Tom Ginader

MSCS 630

Final Project Final Writeup

Due: 5/9/17

Abstract

The purpose of this project is to design and create a mobile app that properly uses an HMAC to ensure authentication and integrity through SMS. An HMAC is a message authentication code that involves a shared private key, and a hash function. The basic process of an HMAC is to take the sender's message and append a shared private key to it. This combination of message and key will then go through a hash function to produce the HMAC. This HMAC will then be appended to the original message. When the receiver receives the message, they can take the message and their own copy of the key and hash the combination themselves. They can then compare that HMAC to the one in the sender's message. If they match, then the message is authenticated. If they don't match, then it is likely that the message was altered before it reached the receiver. This can be used to prevent man-in-the-middle attacks, and it can also be used to prove that the sender is who they say they are. The app is being built in Android Studio, and it is written in the Java programming language. It is open source and can be located at: https://github.com/gin-nader/mscs630ginader/tree/master/prj/writeup/code/MSCS630FinalProject.

Introduction

The motivation behind this project is to take a secure process that I find interesting, and to implement it in a practical mobile app. I wanted to do something different, and something that didn't just encrypt SMS messages between two users. That is something that's being done by a lot of companies and people, so I wanted to build an app that isn't as common. An app like Signal uses an HMAC-SHA256 as well as AES-256 encryption, but a lot of other apps only focus on encryption. Encryption is not the only thing that matters when it comes to security. Something like symmetric key encryption does not ensure authentication or integrity, usually. It just prevents other people from reading your messages, unless you use authentication by using the last block of AES, for example. However, this is not completely secure or efficient. It is much better to use an HMAC instead. So, most of the time, there's no way of knowing that the sender of those messages is who they say they are, if you just use AES encryption. Also, just because a message is encrypted, it does not mean that the message cannot be intercepted and altered before it reaches the intended receiver; and, the receiver would have no idea if the message was changed. Likewise, you don't ensure security by just using an HMAC. Ideally, you would want to use both an HMAC and encryption. But, for the scope of this project, I will be focusing on just the HMAC aspect.

Related Work

As stated above, Signal is an app that does use an HMAC to check for integrity and authentication. But apps that do use an HMAC in addition to encryption are rarer than apps that just perform encryption. WhatsApp is another example of an app that used both an HMAC and encryption, but it contains security flaws based on these two functions.

Methodology

This project is a real challenge because I've never had to build an Android app before, and I've never worked with Android Studio. So, a majority of this project, so far, has just been reading documentation

and looking at tutorials for how to send and receive SMS messages. Figuring out how to send an SMS message between two devices was not too hard to implement. The hardest part was figuring out how to deal with permissions. You need several permissions needed from the user just to send and receive SMS messages. In the newest versions of Android Studio API's, you need to create code that will prompt the user for a given permission, and then respond to their input. But, for the older API's you just have to put the permissions in the manifest xml file. In the end, I opted to use a lower API version due to the scope of this project. This made things somewhat easier.

I also needed a way to receive SMS messages. The reason for this is that you can send SMS messages easily enough, but the default messaging app is still the app that receives the message; and, for my app, I wanted the ability to send and receive messages in the same app. This also ended up being a requirement because I needed to develop my own code to check that the HMAC is valid. In order to do that, I needed my own activity that displayed all of the messages in the phone's inbox.

After more searching, I was able to find documentation for how to implement an activity that would receive SMS messages by taking the messages in the phone's inbox and displaying them in a list view activity. I originally wanted the ability to send and receive messages within one activity, but I opted to put them in two separate activities with a button in the SMS sender activity that takes you to the app's personal inbox. This ended up working out for the better, anyway.

So, after a lot of time and research, I was able to figure out how to send and receive SMS messages between two Android devices. Now, the true challenge is to implement the HMAC with SHA1-160 or better. So far, I have a field for the private key, and a makeshift, barely usable hash function as a placeholder. It can hash the message with the key, then send the message to the other phone. It then takes the message and key and hashes it and compares it to the HMAC in the sender's message. To check if the message has been verified, you can click on the message in the inbox and a toast will appear. The toast will say that the message is verified and has not been altered if the HMAC's match, and if the HMAC's don't match, then it says that the message is not verified and has most likely been altered.

I have added two updates to the project since the milestone was written. I was able to implement the SHA-3 256 bit hashing algorithm thanks to Bouncy Castle's security library. I wish I could have implemented the algorithm myself, but I unfortunately did not have enough time. Too much time was spent learning and figuring out how to use Android Studio. Also, we never had a lab where we implemented a hashing algorithm, so this also made it challenging to try and implement my own. However, with this library, my algorithm now works really well, and it produces a legitimate HMAC. The HMAC is still appended to the end of the message and encoded in a single hex string.

The second update was that I added the functionality for users to input their own key, instead of having a single key hard coded into the program. This greatly increases the security and functionality of my app. It was not as hard to implement as I thought it would be. I am glad that I was able to add this function as it really ties the whole app together, and it allowed me to do some additional experiments involving the same keys and different keys.

Experiments

Some experiments that I've tried have been sending a message normally through my app to another device. Another experiment was sending it through my app, then sending the message back through the default messaging app after stripping the HMAC from the message. I've also sent a message through my app, then I've altered the HMAC, and sent it back through the default app. Finally, I have sent a message through my app, I changed the message but not the HMAC, and then I've sent it back through the default messaging app.

Since the updates to my app, I was able to come up with additional experiments. The first new experiment is what happens when two users use the same key to provide authentication. The second experiment is what happens when two users use slightly different keys, but the sender's HMAC and message remains the same.

Analysis

From these experiments, I was able to develop a list of results. When sending a message normally through my app, it works as intended and when I click on the message, the toast shows that the message has been properly verified. When I strip the HMAC completely, no toast shows up at all. This can be both a good thing and a bad thing. It is good because the user will immediately realize that the message has not been verified. However, ideally, a toast should show up explaining that there is no HMAC in the message. When I altered the HMAC and not the message, the toast properly displays that the message has not been verified. Finally, when I change the message and not the HMAC, the toast properly shows that the message has not been verified.

With the addition of the two new experiments, I can properly confirm the authentication aspect of my app. When the users input the same key, the message is properly authenticated and the toast displays the correct message. When the users use two slightly different keys, the message is not authenticated even though the HMAC remains the same. The proper toast is also displayed, warning the user that the message is not authenticated.

Conclusion

Overall, this has been a very interesting and educational project. I enjoyed implementing an HMAC at an absolute base level. I like that my findings prove that the HMAC actually works in practice, even at an extremely basic level. If someone were to intercept the message and either change the message or HMAC, then my app would notify me. In the future, I would like to add a function where the key is stored in an encrypted file and the app pulls the key from that file. Or, perhaps I could generate the key based on the message, and it would only last as long as the app is running. This way the key doesn't have to be stored anywhere which would improve usability as well as the security of the app. Also, I would like to be able to encrypt the message with AES 128 bit or better in addition to using an HMAC. Ideally, I would also use RSA or a similar public key cryptosystem to set up a secure channel first so AES keys can be exchanged securely.

Finally, I would want to greatly improve the interface and graphical design of the app. Right now, it looks very plain, and it is using the default Android Studio interface. I would try to spend as much time as possible making the app look sleek. I believe that is what would allow the app to gain popularity. When other apps have similar functions, you have to try your hardest to separate your app in a positive way from all the others to make your app stand out. After that, I would be left with a fully functional secure messaging app that I could share with friends, or possibly have it available to the public.

References

Guido Bertoni, Joan Daemon, Michael Peeters, and Giles Van Assche. 2011. The Keccak reference. (January 2011).