Private Document Editing with some Trust

Aaron MacSween
XWiki SAS
Paris
aaron.macsween@xwiki.com

Caleb James Delisle
XWiki SAS
Paris
calebjamesdelisle@xwiki.com

Paul Libbrecht
XWiki SAS
Paris
polx@xwiki.com

Yann Flory
XWiki SAS
Paris
yann.flory@xwiki.com

ABSTRACT

Document editing has migrated in the last decade from a mostly individual activity to a shared activity among multiple persons. The World Wide Web and other communication means have contributed to this evolution. However, collaboration via the web has shown a tendency to centralize information, making it accessible to subsequent uses and abuses, such as surveillance, marketing, and data theft.

Traditionally, access control policies have been enforced by a central authority, usually the server hosting the content, a single point of failure. We describe a novel scheme for collaborative editing in which clients enforce access control through the use of strong encryption. Encryption keys are distributed as the portion of a URI which is not shared with the server, enabling users to adopt a variety of document security workflows. This system separates access to the information ("the key") from the responsibility of hosting the content ("the carrier of the vault"), allowing privacy-conscious editors to enjoy a modern collaborative editing experience without relaxing their requirements.

The paper presents CryptPad, an open-source reference implementation which features a variety of editors which employ the described access control methodology. We will detail approaches for implementing a variety of features required for user productivity in a manner that satisfies user-defined privacy concerns.

CCS CONCEPTS

Security and privacy **Web application security**;

KEYWORDS

trust, online documents, collaborative editing, identities, encryption

ACM Reference Format:
Aaron MacSween, Caleb James Delisle, Paul Libbrecht, and Yann Flory. 2018. Private Document Editing with some Trust. In *DocEng '18: ACM Symposium on Document Engineering 2018, August 28–31, 2018, Halifax, NS, Canada.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3209280.3209535

DocEng '18, August 28–31, 2018, Halifax, NS, Canada

ACM ISBN 978-1-4503-5769-2/18/08…15.00
https://doi.org/10.1145/3209280.3209535

1 INTRODUCTION

In this paper, we present CryptPad [6], an open source [5], cloud-based suite of collaborative editors which is *private by design* [14]. While CryptPad has already been deployed as a production-ready system which is actively used, it is worthwhile to expose both the means and motivation for its development.

User testimonies have indicated previous or contemporaneous reliance on either Google Docs or Etherpad for tasks which required collaborative editing functionality. The technical architecture of either platform is such that the resolution of conflicts when editing is performed by the central server. One side effect of this design is that the contents of the document must be accessible to the server, and therefore legible to the server operator.

In the case of Google Docs, the software is provided as a service by a single provider. Google's privacy policy is very explicit about their right to *"process personal information on behalf of and according to the instructions of a third party"* [22]. As wariness of this business model has become increasingly common, it has been referred to by the unfavourable moniker of *Surveillance Capitalism*: *"constituted by unexpected and often illegible mechanisms of extraction, commodification, and control that effectively exile persons from their own behavior while producing new markets of behavioral prediction and modification."* [44].

While the term *personal information* may seem to imply that such arrangements primarily affect individuals, even very large organizations are subject to the same terms. Indeed, the influence of Silicon Valley companies has become so pervasive that the Danish government felt the need to appoint a *digital ambassador* [40].

As Etherpad has been freely available as an open-source project since it was acquired by Google in 2009, many privacy-conscious users have adopted it for their personal usage. While this may satisfy the privacy concerns of the server administrator, from anyone else's perspective it is simply a change in whom they place their trust. While certain server operators may offer a collaborative service with the promise that they will never read a user's content, users lack both transparency and any recourse in the event that they change their minds.

Whatever a user's preferences may be with regards to their privacy, they are often secondary to their immediate goals. Lacking an option which enables their productivity while respecting their privacy, many have reluctantly continued to accept the shortcomings of the systems mentioned above. Even so, an expanding demographic of technical and politically savvy users have found it increasingly difficult to ignore the value being extracted from their daily interaction with the tools around them.

As a scientific contribution, we intend to describe the system which enables collaborative editing of various document types in a manner which respects user privacy. Beyond that, we hope to address the social and economic factors which have made such a system necessary.

We will first provide a comprehensive overview of the technical and social problems which have contributed to the current challenges in adopting privacy-respecting editing software. This will be followed by a summary of the constraints which have hindered adoption in the past, and influenced the technical architecture of the system for our reference server and clients described thereafter.

Having detailed the context and implementation of the system, we propose a framework for evaluating its suitability for particular use cases, taking into account its resistance to likely attacks. Finally, we specify a variety of improvements to the cryptosystem which would mitigate certain attacks or improve user experience.

2 PROBLEM DEFINITION

For brevity's sake, we'll refer to the *unencrypted counterparts* of cryptographic productivity tools as their **baseline**. As our system offers a variety of editors, it is not always possible to compare against a single competing service, however, generally we will compare against those established in our introduction: Google Docs and Etherpad.

Through consultations with end-users, we have found that requirements are generally well satisfied by our baselines, except those relating to the legibility of documents by the host service. As services with pleasing interfaces are already available, it is expected that new services will provide equally professional designs while addressing privacy concerns.

Notably, those with whom we have consulted have reported that there is not a viable alternative which satisfies their privacy requirements, despite the existence of a number of solutions which integrate encryption into local document editing workflows [3, 4, 7]. Realtime collaboration is a necessary component for our users.

Similarly, users report that despite its proprietary nature, one advantage of Google Docs over Etherpad is the *Drive functionality*, which provides users with both a means of storing arbitrary files as well as an intuitive directory structure through which they may browse their documents. For individuals or organizations that opt to use Etherpad instead, workflows typically incorporate a secondary tool for indexing, such as a wiki. It would seem that an encrypted editing platform lacking any built-in functionality for organizing references to documents stored online, editors would likely resort to a less secure methodology for making that information accessible.

The need for organized collections of documents is not strictly due to collaboration between multiple users, but also between single users operating across a variety of devices. For this reason we will discuss collaborative processes as taking place between *agents* so as to accommodate such workflows. As users expect any change made to be available immediately across devices [33], we consider the *near real-time* propagation of state to be an essential property of not just ordinary documents, but also of document collections.

Finally, high availability is a strict requirement that has come to be expected of modern collaboration systems. While this may seem like an obvious implication given latency requirements, it is worthwhile to state that users require that their documents remain available to collaborators regardless of whether they themselves are online.

To summarize, users expect near real-time collaboration, functionality for the organization of document collections, support for multi-agent workflows, and high availability of documents, all while preserving the confidentiality of document contents.

3 CONSTRAINTS

The implementation of our system until now has been restricted by a variety of complicating factors outside of our problem definition. Those considerations are detailed below.

3.1 Privacy

Whether for the mundane purpose of targeted advertising, or for political agendas justified as *anti-terrorism* or *crime-prevention*, pervasive surveillance has been shown to have an inhibiting effect upon content-authors [24]. Public awareness of mass surveillance, and its effect on Internet denizens' behaviour has increased dramatically in recent years due to media coverage of government leaks, and the complicity of prominent cloud platforms [32].

Public discourse around privacy is frequently framed according to two clear categories of information, whether they be content and metadata, or personally identifying or anonymized. Either variant of this rhetoric obscures the fact that all of this information can be valuable under the right circumstances. Users that have expressed desire for a system which restricts their private information do not necessarily possess

sufficient awareness of the technical details to evaluate the risks associated with each type of information. As such it is left to the authors to model likely risks given average circumstances, and adapt the proposed cryptosystem according to feedback. Fuzzy requirements complicate the design process, and so we assume a minimum requirement to ensure the privacy of the content of the document. Additional properties are considered if the cost of their implementation does not introduce unreasonable complexity.

This may go without saying, but our inability to read user documents makes it such that we cannot monetize user content as our baseline competitors are able to. While this constraint is hardly scientific in nature, it is worth mentioning, as our goal is to provide a working, industrialized system that ensures user privacy. Our ability to meet this goal is unfortunately tied to the project's sustainability in an industry which has embraced the Freemium business model [35].

3.2 Public reception

The necessity of encryption is not apparent to the general public, as evidenced by the fact that approximately 30% of web traffic is not yet protected by transport-layer security (TLS) [8]. Nevertheless, adoption has increased dramatically since the advent of freely available SSL certificates. Failure to adopt encryption is not necessarily due to ambivalence on the topic, as workplace monitoring is mandated by many companies [20]. Users may not be in a position to sacrifice employability, social status, or productivity for privacy.

Though critical for modern networking infrastructure, TLS can be considered an incomplete encryption scheme, as it only protects traffic between agents and servers which possess certificates. A more effective means of preventing the compromise of an agent's private information is to rely on end-to-end-encryption (e2ee).

Additionally, we cannot ignore the role of government regulation and response regarding cryptography. The European Union's decision to implement the **GDPR** [17] presents a strong pro-privacy stance, though its text is clearly at odds with those who support the rise of Silicon Valley as an economic superpower [25]. The value of a collaborative tool is highly dependent on its ability to be adopted by organizations which can span across borders.

3.3 Resource consumption

Computing power has improved drastically over time, but with all else being equal applications including cryptography are at a disadvantage with regards to performance compared to their baseline. For privileged users with access to modern hardware, this may not be an issue, but for those with limited resources and baseline expectations, the resulting user experience may serve as a barrier to adoption.

> Debate remains, however, about whether widespread use of encryption was feasible in the early days of the Internet. The heavy computing demands, some experts say, could have made TCP/IP too difficult to implement, leading to some other protocol — and some network other than the Internet — becoming dominant. (see [41])

Though modern implementations of widely used ciphers have been subject to extensive optimization, there are other operations which incur a significant cost due to information-theoretic constraints. Textual search on documents is typically implemented in a centralized manner, relying on the server which hosts the information to return relevant results. As this type of operation is necessarily at odds with our fundamental requirements, such features require careful consideration, though users may regard search as a relatively basic feature.

3.4 Deployment

The availability of web browsers may serve to lower barriers to adoption, however, it is still subject to many constraints. Some users are not able to choose the browser they would prefer, due to workplace policy, limitations of the host platform, or due to only having access to shared computers (such as at a school or library). Users on corporate or academic networks often find themselves subject to the whims of highly restrictive firewall policies. Users may have to deal with very poor connectivity due to living in very remote, or poorly developed locations. The failure of a web application may stem from any one of these factors, or from an inconsistently implemented web API. In any case, it is expected that the software address such eventualities using only the limited privileges of a web page.

The requirement that no server ever learn the contents of a user's document could be satisfied by relying on a peer to peer architecture, such that the document is never stored on a server. In practice, we find that APIs for peer to peer communication in the browser are insufficiently reliable for our purposes. Until such a time as IPv6 deployment is widespread, and NAT traversal is either no longer necessary or sufficiently ubiquitous that any given agent may effortlessly communicate directly with any other, there is justification to believe that purely peer-to-peer, multi-agent computing (aka *fog computing* [42]) will not become a reliable option.

For the time being, agents which must communicate rely on unrestricted servers for mutual introduction, using either *Session Traversal of User Datagram Protocol through Network Address Translation* (STUN), or *Traversal Using Relays around NAT* (TURN) [31]. Even if that were not the case, the problem of data availability remains, as agents may be theoretically reachable may not have guaranteed uptime. While we await solutions for either of these difficult problems, cloud computing provides a satisfactory solution to address connectivity and resource availability.

3.5 Usability

An increasing number of open-source, user-facing *e2ee* applications have become available within the last decade. Many boast intuitive user interfaces, and can be deployed to a global audience as web services. E2ee pastebins [9], file-upload services [12], and instant messengers featuring voice and

video [19, 26] are all available. The leap from encrypted file upload to a fully featured editing suite introduces many additional user expectations, however. While an interface can offer visual similarities, e2ee imposes severe restrictions as to how a system is implemented.

Despite the support of a growing minority which is profoundly unsatisfied with the current state of the market, and great advances which have been made by a community which is by no means limited to the authors, there remain significant challenges to the engineering of a serious competitor to existing cloud-based editors.

Ultimately, we can expect that current browser APIs will stabilize, network connectivity will improve, and available personal computing hardware will become more capable of exceeding the demands of cryptographic algorithms. By contrast, we cannot expect that the practices of the average end user will become less detrimental to their own well-being [38], as attacks against network infrastructure are adversarial in nature, and thus likely to become progressively more sophisticated as time goes on. Despite the work between 2005 and 2016, when the papers *Why can't Johnny encrypt* [43] and *Can Johnny finally encrypt?* [26] were published, we are led to believe that *encrypting content is too hard for non-specialists*, and that "*Johnny generally cannot encrypt*". Researchers in the field have argued that *ease of use* and *accessibility* must be given their fair consideration as **security properties** [27].

We cannot expect that the complex mathematical properties involved in encryption will become simpler for end users to reason about. As such, we posit that without significant innovation in this regard, the adoption of privacy enhancing technologies within document engineering will not advance beyond its current status quo.

4 ARCHITECTURE

As specified in the *problem definition* section, web-based peer-to-peer approaches depending on technologies like WebRTC are not considered reliable for most use cases. Due to the necessity of publicly available STUN or TURN servers in such implementations, we opt instead to use a conventional client-server model. Wherever possible, client-side implementations allow for truly peer-to-peer collaborative editing; however, in practice at least one of the peers must typically remain available for the purpose of storing and providing document history.

Since it is a core requirement that documents remain illegible to any but the intended participants, the responsibilities of the server are significantly reduced. As such, the vast majority of the application is implemented on clients, an architecture which may seem highly unorthodox compared to baseline implementations.

4.1 Database

CryptPad implements a simple *store and forward server* [23], with a database reflecting the *Kappa architecture* design pattern which relies heavily on **append-only immutable logs** [34]. Each real-time document corresponds to a single append-only log on the server, generally referred to as a *channel*. Each entry in the log constitutes a *patch*: an encoded instruction which transforms a document from one state into another.

The database is considered public information, as it is expected that access-control will be implemented on the client using strong encryption. Client-side encryption renders the content of each channel illegible to the server. Each channel is accessible via a unique, high-entropy hexadecimal identifier. Clients are responsible for maintaining the pairwise list of channels and encryption keys which enable collaboration.

4.2 Key distribution

CryptPad exploits the fact that by design, clients do not send information stored in the *fragment identifier* (the part of the URI after an *octothorpe* (#)) to HTTP servers in their requests [13]. As such, it is considered safe to store channel identifiers (considered public), and channel encryption keys (considered private) within the fragment identifier. By parsing data encoded into the URL using Javascript, clients are able to request and decrypt all information provided by the server. Furthermore, these URLs can be shared with other agents over any medium, providing access to the document. Since these URLs are effectively immutable, documents may link to one another in whatever manner desired by the user.

4.3 CryptDrive

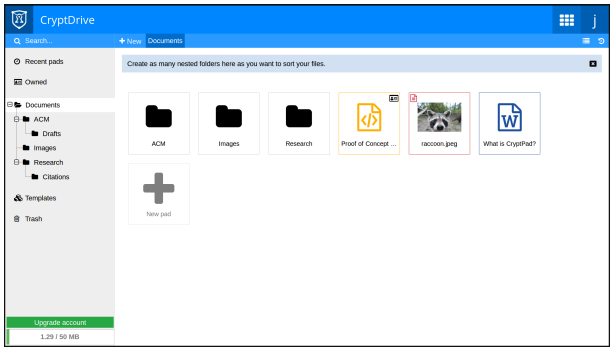

Figure 1: CryptDrive

Users that have *logged in* are able to save their preferences, and access a *virtual file system* known as their *CryptDrive*. Preferences and the contents of a CryptDrive are stored within an arbitrarily nested encrypted data structure, the technical specifications of which will be described below. From the perspective of the server, this data structure is just another real-time document, stored in a *channel*.

Through the use of the above data structure, referred to as the *user-object*, the secret information necessary to collaboratively edit more conventional document types can be stored for later reference. The channel and encryption keys for this special document can be derived from a cryptographically-secure random source and stored using the *localStorage* browser

API, enabling single-device usage. Alternatively, any device with knowledge of the channel identifier and encryption keys can access a shared CryptDrive from multiple devices.

4.4 Identity

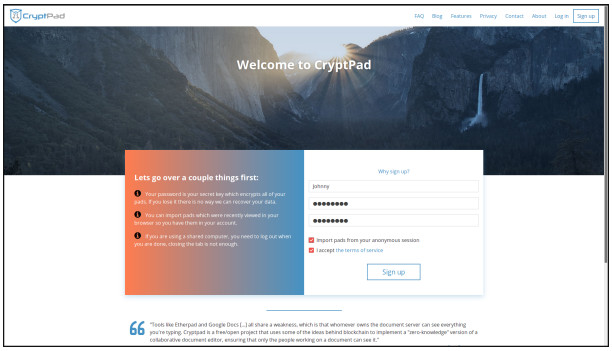

Figure 2: registration

Registration and **Login** for the purpose of cross-device access are implemented on the client using the *Scrypt password-based key derivation function* (as implemented by *scrypt-async.js* [10]). By deterministically transforming a username and password into a high-entropy stream of bytes, it is possible for users to interact with familiar user-interface which consistently yields the information necessary for them to access their *user object* from any device.

Registration and login only differ in that the registration process verifies that a user does not already exist for that exact set of credentials, and initializes user data, while login verifies that such a user *has already registered*.

4.5 Remote Procedure Calls

While many other features could be implemented entirely on the client as well, the server is in a position to do so in a considerably more efficient manner for most tasks, and so this privileged position is leveraged for optimization purposes. For example, a client could check the size of a particular log by streaming it, and counting the number of bytes, incurring a significant load on the network and the server, which would be responsible for reading and streaming the log. Instead, the server performs a single operation which is orders of magnitude less intensive, and sends the result to the client.

The common task of authenticating users for the purpose of quota management and remote procedure calls (**RPC**s) is achieved through the use of public-key cryptography. Authenticated clients each possess an *ed25519 signing keypair* (as implemented within *tweetnacl.js* [11], and each request is signed to prove that they possess the *private key* corresponding to the *public key* which constitutes their identity. All server-enforced access control is based upon this pseudonym, such that the server need not ever record a password or username.

5 SERVER IMPLEMENTATION

Our reference implementation server is written in Javascript (NodeJS), to facilitate the reuse of code on clients. It implements both a static web-server and an API server. In production environments, the responsibilities of the web-server are fulfilled by a dedicated process, as Javascript's single-threaded model is not ideal for the task.

5.1 API server

Low-latency client-server communication is achieved through the use of websockets, and so the availability of the WebSocket API acts as a hard requirement for our supported clients.

The server acts as a relay between clients, effectively behaving like a group messaging server. Upon connecting to the server, each user is assigned a temporary, unique identifier. Clients may then join *channels*, the transient membership of which is maintained by the server. The server also implements direct messaging, which clients can leverage to communicate with other clients using their temporary identifier.

Channel broadcast is defined as the delivery of a message (encoded as JSON) from one member to every other in the channel. Broadcast messages are appended to the channel's log.

Our implementation uses a pluggable storage API which is currently backed by an abstracted interface for writing to the host file-system. Each channel corresponds to a distinct file, with each message stored as a *serialized JSON object*, and discrete messages delimited by newline characters.

New channels are created by writing the immutable properties of the document as a JSON map to the first line of its file. The properties to be encoded will be detailed in the client specification below.

The protocols implemented for websocket communication also include the *RPC* functionality mentioned above. Clients can leverage this functionality to upload encrypted files, request the deletion of a channel which they *own*, or interact with the list of files which are counted against their quota. Channels and encrypted files unaccounted for by any user's quota are eventually removed after a period of inactivity.

5.2 HTTP Server

In addition to the API server described above, CryptPad also relies on the usage of a traditional HTTP server for delivering static content to clients. While static web servers are generally a fairly unsophisticated component of a web application, CryptPad relies on some advanced configuration to mitigate certain attacks.

In implementing a complex, multi-user cryptosystem in the browser, we have presented adversaries with a considerable attack surface. Fortunately, a great advancement in the field of web application security in the **Content Security Policy** (CSP) specification [37].

Through the use of CSP headers sent as part of the HTTP responses, a server is able to inform a client which resources it should consider safe. CryptPad's static content is served over two separate domains, with CSP and the same-origin

security policy enforcing limitations on what data is accessible. Any interaction with sensitive data is performed on the *safe domain*, while most other computation is executed in an iframe on the *unsafe domain*. Any communication that must occur between the two domains is achieved via the iframe's *postMessage* API, with limitations as to what information can be shared enforced by code running on the *safe domain*. Through this methodology, even in the event that a malicious actor is able to leverage a cross-site scripting (XSS) vulnerability, the value of the data that can be extracted is severely restricted. A similar approach has been detailed in 2013, described as *CryptFrames* [39].

6 CLIENT IMPLEMENTATION

As noted above, considerably more of CryptPad's functionality is implemented on the client than on the server. This has significant benefits in terms of scalability, as a single server can support many more agents than would be possible if it were responsible for the same responsibilities.

ChainPad, the document consensus engine, and its various extensions (presented below in dedicated sections), play a central role in obviating the need for server-based conflict resolution. Considerable development effort is avoided by depending on third-party editors wherever possible, and merely adapting their interfaces for use with our encryption and data replication protocols.

6.1 ChainPad

Described as a *Realtime Collaborative Editor Algorithm based on Nakamoto Blockchains* [2], is provided as a reference implementation in ECMAScript 5 (Javascript). It is capable of execution in all modern Javascript runtimes, including the most popular browsers and NodeJS. Though ChainPad was developed independently, a similar approach was employed in the SPORC collaborative editing algorithm [21].

Its methodology assumes that agents participating in a collaborative session have some means of communicating with one another, and that the collaborative document can be represented as a UTF8 string. Documents are initialized as the *empty string* (`""`). Clients track the document according to two notions, the *authoritative document* or **authDoc** for which all connected agents have reached consensus, and each particular agent's subjective view of the document or **userDoc**.

Each agent is able to set their userDoc to any UTF8 string. ChainPad is used to formulate their revision to the document as a *patch*, a sequence of operations with sufficient information for peers to transform their representation of the document to match that of the agent making the modification.

An operation is defined as:

(1) an offset (a number of UTF8 characters relative to the start of the document)
(2) a number of characters to remove (relative to the offset)
(3) a string to insert at the offset

A document's history can be replayed, with its integrity assured by the inclusion of the sha256 hash of the preceding patch known to the editing user, forming an immutable hash-chain comparable to that popularized by Bitcoin, but without the imposed difficulty of proof-of-work computations.

Patches can be encrypted client-side before being sent over the network. Concurrent patches are reconciled using a simple operational transformation algorithm implemented on all clients, which formulates a new patch taking into account incoming patches.

Patches are considered to apply to the *userDoc* until the client receives acknowledgement that they have been delivered and accepted, at which time they are considered to be a part of the *authDoc*.

Patches which result in document states considered invalid by contemporary clients are rejected. A divergence between at least two agents is known as a *fork*. Forks can be the result of intermittent connection loss or other circumstances, but in any case they are resolved by the *longest valid chain* being considered the authoritative state of the document.

This scheme would seem to imply that to participate in a document, an agent must synchronize its entire history before attempting to append new patches. In practice, this is avoided via a simple checkpointing mechanism, whereby every 50th patch consists of the removal and reinsertion of the entire document.

6.2 Encryption

CryptPad derives its name from the fact that any time an agent broadcasts messages (either to contemporary agents, or strictly for the purpose of adding a revision to the document's history), it encrypts its messages before communicating them to the server. In most situations, the cryptography employed is the *authenticated symmetric xsalsa20-poly1305 cipher*.

Newly created messages are signed using *ed25519* keys, the public keys for which are shared with the server, such that it can reject invalid messages which could otherwise be inserted into the channel history. Thusly, there are four possible configurations for clients:

(1) agents with both the symmetric encryption key and the shared secret signing key are able to craft valid patches and append to the document history
(2) agents with only the symmetric encryption key are able to view the document, but not add or revise content
(3) agents possessing the signing key alone are able to insert invalid messages into the document history, Such agents can inconvenience legitimate editors with invalid content which will be ignored, however, they cannot read or modify the resulting document's content
(4) agents with neither key (including the server) are able to download publicly available encrypted content, but not read the decrypted contents, nor append new patches to the history

The scheme detailed above is implemented as a configuration point, which allows for any number of other ciphersuites to be employed depending on the intended application, such as:

(1) disabling encryption in trusted environments where secrecy is not a requirement
(2) enabling one-to-one encrypted collaboration channels based on public key cryptography
(3) facilitating crypto-agility, upgrading to more resilient ciphers as they become available [15]

6.3 ChainPad-ListMap

ChainPad only supports UTF8 documents, which would seem to limit its application to simple plain-text documents. ChainPad-ListMap [1] makes it possible to collaboratively edit documents represented as arbitrarily-nested JSON data types.

It is implemented primarily with ECMAScript 5, using only the *Proxy* API specified within ECMAScript 6 if it is available, falling back to alternative behaviour otherwise.

Changes made to the collaborative data structure are transformed into patches compatible with ChainPad by first transforming the structure into a canonical representation in the form of a UTF8 string, and then proceeding to formulate patches which can be communicated to other clients. Conversely, upon receiving patches, they are applied against the UTF8 representation, and then transformed into an object, after which a structural comparison is made. Differences are encoded as instructions to update the exposed proxy such that they match the state reflected within ChainPad.

Applications can be built on top of this data structure by observing changes to the structure via a simple callback-based pattern-matching API. This has been employed to great effect within the *CryptDrive* application, among other use-cases.

7 EVALUATION

Given the incommensurability of performance and privacy, it can be difficult to make a clear comparison of encrypted productivity tools against their baseline. We adopt the evaluation criteria proposed by the authors of *Depot* [30]; in their words, *"What is the price of distrust?"*.

In our system, particular features may have equivalents to the of the baseline implementation, but at a greater cost in terms of computational resources. Alternative features may be available, but their properties may differ from the conventional workflow, requiring adjustment on the users' part. Finally, features may be conjectured or proven to be impossible to implement in a privacy-preserving manner, necessitating that users either accept their omission or accept the loss of privacy.

On the topic of real-time collaboration, latency introduced by our consensus algorithm is orders of magnitude smaller than the average latency between users and the host server, which is typically the case for our baseline comparisons as well. Furthermore, our server is able to handle much more load with fewer resources, by offloading the majority of the processing to clients.

The organization of documents via an agent's *user-object* consumes more resources compared to our baselines. Since the agent's personal database is effectively implemented on the client, insertions, modifications, and deletions all require that the client instantiate a complete ChainPad session. While there are avenues for decreasing the expense of this synchronization, they have yet to be implemented.

There are significant usability challenges to our approach for multi-agent workflows, foremost of which that multiple users can use the same username, and that password change is not possible. Solutions will be detailed in the *future development* section, though, it should suffice to say that if the server were able to reset a user's password to recover their documents, it would be possible for a compromized service provider to reset the password for the user's adversaries. Our unconventional approach can result in user confusion in average cases, and at worst, users that do not understand that we cannot recover their data are liable to lose access to it.

CryptPad competes well in terms of the availability of documents, as our cloud-service methodology does not differ significantly from our baselines. With that being said, our patch-based document format does incur a penalty with regards to network transfers. This overhead makes tasks like conventional textual search impractical.

Our platform excels where user privacy is concerned. Our service operates not only without knowledge of the user's content, but by reusing our document synchronization algorithms for generic user data, we avoid learning metadata like document names, file types, and user tags. Furthermore, it is sufficiently generic to implement editing of a variety of document types.

8 ATTACK VECTORS

In order to ensure user requirements, our service must be able to withstand a variety of malicious behaviour. Some common attack vectors and their efficacy are detailed below.

8.1 Brute force decryption

The history of CryptPad's channels are considered public information. Barring any unforeseen vulnerabilities in our cryptographic dependencies, brute force decryption is not considered a viable attack vector due to the widely believed efficacy of the ciphers we have employed, and the size of the keyspace against which any attacks would be directed.

8.2 Denial of Service

DoS attacks, and more notably *distributed denial of service* (DDoS) attacks are unsophisticated, though they can be effective on a temporary basis. It is very likely that in the event of a sustained attack of this type, an experienced service operator would be able to take steps to mitigate its effectiveness, such as restricting access from particular IP ranges.

8.3 Ongoing device access

In the event that a user's device becomes compromized on an ongoing basis, either remotely or via regular local access, there is practically nothing that the service provider can

do to aid the user. Such adversaries would be able to log a user's keystrokes and learn their secret credentials. In any case, an adversary with the ability to fully compromize a device would not have any less difficulty affecting a user of a baseline service. As such, we consider such attacks out of scope.

8.4 Restricted/momentary device access

A sophisticated adversary could leverage temporary access to discover the channel id and encryption key used to secure a user's CryptDrive. At present, the cryptosystem employed to secure a user's CryptDrive *does not* implement *forward-secrecy*. Unfortunately, the implication of this is that given the right circumstances and expertise, an attacker can gain ongoing access to a user's CryptDrive and all the documents referenced therein.

8.5 MitM

A *Man in the Middle* (MitM) attack can be mitigated by serving CryptPad over *HTTPS*. In the event of an exploit that renders HTTPS ineffective, an adversary could modify the Javascript sent to the client such that it would leak their encryption keys, to a result comparable to that of *momentary device access* detailed above. Such an exploit would affect most of the world's services, including our baselines.

8.6 Passive surveillance of out-of-band communication

Assuming users communicate over a separate channel, such as e-mail or an instant messaging platform, the operator of that channel would be able to observe the secret information necessary to join a collaborative session themself. The CryptPad API server does not do anything to prevent such an attack, however, membership of the channel is broadcast to all members, so such an obverver would be visible if they joined a session. Users can effectively mitigate this class of attack by either using a third-party encrypted messenger, or by using the encrypted channels provided by the CryptPad platform itself.

8.7 Passive server exposure

This vector assumes that the operator of the service is not acting out of malicious intent, however, they may have been the recipient of a subpoena or other legal mechanism which compels them to provide passive access to their server, reading the filesystem contents, and observing network traffic.

Such a situation is not catastrophic, as all explicit content is encrypted before being sent to the user, however, such an observer could still gather considerable information by observing HTTP logs or network access. As a web service, common details such as a user's IP address, user-agent string, and activity patterns can reveal information which may be leveraged against them in the future.

To limit what metadata is exposed, users can access the service using the Tor anonymization network, and employ a variety of similar techniques which would make them difficult to identify. Even so, users who authenticate with the server using its public key methodology will still be identifiable via their persistent pseudonym.

8.8 Corrupt server

A fully corrupt server is a disastrous scenario, whether due to the actions of a malicious administrator or through their incompetence leading to third-parties gaining unauthorized access.

Anyone in control of a server has all the capacity of a MitM attack, such as serving malicious Javascript to leak all of their keys. Such an attack is considered an *active* attack, however, which most adversaries will hesitate to employ as it reveals their intentions and capacity in the event that they are caught.

Should evidence of unauthorized access be detected, the operator can shut down their server and notify users of a breach, just in case any of the Javascript was modified with malicious intent. So long as the attack goes undetected, any user accessing and executing the compromized source will be at risk of leaking all of their secret data.

9 FUTURE DEVELOPMENT

The following remarks concern research or development which has not yet been integrated into the platform. The following sections will address the shortcomings of the system which we have acknowledged.

We consider it well within the scope of the platform to make it such that a user's likelihood to be compromized is minimized, whether through an architectural flaw in the platform, or through patterns of usage which make their information likely to be leaked. Furthermore, if the platform is sufficiently unsatisfying that users prefer to collaborate via a platform which does not employ strong encryption, we consider that their information has been compromized all the same.

Given this and the various attack vectors which could still be effective, our future plans include developing features which would both make users more likely to adopt the platform for their daily use, and other steps to directly limit the efficacy of the attacks described above. Future topics of interest can be classified under *usability*, *scalability*, and *improved security*.

9.1 Usability

For users that access CryptPad using low-resource systems (when compared to current average specifications), we hope to optimize key points in our algorithms such that they are satisfied with the performance-privacy tradeoff described in our *evaluation criteria*. For users accessing the service over poorly connected networks, we hope to be able to decrease any overhead incurred by our on-wire encoding such that fewer bytes must be transmitted to the server.

In terms of workflows, the management of passwords is a topic clearly in need of improvement. While we feel our position against password reset is justified, we also see the

difficulty and frustration experienced by users who have lost or forgotten their password. Unfortunately, the widespread availability of password reset features for web services has trained users to create new accounts without considering a long-term strategy for securely and resiliently storing their credentials. We are considering means of recovering access to a user's account based on *Shamir's Secret Sharing* [36] or similar *(t, n)* threshold-based secret sharing schemes, relying on the division of shares amongst a combination of friends and distinct cloud services, though more research into the viability of this as an intuitive solution is required.

Finally, in terms of usability, we would like to simplify the process of sharing access with other users to address its current shortcomings. One approach is to enable password-protection for documents such that even if the URL leaks, users must correctly guess a passphrase to access the contents. Other schemes include using public key encryption for links, such that only the intended user (who must possess a public key which is readily available) would be able to decrypt the message to access the actual shared secret. This could be made transparent to the user via a dedicated app for decrypting such secrets, and would appear to behave superficially like a redirect through a link-shortening service with which many users are already accustomed.

9.2 Scalability

Some features within the platform are tightly coupled, such as the RPC and storage mechanisms. Furthermore, our reference implementation is a single Javascript process, meaning that all processing is executed on a single thread. While this detail has simplified development, it also makes it difficult to provision additional computing resources to the API server, which places a hard limit on how many users can be supported by a particular instance of CryptPad.

So long as a particular API server must be hosted by a single machine, it will continue to be necessary for users who are farther away from that server to tolerate greater latency. A more flexible, distributed architecture will not only accomodate more users, but provide a better user experience for these users.

9.3 Security improvements

In conjunction with the user-experience improvements to our password management scheme, there is also room for improvement in terms of mitigating password-related security issues. Currently, users whose passwords are exposed have no means of changing their passwords. This is a separate issue from *password recovery*, because we assume a user still has access to their account. In the event that another user learns their password, or they realize that another service that shares that password was compromized, this is a critical workflow. Fortunately, there is already useful research into applications for password-based architectures which do not rely on trust, for use in peer-to-peer computing [28]. While CryptPad is not strictly peer-to-peer, these same methods can easily be adapted for our admittedly simpler use case. Should the content of a user's CryptDrive be exposed, it would be prudent to limit the damage through an improved scheme which can promise *forward secrecy*.

The effectiveness of less severe attacks like *passive server exposure* for collecting metadata could be limited through application of the various techniques developed in the field of private information retrieval [16]. Failing that, we can continue to ensure that the platform remains accessible to users connecting through the Tor network.

Finally, the most severe situation, in which a corrupt server delivers malicious Javascript, must be addressed. The most promising approach for this threat is to implement a *trust on first use* (TOFU) bootloader which verifies authenticity of deployed scripts via the *sub-resource integrity* API [18]. This bootloader could then provide a root signature signed by the developers, with a public registry of verified signatures made available for auditing purposes, similar to the approach leveraged for *certificate transparency* [29].

10 CONCLUSION

Our system has proven to be highly satisfactory for our growing user base, who until now have been sorely underserved by existing industrial competitors. Even so, considerable research and development is still required in order to provide some functionality which users have come to expect. Ultimately, demands are not strictly limited to editing functionality. Users require support for a variety of workflows, including categorization, discoverability, and private inter-user communication in order to guarantee the confidentiality of their content.

Our research indicates that a multi-disciplinary approach is most effective when designing privacy-enhancing technologies. At a minimum, there is evidence to believe that attempts must consider cryptography, cognitive ergonomics, and application engineering in order to serve users attempting to protect themselves from online surveillance.

We expect the demand for privacy-preserving collaboration platforms to grow as more of the world's productivity practices depend on cloud-based services. In order to prevent cooption or compromize of such platforms, techniques ensuring the integrity of deployed systems will need to be employed. At present, the system still requires users to place some trust in their server administrators, however, the level of trust is reduced compared to that of our competing services. Notably, as most security is implemented clientside, there is a degree of accountability introduced, empowering users to *trust, but verify*.

REFERENCES

[1] 2018. chainpad-listmap source code. (2018). Available from https://github.com/xwiki-labs/chainpad-listmap; consulted 2018-04-10.
[2] 2018. chainpad source code. (2018). Available from https://github.com/xwiki-contrib/chainpad; consulted 2018-04-10.
[3] 2018. crypt-editor source code. (2018). Available from https://github.com/alseambusher/crypt-editor; consulted 2018-04-10.
[4] 2018. CryptoTE source code. (2018). Available from https://github.com/bingmann/cryptote; consulted 2018-04-10.

[5] 2018. CryptPad source code. (2018). Available from https://github.com/xwiki-labs/cryptpad; consulted 2018-04-10.

[6] 2018. CryptPad.fr. (2018). Available from https://cryptpad.fr/; consulted 2018-04-10.

[7] 2018. Encryptpad source code. (2018). Available from https://github.com/evpo/EncryptPad; consulted 2018-04-10.

[8] 2018. HTTPS adoption statistics provided by letsencrypt.org. (2018). Available from https://letsencrypt.org/stats/; consulted 2018-04-10.

[9] 2018. ncrypt source code. (2018). Available from https://github.com/Upload/Up1; consulted 2018-04-10.

[10] 2018. scrypt-async-js source code. (2018). Available from https://github.com/dchest/scrypt-async-js; consulted 2018-04-10.

[11] 2018. tweetnacl-js source code. (2018). Available from https://github.com/dchest/tweetnacl-js; consulted 2018-04-10.

[12] 2018. Up1 source code. (2018). Available from https://github.com/Upload/Up1; consulted 2018-04-10.

[13] Tim Berners-Lee. 1997. URI References: Fragment Identifiers on URIs. (1997). Available at https://www.w3.org/DesignIssues/Fragment.html.

[14] A. Cavoukian. 2012. Privacy by Design [Leading Edge]. *IEEE Technology and Society Magazine* 31, 4 (winter 2012), 18–19. https://doi.org/10.1109/MTS.2012.2225459

[15] Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray Perlner, and Daniel Smith-Tone. 2016. *Report on Post-Quantum Cryptography*. Technical Report. NIST. See https://csrc.nist.gov/publications/detail/nistir/8105/final.

[16] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. 1995. Private Information Retrieval. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS '95)*. IEEE Computer Society, Washington, DC, USA, 41–. http://dl.acm.org/citation.cfm?id=795662.796270

[17] European Commission. 2018. General Data Protection Regulation. Available online at https://ec.europa.eu/info/law/law-topic/data-protection/data-protection-eu_en.. (2018). Onlline, accessed 2018-02-15.

[18] MDN Contributors. 2017. *Subresource Integrity*. Mozilla Developer Network. Available at https://developer.mozilla.org/en-US/docs/Web/Security/Subresource_Integrity; consulted on 2018-02-15.

[19] Ksenia Ermoshina, Francesca Musiani, and Harry Halpin. 2016. End-to-End Encrypted Messaging Protocols: An Overview. In *Internet Science - Third International Conference, INSCI 2016, Florence, Italy, September 12-14, 2016, Proceedings*, Vol. 9934. Springer, Cham, 244–254. https://doi.org/10.1007/978-3-319-45982-0_22

[20] Laura Evans. 2007. Monitoring Technology in the American Workplace: Would Adopting English Privacy Standards Better Balance Employee Privacy and Productivity? *California Law Review* 95, 4 (2007), online. Available at https://scholarship.law.berkeley.edu/californialawreview/vol95/iss4/4/.

[21] Ariel J. Feldman, William P. Zeller, Michael J. Freedman, and Edward W. Felten. 2010. SPORC: Group Collaboration Using Untrusted Cloud Resources. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI'10)*. USENIX Association, Berkeley, CA, USA, 337–350. Available from https://www.usenix.org/legacy/events/osdi10/tech/.

[22] Christian Fuchs. 2011. Web 2.0, Prosumption, and Surveillance. *Surveillance and Society* 8, 3 (2011), 288–309. Available from http://www.surveillance-and-society.org.

[23] A D Gelman, H Kobrinski, L S Smoot, S B Weinstein, M Fortier, and D Lemay. 1991. A store-and-forward architecture for video-on-demand service. In *IEEE International Conference on Communications*. IEEE, Piscataway NJ, 842–846.

[24] Keith N. Hampton, Lee Rainie, Weixu Lu, Maria Dwyer, Inyoung Shin, and Kirsten Purcell. 2014. *Social Media and the 'Spiral of Silence'*. Technical Report. Pew Research Center. Available from http://www.pewinternet.org/2014/08/27/the-spiral-of-silence-on-social-media/.

[25] Stephanie Hare. 2016. For your eyes only: U.S. technology companies, sovereign states, and the battle over data protection. *Business Horizons* 59, 5 (2016), 549 – 561. https://doi.org/10.1016/j.bushor.2016.04.002 THE BUSINESS OF PEACE.

[26] Amir Herzberg and Hemi Leibowitz. 2016. Can Johnny Finally Encrypt?: Evaluating E2E-encryption in Popular IM Applications. In *Proceedings of the 6th Workshop on Socio-Technical Aspects in Security and Trust (STAST '16)*. ACM, New York, NY, USA, 17–28. https://doi.org/10.1145/3046055.3046059

[27] Nadim Kobeissi and Arlo Breault. 2013. Cryptocat: Adopting Accessibility and Ease of Use as Security Properties. (2013). arXiv:1306.5156 http://arxiv.org/abs/1306.5156

[28] Gunnar Kreitz, Oleksandr Bodriagov, Benjamin Greschbach, Guillermo Rodríguez-Cano, and Sonja Buchegger. 2012. Passwords in peer-to-peer. In *12th IEEE International Conference on Peer-to-Peer Computing, P2P 2012, Tarragona, Spain, September 3-5, 2012*. IEEE, Piscataway NJ, 167–178. https://doi.org/10.1109/P2P.2012.6335797

[29] B. Laurie, A. Langley, and E. Kasper. 2013. *RFC 6962: Certificate Transparency (Experimental Protocol)*. Technical Report. IETF. Available from https://tools.ietf.org/html/rfc6962.

[30] Prince Mahajan, Srinath T. V. Setty, Sangmin Lee, Allen Clement, Lorenzo Alvisi, Michael Dahlin, and Michael Walfish. 2010. Depot: Cloud Storage with Minimal Trust. In *9th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2010, October 4-6, 2010, Vancouver, BC, Canada, Proceedings*, Remzi H. Arpaci-Dusseau and Brad Chen (Eds.). USENIX Association, Berkeley, CA, USA, 307–322. http://www.usenix.org/events/osdi10/tech/full_papers/Mahajan.pdf

[31] R. Mahy, P. Matthews, and J. Rosenberg. 2010. *RFC 5766: Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)*. Technical Report. IETF. Available from https://tools.ietf.org/html/rfc5766.

[32] Alex Marthews and Catherine E. Tucker. 2017. *Government Surveillance and Internet Search Behavior*. Technical Report. MIT Sloan Business School. Available from https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2412564.

[33] M. Asif Naeem, Gillian Dobbie, and Gerald Webber. 2008. An Event-Based Near Real-Time Data Integration Architecture. In *Proceedings of the 2008 12th Enterprise Distributed Object Computing Conference Workshops (EDOCW '08)*. IEEE Computer Society, Washington, DC, USA, 401–404. https://doi.org/10.1109/EDOCW.2008.14

[34] Milinda Pathirage. 2018. Kappa Architecture.com. (2018). Available from http://milinda.pathirage.org/kappa-architecture.com/; consulted 2018-02-15.

[35] Nicolas Pujol. 2010. Freemium: Attributes of an Emerging Business Model. (Dec 2010). Available online at SSRN https://ssrn.com/abstract=1718663.

[36] Adi Shamir. 1979. How to Share a Secret. *Commun. ACM* 22, 11 (Nov. 1979), 612–613. https://doi.org/10.1145/359168.359176

[37] Sid Stamm, Brandon Sterne, and Gervase Markham. 2010. Reining in the Web with Content Security Policy. In *Proceedings of the 19th International Conference on World Wide Web (WWW '10)*. ACM, New York, NY, USA, 921–930. https://doi.org/10.1145/1772690.1772784

[38] Jeffrey M. Stanton, Kathryn R. Stam, Paul Mastrangelo, and Jeffrey Jolton. 2005. Analysis of end user security behaviors. *Computers & Security* 24, 2 (2005), 124–133. https://doi.org/10.1016/j.cose.2004.07.001

[39] Emily Stark. 2013. *From client-side encryption to secure web applications*. Ph.D. Dissertation. MIT. Accessible at https://dspace.mit.edu/handle/1721.1/82382.

[40] The local. 2017. Denmark names first ever digital ambassador for Silicon Valley role. *The Local.dk, Denmark's news in English* 20170526 (May 2017), online. Available online at https://www.thelocal.dk/20170526/denmark-names-first-ever-digital-ambassador-for-silicon-valley-role.

[41] Craig Timber. 2015. Net of insecurity: A flaw in the design. *The Washington Post* 2015-05-30 (30 May 2015). See http://www.washingtonpost.com/sf/business/2015/05/30/net-of-insecurity-part-1/.

[42] Luis M. Vaquero and Luis Rodero-Merino. 2014. Finding Your Way in the Fog: Towards a Comprehensive Definition of Fog Computing. *SIGCOMM Comput. Commun. Rev.* 44, 5 (Oct. 2014), 27–32. https://doi.org/10.1145/2677046.2677052

[43] Alma Whitten and J. D. Tygar. 1999. Why Johnny Can'T Encrypt: A Usability Evaluation of PGP 5.0. In *Proceedings of the 8th Conference on USENIX Security Symposium - Volume 8 (SSYM'99)*. USENIX Association, Berkeley, CA, USA, 14–14. http://dl.acm.org/citation.cfm?id=1251421.1251435

[44] Shoshana Zuboff. 2015. Big other: surveillance capitalism and the prospects of an information civilization. *Journal of Information Technology* 30, 1 (March 2015), 75–89.