Pass, No Record: An Android Password Manager

Alex Konradi, Samuel Yeom

December 4, 2015

Abstract

Pass, No Record is an Android password manager that allows users to securely retrieve passwords from a server and use them through a virtual keyboard, which automatically enters the passwords upon the user's consent. After logging into the app with a master password, the user can add, update, and delete passwords, which are encrypted and then stored on an untrusted remote server. The system is designed such that the passwords remain securely encrypted even if the server is compromised. An independent web client allows the user to change the master password and remotely log out of the Android client in case the Android device or the master password is lost or stolen.

1 Introduction

Most websites that implement authentication ask the user to create an account by picking a password that only the user knows. For maximal security, the user should not reuse the same password on multiple websites. Otherwise, an attacker that learns a user's password by compromising one website can use the same password to gain unauthorized access the same user's accounts on other websites. However, the typical user maintain dozens of accounts, and as the number of accounts increases, it becomes increasingly infeasible to maintain unique passwords without resorting to insecure techniques, such as writing the passwords down.

Pass, No Record provides a more secure password storage in the form of a remote server and a client that the user installs on their Android device. The only password that the user has to remember is the Pass, No Record master password, which gives the user access to the password manager. The site-specific passwords are encrypted before being stored on the server, making it much harder for the attacker to figure out the plaintext site password even if they somehow gain access to the password storage. When these passwords are retrieved from storage, they are entered into the password field automatically by the client. Not only is this more convenient for the user, but it also makes the password less vulnerable to physical observation. Even if the user loses their Android device, they can prevent others from accessing the passwords by simply using our web client to remotely log out of the Android client.

In Section 2, we enumerate the components of Pass, No Record and describe how they interact with each other. We define our threat model in Section 3 and describe the cryptographic primitives and protocols that we use. Finally, Sections 4 and 5 discuss the implementation details of the various functionalities.

2 Architecture

Pass, No Record consists of three components: server, Android client, and web client. Most of the user's interactions with Pass, No Record will be through the Android client, which communicates with the server through HTTPS. When the user saves a password, the Android client encrypts it and sends it to the server, where it will be stored in the encrypted format. Conversely, when the user requests a password, the Android client retrieves the encrypted password from the server and decrypts it. Even though the Android client requires Internet access to store and retrieve passwords, this is not a problem because the user needs Internet access to log in to a website anyway.

The web client offers limited functionalities whose purpose is to let the user recover if their Android device or their master password is stolen. Unlike the Android client, the web client runs server-provided code. Therefore, to limit the effect of a server compromise, we recommend that users do not use the web client unless necessary.

3 Security

As discussed above, the server provides the code on the web client, so we consider the web client to be a part of the server for the purpose of security evaluation.

3.1 Threat Model

Our threat model is an attacker that does not know the master password and controls either the server or the Android client, but not both. If the attacker controls the server, the server can ignore requests from the client or give incorrect passwords as responses. However, the attacker should not be able to decrypt the stored passwords even if they read all network traffic into and out of the server. If the attacker controls the client instead, they can use the client to access the user's passwords if the user is logged into the client. However, the attacker should be denied access to the user's passwords as soon as the user remotely logs out. Finally, in both cases, the attacker should not learn the master password.

3.2 Password Encryption

We provide secure storage of passwords by requiring the client and the server to collaborate to retrieve the stored passwords. The client encrypts the passwords before sending them to the server for long-term storage. Then, the client must

communicate with the server to retrieve the encrypted password, which the server cannot decrypt because the encryption key does not leave the client.

We use AES in the Cipher Block Chaining mode with random initialization vectors to encrypt the passwords. This brings up the question of how to choose the AES private key. We cannot simply store it on the server because we do not want to give the server the ability to decrypt the passwords. Since our threat model includes an attacker who has control of the entire Android device, we would also like to avoid storing it permanently in the Android device. Instead, we derive the key from the master password by applying 1000 iterations of PBKDF2 with HMAC-SHA256 as the pseudorandom function. The salt for PBKDF2 is stored on the server and is given to the client upon request.

3.3 Proof of Identity

In addition, we want to be able to authenticate the user by checking whether the master password that the user entered is correct. We do not want to do this by sending the server the master password because the AES private key is derived from the master password. Instead, authentication involves a Proof of Identity (PoID), which the client derives from the master password by using PBKDF2. To make sure that the two applications of PBKDF2 result in different outputs, we use different salts. This takes some care because the client retrieves both salts from the server, which may be compromised by an attacker. In particular, if the server tricks the client into thinking that the key salt is the PoID salt, the client will send the AES private key to the server, thinking that it is the PoID. To prevent this, we make the set of possible key salts and the set of possible PoID salts disjoint.

3.4 Authentication Token

The above techniques are sufficient to create a secure system, but it requires the user to enter their master password for every transaction. For convenience and usability, we want to give the user the option of staying logged into the Android client. One way to do this is to store the PoID on the client while the user is logged in. However, this allows the attacker to obtain the PoID if they control the device while the user is logged in. Instead, we use a random token, which the server generates when the client first authenticates with the PoID. Then, the client can store the random token and use it for future authentications until the user logs out.

In order to limit the impact of server database breaches, the server hashes the PoID and the tokens before storing them.

4 Android Client

4.1 Logging in

The Android client supports a single logged-in user per device, with unlimited devices logged in for each account. Upon installing the application, the user will be prompted to log in; after entering an unrecognized username, they will be prompted to create an account. Behind the scenes, the client makes two requests to the server.

The first is to retrieve the salts for the specified account; if the server indicates that the username does not exist, the user is prompted to create a new account. If the user chooses to do so, the client first generates salts for both the PoID and secret key. It then uses these salts and the password the user provides to compute the PoID and private key, and sends the salts, the PoID, and the username to the server. The client then saves the username, private key, and the token provided by the server, and indicates to the user that they are logged in.

If the initial salt retrieval mentioned above is successful, the client computes the user's PoID and attempts to authenticate with the server by sending only the PoID and username. Upon successful authentication, the client saves the username, the authentication token that the server sends, and the private key that is computed locally. If authentication is unsuccessful, the client prompts the user for the correct password.

4.2 Storing and Retrieving Passwords

Password storage and retrieval is the main feature of Pass, No Record, and the Android client attempts to perform these functions as easily and transparently as possible.

Both processes are initiated by selecting a password field in the browser. Upon selection, the client attempts to prefetch the user's password for the site in the background.

If the server responds that the password does not exist, the user is presented with the option to create one for the site. After either providing a password or entering one by hand, the user requests that the password be saved. The client then encrypts the password with the saved private key and uploads it to the server, using only the saved token for authentication.

If the prefetch succeeds, the user is presented with the option to fill the password field automatically. Selecting this causes the client to decrypt the prefetched password in memory and send it to the browser as if the user had entered it normally. In either case, the decrypted password is not stored in memory within the Android client, though since it is passed via Android's inter-process communication (IPC) system, it could be observed by a malicious operating system or process with root privileges.

Since the password is not cached, a user can request that the server deauthenticate a logged-in device by rejecting its token, in which case any requests

made afterwards will fail.

5 Web Client

5.1 Remote Logout

When a user loses their Android device, they may want to remotely log out of the Android client to prevent unauthorized access to their passwords. Remote logout allows this by invalidating all existing authentication tokens associated with a user. To use this functionality, the user should go to the remote logout webpage, which prompts the user for their username and their master password. After the user enters these credentials, the JavaScript code on the web client first sends an HTTP request for the PoID salt associated with the username. Then, the web client uses the entered master password and the PoID salt to derive the PoID. Finally, the client sends the username and the PoID to the server, which verifies the PoID and invalidates all tokens for that user.

5.2 Changing the Master Password

Anyone who knows a user's master password has unlimited access to the user's Pass, No Record account. Therefore, once a user learns that their master password is stolen, they should change it as soon as possible. The current design has the flaw that anyone who knows the master password can change it, but one way to mitigate this is to send a verification email to the user.

With respect to implementation, changing the master password is similar to remote logout, but it is complicated by the fact that the AES private key also changes with the master password. This means that all stored passwords must be decrypted with the current key and then re-encrypted with the new key. When the user enters their username, the current master password, and the new master password through the web client, it retrieves the salts from the server and computes both the current and the new AES private key and PoID. Then, the client uses the current PoID to request all of the user's passwords, which are decrypted and re-encrypted with the appropriate keys. Finally, the client sends the username, the current and new PoIDs, and the newly encrypted passwords to the server, which verifies the current PoID and updates it along with the stored passwords.