Note on KeeeX's permanent availability, resilience, scalability, sovereignty, compliance, auditability and interoperability

This document explains the core properties of KeeeX algorithms and implementations, most importantly those relating to operations sustainability and scaling. Among covered questions are: what happens if KeeeX disappears.

Contents

We hope that you enjoy the unique possibilities brought by the KeeeX technology as much as we do!

Technology

Keeexed data is under your full control thanks to on premises and client-side solutions. The keeexing algorithms are published by patents and to a large extent by published and archived verifier code. Implementations use open source battle tested cryptographic algorithms.

KeeeX leverages two active FR/UE/US patents

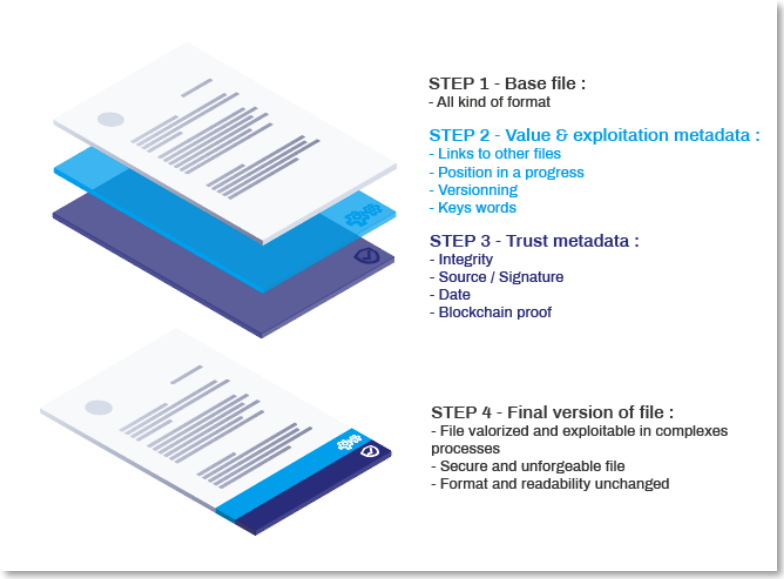
These patents describe a process allowing for embedding proofs and other metadata within files with no alteration of functional contents regardless of their format. This multi effect process is called "keeexing". The patents do reveal the core principles if not the specifics of an implementation. This ensures permanent availability of the technology in the future, whatever happens.

The keeexing process embeds metadata that yield standalone self-probative files

This process allows to:

- **verify files** for their integrity and authenticity without the need for a separate infrastructure and without time limits or business model
- **chain files** together using provably verifiable references, in the form of a variation of their hashes
- **index files** with their own hashes thanks to bubble babble ascii encoding
- let users **name files** by their hash since it is bubble babble encoded hence always pronounceable
- **embed multiple signatures**, while preventing unexpected signers, and supporting delayed signing

- **embed more properties or data** - for instance geotagging information in a pdf file, or technical json information in a picture
- **embed variable "hash protected" information** - i.e. data that is not taken in account when computing the hash because it is known after keeexing takes place (like signatures). Among examples are a blockchain anchoring certificate or timestamp, or a link to a further version of a file, a further statement made and signed by the original author…



User files can be processed on client side.

Original files and their keeexed counterparts may never leave a user's premises. They reside on their side and are thus permanently available according to the desired settings. We however also provide cloud processing for specific needs or to obey technical constraints.

KeeeX processing scales without limits

Because the keeexing process is distributed across all systems that require it, scalability is unlimited.

KeeeX metadata are embedded as clear text ascii within files at non-destructive and value adding locations

They always:

- **prevent from damaging** the operational features of files.
- **allow for their indexing** by search engines when possible
- **help discover them** using low level operating system indexers or search functions
- **help gather for deletion or compression** based on KeeeX metadata, therefore easily addressing key GDPR requirements

Among such locations are file comments, XMP or Dublin core metadata, EXIF….

KeeeX metadata are expressed in a simple box language

All statements are enclosed within "keeex" … "xeeek" boundaries. The syntax for embedded:

- **hashes** is `keeex self …, {extra metadata} xeeek`
- **properties** is `keeex prop …, {extra metadata} xeeek`
- **references** to other files is `keeex ref …, {extra metadata} xeeek`
- **protected data** is `keeex protected …, {extra metadata} xeeek`
- **signatures** is `keeex signature …, {extra metadata} xeeek`

`{extra metadata}` are expressed in a JSON like syntax and allow for specifying parameters or further cryptographic evidence.

Since keeexed files contain their own hash (within `keeex self … xeeek` boundaries) this hash cannot be computed from the original file. Instead it is computed by a straightforward and paper documented process (simplified here) involving:

- **stripping the file from every declared hash**, whether enclosed in `keeex self … xeeek` or present elsewhere in the file (KeeeX allows for repeating the hash in multiple places, which is useful for displaying self QRCodes for instance when applicable)
- **stripping the file from protected sections and signature values**
- **appending a salt**

Because the KeeeX hash is not the plain file hash, and it combines so many features, it is called an **IDX** in KeeeX jargon. Please note that the plain hash of the original file can be registered as value of a KeeeX property for later use.

Keeexed files are protected from first to last byte

When a keeexed file contains no protected statement, any attempt to change a single bit of the file will result in a file detected as damaged by verification:

- changing contents results in corrupt hash
- changing hash value alone results in corrupt hash
- changing signature public key results in corrupt hash and signature value
- changing signature value results in corrupt signature

This significantly differs from other digital signature schemes, like for PDF for instance. A signed PDF file remains valid after keeexing: it is verified as genuine for the two schemes at once. On the other hand, a keeexed PDF cannot be signed using PDF signature algorithms without destroying the validity of KeeeX probative metadata.

Verification is possible for every file using a simple universal algorithm

Thanks to a universal box language, verification remains format agnostic: it suffices to scan the file searching for the abovementioned patterns to first compute a hash to compare with self, then check the validity of signatures.

KeeeX verification scales without limits

Again, the verification process is distributed by nature.

Keeexed files are permanently protected and verifiable

By using battle tested ultra-robust Bitcoin cryptography, by allowing for the embedding of multiple hashes according to various algorithms, by allowing for the injection of multiple digital signatures, that combine Bitcoin EC and X509 signatures, KeeeX achieves unprecedented levels of resilience.

Loss of probative information is impossible which warrants perpetual verifiability.

This is so because all metadata is stored inside the file.

Current knowledge about cryptography ensures definitive unforgeability

Neither state of the art hashing or signature algorithms suffer potential attacks. Hashes algorithms and Bitcoin signatures are used to protect billions of dollars.

Quantum computing is not a known threat to hashing algorithms as far as we know.

Elliptic cryptography instead exhibits a fragility, but the use of Bitcoin algorithms warrants that risks on that side would be mitigated long before they become significant, thereby giving time for important files to be re-signed.

On the perpetual availability of KeeeX verification and processing

Keeexed files are forever verifiable forever even without KeeeX. Keeexing will be possible forever whichever the future of KeeeX.

The keeexing process solely uses off the shelf open source cryptographic algorithms

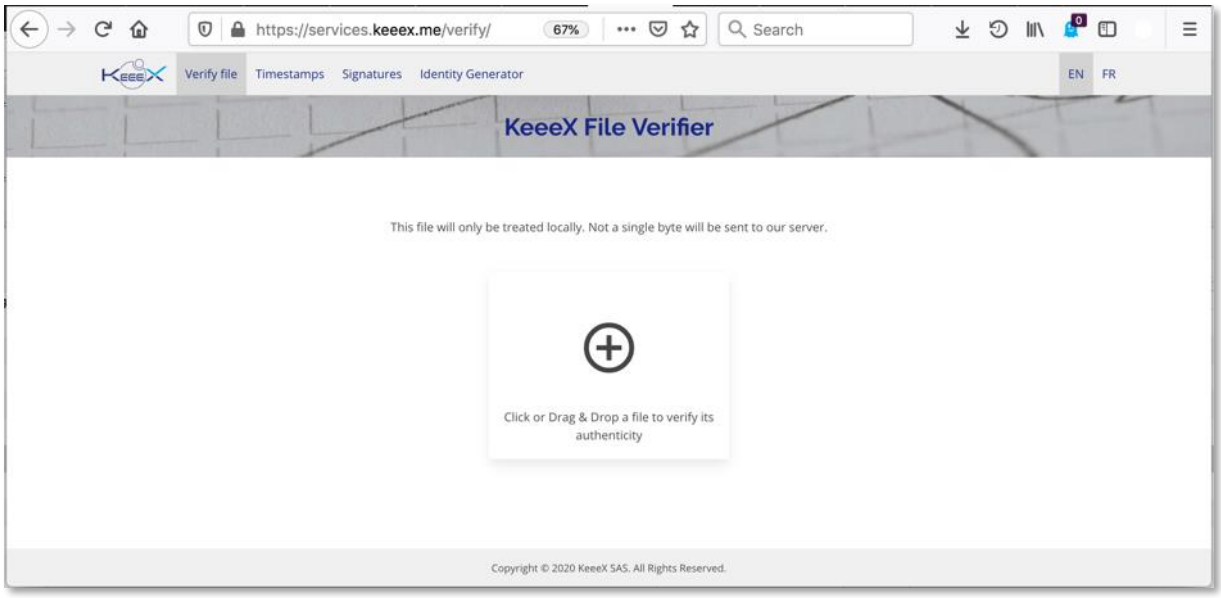
These battle-tested algorithms are available from **OpenSSL** and most common **NodeJS** libraries. This includes the **Bitcoin** cryptography: hashes using **Sha256** or more, elliptic cryptography using the **secp256k1** curve or others (according to Curve https://safecurves.cr.yp.to/) for signatures (ECDSA), computing shared secrets (ECDH) and in some instances encryption. KeeeX also uses more OpenSSL functions for **X509** digital signatures and server certificates.

Offline capable keeexed file verifier implementations abound

This is so on KeeeX customer portals or our own. The Javascript verification code runs client side, with no data sent to our servers except the file hash for retrieving blockchain evidence when appropriate. Beyond this mandatory requirement (public blockchain verification requires a web access), all KeeeX verifiers run offline, yet providing data integrity and authenticity evidence.

A KeeeX's verifier is located at:

- https://services.keeex.me/verify



The verifier displays most details regarding probative information that can be extracted from the file, including:

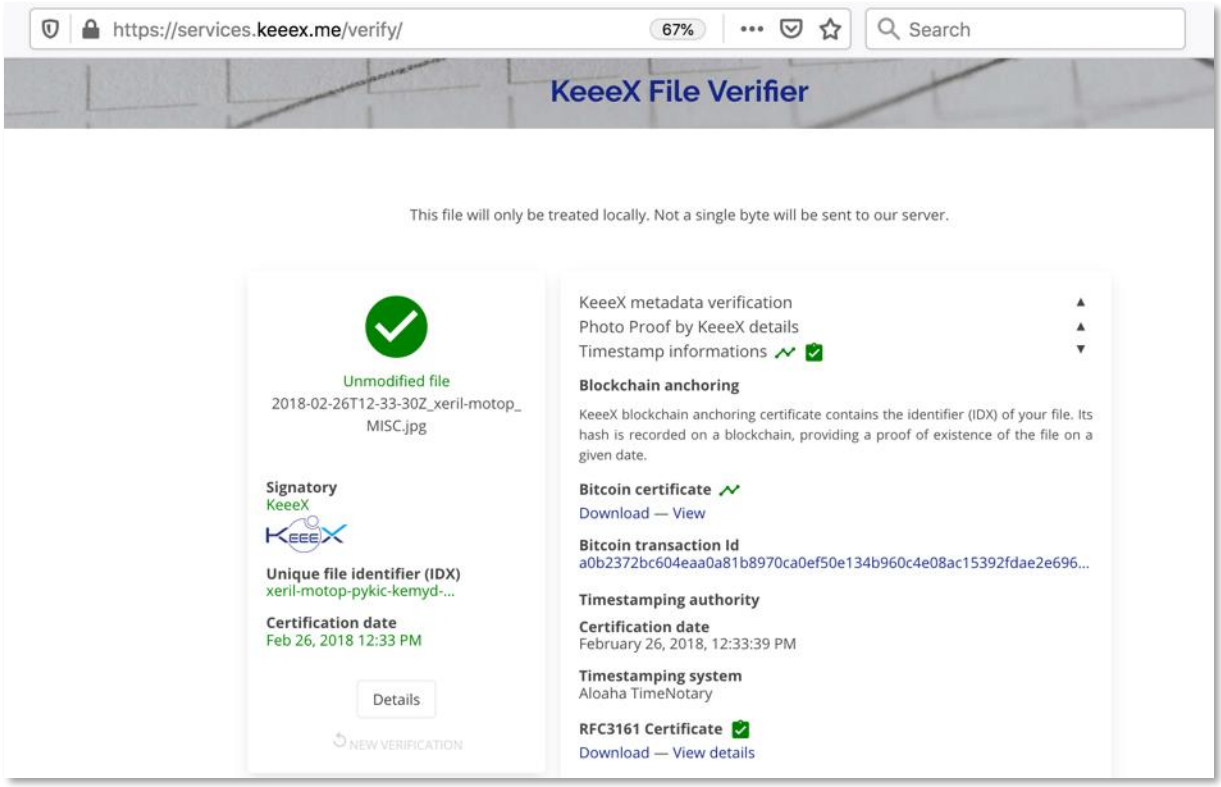
- IDX and other hashes if any,
- signer identities,
- author and multi signatures,
- links to timestamp and blockchain information,
- links to web searches of identities,
- links to signature verifier including third party Bitcoin signature verifiers plus
- geotagging information if available.

If you would like to check by yourself, demo files are available from https://keeex.me:

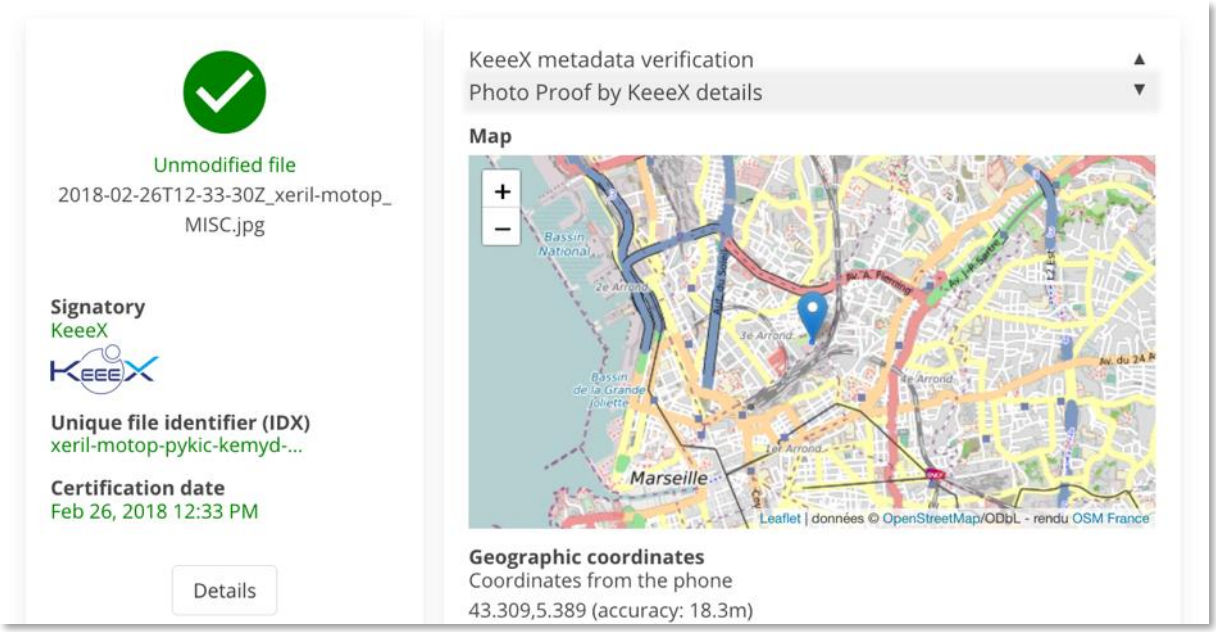
- valid keeexed file: https://keeex.me/wp-content/uploads/Logo-13.11.2020_kx_xizak-gozag.png
- invalid corrupt file: https://keeex.me/wp-content/uploads/Logo-13.11.2020_kx_xizak-gozag-1.png

Of course, the file name has nothing to do with the verification process. Below are details displayed when verifying a picture that contains geolocation information:

(test for instance with: https://keeex.me/wp-content/uploads/2021-03-10T16-28-02Z_xotiz-zavuv_MISC.jpg)



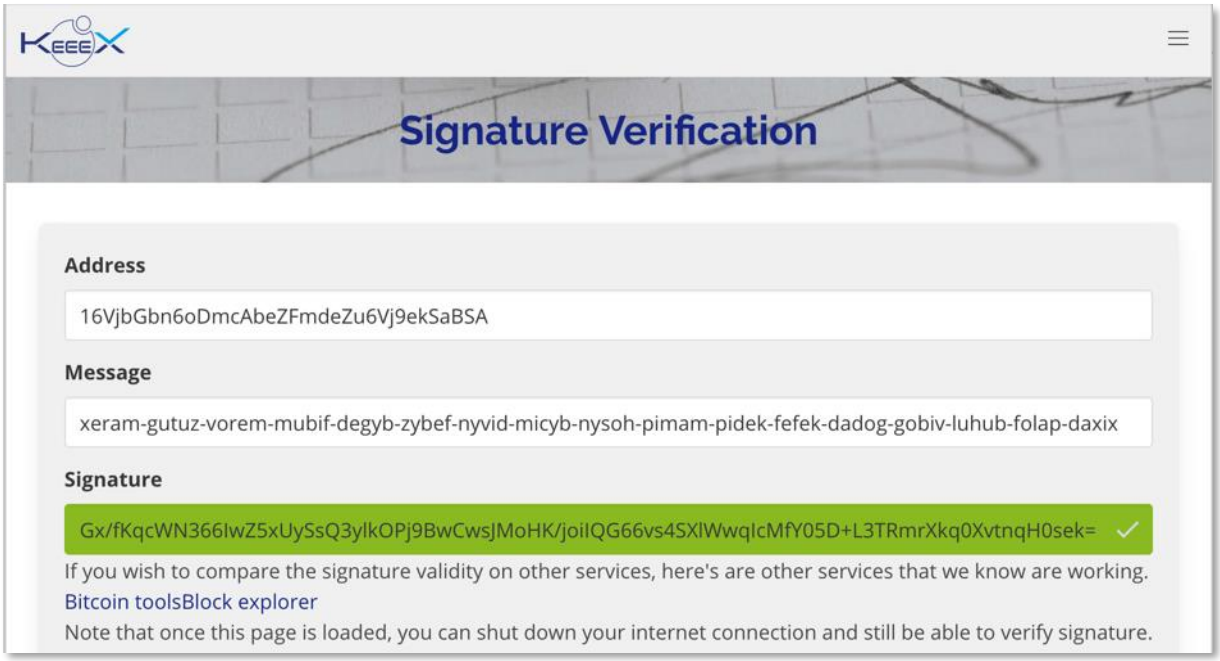
And



Signature verification can be performed using independent online Bitcoin message signature verifiers

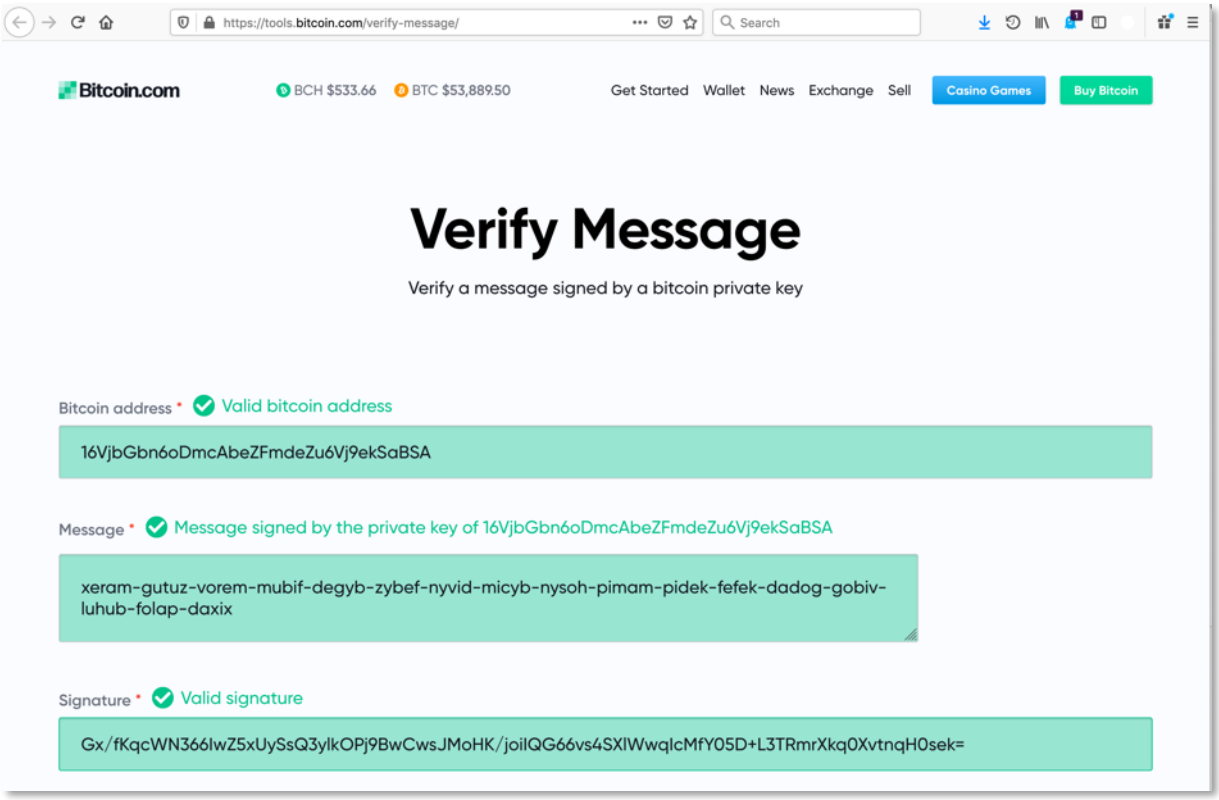
Below is what you get when clicking on the signature part in the details section of the verifier.

(cf: https://services.keeex.me/signature-verification/?adr=1BPy...oUcgX&msg=xuvah-...-molif...)



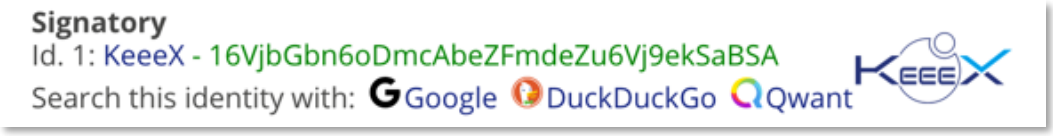
When clicking on the Bitcoin tools link above for instance (but you might as well use any Bitcoin message signature verifier from any app), you reach a signature verifier independent from KeeeX. Simply copy and paste the Bitcoin address used for signing, the file's IDX (hash) as a message, and the signature value, to get an independent proof that the file has a valid signature. Of course, all these elements are present in cleartext in the file you are willing to verify:

(cf: https://tools.bitcoin.com/verify-message/)



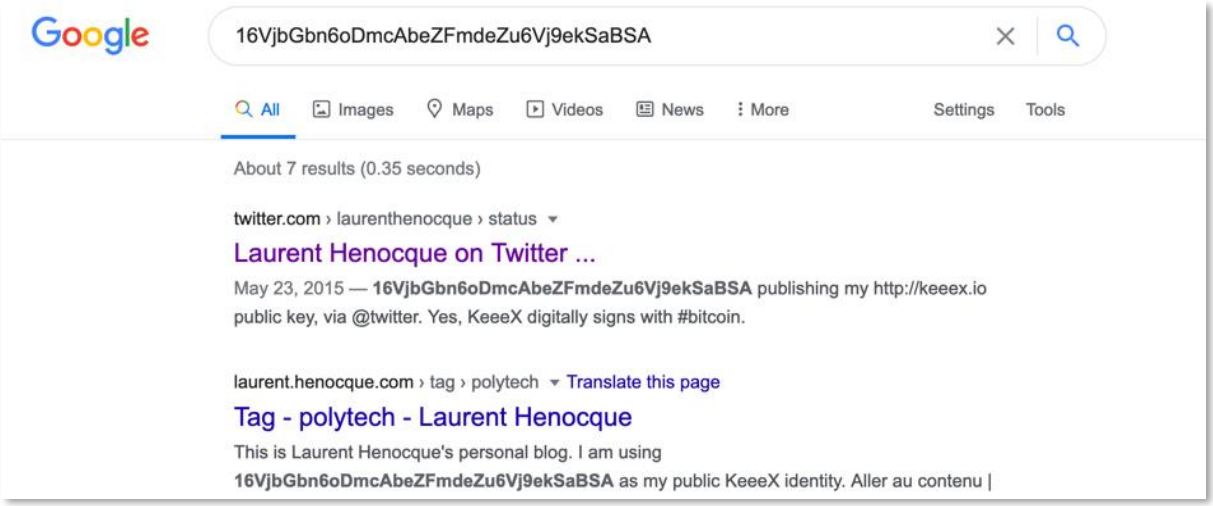
Signer identity may be searched online

Some search engines do index the Bitcoin public key strings. This is one advantage of using these as digital identities. You may have noticed the clickable search engine links on the KeeeX verifier details:

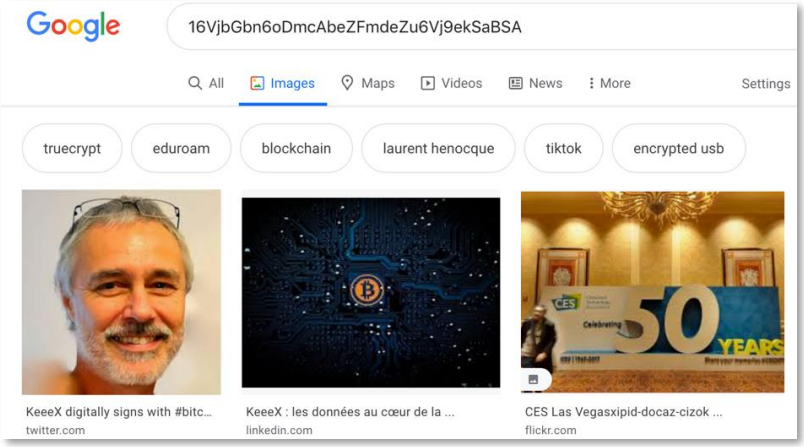


Well if so happens that the author has published their identity, you may get further details:

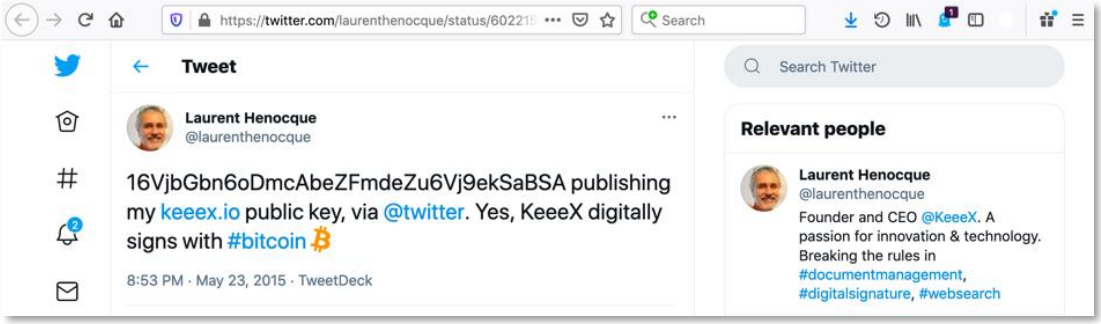
(cf: https://www.google.fr/search?q=16VjbGbn6oDmcAbeZFmdeZu6Vj9ekSaBSA)



Including images:



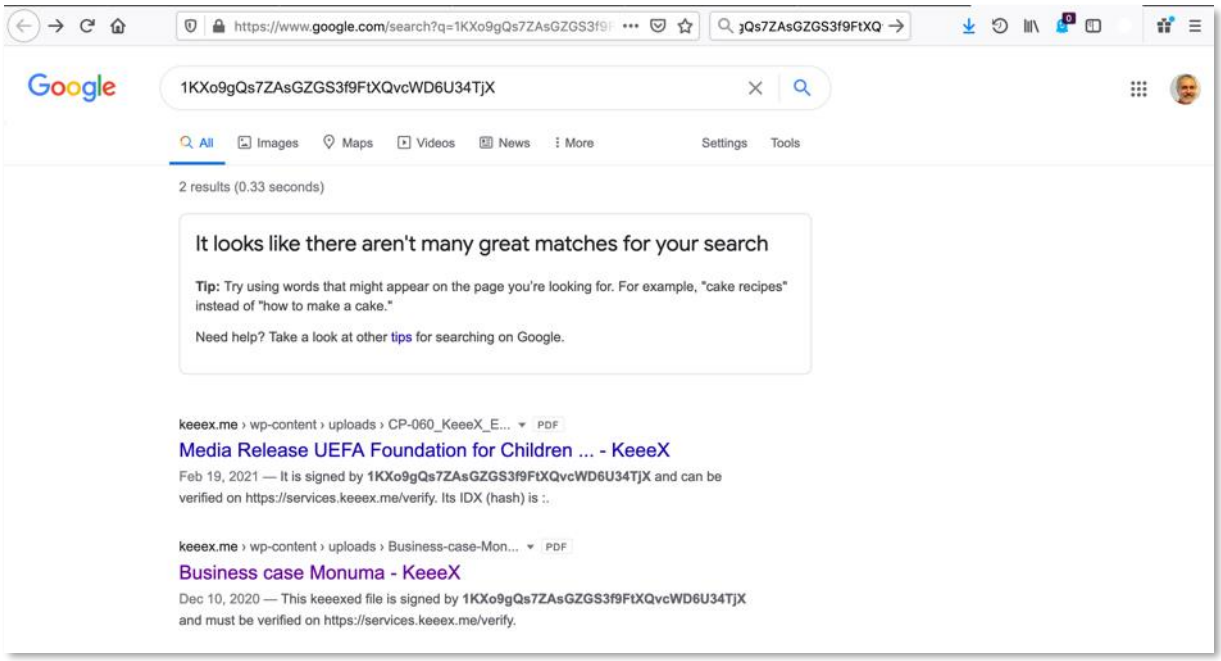
And clicking the links yields details:(cf: https://twitter.com/laurenthenocque/status/602215777626890241)



Web indexation engines may find documents searched by the identity of their author

KeeeX tools make their best efforts to place metadata in places that may be of interest to search engine crawlers. Pdf files may also be equipped with a visible overlay recalling the signatory identity and the file hash. Because these words are so unique, this is pretty cool: you just get the exact matches for some identity. As seen below the search engine may even complain that there are not so many results :-)

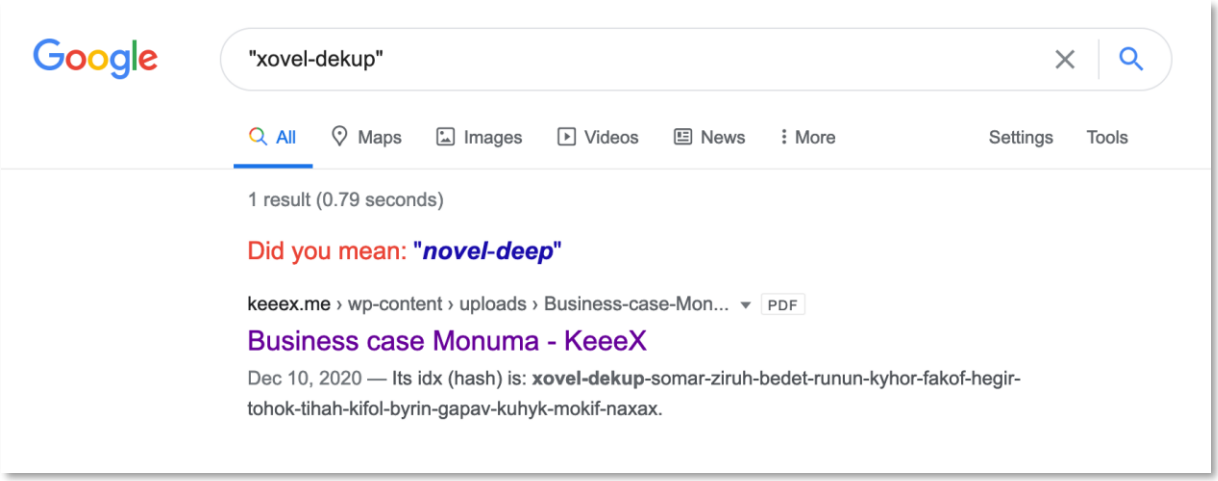
Let's try: https://www.google.com/search?q=1KXo9gQs7ZAsGZGS3f9FtXQvcWD6U34TjX



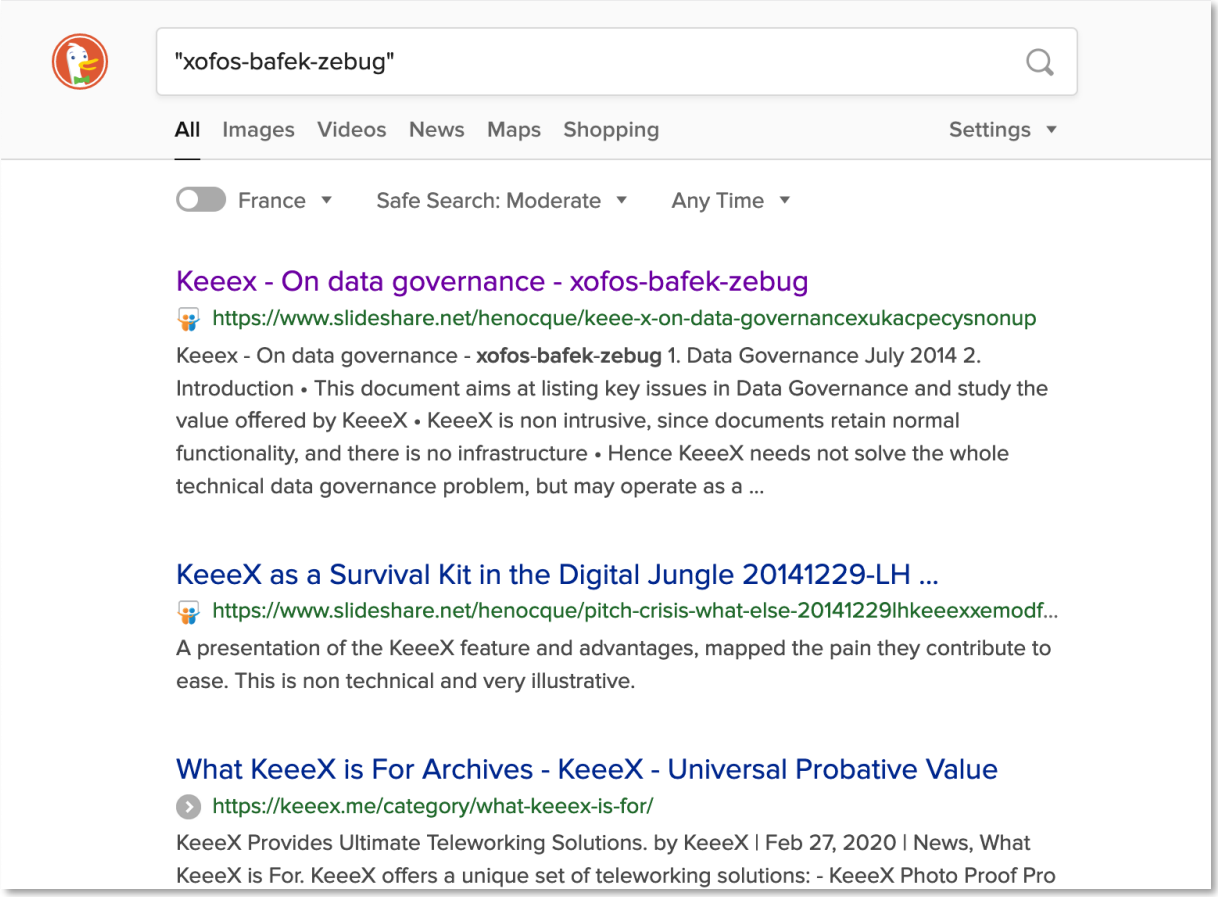
And Web indexation engines may also find files by their embedded hashes!

What works with public keys above also works with the file's hashes (idx).

Try again: https://www.google.com/search?q=%22xovel-dekup%22

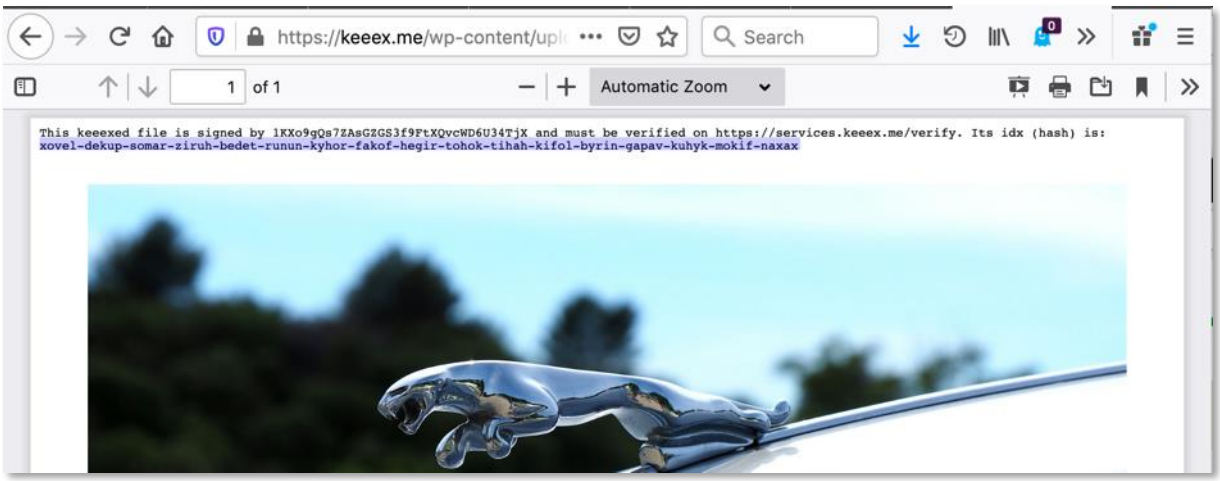


The above properties can be used to track leaked files down to the dark web, or websites grabbing your copyrighted data, or just to list the exact places where a file is mentioned by it's idx. Look for instance at https://duckduckgo.com/?t=ffab&q=%22xofos-bafek-zebug%22



Search engines index the pdf overlay mentioning the keeexed file IDX (hash)

By clicking on the link to the Monuma business case above (https://keeex.me/...Business-case-Monuma-keeexed-2020-12-10-xovel-dekup.pdf) you see this in action:



Former offline capable KeeeX verifier implementations are cloned on the web archive

The Web Archive is also dubbed the "Wayback Machine".

The general place for searching snapshots is here:

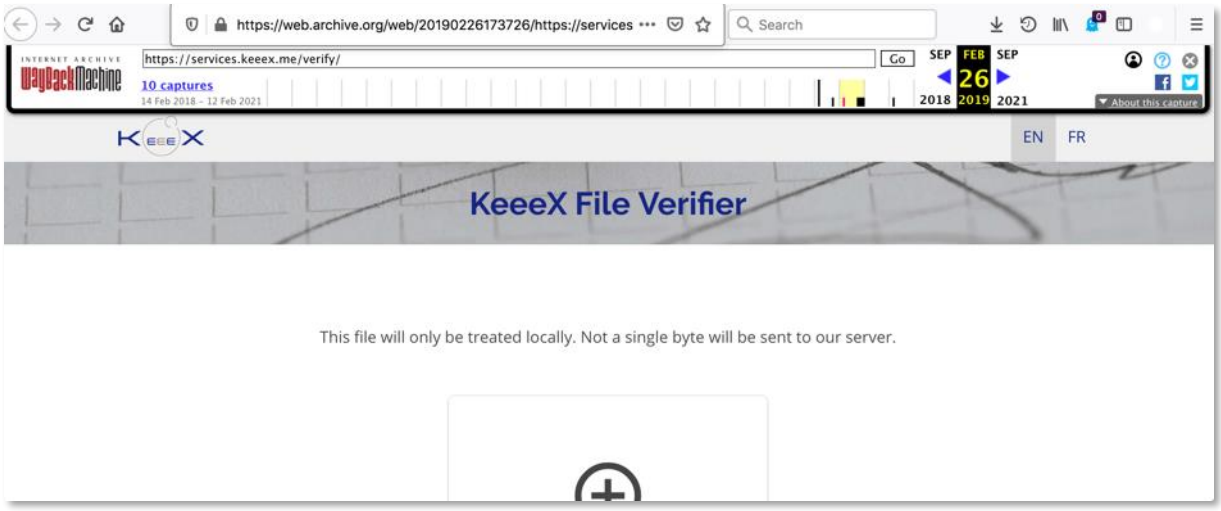
- https://web.archive.org/web/*/services.keeex.me/verify

possibly using test files from https://keeex.me if you don't have any.

Note that all snapshots work but may not display neatly due to awkward resource fetching by the wayback engine in some instances (we are working to prevent this)

You may use a visually happy recent snapshot

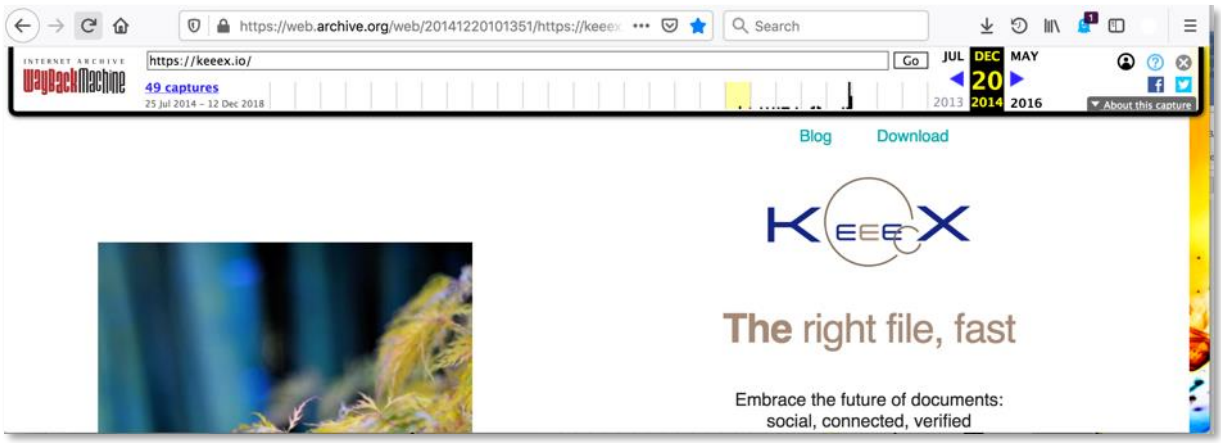
- https://web.archive.org/web/20190226173726/https://services.keeex.me/verify/



The historic website from 2014 is still alive and kicking

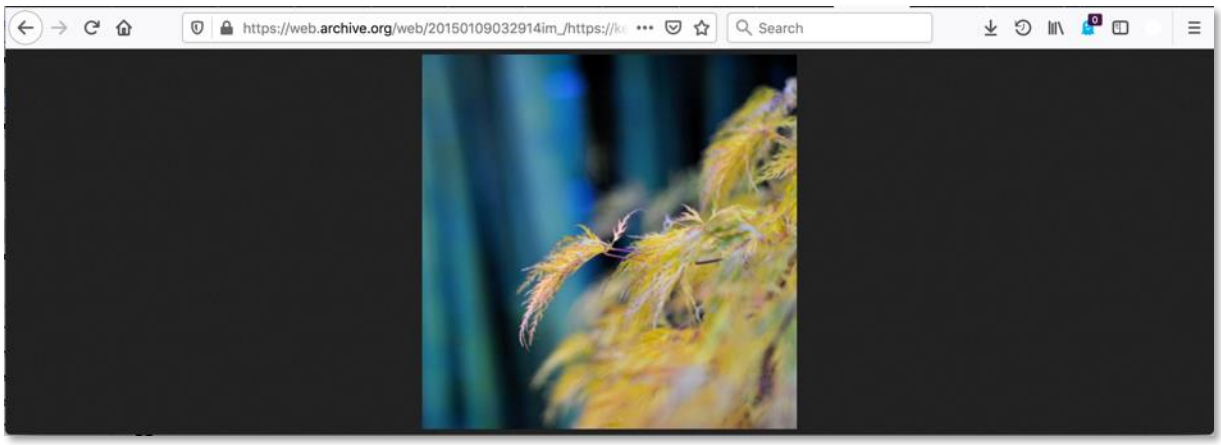
It is there (enjoy the rolling logo!):

- https://web.archive.org/web/20141220101351/https://keeex.io/

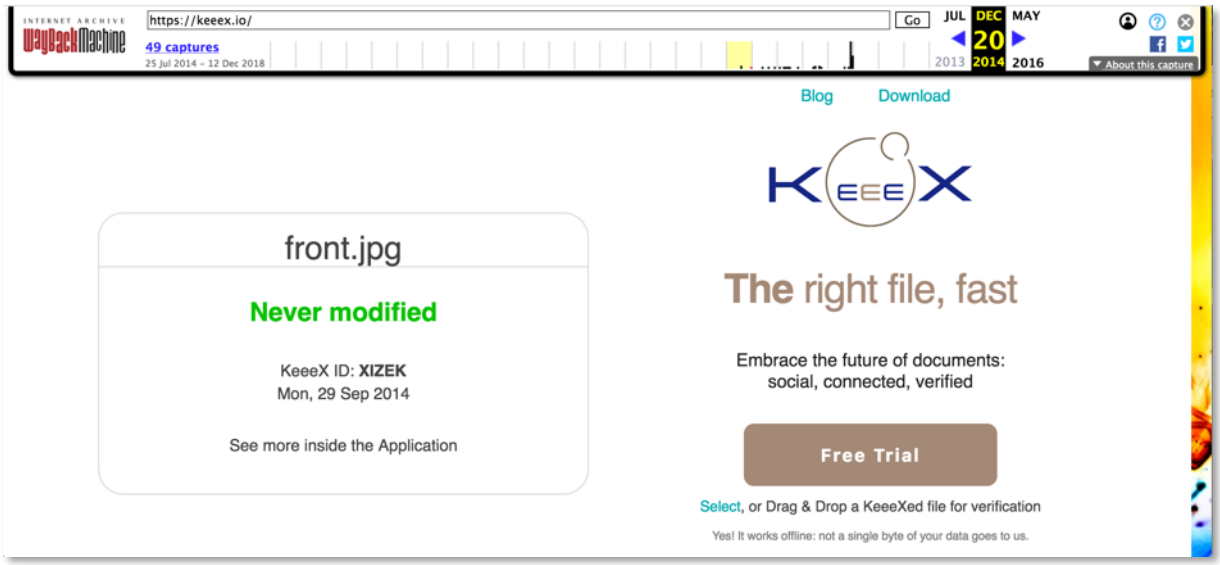


This website archive can be tested by dropping a picture from the site (download then drop):

- https://web.archive.org/web/20150109032914im_/https://keeex.io/images/front_keeex_xizek-morun.jpg



That yields the result below:



KeeeX verifier code is available forever

This is so because archived versions of KeeeX sites run a client-side JS.

We are working on the open sourcing and public exposure of a reference implementation

This will be published in Javascript that will complement the current SDK version that is compressed and minified for ease of deployment and download speed.

KeeeX processing code can be guessed from verification, hence is available in case of failure

Of course, should the KeeeX company cease operations without follow-up, verification code and sample files provide enough details to any programmer to implement a keeexing processor.

A snapshot of the keeexing code has been deposited at the French Agence pour la Protection des Programmes (APP) and would thus be made accessible in case of a fatal failure.

KeeeX's approach to blockchain anchoring and timestamps ensures auditability

KeeeX approach to timestamping and blockchain anchoring supports scalability, time accuracy, human auditability, file level proof record control, resource savings, separation. Proof records are permanently available even without KeeeX.

As should be clear from the above, KeeeX does not per se require any blockchain for assessing that a file is genuine. The situation changes when time and proof of existence come into play:

- prove that a file existed at a very precise time.
- provide an unforgeable, permanent, publicly auditable proof of existence of a file

Preamble and position statement:

we have discouraged since inception the use of private or permissioned blockchains, that provide little to no probative value. Even some public blockchains may experience severe issues, as illustrated by the Ethereum backtrack for TheDAO (this was a governance decision), and more recently by a two-months reorg of the Verge blockchain (this was a consensus attack of unprecedented scale).

Public blockchains are not good at delivering accurate time information

at least cheaply. This is so because blockchain writes are extremely expensive, at variable and uncontrollable costs, or are mutualized, and then performed at larger scale intervals, typically several hours on Bitcoin.

Dealing with time typically requires the use of industry accepted RFC3161 timestamps

as obtained from a time server: a program producing a digital signature of the file hash and certified date. Such trusted servers deliver dates precise to the second, that provide an unforgeable complement to the (possibly spoofed) system date inserted in the file. KeeeX uses time servers to create such timestamps.

Public Blockchain writes are forever unforgeable and publicly available.

This is a long-dreamed promise (at least so for Bitcoin to date), that operationally applies to the recording of tiny data chunks (typically file hashes, using the OP_Return statement for instance in Bitcoin, or dedicated smart contracts in Ethereum). Such writes provide the definitive seed of an audit trail that may cascade to millions of collectively "blockchain anchored" files thanks to what is called a Merkle tree: registering the hash of a file that itself recursively contains up to millions other file hashes.

Several options exist to achieve mutualized multi-file anchoring, notably:

- **cryptographic black boxes** (https://opentimestamps.org/ or https://chainpoint.org/ are opensource)
- **textual auditable summary files** (that we call "blockchain certificates").

Blockchain traceability must not come at the expense of human auditability.

Auditing must be doable by a lawyer, without the help of more tools than what is simply available from the blockchain itself and files. In that respect:

- any option requiring the use of a blockchain node web service (Ethereum) is unacceptable, because it may produce corrupt results by a blockchain bug or attack or by lagging the chain
- any option requiring the use of hidden cryptography available by sending the file to a remote server is unacceptable of course
- any option requiring the use of hidden cryptography available by sending the file hash to a remote server is unacceptable too

KeeeX produces human readable keeexed anchoring summaries

These are made available to our customers at the desired rates and under the chosen format, that:

- **can be permanently verified** using any available KeeeX verifier as above stated
- **can be stored as a file on user's premises**
- **have their IDX registered on a public blockchain** (Bitcoin by default, Ethereum upon demand)
- **setup a permanent auditable link** from the blockchain itself to the files
- **only need to be downloaded once by an implementation**, whichever their size, and whichever the number of files involved (which yields enormous savings in large volumes)
- **have their own hashes copied in KeeeX metadata** of concerned files within a protected statement
- **may have their full content stored in the file's KeeeX metadata**
- **are permanently searchable from the blockchain** using an open access indexing explorer. For bitcoin for instance use https://blockchair.com/bitcoin/

We also support ChainPoint

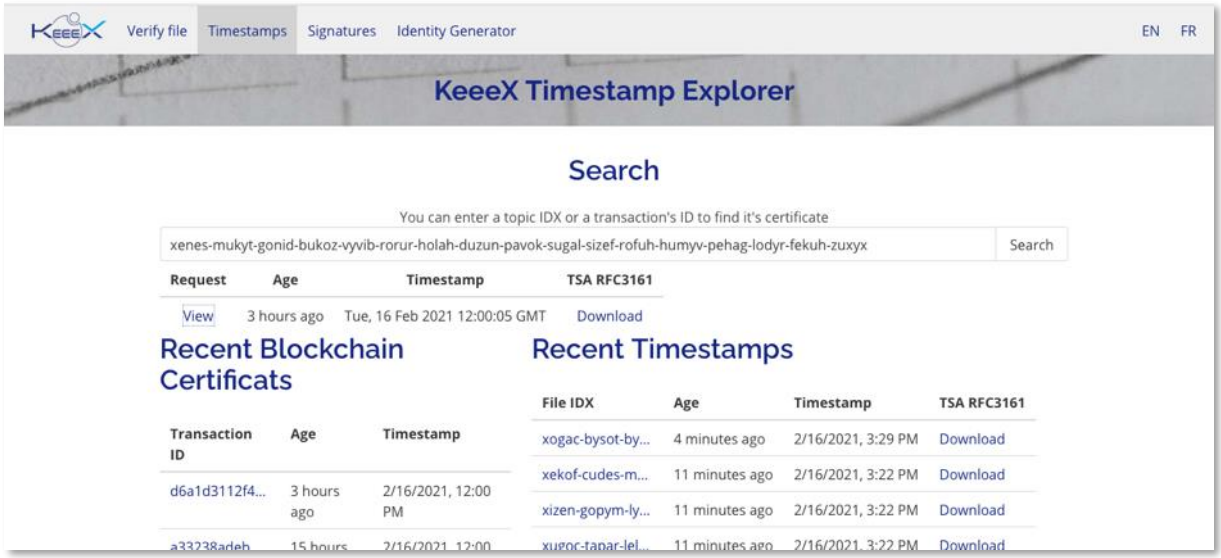
in complement as per customer's request, with the extra feature that finalized ChainPoint proofs are embeddable as protected metadata (cf: https://chainpoint.org/). Note that using chainpoint is most often inappropriate because proofs cannot be visually audited and the purpose of interleaving requests from many sources does not reflect a reality where some users request for anchoring very large batches of files at once.

All the above can be seen at work.

Suppose that you own a file IDX (see xenes-mukyt-…fekuh-zuxyx below) but have not yet downloaded the certificate.

First start with the keeex timestamp explorer:

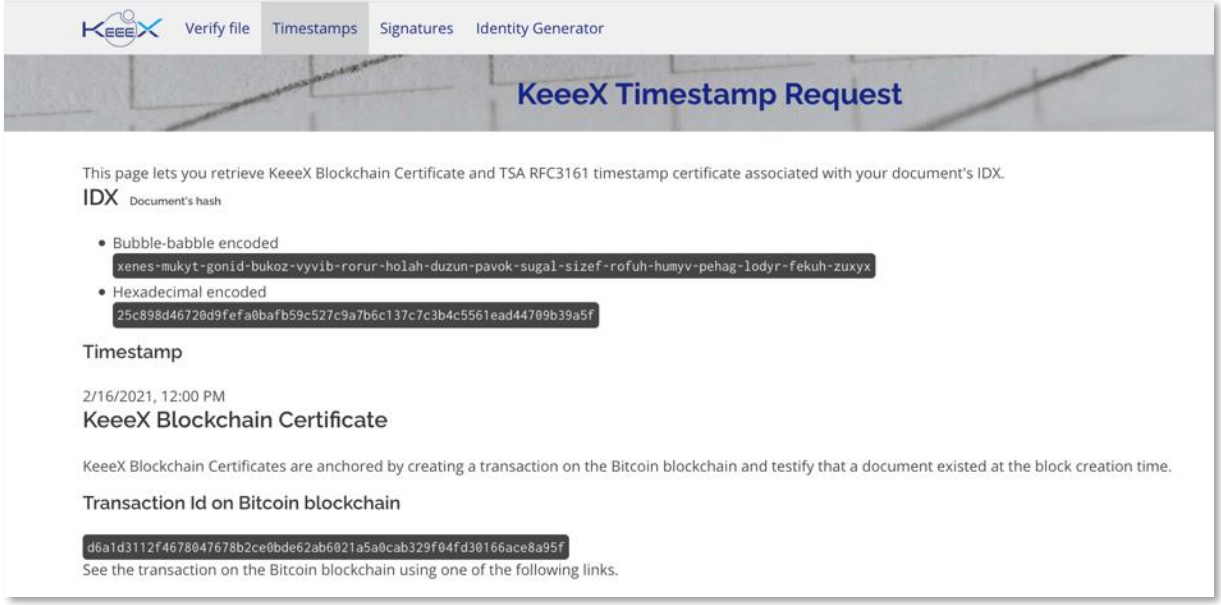
- https://services.keeex.me/timestamps/



Now search a file IDX

You can grab an IDX by clicking on a certificate in left column, then pasting it inside the search box at the top. Also possible is to click a link on the right column but these IDX are not yet anchored. Search yields:

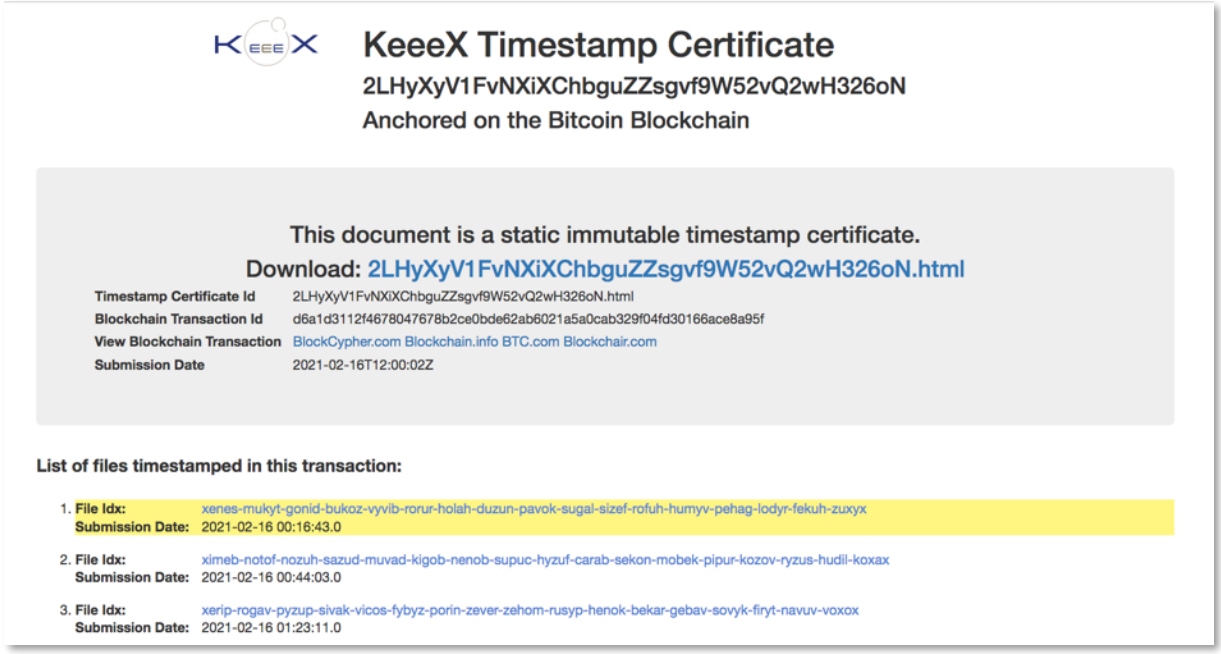
- https://services.keeex.me/timestamps/idx/#idx=xenes-mukyt-gonid-bukoz-vyvib-rorur-holah-duzun-pavok-sugal-sizef-rofuh-humyv-pehag-lodyr-fekuh-zuxyx



Click on the "download certificate" link

The searched item shows up highlighted in yellow color:

- https://services.keeex.me/timestamps/kxc/2LHyXyV1FvNXiXChbguZZsgvf9W52vQ2wH326oN.html#xenes-mukyt-gonid-bukoz-vyvib-rorur-holah-duzun-pavok-sugal-sizef-rofuh-humyv-pehag-lodyr-fekuh-zuxyx



KeeeX's anchoring certificates are plain files

What you get is not a web page!

An anchoring certificate should (or must…) be kept for future user proof records. Most solutions will automatically retrieve and store these documents.

When opened directly from the timestamps explorer or even from a disk file the same certificate file shows up without highlighting:

- https://services.keeex.me/timestamps/kxc/2LHyXyV1FvNXiXChbguZZsgvf9W52vQ2wH326oN.html

Now lookup the certificate idx

This is possible from any Bitcoin explorer indexing OP_Return data in transactions.

- https://blockchair.com/search?q=2LHyXyV1FvNXiXChbguZZsgvf9W52vQ2wH326oN

The blockchain transaction that references the string 2LHyXyV1FvNXiXChbguZZsgvf9W52vQ2wH326oN can be found by clicking the link in the above search results and from other blockchain explorers as well:

- https://live.blockcypher.com/btc/tx/d6a1d3112f4678047678b2ce0bde62ab6021a5a0cab329f04fd30166ace8a95f/

Smart Contracts properties and issues

The blockchain hype has led to the development of solutions for asset management and traceability that have shown their definitive limitations for industrial use, both in the form of public and private blockchains running smart contracts.

What precedes has to do with keeexing and verifying atomic files. A large interest in current applications of blockchain technology concerns verifiable code execution, as was initiated by the Ethereum blockchain. Very often the acronym DLT (Distributed Ledged Technology) is used to denote the applications. What follows does not enter into any detail of blockchain technology, for which documentation otherwise abounds.

A smart contract is code running on a decentralized, consensus based, blockchain network

Such a smart contract can be triggered by calling its functions, some of which may modify the blockchain state (as e.g. "`transfer(A,B,C)`") whereas others just query the state (as e.g. "`getBalance(A)`").

Computation and data storage come at a cost (remember that blockchain state is stored "forever").

Gas denotes the smart contract costs for having the miners run the code and store the data.

Fees relate to processing the transaction alone (writing the results in a block).

The Bitcoin blockchain also supports a form of smart contracts

It should be noted that the Bitcoin blockchain supports decision tree smart contracts, with outstanding features and a different, pragmatic approach (thanks to Taproot, Schnorr signatures, MAST):

- *undistinguishable from normal addresses*
- *hidden from plain sight*
- *only the traversed decision tree is exposed*
- *allowing for time constraints and default fallback behavior*

Bitcoin smart contracts however do not support storing state as Ethereum does. The term smart contract today denotes the variant made popular by Ethereum and the Solidity programming language.

Smart contract execution fees and gas are variable and important, which restricts their use

On public smart contract blockchains like Ethereum, in general, the transaction cost is variable, driven by protocol changes for gas and a market of offer and demand for fees (basically the most funded transactions fire first, while insufficiently funded transactions end up in limbos or fail. This raises considerable issues when going in production.

So far it seems that public smart contract blockchain costs limit their use to applications where users accept:

- to deal with a blockchain wallet
- to pay possibly significant fees

Such use cases are the ones where users enroll into speculative or financially driven interactions, as for instance Decentralized Finance (DeFi), high value asset tokenization or crypto collectibles.

We believe that public smart contract blockchains cannot be used in industry

This is due to:

- **the complexity of managing the related crypto currency** (holding, storing, using)
- **the variability of operational costs** induced by the gas and fees
- **the complexity of computing the gas and fees** that should reasonably warrant that a transaction gets accepted in due time

- **the risk that a transaction gets delayed** for long and unpredictable time (by a suddenly loaded transaction pool shared planetary wide)
- **the choice to make for accessing the blockchain** either via the api towards a mutualized node or via a self-hosted node, which both yield risks of their own kind

plus further cybersecurity and more risks, including the risk that your (own or shared) blockchain node:

- **fails** (a bug in the blockchain, a fork…)
- **stops responding** (a DDOS attack… this happens bi-weekly on Ethereum)
- **produces wrong results** due to a cyberattack or insider job
- **produces wrong results because it lags the chain** (keeping in pace today requires huge amounts of resources)
- **bans your requests**
- …

Private or permissioned blockchains are not great either

Because of these above reasons, many have though that deploying private or permissioned blockchains was the way to go, under the governance and operations of a consortium. Indeed, under such settings there are no gas or fees, and scalability and operational efficacy seems easier to achieve. Unfortunately, things don't get better there since:

- **consortium members must host and run their own nodes**
- **leaving a consortium is almost impossible** if one expects to keep their own data
- **probative evidence remains very weak** with a consortium blockchain since IT people have control and parties can collude (most often such a blockchain would be created by companies competing on the same market)
- **proofs can hardly be shared** to external parties (which most users are)

KeeeX Stories enables scalable, operational, cost effective, auditable traceability

KeeeX's approach to process verifiability combines the best of web service design for distributed applications with the permanent unforgeability and censorship resistance of the Bitcoin public blockchain, used to freeze the valid states of a keeexed sidechain. This results in outstanding scalability, full control over operations costs, full ownership and perfect file level auditability of proof records.

KeeeX has working knowledge of exploiting smart contracts for our customers.

We have created and permanently operate since 2017 Solidity smart contracts in Ethereum for several large industries, on private or public Ethereum network. We have thus identified the major drawbacks of their operational use, in situations that require visibility on costs, auditability, sharing of proofs, predictable scalability.

We may start with the following observations:

- **Today's industry abounds with examples of distributed web services delivering zillions of files** via scalable, elastic infrastructures, linked with high availability and throughput object storage and distributed databases. Should the use of blockchain require to trash such solutions relying upon distributed and cryptographically access controlled servers in favor of ultra-slow and expensive decentralized and consensus based virtual machines?
- **Industrial applications massively do not need to fight against censorship nor to publicly expose the right to perform transactions.** This is so because every such application operates within the boundaries of an access control, whether implemented using JWT, api keys, login / password schemes, Bitcoin or Fido U2F signing capability… Therefore, in general the entry level to a traceability solution needs not be a raw blockchain api.
- **Industrial traceability however requires perfect auditability of proofs records.** We claim that they must be available in the form of files.
- **Industrial blockchain traceability must support the declaration of correctives**. This is so because humans and systems make errors, despite the fact that no write can be removed or erased in a true blockchain grade solution.
- **Industrial blockchain traceability must not depend upon rigid data specifications and programming interfaces**. Instead it must be flexible enough to start operating fast, without waiting for standard specifications or consortium agreements that take months if not years to come out. It must also be flexible and adapt to change easily while still supporting backwards compatibility.
- **GDPR compliance requires that private data remain on their author's or user's premises** and are shared using state of the art, sovereign, access control based distributed data models.
- **The traceability solution must not behave as a marketplace for their user's data.**

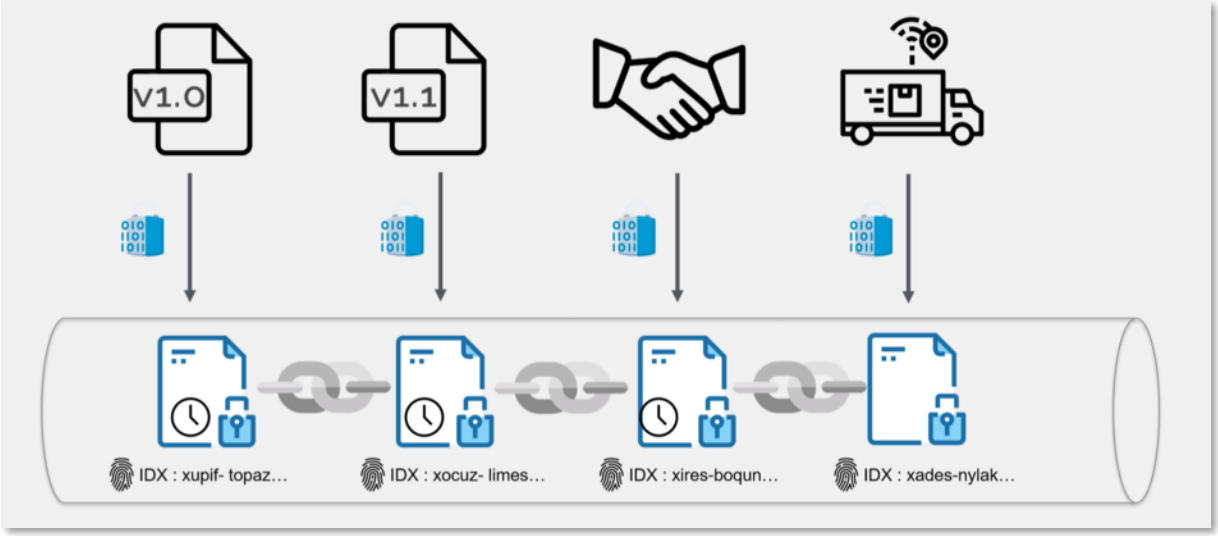
We have some answers.

KeeeX stands as a middle ground approach between full or no blockchain

We leverage the best of our patents, of modern web service design and architecture, and of public blockchain source of audit to deliver scalable yet provably unforgeable traceability solutions.

The KeeeX technology allows for chaining files by their IDX.

We leverage this possibility by building unforgeable, multi signed, sequences of metadata files, that implement unforgeable, time relative sequences of events, data or documents in a registry.



KeeeX Chains behave as Bitcoin sidechains

Bitcoin anchoring allow for freezing and making easily verifiable the valid states of Keeex Chains data structures at regular time intervals. Support for simultaneous use of multiple distinct blockchains is available.

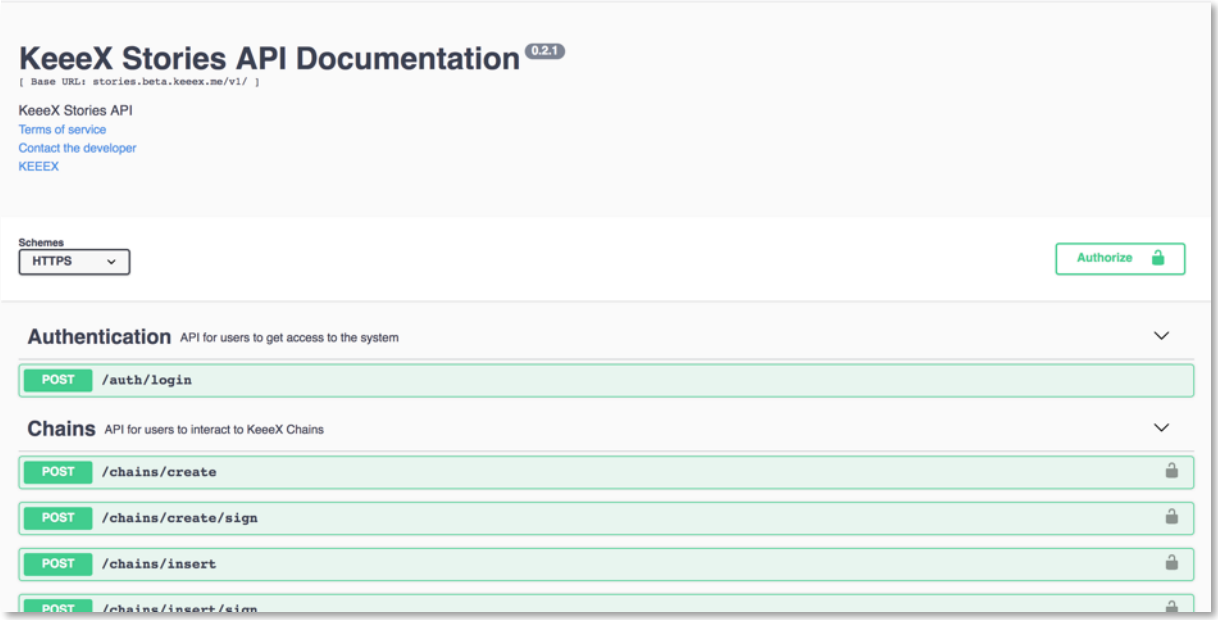
KeeeX Stories exposes a simple API for creating chains

where the genesis element is computed from a given reference, in the context of a community and scope, which occurs as a very generic use case. This allows for delivering extremely flexible, custom traceability implementations.

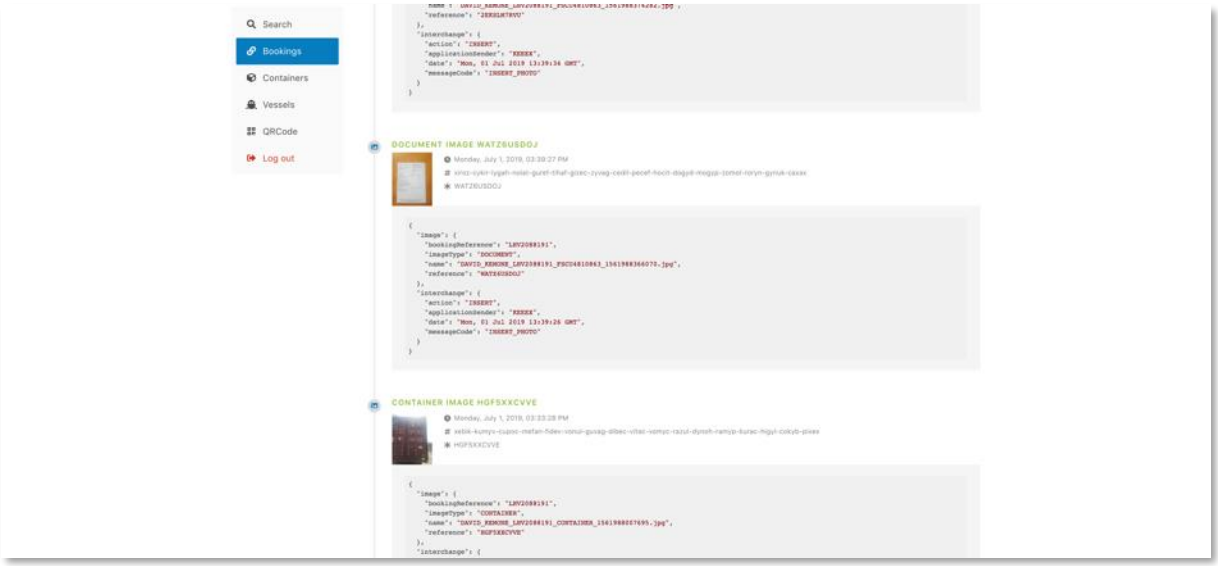
Simple live swagger documented are available for all KeeeX solutions.

Below is the example of the KeeeX Stories api swagger:

- https://stories.beta.keeex.me/documentations/v1/



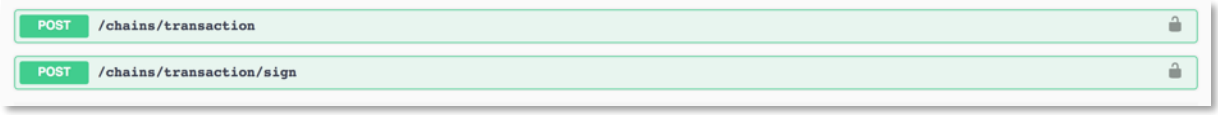
Next Figure illustrates the example of a story pertaining to a logistics booking, as presented in a web portal.



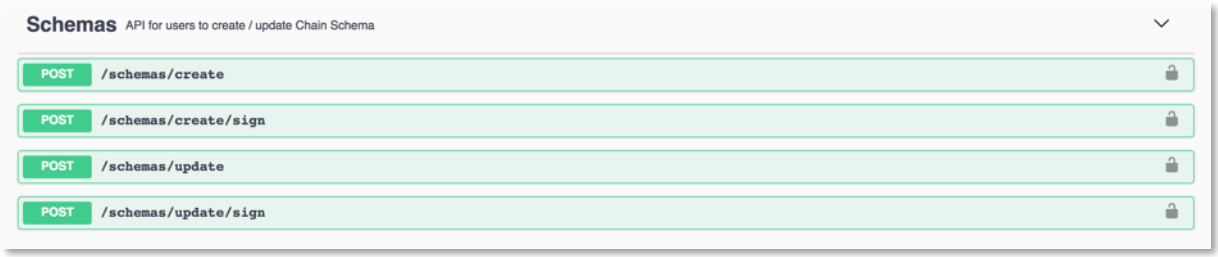
Writes to stories and chains

- **are digitally signed** using Bitcoin signatures plus X509 server certificates when required.
- **can be transactional over multiple simultaneous chains** thereby ensuring operational continuity without the risk of dealing with corrupt data.
- **can be subject to arbitrary user defined conditions**. Most importantly, virtual transfers of ownership or responsibility can be acknowledged by dual signatures where a first Story entry denotes "A transfers to B" (signed by A) and no other entry can occur before "B acknowledges transfer from A" (signed by B.
- **can be restricted to data matching published, signed and versioned keeexed data schemas.** Our best practice is to open source public data schema specifications. Such specifications can be fetched, branched, merged as standard open source code (example below)
- **cannot be modified or deleted.** This is due to blockchain control and cryptographic interleaving of elements. However, the standard user makes errors.
- **can implement edits or intent to delete previous elements**
- **cleanly separate public metadata (available from an explorer) from private Data**

Transactional writes are available at api level

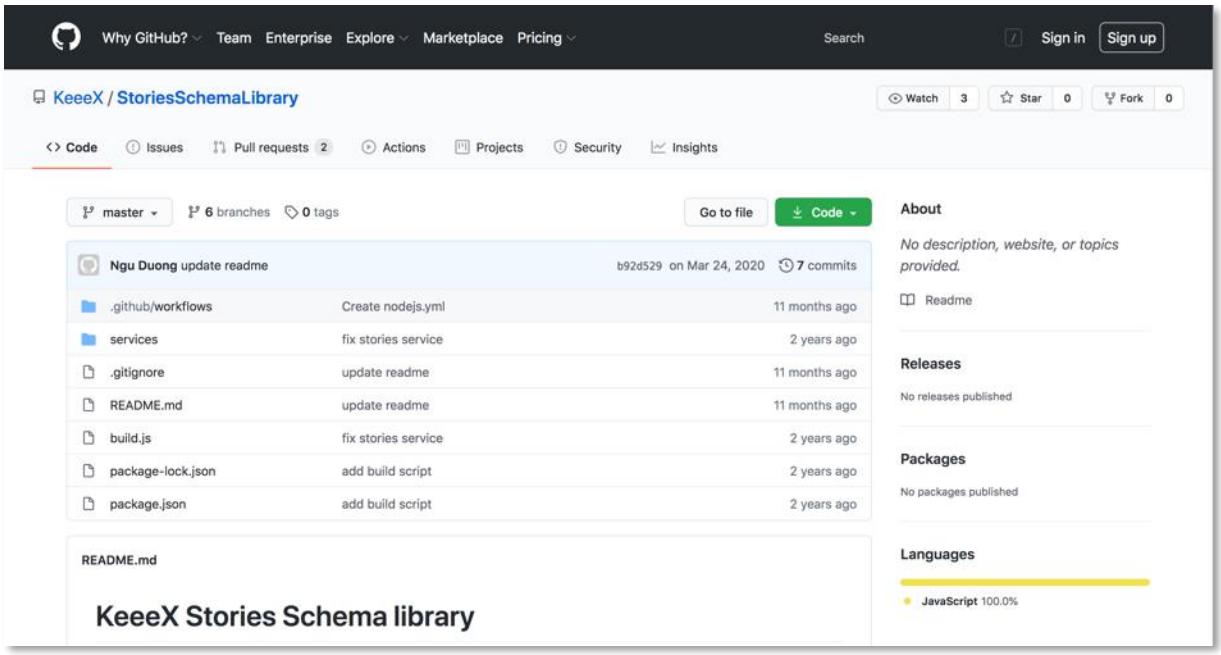


Data schemas are fully integrated into the KeeeX Stories api



A repository for supply chain related data schemas can for instance be explored at:

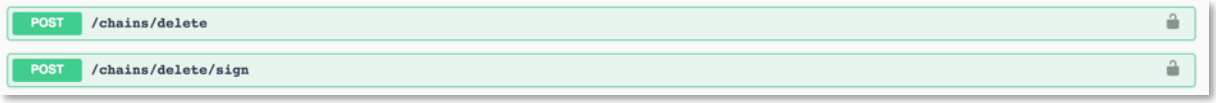
- https://github.com/KeeeX/StoriesSchemaLibrary



Private data can, under entire owner control, be:

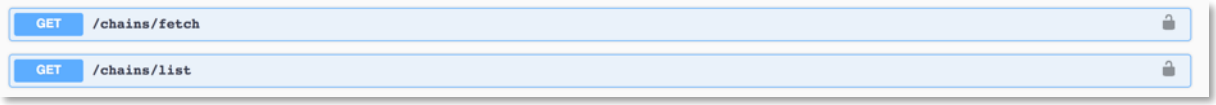
- **self-hosted**
- **deleted** (api automates a specific write)
- **versioned** (also requires a new entry)
- **served according to owner's access control, legal rights, business model** (which means that only the accredited users will be granted access)

Declaration of private data deletion is accounted at api level



A given KeeeX Story can be downloaded as a zip

This has many applications, including for further auditing if required. Read accesses are otherwise paginated.

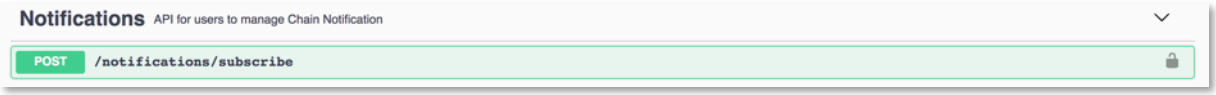


A KeeeX Chains server can be hosted on premises.

A KeeeX Stories proxy can and should perform as a sidechain observer for their own interest.

This involves storing a local copy of the elements of interest and removes the dependency from KeeeX.

A notification service allows for seamless integration by webhook subscription



Summary

We have detailed the properties that ensure that

- **Keeexed data is under full user control**, on their premises, thus matching the highest possible sovereignty and GDPR compliance levels
- **KeeeX proofs are under full user control**, either ascii embedded inside files or available as human auditable files and verifiable without any limitation or duration, with or without KeeeX, using off the shelf open source algorithms, when necessary almost manually and without KeeeX. Forever.
- **KeeeX proofs may address the highest desired regulatory compliance levels by use of eIDAS certificates**
- **The technology will be available forever**, as well as many programs (including the verifier)
- **The solutions may scale up to arbitrary levels**
- **Simple generic APIs plus adaptable and verifiable data schema specifications** warrant independence from standards, speed of development, extreme flexibility, maximal interoperability