Skiff
End-to-end encrypted, Web3-native Workspace

May 2022

1 Overview

This whitepaper outlines a threat model, a set of desired security properties, and a high-level system design for Skiff - a private, decentralized workspace and email product. In ensuring total privacy for communications and collaboration, Skiff helps individuals and teams work more freely and effectively.

This whitepaper proposes a system where no sensitive document information (including all documents and document titles) is ever stored, seen, or accessible to anyone except its creator and their chosen collaborators. In the mail context, no email sent between Skiff users can ever be seen by Skiff, or by anyone else. This is achieved using end-to-end encryption as well as additional safeguards, including robust authentication methods, out-of-band key verification ("mark as verified"), and two-step authentication.

At a more intuitive level, Skiff uses the latest in privacy, cryptography, and decentralization to keep users' work and personal data private and truly owned by each individual.

2 Threat Model

Skiff's threat model assumes that adversaries may access any data sent over a network to or from a client (even data sent over encrypted network connections). It also assumes that data stored with a cloud provider cannot be assumed to be confidential.

As a result, we assume the server is "honest but curious:" Skiff's servers will not deliberately deny access to data, and will not serve compromised client applications over the web, but that data sent to Skiff or stored in Skiff databases may be compromised. For the most security-conscious users, this second component of the threat model (maintaining honest client applications) is supported by using subresource integrity and out-of-band public key verification (see further sections of this document). Additionally, native iOS, Android, and macOS applications are currently available (with more coming), thereby further protecting users from a network threat model.

Given our mission to build privacy-first software, our threat model and system design also attempt to protect users from abuse and discoverability or de-anonymization. This includes designing server endpoints to prevent user enumeration, and building mechanisms for users to block others users or remove themselves from (or report) shared content.

2.1 Desired Properties

1. **End-to-end encryption of all sensitive information**: The contents and certain metadata (including title, time created, and last modified date) associated with every document created or uploaded by a user is only visible to the user or shared collaborators.

   All emails sent between Skiff users also remain end-to-end encrypted and only visible to their creators. For emails sent to or received from external services, Skiff will never store a plaintext copy of any sensitive information (subject, contents) in the email.

2. **Resistant to man-in-the-middle attacks**: Even without a fully trusted communication channel between the server and clients, our model will never reveal email subject, email content, or document title, contents, and other information.

3. **Resistant to user abuse attacks**: As we expand and improve our services, we are particularly sensitive to user abuse as a threat vector (as discussed in our threat model). One user abuse attack includes sharing unwanted information or documents with a particular target user.

4. **Resistant to impersonation attacks**: Our model must be resistant to impersonation of any of the parties (server or clients). It should also inhibit impersonation of other users.

5. **Makes phishing difficult**: Skiff seeks to prevent adversaries from compromising user accounts even if their

password is compromised, including using 2FA, and and/or hardware tokens.

6. **Uncompromisingly usable:** Skiff is significantly more private than than the vast majority of email and collaboration platforms. Given this, building a usable, responsive, and intuitive product with these security properties is critical to ensuring that users do not switch to less-secure alternatives due to poor usability.

2.2 Open Source

Open-sourcing enables everyone to review, use, and contribute to Skiff's products - making our community stronger and furthering our mission. The Skiff Mail client has been completely open source since day one. All Skiff cryptography libraries, such as a typed envelope library for versioning and authenticating data, are open-source and usable by other developers. Finally, Skiff's UI library is also open source, allowing others to build components using our privacy-first design system.

3 System design

3.1 Overview and encryption protocols

Public-key authenticated encryption allows us to securely and privately share access to end-to-end encrypted documents in our security model. Under this schema, each user is issued a long-term public signing key and a medium to long term public key for encryption. Each public key is associated with a corresponding private key generated using Curve25519. We use tweetnacl-js as our encryption library for both asymmetric public-key authenticated encryption (tweetnacl.box) and secret-key authenticated encryption using xsalsa20-poly1305 (tweetnacl.secretbox). Both algorithms ensure both confidentiality and authenticity of encrypted data (AEAD).

While a user's encryption and signing public keys can be freely shared and used to securely send or verify information, all private keys - which can decrypt information or generate signatures - must be kept private. We discuss how this is done in the following section.

3.2 Login, creating accounts, and private key storage

For a new user Alice, our account creation and login system is designed to:

1. Keep Alice's private keys safe
2. Ensure sensitive information - such as Alice's password and private keys - are never sent beyond her browser window, and
3. Resist brute-force and dictionary attacks.

Account creation occurs according to the following process:

1. Bob enters his email and generates a secure password (more than n digits, upper/lower case letters, numbers, and special characters). Random encryption and signing keypairs are generated (tweetnacl.js) for Bob's account in-browser.

2. On the browser, we run Argon2id to derive a symmetric key from Bob's password. Using HKDF, this symmetric key is used to generate one symmetric key for login (using the Secure Remote Password protocol), and another symmetric key for encrypting Bob's secrets (i.e. private keys). This second key is called Bob's `password_derived_secret`.

3. We use Bob's `password_derived_secret` to encrypt sensitive data associated with Bob's account; this encrypted data is called Bob's `encrypted_user_data`. This includes Bob's private keys but not his password. This encrypted information is stored by our server but can only be decrypted with Bob's `password_derived_secret`. The next time Bob logs in, Bob downloads his `encrypted_user_data` from the server, then decrypts the `encrypted_user_data` in-browser by using the `password_derived_secret`. The `password_derived_secret` never leaves the browser.

In this system, Bob's public keys are publicly visible and shareable with other users, while his private keys are encrypted end-to-end. His password and `password_derived_secret` are never stored, not even as encrypted data. Bob's `password_derived_secret` and password are also never sent over any network, even as encrypted data.

We use the secure remote password (SRP) protocol to authenticate user login. After Bob is authenticated using SRP, the server sends Bob's `encrypted_user_data` as well as a signed JSON Web Token (JWT) to indicate that Bob has properly logged in within a certain amount of time. In our security model, the time-limited JWT is generally used for "read-only" operations, including downloading encrypted documents. This is discussed in further detail below.

3.3 Crypto wallet login

Skiff also supports signup and login with a crypto wallet - currently MetaMask and Brave Wallet. When you connect to Skiff with a new Ethereum wallet (with public key `eth_pubKey`, Skiff first verifies an attestation that you own a particular address. To do this, Skiff generates a challenge c message with your wallet address and a random string for you to digitally sign `sign(c,nonce)`, all done through Metamask's interface. This challenge-response authentication

model prevents man-in-the-middle attacks and other impersonation strategies that could compromise your information's security.

Now, Skiff randomly generates a password and encrypts it with your public key derived from your wallet using the Metamask API. Skiff stores this encrypted result (which can be decrypted with your MetaMask wallet) for future reference. Your unencrypted password never leaves your browser (thereby preserving end-to-end encryption) and only you hold the keys to unlock your encrypted password in your wallet.

Today, tens of millions of people use crypto wallets every month. By allowing these individuals to port their crypto identity into Skiff, we hope to bring private collaboration products to a far larger audience through an existing and well-understood line of products. Our team is excited about future possibilities enabled by using crypto wallets as public key infrastucture.

3.4 Two-factor authentication (2FA)

On login, a user can setup two-step authentication to add an additional level of security to their account. Currently, we support two-step authentication using a one-time password from the following authenticator apps: Authy, Google Authenticator, and Duo Mobile.

If Alice chooses to set up two-step authentication, she generates a secret using an authenticator library (`otplib`) in-browser; this is used to display a QR code to her. Alice is then prompted to register her device using the QR code, read an OTP from the device, and enter that OTP once. If the code is successfully entered, Alice's 2FA secret is sent to the server over HTTPS.

When the server receives the 2FA secret, it encrypts that secret using secret-key authenticated encryption (tweetnacl.secretbox), and stores it in Skiff's database.

This 2FA key is later decrypted by the server and used to check Alice's one-time password when she tries to login again. Note: The cloud database provider adds an additional layer of symmetric encryption with key rotation. WebAuthN can be integrated as an additional verification method here.

In end-to-end encrypted systems, two-factor authentication does not notably increase privacy, as all data is already kept private to users. However, 2FA can be a critical defense against phishing, device or password manager compromise, or other attacks where a user may unwittingly expose their password.

3.5 Account recovery and password changes

When users forget their password, it's convenient for them to have a way to recover their account and reset their password. This is achieved using a recovery key (a symmetric key similar to the `password_derived_secret`). A user can enable account recovery in their settings, which generates a recovery key. The recovery key is used to encrypt the user's private data (i.e. private keys); this encrypted user data is stored by Skiff but inaccessible.

When a user requests account recovery, their identity is first verified through email. The server sends an email to the user containing a random eight-character alphanumeric passcode which they can use to prove access to their email account. After this step, the user enters their email, recovery key, and a new password. The client hashes the recovery key, which is sent to the server. The server checks if the hash matches the stored recovery key hash, and that the client retains the correct email code. If both match, the server sends the encrypted user data to the client, which then decrypts it with their recovery key. Note that storing a hash of the recovery key is not like storing a hash of a user's password; while a password may be predictable and reused, the recovery key is a randomly generated symmetric key.

From this point, the process is similar to choosing a new password when an account is created - a `password_derived_secret` is derived from the new password, which is then used to encrypt the user's private data, and the encrypted user data is uploaded to the server, replacing the previous data encrypted with the old `password_derived_secret`.

In future implementations, account recovery can be accomplished using *n*-of-*m* Shamir Secret Sharing to maintain end-to-end encryption while increasing usability. For example, using 2-of-3 secret sharing, one secret share could be stored in the user's device, another stored by Skiff's server, and another provided for "paper" storage. In this model, usability may be significantly higher for many use cases.

4 Skiff Pages: Documents and Files

A Document (or, known as a "Page" in-app) is the fundamental unit of collaboration on Skiff. When interacting with our platform, users create, share, and save documents of different kinds, including rich text documents, PDFs, and folders.

Skiff's document model keeps both document metadata and document contents private to shared collaborators (and hidden from all others, including Skiff). Document metadata includes information including the document's title, icon, icon color, and more. Document contents include the data stored inside a document (such as the content of a text document or spreadsheet). Both metadata and contents are end-to-end encrypted - keeping all content, title, and other information private to document collaborators. In the next sections, we describe how document end-to-end encryption is maintained, including across sharing, unsharing, and link sharing.

4.1 Document encryption model

Every document is associated with a short-term symmetric `session_key` as well as an asymmetric "hierarchical" keypair. The `session_key` is used to encrypt all document con-

tents and metadata that are stored on the server. In order to support real-time collaboration and simple sharing mechanisms, all collaborators - say Alice and Bob - have access to the same session_key. However, because Alice and Bob have different asymmetric key pairs, they each have unique encrypted copies of the same session key (encrypted with public_key_Alice and public_key_Bob, respectively).

Using this symmetric session_key, we can outline processes for creating, editing, and sharing documents. In the following section, we share more about how a document's hierarchical key can be used for constructing a scalable filesystem out of many thousands of documents.

4.2 Opening a document d_1

To open a document, Bob:

1. Downloads the stored copy of session_key_d_1 that has been encrypted with public_key_Bob. Call this key session_key_d1_Bob.
2. Decrypts session_key_d1_Bob using private_key_Bob.
3. Using session_key_d1_Bob, Bob is now able to decrypt, read, and edit the contents or metadata of d_1.

In the following sections, we describe the process by which Bob downloads and recursively decrypts a filesystem containing thousands of documents on Skiff.

4.3 Sharing a document d_2

Reading and editing document d_2 requires access to the session key session_key_d2. Say Alice creates a document d_2 (and is the only shared user on d_2). Now, she wants to share a document with Charlie. Alice:

1. Sends a request to retrieve Charlie's public key. In future sections, we describe a mechanism for out-of-band public key verification to reduce trust in the network and enable users to independently verify users' public keys.
2. Encrypts session_key_d2 with Charlie's public key, and sends this encrypted key for future storage for Charlie. Alice also includes a signature on the encrypted key in the request.
3. When Charlie signs in and loads the Skiff dashboard, Charlie will be able to access all document IDs and session keys - now including session_key_d2_Charlie - that he can access.
4. Charlie decrypts the session key, queries the server for the metadata and contents of d_2, and can now read the document.

5 Real-time collaboration

Real-time collaboration among shared users on a document is end-to-end encrypted using the document's session key. On Skiff, conflict resolution is performed using a CRDT, which allows each collaborator to maintain an in-browser copy of the document and perform change resolution as live document updates are received from other users.

To initiate a collaboration session using the CRDT, two shared users open the document and create a WebSocket connections to a shared WebSocket "room" allowing messages to be passed to other participants. They include the unique identifier of the document for collaboration. At this point, both users begin broadcasting end-to-end encrypted updates to the CRDT through this WebSocket connection to other users listening for collaboration updates. Each user decrypts the CRDT updates (using the document symmetric key) and applies these updates to their local copies of the document data structure. After applying updates, all collaborative users arrive at a final version of the shared document.

6 Building a filesystem

In order to support user filesystems in Skiff, each document d_c also contains a unique identifier representing its parent, d_p, which can be null to signify that a document is at the root of the filesystem. Filesystem construction requires changing behavior for sharing and unsharing, particularly around scalability for large or many documents, which is explained in the following sections. Skiff's collaboration models are designed to efficiently scale to large numbers of documents and users. In order to maintain absolute privacy of all document metadata and contents, these parent-child relationships are only expressed in document UUIDs (which reveals nothing about document titles, for example).

6.1 Scalable sharing and unsharing

Scalable sharing and unsharing requires rethinking the basic document cryptographic model. While sharing limited numbers of documents may be possible by generating per-user copies of encrypted keys, an $O(n)$ sharing operation for an organization with 10,000+ documents is unscalable and very prone to race conditions.

Given this, we introduce the idea of per-document hierarchical keys - asymmetric keypairs generated for each document d_p used to encrypt and decrypt each child document d_c's private keys. Now, each document maintains:

1. A per-document, symmetric session key used to encrypt metadata and contents, and
2. A per-document asymmetric poly1305 key pair (public_hierarchical_key_d_c, private_hierarchical_key_d_c) used to share

session_key_d_c and private_hierarchical_key_d_c with d_c's parent d_p by encrypting this private key with d_p's public_hierarchical_key_d_p.

Now, a user u_p shared on document d_p will maintain a copy of d_p's session key (session_key_d_p) and private hierarchical key (private_hierarchical_key_d_p) encrypted with u_p's encryption private key. When u_p wants to access d_p, u_p can use their encryption private key to do so.

To access d_c, we introduce an additional document field "parentKeysClaim" that stores d_c's private_hierarchical_key_d_c and session_key_d_c encrypted with d_p's public hierarchical key. When u_p wants to access d_c, u_p first decrypts private_hierarchical_key_d_p, and then decrypts d_c's private hierarchical key and session key. This model can now continue recursively to decrypt the entire tree of documents beginning with d_c!

6.2 Sharing a document d_r (recursive filesystem)

Similar to our simple sharing case above, Alice now wants to share a user Bob on a document d_r, where d_r has is the root of a tree of hundreds or thousands of documents.

In this case, Alice simply encrypts d_r's private_hierarchical_key_d_2 and session_key_d_2 with Bob's private encryption key. Bob can now decrypt all of d_r's child documents recursively. This sharing operation is magically $O(1)$!

6.3 Unsharing a document d_3 (recursive filesystem)

We now assume that Alice wants to unshare Bob from a root document d_3, where d_3 is the root of a tree of hundreds or thousands of additional documents. To unshare Bob from document d_3, Alice simply deletes Bob's encrypted key copies from d_3's key register.

Given this formulation, unsharing is also $O(1)$ for a client.

6.4 Expiring Access

Say Alice wants to share user Dorothy on a document with an expiration date. User Dorothy's access to d_4 can be expired as follows:

1. Alice - who is going to unshare Dorothy - prepares a request to expire Dorothy's access to d_4 with a given authenticated expiry_date

2. After checking permissions, and ensuring expiry_date $>$ today_date $+ \delta$, the server adds an (authenticated) expiry_date field to Dorothy's permission entry.

3. When d_4 is modified or re-downloaded by any shared user U after Dorothy's permission expires, U's client recognizes that the permission entry has expired and re-encrypts document d_4 (as in typical unsharing). Under this model, Dorothy is blocked by the server from accessing d_4, and all future versions of d_4 will be encrypted with a new key. This maintains our desired characteristics of end-to-end encryption where only shared collaborators can decrypt document contents. Although there may be a period of time between "expiring" Dorothy's access and another user re-encrypting the document, the document will remain unchanged during this period, and Dorothy will be unable to access future modified versions.

6.5 Link sharing a document d_2

Link sharing is a critical usability feature for modern collaboration. Unlike all link sharing schemes in use by collaborative products today, Skiff's link sharing mechanism maintains end-to-end encryption such that not even Skiff can gain access to a document's URL. In this section, we discuss **"end-to-end encrypted links."** These links enable sharing single pages, embedded files, and entire recursive subtrees of documents, such as wikis, blogs, or websites.

Skiff's end-to-end encrypted links store information in the URL that remains private to the client (using a URL fragment) and employ an authentication technique extremely similar to the user login method documented above. In order to generate a sharable link for d_2, Alice:

1. Generates a random key link_key_d2, and encrypts session_key_d2 with link_key_d2, and encrypts the link_key with session_key_d2. It is critical that Alice encrypts the link key with the session key so she can recover the document link when redownloading the document (for example, after logging out and logging back in).

2. Using link_key_d2 as a "password," Alice generates a salt and verifier used for Secure Remote Password. As in user account creation, the salt, verifier, and encrypted keys are stored by the server. However, it is impossible for the server to decrypt the link_key or the session_key.

Alice now has access to a URL link https://<client_name>/<docID>#link_key that can be securely transmitted to another individual, either through an end-to-end encrypted messenger or another communication channel.

Note: As noted above, Alice encrypts link_key_d2 with session_key_d2 in addition to encrypting session_key_d2 with link_key_d2 in order to recover the full link URL when accessing the document.

In order for Bob to access d_2 given the link URL, Bob:

1. Parses the URL to extract `link_key` and `docID`
2. Performs the SRP login process to request the salt and send a proof back to the server, which responds with the encrypted `session_key_d2`
3. Bob then decrypts `session_key_d2` with `link_key_d2`. At this point, Bob can download and read document d_2.
4. If Alice has enabled editing for the link, Bob sends a copy of `session_key_d2` encrypted with Bob's public key. The server stores this key and a new permission entry on d_2 for Bob to access in the future.

In the future, in order for Alice to access the link, Alice uses the version of `link_key` encrypted with the `session_key` to reconstruct the link URL.

6.6 Email invitation links for document d_3

Users frequently want to invite friends and collaborators (who do not have Skiff accounts) to their Skiff documents or workspaces. In order to allow these new collaborators to create accounts and start collaborating, Skiff generates a temporary, publicly viewbable or editable link that is included with an onboarding invitation link.

On sharing a user who does not have a Skiff account, a temporary public link is sent via the Skiff invitation system and embedded in an account creation link, much like sending an email from a user of an end-to-end encrypted email service to an external user. When a user with the given email creates an account, the public link information - including d_3's UUID and link secret - is used to perform SRP and join document d_3 as a viewer or editor.

7 Document chunks and expiration

In our data model, document contents are represented as an array of chunks. Each chunk individually contains a piece of encrypted data (encrypted with the document `session_key` and authenticated by the user encrypting the individual chunk). Splitting a document into chunks can yield significant efficiency gains depending on the size and type of document stored.

The sequence of chunks is authenticated to prevent unintentional reordering or omission of pieces of the document by Skiff servers or any malicious actor. This includes authenticating the chunk's zero-indexed sequence number and a boolean flag indicating whether chunk C_n is the final chunk in the document. When adding a new chunk, the client updates the signature on the previous chunk (C_{n-1}) to flip this boolean field (to indicate that the old previous chunk is no longer the final chunk).

To support chunks with expiring content (which could be used to support expiring documents or messages), we add an `expiry_date` property as well as an `expiring_content` field to store an expiring encrypted message. When included, the `expiry_date` of a chunk is also authenticated with the entire chunk. After `expiry_date` has passed for chunk c_a in document d_m, the server:

No longer includes the `expiring_content` of chunk c_a when clients download a copy of d_m, and Regularly deletes all expired `expiring_content` fields from the database; for example, the server will run a job every day to delete `expiring_content`.

In addition to requesting a new copy of the document from the server, the client removes a chunk from its stored copy of the document whenever it expires, just in case a user's connection to the server is severed.

8 Skiff Mail

Skiff Mail extends the Skiff Workspace to enable privacy-first and end-to-end encrypted communication. This section breaks down different cases for sending and receiving messages on Skiff Mail, including to and from external email addresses.

8.1 Email sending and receiving - Skiff Mail to Skiff Mail

Sending an email from one Skiff address to another is very similar to sharing a Skiff document with another user. Consider the scenario where Alice wants to send an email e_a to Bob, where both Alice and Bob are Skiff users (say `alice@skiff.com` and `bob@skiff.com`).

When Alice clicks "send" in her Skiff Mail client, Alice's Skiff client generates a random symmetric key sk_{e_a} to encrypt the email subject and content corresponding to e_a. This symmetric key sk_{e_a} is subsequently encrypted with Alice's and Bob's encryption public keys for future access, as with a document shared on Skiff's collaboration platform.

Now, as in Skiff's collaborative workspace, the receiver can log in and decrypt the email's symmetric key to gain access to the sent message.

Under this model, any email sent and received via Skiff Mail is completely end-to-end encrypted and private to Alice, Bob, and any other recipients. No third party (not even Skiff) ever sees or processes the message content.

8.2 Email sending - Skiff Mail to External

When Alice sends an email e_b from her Skiff Mail client to an external email address, Alice's Skiff client will still generates a symmetric key sk_{e_b} and encrypt this key with Alice's public key for future access. However, in order to send the mail to an external, unencrypted email address, Alice's client also

encrypts this email with the public key of a decryption service that temporarily processes this mail for external receipt.

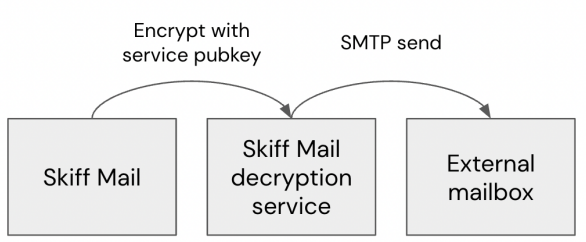

Figure 1: Mail sending from Skiff Mail to an external service.

To send the email e_b externally to Charlie (say `charlie@external.com`), Alice's Skiff client now encrypts sk_{e_b} with the public key of the decryption service `pubkey_decrypt`. This decryption service ephemerally decrypts the message e_b, composes outgoing MIMEs needed to send e_b to Charlie at `charlie@external.com`, and sends e_b message to the external email address.

Now, Charlie - a non-Skiff recipient - can read Alice's sent mail. The key used by the decryption service to send this message externally is discarded and deleted, leaving no way for a third party (not even Skiff) to access any sensitive information inside e_b in the future. As in the Skiff-to-Skiff mail case, Skiff does not store an unencrypted copy of this externally sent mail.

8.3 Email receiving - External to Skiff Mail

Receiving mail from an external, unencrypted email address requires similar operations to the sending case. Say Alice is receiving mail from another external address, `dave@external.com`.

When Alice's Skiff account (`alice@skiff.com`) receives an incoming unencrypted email e_d from Dave, the mail e_b is immediately encrypted by a discrete encryption service with Alice's Skiff public key `public_key_Alice` with a random symmetric key sk_{e_d}. Similar to sharing a Skiff document, Alice now maintains a copy of sk_{e_d} encrypted with her personal public key.

Now, when loading her Skiff inbox, Alice can decrypt the encrypted copy of email e_d in order to read the message. The encryption service discards and deletes its initial randomly generated copy of sk_{e_d}, now ensuring that every message inside Alice's inbox remains end-to-end encrypted and completely private to her.

This encryption step - taking the unencrypted, external email e_b and transforming it into an encrypted message on Skiff Mail - ensures that Skiff never stores copies of users' unencrypted mail, even when sent by an external service.

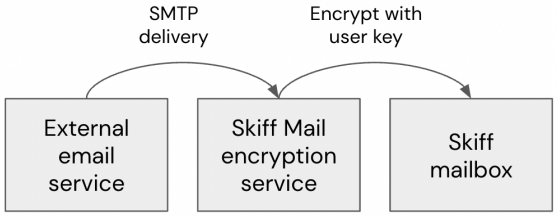

Figure 2: Mail encryption for messages sent from an external, unencrypted provider to Skiff Mail. Skiff never stores an unencrypted copy of user mail.

8.4 Mail Threading

Threading new mail is a critical usability and privacy preserving operation for all Skiff Mail users. Threading ensures that each individual message in a Skiff user's inbox is associated with all related replies, reply-alls, and forwards. In long threads with hundreds of messages, various replies and forwards, and different recipient lists, threading can be complex and difficult.

However, threading has been a critical and largely solved problem as email clients have become used by billions of people worldwide. Skiff Mail uses the JWZ threading algorithm, which is documented by JWZ at this link. Using the JWZ threading algorithm, which relies on various message identifiers, Skiff Mail assigns thread IDs to incoming emails, whether from Skiff or from an external service. On receipt of a new email, the algorithm is run again, ensuring that replies and forwards are properly grouped with their corresponding messages.

9 Public key verification

Private communication requires trust in mechanisms to receive and verify other users' public keys. Skiff allows other users to view and verify other users' public signing keys through a user interface for "verification phrases" - a bip39 (or alternate) encoding of another user's signing public key.

Each individual's encrypted user data (see above) includes a data structure to store public signing keys for other verified users. Given this model, when user Alice marks user Charlie as "verified," Alice stores a copy of Charlie's public signing key in her encrypted user data (or another end-to-end encrypted data store maintained by the server). In all future interactions with Charlie (adding to a group, sharing on a document) this stored copy of Charlie's public signing key is verified against the one distributed by Skiff's server, thereby ensuring that Alice has a correct and up-to-date copy of Charlie's public key.

10 Conclusion

Skiff's security model is designed to enable end-to-end encrypted, scalable, and decentralized collaboration and communication agnostic to document type. In this model, all sensitive information - from document titles to email content - is kept private to creators and intended recipients and never shared with a third party.

Today, privacy is often neglected by the communication and collaboration products trusted by billions of people. Through innovation, community, and new security technology, we at Skiff hope to build a new, user-centered, and end-to-end encrypted ecosystem. New technology is constantly helping unlock this transformation, from crypto wallets (which provide public keys to millions of users) to client-side search indexing algorithms.

The model presented in this paper is the beginning of a platform to build even more private, performant, and user-friendly consumer software.

11 Contact

Please reach out to hello@skiff.org with any questions or suggestions!