University of Pisa

Cybersecurity Project 2018
Master's Degree in Computer Engineering
Filippo Scotto – 501743

SecureChat

A SECURE WEB CHAT MADE IN JAVASCRIPT

Contents

Introduction

The app and the technologies

The App

SecureChat is a client-server Instant Message service written in **Javascript**, that used cybersecurity mechanism to provide a **TOFU**[1] *end-to-end encryption*. Users must be registered to the service in order to use the chat: at the moment of the registration[2] each user must provide a 1024-bit RSA public key and verify their identity.

The users trust the server authority on public key certification, also they trust that the first time the server provides a public key of a certain user it is correct. This assumption is required to correctly fulfill the end-to-end encryption requirement (*the server, if compromised, could be a **Man-In-The-Middle** and provide the wrong public key for a user and by that it could be able to read the messages of the next sessions*).

During the login phase the users must specify their username, their public keys and their private keys (***PEM** format strings*). The login message will be signed by the user using its private key (and to guarantee protection from replay attacks the user will add a nonce randomly generated). The server then will verify the signature and generate a fresh session key for the user, and if everything is ok the user is considered logged in. After the login phase, the users can see the list of the connected and available users and they can request a secure chat by clicking on the username of a specific user. For example consider a user Alice (sender) and Bob (recipient). The server will provide to Bob the public key of Alice (signed with the server private key and with a nonce for freshness). If Bob accepts the request it will generate a session key (encrypted with the public key of Alice and signed with the private key of Bob) and send it to the server. The server will verify the authenticity and the freshness of the key and it will send it to Alice. Alice *trust* the server authority on the session key, so if it appears to be signed by the server Alice will take the key for good. Finally Alice and Bob can start an encrypted chat where they are the only entities able to read the messages.

The Technologies

The server is an HTTP web server powered by **Node.js**[3] and it hosts the web app accessed by the clients. Both client and server are written in Javascript and they share

[1] Trust On First Use

[2] **Registration** is not going to be a part of this project, everything works under the assumption that the users are already registered and they have provided in a safe way their public key (e.g. in person).

[3] Node.js is a JavaScript runtime built on Chrome's V8 Javascript engine: https://nodejs.org/en/about/

the same library for the encryption (**forge**[4]). Communication is possible thanks to the **Socket.IO** library.

- All the ciphertextes are encrypted using **AES-CBC-128**;
- Each user has a pair public key and private key (**RSA-1024bit**)
- The Digital Signature of the public keys and the MAC are done using **SHA-256**;
- All the random values are generated using **Fortuna**[5] algorithm (a criptographically secure PRNG);
- All the messages are sent using **JSON** format.

[4] Forge is a library written by digitalbazaar available at https://digitalbazaar.com/forge/

[5] Fortuna Algorithm: https://en.wikipedia.org/wiki/Fortuna_(PRNG)

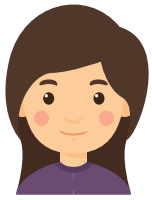
The Protocol

Protocol description and BAN Logic

The protocol is composed by three main phases:

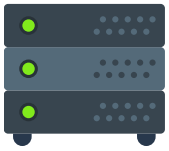
- Login Phase
- Listing Phase
- Chat Phase

Login Phase



1. *login*, Alice, N_A, $\{H\}_{PrivA}$

2. *grant*, N_A, $\{K_{AS}\}_{PubA}$, $\{H\}_{PrivS}$



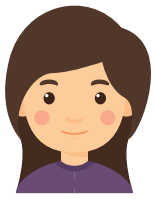
Pub_A, $Priv_A$, Pub_S

Pub_A, Pub_S, $Priv_S$

Let's consider the user Alice, the goal of this phase is to provide to both the user and the server a shared 128 bit key **K_{AS}** that should be used for the following interactions between Alice and the server:

1. **Alice** prepare a *login message* containing her username, a fresh nonce randomly generated and a signature. Then she sends the message in the clear to the **server**;
2. The **server** verify the identity of the user, then it generates a fresh session key **K_{AS}**. If everything is ok, it sends to **Alice** a *grant message* containing the session key and the previous N_A (signed with its private key);
3. **Alice** verify the key authenticity using the public key of the server and check the nonce Na.

Listing Phase



3. $\{list\}_{KAS}$

4. $\{<list\ of\ users>\}_{KAS}$

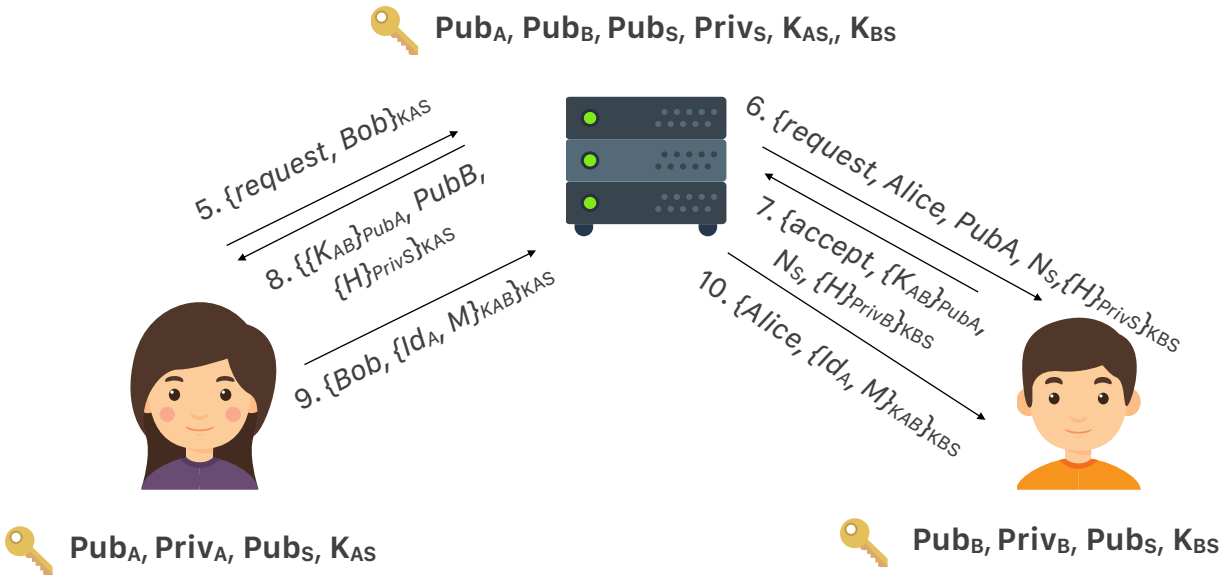
Pub_A, $Priv_A$, Pub_S, K_{AS}

Pub_A, Pub_S, $Priv_S$, K_{AS}

Alice is now logged in and she can use the session key to encrypt the messages:

1. **Alice** sends a *list message* encrypted using the session key to the **server**;
2. The **server** sends a list of the available users encrypted with session key;

Chat Phase



Let's suppose that Alice finds out that the user Bob is connected and available, so she decides to start a chat session with him. The goal of this phase is to provide to both Alice and Bob a session key to exchange messages:

1. **Alice** sends a *request message* encrypted using the session key to the **server**. The message contains the username of the requested user (**Bob**) and it's encrypted with the session key K_{AS};
2. The **server** sends to **Bob** the request from **Alice** in an encrypted message (using the session key K_{BS}). The message contains the public key of Alice, a fresh nonce N_S and it's signed by the server (*a trusted authority*);
3. Let's suppose that **Bob** accepts the request (*of course, first he needs to verify the signature of the request message*): he sends an encrypted message to the **server** that contains the session key K_{AB} (randomly generated by Bob and encrypted using Alice's public key) and a signature;
4. The **server** verify the response message from Bob. The server can't see the value of the session key chosen by Bob, but it can says if it is fresh by checking the nonce. If everything is fine, it sends to Alice an encrypted message containing the key;
5. After the verification of the message, **Alice** can extract the session key K_{AB}. The following text messages between Alice and Bob will be encrypted using the KAB and delivered by the **server** (that will not be able to read the body of the message, but only the header). Each *text message* has got a message id (intitalized with a randomly generated value, and then increased at each message to avoid store and replay attacks).

BAN Logic Analysis

Let's suppose **Alice** wants to connect to the server to establish a secure chat with **Bob**.

Assumptions:

As we've already said, the protocol works under the assumptions that the server is a trusted authority on keys (both public keys and session keys). The server verify if the session key were really sent by the user and if it is fresh. The other user believes what the server says.

- Alice **believes** K_A is the **public key**(Alice);
- Bob **believes** K_B is the **public key**(Bob);
- Alice **believes** K_S is the **public key**(server);
- Bob **believes** K_S is the **public key**(server);
- The server **believes** K_A is the **public key**(Alice);
- The server **believes** K_B is the **public key**(Bob);
- The server **believes** K_S is the **public key**(server);
- Alice **believes** the server **believes** K_B is the **public key**(Bob);
- Bob **believes** the server **believes** K_A is the **public key**(Alice);
- Alice **believes** the server is an **authority**(K_B);
- Bob **believes** the server is an **authority**(K_A);

- The server **believes** K_{AS} is a **shared key**(Alice, server);
- The server **believes fresh**(K_{AS});
- The server **believes** K_{BS} is a **shared key**(Bob, server);
- The server **believes fresh**(K_{BS});
- Alice **believes** the server is an **authority**(K_{AS});
- Bob **believes** the server is an **authority**(K_{BS});

- Bob **believes** K_{AB} is a **shared key**(Alice, Bob);
- Bob **believes fresh**(K_{AB});
- Alice **believes** the server is an **authority**(**fresh**(K_{AB}));

- The server **believes fresh**(N_S);
- Alice **believes fresh**(N_A);
- Alice **believes fresh**(ID_A);
- Bob **believes fresh**(N_B);
- Bob **believes fresh**(ID_B);

Goals:

- Alice **believes** K_{AS} is a **shared key**(Alice, server);
- Alice **believes** the server **believes** K_{AS} is a **shared key**(Alice, server);
- The server **believes** Alice **believes** KAS is a **shared key**(Alice, server);
- Bob **believes** K_{BS} is a **shared key**(Bob, server);
- Bob **believes** the server **believes** K_{BS} is a **shared key**(Bob, server);
- The server **believes** Bob **believes** K_{BS} is a **shared key**(Bob, server);
- Alice **believes fresh**(K_{AS});

- Bob **believes fresh**(K_{BS});

- Alice **believes** K_{AB} is a **shared key**(Alice, Bob);
- Alice **believes fresh**(K_{AB});
- Alice **believes** Bob **believes** K_{AB} is a **shared key**(Alice, Bob);
- Bob **believes** Alice **believes** K_{AB} is a **shared key**(Alice, Bob);
- Bob **believes** Alice **said** M;

Protocol:

Login phase:

$$\begin{aligned} &M1 : A \rightarrow S \quad login,\ Alice,\ N_A, \{H\}_{K_A{}^{-1}} \\ &\boldsymbol{M2 : S \rightarrow A \quad grant, N_A, \{K_{AS}\}_{K_A}, \{H\}_{K_S^{-1}}} \end{aligned} \tag{1}$$

Listing phase:

$$\begin{aligned} &M3 : A \rightarrow S \quad \{list\}_{K_{AS}} \\ &M4 : S \rightarrow A \quad \{< list\ of\ users >\}_{K_{AS}} \end{aligned} \tag{2}$$

Chat phase:

$$\begin{aligned} &M5 : A \rightarrow S \quad \{request,\ Bob\}_{K_{AS}} \\ &M6 : S \rightarrow B \quad \{request, Alice, K_A, N_S, \{H\}_{K_S{}^{-1}}\}_{K_{BS}} \\ &M7 : B \rightarrow S \quad \{accept, N_S, \{K_{AB}\}_{K_A}, \{H\}_{K_B^{-1}}\}_{K_{BS}} \\ &M8 : S \rightarrow A \quad \{accepted, \{K_{AB}\}_{KA},\ K_B, \{H\}_{K_S{}^{-1}}\}_{K_{AS}} \end{aligned} \tag{3}$$

Actual chat between Alice and Bob:

$$\begin{aligned} &M9 : A \rightarrow S \quad \{Bob, \{ID_A, M\}_{KAB}\}_{K_{AS}} \\ &M10 : S \rightarrow B \quad \{Alice, \{ID_A, M\}_{KAB}\}_{K_{BS}} \end{aligned} \tag{4}$$

Idealized protocol:

$$\begin{aligned} &M1 : A \rightarrow S \quad \{h(N_A)\}_{K_A{}^{-1}} \\ &M2 : S \rightarrow A \quad \{sharedkey(K_{AS}, A, S)\}_{K_A}, \{h(N_A, \{sharedkey(K_{AS}, A, S)\}_{K_A})\}_{K_S{}^{-1}} \\ &M3 : A \rightarrow S \quad \{list, sharedkey(K_{AS}, A, S)\}_{K_{AS}} \\ &M6 : S \rightarrow B \quad \{pubkey(K_A, A), N_S, \{h(pubkey(K_A, A), N_S)\}_{K_S{}^{-1}}\}_{K_{BS}} \\ &M7 : B \rightarrow S \quad \{N_S, \{sharedkey(K_{AB}, A, B)\}_{KA}, \{h(N_S, \{sharedkey(K_{AB}, A, B)\}_{KA})\}_{K_B{}^{-1}}\}_{K_{BS}} \\ &M8 : S \rightarrow A \quad \{\{sharedkey(K_{AB}, A, B)\}_{KA}, \{h(\{sharedkey(K_{AB}, A, B)\}_{KA})\}_{K_S{}^{-1}}\}_{K_{AS}} \\ &M10 : S \rightarrow B \quad \{\{ID_A, M, sharedkey(K_{AB}, A, B)\}_{KAB}\}_{K_{BS}}. \end{aligned}$$

Analysis:
This first part of the analysis is the same for both Alice and Bob. We are going to skip Bob's login and listing phase.
Note: ***S****: The server,* ***A****: Alice,* ***B****: Bob.*

After message **M2**:

$$\frac{A \triangleleft \{sharedkey(K_{AS}, A, S)\}_{K_A}, \{h(N_A, \{sharedkey(K_{AS}, A, S)\}_{K_A})\}_{K_S^{-1}} \quad A \mid\equiv \#(N_A), A \mid\equiv publickey(K_A, A), A \mid\equiv publickey(K_S, S)}{A \mid\equiv S \mid\sim sharedkey(K_{AS}, A, S)}$$

$$\frac{A \mid\equiv \#(N_A),\ A \mid\equiv S \mid\sim sharedkey(K_{AS}, A, S)}{A \mid\equiv S \mid\equiv sharedkey(K_{AS}, A, S)}$$

$$\frac{A \mid\equiv S \mid\equiv sharedkey(K_{AS}, A, S), A \mid\equiv S \Rightarrow sharedkey(K_{AS}, A, S)}{A \mid\equiv sharedkey(K_{AS}, A, S),\ A \mid\equiv \#(sharedkey, K_{AS}, A, S)}$$

Alice believes that the server said that K_{AS} is a shared key between Alice and the server. But Alice believes N_A is fresh and trust the server authority on K_{AS}. Alice believes that the key K_{AS} is fresh and shared with S.

After message **M3**:

$$\frac{S \triangleleft (list, sharedkey(K_{AS}, A, S)), S \mid\equiv sharedkey(K_{AS}, A, S), \quad S \mid\equiv \#(sharedkey(K_{AS}, A, S))}{S \mid\equiv A \mid\equiv sharedkey(K_{AS}, A, S)}$$

The server believes that K_{AS} is a shared key between Alice and the server. The server believes that Alice believes that K_{AS} is a shared key between Alice and the server.

After message **M6**:

$$\frac{B \triangleleft pubkey(K_A, A), \quad B \mid\equiv S \mid\equiv pubkey(K_A, A), \quad B \mid\equiv S \Rightarrow pubkey(K_A, A)}{B \mid\equiv pubkey(K_A, A)}$$

Bob sees that server said that K_A is the public key of Alice, just like what happened after M2, Bob believes that the server said that K_A is the public key of Alice. And since Bob believes also that the server is an authority for the public key of Alice, Bob believes that K_A is the public key of Alice.

After message **M8**:

$$\frac{A \mid\equiv S \mid\sim (sharedkey(K_{AB}, A, B)), A \mid\equiv S \mid\equiv \#(sharedkey(K_{AB}, A, B)), \quad A \mid\equiv S \Rightarrow \#(sharedkey(K_{AB}, A, B), B \mid\equiv \#(sharedkey(K_{AB}))}{A \mid\equiv \#(sharedkey(K_{AB}, A, B), A \mid\equiv sharedkey(K_{AB}, A, B), \quad A \mid\equiv B \mid\equiv sharedkey(K_{AB}, A, B)}$$

Alice sees the server said K_{AB} is a key shared between Alice and Bob, but Alice also trust on the server for the authenticity and the freshness of the key. So Alice believes that the key is fresh and authentic.

After message **M10**:

$$\frac{B \triangleleft (ID_A, M, sharedkey(K_{AB}, A, B)), B \mid\equiv sharedkey(K_{AB}, A, B)}{B \mid\equiv A \mid\sim (ID_A, M)}$$

$$\frac{B \mid\equiv \#(sharedkey(K_{AB}, A, B)), B \mid\equiv A \mid\sim sharedkey(K_{AB}, A, B)}{B \mid\equiv A \mid\equiv sharedkey(K_{AB}, A, B)}$$

Bob believes that the message M is said by Alice. Since Bob believes the key K_{AB} is fresh and shared between Alice and Bpb, then it means that Alice believes K_{AB} is a shared key between Alice and Bob.

Appendix

Sourcecode and link to try the app

Sourcecode

The sourcecode of the entire project is available at:
https://tinyurl.com/Cybersecurity501743

Try it

The application is available and ready to be used at:
https://websecurechat.herokuapp.com/