3NWeb

Approach and implementation overview

Contents

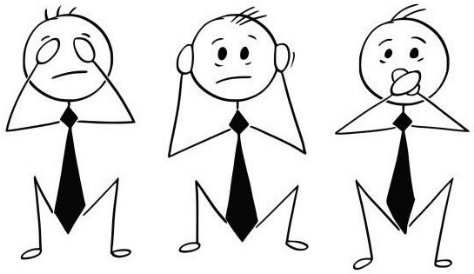
I. Ecosystem and protocols

I.1. Servers, trust and 3N principle

Every story about privacy violation or a leak of sensitive data has a server in it. Either server with users' content and observed metadata is hacked into. Or server's owner, a service provider, is forced to give away users' data. Or service provider sells the data. In any case, server can't be completely trusted.

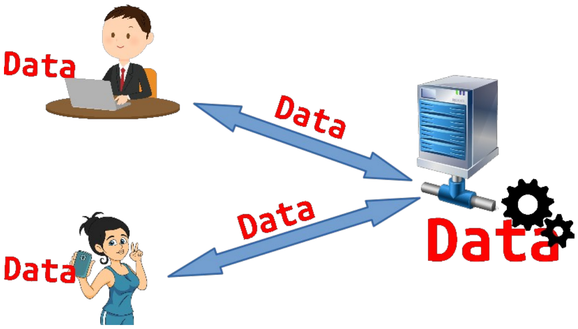
But we want to create an ecosystem on the Internet which users can trust, which users can use for different tasks. Let's formulate client-server trust relationship as an actionable principle:

1. No plain text user content should be given to server.
2. No unnecessary metadata should be present or generated in client-server interaction.
3. Nothing can be abused on the server, when the first two criteria are met.

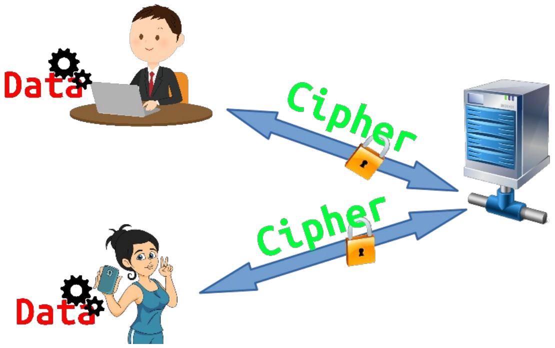


One may think about 3N principle as a manifestation of the Principle of Least Authority. When client gives data to server, it implicitly gives server ability and authority to do anything with provided data[1]. And when client wants to limit this implicit authority, it should give server no data.

I.2. Data and computation models



Server-side computation model



Client-side computation model

3N principle is slightly incompatible with how cloud-backed applications are usually written today. Many applications place data processing on a server side, requiring clients to give data to server in plain text. Connection to the server may be encrypted in transit, but data is decrypted on the server, and it is essentially a plain text for the server. This data transfer goes against 3N principle, leaving users vulnerable.

To follow 3N principle, data processing must happen on client side, where data can be in plain text. Any data that goes via server should be end-to-end encrypted, so that only users have decryption keys.

Evidently, centralized server-side computation is simpler to code than decentralized client-side computation. Also, some tasks like processing of payment transactions seem

[1] It seems that the third-party doctrine in USA was argued into existence the same way. Quoting the Supreme Court in United States v. Miller, 425 U.S. 435 (1976): "The depositor takes the risk, in revealing his affairs to another, that the information will be conveyed by that person to the Government."

impossible without third party side computation, be it banks' servers or bitcoin network of servers. Yet, many tasks can be performed on a client side.

Decentralized client-side computation needs some utility level protocols to allow data and synchronization flows between clients via server. When protocols are universal, server can't know which applications run on clients, and it can't censor them[2].

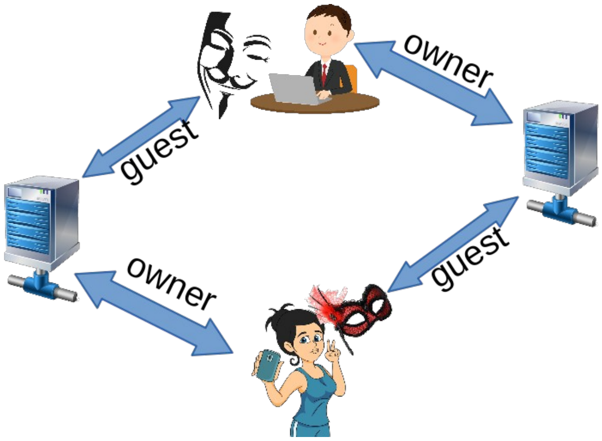
I.3. Metadata reduction

In server-client architecture server acts as a place with some resources (files, event streams). In some services like DNS, clients only consume the resources. Services like WordPress application allow some clients to administer resources, while everyone else can only consume them. Clients with administrative privileges must prove their identity to server, while other clients can stay anonymous.

Let's call privileged clients owners, as they own created resources[3], and call other clients guests.

Protocols should keep guest clients anonymous. And in some cases owner should be able to create access tokens for guest clients, keeping them anonymous to the server, while realizing access control function.

This approach reduces amount of metadata inherent in protocol use. Resource creation implicitly connects resource to the owner, but everyone else should stay anonymous to the server, so that it can't compile pairs of who deals with who.



Guest clients anonymous to servers

Servers may log IP addresses of clients, but this is mitigated by clients connecting to servers via either VPNs or Tor to hide IPs, associated with users.

I.4. Federation, classical and web-style

Wikipedia carries the following description of federation in information technology: "A federation is a group of computing or network providers agreeing upon standards of operation in a collective fashion."

From user experience perspective, federated service means that users on one domain can deal with users on another domain[4]. There are at least two architectures that deliver federated user experience.

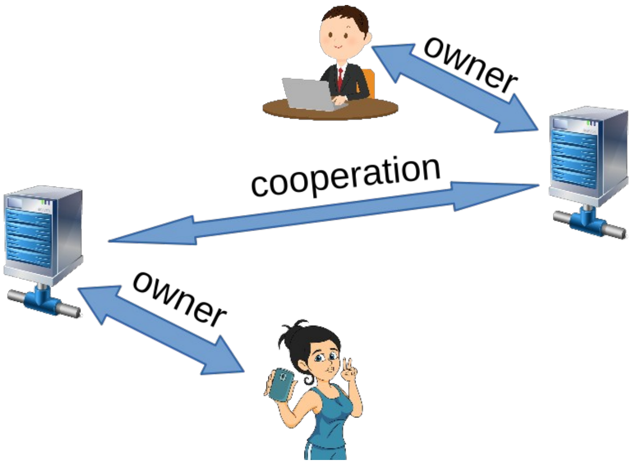
[2] There are also two economic effects here. With utility service provisioned by user, app developer only needs to provide code to run on client side. In contrast, when computation runs on server side, developer additionally needs financial resources to contract cloud resources and to keep 'em running. And when original developer stops feeding the cloud, users loose their app functionality. Thus, client-side computation model with utility platform-like services procured by users (a) opens up market to more app developers, and (b) ensures usefulness of app even when developer is gone.

[3] In service provider practice these owner clients will be associated with an account that pays for the services. So, word owner may also have an economic meaning.

[4] Very often domains are owned by service providers, and federation translates into competition in a market.

First federated architecture can be called classical, cause it appears in old protocols, like SMTP (1982). In classical federation client of a domain talks to the server responsible for client's domain. When client wants to send message to any other domain, it gives message to server, and tells it who the recipient is, so that server can deliver this message on behalf of the client.

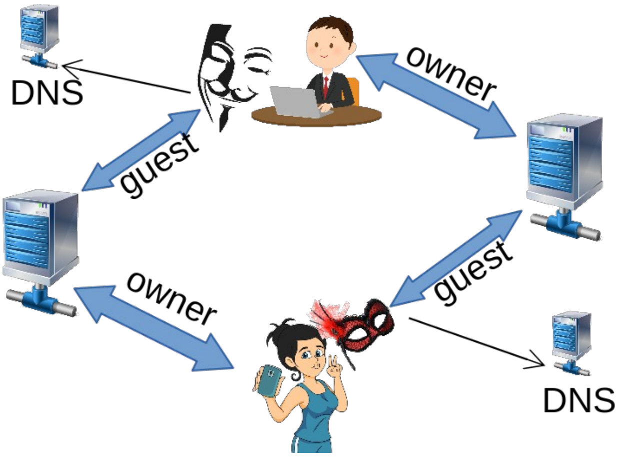
Unfortunately, classical federation requires clients to tell server who they deal with. Even if clients encrypt end-to-end their content, clients give servers communication metadata[5], violating 2nd N of the 3N principle. More so, classical federation assumes active cooperation between servers of different domains. Dominant vendor may stop cooperating with smaller domains, forcing users to abandon smaller providers.



Classical federation

There are no fixes to these two defects of classical federation. But people are fixated on classics, and as a result, there is a debate[6] going on, suggesting a false choice between privacy of users and freedom to choose service providers for the same functional utility. This choice is false, cause there is another federation style, web-style federation.

Ever since 1990's web uses two protocols for operation: DNS and HTTP. Naming (DNS) provides unique identification of resources on the Internet. Web client gets server's IP address from DNS and talks to server directly with HTTP.



Web-style federation

If resource is a message box, identified by address Romeo@Montague.domain, Juliet's client can anonymously get IP from DNS for Montague.domain server, anonymously connect to it and leave a message in Romeo's message box. When Romeo decrypts message with Juliet's key, he is assured that this message is from Juliet. Process repeats symmetrically when Romeo's client delivers message to Juliet@Capulet.domain. Essentially, this is a base for messaging protocol that has web-style federation and follows 3N principle.

We use 3NWeb name to identify a set of 3N-respecting web-style federated protocols that provide a practical platform for writing modern Internet-connected apps with computations on a client side. Currently 3NWeb has identity protocol MailerId, messaging protocol ASMail, and storage protocol 3NStorage.

[5] Infamous quote of the former NSA and CIA director Michael Hayden: "We kill people based on metadata".

[6] In late 2010's: https://signal.org/blog/the-ecosystem-is-moving/ and in 2020: https://matrix.org/blog/2020/01/02/on-privacy-versus-freedom/

I.5. Users and services

I.5.1. User ids

Users need a global identification system. In email[7] we have globally unique identifiers in a form *<user-id>***@***domain*. DNS system ensures uniqueness of a domain. And *<user-id>* identifies one user in the domain.

In 3NWeb user identities follow the same principle of tying users to domains.

User identifier string in 3NWeb has form *<user-id>***@***domain*. Part *<user-id>* can have any characters, except character **@**. Before ids can be compared, they are brought to a canonical form by removing all white space and low-casing all characters.

For example, *Bob Marley* **@***3nweb.com* (note spaces in id) and *bob marley***@***3nweb.com* correspond to the same canonical id *bobmarley***@***3nweb.com* that uniquely identifies exactly one user.

I.5.2. Locating user services

User's 3NWeb services are listed in DNS `TXT` records of user's domain.

If domain is an onion domain[8], DNS is not used, and respective Tor service should have well-known resource[9], `/.well-known/`, with service records.

Domain's service record identifies server that services users of this domain. Records generally use very flexible URI format starting with authority part[10]. This allows to use service providers, or Tor services, or use ports and paths when many services are packed into the same home server box.

As an example, *3nweb.com* domain has following `TXT` records in DNS:

- for MailerId service: `mailerid=mailerid.org`
- for ASMail service: `asmail=3nweb.net/asmail`
- for 3NStorage service: `3nstorage=3nweb.net/3nstorage`

I.5.3. Identity protocol MailerId

Just having an identity is not enough. There needs to be a way to prove one's own identity, for example, when logging into some service. For this we need an identity service.

Around 2011 Mozilla developed non-tracking identity protocol BrowserId, later called Mozilla Persona[11]. Persona authentication system had a cryptographic process for non-tracking verification of identity, embedded into browser use case, and connected to a particular Verified Email Protocol.

[7] Address specification in RFC 5322: https://tools.ietf.org/html/rfc5322#section-3.4
[8] Special .onion domain, RFC 7686: https://tools.ietf.org/html/rfc7686
[9] Well-known resource, RFC 5785: https://tools.ietf.org/html/rfc5785
[10] URI parts, RFC 3986: https://tools.ietf.org/html/rfc3986#section-3.2
[11] Mozilla Persona: https://en.wikipedia.org/wiki/Mozilla_Persona

Hackers exploited browser and showed how Persona server may track users. But BrowserId's main cryptographic process is solid. We use it in a protocol, which we call MailerId.

MailerId server serves two types of clients: (a) users, and (b) relying parties.

User's client generates a random pair of keys that can be used for signing. Then client logs into MailerId and asks to make a certificate tying user's id to a public signing key. This stage is called provisioning: client provisions a user certificate, valid for a day.

With certified signing key, user client can sign assertions to log into other sites/services. In this case site/service is a relying party that checks signatures in assertions and certificates all the way up to some root MailerId certificate. Then relying party anonymously retrieves root certificate from the MailerId server for a final validation.

With certified signing key, user client can also sign a messaging key. In this case, some other client becomes a relying party. It verifies a chain of signatures, and anonymously retrieves root certificate from the MailerId server for a final validation.

Anonymity of relying party and sameness of root certificate retrieval ensures that MailerId server knows neither which user identity is used, nor where it is used.

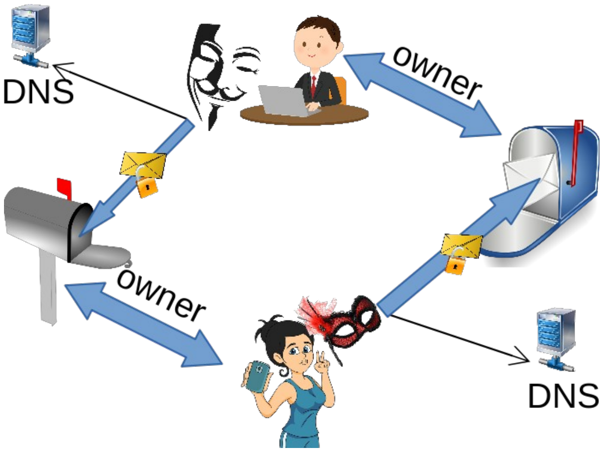
I.6. Asynchronous messaging in 3NWeb

I.6.1. Authenticated Secure Mail protocol (ASMail)

Asynchronous messaging protocol in 3NWeb is called Authenticated Secure Mail, or ASMail for short.

ASMail server has two types of clients: (a) owners of inboxes, and (b) senders who bring messages to inboxes.

Sender encrypts message objects to public keys known only to recipient (end-to-end to encryption) and anonymously leaves message in recipient's inbox. Recipient learns sender's identity after decryption of a message. This way sender identity is always authenticated to recipient, and is unknown to server.



Authenticated Secure Mail

There are a few technical details to ensure security, some of which are:

- ASMail protocol implements key rotation.
- There is a concept of introductory keys to start messaging. These can be distributed via ASMail server, or out of band.
- Introductory keys are signed with MailerId signatures.

- Sender uses an ephemeral introductory key, sticking all identity and certificate information inside of encrypted envelope of the first message.
- Servers don't dictate users what encryption algorithms are used for messages.

Owner of the inbox may open an event stream to be notified about new messages as soon as they are delivered. This provides an experience of almost instantaneous message delivery. But, unlike realtime video and audio chats, ASMail is an asynchronous messaging protocol.

I.6.2. ASMail message format

We should talk about two perspectives: (a) what server sees, and (b) what format users see in unencrypted form.

Server is given one or many opaque encrypted binary blobs. Each blob is labelled with some random object id. One object is marked as main.

Main object has a section with encrypted key that open the rest of it. This section is encrypted with public keys that sender and recipients use. In other words keys that open message objects are randomly generated and are not connected to employed public key encryption.

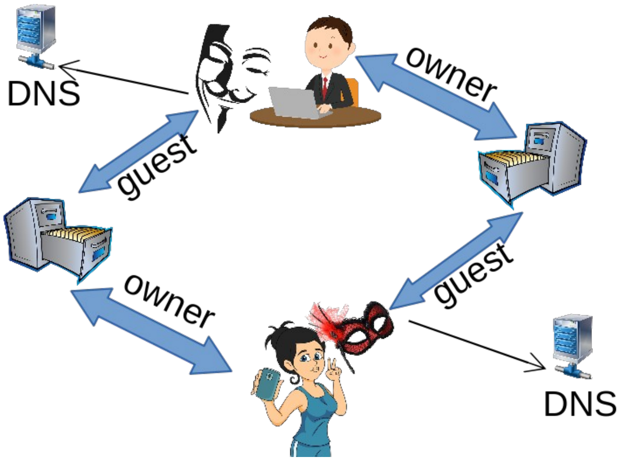
Unencrypted main object is a JSON with different fields. Some of these can be JSON representation of folder with attachments. Folder carries keys to open other child objects in the message, and when those objects are folders, they contain keys to open other child objects in a cascading manner. From an application point of view, attachments folder is a self-enclosed file system that can have directories, files of any size. Sender can literally deliver a multi-gigabyte message, when inbox size and recipient's preferences allow it. This is why ASMail also stands for and is pronounced as AweSome Mail.

ASMail uses XSP format to pack AEAD ciphers. This format is borrowed from 3NStorage, where it is used more extensively.

I.7. Storage

I.7.1. 3NStorage protocol

3NWeb storage protocol is called 3NStorage. This protocol provides minimal server functionality that lets client side build multi-device synchronized file systems. Clients store in 3NStorage randomly named opaque encrypted binary blobs. When decrypted on client side, objects become folders and files, but server doesn't have access to file tree structure, it doesn't know names of files. File system metadata is never given to 3NStorage server.



3NStorage

The protocol has a concept of object versioning. This lets clients track changes in storage and perform synchronization steps accordingly. 3NStorage server also provides an event stream for a push-style change notification.

Every storage is owned by some user. To share access, user client software will create randomly named sharing accounts with limited access to encrypted blobs. Credentials to new account, together with some key material, and together with pointer to object that hold root of a shared file tree, constitute a structure that allows other clients to access shared folders and files while staying anonymous to the server.

I.7.2. XSP format for NaCl-encrypted files

Secret key authenticated encryption is used to encrypt storage and message files. NaCl library[12] is used. This library was put together by cryptographers. NaCl’s API surface limits possible mistakes when using crypto, unlike other libraries that provide bare crypto primitives to developers[13].

NaCl library produces nice cipher package, but it is convenient by itself only with small files while bigger ones need to be split into fragments. Thus, there is a need for format that addresses other requirements. Here is the laundry list:

- file should be split into segments;
- it should be possible to randomly read segments;
- segment should have poly+cipher, allowing to check segment's integrity;
- it should be possible to randomly remove and add segments without re-encrypting the whole file;
- segments' nonces should never be reused, even when performing partial changes to a file, without complete file re-encryption;
- it should be possible to detect segment reshuffling;
- there should be cryptographic proof of a complete file size, for files with known size;
- there should be a stream-like setting, where segments can be encrypted and read without knowledge of a final length;
- when complete length of a stream is finally known, switching to known-length setting should be cheap.

In a client-server environment it is possible to have two different clients uploading same sections of objects with different data. Usually same section implies same nonce, leading to nonce reuse. That’s why an XSP format[14] was developed, specifically hardened for client-server environment.

[12] NaCl library: https://nacl.cr.yp.to/
[13] Comment on crypto APIs: https://twitter.com/matthew_d_green/status/613703051149463553
[14] XSP specification and TypeScript implementation: https://github.com/3nsoft/xsp-files

II. Implementation

Libre Software implementations of client and server sides of 3NWeb ecosystem are worked on today, and are available in git projects under https://github.com/3nsoft/ group account.

II.1. Spec server

Project https://github.com/3nsoft/spec-server develops an implementation for all 3NWeb protocols. Protocols are defined over HTTPS transport. HTTP backends can be written in many ways. This implementation aim to provide an all-in-one service package. Current code is based on NodeJS.

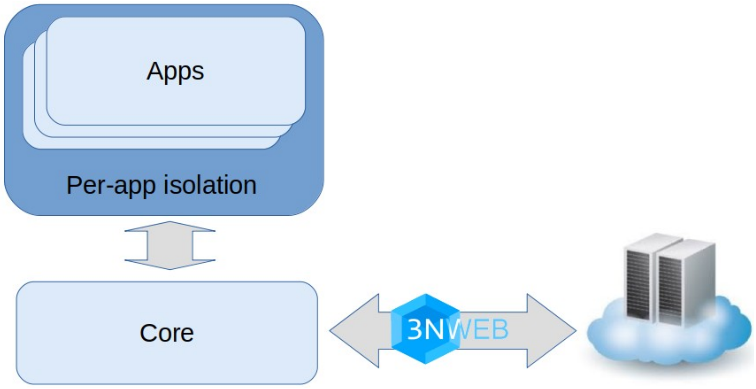
Besides providing an implementation, spec-server project crystallizes a specification test suite that can be used as a functional validation of conformance to 3NWeb standards, when these are put forward.

Spec server is also used in integrated tests of a client platform.

II.2. Client platform

Project https://github.com/3nsoft/core-platform-electron develops an implementation of a client side platform that can run different apps.

Client platform has a core that is responsible for credentials, encryption, communication with servers. Core provides a simple API surface to apps. For example, when an app receives a message, it is never given in a raw form, app sees all attachments as a file system. Similarly, apps see synchronized storages as file systems.



Current implementation is based on Electron[15] framework. Core uses main process, while apps run in constraint renderer processes with associated sandboxed browser window.

Core injects into app’s context different capabilities like app storage, user storage, messaging, etc. Set of capabilities depends on app’s manifest. Core allows app to connect only to sites in app’s manifest. This architecture of giving only necessary capabilities follows the Principle of Least Authority, cause every capability carries an implicit authority to use it.

[15] Electron framework: https://www.electronjs.org/

Client platform project de facto develops a standard for apps, called 3NWeb apps. Like web apps[16], 3NWeb apps are written as HTML+CSS+JavaScript set that browser window can run.

Client platform will have code of 3NWeb user apps in a synchronized storage. In this way the system transcends boundaries of single device, allowing to have chromebook-like user experience without giving away one's digital life to cloud vendor.

II.3. 3NWeb apps

3NWeb app is a set that runs in browser window. At this moment it is HTML+CSS+JavaScript and WASM[17]. Core injects capabilities as `w3n` into `window` object. TypeScript definitions in https://github.com/3nsoft/core-platform-electron/tree/master/src/api-defs describe capabilities API.

Project https://github.com/3nsoft/personal-client-desktop carries a messaging application with mail-like interface. This code shows that 3NWeb app can be written without ever touching intimate details of 3NWeb protocols and related encryption.

II.4. Present and Future

Foundational concepts of 3NWeb work and are already implemented in code. 3NWeb works now on desktop clients (https://3nweb.global/downloads/), and there is room for it to grow in the future:

- We need to create more 3NWeb apps to serve end user needs, like calendar, messaging with chat-like user interface, and so on.
- Different use cases in 3NWeb apps may eliminate areas for an additional tweaking of both 3NWeb app API and 3NWeb protocols.
- We need to port client platform to Android and iOS mobile operating systems. Note that 3NWeb apps only need to adopt responsive design[18] to be more convenient on handheld devices.
- As WASI[19] matures, 3NWeb app API will include other options for writing cross-platform 3NWeb apps.

[16] Web application in wikipedia: https://en.wikipedia.org/wiki/Web_application
[17] WASM in wikipedia: https://en.wikipedia.org/wiki/WebAssembly
[18] Responsive web design in wikipedia: https://en.wikipedia.org/wiki/Responsive_web_design
[19] WASI, WebAssembly System Interface: https://wasi.dev