Aalto University
School of Science

ELEC-E7320 - Internet Protocols D

Collaboration tool - Group G

Aarni Haapaniemi aarni.haapaniemi@aalto.fi
Zong-Rong Yang zong-rong.yang@aalto.fi
Prerna Gupta prerna.gupta@aalto.fi

Step 1: Discuss the use cases with the team.

The use cases define how users would interact with the app. You can use the following template to describe use cases. You should be able to clearly define entities and their roles in the system after this step.

Use case: Collaboration Tool

Actors involved:

1.Client–administrators

2.Client–collaborators

3.Server

Preconditions:

1.The user can create a document, who will become the administrator for that document.

2.The administrator will get a token for the the document, it can be shared to other users to access the document so that they can have permission to edit contents.

3.Only administrator can delete the document.

4.Anyone without the document's token is unable to read or edit the file.

5.The service quality ensured for up to 10 concurrent collaborators, the collaboration tool doesn't limit the amount of users in the same document.

Steps:

1.Client A creates the documents on the collaboration tool and becomes the administrator. Then client A can access the options for the document. A random token will be generated for every creation of a new document, which can be further distributed to others(client B, C…).

2.If client B wants to connect to the documents and get permission to read and edit the files, he/she can ask client A(administrator) to modify the document's collaboration option.

3.When client B gets the token from client A(administrator), client B can enter the file and have permission to edit it.

4.Another collaborator (client C, D, E…) can ask for a token to get permission to read and edit the documents.

5.Only client A (administrator) has the rights to delete the document.

Error:

1.If collaborators (Client A, Client B, Client C…) lose internet connectivity during editing, their changes may not be successfully synchronized. The collaboration tool should provide a warning message and try to reconnect to the document page.

2.If the token is not correctly generated when creating the document because of internet errors, the collaboration tool should inform the administrator(client A) to re-generate the document again in order to get the right token.

Post-Conditions:

1.The collaboratively edited document is saved and accessible to all collaborators. Any changes made during the collaboration session are reflected in the document, and the document status is updated on the server.

2.If there are more than 10 collaborators online at the same time, the server can send a pop-up message to each participant that may affect the performance of the collaboration tool service, therefore it's better to decrease the total number of users.

Use cases:

- editing the document
- authenticating the document owner
- deleting / creating the document
- something else?

```
Use case name:
Actors involved: (e.g. client A, client B, server and etc.)
  Preconditions:
  Steps:
  The client xxxx
  xxx
  xxx
Error:
  e.g. If xxx does not xxx, xxx will be xxx
Post-conditions:
  e.g. The client displays xxx
```

Use case name:

Modifying a document

Actors involved:

Client, server

Preconditions:

Active connection between server and client

Client has the token to access the document

Steps:

The client A modify the content in the document, and for that client it triggers local updated response on the document.

Server receive the modifying request from client and validate the identify of the client.

Server transmit the modifying content to all other clients online.

Ohter clients(client B,C,D…) receive the modifying content from server.

The changes will show on other client's document page. Error:

Server ignores the modifying action.

Client's token cannot be varified.

Online client losts the connection with Server. Post-conditions: All clients now have the same contents on the document page.

delete doc

Use case name:

Deleting a document

Actors involved:

administrator and server

Preconditions:

1. Active connection between server and administrator
2. Client has the token to access the document

Steps:

1. Inform collaborators,It is crucial to communicate the removal of a document to ensure that others are aware of the changes.
2. Only Admin will be able to delete the document.
3. Admin will instruct the server to delete and then server will validate and confirm that the request is from authorized admin or not.
4. Server will take action accordingly once it has verified the information provided above.
5. The changes will show on main tool page.

Error:

1. Server ignores the deleting action.
2. Administration losts the connection with Server while deleting the document.

Post-conditions:

1. All clients will be able to notice that the document is no longer there after logging into the tool.
2. The document will be permanently deleted, recovery might be impossible.

Example use case: creating a document

```
Use case name:
    Creating a document
Actors involved:
    Client, server
Preconditions:
```

```
    Active connection between server and client
Steps:
    The client generates a new public-key encryption keypair (sk, pk)
    Client sends a "Create doc" request to the server + their public key pk
    Server responds by generating a random challenge using pk
    Client decrypts the challenge with their secret key sk
    Server checks if the decryption looks ok
Error:
    Decrypted challenge does not match:
        Server ignores "Create" request
    Something else???
Post-conditions:
    A document is created by the server
    The client now has access to the document via a token
    The client is now called the "Admin" of the document because they own sk
```

Step 2: Go through the use cases and summarize the functional requirements.

Note that use cases do not typically describe non-functional requirements and constraints. You can identify nonfunctional requirements by analysing the operational environment (e.g. network environment, potentially large number of users, etc.), the expectations from user experience perspective (e.g. low latency, high resolution), security (e.g. data encryption, join approval), and any other relevant concerns.

- Number of users: service quality ensured for up to 10 concurrent collaborators, but more can technically join
- encryption: WSS/TLS
- no join approval other than a token
- other things?

Encryption:

Public-key encryption makes use of two keys: a secret key known only to the administrator and a public key shared with the server. To access the document, both admins and non-admins utilize a token. Non-admins are currently authenticated only through the token.

Functional Perspective:

The system should have a feature that allows administrator(Client A) to invite teammates(Client B, Client C…) by entering their usernames or email addresses.

Modifying:

The system should provide real-time updated contents for clients, and avoid conflicts during the process.

Deleting:

The system must inform collaboratos about the deletion of document because only admin has the authority to initiate the process of document deletion.

Step 3: Architecture design.

You could consider, for example, whether you apply a layered architecture for your system and protocol design, and whether you want to follow a distributed or centralized architecture in your system. You should consider the service provided by the protocol (or protocols) you will use in your system, and assumptions about the environment your protocols are being executed.

Introduction

Notation

- The document owner (Admin) is the person who created the document. Only it can delete the document, and initially only it knows the token associated with said document.
  - Should there be a feature for editing the token?
- Collaborators, who can view and edit the document. Anyone who has access to the token is considered a Collaborator, and they may freely choose to share the token.
- The Server/database, who authenticates the Admin and Collaborators, stores the document, etc… Once the server authenticates an Admin, it serves them a token associated with the document. Collaborators are authenticated using the token that was shared with them.

Protocol Details:

Our system employs Websocket Secure (WSS) as a protocol to provide a secure communication channel. We want the system to have some notion of security, and thus opted for authentication for the Admin, achieved through Public Key Encryption (PKE). Once the Admin is authenticated, it receives a token that it can share to collaborators in order to edit the document.

Service Description:

The tool delivers real-time editing capabilities, allowing multiple users to collaboratively edit documents. Anyone can create a new document, and anyone can join a document as long as they know the token assigned to it. Documents are stored securely in a centralised database, meaning that they can be accesses after a session has ended. The Admin of a document can choose to remove a document, which triggers immediate deletion from the database and closure of the session for all collaborators.

- Documents could be removed if they are not accessed after some time period? perhaps out of scope.

Security Measures:

Our tool prioritizes security with Websocket Secure, PKE and token-based authentication, ensuring confidential admin access and secure collaboration. This approach aligns with the industry's best practices for safeguarding sensitive data, but could potentially be negative in terms of UX if the Admin loses access to their skecret key.

Environment Assumptions:

Although we use WSS, our authentication method for Admin do not rely on there being an encrypted connection between the Admin and the Server, as it uses PKE for authentication. However, we do not assume that each client is in a trusted network, and as such we want to prevent un-authorised parties from reading sensitive data such as the tokens or the data sent to the document, which we achieve by using WSS.

Furthermore, we assume that the latency in the network & server computation is low enough to allow for true real-time editing.

Architecture overview

Application Layer:
WebSocket Secure (WSS): At the top of the stack is the WebSocket Secure protocol, which we use for communication between the client and server.

Transport Layer:
WSS is sent over a TLS session, and thus we will use this in the transport layer.

Network Layer:
Internet Protocol (IP): The network layer involves IP, responsible for routing data packets between the client and server. IP ensures the delivery of encrypted WebSocket data packets over the internet.

Other layers? TCP, Link layer, Physical?

Sequence diagram

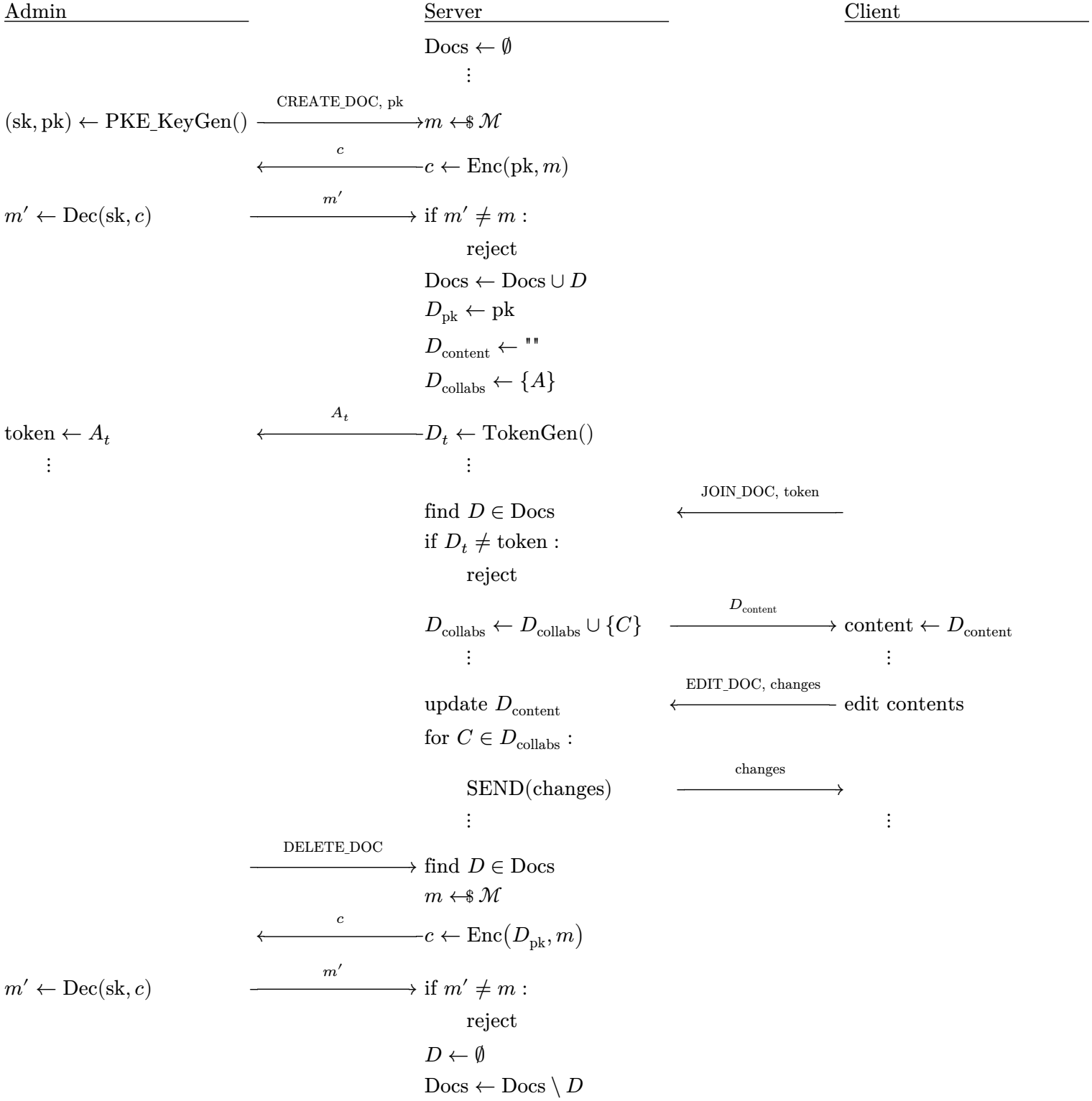
Note: I think the sequence diagram below is more relevant to step 4? :D oh well

Notation used in the sequence diagram:

Authentication works by using public key encryption, which comes with PKE_KeyGen, Dec and Enc oracles. PKE_KeyGen returns a secret and public key (sk, pk), which have the property of $\text{Dec}(\text{sk}, \text{Enc}(\text{pk}, \text{msg})) = \text{msg}$. An arrow ($\leftarrow$) refers to variable assignment, whereas an equal sign ($=$) refers to an equality check. An arrow with a dollar sign ($\leftarrow\!\!\$$) refers to a randomised assignment from a set of multiple possible values. The rest of the notation ($\emptyset, \in, \cup, \setminus$) is standard set theory notation.

Discussion question:

Does public key encryption make sense for this application, or should we use passwords instead? For PKE implementation, see e.g. Web Crypto API



Checkpoint 1 (Friday, 16.2.2024):

Submit a summary of functional/non-functional requirements, a draft of the architecture design, and a brief description about the next steps. Submit a presentation (PowerPoint is sufficient) by 16.2.2024 and present it in a peer review session on Tuesday, 20.2.2024. Feedback will be given to each group.

Step 4:

At this step, you start to define messages to be exchanged between entities. One way is to draw a message sequence diagram for each use case and define the message format used in the messages. It is also recommended to draw finite state machines for each entity. For example, if a protocol is about sending a file from A to B, then you can draw one state machine for the sender and another one for the receiver. Protocols must be prepared to deal appropriately with every feasible action and with every possible sequence of actions under all possible conditions. When working on your design, analyse how the related protocols work in other similar systems (as referenced in topic descriptions), and try to think if you can make a better design in some respects in your solution. You can choose to first make your own designs, and then compare with previous ones. Alternatively, you can first study the existing solutions, identify their limitations, and try to design a better one. Note that it is not acceptable if you simply copy the design of any existing protocol/software.

Things we need to do

- Database (backend)
- encryption (web crypto API & WSS) (front & backend)
- edit capabilities (frontend & minor backend)
- efficient broadcast of changes (mostly backend)
- file upload (frontend)
- measure Round Trip Time, aka delay between client and server (RTT) (backend)

Distributing tasks:

- Aarni
  - Encryption
  - Change broadcasting
- Zong-Rong
  - Database
  - File Upload
- Prerna
  - Edit capabilites
  - RTT

Database:

- JSON file for each document
  - What the document contains
  - who is the admin
  - token

Each time a user finishes doing something on the frontend, send these changes to the backend and the server will process (save the new document) the request and tell all the clients what the document now looks like

Step 5: Implement and test the protocols.

This can be done without actual user interface or developing a very simple user interface (e.g. command line). You should check if the state machines have been implemented correctly, and test the performance, efficiency and scalability (e.g. latency and traffic size when the number of clients increase).

Step 6: Implement the application logic using the designed protocols.

The user interface can be simple, the focus on this course is on communication and protocols. You may want to iterate between steps 5 and 6: you may notice during application implementation that your protocols have shortcomings and faults, or you may just want to improve something in the protocols as you learn more. Checkpoint 2 (Wednesday, 6.3.2024): The main functions of the protocols and the application should be ready for testing. Ensure that the current version is pushed to git repository and notify course personnel about the progress.

Step 7: Test the application and demonstrate it to the course personnel.

If you want to test your implementation in more challenging network environment, you can use e.g. Mininet to create a virtual network topology in your local machine with simulated delay and packet loss. Mininet is available only for Linux. Step 8: Write the final report.