T21 - Groove Galaxy

Safe Sound Cryptography library, infrastructure and functionality

99078: Guilherme Carabalone
99095: João Furtado
93634: Diogo Gonçalves

Secure Documents

Protect: Encrypts the audio content of a JSON file with a symmetric disposable key and a mix of timestamp and a random number to guarantee freshness.

Unprotect: Decrypts the audio content with the disposable key.

Check: Verifies the authenticity of a media document by checking the timestamp, random number, and hash.

Secure Documents: Design

Our library guarantees <u>confidentiality</u>, <u>authenticity</u> and <u>integrity</u> of the media:

- **Confidentiality:** Disposable key is encrypted by a symmetric key which is only shared by the client and server.
- **Integrity:** A HMAC is used.
- **Freshness:** Timestamp and random number are used.
- **Authenticity:** Freshness + Integrity guarantees authenticity.

Critical information

The document's crucial information was identified as solely the audio data.

We encrypt the audio with a disposable key, and this disposable key is encrypted with the long term key (FK). This ensures confidentiality and that we don't overuse the long term key.

Freshness

We use a combination of a random number and timestamp to guarantee freshness for:

- **Unpredictability:** Random numbers can prevent an attacker manipulating the system by predicting the nonce with a counter for example.
- **Only one server:** We don't need to worry about clock synchronization between multiple servers.

Integrity

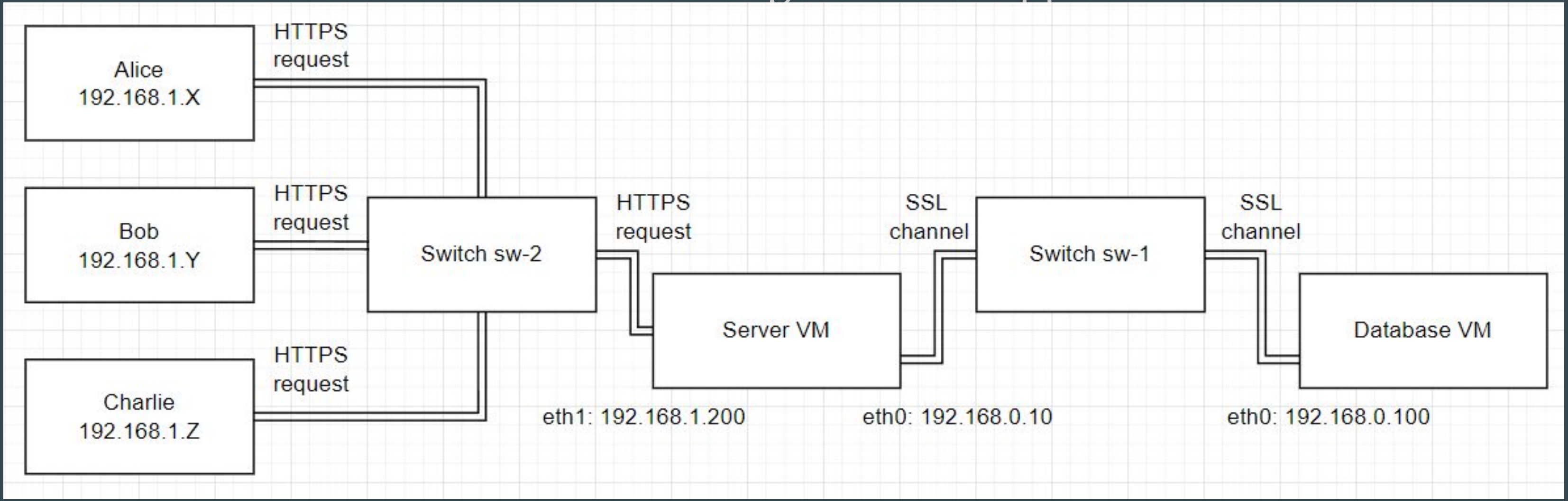
We used a HMAC over DS to ensure integrity:

Non-Repudiation: We didn't need to guarantee non-repudiation.

Key Management: In the beginning we assumed that the user and server already shared a secret key, avoiding key distribution.

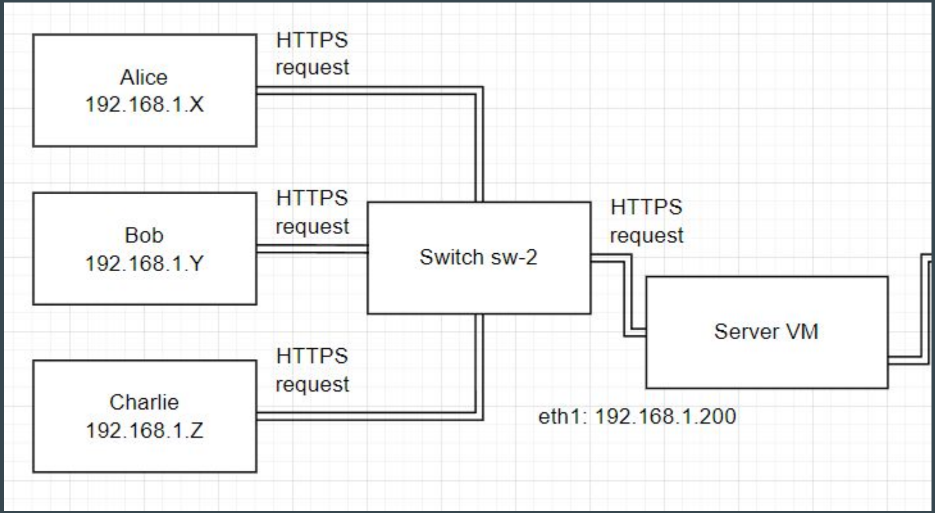
Infrastructure (General)

- 3+ VMs running Kali Linux.
- Server VM: runs a Flask server accepting HTTPS requests.
- Database VM: runs PostgreSQL database.
- Client VM: one or more VMs running our client application.



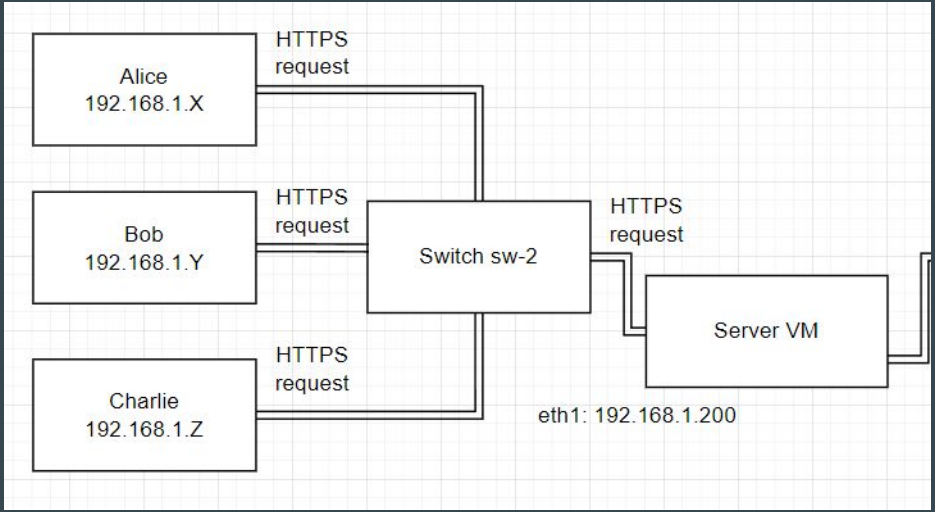
Client

- This VM is mostly left at the default stage, to run a client we only need:
  - A copy of server certificate.
  - The client script.
  - The installed dependencies
  - Exposure to the sw-2 switch that handles communication with the server.
- The client can only communicate with the server via TLS/SSL using its copy of the certificate signed by the server, since the firewall blocks any packets that do not come from port 443



Server

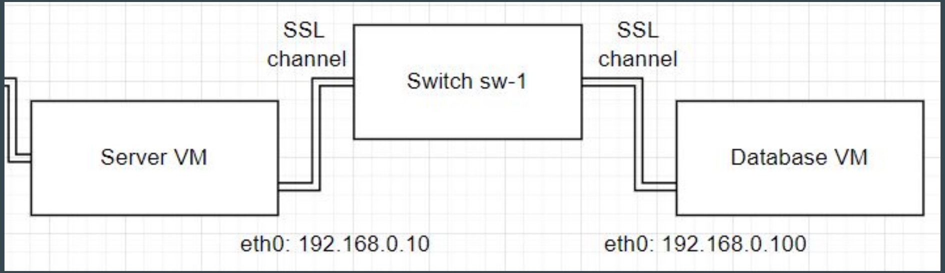
- The server:
  - Is exposed to two switches, one for server-client communication (sw-2) and one for server-database communication (sw-1)
  - The server only listen for requests coming from port 443
  - Denies telnet connections
  - Denies ICMP echo requests



- To handle secure communication with its clients, the server emits a self signed certificate that:
  - Is generated with a 4096 RSA key
  - Is valid for 365 days
  - Is explicitly intended for communication to 192.168.1.200 (server IP exposed to sw-2)
  - Is pre-distributed to every client.
  - The client uses this certificate to make HTTPS requests to the server

Database

- The database:
  - Is exposed to sw-1, to communicate with the server
  - Only listen for requests coming from port 443
  - Only allows connections to the database server coming from the server IP, database IP, or localhost.
  - Denies telnet connections
  - Denies ICMP echo requests



- Database secure channel:
  - The database server uses SSL/TLS to communicate.
  - The database VM generates a self-signed certificate that is used by the postgres server to secure communication.

Security Challenge

Use cryptography options that allow playback to quickly start in the middle of an audio stream, optimizing user experience without compromising security.

Add the concept of family sharing, where individual users can be members of the same family, and a protected song should be accessible to all family members without modification.

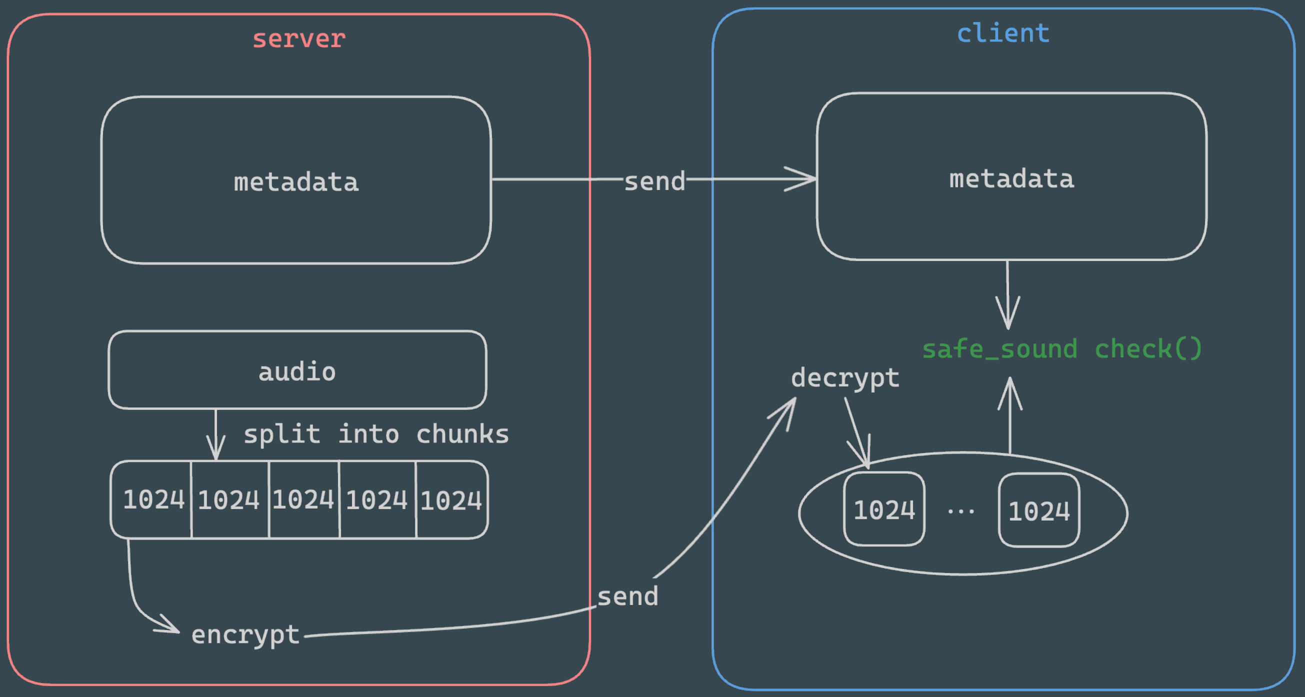
Each user still only has its own key, so, some dynamic key distribution will have to be devised.

Security Challenge

Streaming: despite our efforts to make streaming work, we ended up not accomplishing this one goal as intended.

The idea we tried to implement was changing the cipher mode from CFB to CTR and streaming the song data in 1024 byte chunks. The JSON with the metadata would be sent in one chunk and then the encrypted audio content would be streamed, with each chunk being decrypted by the client on arrival. After the audio content was fully received and decrypted, it would be added to the JSON and we would use our library to check its integrity.

Streaming


server
metadata
send
audio
split into chunks
1024
1024
1024
1024
1024
encrypt
send
client
metadata
safe_sound check()
decrypt
1024
...
1024

Security Challenge

Family Sharing: with the requirement to add family sharing to our application, some things had to change. Firstly, we introduced a new table to the database, representing a Family.

At first, we had a single key which the system’s intervenients already knew beforehand. With the introduction of families, there was the need for users belonging to the same family to be able to decrypt the audio content of a song which was bought by a family member, so we needed to design a new key system.

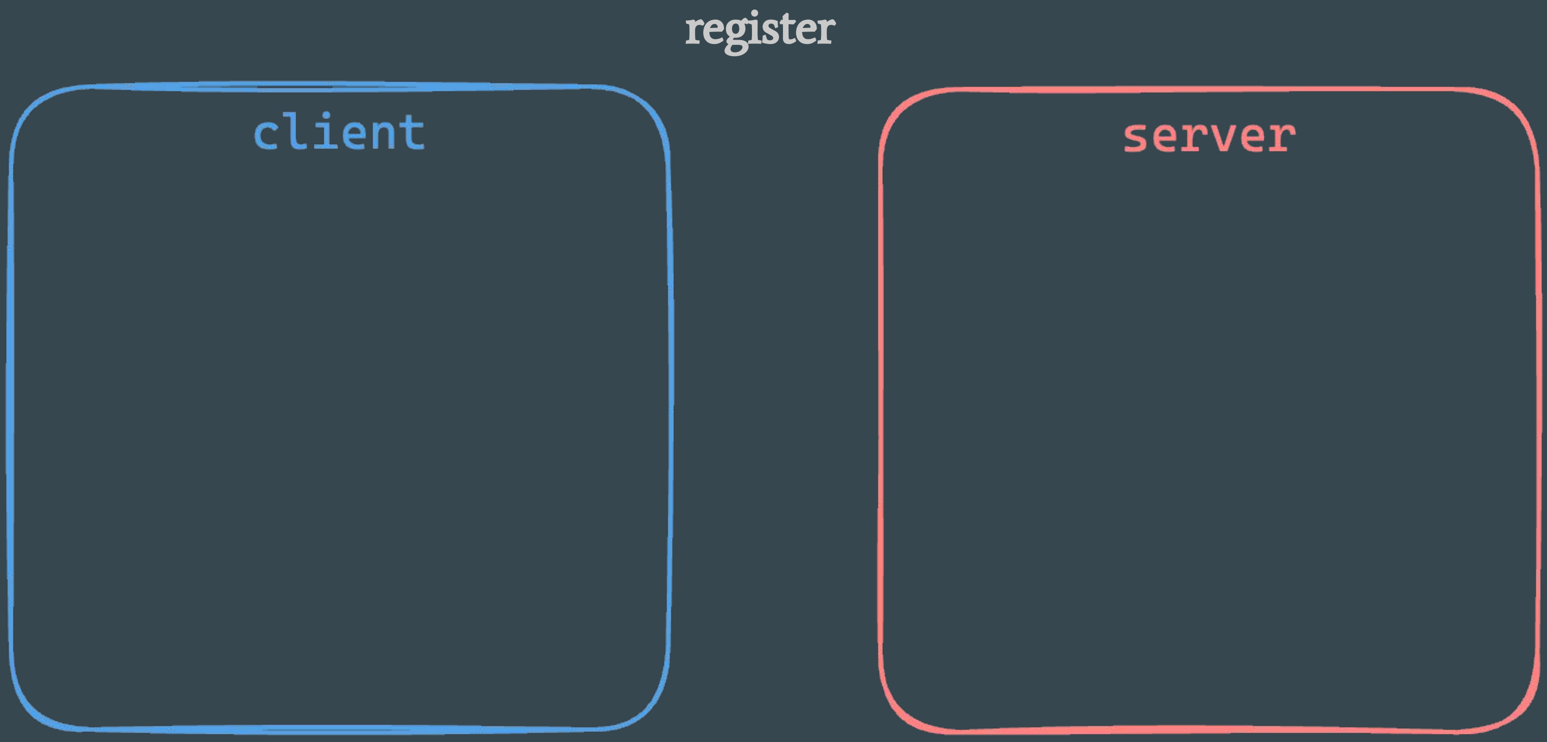
Family Sharing

We added the concept of **Family Key(FK)**: a key unique to each family used to secure media a family can access.

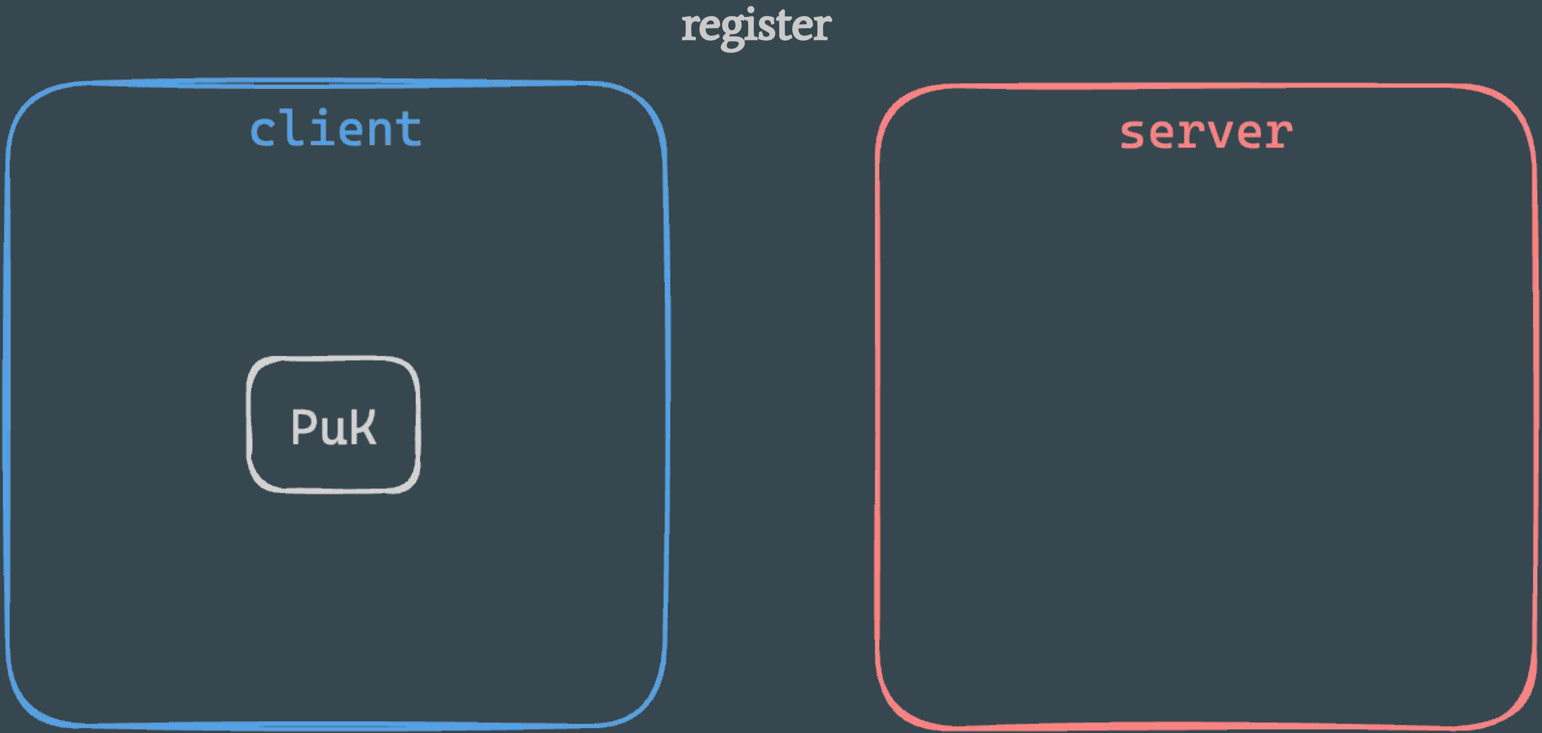
Assume the previously known keys are FKs (at the beginning, each family is comprised of a single user).

On register, a client generate a RSA public-private key pair, and sends its Public Key (PuK) to the server, encrypted with its FK. The server decrypts with the FK and stores the user's PuK.

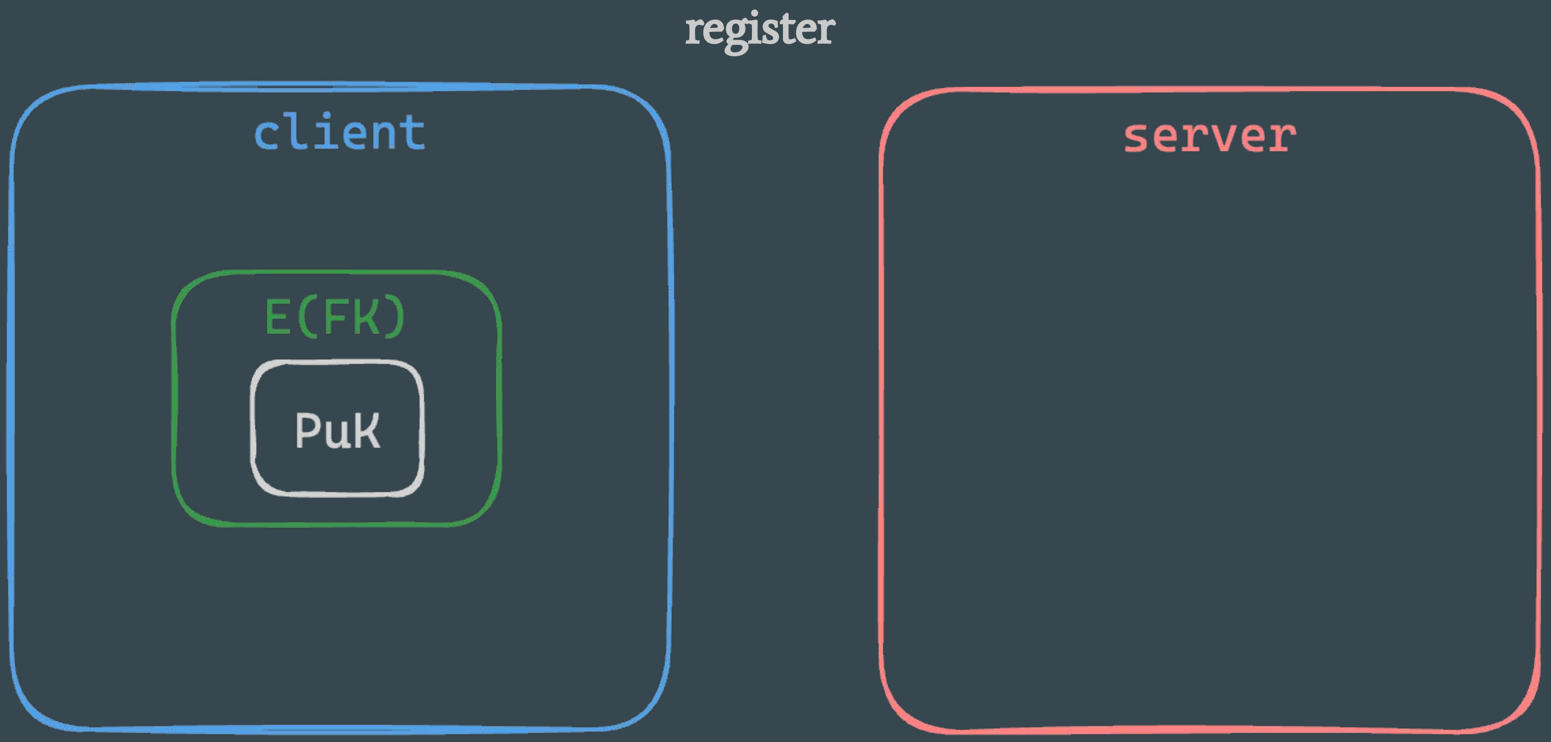
Key Distribution



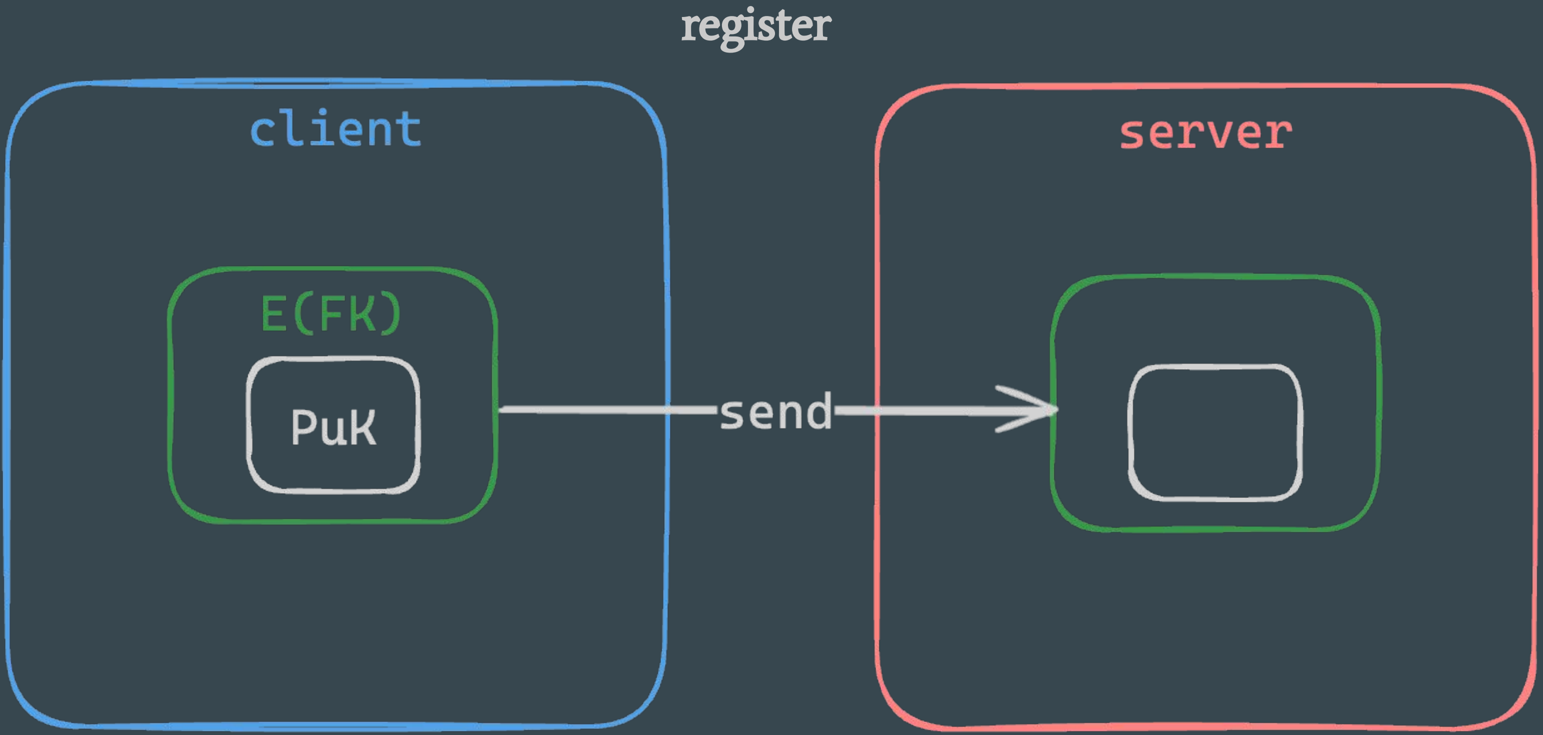
Key Distribution



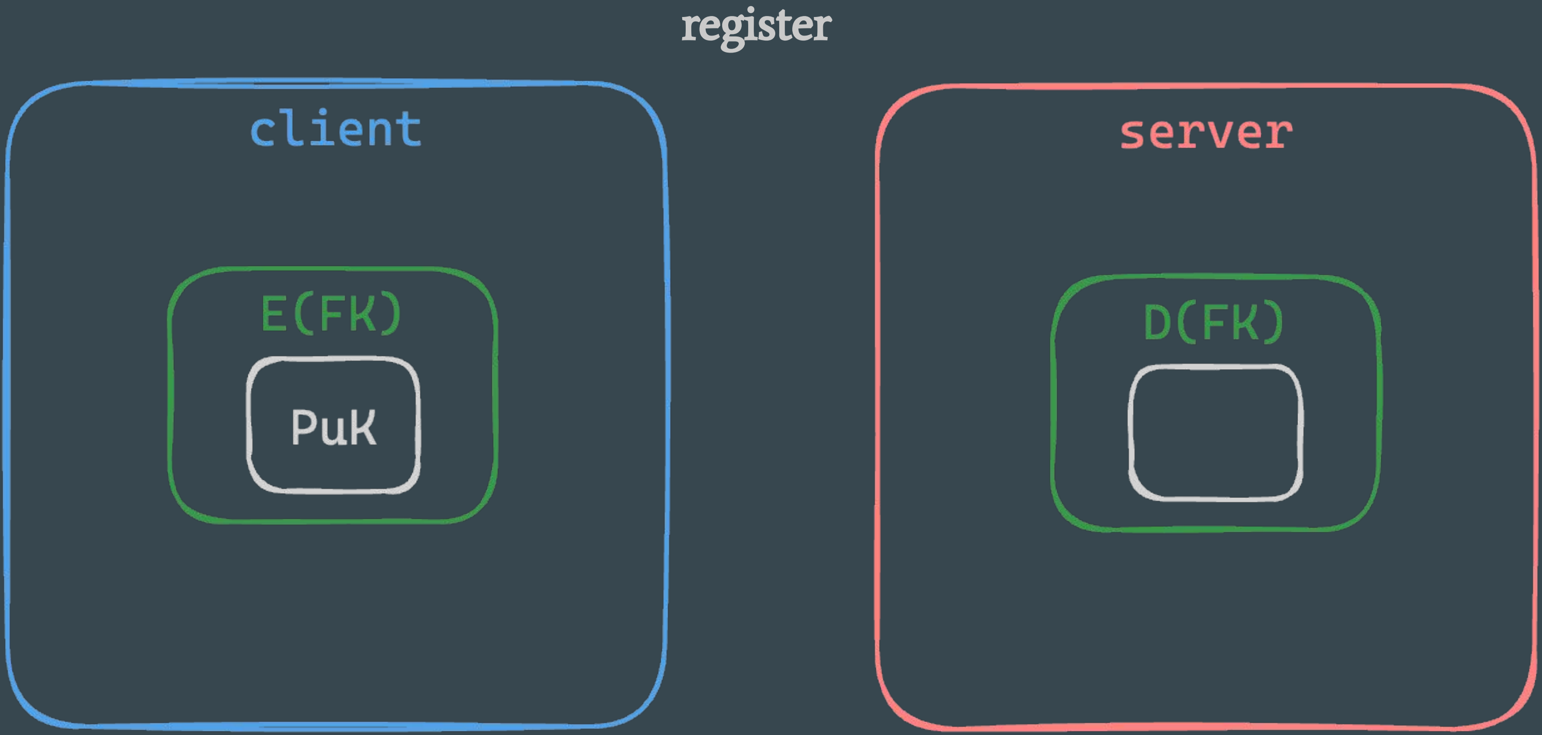
Key Distribution



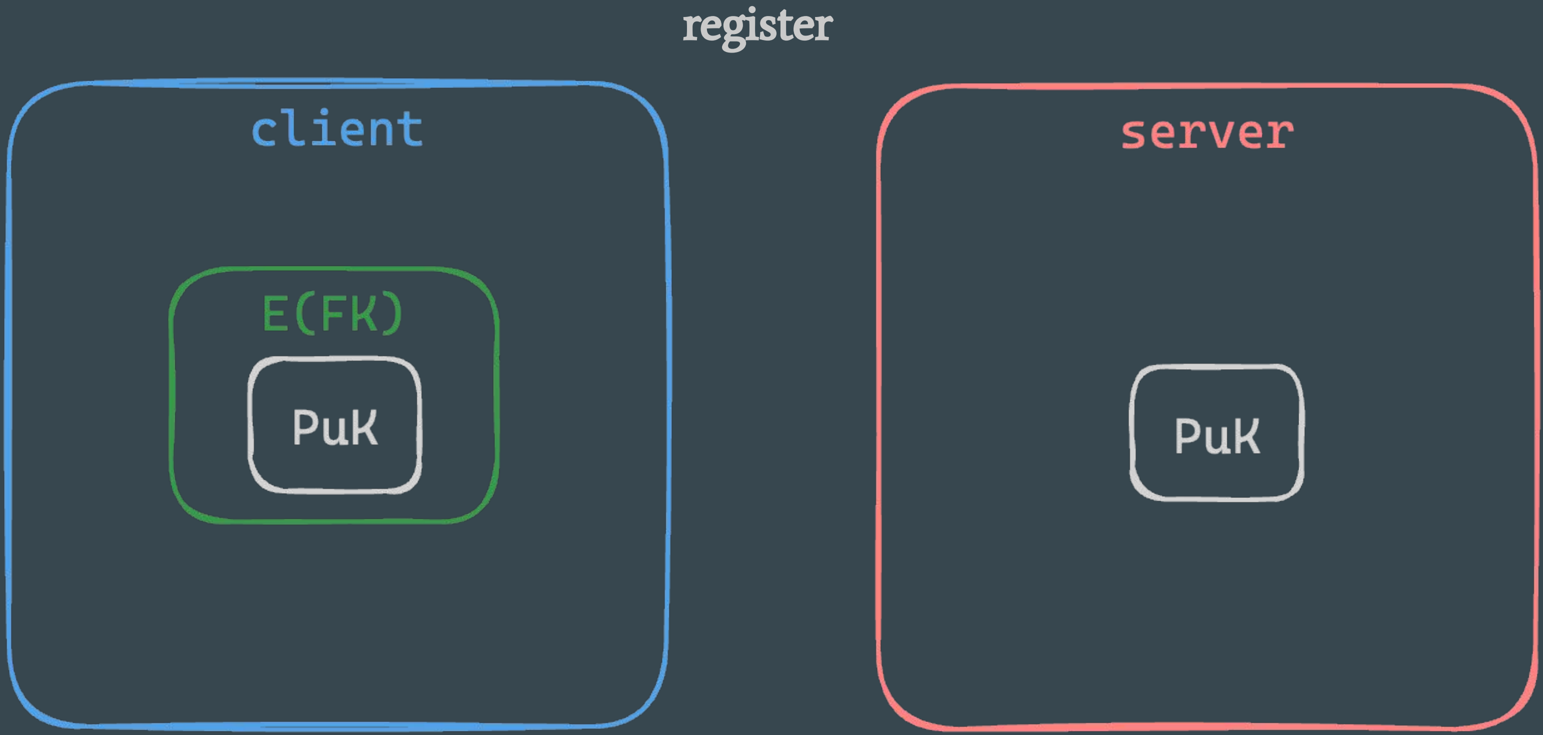
Key Distribution



Key Distribution



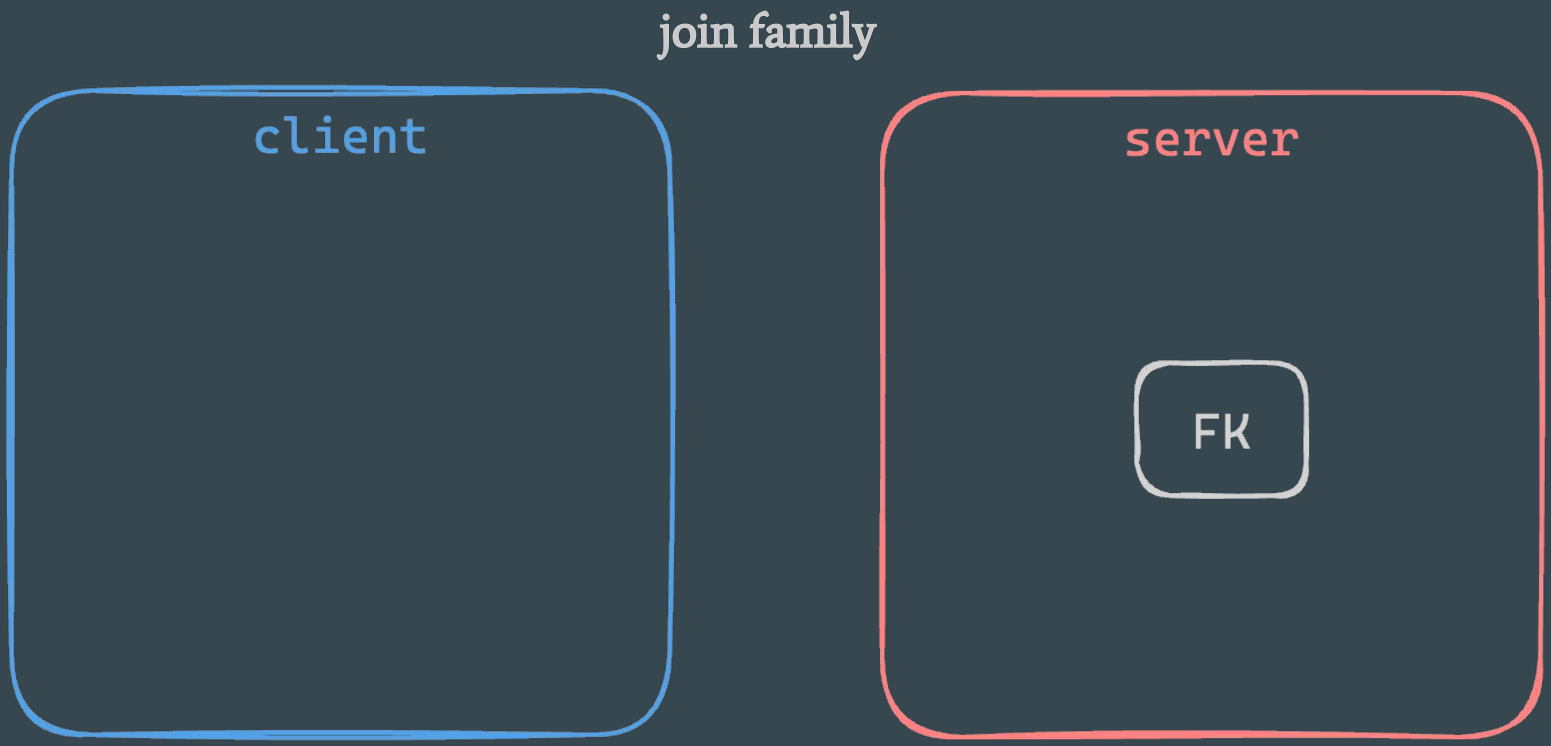
Key Distribution



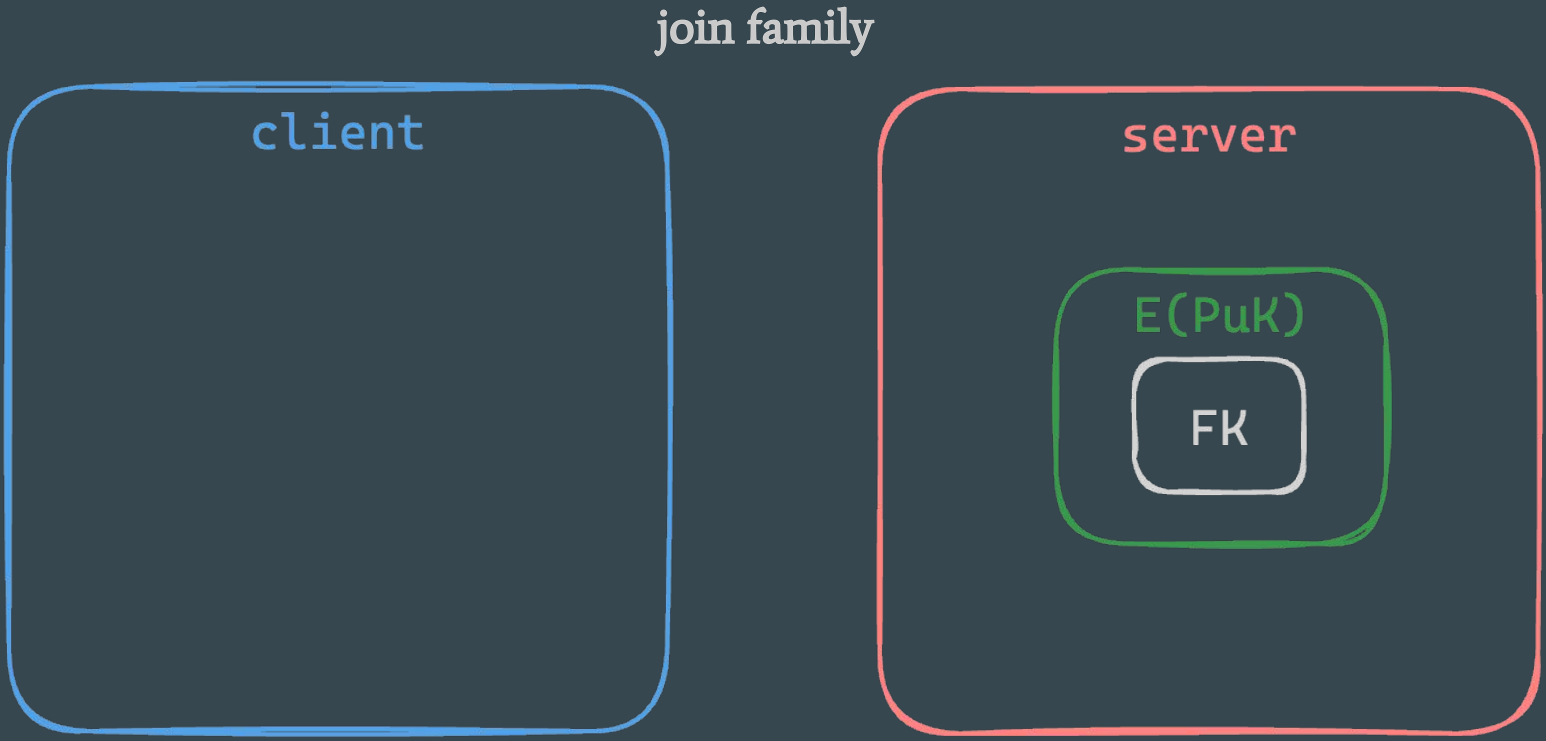
Key Distribution

When a user joins a family, the server sends that family’s FK encrypted with the user’s PuK. In the client side, the message is decrypted with the user’s Private Key (PvK) and the new FK is stored.

Key Distribution

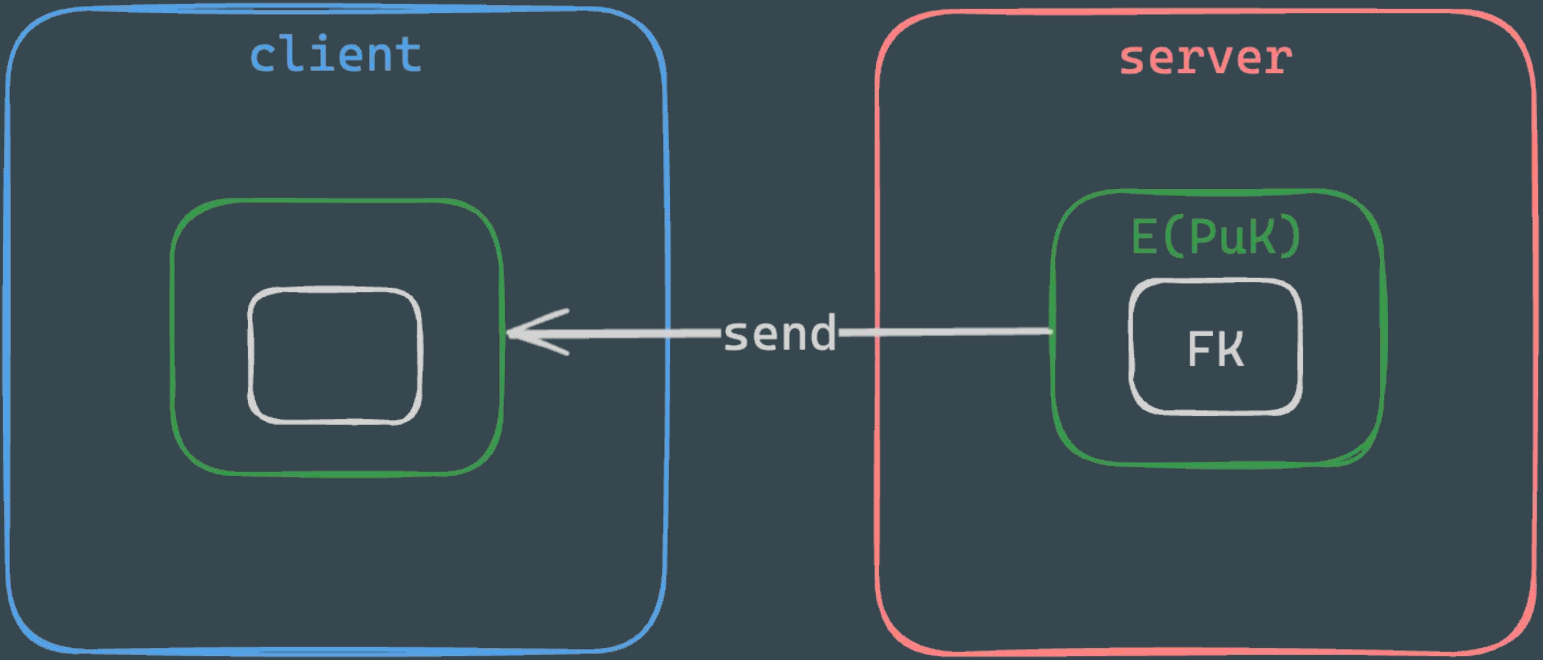


Key Distribution

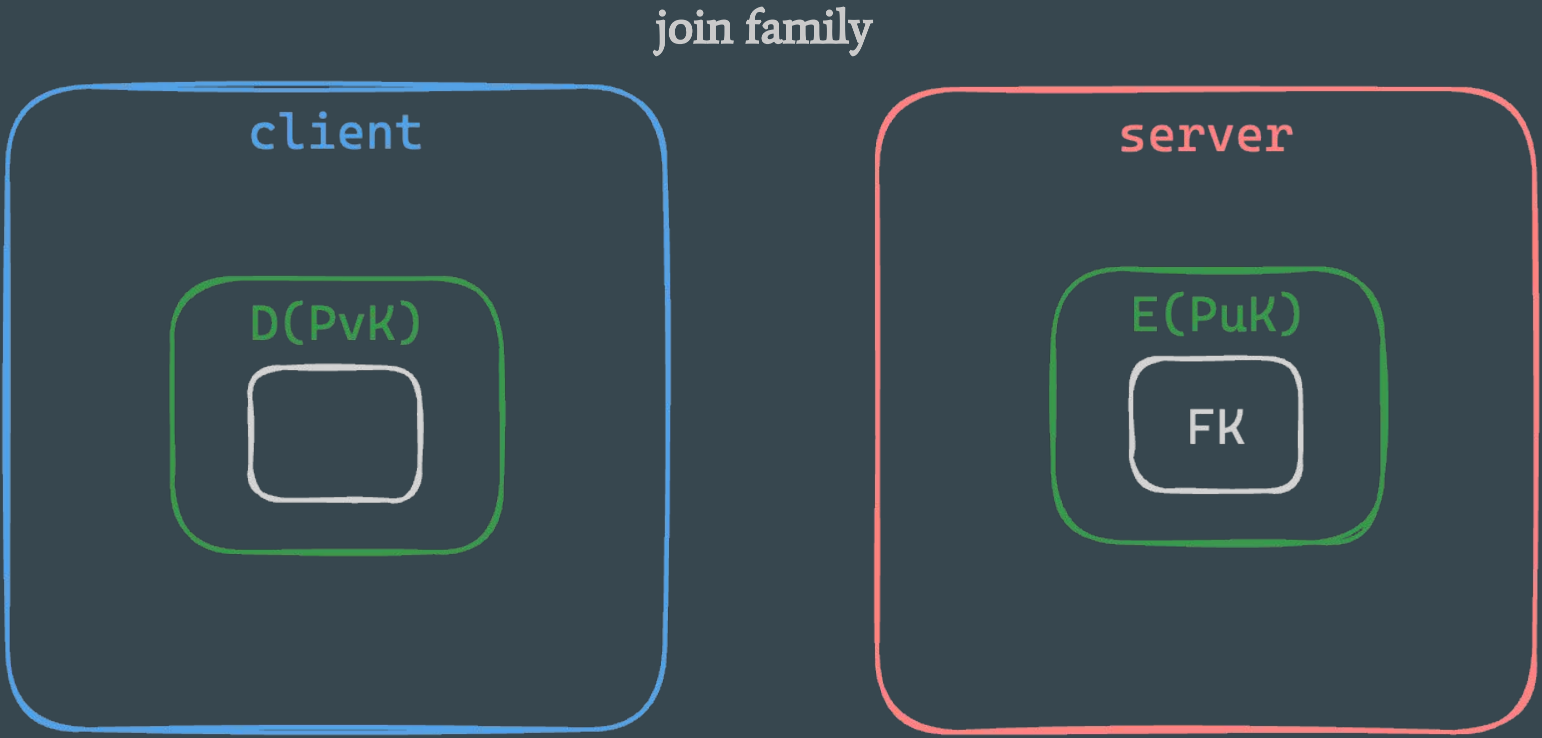


Key Distribution

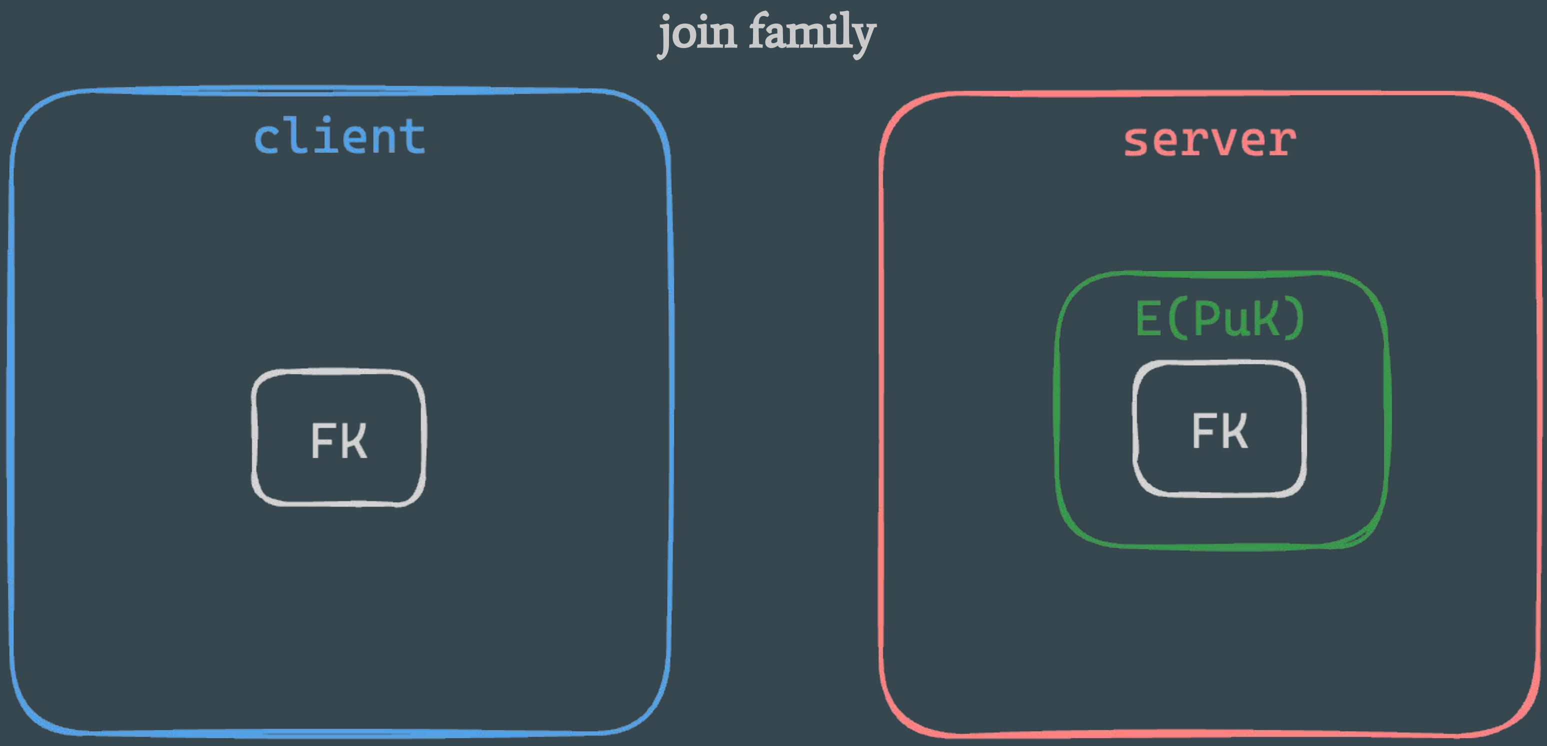
join family



Key Distribution



Key Distribution

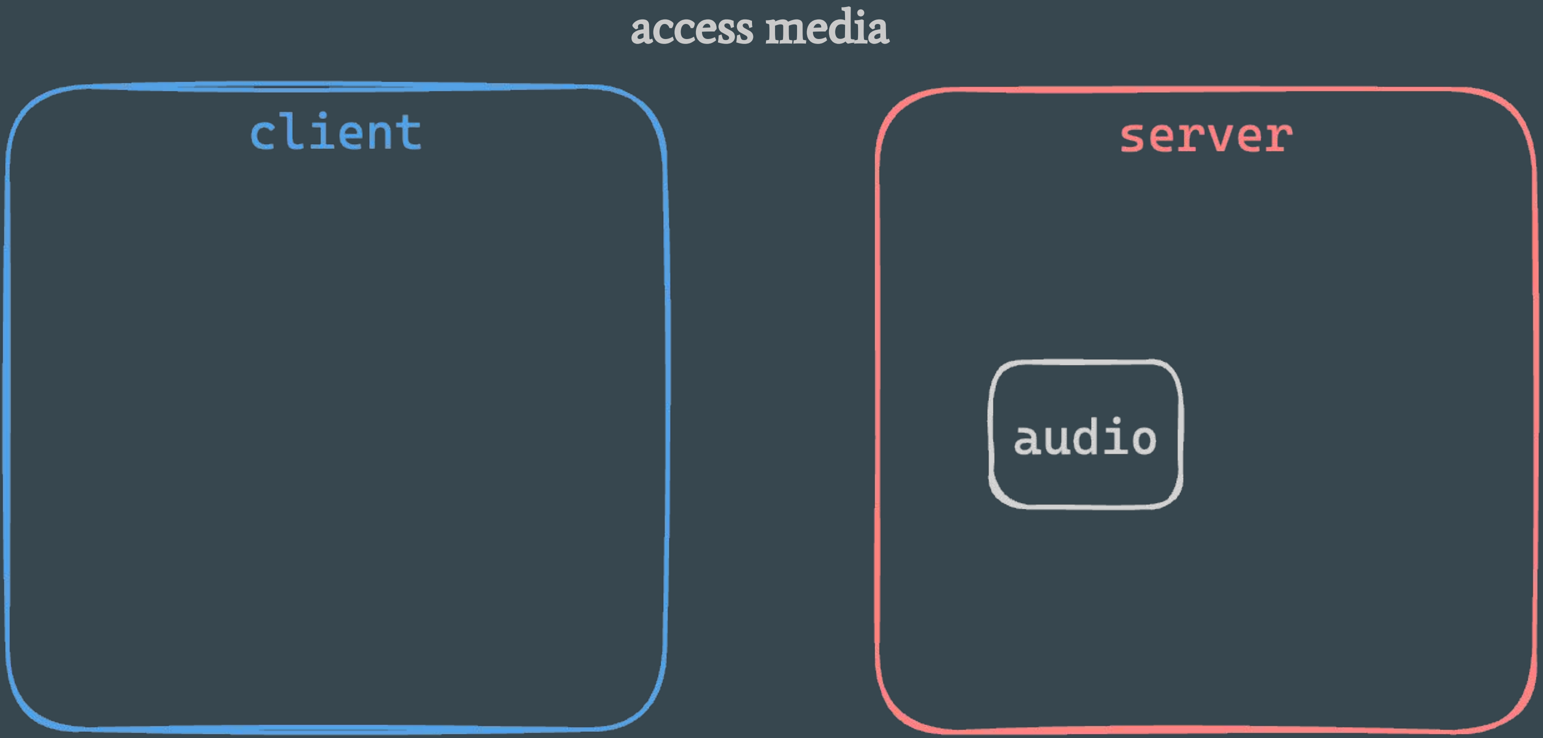


Family Sharing

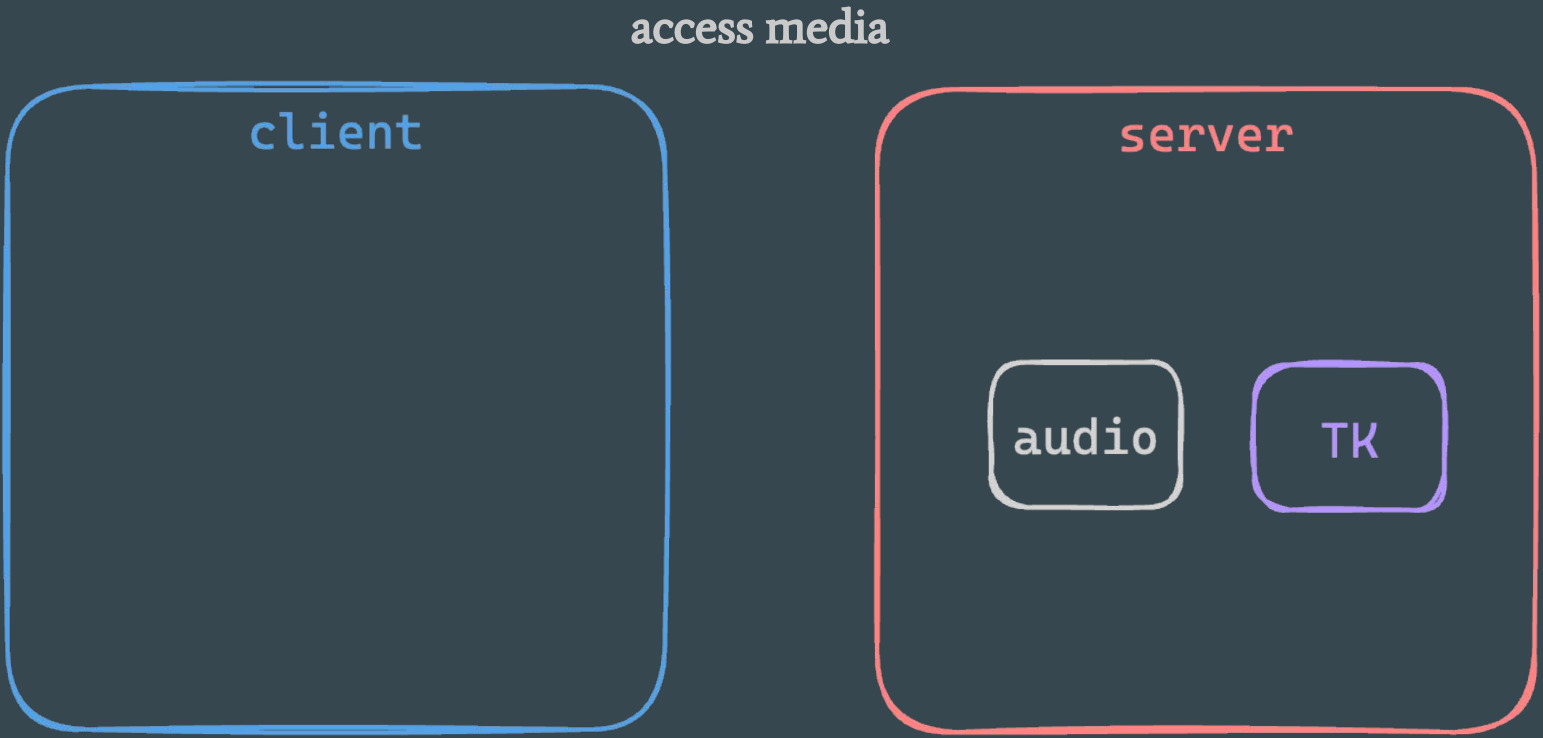
When sending audio content, the server generates a Throwaway Key (TK) and encrypts the song data with it. Then the server bundles the encrypted audio content together with the TK itself and encrypts this bundle with FK. By encrypting with a single-use key, we make the system more resilient to cryptanalysis when compared with encrypting with the FK alone.

In the client side, the received message is decrypted with the FK, obtaining the TK. Then the encrypted audio is decrypted with the TK, obtaining the desired data.

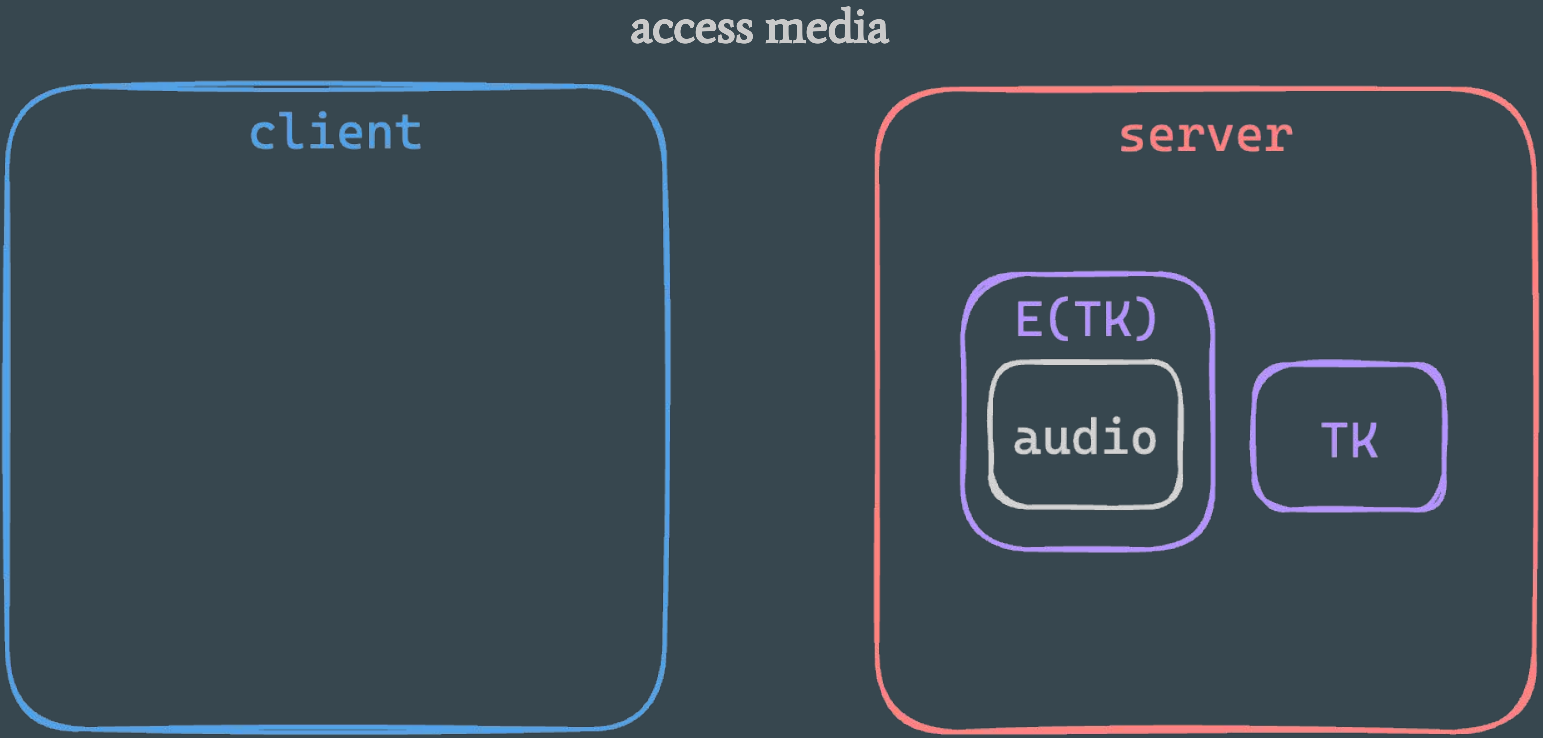
Family Sharing



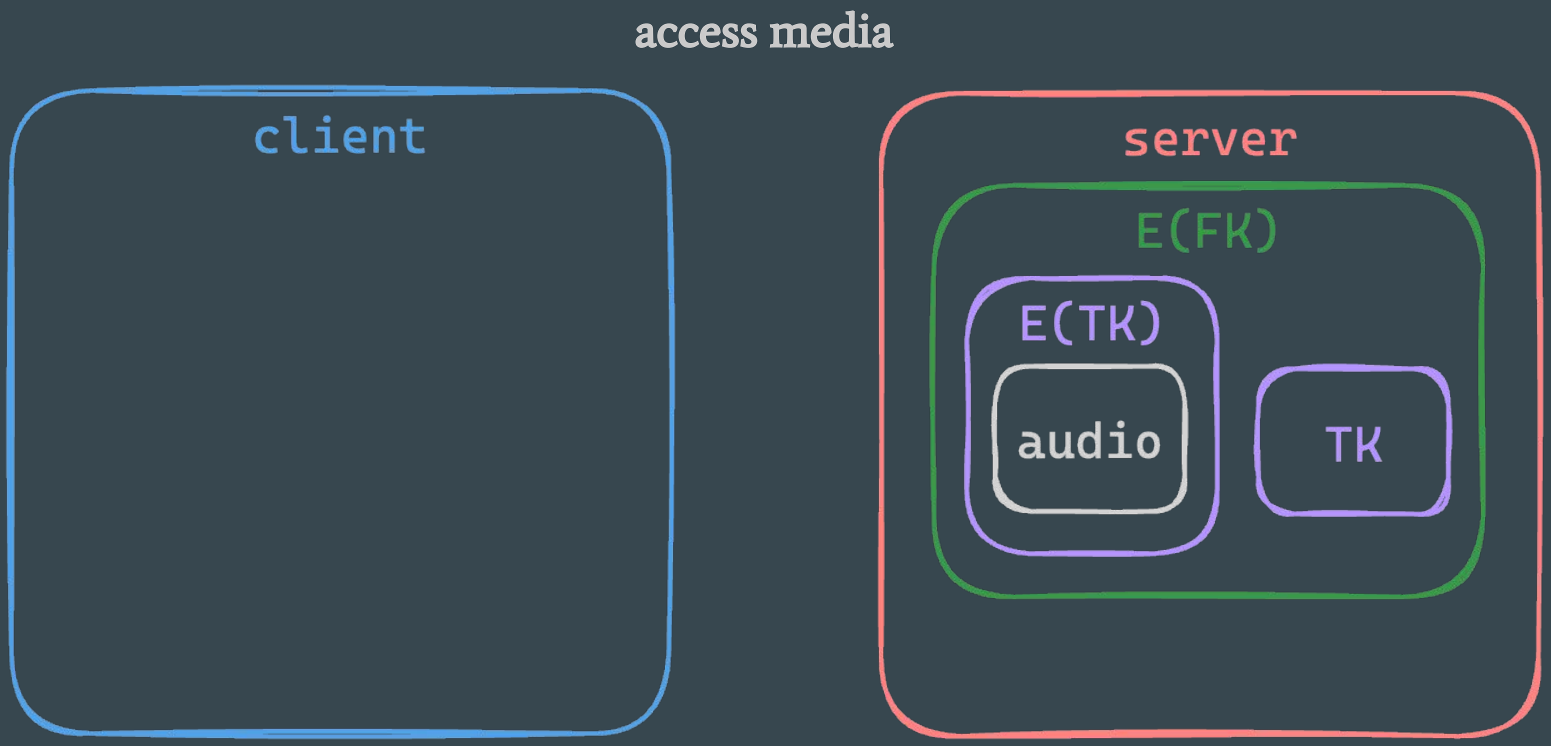
Family Sharing



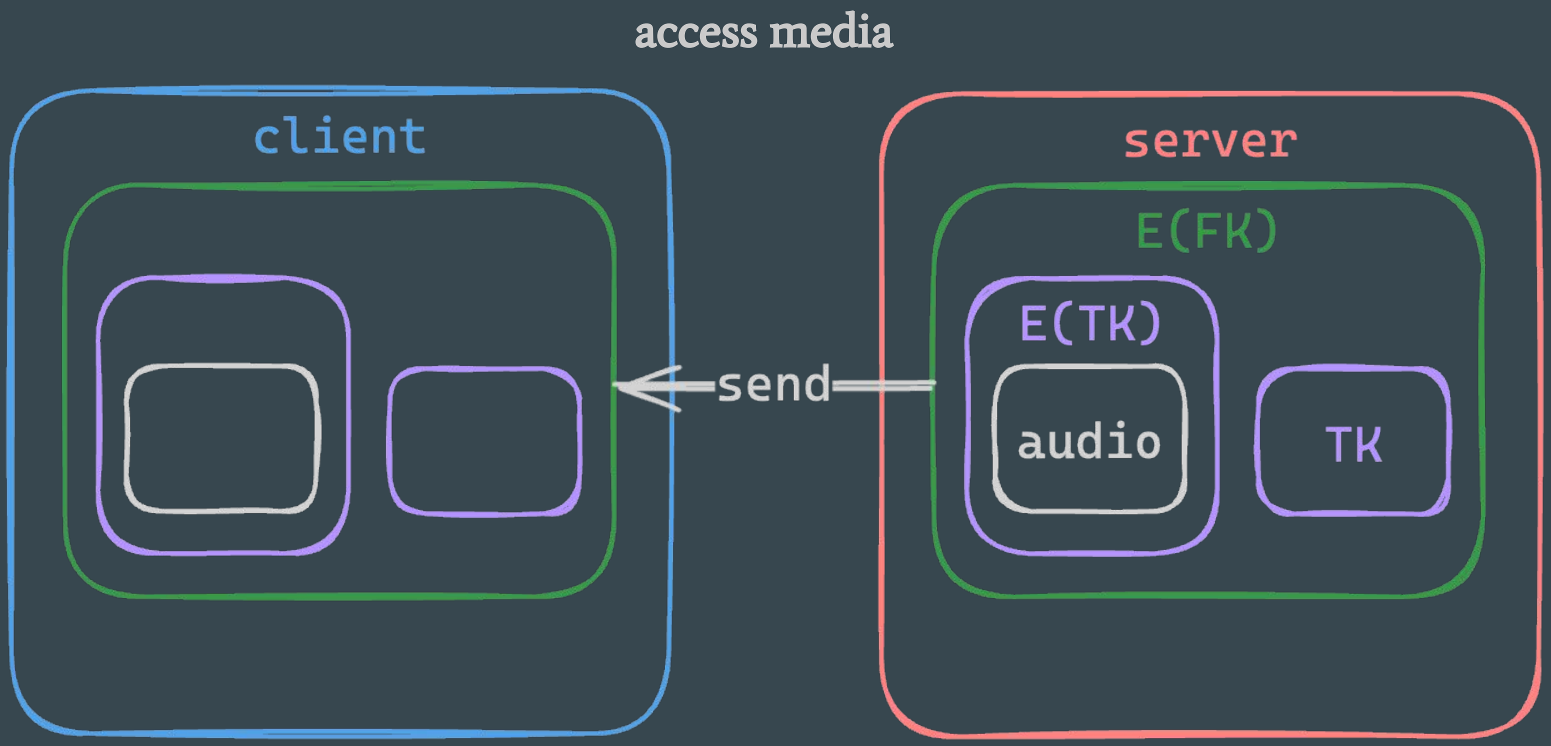
Family Sharing



Family Sharing

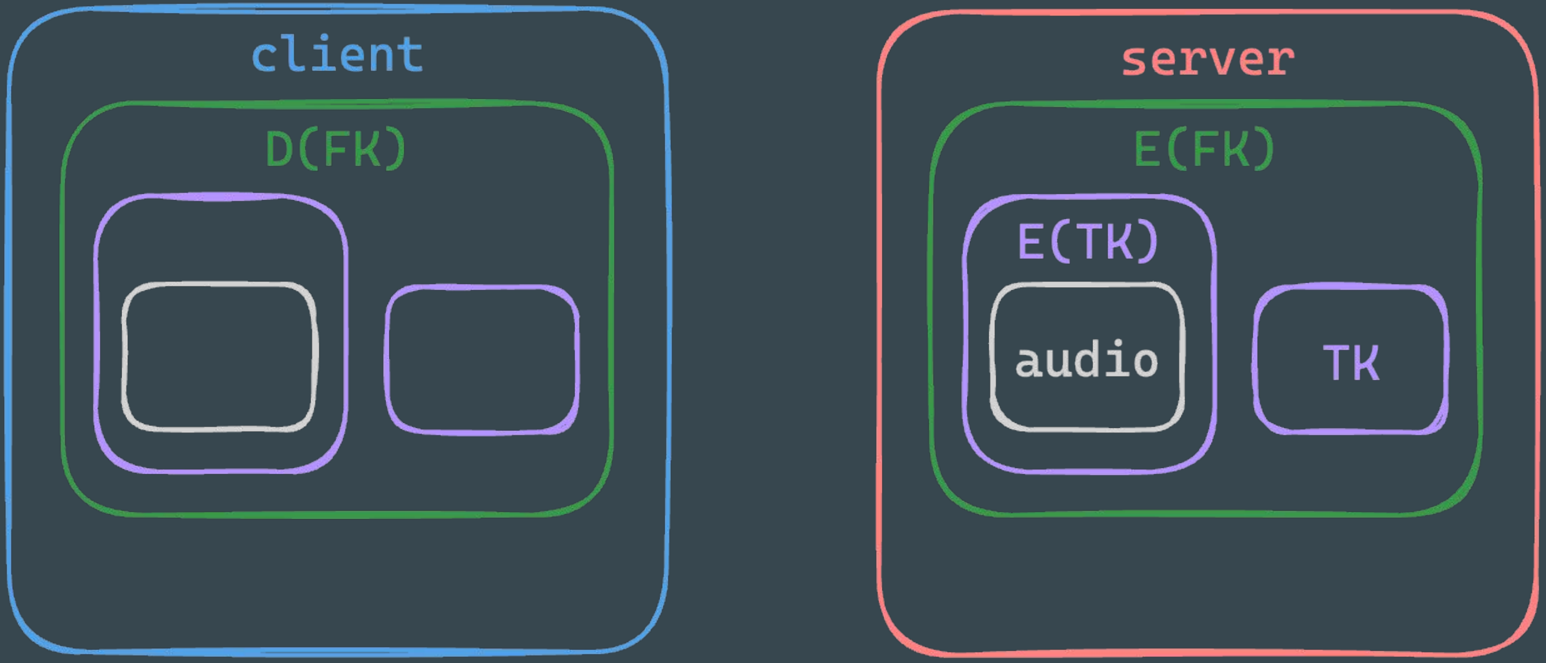


Family Sharing



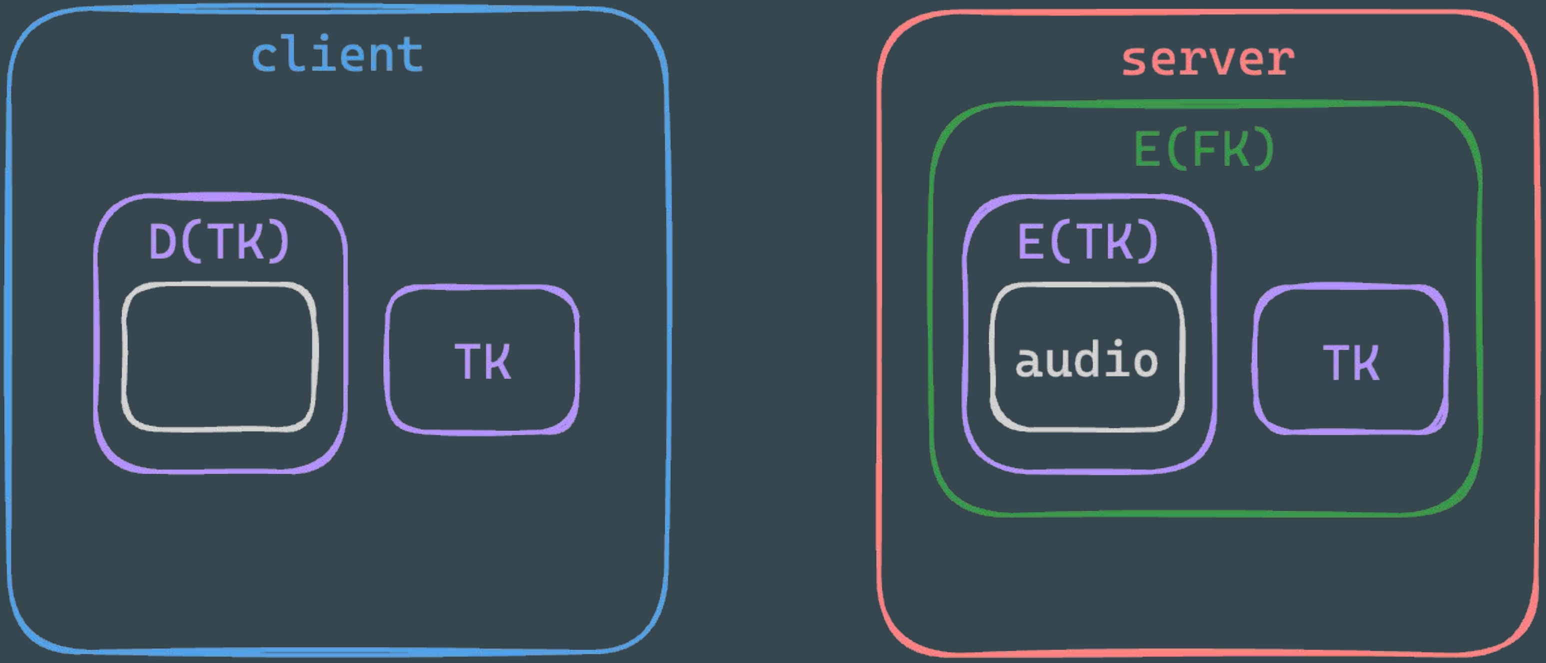
Family Sharing

access media

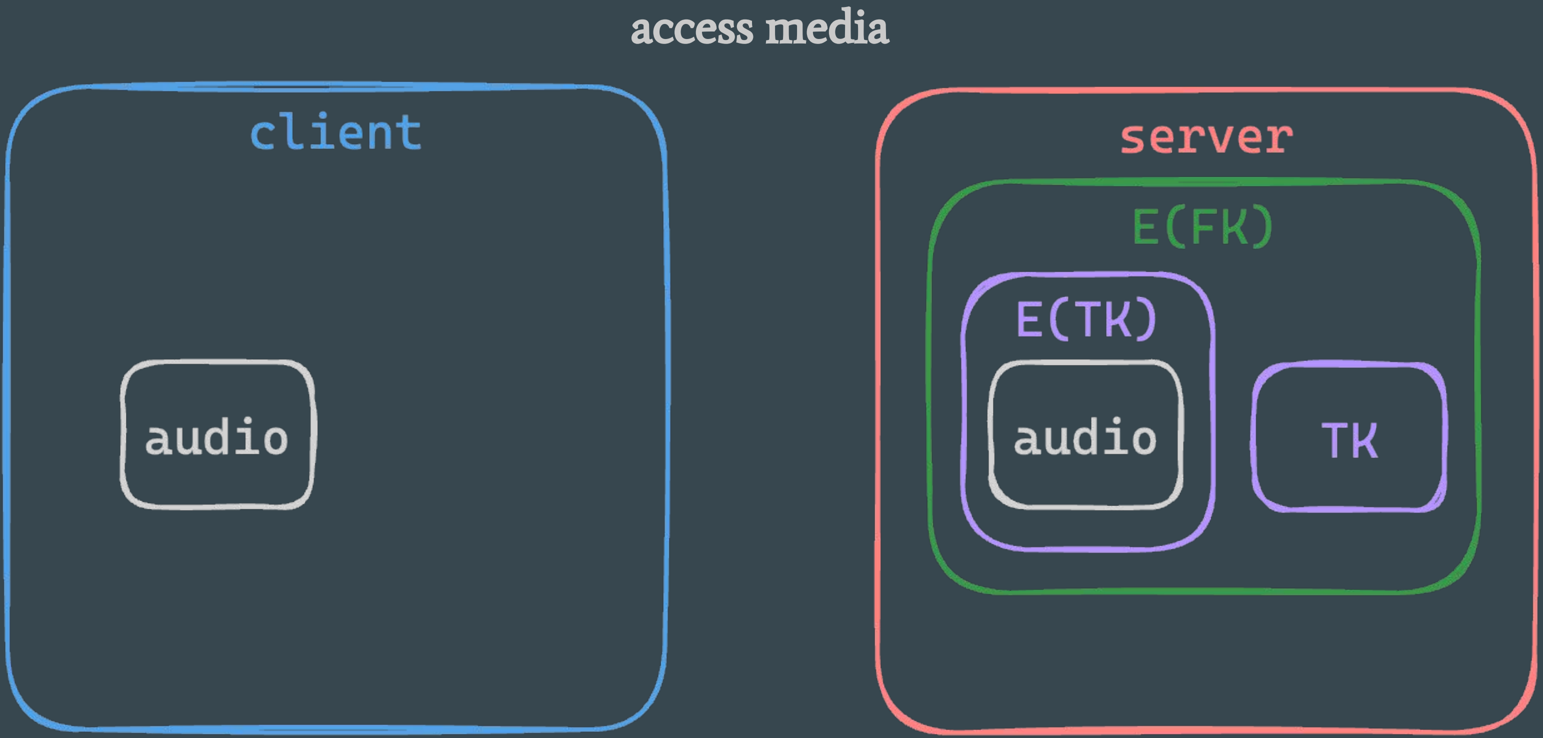


Family Sharing

access media



Family Sharing



Results and Conclusion

- Successfully developed a robust security application by employing safe key distribution and developing a secure documents library while using a safe infrastructure.
- Simulated man-in-the-middle attacks, ICMP floods, and port scans. The application demonstrated resilience against these simulated attacks.
- **Future Improvements:** audio streaming

Questions and Answers