AIS (Advanced Intergalactic/Interracial/Interesting Secure) Chat

Or, Annie-Isabel-Simon Chat

Design Paper

Annie Ke
Isabel Hurley
Simon Posada Fishman

Applied Cryptography, Fall 2017
Aquincum Institute of Technology
Budapest, Hungary

Table of Contents

Overview and purpose

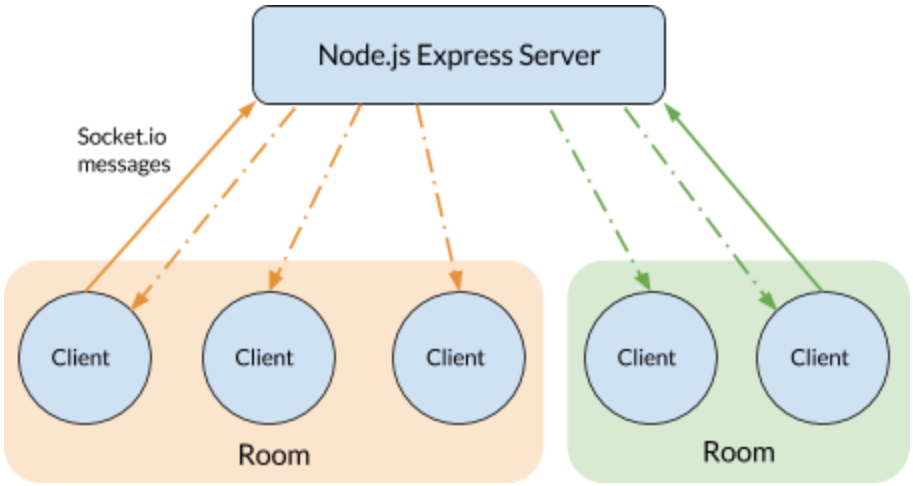
AIS Chat is a messaging application built with privacy in mind. It provides a secure and ephemeral space to have text based conversations. Messages are encrypted end-to-end and only live in a per-session basis. Multiple modes of authentication are supported and users can join and create rooms to communicate with groups.

System Architecture

Client-Server Setup

- AIS Chat uses a React frontend for direct client usage, and an Node.js Express server with socket.io as the backend chat server.
- Chat rooms are implemented through network sockets for client-server real time communication. By using socket.io's built-in rooms and namespaces, we can broadcast socket messages from the server to specific groups of multiple users.
- Each room is identified by a unique roomID, and upon authentication, users may be added to existing rooms or create a new ones. The server keeps track of the users active in a room, storing no message history.

General Operation Scheme



Cryptographic Libraries

Top choices:

Crypto-js

"Growing collection of standard and secure cryptographic algorithms implemented in JavaScript using best practices and patterns. They are fast, and they have a consistent and simple interface."

Stanford Javascript Crypto Library (sjcl)

"Project by the Stanford Computer Security Lab to build a secure, powerful, fast, small, easy-to-use, cross-browser library for cryptography in Javascript."

WebCrypto (W3 specification)

"This specification describes a JavaScript API for performing basic cryptographic operations in web applications, such as hashing, signature generation and verification, and encryption and decryption. Additionally, it describes an API for applications to generate and/or manage the keying material necessary to perform these operations. Uses for this API range from user or service authentication, document or code signing, and the confidentiality and integrity of communications."

Compatibility *Examples* *Node.js implementation*

Conclusion:

Crypto-js is a community driven open source framework, it has the simplest and most friendly interface, but it is the slowest and least professionally endorsed. On the other hand, SJCL and WebCrypto are endorsed by many industry professionals and have been built by crypto experts. Out of those two, SJCL has been around for longer and has a more established interface. WebCrypto is the newest of all and is a W3 specification, it is expected to set the standard for web cryptography. Each browser has its own implementation of the API, and it is widely supported but not universal yet. Of the three, WebCrypto seems to be the least friendly API; however, it greatly outperforms the rest. Due to these advantages and it's promising future as a standard, we are inclined to use WebCrypto. However, we will still consider using SJCL with a tradeoff in performance in case we run with any major issues using WebCrypto.

Attacker Models

Eavesdropping Server

The attacker may hack into the server and view the messages. All messages should be encrypted on the client side, before being broadcast, to prevent this.

Message Replay

The attacker may eavesdrop and replay a message without decrypting it. We include a timestamp in the message format, and time in the clients will be synced from a trusted time source. Messages will only be accepted on a given timeframe.

Impersonation

The attacker may try to assume the identity of someone else in the chat. We use unique signature key pairs for every user, issued with a "trusted CA", and sign every message to verify the identity of the sender and prevent this attack.

Man in the Middle

The attacker may modify messages as the message travels from sender to recipient. We prevent this attack by digitally signing each message to ensure the identity of the sender as well as the integrity of the message.

Key Compromisation

The attacker may uncover the message keys in some way. We do periodic key refreshments so that only a portion of the communications are compromised if a key is uncovered.

Security Requirements

Authentication

AIS chat uses Facebook and Google authentication for users. If a natively-implemented authentication method is desired, AIS Chat may also implement user authentication through Passport.

Secrecy

Secrecy is achieved through the end-to-end encryption of all messages. Only the users active in the room at the time the message was sent can decrypt it. Extra secrecy is provided by the ephemeral nature of the application, since a message history is nonexistent in the server side.

Integrity

Every message is digitally signed to demonstrate authenticity. Nonces are used to protect against replay.

Freshness

The freshness of group keys and messages are assured with a timestamp. All messages are marked with the current time, which we request from the Google Maps Time Zone API, a trusted third party. The timestamp will be encrypted and signed with the rest of the message.

Secure Channel Protocol

Shared Group Key Protocol

Authentication

When a user authenticates itself (either through a common password method or OAuth), a signature key pair is generated in the client and the public key is sent to the a “trusted CA” that we will simulate in the server, which then creates a certificate with

the user's account information and his public signature key, signs it, and stores it in a database accessible by other users.

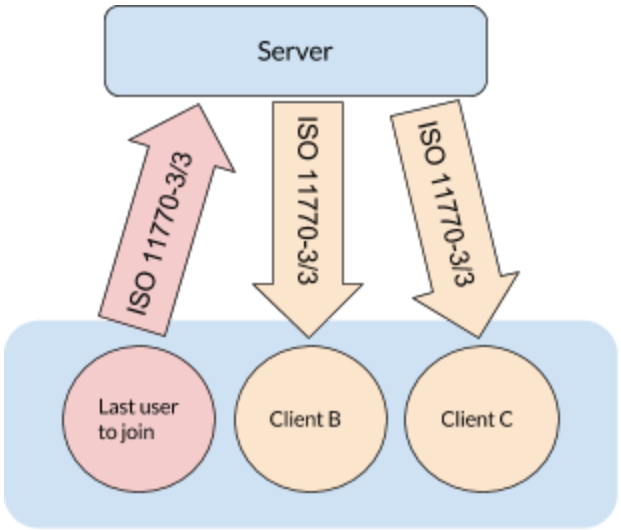
Chat initialization

When an authenticated user joins a chat room, and RSA signature key pair is generated in the client, these pairs are only used for one key exchange, and are re-generated every time the group key is refreshed. A few initialization messages are then exchanged:

1. *Hello* - this message is sent by the new user to every participant already in the chat and includes the basic user info and the public keys of the user, both for encryption and signing.
2. *Acknowledge* - the users in the room reply to the *hello* message with their own information and public keys.

Group Key Establishment

Once all the public signature keys are exchanged, the creator of the chat generates a group key which will be shared by all the participants. She then broadcasts it to the other participants, encrypting it with each participant's public key, and signed with the creator's own signature, as specified in the ISO 11770-3/3 protocol. Upon receiving the group key, each participant decrypts the key and verifies the timestamp and that the key was sent by the creator using the certified signature key database stored in the server. If the key passes the verification, each participant stores the group key for this chat session.



$A \rightarrow B: B, T_A, E_B(A, K_{AB}), Sig_A(B, T_A, E_B(A, K_{AB}))$

ISO11770-3

Group Key Verification

To verify that each member has the correct group key, after storing it, each chat participant broadcasts their own public signature key encrypted with the group key and signed with their private signature key. Upon receiving each group-key-encrypted signature key, each chat participant verifies the signature and that the decryption of the key they just received is identical to the one stored in the server, and if every chat participant successfully verifies signature keys, then the group chat officially begins.

Message Encryption & Format

In future message emissions, each chat participant encrypts the message with the group key, then digitally signs it. Upon receiving a message, chat participants decrypt the message with the group key and verifies the message with the sender's public signature key. Messages are sent to and emitted by the server, which only has access to the encrypted messages. Timestamps are used to protect against replay attacks, and only messages on a given timeframe will be accepted.

Format:

Sender | Signature $_{\text{Sender}}$ **[Enc** $_{\text{Group Key}}$ **(*Header* | M)] | Enc** $_{\text{Group Key}}$ **(*Header* | M)**

Header = Message_media_type | Timestamp | Sender

Additionally, our sockets.io server supports verification methods for when the server has received the message and when it has successfully been broadcasted to all participants.

Key refreshing

Our service will refresh the group key periodically to ensure security. It will do this in two ways; first, the key will only be valid for a certain time period, and the protocol will be reenacted when it expires. Second, every time a user joins the chat or leaves the chat, the key will be reestablished, with the last person to have joined initiating the protocol. These two measures provide better forward secrecy.

Cryptographic Primitives

RSA OAEP

Purpose: public-private key generation for individuals; digital signatures

Key-Transportation (ISO 11770)

Purpose: Key-transportation for generated group key

AES256 in CBC Mode

Purpose: General message encryption