Signa-Secure

By: Rivka Nadell and Sara Teigman

Final project, semester 1, year 2021.

Computer Science Department at Jerusalem College of Technology

Instructor: Barak Einav

Table of Contents

[Introduction 1](#_Toc62039674)

[Project Purpose 2](#_Toc62039675)

[Why Choose Us? 2](#_Toc62039676)

[Design & Architecture 3](#_Toc62039677)

[Project Flow 3](#_Toc62039678)

[Flow Explanation 3](#_Toc62039679)

[How To Use Our Project 4](#_Toc62039680)

[Security analysis 5](#_Toc62039681)

[Mode of Work 5](#_Toc62039682)

[Challenges We Faced 5](#_Toc62039683)

[The Future of our Project 6](#_Toc62039684)

[Adding Cryptography 6](#_Toc62039685)

[Expanding Existing Extension 6](#_Toc62039686)

[Conclusion 7](#_Toc62039687)

# Introduction

The name of our project is Signa-Secure. It’s a google chrome extension for Gmail that uses asymmetrical keys to confirm the identity of the sender of a signed email. The sender can use a private key to sign the contents of an email and send it to the receiver. The receiver in turn can use the public key of the sender to confirm that it is indeed an authentic email received from the correct source.

# Project Purpose

The purpose of our project is to allow the users to authenticate the source of their emails. This allows for message integrity, especially when receiving sensitive data.

## Why Choose Us?

Our email authentication project is better than other existing projects of its kind in many ways. Among them, the private key which allows email signing is stored in the flash storage of the DAL-applet, which is a well protected storage area. Other existing solutions store the private key in the regular software or operating system in use.

Another reason why our solution trumps existing solutions is that since we only have the one feature of signing and verifying emails, it’s a very light extension to download and doesn’t come with a bunch of unnecessary files. The limited features in our project make it light, but it can also be a downside. Some users may want an extension that comes with a few other security features that ours doesn’t yet offer.

# Design & Architecture

## Project Flow

3

2

1

Extension injected

Email client

Extension backround

Extension content

9

4

server

8

5

DAL applet

C# - host app

6

7

## Flow Explanation

1. The extension accesses the email body and source email address from the email client, we used an injected script to be able to do this.

2. The injected script then sends the retrieved data to the content scripts. (the injected script can’t send messages directly to the background script)

3. The content script then forwards the data to the background script

4. The background script the accesses the server and forwards the data to be signed

5. The server forwards the data (email address + email body) and sends it to the C# host application

6. The C# host app call the DAL-applet to

a. Return the public key

b. Sign the email body with the private key

7. The DAL-applet

a. Returns the public key connected to the specific users email address, if no public key exists yet, generates keys and saves them in the database.

b. Signs the email body with the private key and returns it to the C# caller.

8. The C# host application returns the signed email body along with the public key to the server.

## How To Use Our Project

How to use project: install extension. run server. Run c#. open email. And click on compose button or respond to an email and click on the relevant button – sign or verify. And watch the magic unfold

In order to use our project you have to first install the extension on your chrome browser. Then to actually start the process you have to run the python server code(server2.py). after the server is started you have to connect both sides to the server:

First connect the Back-end side by running the C# code. After that you hove the server connected to the host and DAL-applet so you can use Gmail as normal. Go to your Gmail account and click compose to start writing an email, do that as you usually do. When you are finished writing you’ll notice that there is another button next to send- which says “sign”. That button helps you sign your email if you would like. After you press sign you can continue on to send and then you are done! You sent your signed email.

In order to verify your email you open the email you wish to verify and then, when you are inside the email in question, click the menu button that is next to the reply button. At this point a verify button should appear on the toolbar. Click it a voila! You verified your email.

# Security analysis

Our security objectives are to protect Gmail users from malicious minded people who try to pretend to be someone they are not through email. They use the fact that sending an email is not face to face as a way to trick someone into thinking they are who they say they are. Once they gain access to someone through email they can request and receive sensitive information that’s not meant for their eyes. We implemented this objective by creating an extension to the Gmail service that allows for a user to sign their emails and in that way, confirm to the receiver of those emails that the source is indeed legitimate. The receiver does this by verifying the email and once its verified, the receiver can be certain of the emails source.

Our extension protects against threats such as phishing emails and such where the attacker tries to use a false email address to gain the trust of others. What it doesn’t protect from is if the attacker gains access to the actual Gmail account of the sender. In that case, at the point our project is up to, we have no way of knowing that the email account has been compromised.

# Mode of Work

While working on this project we found it easier to work together most of the time. We would share the screen from one of our computers and research and code that way. We did end up working on own towards the end of the project.

Some of the back-end with the DAL, C# and server was done by Rivka,

and some of the front-end messaging between files was done by Sara.

## Challenges We Faced

During each step of our project journey we encountered many obstacles and challenges. Starting with the front-end, we had little to no background on the JavaScript language as well as all the add-ons that go with it. We also didn’t really know anything about creating our own chrome extension. Most of these challenges we were able to overcome due to hours upon hours of research and google searches. We tried a lot of different ways of doing the above until we finally got a working project and then built up from there. For the back-end part of the project, we did know what we were doing because it was using the material we learned about in class and we had to alter it to our requirements. It did take a lot of debugging and fixing things up, but ended up working in the way we wanted. The really big challenge for us was the connection between the chrome extension and our C# host app. We tried a lot of ideas that we found in our research but nothing seemed to work. After almost giving up multiple times we finally tried to use a local server to connect the two parts of the project. Adding the server as a component in our project was also a lot of new research and learning how to use it with two programs connecting to it, but that was finally the right idea to give us the connectivity that we needed in order for our project to work as it should.

# The Future of our Project

## Adding Cryptography

Since this chrome extension is security minded, it would be nice to add a feature of encrypting and decrypting emails. That way the email would be even more secure in the fact that if it was intercepted, it would be unreadable. We would have liked to add this feature to our project but due to that lack of time we had to stick to one feature.

## Expanding Existing Extension

Another thing we would have liked to add is extending our extension to work for multiple accounts per computer. Right now our extension is a bit limited and if you try to use another account on the same computer, it will overwrite the first accounts saved data in the DAL-applet and cause the system to have to generate a new set of keys for the previous account.

We would also have liked to add a cool feature that allows you to export your own set of keys (private and public) to another computer and be able to use it form there without having to set up anew and start from scratch.

Right now our project works with multiple components that have to be run separately. In the future we would like to make our project run automatically with just buttons in the Gmail web page.

We would have liked to add a couple of things to make our existing project closer to the one we envisioned, like:

- enhance our Graphic User interface for our extension.

-save all public keys in a file in the C# code so when verifying a signature we can tell it’s the same public key being used for that email address. This is necessary in a case when someone hacks into the users email account and sends emails directly from their account. They can generate their own set of keys and use the private one to sign and send their own public key to verify. In this case the current system will allow it but if the public keys were saved originally then there would be an inconsistency and the verification would fail.

# Conclusion

This project was very exciting and even thrilling at times and we enjoyed making it. Throughout the duration of this project we did come to large bumps in the curvy road and some bumps were larger than other. We tried our best to work through every setback and stay positive even after hours of work had to be deleted with one click. Through it all we persevered and we’re glad that we got to the other side. We want to thank our instructor Barak Einav for being so dedicated to the course and teaching us all about the back-end part of our project.

.

Why? What? How?