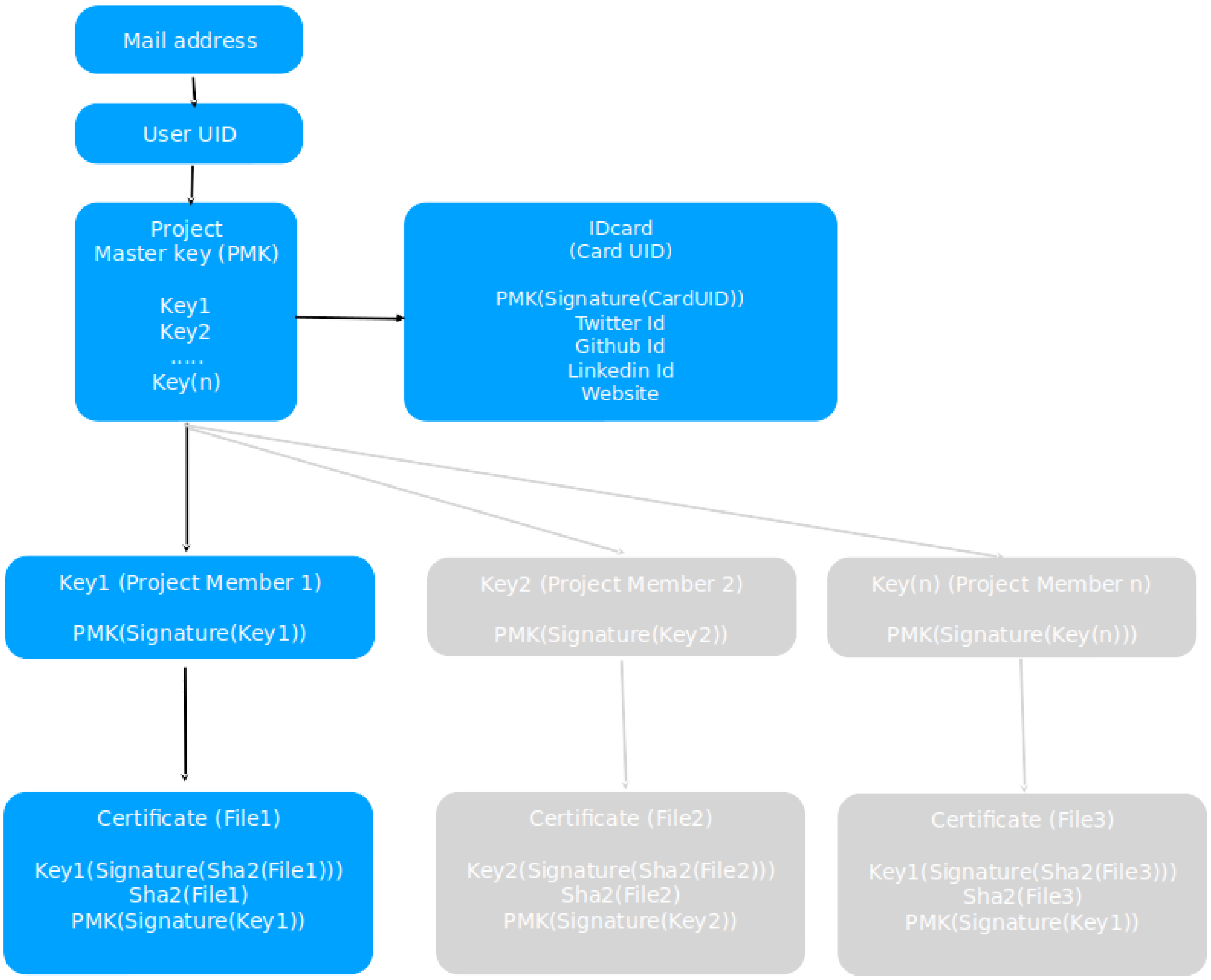
- A proven integrity of the file
- A proven and verifiable identity
- A proven link between the file and the identity
- A timestamp
- Tamper-proof properties

We design integrity with hash functions and ownership with digital signatures and proven social identities. We package them together in a certificate. We detail here how we designed Fireblock.

Fireblock structure overview

Here is a simple graph of how we designed Fireblock. We assume that a user will work on various different projects with potentially many members in each project.

The graph below show the structure of one project.



We will now dive into the structure of each element.

1.1.Project

At the top level, when registering, user creates a Fireblock user unique identifier (UserUID). Nested under, he will create projects (P1, P2,...,Pn). Indeed, we assume that a user will work with other users on various projects. We do need to organize workflow efficiently.

A project P is an object in which there is:

- A Project Master Key (PMK): an ECDSA key locally generated and managed by the user account
- Delegated Keys (DK) and Members Keys (MK): ECDSA, PGP or Ethereum digital keys
- A link to the Project IDcard

1.1.1.Project Master Key

Each project has a Project Master Key (PMK): an ECDSA key generated locally and managed by the UserUID.
This key is used to:

- sign the IDcard project
- sign and manage delegated and members keys, hence ensuring they are validated by the UserUID.

PMK cannot be used to sign certificate, they represent the project.

Infrastructure operations:

Secure/Unsecure/Upgrade/Close/Revoke

1.1.2.Project keys

A project can have multiple keys (K). Those are used to create signed certificate.

They are cryptographically linked to the Project by two signatures of a message to acknowledge their link.

PMKsign(K)   Ksign(PMK)

Note: Those keys are generated either in the same UserUID account (delegated key) or in another account (members key).

Certificate structure.

1.How do we prove integrity?

Checksum is the most basic tool that gives a cryptographic fingerprint of any digital assets. It is widely performed, especially in business context. We use sha2-256 hash functions to prove integrity.

For d the digital asset, the checksum is defined as:

Integrity of a file is guaranteed by verifying its hash.

2.How do we handle identities?

At the top level, when registering, a user creates a Fireblock user unique identifier (UUID). Nested under, he will create Projetcts (P1, P2,...,Pn). Each Project will have its own IDcard which contains:

- Social media identifier: Twitter, Github, Linkedin,.. accounts
- Digital key: ECDSA Master Project Key fingerprint (detailed in 3)
- A post on each social media with a reference to his UUID and his digital key

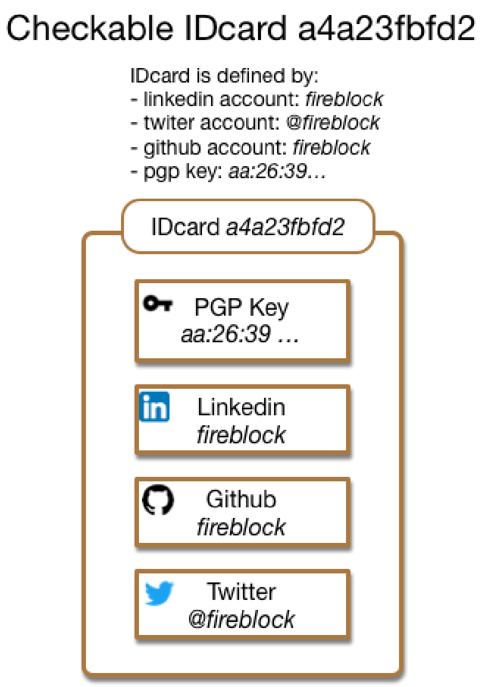
A JSON file is generated using author's provided informations. We will use sha3(IDcard) as its Unique Identifier (UID):

Independently, social identities give a good sense of who we are. Credibility can be easily checked, especially for organizations. We made them self-validated by the owner so that anyone can verify them. We parse the JSON file to check coherence.

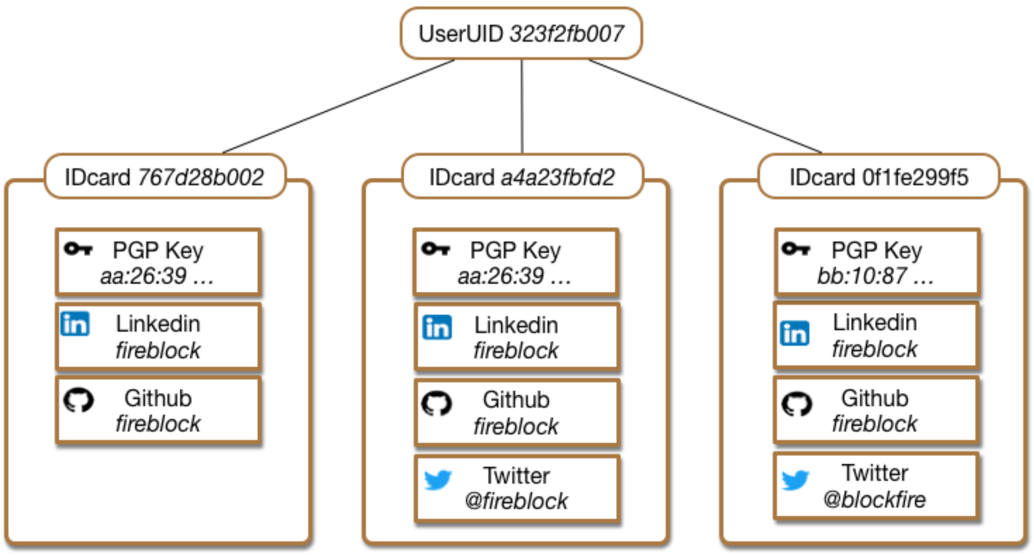
The IDcard also contains the fingerprint of the digital key to allow signatures verifications.

Project owner's identity is provided by his social identities and his project digital key (ECDSA).

Fireblock IDcard example:



One user can have multiple IDcards. Social identities and digital keys are dynamic and can be modified over time. Each Fireblock user is assigned a unique user identifier (userUID). Each user has his own list of associated IDcards with his userUID.



3.How do we prove ownership?

We use asymmetric signatures protocols, such as used in PGP and Ethereum to define property of a digital asset. Indeed, anyone signing the couple (integrity(d), IDcardUID) is de facto considered as the owner of this pair. In practice, ownership is seen at large, including that one can have had an asset at a certain point of time and claim it this way, not necessarily pretending he is the creator of this asset.

Our web client uses PGP protocol, while Ethereum addresses are prefered in our future native client.

25 years after it's launch, PGP is still a robust solution to sign a message. We made it less of a burden by driving you through the signature process in two easy steps. Signature is made in your browser with the commonly used PGP standard, without ever asking you to provide your private key. You sign a set of both hash of your digital asset and your IDcard.

PGP signature of the digital asset's fingerprint and the associated IDcard proves ownership.

sha3(IDcard): sha3 of the IDcard. Sha3 (keccak256) hash function is preferred here for computations in our smart contract.

Ownership of a digital asset is guaranteed by verifying its signature.

4. Fireblock's certificate structure:

Our certificate is a tuple defined as:

Now, we want this structure to become a proof. As it stands, it's not enough because a timestamp and a tamper-proof property is needed. The blockchain gives us a powerful tool for this.

5.Tamper-proof and timestamp

We use an Ethereum blockchain as a reliable and decentralized database that provides an efficient storage mechanism.

Ethereum is a blockchain with a built-in Turing-complete programming language, allowing anyone to write smart contracts and decentralized applications where they can create their own arbitrary rules. Ethereum blocks contain a copy of both the transaction list and the most recent state. In Ethereum, the state is made up of objects called "accounts". There are two types of accounts: externally owned accounts, controlled by private keys, and contract accounts, controlled by their contract code.

The blockchain is built by adding every 15s a block, with each block containing a timestamp, a reference to (i.e., hash of) the previous block and a copy of both the transaction list and the most recent state. Over time, this creates a persistent, ever-growing, "blockchain" that continually updates to represent the latest state of the ledger.

To store a certificate on the blockchain, we designed our own smart contract (contract account). Contracts in Ethereum should be seen as a code that live inside of the Ethereum execution environment, always executing a specific piece of code when "poked" by a message or transaction, and having direct control over their own key/value store to keep track of persistent variables.

Our contract allows to:

- Seal certificates in the blockchain
- Validate IDcard authenticity
- Verify certificates

Executing the contract goes through transactions. The term "transaction" is used to refer to the signed data package that stores a message to be sent from an externally owned account. In our actual web client, a Fireblock account is used, although in the future native client, private accounts transactions will be implemented.

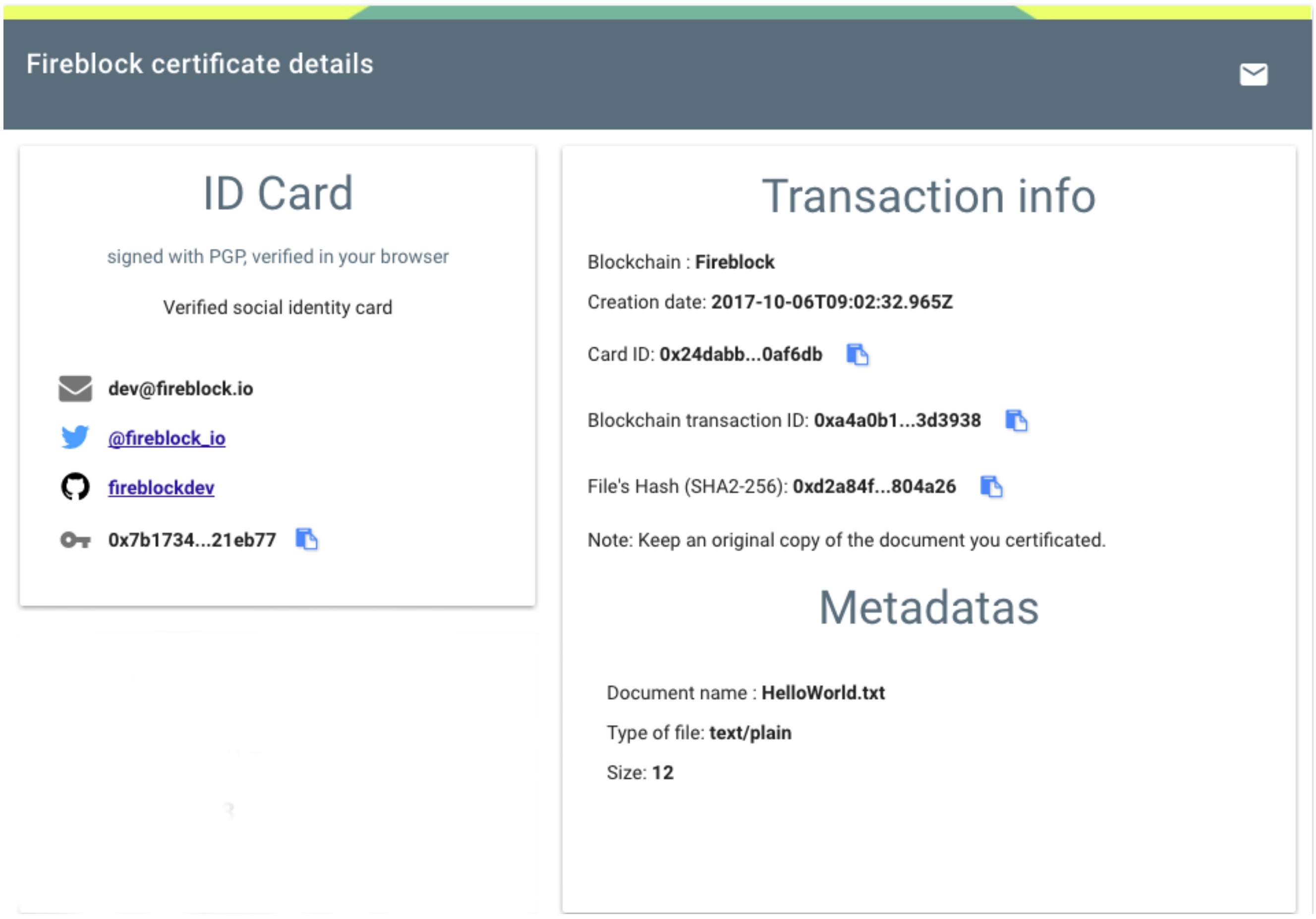
A blockchain transaction is referred in the blockchain by its unique transaction ID which gives an immutable and auditable timestamp: Tx()

Our certificate is engraved in the blockchain which is tamper-proof by design and gives full auditability thanks to its distributed inherent property.

Our Proof is a tuple defined as:

Integrity and ownership are guaranteed tamper-proof and timestamped by our blockchain.

Here you can find an example of a simple file certificated with Fireblock.io:


Fireblock certificate details

ID Card

signed with PGP, verified in your browser

Verified social identity card

dev@fireblock.io

@fireblock_io

fireblockdev

0x7b1734...21eb77

Transaction info

Blockchain : **Fireblock**

Creation date: **2017-10-06T09:02:32.965Z**

Card ID: **0x24dabb...0af6db**

Blockchain transaction ID: **0xa4a0b1...3d3938**

File's Hash (SHA2-256): **0xd2a84f...804a26**

Note: Keep an original copy of the document you certificated.

Metadatas

Document name : **HelloWorld.txt**

Type of file: **text/plain**

Size: **12**

We created a certificate that proves ownership of a digital asset, thanks to digital signature, and provenly linked to social identities and a public fingerprint.

We allow users to certify any digital assets without storing any of them on our servers. Users keep total control and do not ever upload any file to Fireblock.io.

Hence any potentially distributed or exchanged digital asset could be certificated by different users. We could have a situation where a file F is certificated either by Alice and by Bob. Fireblock do not provide ex-ante verification of the rights attached to a file. We assume users will check the IDcard provided in the certificate to judge by themselves of the legitimacy of either Alice or Bob. If Alice is legitimate, her Twitter, Github, Linkedin,.. accounts are supposed to speak for her and enforce her certificate for everyone.

It's important to understand that an IDcard and a file work as a pair.

6.Fireblock's certificate verification

We designed a verification process that goes through querying certificate's data to the blockchain.

We designed our smart contract by indexing on the blockchain our proof with the key:

By design, there is only one certificate which correspond to a pair d, IDcard.

We provide a verify function with two inputs: d, IDcardUID

```
function verify(d, IDCardUID) {
    var integrity = sha2(d);
    var key = sha3(integrity, IDCardUID)
    Proof p = Certificates[key]
    if (p) {
        // verify signature used in the certificate
        if (isSignatureValid(p, Certificates[key].fingerprint)) {
            return true
        }
    }
    return false
}
```

Our tool enables anyone to verify that a digital asset has been certificated by a given IDcard.

Fireblock in motion

Fireblock is for anyone distributing digital assets. These assets are either downloaded, transferred, exchanged by any means, but on the road, they could be altered or compromised. Users have different ways to deal with that:

Let's say that Alice is the owner of a digital asset, and Bob and John two users of Alice's asset.

1.Standard case

Alice has integrated a verify button which uses our verify function on her website.

Bob possesses Alice's asset and wants to perform security and ownership verifications. The asset could have been downloaded from anywhere on the web, including Alice's website. It could have been transfered by John who might not be trusted by Bob.

Bob knows the origin of the asset and goes to Alice's website. He clicks the verify button and selects the downloaded file. He checks if it matches.

```
function verify(d, IDCardUID) {
    var integrity = sha2(d);
    var key = sha3(integrity, IDCardUID)
    Proof p = Certificates[key]
    if (p) {
        // verify signature used in the certificate
        if (isSignatureValid(p, Certificates[key].fingerprint)) {
            return true
        }
    }
    return false
}
```

We can see that our function takes two inputs (d, IDcardUID). But there is only one button on Alice's website. We index Alice's IDcard under her user unique identifier (UserUID) in our database. Hence, Bob provides d, the digital asset, and Alice's userUID by clicking the verify button embedded on her website. Fireblock works then like an etherscan retrieving the most relevant set (d, IDcardUID) and then asking the blockchain the certificate.

Benefits:

- Integrity check
- Digital signature check
- Social identities check
- Timestamp check
- Fraud protection

Bob is now fully confident that the digital asset has been certified by Alice which is a well-known trusted counterparty with engaged social identities.

Alice build trust and care for her users and do not take the risk to compromise her brand by providing an easy checkable certificate for each of her asset.

Note that Bob do not need to trust Fireblock.io. Indeed, all certificate's data can be retrieved directly from our blockchain.

2.Search case

Alice has integrated a verify button which uses our verify function on her website.

She distributes her assets with a pre/sufix reference to her Fireblock's unique identifier (UUID) like:

```
aliceDigitalAsset_F625438765
```

Bob managed to possess this asset but do not know the origin of the file. However, he knows that it has been certificated using Fireblock. Hence, he goes to our website, clicks the search button and selects the downloaded file.

A chronological list of all the certificates associated to this digital asset shows up. The first occurrence is most probably the one created by the author of the asset. Bob finds Alice's trustworthy certificate and is now confident to use safely the digital asset.

Benefits:

- Social identities check
- Integrity check
- Signature check
- Fraud protection

We provide a front-door to verify any certificate created using our solution.

3.Worst case

Bob possess a digital asset provided by John. He doesn't trust John. He does not know the origin of the asset and there is no Fireblock reference in the name of the asset.

Unfortunately, this is the worst case, and Bob is left alone with a potentially compromised asset. He shouldn't use it, or at his own risks.

We help to create a web of trust where digital assets can be safely and arguably exchanged and traded.

Security

Our rules

We respect this chart of good behavior:

- Rule 1: our software doesn't copy or move your private key from your computer. If you don't trust our frontend, you can use your GPG tool.
- Rule 2: our software doesn't copy or move the files to the certificate. We only compute the sha2-256 value directly in your browser, without uploading any file. If you don't trust our frontend, you can use **sha256sum** to compute this value and copy it in the frontend.
- Rule 3: As a consequence of rule 2, we do not, and cannot provide any ex-ante verification of the ability of anyone to certificate a specific digital asset.
- Rule 4: We will never ask your social identities' passwords (twitter, github, google+, linkedin).
- Rule 5: Be aware that if one of these rules is bypassed, it's not our software.

If you consider your private digital key is not safe anymore, we recommend to revoke the associated IDcards.

Data Security

Digital signature

Our software uses asymmetric keys to sign and verify content:

- PGP Keys: we recommend the gnu gpg tool available on windows, OS X and Linux
- Ethereum Keys: we use these keys in our softwares to write in the blockchain
- These two technologies rely on two keys:
- Private key: Used to create signatures and must not be shared. Save it in a safe place. Do not you lose your private Key, you will not be able to recover it.
- Public key: Used to verify a signature and given to anybody. A fingerprint of a PGP Key is an UID of the PGP Public Key computed by using 160 bits of sha1(PGP Pub Key).

A user has to register a PGP public key to sign a document. This operation creates a mapping (hashtable) with the fingerprint used as key and the following structure:

- Public key: armored txt format
- Revoke flag: a boolean
- Begin: block number of registration
- End: block number of closing

We designed PGP keys with lifecycles to allow easy key rotation procedures, which is a recommended pattern in many organizations. A PGP key is considered valid between the begin and the end block. At the end of its lifecycle, the PGP key can be dropped. Anyway, if a key is compromised, you have to revoke it. To register a key, drop a key or revoke a key, the owner has to provide a signature.

ID Card

The ID Card is the core object of our software. An ID Card is a structure with a JSON text field saved in our secured database and indexed in the blockchain with its Sha3: SHA3(IDcard)

```
[
    { "uid": "323f2fb007", "provider": "fireblock", "date": "20170909" }
    { "uid": "0x99090eae43316b2ba65ec52bcd5834a3e07edb2c", "provider": "ecdsa", "date": "20170909" }
    { "uid": "fireblockproof", "provider": "twitter", "proof": 
"https://twitter.com/fireblockproof/status/918192206665736193", "date": "20170909" },
    { "uid": "fireblock", "provider": "github", "proof": 
"https://gist.github.com/fireblock/bf65e1f38d5f5c5793ae5645cfaf6ce0", "date": "20170909" }
]
```

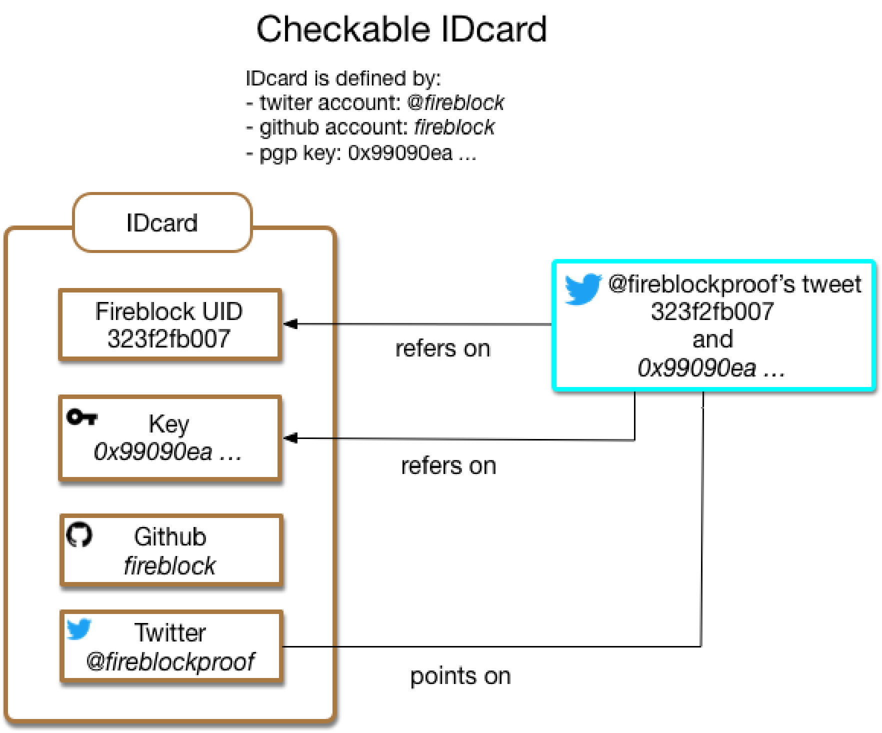
This is a JSON array of providers:

- uid: unique identifier
- provider: fireblock, twitter, github, linkedin
- proof: an url to a proof content
- date: statement date

Note: provider fireblock gives the user unique identifier (UserUID) of project owner; date is a metadata referring the IDcard creation date.
Note: the provider “ecdsa” defines a project key by its ECDSA fingerprint; date is the key creation date.

Each of the Idcard’s elements are checkable by a click.

For this example, twitter proof is checked. The proof url belongs to the twitter user “fireblockproof”. The tweet contains “`323f2fb007`” and the ECDSA fingerprint “`0x99090eae43316b2ba65ec52bcd5834a3e07edb2c`”. If there’s no tweet or a tweet associated to another user or tweet without “`323f2fb007`”, the card is considered as invalid. A verification is processed for each social identity. For a ECDSA key, the uid is the fingerprint (an UID of the ECDSA Pub Key).



In this example, creating a false ID card would need the attacker to control the twitter account, the github account and have the private ECDSA Key.

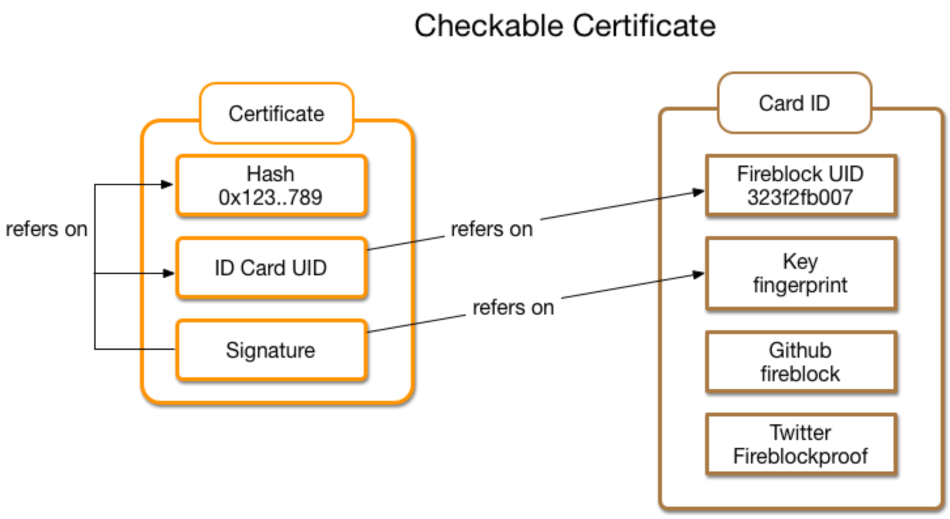
Note: if the user @fireblockproof later removes the tweet, All certificates associated to this ID card will be considered as invalid.
Note: an ID card contains one digital key.

Note: We designed an upgrade process that will allow one user to outdated an IDcard and associate an updated one with all of his certificate.

Certificate

A certificate is a structure:

- H(d): hash of the digital asset
- sha3(IDcard): a reference to the ID Card
- sign(H(d), sha3(IDcard)): a signature



To create a false certificate, you need the private key. As IDcards, certificates are checkable.

Platform Security

Our platform consist in multiple software components:

- Blockchain: our source of truth.
- Backend: a web application that provides an API for the frontend. Its role is to write the ID cards and the certificates. This role is called **writer**.
- Frontend: the web application presented to users.
- Validator: a software which makes verifications on the blockchain
- Maker: a software which realize validated operations.

Blockchain

We use a public Ethereum Proof of Authority (PoA) blockchain. Proof-of-Authority is a replacement for Proof-of-Work, it does not depend on nodes solving arbitrarily difficult mathematical problems, but instead uses a set of "authorities": nodes that are explicitly allowed to create new blocks and secure the blockchain. they are called the Sealers.
We made a fork of the clique Ethereum protocol (rinkeby) where we added one rule: only the master address can create a contract. Other addresses can only exchange currencies or use contracts deployed by the master. It allows us to focus the blockchain as a reliable decentralized storage database, and nothing else.

Backend & Validators

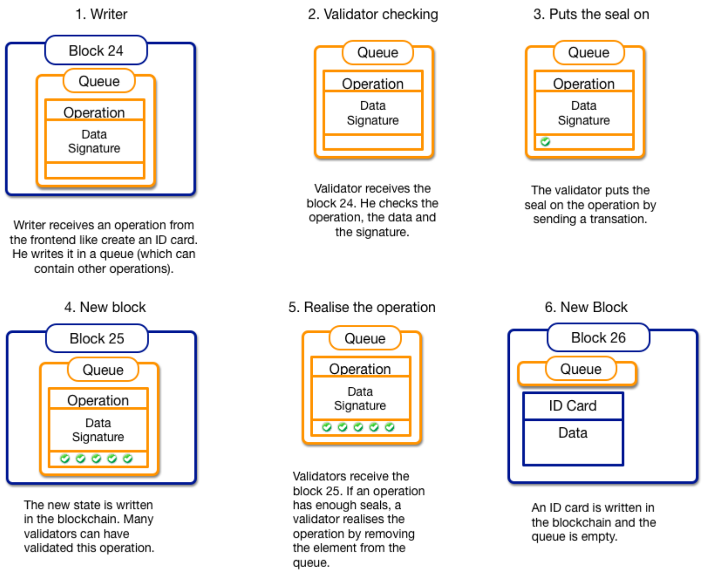
On many internet platforms, the weak point is the server deployed on a public IP. If this server is compromised, a user can impersonate everyone:

- He can use every account to register fake id cards or certificates.
- He can revoke id cards and revoke all id cards associated. A lot of work would be lost.
- He can modify our software code to execute all valid operations for the writer role on our fireblock contract.

To solve this vulnerability, we propose a mechanism similar to a Proof of Authority, but embedded in Validators nodes:

- A digital signature is asked for each operation which need safety. This operation is transmitted to the backend.
- The backend writes the operation in a queue.
- Special nodes called **validators** check the operation and the signature. If the operation is valid, they put the seal on.

- If the operation has enough green flags, the operation is valid and another special node, the **maker** can call a function to realize the operation.


1. Writer
Block 24
Queue
Operation
Data
Signature
Writer receives an operation from the frontend like create an ID card. He writes it in a queue (which can contain other operations).
2. Validator checking
Queue
Operation
Data
Signature
Validator receives the block 24. He checks the operation, the data and the signature.
3. Puts the seal on
Queue
Operation
Data
Signature
The validator puts the seal on the operation by sending a transation.
4. New block
Block 25
Queue
Operation
Data
Signature
The new state is written in the blockchain. Many validators can have validated this operation.
5. Realise the operation
Queue
Operation
Data
Signature
Validators receive the block 25. If an operation has enough seals, a validator realises the operation by removing the element from the queue.
6. New Block
Block 26
Queue
ID Card
Data
An ID card is written in the blockchain and the queue is empty.

Note: We also have checkers nodes for blockchain, they emit alerts but don't write anything. To compromise our platform, you need to compromise writers, validators and checkers. As a checker node doesn't write anything, it's quite difficult to identify them.
Note 2: Writers and validators are changed regularly; the platform is redeployed with the new writer to ensure maximum security.

Frontend security

The frontend is a SPA (Single Page Application) and consists in few important files distributed by our website:

- One html file.
- Two JS files (logic of the frontend).
- Few css.
- Images.

Validators and checkers regularly check these files on our website with our certificates and send us alerts if needed. Heavy client will do the same.

Heavy client security

Not yet implemented

Full client

Our goal is to provide a decentralized client. It uses geth, an ethereum client that will allow users to connect directly the blockchain and don't use our API or backend. Each decentralized client uses an ethereum account with a public, a private key and an address.
This client will create transactions, therefore it needs gas.

Registration

The ethereum account is created on your client. This operation can be made offline. To register this account to your ID Card, we propose to use the same protocol as rinkeby. With a gist, a tweet, a linkedin publication, you tell us your ethereum account and we send you gas for few transactions. Then, you can use the Fireblock contract to register this ethereum address in the current ID Card.

Usage

When your ethereum account is verified, you gain the "**user**" role. So you can safely use the operations allowed in the Fireblock smart-contract. No need of all verifications of the PGP mechanism.

CLI and GUI decentralized client

Fireblock team will provide two tools:
1. command line tool to integrate workflow enterprise
2. GUI tool for end user

The diploma case, an easy implementation

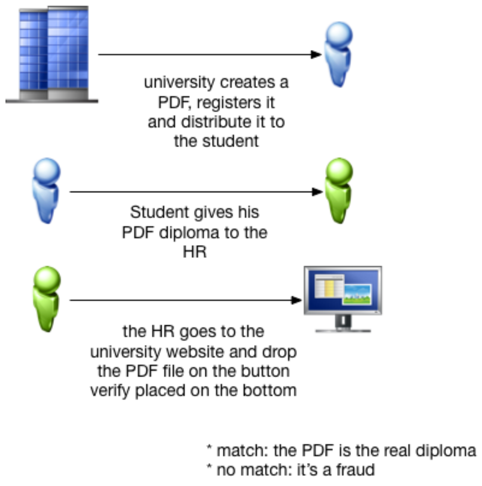
The blockchain provides a reliable tamper-proof storage for digital diplomas. Fireblock proposes an easy solution for this case. In this document, we explain how to create your own implementation in few operations.

Problem

A university wants to provide a unfalsifiable and checkable digital diploma in PDF format. The goal is to allow anyone (recruiter, companies) to check that someone claiming a grade in this particular university is effectively a graduate.

Solution

We propose the following workflow:


university creates a PDF, registers it and distribute it to the student
Student gives his PDF diploma to the HR
the HR goes to the university website and drop the PDF file on the button verify placed on the bottom
* match: the PDF is the real diploma
* no match: it's a fraud

Diploma creation

Prerequisite: you already created your diplomas as PDF files.

1. A university creates an account on Fireblock.io and associates his social identities (Twitter/Linkedin) with technical identities like (www or dns) and a PGP key to maximize security. This step is a done one time by promotion. We recommend to use one PGP key each year.
2. This university selects the PDFs diplomas and drop them on the *create certificate* box on Fireblock.io. We support multiple files registration to facilitate massive registration, and our API can also be used.
3. The university inserts a verify button on the bottom of the university site, allowing anyone having the PDF diploma to check its validity. We provide the HTML code or you can use your own designed button with the url we provided.
4. The university sends the PDF to the student.

That's it!

HR check

After a successful interview, a recruiter wants to hire one student and verify his diploma:

1. The student gives the recruiter the PDF file of his diploma.
2. The recruiter goes on the university website. He clicks on the VERIFY button and drag'n drop the file.
3. 2 results can occur:

- It's a match: a certificate has been created by the university for this file. Social identities and digital signature can be checked as well.
- No match: the diploma is probably a fraud.

That's it!

Fireblock is powerful

We understand that a diploma his granted forever. However, universities can see their social identities evolve through a lifetime.

We designed Fireblock to allow the author of a certificate to dynamically change his IDcard without the pain of losing all the previous certificates. It's designed to last.

Developer distributing digital assets

Problem

You're an independent software editor and your software is distributed through many websites. Few compromised files are distributed with ads or malwares (by modifying your package or installer). You want to provide a secure way for your users to verify the origin of the files they download.

Solution

Digital assets certificate creation

1. The developer creates an account on Fireblock.io and associates his social identities (github/twitter/reddit/website) and a PGP key to maximize the certificate's safety.
2. For each package, he creates a certificate by dropping the file on the *create certificate* button on Fireblock.io.
3. He inserts a verify button in the download sections on his website and he publicize the fact that users can check packages using this simple tool.

That's it!

User check

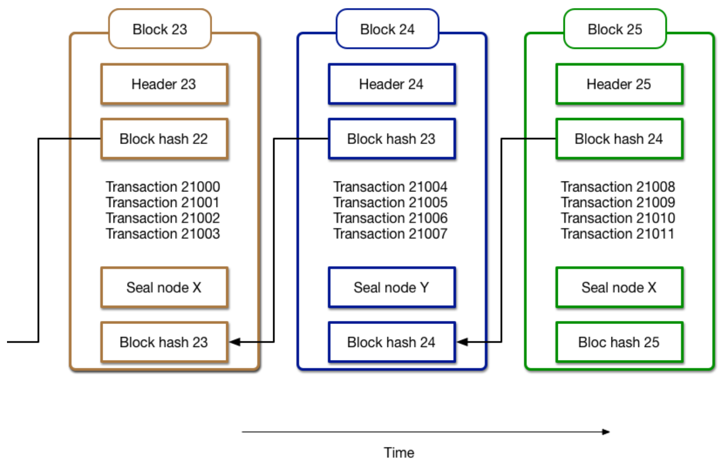
A user downloads a file on websites like CNET or from a torrent and wants to check the integrity and ownership of the file:

1. He goes to the author website or on Fireblock.io.
2. He sees a verify button and drag'n drop his downloaded file.
3. He sees a 'It's a match' and check the social identities. He knows if the file provided is the real one created by the author.

Blockchain

Proof of Authority

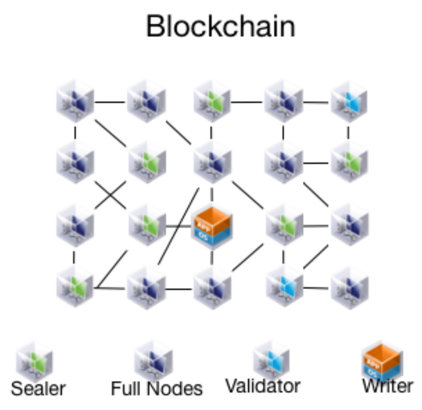
Fireblock uses an ethereum blockchain to provide its services. Each 15s, a new block with validated transactions is created:



Currently, we run our blockchain with a Proof of Authority consensus algorithm (PoA) but we will provide an implementation for the official ethereum blockchain based on Proof of Work (Proof of Stake in the future). Our PoA uses the clique protocol defined by the EIP 225 with a minor modification: only one address can create contract. Our implementation is derived from the official geth tool and our code is open source and available on github:
https://github.com/fireblock/go-ethereum

Our Proof of Authority blockchain connect different types of nodes:

1. A sealer is a node designed by the owner of the blockchain to create blocks. In a Proof of Work blockchain, they're miners.
2. Other nodes: they retrieve, check all blocks and can send transactions. They're 3 types:
   - Writer Node: He creates ID cards and certificates. The Backend pilots him.
   - Validator Node: He checks what the writer node does
   - Full Node: used by any partner to participate in our blockchain (light nodes are possible too)



We made our blockchain public, so that anybody can independently control every actions, transactions,..

Note: our PoA uses the same algorithm as the rinkeby blockchain.

smart contract

An ethereum blockchain is not only a safe data repository but also a place to define logic and behaviors. We use a tool similar to a software program called smart contract. **A Data written in a blockchain can be modified if the smart contract allows it.** Fireblock uses only one smart contract by design to ensure maximum readability and avoid vulnerabilities. Our smart contract defines our rules and cannot be bypassed. This is the core element of our security. The smart contract is written in Solidity language and is open source.

The contract is deployed on the blockchain, so it's available through an address and API. On our platform, the Backend, the Validators and the full clients uses our smart contract:

- The Backend on our server
- The Validators and full clients on computers with internet connection

To define permissions, Fireblock defines roles in its smart contract:

- Owner: the contract creator. His role is to promote or revoke administrators
- Administrator: His main role is to add, remove or revoke writers and validators.
- Writer: It's a proxy role. A writer is a role used on the Fireblock server to create ID cards and certificates.
- Validator: His role is to check what every ID cards and certificates created by the writer. This role is given to reliable partners of Fireblock.
- User: a user registered in our contract.
- Anonymous: a user of the blockchain.

The smart contract defines the following structures:

- Proof: a Fireblock certificate with hash, card ID, signature, metadata, a flag to revoke it
- Card: an ID Card (a JSON text with mail, twitter, github, linkedin entries)
- Key: a PGP key or an Ethereum address

Note: a structure is similar to a structure in C/C#/Java or Object in JS.

The smart contract define few functions:

- signWriter (hash, card, metadata, signature): create a certificate and the id card if not present (restricted by role writer)
- addKey (fingerprint, PGP Pub Key or Ethereum address): register a PGP Pub Key or an Ethereum address
- approveCard (card UID): approve a card
- approveProof (Proof UID): approve a certificate
- verify (hash, card ID): return an array [flag, flag2, signature] flag is true if it is a registered Certificate, signature is the PGP signature if it's present
- getKey (fingerprint): return the PGP or Ethereum Public key associated to the fingerprint
- getProof (hash, card ID): return a proof with its metadata, signature and if it is revoked.
- getCard (card ID): return the ID card

These functions are used on the backend by writers or by full distributed clients.

References

1. Dr Gavin Wood. Ethereum: A secure decentralized generalized transaction ledger. http://gavwood.com/paper.pdf. 2014
2. Ethereum WhitePaper.
3. Ethereum Go Github Repository
4. Peter Szilágyi. Clique PoA protocol. 2017
5. Zimmermann, Philip (1995). *The Official PGP User's Guide*. MIT Press. ISBN 0-262-74017-6.
6. http://openpgp.org/
7. Decentralized Public Key Infrastructure by (alphabetical by last name) Christopher Allen, Arthur Brock, Vitalik Buterin, Jon Callas, Duke Dorje, Christian Lundkvist, Pavel Kravchenko, Jude Nelson, Drummond Reed, Markus Sabadello, Greg Slepak, Noah Thorp, and Harlan T Wood