Summer Training (CS48003) report submitted to

Indian Institute of Technology Kharagpur

in partial fulfillment for the award of the degree of

Master of Technology

in

Computer Science And Engineering

by

Sachin Kumar

(15CS30025)

Sensitive Data Renderer

Realized by :

***Visa Inc. Technology Center*,Bangalore**

Supervised by : Shibabrata Bhattacharjee (Director)

Address : Aquamarine building, L 4 &5 Bagmane WTC, Mahadevpura, Doddanekundi, Bengaluru, Karnataka 560037

Abstract

I have done my summer training at **Visa Inc. (bangalore, India).**The project we have done there is described below and visa also file a patent of my project.The project is described below. so, Sending sensitive information over HTTPS isn't totally verify as it is inclined to MITM attacks as authentications can be fashioned. Regardless of whether the MITM attack doesn't occur, the customer side of the system association can be hijacked. Clipboards can be captured, and pernicious contents can be included Into the program script. Usually, sensitive information like PAN numbers have appeared as content on the program, this can be effectively scarped of through the above attacks We need a protected mechanism/service by which we can render sensitive data on the customer side without it being undermined.So we have designed and developed four micro-services to solve the problem.

Contents

1. INTRODUCTION

Sending sensitive data over HTTPS is not completely secure as it is prone to MITM attacks as certificates can be forged.Even if MITM attack does not take place, the client side of the network connection can be hijacked. Clipboards can be hijacked, and malicious scripts can be added Into the browser script. Usually sensitive data like PAN numbers are shown as text on the browser, this can be easily scraped off via the above attacks We need a secure mechanism/service by which we can render sensitive information on the client side without it being compromised.

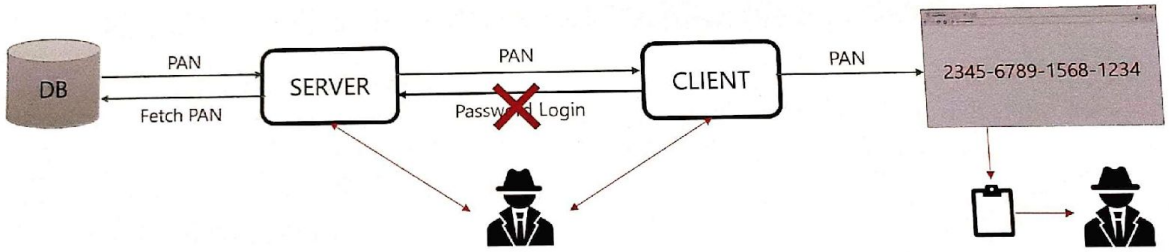


Fig 1. Problem with current data transfer channel

we have solved the problem by the combination of four micro services such as

- Multi Factor Authentication Module
- Text to media convert
- Encryption and Decryption Module
- Media Rendering Module

Fig 2. Four micro services to solve the problem

We are using password login as first factor and Time-based OTP often called TOTP as second factor of authentication to confirm the user's identity. After authentication, we fetch information

from DB and convert it into video to prevent copying. We are playing video in Shaka-player so it cannot downloaded if we render video using simple video tag. It can be downloaded so we are using Shaka-player to play video which provide DRM protection, Shaka-player is open source library provided by google. Even if MITM gets video, it is encrypted to prevent information retrieval. we are doing key exchange with the help of steganography and TOTP.

2. The Problem Statement

2.1 Attack Vectors

1) DRM Problem
    - Clipboard Hijacking
    - Malicious/Curious JavaScript
    - Malicious Chrome Extensions
2) Multi Factor Authentication
    - Session Hijacking
    - Account Takeover

2.2 Password Problems

- Password Breaches Make It Difficult for Users to Protect Their Individual Accounts.
- Since users have to create their own passwords, there's always a chance they won't create secure credentials. In fact, around 90% of user-generated passwords are considered weak and easily vulnerable to hacking. As technology evolves, so do the tools hackers use to crack people's credentials. Aside from merely guessing your password, a brute-force attack is the most common technique hackers use.
    1. Brute Force Attack
    2. Dictionary Attack
    3. Keylogger Attack
- A hacker uses a program to track all of a user's keystrokes. So, at the end of the day, everything the user has typed-including their login IDs and passwords-have been recorded

3. The solution

3.1 Mitigation of Attack Vectors

The DRM Mitigation

It provides a set of access control policies which mitigate against the modification, use and distribution of copyrighted media files. Once a media file, like a video file is DRM protected, a user cannot, in simple terms, rip it off from the browser as the access policy restricts this. Hence, the following attack vectors: Clipboard Hijacking, Curious JavaScript and Chrome extensions, will not be able to get the media file. We can use this to our advantage in developing the service. Instead of showing the sensitive information as text on the browser, we can do the following:

1. Convert the information into a video file.
2. Play this video on the browser using SHAKA PLAYER by Google.
    a. Shaka Player is Open Source
    b. Shaka Player provides DRM protection
    c. Shaka Player can be easily integrated into the client side.

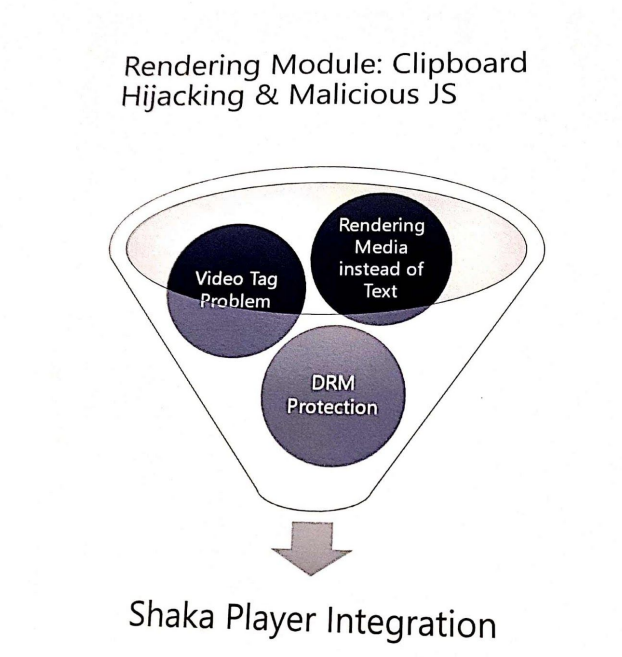


Fig 3. DRM Problem -shakaplayer solution

Multi Factor Authentication (MFA)

MITM attacks often lead to session hijack and account takeover. It is easily possible during key exchange between the client and the server which results in "secrets not being secrets" anymore. Therefore, authentication came into play where you must prove your identity to start the communication. However, certificates can be forged nowadays, and hence single factor authentication is not preferred anymore Now we come to MFA, where authentication/confirmation of a users' identity is confirmed by a combination of two or more of these following factors:

- Something they know -> knowledge
- Something they have- possession
- Something they are -> inherence

We can use MFA technology in our service to reduce the risk of MITM to almost NIL Google Authenticator provides 2FA services which can be integrated into the service.

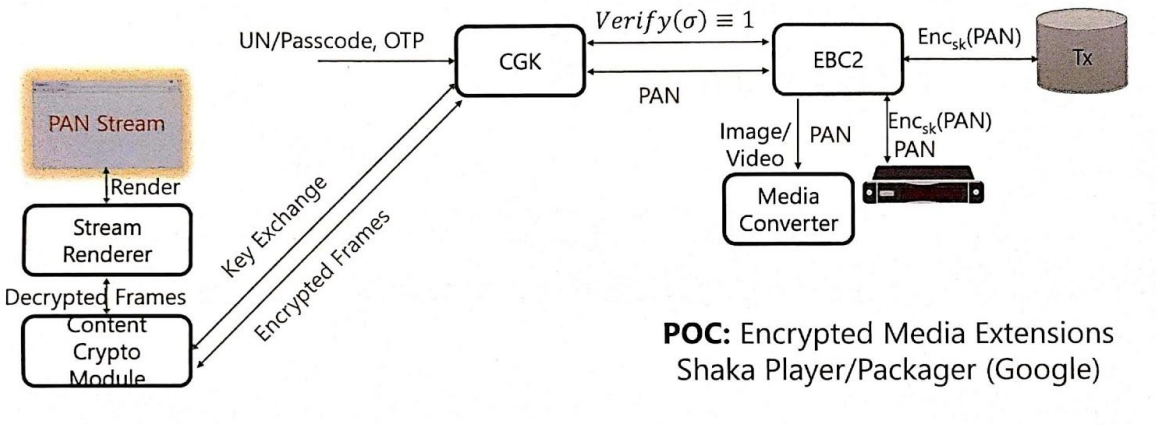


Fig 4.Techincal view of solution

ReCAPTCHA V3

reCAPTCHA is a service that protects your site from spam and abuse. It uses advanced risk analysis techniques to tell humans and bots apart. reCAPTCHA v3 returns a score for each request without user friction. The score is based on interactions with your site and enables you to take an appropriate action for your site.

Advantages:

- Spam & abuse protection for your website.
- Low friction, effortless interaction for users

Integration

a. Load the JavaScript API with site-key.
b. Call grecaptcha.execute on submit form action.
c. Send the token to your backend with the request to verify.

3.2 Complete solution

Man In The Middle (MITM) attacks often lead to session hijack and account takeover. Therefore, authentication came into play where you must prove your identity to start the communication. However, certificates can be forged nowadays, and hence single factor authentication is not preferred anymore So, we come to MFA, where the second factor is based on OTP.

So, we come to MFA, where authentication of a user's identity is confirmed by a Combination of two or more factors such as Something you know, something you have and something you are. For example,Username and password are the things which we know, Mobile Phone is the thing which we have, and Biometrics is part of who we are.

There are certain vulnerabilities associated with the currently used OTP based 2 factor authentication And our approach aims to overcome these issues. Before moving on to our approach, let us first understand the problems faced by the current OTP based 2 factor authentication. One of the major set back is that OTP is sent through SMS or phone call. As we know, Mobile sim can be stolen, or the SMS can be intercepted by service provider. Moreover, SMS delays and less network coverage also cause problems. To solve all these problems, we are using password login as the first factor of authentication and TOTP as a second factor of authentication.

So, what is TOTP?

A time-based one-time password (TOTP) is a temporary password, which is valid only for a certain time-interval, which in our case is 30 seconds Each TOTP is generated by the algorithm which depends on two factors one is the current time of the day and the shared secret is the second factor. To begin with, the user gets the TOTP through Google authenticator, which is an android application. While the server generates its TOTP through a java package provided by Google.

So, now the question is how these two entities are synced. Whenever the user wants to see any sensitive information for the first time, he will be shown the QR code along with the TOTP form on the screen, Now he should scan the QR code on the screen with the help of the Google authenticator application present on his phone. After this step, OTPs will be generated in the Google authenticator application. This generated TOTP has to be entered in the TOTP form on the screen. After these major steps have been completed. The TOTP will be hashed and sent to the server for authentication.

what does this QR code contains?

We are sharing a shared secret via QR code. This shared secret is created using a secure Key Encryption Key often called KEK and username. Key Encryption Key Is a 32.character key which should be stored in a secure storage and it never been shared. Key encryption key is common for all users. We are combining the username and KEK to create shared secret. We are

using username to make shared secret unique and Key Encryption Key to make it secure. So, we are generating shared secret dynamically. It can be generated any time only if you have Key Encryption Key Suppose there are 2M users, by traditional 2FA method using OTP we have to store 2M OTPs, while in our system we don't have a need to store TOTP or shared secret. So, it eliminates the need of maintaining server state. And save storage.

When a repeated user wants to see some sensitive information, is Google authenticator application already synced and has TOTP, he has to enter TOTP only.

So, we are sharing shared secret only once at the time of registration. After that it never been shared over network.

Thus, authentication problem gets solved.

After authentication, we are fetching data from DB and encrypt it. We are using TOTP for key exchange. And on the client side we are extracting the Key using TOTP.

Usually sensitive data rendered as text.so, hackers can copy the text and hijack the clipboard to get that data.

To solve this, we are converting the text into video. If we use normal video tag, video can be downloaded easily by some malicious JavaScript or chrome extensions. Now DRM problem comes in picture. To overcome this problem, we are using Shaka-player provided by Google. We are playing video in shaka player which provides DRM protection.

What is DRM ?

DRM provides a set of access control policies which mitigate against the modification, use and distribution of copyrighted media files. Once a media file, like a video file is DRM protected, a user cannot, in simple terms, rip it off from the browser as the access policy restricts this. Hence, the following attack vectors: Clipboard Hijacking, Curious JavaScript and Chrome extensions, will not be able to get the media file.

4. Conclusion

So, we have provided four micro services to solve the sensitive data transfer and rendering problem.To overcome the session hijacking and issues with password and OTP , we have used Multi Factor Authentication using Time-Based OTP. By generating TOTP and shared secret dynamically we have saved the storage.To solve the clipboard hijacking problem we converted text into video.We used shaka player to solve DRM problem.We also added an extra layer of encryption using steganography and hash ofTOTP as encryption key which eliminate the need to share encryption key which make it more secure.

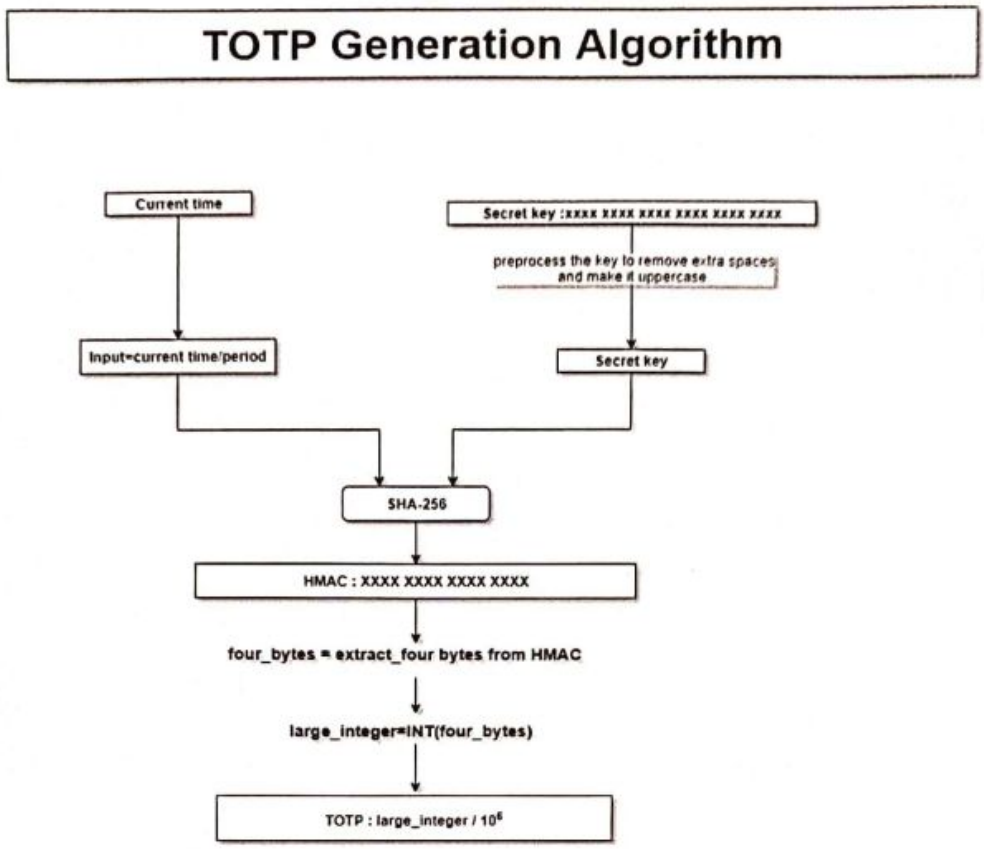
References

- MFA https://play.google.com/store/apps/details?id=com.google.android.apps.authenticator2&hl=en_IN
- recaptcha v3 https://developers.google.com/recaptcha/docs/v3
- Google auth https://developers.google.com/identity/protocols/OAuth2
- Maven https://maven.apache.org/

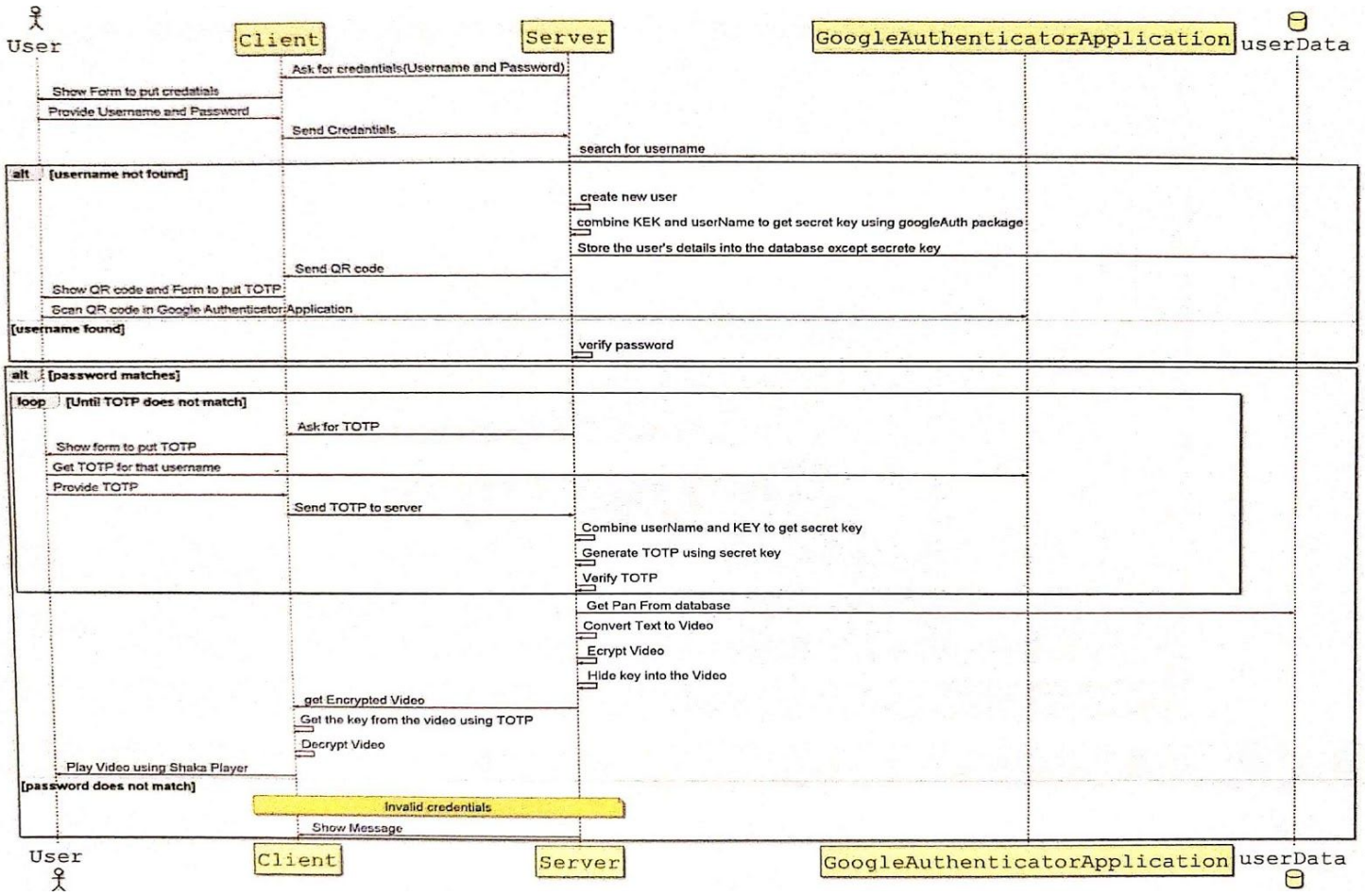
APPENDIX

1. TOTP generation algorithm

- Generate Secret key using google auth package provided by google.
- Secret key is the 56-character token, generated by authenticator itself.
- Decode it in base32 (Allowing characters form (A-2) and [0-9)
- Time used is the current epoch time and take the quotient when we divide it by 30. (30
- is used as we need a new code every 30 seconds)
- Generate a hash using both. Hashing algorithm is HMAC-SHA256
- Choose the offset (last element of hash). Remove most significant bit. Take modulo with
- one million and append "0" in front, if needed.



2. Activity Diagram of project of solution



3. Application Flow

When a new user comes it shows a QR code to synchronize the server with the Authenticator application. The user will scan the QR code using Google Authenticator mobile application. After that application will start generating TOTP for that particular user, TOTP will change in every 30 seconds. On the server side when a new user registered a secret key generated using google auth package and saved with the username and password in database/file. This secret key used for generating OTP when user try to login, and verify the user entered TOTP.If user entered TOTP match with the generated TOTP, user will see the PAN number.