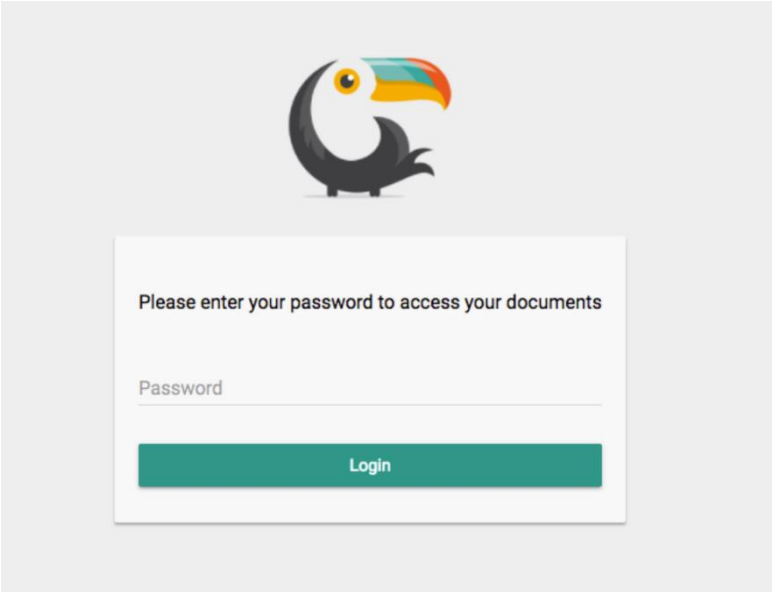
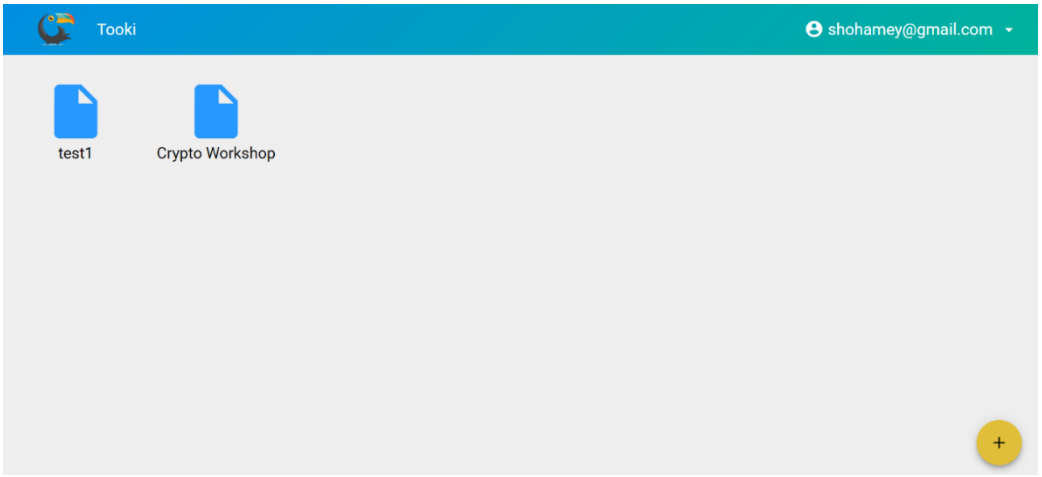
# Secure Collaboration Editing

Itay Radozki, Roey Shamir, Eyal Shoham



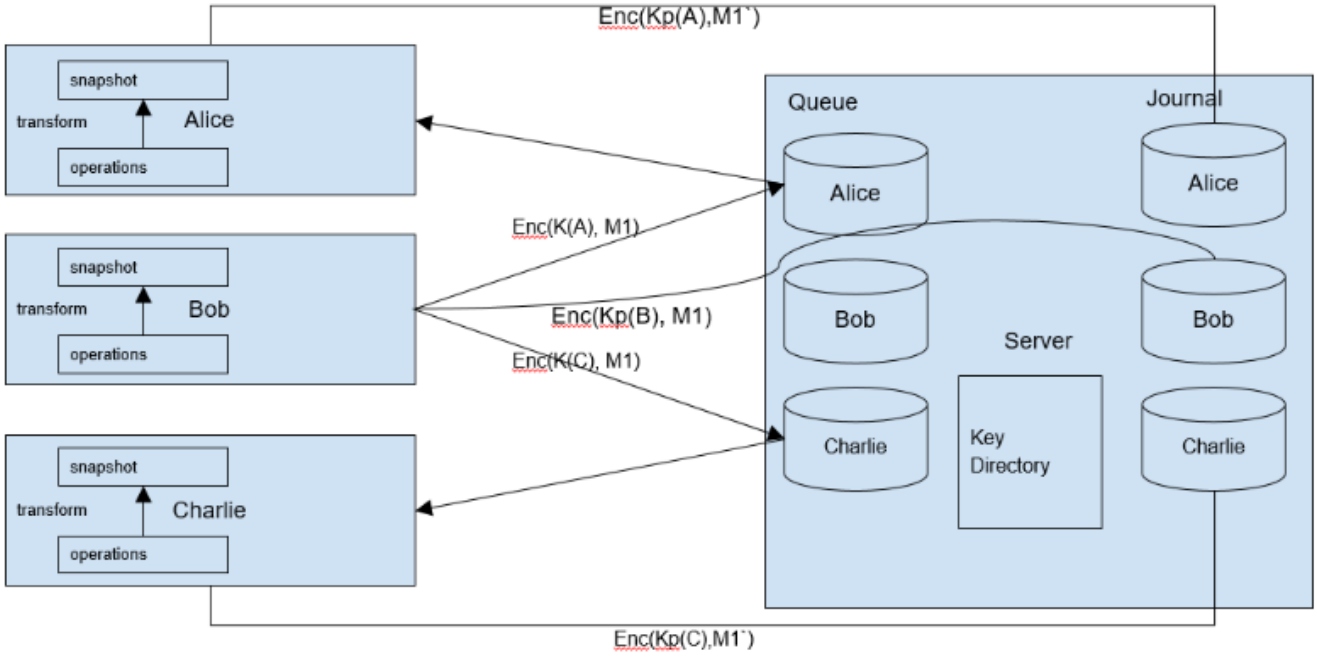


## Project overview and Motivation

Our project was to create encrypted collaborative editor. In our day's end to end encryption are very popular among chat applications like whatsapp, telegram and etc.
The most popular collaborative editor like google docs is not end to end encrypted, meaning that if someone get access to the server he can retrieve all the information.
We chose to implement our editor as a web base solution. We used the signal protocol (developed by open whisper systems) for end to end encryption (also used by whatsapp, facebook messenger). The signal protocol provide end to end encryption, forward secrecy and future secrecy.

## Scheme

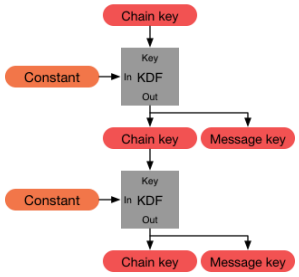


# Signal protocol

- Encryption and authentication of messages
- The protocol provides:
    a. confidentiality
    b. integrity
    c. authentication
    d. participant consistency
    e. destination validation
    f. forward secrecy
    g. backward secrecy (aka future secrecy)
    h. message repudiation
- X3DH key agreement protocol
    a. Implement Each user have Identity key, a lot of one-time prekeys, signed prekey=> Session key for the ratchet key pair
- double ratchet (DH+Symmetric-key)
    a. Used by two parties to exchange encrypted messages based on a shared secret key. Typically the parties will use some key agreement protocol (such as X3DH) to agree on the shared secret key. Following this, the parties will use the Double Ratchet to send and receive encrypted messages. The parties derive new keys for every Double Ratchet message so that earlier keys cannot be calculated from later ones. The parties also send Diffie-Hellman public values attached to their messages. The results of Diffie-Hellman calculations are mixed into the derived keys so that later keys cannot be calculated from earlier ones. These properties gives some protection to earlier or later encrypted messages in case of a compromise of a party's keys.
    b. symmetric-key ratchet (for backward secrecy):
        i. Every message sent or received is encrypted with a unique message key. The message keys are output keys from the sending and receiving KDF chains. The KDF keys for these chains will be called chain keys.
            1. We define a KDF as a cryptographic function that takes a secret and random KDF key and some input data and returns output data. The output data is indistinguishable from random provided the key isn't known. If the key is not secret and random, the KDF should still provide a secure cryptographic hash of its key and input data.
        ii. The KDF inputs for the sending and receiving chains are constant, so these chains don't provide break-in recovery. The sending and receiving chains just ensure that each message is encrypted with a unique key that can be deleted

after encryption or decryption. Calculating the next chain key and message key from a given chain key is a single ratchet step in the symmetric-key ratchet.

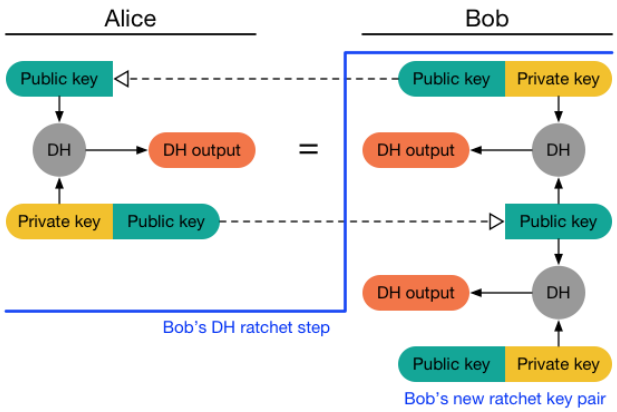
iii. Because message keys aren't used to derive any other keys, message keys may be stored without affecting the security of other message keys. This is useful for handling lost or out-of-order messages.




c. DH ratchet (forward secrecy - can’t passively decrypt future messages)

i. Each party generates a DH key pair which becomes their current ratchet key pair. Every message from either party begins with a header which contains the sender's current ratchet public key. When a new ratchet public key is received from the remote party, a DH ratchet step is performed which replaces the local party's current ratchet key pair with a new key pair.

ii. This results in a "ping-pong" behavior as the parties take turns replacing ratchet key pairs. An eavesdropper who briefly compromises one of the parties might learn the value of a current ratchet private key, but that private key will eventually be replaced with an uncompromised one.



## Server

- Queue for each client
    a. **Reason**: The queue is relevant when the client is not connected to the server and other clients send operations.
    b. The queue holds the operations from other clients that weren't read by the client.
    c. The operations in the queue are encrypted using Signal Protocol
- Journal for each player
    **a. Reason:** The journal is relevant when the client read operations, sign out and sign in again. Then he will be able to get all the operations again
    b. The journal holds the operations from all clients (include journal owner).
    c. The owner sends the operations encrypted with his **half of the PBKDF2 (AES – GCM).**
    d. The owner sends the "raw" operations **before** transforming them.
- Key directory:
    a. The client private key encrypted with a his PBKDF and AES-GCM algorithm
    b. Public keys of all clients
- What the server knows:
    a. Who are the connected users
    b. Timing and amount of messages that every user sends
    c. Size of the messages

# Client

- Editing the document
    a. Sends a message to the server with the client's operation - Insert / Delete, Version, Unique hash
    b. Messages are encrypted with signal protocol
- Memory:
    a. Snapshot
    b. Exchanged keys for every other client
    c. Connected users list
- Changes:
    a. Sends a message to the server with the client's operation: Retain, Insert / Delete, Version, Unique hash
    b. Messages are encrypted with the session key for every user
- Transformation:
    a. Tie break for the same version is done by comparing the messages hash
    b. Increments the version after each transformation
    c. Use our **transformation algorithm**
- New Member:
    a. Must be invited by someone from the group
    b. They exchange keys
    c. Generate the keys when registered
- Login
    a. PBKDF2
        a. The client enter a password
        b. We apply PBKDF2 hash algorithm.
        c. Send the first half of the result to server
        d. Keep the second half as a secret passphrase
        e. Every login the first half of the PBKDF2 is sent to server for authentication. And the second half use to encrypt the journal and the private key
    b. Get the private key from the server and decrypt it using the second half of the PBKDF2.

# Transformation Algorithm

- Since in our implementation all the messages are encrypted end to end. The server can't decide the order of the incoming messages. So we need the client to know the order of the messages from all users. We construct an algorithm to determine the order of the messages by version and unique hash in case of tie break.