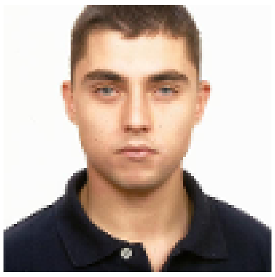
Instituto Superior Técnico — Taguspark

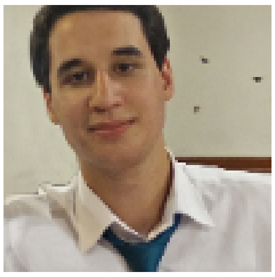
Network and Computer Security

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. Communication with the server may only happen whilst the user has their smartphone nearby the client application.
7. The system must provide fault tolerance. If the system crashes after decrypting the files, make sure that the files are not stored encrypted.
8. 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 (e.g. PC) cannot be trusted. However, we assume that the user's device is not compromised while the user is using the app, 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, separating the client logic (JavaScript) from the server logic (PHP). There will be a register and login forms, for which we shall sanitize user input and make use of SQL parameters in order to prevent XSS and SQLI respectively. The user's password shall be relatively secure (min. 10 chars, contain lowercase, uppercase, digits and symbols) and shall be stored hashed and salted in order to prevent it from being easily obtained by potential attackers, should the server's database be compromised. 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 user shall be able to download files from the website, encrypted with a new random symmetric key. We shall make a simple Android app that will generate the user's long-term RSA key pair at its installation and will store it in the user's smartphone. 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 with the user's public key.

Our client application (website's JS) shall be able to view these encrypted files that were downloaded from the server (for simplicity we'll consider only txt files). 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 nor communicate with the server. We plan on using a stream cipher for encrypting / decrypting, as in theory we can have large files that don't fit in volatile memory. For binary fails the alternative would be decrypting to the disk and encrypting when proximity is lost, but this does not prevent other kinds of faults (e.g. PC shutdown).

For registration and login, a shared AES key shall be obtained using a secure protocol, we propose 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, a MIC can be generated using Galois/Counter Mode (GCM) mode. Our only authenticity guarantees are that either the client or server created the message. We could provide non-repudiation by signing the message and having freshness, but we'll not consider this for this project. DH provides us with perfect forward secrecy as long as we generate new key pair values every session, but it is susceptible to man-in-the-middle, so we shall sign the exchanged public values with long-term RSA private keys.

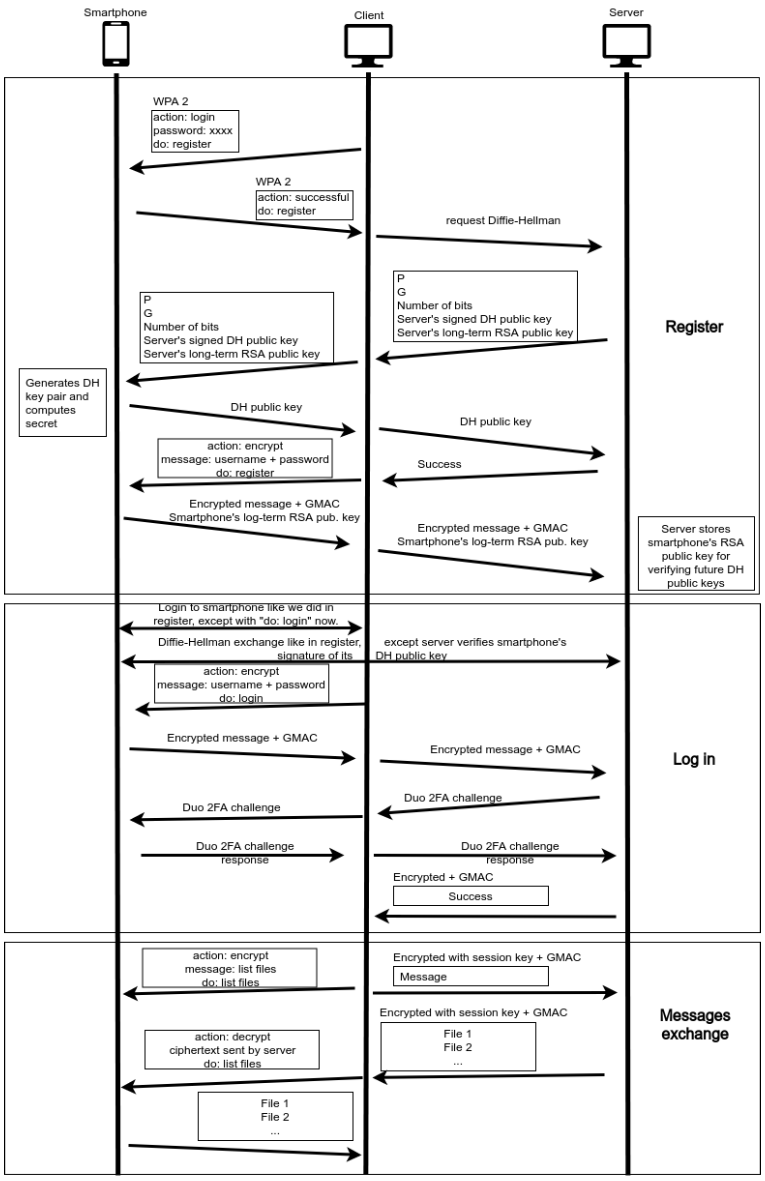

Smartphone
Client
Server
WPA 2
action: login
password: xxxx
do: register
WPA 2
action: successful
do: register
request Diffie-Hellman
P
G
Number of bits
Server's signed DH public key
Server's long-term RSA public key
P
G
Number of bits
Server's signed DH public key
Server's long-term RSA public key
Register
Generates DH key pair and computes secret
DH public key
DH public key
action: encrypt
message: username + password
do: register
Success
Encrypted message + GMAC
Smartphone's log-term RSA pub. key
Encrypted message + GMAC
Smartphone's log-term RSA pub. key
Server stores smartphone's RSA public key for verifying future DH public keys
Login to smartphone like we did in register, except with "do: login" now.
Diffie-Hellman exchange like in register, except server verifies smartphone's signature of its DH public key
action: encrypt
message: username + password
do: login
Encrypted message + GMAC
Encrypted message + GMAC
Log in
Duo 2FA challenge
Duo 2FA challenge
Duo 2FA challenge response
Duo 2FA challenge response
Encrypted + GMAC
Success
action: encrypt
message: list files
do: list files
Encrypted with session key + GMAC
Message
Encrypted with session key + GMAC
Messages exchange
action: decrypt
ciphertext sent by server
do: list files
File 1
File 2
...
File 1
File 2
...

Figure 1: Secure communication

3 Results

- **XSS and SQLi:** To mitigate cross-site scripting we sanitize all the inputs that are not 100% controlled by us (user inputs, server messages and smartphone messages) that end up being displayed. These validations are deployed both in the client side (JavaScript) and in the server side (PHP).

  To avoid SQL injections we used prepared statements to ensure that the values passed from the client to the server are compiled before being executed, thus not affecting the database in an undesirable way.

- **2FA:** 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 authenticate in the application only upon inserting the correct password and successfully respond to the Duo's challenge.

- **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 in the same network. If connection with the smartphone is broken, proximity fails and the client is logged out, moreover losing access to the keys to decrypt the files.

- **WPA2:** We assume that the client and the smartphone are connected through WiFi under the WPA2 or WPA3 protocols. Making the connection between these two secure. In our project we simulated a smartphone using a virtual machine so when testing we did not use WPA2 for the communication.

- **Password strength:** When creating a new account the user needs to enter a password that must be composed by at least 10 characters, and contain at least a lowercase, an uppercase, a digit and a symbol. By doing this we prevent brute-force attacks, and also by requiring this complexity we can mitigate common dictionary attacks. The users' passwords are stored in the server's database hashed and salted.

- **OSI layer:** We deal with the application, presentation and session layers which allow us to have some confidentiality in regard to the content of the messages. Still, we are unable to provide confidentiality in terms of knowing that a communication is going on and which resources are involved.

- **JS crypto issues:** We don't encrypt things on the client, and use the smartphone instead. One reason for this is the lack of JavaScript crypto libraries. The bonus is that it leverages proximity. The downside is that it is harder to program.

- **AES GCM:** Using AES operating in Galois/Counter Mode (GCM), we have a Galois Message Authentication Code (GMAC) that grants message integrity, so we can prevent bit-flip attacks on the payload.

  This approach is also useful because with GMAC we do not need create our own MACs or sign the messages' hashes. GMAC can only be generated by either the client or the server and that ensures some authenticity, however it does not ensure non-repudiation, for that we would need to sign the messages' hashes and implement freshness.

- **Diffie-Hellman:**

  We made use of DH protocol for establishing a shared secret (at login and register) in order to have perfect forward secrecy. P and G values are always the same, but private and public values are different each session. Because DH is susceptible to man-in-the-middle, we sign the public values with long-term RSA private keys. Except on the client's public value at register because the server has no way to verify the public value. Because we use Java and PHP we had to convert PEM to DER formats and vice-versa.

4 Evaluation

- **Brute force:** Requiring a lengthy password with a wide range of possible characters mitigates brute force, for now, yet is inconvenient for the user and may . A better approach would be to limit the number of login attempts within a time period.

- **Duo registration:** Currently the user's smartphone is associated with Duo on the first login. It should be done at registration to avoid an attacker hijacking the account by potentially finding the user's server login password and associating his own smartphone with the account, should the attacker make the first login.

  However, because inserting the user in our database could possibly fail (e.g. mysql down) 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 considering the time we had available.

- **Check server's certificate:** We do not check if the server's certificate is valid as this would require a trusted 3rd-party. This is of course an essential step for a real world application.

- **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 is 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 message we are sending. Since we use a stream cipher the length of the ciphertext is the same as the plaintext. Moreover, everyone can know what kind of resource the user is accessing in a time period (e.g. login.php, register.php, files.php)

- **Non-repudiation:** Our solution does not implement non-repudiation nor logging. It is also possible to perform replay attacks within a session as we do not have freshness implemented.
- **Encrypting files + Fault tolerance:** We were unable to implement file encryption / decryption in time. But we implemented .txt viewing in our client app and proximity detection, which would be the building block for file decryption fault tolerance.

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