Workspace: A Data-Oriented, Decentralized Collaboration Web App

Tianyuan Yu
UCLA
tianyuan@cs.ucla.edu

Xinyu Ma
UCLA
xinyu.ma@cs.ucla.edu

Varun Patil
UCLA
varunpatil@cs.ucla.edu

Yekta Kocaoğullar
UCLA
ykocaogullar@g.ucla.edu

Lixia Zhang
UCLA
lixia@cs.ucla.edu

ABSTRACT

Remote collaborations are one of the popular applications on today's Internet. However, the existing collaborative apps are all hosted on the cloud servers, largely owned by a small number of providers. In these apps, although contents are generated by end users, they can only collaborate through connections to clouds. This paper explores an alternative path to building collaborative applications by making use of named, secured Web Objects (SWO). Inspired from the original data-orientation vision of web, we developed a decentralized collaborative application dubbed *Workspace*. Workspace users establish trust relations among each other, secure their data productions directly, exchange SWO through rendezvous points, and support users with intermittent connectivity.

CCS CONCEPTS

• **Information systems** → **Web applications**; **Internet communications tools**; • **Networks** → **Network design principles**.

KEYWORDS

Decentralized Web, Local-First Software, Information-Centric Networking

ACM Reference Format:
Tianyuan Yu, Xinyu Ma, Varun Patil, Yekta Kocaoğullar, and Lixia Zhang. 2024. Workspace: A Data-Oriented, Decentralized Collaboration Web App. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

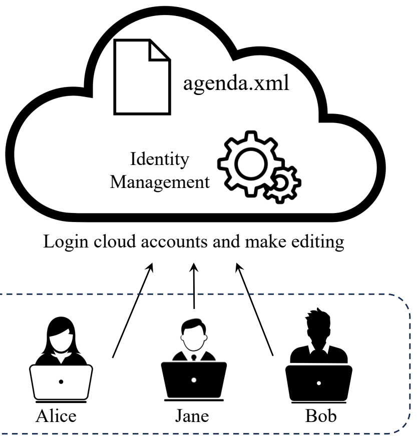


Figure 1: Alice and Bob collaborate on a shared document hosted by clouds

1 INTRODUCTION

The success of clouds has centralized web-based applications onto cloud servers, which are largely offered by a small number of cloud providers. Although end users generate a large percentage of web contents, they do not have direct control over their data, nor can they collaborate directly on their own contents without the cloud servers.

Figure 1 depicts a meeting agenda creation scenario with Alice and others being the meeting organizers. Alice sets up a shared online document "agenda.xml" and invites her co-organizer Bob join the collaborative editing. Accepting the invitation, Bob needs to login to his cloud account in order to make changes, even when they are in the same office. In this example, all contents are generated by the users, who have authenticated each others through out-of-band trust relations. The users' devices, their phones, tablets, and laptops, can also directly reach each other via local WiFi or bluetooth connectivity. Yet the existing web app implementations do not allow them to directly exchange document in a decentralized, cloud independent way.

In a companion paper, we defined Secure Web Objects (SWO) [9] as semantically named and cryptographically Secured Web Objects. We believe that SWO opens a new opportunity to revisit the collaborative web application design. SWO enables web developers to write decentralized applications as producing, exchanging, and consuming SWO.

We believe an ideal collaboration environment should be a shared cyberspace where people gathered by social relations benefit from joint efforts. The shared cyberspaces naturally form *workspaces* among different groups of people. In each workspace, collaborators work on the same set of public documents and invite others to join the efforts equally. Everyone sees the content updates made by others. More importantly, people should be able to work without connecting to centralized servers. Whenever a rendezvous point among collaborators is available, SWO that include document changes shall be exchanged among those rendezvoused collaborators.

This aforementioned decentralized application model requires a system design that provides the following functions:

Conference'17, July 2017, Washington, DC, USA
2024. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

- **Semantic user identity:** Identifying users (*i.e.,* collaborators) by application-wide semantic identifiers, which can be authenticated without connecting to cloud.
- **Rendezvous:** Providing a rendezvous point to let users reach each other and exchange SWO without going through the cloud.
- **Asynchronous communications:** Asynchronously broadcasting SWO to other workspace users, so that users are not required to be reachable at the time when new SWO are produced.
- **Update consistency:** Consistently updating the documents for workspace users based on ongoing changes made by others, so that eventually all users share the same view of the workspace.

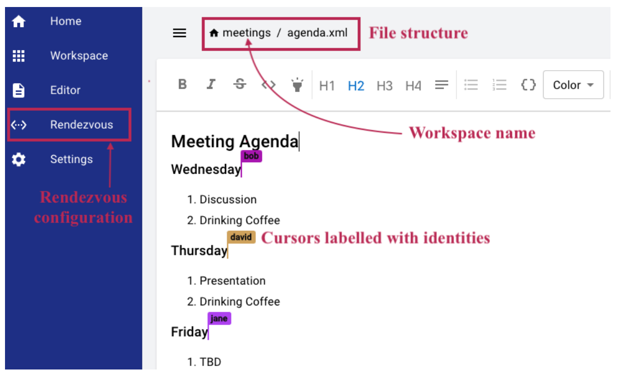
This paper presents an demonstrative example of how the SWO concept enables decentralized collaborative application design, by describing a collaborative text editing application we have built, named *Workspace*. The structure of this paper is as follows: §2 explains how we implemented these functions and how we implement the required functions within Workspace. §3 concludes the work.

2 FROM SECURE WEB OBJECTS TO APP

In this section, we describe how we develop the application *Workspace* following the decentralized collaboration model defined earlier. We first give an overview of Workspace from user perspective, then explain how the system functions are implemented.

2.1 Workspace Overview

Alice wants to join the workspace *meetings* and edit on a shared document "agenda.xml". She starts by claiming her *meetings* identity and mutually authenticate with her colleagues (§2.4). Afterwards, her Workspace instance searches for usable rendezvous point for Alice and connects to the closest available one (§2.5). On the rendezvous point, Alice's Workspace instance exchanges SWO with other users in the *meetings* workspace (§2.2). Receiving SWO form others, Alice verifies and decrypts "agenda.xml" document's change made by other users (§2.3), and adds a new line to "agenda.xml", which is also represented in SWO and exchanged with other users on the rendezvous.



Meeting Agenda

Wednesday

1. Discussion
2. Drinking Coffee

Thursday

1. Presentation
2. Drinking Coffee

Friday

1. TBD

Figure 2: Alice joins the "meetings" workspace to collaboratively edit the agenda with her colleagues

2.2 Shared Files to Shared Data Structures

The *meetings* workspace consists of multiple folders, each can contain multiple files. Workspace represents this file hierarchy as a collection of shared data structures, referred to as *document*. From Workspace's perspective, meeting organizer collaboration is driven by users making changes to the *meetings* document. The file creation of "agenda.xml" is a change made to the file index structure, and the text Alice added to the agenda are a series of changes on text structure. By exchanging updates, all users will get everyone else's updates made to the shared document.

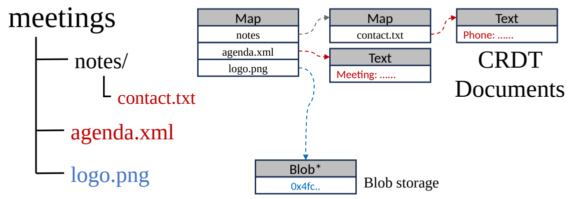


Figure 3: Workspace represents file structure as a document, which is a collection of shared data structures.

Content consistency is an important requirement that collaborative applications need. If multiple users are editing an agenda item concurrently, the system needs to ensure that eventually all users in *meetings* see the same agenda. Workspace achieves this eventual content consistency by defining the shared data structures with Conflict-free Replicated Data Types (CRDT) [6], so that concurrent updates from different users can be merged without conflicts.

Different files and file structures are mapped to different data structure types of CRDT, which support different operations. Folders are mapped to *Maps*, which maintains a mapping between files and CRDT data structures. Text files are mapped to CRDT *Texts*, allowing real-time collaboration. Other types of files, which do not support real-time collaboration, are transformed into versioned, immutable binary string.

2.3 Secure Web Objects Driving Workspace

CRDT makes the data structure(s) changes the basic object that drives the Workspace application logic, and we organizes those objects under URI-like namespaces with the concept of SWO. The SWO namespace starts with each user's Workspace identity (discussed later in §2.4) and is followed by a sequence number, indicating the number of updates have been made by this user. Figure 4 gives an example of the Workspace naming convention, where Alice's changes are named sequentially under her *meetings* identity prefix.

Workspace secures SWO by encrypting and signing them with keys, and keys themselves are also named and secured as SWO. As Figure 4 shows, when Alice's Workspace instance signs SWO with her private keys, the signature can carry a URI-like key locator point to her certificate SWO. Data content are also encrypted by Alice's key with a key locator pointing to the encryption key SWO.

Workspace drives its application logic with synchronizing SWO publications in the *meetings* namespace, and decouples SWO content security from communication channel security. Considering

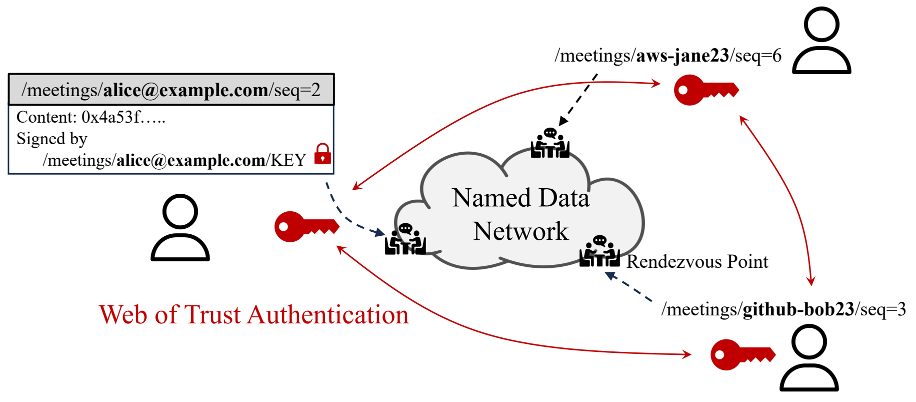

/meetings/**alice@example.com**/seq=2
Content: 0x4a53f…..
Signed by
/meetings/**alice@example.com**/KEY
/meetings/**aws-jane23**/seq=6
Named Data Network
Rendezvous Point
/meetings/**github-bob23**/seq=3
Web of Trust Authentication

Figure 4: A scenario where three users exchange SWO in the "meetings" workspace. Through peer authentications and the web of trust, Bob and Jane are able to verify Alice's SWO, which includes the latest change she made to the CRDT document.

authenticated keys and certificates (discuss shortly in §2.4) as SWO no only simplifies the Workspace communications, but also open opportunities to specify schematized security policies as name mapping enforcement between SWO names (for example, forcing SWO must be signed by the certificate SWO under the same user prefix).

2.4 Security Bootstrapping

Workspace uses hierarchical, semantically meaningful names as identifiers. As shown in Figure 4, Alice's Workspace identity is "`/meetings/alice@example.com`", which concatenates her email with the Workspace context.

Alice's public key is wrapped into a self-signed certificate, bound to his identity. Anyone who trusts this self-signed certificate can verify packets generated by Alice. However, it is necessary to authenticate this certificate securely in the workspace in a web of trust fashion. Workspace user's web of trust authentication consists of two procedures: *origin authentication* verifies the origin identity of the owner of a certificate, and *endorsement* propagates the result of origin authentication.

Origin Authentication: To verify the origin identity of the key owner, one can make use of existing trust relationship in the Internet. If the existing identity provider offers some resources modifiable by the user (*e.g.,* Tweet, Gist, DNS Records), the verifier (say Bob) can use a similar method to Let's Encrypt ACME [1] and Keybase [2]. For example, the origin authentication let Bob sends a random nonce to Alice, asks her to sign it and post the signature to a channel that restricts write access only to the origin identity owner. If the identity provider provides an identity authentication protocol such as OpenID [5] or SAML [3], Bob can request a signed identity assertions containing a binding to the public key. Such protocol typically attaches a unique identifier to each request specified by the requester (*e.g.,* OIDC's nonce and SAML's request ID), which can be exploited to provide the binding between identities and keys.

Endorsement: Having authenticated Alice, Bob signs Alice a endorsement certificate, thereby she can present Bob's endorsement when others try authenticating her. Eventually, every user in same workspace will have certificates signed by everyone else, and a fully meshed web of trust is formed in workspace.

Cloud Independence: Workspace only uses origin identities for bootstrapping. After receiving an endorsement certificate, a user is bootstrapped into the web of trust and no need to communicate with today's authentication servers, and this web of trust runs completely independent from today's clouds. When there does not exist any trust-worthy identity provider, one can use out-of-band channel to transmit the self-signed certificate directly to bootstrap a new user as a fallback.

2.5 Synchronizing SWO at Rendezvous

Synchronizing SWO publications in "meetings" namespace requires a rendezvous that enables efficient SWO exchanging among Workspace users by names. Ideally, such rendezvous is a SWO exchange network that forwards Workspace SWO requests and response by names, and anyone who has Workspace content can join.

We have setup a testbed network [8] using a data-centric Internet architecture Named Data Networking (NDN) [10], to serve as the rendezvous that Workspace needs. This data-centric network testbed has 20 nodes across four continents, and is managed by collaborator universities in a decentralized fashion.

When started up, Alice's Workspace instance connects to the geographically closest router in the testbed, and registers her user prefix "`/meetings/alice@example.com`" to the router. NDN Sync [4] protocol ensures keeping all Workspace instances in *meetings* up-to-date regarding the current SWO production state, and instances can subsequently fetch the SWO they desire directly from each other, eliminating a central control node.

A key reason we used NDN testbed for rendezvous is that the testbed purely serves inter-connectivity purpose, and do not understand or parse the underlying SWO. Users or organizations may also choose to run their NDN networks in a decentralized fashion, and optionally connect these nodes to the existing testbed. Traditional clouds, on the other hand, must understand the semantics of the underlying data to be able to route them to the correct clients, and thus need to be centrally controlled and secured.

For situations where reaching the NDN testbed may not be possible, *e.g.,* in an airplane, we also provide the option of using PeerJS to establish direct $n \times n$ connectivity between all Workspace users in a group using WebRTC.

2.6 Local-First Collaboration

The Workspace application is supposed to work both offline and online to the rendezvous. This leads to two requirements: (a) necessary data must be stored locally, and (b) must support collaborations through asynchronous communications.

To provide offline access, we currently use browser's origin private file system (OPFS), which is a file-like storage persistent on the disk and private to the web application. All SWO received are stored in its original bytes, so the security properties (such as signatures) are preserved. For example, if Bob receives an SWO generated by Alice, he can forward this change to Jane, while its original signature by Alice is still valid.

Online storage is optional for the Workspace functionality if user online time periods widely overlap. Workspace does not bind itself to any specific cloud storage providers. Instead, Workspace prefers an in-network SWO storage instead of application database servers. Such in-network storage should have following properties:

- *Generality.* A generic storage is not designed for specific application. Multiple applications can share the same storage to improve cost efficiency if needed. An application like Workspace may also migrate to another storage provider easily.
- *Untrustworthiness.* A generic storage is not trusted by the application users. It cannot decrypt data generated by application users, nor can it author any changes by itself.
- *Simplicity.* The only functionality needed is to store and serve SWO. Extra functionalities such as access control is performed by the application, making a generic storage easy to implement and maintain.

Based on the features described above, we believe that a in-network SWO storage should be provided as *network services* by the network operators, instead of an application service of the cloud. As a short term solution, our current implementation used PythonRepo [7] as a prototype of such service in NDN testbed. We identify the generic storage service to be an important component of local-first software and expect to see more work on this topic coming out in future. When such a service is unavailable, a traditional cloud storage (such as Google Cloud Storage and AWS S3) can be used as a fallback.

3 CONCLUSION

Today's web applications run on cloud servers. Users connect to the cloud via secure connections, and the cloud identifies and authenticates users, serves as a rendezvous place for users to meet and a secure castle to host data and run apps.

In this paper we use the development of Workspace as a use case to show that, by using SWO as the basic building block, one can build decentralized web applications by solving three basic problems: unique user identities, decentralized security, and rendezvous. In Workspace, users obtain cloud independent identities and mutually authenticate each other through web of trust; they rendezvous over name-based NDN connectivity to exchange and synchronize SWO publications, which are secured directly by users, under Workspace's namespace. We map the Workspace file structure into a collection of shared data structure, and use CRDT as the solution to achieve eventual consistency among all users sharing the same document. Users can also work on documents without Internet connectivity, and synchronize SWO with others when rendezvous becomes available.

REFERENCES

[1] Josh Aas, Richard Barnes, Benton Case, Zakir Durumeric, Peter Eckersley, Alan Flores-López, J Alex Halderman, Jacob Hoffman-Andrews, James Kasten, Eric Rescorla, et al. 2019. Let's Encrypt: an automated certificate authority to encrypt the entire web. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security.* 2473–2487.

[2] KeyBase contributors. 2024. https://keybase.io/. Accessed: 2024-2-3.

[3] John Hughes and Eve Maler. 2005. Security assertion markup language (saml) v2. 0 technical overview. *OASIS SSTC Working Draft sstc-saml-tech-overview-2.0-draft-08* 13 (2005), 12.

[4] Philipp Moll, Varun Patil, Nishant Sabharwal, and Lixia Zhang. 2021. A brief introduction to state vector sync. *NDN, Technical Report NDN-0073, Revision 2* (2021).

[5] Natsuhiko Sakimura, John Bradley, Mike Jones, Breno De Medeiros, and Chuck Mortimore. 2014. Openid connect core 1.0. *The OpenID Foundation* (2014), S3.

[6] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. 2011. Conflict-free replicated data types. In *Stabilization, Safety, and Security of Distributed Systems: 13th International Symposium, SSS 2011, Grenoble, France, October 10-12, 2011. Proceedings 13.* Springer, 386–400.

[7] NDN Team. 2024. https://github.com/UCLA-IRL/ndn-python-repo. Accessed: 2024-2-3.

[8] The NDN Team. 2024. NDN Testbed. Online at https://named-data.net/ndn-testbed/.

[9] Tianyuan Yu, Xinyu Ma, Varun Patil, Yekta Kocaogullar, Yulong Zhang, Jeff Burke, Dirk Kutscher, and Lixia Zhang. 2024. Secure Web Objects: A Data-Oriented Paradigm. In *Submission to ACM Web Conference 2024.*

[10] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, KC Claffy, Patrick Crowley, Christos Papadopoulos, Lan Wang, and Beichuan Zhang. 2014. Named data networking. *ACM SIGCOMM Computer Communication Review* 44, 3 (2014), 66–73.