Instituto Superior Técnico — Taguspark

Network and Computer Security

Smartphone as a security token (Report)

Group 18

ist173891 — David Gonçalves
david.s.goncalves@tecnico.ulisboa.pt

ist194104 — Leonardo Troeira
leonardo.troeira@tecnico.ulisboa.pt

ist194122 — Francisco do Carmo
francisco.baptista.carmo@tecnico.ulisboa.pt

1 Problem

Password-based authentication has the advantage of being simple and convenient to use. However, there are too many attacks that can be performed remotely that render this authentication mechanism not secure enough on its own—phishing, XSS, SQLI, rainbow tables, dictionary attacks, brute force... to name a few.

As such, if we're seeking stronger security mechanisms, one option is to make use of multi-factor authentication, usually involving knowledge and possession. This poses a trade-off between security and usability however, since the user will be needing to carry something with him all the time (e.g. smartphone).

Besides authentication mechanisms, we're also faced with the challenge of data possibly being stolen, either by sniffing network packets or acquiring control over the victim's file system, among other possible attacks. Therefore, assuring that data is stored and transmitted securely (relying on encryption algorithms) is a must. And again, for additional security, we should have another factor of authentication.

1.1 Requirements

1. The web server must have mechanisms to prevent XSS and SQLI.
2. Communication between client and server must be done through a custom secure channel.
3. The system must provide two-factor authentication.
4. User files must be stored encrypted.
5. User files must be decrypted only if the user has their smartphone nearby the client application.
6. The system must provide fault tolerance. If the system crashes after decrypting the files, make sure that the files are encrypted again.
7. Communication between the smartphone and client application must be done using a secure protocol.

1.2 Trust assumptions

Our problem assumes that the network and user's device cannot be trusted. However, we assume that the user's device is not compromised while it is in use, and that the smartphone and server are not compromised either. We also assume that the server's certificate is trusted.

2 Proposed solution

For our solution, we shall create a simple website—on a server machine—with a register and login forms, for which we shall obviously sanitize user input and make use of SQL parameters in order to prevent XSS and SQLI respectively. The user's password shall be stored hashed and salted in order to prevent it from being easily obtained by potential attackers. As an additional security layer for authentication, we shall use 2FA with the help of Duo SDK, thus leveraging the smartphone's proximity to the user as part of an increased security solution. The server shall also log any user actions.

The user shall be able to download files, from the server's website, which come encrypted with a symmetric key. The user's key pair shall be generated every session, and they shall be stored in the user's smartphone, which will necessarily need to be connected to the user's device in order to be able to receive it, and we shall make a simple JAVA app for this purpose. The smartphone and user's device shall connect through Wi-Fi, utilizing a secure protocol (e.g. WPA2/3). For extra security, the user's private key should be symmetrically encrypted with a password, which should be different than the user's login password (e.g. KeyStore). The symmetric key used to encrypt each file shall also be stored in the smartphone encrypted in the same fashion as the user's private key.

We shall create a simple client application for viewing these encrypted files that were downloaded from the server. In order for the client application to decrypt these files, the smartphone must be connected to the user's device. The system shall be fault tolerant—as soon as this connection is lost, the client application shall clean its state, thus the user shall no longer be able to view the encrypted files. We plan on using a stream cipher for this purpose, as in theory we can have large files that don't fit in volatile memory.

For registration and login, an AES key shall be exchanged using a secure protocol (e.g. Diffie-Hellman). Subsequent communication between the user and the server shall be done using symmetric encryption with this key, in order to guarantee confidentiality. For integrity, each message shall be hashed together with random padding (using SHA-2) and for authenticity this hash shall be signed with the sender's private key.

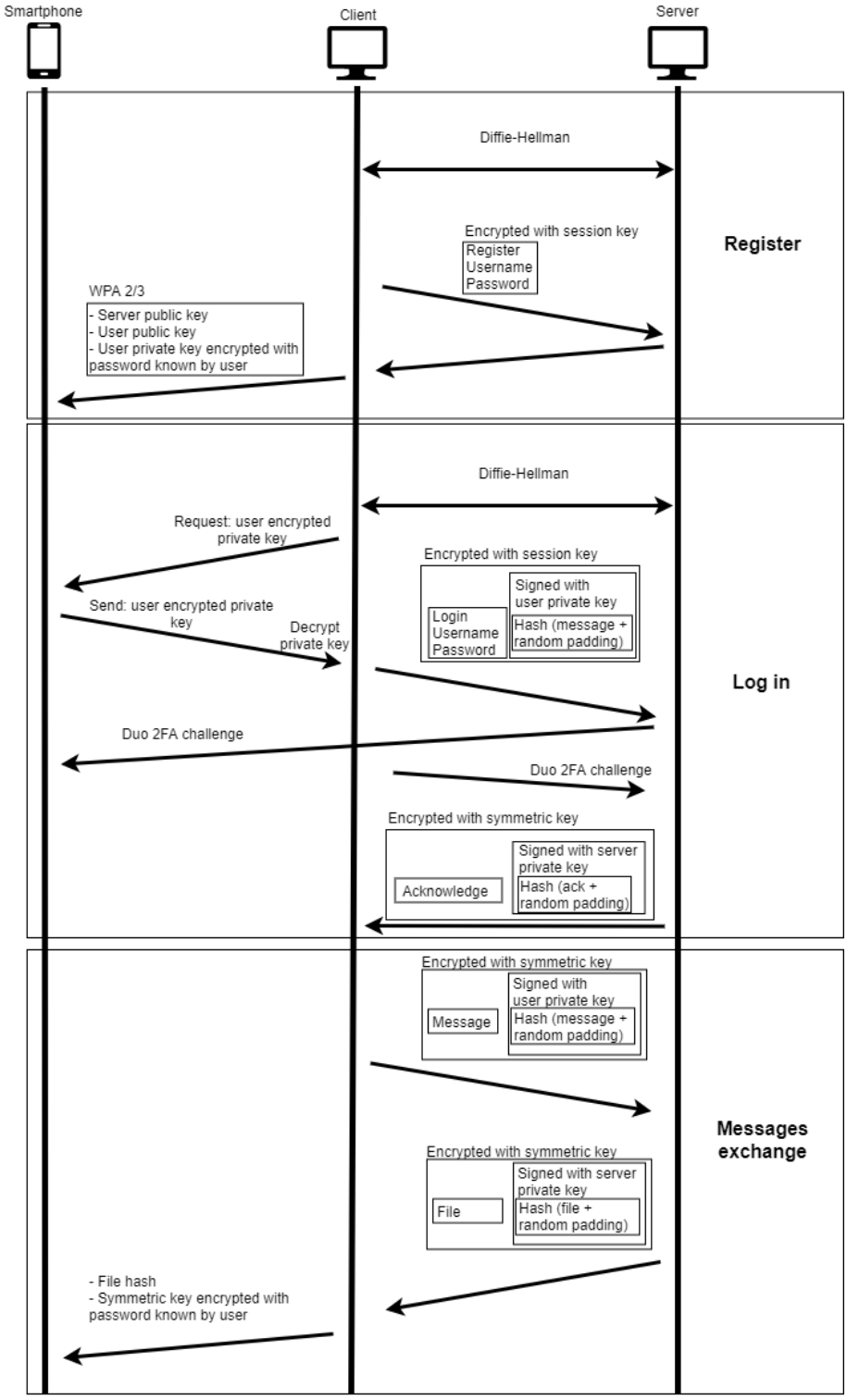

Figure 1: Secure communication

3 Results

- **2FA:** In order to increase strong security authentication, we proposed to implement a 2 factor authentication service in order to authenticate the client.

  We decided to use Duo SDK, because it is a user-centric zero-trust security platform that allows an application to protect sensitive data.

  In our approach the 2FA mechanisms are: "something the user has" (phone) and "something the user knows" (password).

  We managed to integrate Duo in our application successfully, making the user to authenticate in the application only upon inserting the correct password and successfully responded to the Duo challenge (push or sms on user phone).

- **Proximity:** We assure that the user can only view, download or upload files (be logged in the system) when he is close to the computer that he is logged in. For this matter, the machine where the user is operating and the smartphone require to be connected to the same network. The client app pings the smartphone every 5 seconds, if the smartphone is down, proximity is broken and the client is logged out, making it impossible to view, upload or download files.

- **WPA2:** We assume that the client and the smartphone are connected through WiFi under the WPA2 protocols. Making the connection between these two secure. In our project we simulated a smartphone using a virtual machine so we when testing we do not need to use WPA2 for the communication.

- **XSS and SQLi:** To mitigate cross-site scripting we sanitize all the inputs that are not 100% controlled by us. These validations are deployed both in the client side (JavaScript) and in the server side (PHP). One additional benefit of some of the validations is that they also make the user choose stronger passwords when registering a new account.

  To avoid SQL injections we used prepared statements to ensure that the values passed from the client to the server do not affect the database in an unexpected and incorrect way.

- **Password strength:** When creating a new account the user needs to enter a password that complies with the following regular expression "(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[!@#\$%\&̂]).{10,}". The password must be composed by at least 10 characters, of these 10 characters at least one of them must be a lowercase letter, one of them must be an uppercase letter and the user must also use at least a digit and a special character. By doing this we ensure that the user creates stronger passwords, and for example by adding the special character we can stop/mitigate common dictionary attacks.

  We also assume that the user will not disclose his password to anyone.

  The users' passwords are hashed and salted before being saved in the database.

- **OSI layer:** .....
- **JS crypto issues:** We don't encrypt things on the client (and use smartphone instead): lack of javascript crypto libraries
- **AES GCM:** Using AES GCM we created a Message Integrity Check (MIC) and with that we can prevent attacks (e.g. bit-flip attacks) on the encrypted packets. The MIC adds a few bytes to each packet and that makes the packets tamper proof.

  This approach is also useful because now with a MIC we do not need to sign the hashes as we had proposed before. This key (MIC) is only known by the client and the server and that ensures authenticity, however we cannot ensure non-repudiation, for that we would also need to ensure freshness.
- **Signed hash with padding:** .....
- **Diffie-Hellman:**

  We made use of DH in the secure channel, to ensure that we can transmit sensitive information in a non-secure channel.

  - P and G always the same Prof: Sim, PFS mantém-se nessa situação. Aliás, até é normal o P e o G serem iguais em todas as sessões de Diffie-Hellman. Existem até valores standards (https://www.ietf.org/rfc/rfc3526.txt seção 3 por ex.) que podem usar para o primo e o gerador, para que não tenham que ser vocês a criar esses números de modo a que eles tenham as propriedades que devem ter. - public values signed with RSA - PEM (+ DER) format to comunicate public keys between PHP and Java - Client acts as middleman for secure communication (doesn't encrypt / decrypt anything himself) - Smartphone -¿ Client -¿ Server
- **Problems:**
  - **Duo registration:** Currently the user's device (smartphone, etc) is associated with Duo at first login. It should be done at registration to avoid an attacker finding the user's login password and associating his device with the account, should the attacker make the first login.

    However, because inserting the user in our database could fail after associating the user's device with Duo, this would require deleting the user's account at Duo's Admin panel. For this, we'd need to use a PHP third-party library for making REST requests to the Duo's API and integrate it with our Duo account.

    This would give us some work to implement and is not the most important security fault of this project, so we chose to focus on other aspects instead given the time we had available.
  - **Check server's certificate:**
  - **Confidentiality:** It is possible to know that a communication is going on and with that information the attacker can try to infer what kind of message in being passed in the channel at that moment, and with more communications listened

the attacker can start to observe a pattern that can lead to discover what kind of pattern each message creates. Everyone can know what kind of resource the user is accessing in a time period (e.g. login.php, register.php, main.php)

4 Plan

4.1 Versions

- **Basic:** For our basic solution we shall create a website with a register and a login forms, with mechanisms to prevent XSS and SQLI attacks, that stores passwords hashed and salted, creates a key-pair for each user upon registration and gives it to them. The server shall also be able to encrypt trivial text files—context doesn't matter—with a symmetric key shared with the user at login, and allow the user to download them. We shall then create a simple client application (typically JavaScript on the website) so that these files can be viewed by the user, using a stream cipher.

- **Intermediate:** On top of the basic version, we shall implement 2FA using Duo SDK and add another use for our smartphone, which is proximity detection. The user's key pair, server public key and symmetric key for each file shall be stored in the smartphone, we'll design a simple Android app to take care of this task. The communication with the smartphone shall use a secure Wi-Fi protocol (WPA 2 or 3). We shall also log user actions on the server side.

- **Advanced:** Finally, to complete our solution, we shall guarantee that as soon as the connection to the smartphone is lost or the client application crashes, the encrypted files shall no longer be visible. Additionally, we'll design our own custom secure channel for communication between server and client, as explained in the previous chapter. We'll also encrypt the user's private key and files' symmetric keys using an AES key derived from a password that only the user knows about.

5 References

- **Duo SDK:** Duo's trusted access solution is a user-centric zero-trust security platform to protect access to sensitive data at scale for all users, all devices and all applications.

  https://duo.com/product/every-application/supported-applications/apis

  Found: Yes

  Installed: No

  Tested: No