

How TrueProfile.io[®] is using the Ethereum Blockchain to empower users to own their data again

by René Seifert & Florian Weigand (August 2019)



TrueProfile.io[®]

Powered by The DataFlow Group

Summary

TrueProfile.io is the new standard of document verification for diplomas, employers' references, licenses and other trust-based objects from its issuing source. It puts the individual in control of safe-guarding their data and allowing it to be shared in a variety of ways with whomever they choose. As one major element, TrueProfile.io stores this verification on the Ethereum Blockchain so that customers can access proof of their verification independently of the service - even if TrueProfile.io should cease to exist. In essence, this concept similarly supports identity services both in a traditional sense and in a self-sovereign one. In a nutshell, the validated data belongs to the user and not to the company validating it.

The basic concept of TrueProfile.io[®]

TrueProfile.io is the industry leader for the verification of applicant qualifications such as diplomas or employers references. TrueProfile.io, powered by the DataFlow Group (founded 2006), conducts Primary Source Verification (PSV) for every submitted document by reaching out to the Issuing Authority (IA) like a university, an employer or a licensing body to verify the authenticity of the document. Furthermore, during verification, the DataFlow Group manually checks that the Issuing Authority is accredited and legitimate¹.

If these conditions are met, TrueProfile.io issues a so-called TrueProof - the basic building block of its service which is a single positively verified document. TrueProfile.io uses Blockchain technology to store a TrueProof so that customers can access their TrueProof independently of TrueProfile.io and even if at some point in time TrueProfile.io should no longer be in operation.

¹ The manual checks by TrueProfile.io ensure that only correct information is written on the Blockchain, so entries like the following are not possible:

<https://www.verisart.com/works/23f2c64a-08c6-4a42-8013-84ac8422dff8>

01 Introduction

In 2009, the Bitcoin white paper² was published by a person or group called Satoshi Nakamoto. Bitcoin is a decentralized transaction system based on Blockchain technology. It is decentralized in a way which means that no central third party has control over the transactions written on the Blockchain. Furthermore, transactions accepted by the network are immutable as the network protects them from manipulation.

To achieve this, all transactions need to be ordered in relation to the time they occurred. These transactions are bundled in blocks that are chained one after another; the following blocks secure the previous blocks and this process secures the integrity of the system. In Bitcoin, one block per approx. 10 minutes is written. Ordering transactions by time and chaining them together is perfectly usable for timestamping³ and is a method already implemented by Satoshi

Nakamoto. For example, the following text is written in the first block of Bitcoin (Genesis block): "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks"⁴. This data is stored forever and can not be changed without changing all blocks which are built on the Genesis block. That's why a Blockchain is the perfect way to carbonate data forever.

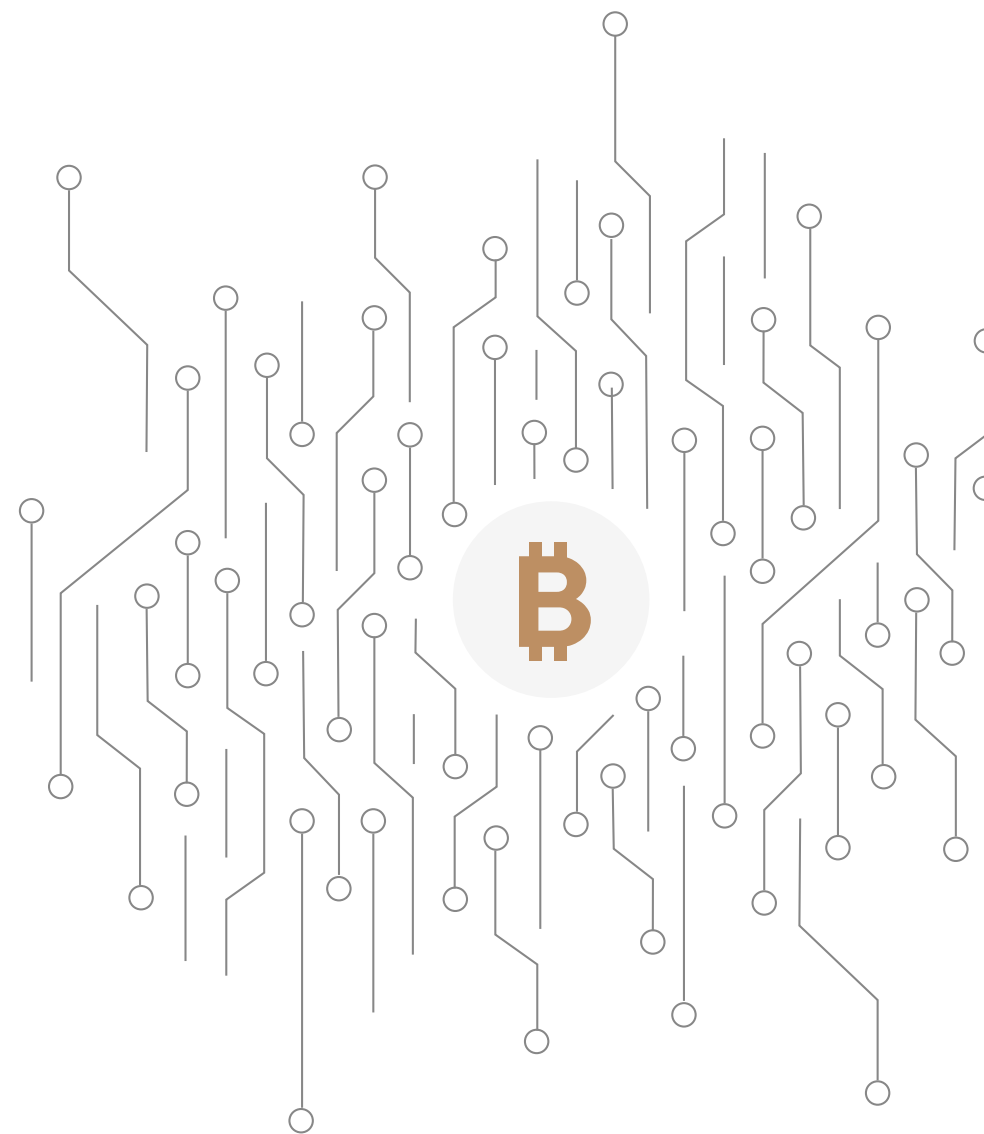
At the end of 2013, Vitalik Buterin⁵ took the idea of Bitcoin and introduced the concept of smart contracts. For Bitcoin, the software code that is executed for one transaction is defined in a limited set of Opcodes⁶, each of which performs a specific command or function on the Bitcoin Blockchain. Buterin suggested to use a Turing complete programming language instead of the limited set of Bitcoin Opcodes. Which makes it possible to bind any arbitrary software code to an Ethereum address - now known as a smart contract.

² <https://bitcoin.org/bitcoin.pdf> ³ <https://arxiv.org/abs/1502.04015> ⁴ https://en.bitcoin.it/wiki/Genesis_block

⁵ <https://github.com/ethereum/wiki/wiki/White-Paper> ⁶ <https://en.bitcoin.it/wiki/Script>

In contrast to Bitcoin, this leads to more flexibility as smart contracts can be added over time without any changes to the underlying Blockchain software structure.

But this flexibility does not come for free. The main issue with this new approach is that once a smart contract is deployed the logic can not be changed anymore. If the code is flawed, tokens managed by this contract can be, for example, frozen⁷ or stolen⁸. Furthermore, Ethereum has big issues with the transactions count and the data volume sent to the smart contracts. By July 2019, a full archive node of the Ethereum network needs to store almost 3 TB of data while a Bitcoin node 'only' needs to sync approx. 235 GB¹⁰ of data. In the [Appendix A](#) section we give a detailed explanation why we, for our use case, went for Ethereum and not Bitcoin as the underlying technology.



⁷ <https://blog.comae.io/the-280m-ethereums-bug-f28e5de43513>

⁸ [https://en.wikipedia.org/wiki/The_DAO_\(organization\)#History](https://en.wikipedia.org/wiki/The_DAO_(organization)#History)

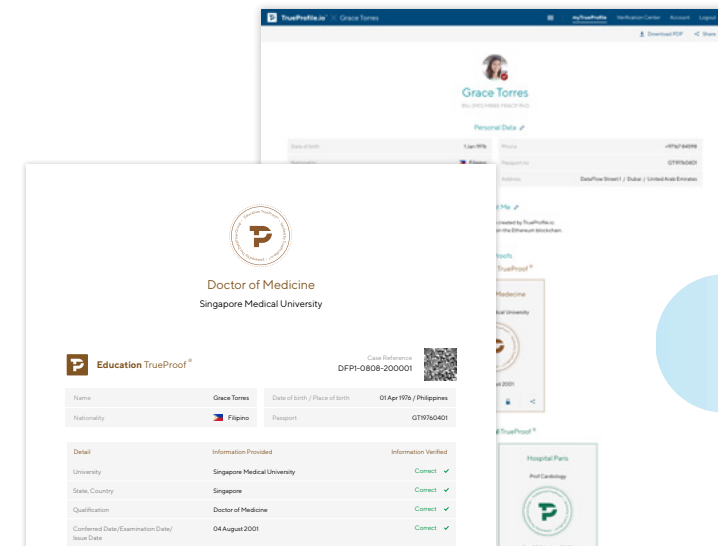
⁹ <https://etherscan.io/chartsync/chainarchive>

¹⁰ <https://www.blockchain.com/en/charts/blocks-size?timespan=2years>

02 Pragmatic Standard with a Hybrid Approach

Blockchain purists, obviously, will argue that the sheer existence of a centralized unit like TrueProfile.io/the DataFlow Group makes a travesty out of a radically decentralized set-up without any mediator whatsoever. On a theoretical level, they might be right. Yet, TrueProfile.io has purposefully chosen this path as it sees the world in pragmatic terms as what it is today and how it can serve users who intend to present themselves as trusted candidates – especially in an international, professional context. Instead of trying to onboard every university, every employer and every licensing body around the world to hold and safeguard their private keys for signature, TrueProfile.io is completely technology agnostic.

When it comes to obtaining its verification from the issuing source, the methods can range from writing, calls or even personal visits to the site. Based on this (positive) input, TrueProfile.io will issue a TrueProof. All the TrueProofs belonging to a user will be collected on their myTrueProfile page on TrueProfile.io¹¹.



¹¹ See a sample a myTrueProfile page
<https://trueprofile.io/true-profile/7f86f18473192e41489d1b3b6362e02>

This hybrid approach extends even further: TrueProfile.io puts the user in control of their verified data and enables them to expose this information to third parties of their choice:



User-triggered (“Member”)

TrueProof PDF

Storing the PDF hash on the Blockchain. The member might send a TrueProof PDF to any 3rd party who can then check its validity against the Blockchain | de-centralized approach.¹²

TrueProof JSON¹³ (pure data object)

Bringing the idea of hashing the PDF to the next level and store the hash of the plain data object in form of JSON on the Blockchain | de-centralized approach.

MyTrueProfile.io

Making the entire myTrueProfile public on a randomized link and sharing it with 3rd parties of the user’s choice | centralized approach.

Employer-induced (“Business Partner”):

A Business Partner sends pre-paid voucher codes to an applicant who in turn signs up, redeems the voucher and conducts PSV. Via their log in, the Business Partner is presented with the TrueProofs and the myTrueProfile of the applicant while the member safeguards their TrueProofs on their myTrueProfile for future utilization.

Platform-driven (“TrueProfile.io Connect”)

Bringing trust to 3rd party profile-based services by integrating the attribute “verified” using the TrueProof JSON.

¹² For a live demo of the Blockchain in action scroll down on <https://www.trueprofile.io/member> to “Take a free tour of our services”

¹³ <https://en.wikipedia.org/wiki/JSON>

Through all of these means, centralized and decentralized, TrueProfile.io aims to further the aspiration of becoming the standard for document verification. But what does that “standard” mean and how do we get there? Let’s first declare what it will not be or rather how it will not be achieved: It is very unlikely that any supra-national body like the United Nations, The Hague Convention or any similar consortium will all of a sudden and solemnly declare something to be “The Standard for Document Verification”. By contrast, the standard will be achieved initially through adoption from a few key actors, ideally in one geography - and/or industry-cluster, before spreading out further horizontally and vertically until it will accelerate momentum towards broad usage and acceptance.

In order to reach this place, is TrueProfile.io not subject to the archetypical chicken-and-egg problem where individuals will want to have a verification only if it’s sufficiently accepted by employers, immigration authorities, regulators and vice versa, employers, immigration authorities and regulators will only accept individuals with TrueProofs if there is a critical mass of them?

Yes, indeed, therefore TrueProfile.io strives to drive both utilization and acceptance of TrueProfile.io from both sides of the market starting with present and past applicants from the DataFlow Group. From there it extends to the “outer world” of individuals who need to present trusted documents through efforts in sales, business development and partnerships. Once a certain critical mass is reached, TrueProfile.io will start engaging strongly via lobbying and direct conversations with governmental bodies, universities and employers to seek acceptance for its standard. Narrating these points of acceptance back to the B2C side of document owners will let them be safely convinced that TrueProofs are being broadly accepted and hence will embark utilizing them.



03

Extension of TrueProfile.io[®]

In a subsequent step, once creating and administering cryptographic keys has evolved further into mainstream, the Issuing Authority of documents would require a profile (aka identity on the Blockchain) themselves. In this scenario, a service like TrueProfile.io would sign initially whereas the Issuing Authority would co-sign the data as well. In a final stage of full decentralization, Issuing Authorities could be enabled to sign the TrueProofs directly.

Another direction where TrueProfile.io is perfectly lending itself towards is a hybrid approach of building a digital identity of users who possess the property of

being portable. Combining an “Identity TrueProof” based on banking KYC-robust online verification of government issued documents (ID or passport) with one or several self-sovereign identity initiatives like uPort¹⁴ would establish an interoperable framework for further network distribution. Looking from another perspective, opening up the export and inclusion of document TrueProofs into these self-sovereign identities under the full control of the individual should be made available. Last but not least, considering participation in wider initiatives like ID2020¹⁵ appears to be another logical option.

¹⁴ <https://www.uport.me/>. ¹⁵ <https://id2020.org/>

04

The strength of data on a Blockchain

As the Blockchain is replicated across many thousands of computers, it would be a waste of storage to store the full documents on all replicated nodes. It would also become critical for privacy if CV data was stored on a publicly accessible Blockchain like Ethereum. That is why TrueProfile.io proves the authenticity of a document by storing the documents fingerprint, but not the document itself on the Blockchain. The fingerprint is comparable to the human fingerprint. If a fingerprint is found, it ensures that the fingerprint came exactly from this document as no other document can create the same fingerprint. Fingerprints also solve the privacy concerns, as the fingerprint of a document does not reveal any information about the document's content. For example, in the same way that a human fingerprint does not give any information about the owner's hair colour. In computer science those fingerprints are called hashes. For Ethereum and Bitcoin the hash named SHA-256, member of the SHA-2¹⁶ family, is used.

¹⁶ <https://en.wikipedia.org/wiki/SHA-2>

05

Technical design of our hashing algorithm

In this chapter we want to describe the hash mechanism we implemented at TrueProfile.io.

While calculating the SHA-256 of a file like the TrueProof PDF is a straight forward calculation, the hash of a data object in JSON data format is not. There has been two major issues for us: the normalization problem and the wish to validate only a subset of the JSON object.

¹⁶ <https://en.wikipedia.org/wiki/SHA-2>

Let's start with the normalization issue. For a JSON object, unlike others, the order of the elements does not matter. Let's look at two example JSON objects:

```
{  
  "first_name": "Grace",  
  "last_name": "Torres"  
}
```

-and-

```
{  
  "last_name": "Torres",  
  "first_name": "Grace"  
}
```

As you see the two JSON object have the exact same meaning - a person's first and last name - so the hash of the two objects should return the same value. Implementing a naive JSON to string and hash, the resulting string would not cover this.

In addition to ordering, the representation of numbers can also cause issues. Let's look at this example:

```
{  
  "name": "Grace Torres",  
  "grade": 4  
}
```

-and-

```
{  
  "name": "Grace Torres",  
  "grade": "4.0"  
}
```

Other issues such as if there is a space between the key and value in the JSON representation will also matter if a standard JSON to string implementation is used. All examples will not produce the same hash with a JSON to string method which will be hashed after. That's why we needed to implement a smart hashing mechanism.

The next goal was to let our users decide what data they want to share. For a standard hashing implementation it is always mandatory to reveal the full data object to reproduce the hash. At TrueProfile, io we take privacy very seriously. For example, if a member wants to only share certain parts of his TrueProof on LinkedIn we needed a mechanism which allows a user to hide specific data and still generate the same hash.

To keep specific data private, it is mandatory not to hash the full JSON object once, but instead each element of the JSON object individually. The individual hashes can then be joined together and hashed again (similar to a Merkle Tree¹⁷). The user can then choose which values they want to reveal from the full object. Unfortunately this approach leads to one serious issue. When each element is hashed individually and, for example, a date field like 'Grade' has a finite set of possible values an attacker who knows the hash of this element can simply try all finite possibilities and will only get a matching hash if the input value was correct. So, the attacker will basically be able to use brute force to gain the grade from the hash value they obtained. To mitigate this issue we need to add a random salt value to each element and return it to our users.

Our C# implementation is based on Ben Laurie's version ObjectHash¹⁸. We take a JSON object and normalize it so that e.g. the ordering and the casting issue between int and float does not matter.

The full implementation
is open source under MIT
licence and can be found
on Github:

<https://github.com/weigandf/notarization-objecthash>

¹⁷ https://en.wikipedia.org/wiki/Merkle_tree

¹⁸ <https://github.com/benlaurie/objecthash>

06

Why we need a smart contract

Changing existing data on a Blockchain is impossible - a Blockchain can not forget data, because it stores data immutably. But imagine a (nearly impossible) case where TrueProfile.io issued a TrueProof and after some time it became clear that the data is fraudulent. Due to the restriction of a Blockchain, the user could always use the certified data from TrueProfile.io and mislead other people with the invalid data. Thus, we need to implement a protocol on a Blockchain that provides a subsequent transaction to revoke the original document.

That is why for our use case we would need a Blockchain to support the following three methods:

> ADD

Add new hash to the Blockchain

> REVOKE

Revoke a hash from the Blockchain if fraud is detected

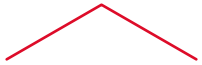
> IS VALID

Check if a hash is valid by checking it on the Blockchain

07 Technical details for our smart contracts



In the following section, the smart contracts implemented by TrueProfile.io are described on a technical level. You might want to skip this part if you are not familiar with the development of smart contracts in Solidity.



Our smart contracts are open source and under MIT license. They can be found on Etherscan:

<https://etherscan.io/address/0x000d2d31815990fca6f69dfd978c4d4a56b2ed6b#code>

The full implementation is split into three different smart contracts:



1 – Accessible

Refers to the smart contract which manages the access rights of certain calls in smart contracts by implementing modifiers. The smart contract 'Accessible' is extended by the two other smart contracts. Thus, the other two smart contracts inherit the modifier properties.

2 – TrueProfile.io Storage

Refers to the smart contract which is used to store the full state of the TrueProofs which are added or revoked.

3 – TrueProfile.io Logic

contains all the logic which is needed to add and revoke TrueProofs. There is no data storage in this contract. All data is sent and retrieved from the storage contract.

Here are the detailed descriptions of the three smart contracts:

1 – Accessible (Subclass)

Accessible is a subclass to manage the access rights of all smart contracts who inherit it. Each contract inheriting from Accessible will have an Owner that the smart contract belongs to. Other than that, each contract will have a list of addresses which can access certain methods of the smart contract. So there are two different access levels:

Accessible

multiple addresses which can be added and removed only by the owner

Ownership

the one address who created the contract

Only the owner of the contract is allowed to add or remove addresses which can access functions of the smart contract. By default, the first owner of the smart contract is the address from which the contract was created. The owner can transfer the ownership of the contract to a new owner of their choice.

2 – TrueProfile.io Storage

The contract used for storing all data is called the TrueProfileStorage contract. It inherits from the smart contract 'Accessible' and thus has an owner (the creator of the TrueProfileStorage contract) and a list of persons who can access the smart contract.

This smart contract is used to abstract all Ethereum storage operations and this manner handles the full state of the smart contract. In the next section, we will introduce the TrueProfileLogic contract. The logic will have direct access to the storage and can manipulate the state of the contract only by calling the storage contract.

This contract defines a variety of mappings to store all needed data. The most important mapping is the signature storage. This mapping is used to store a corresponding signature for a hash it commits to. Other than the signature, there are multiple other mappings. They are currently not used, but later if the logic is changed, the mappings can be used to store any arbitrary data if needed in the logic layer.

The TrueProfile.io Storage contract provides a CRUD (create, read, update, delete) interface to access all those mappings. Access is only possible if the administrator added the address as with access enabled.

3 – TrueProfile.io Logic

This contract is used to access the storage introduced in the last section. The current implementation of the smart contract has three main functions:

A / Add TrueProof

This function is used to add a new TrueProof to the contract. The function takes a signature as input, validates it against the hash and checks if the callee belongs to the list of addresses who have been granted access to the smart contract. If everything is correct, including the signature, the TrueProof hash is added to the storage contract.

B / Revoke TrueProof

This function is used to revoke a TrueProof. There is a low probability that a TrueProof was issued due to a mistake from an Issuing Authority or outright fraud. However, in this instance there is a revoke function which can be called to mark the hash as invalid. The revoke function also stores a reason ID for the revocation in the Blockchain. This helps to prevent misuse of the system. We don't want to validate something forever which is not correct.

C / IsValid TrueProof

The isValidTrueProof function provides an interface to check if a certain hash is currently a valid TrueProof. The contract checks if a valid signature is stored for a certain hash. It first validates the signature for accuracy and in a second step whether the TrueProof was not revoked. If both checks return true the function returns valid. As this is a direct function of the smart contract, this method can also be called directly from third party services like Etherscan¹⁹.

¹⁹ <https://etherscan.io/>

Appendix

Appendix A

Why Ethereum was chosen as the underlying protocol instead of Bitcoin

Bitcoin is the first implementation of a Blockchain that is widely²⁰ used. Bitcoin has many advantages over other Blockchain implementations like Ethereum. The most important aspect for time stamping is the high hashrate of Bitcoin²¹. Modifying data becomes more expensive as a high hashrate implies also high costs associated with the energy used for hashing²². A second advantage of Bitcoin is the simplicity of the script language it uses (OP codes²³). A simplistic language makes it easier²⁴ to prove the correctness of a script but on the other hand complicates development. Currently, the functional language Ivy²⁵ tries to make the Bitcoin script language easier to use by wrapping the script language in a function language.

Due to the simplistic design of Bitcoin, the range of features that can be developed on top of Bitcoin are very limited. Ethereum, on the other hand, allows for the development of more complex scripts (smart contracts). On Ethereum, there is a variety of available languages like Solidity, LLL, Vyper and others. Solidity is the most widespread language used for smart contract development. Complex scripts come with a drawback: the security of the smart contracts is not formally verifiable²⁶ which increases the sensitivity to bugs or critical issues²⁷. Therefore, substantial tests are necessary.

²⁰ <https://coinmarketcap.com/de/>

²¹ <https://blockchain.info/de/charts/hash-rate>

²² <https://gobitcoin.io/tools/cost-51-attack/>

²³ <https://en.bitcoin.it/wiki/Script>

²⁴ <https://hackernoon.com/smart-contracts-turing-completeness-reality-3eb897996621>

²⁵ <https://github.com/ivy-lang/ivy-bitcoin>

²⁶ Formal Verification of Smart Contracts: Short Paper - HAL-Inria, K Bhargavan et. al.

²⁷ <https://hackernoon.com/yes-this-kid-really-just-deleted-150-million-dollar-by-messing-around-with-ethereums-smart-2d6bb6750bb9>

Appendix B

What the implementation in Bitcoin would have looked like

As already mentioned, for the certification of documents three main features are needed:

> ADD

Add new hashes of proven documents to the Blockchain

> REVOKE

Revoke a hash from the Blockchain if a TrueProof was issued by accident

> IS VALID

Check if a document is valid by checking the Blockchain

Changing data on a Blockchain is impossible by the definition of a Blockchain provided previously. A Blockchain can not forget data, it stores data immutably. Thus, we need to implement a protocol on a Blockchain that provides a subsequent transaction to revoke the original document. A simple protocol on Bitcoin might look like this:

Take the hash of the document and store it using the

`OP_RETURN` command.

Prepend a protocol description before the hash like:

`TP ADD <HASH>`.

Revoke it by prepending the following protocol description:

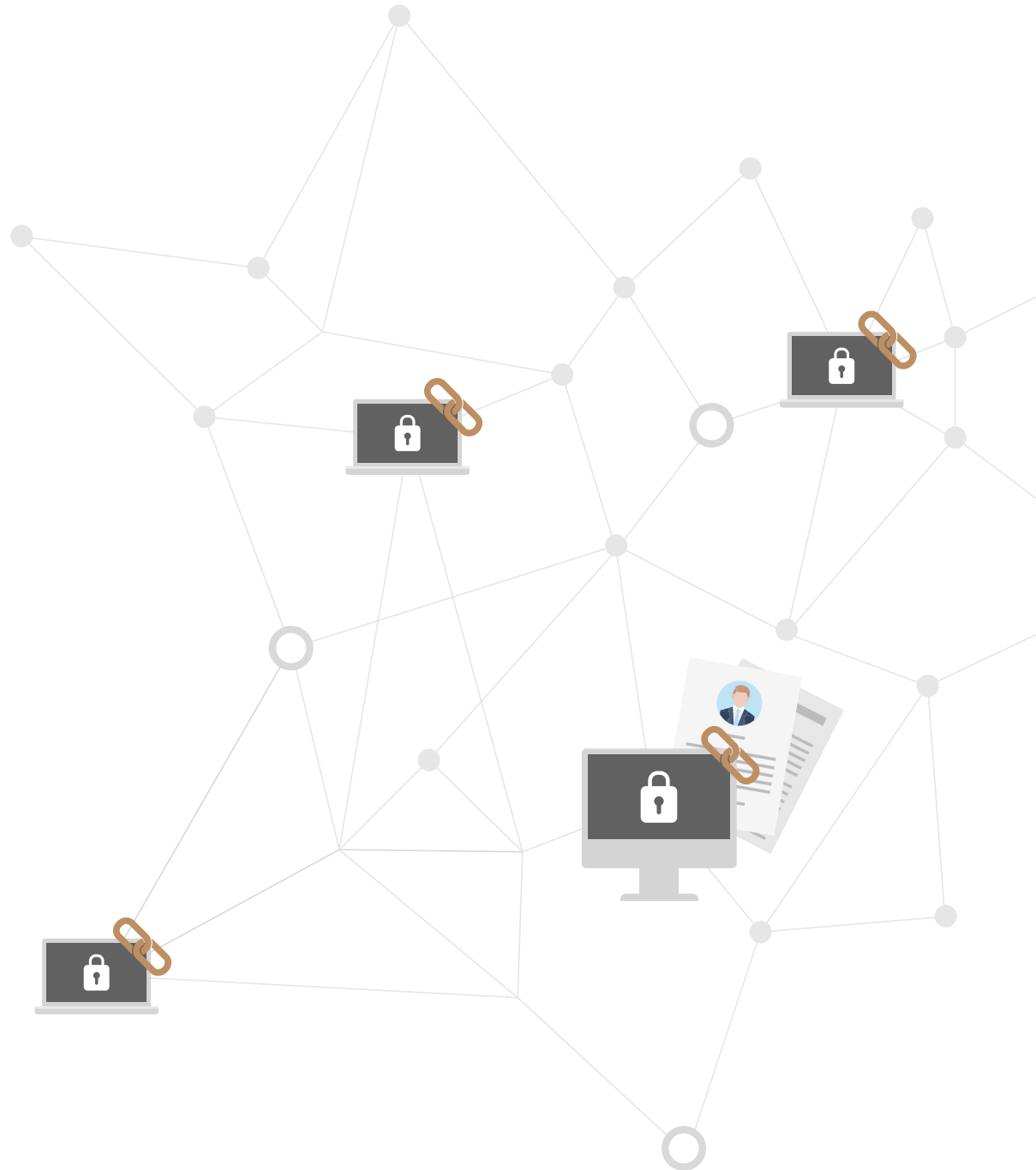
`TP REVOKE <HASH>`

The main drawback with Bitcoin as an underlying protocol is that the protocol definition is not code by law. This protocol is simply enriched metadata and needs to be communicated. Other than that, it would be required to search through the full Blockchain and check if once a TP ADD was found and no TP REVOKE is following.

The next problem of a Bitcoin based protocol is the management of rights. If a document hash is accessible it can simply be revoked by generating the TP REVOKE protocol data and then use a transaction with OP_RETURN to store it in the Bitcoin Blockchain. But not everybody should be allowed to revoke or create a TP ADD or REVOKE protocol entry in the first place.

The rights management issue can be solved by including only a subset of allowed addresses to the protocol - the protocol is then permissioned. All other not defined addresses can not create valid protocol entries. But due to the, by design, limited capability of the OP codes in Bitcoin it is not possible to implement a 'code is law' protocol on Bitcoin.

That is why for this use case, implementing a smart contract is currently the best way to go. TrueProfile.io wants to use the Blockchain for what is possible today, so for that reason TrueProfile.io takes every TrueProof PDF hash and stores it on the Blockchain.



Ready to
get started?



www.trueprofile.io