Soteria

protect your privacy in a secure, simple,
and anonymous manner

Rotem Hemo, Maria Messick, Adrian Mora

May 2017

Abstract

The number of inspections of electronic devices by the U.S. customs and border protection has spiked over the last three years. While there are official guidelines for the types of travelers to search, travelers are also randomly selected[3] for inspection and no clear rules govern what the customs official can or cannot do. There is a strong need to protect traveler's privacy but there is no solution that enables protection of data in a secure, simple and anonymous manner. We present **Soteria**, a secure, simple and anonymous solution that allows users to protect their data while going through customs. We implement Soteria and deploy it on Heroku[1].

1 Introduction

President Trump recently signed an executive order that would increase the extent of vetting required for travelers to the United States, specifically for those travelers coming from Muslim-majority countries. His executive order has led to increases in customs seizing devices and even going so far as to demand passwords to devices and other personal accounts. Security Secretary John Kelly even warned that he might put in place laws requiring citizens from 7 majority-Muslim countries to hand over their social media passwords or be denied entry to the country [3]. Even before Trump, there had been a huge increase of electronic media searches that went from 4,764 in 2015 to 23,877 in 2016 [2].

Many foreign travelers see these searches as infringing on their rights and want to prevent the United States government (or any other customs agency) from accessing their personal data. Some travelers have been known to take steps to secure their data, such as keeping devices with only minimal data, but these methods are known to be inconvenient. Our project aims to help the average person (who has minimal knowledge of device security) keep the information

[1] As for May, 2017, Soteria is deployed on Heroku - `https://soteria-project.herokuapp.com`. Soteria's source code is available on GitHub - `https://github.com/rotemh/soteria`

on his/her devices protected from customs agents in a convenient and secure manner.

2 Related Work

There are other services in the market that address some of the concerns that this project also seeks to address. Notably, there are are projects that will encrypt all of your data, programs that allow a user to remotely destroy his/her data, and platforms that allow the user to store sensitive information remotely, on the cloud. However, none of the related work addresses all of these concerns in a single platform.

2.1 Whole Disk Encryption

There are several platforms that allow a user to encrypt all of his/her files. These include but are not limited to

- **BitLocker:** This program ships with Windows operating systems, and performs full-disk encryption.
- **TrueCrypt:** TrueCrypt is open-source, which makes it a more accessible program. However, there are concerns about its current security; it may not be as secure as it once was, and many users are moving away from this platform.
- **PGP:** A user can deploy PGP to encrypt texts, emails, files, and whole disk partitions.

2.2 Remote Data Destruction

These tools allow you to destroy sensitive documents from a remote location. This is useful if a third party has access to your device, and you want to prevent them from accessing any information.

- **xTool Remote Delete:** One of the programs offered by xTool Mobile Security. Enables customers to remotely delete data and files.
- **LoJack for Laptops:** Formerly known as computrace. This is a laptop theft recovery software that provides users the ability to remotely lock, delete files on, and locate the missing or stolen device.

2.3 Cloud Storage

A cloud storage service allows a user to store files and data in a remote location, independent from the user's physical device. This is useful if a user anticipates that his/her device may be taken from him/her for any reason, but wants to maintain a backup independent of the device.

- **Google Drive:** One of the most popular cloud storage solutions. Allows a user to store up to 15GB of any type of file or data for free.
- **Dropbox:** Another very popular cloud storage solution. Same main features as other providers.

3 Assumptions

There are several assumptions we made about what customs officials can and can't do at the border that we took into account when designing our security system. We have listed them below:

- Laws exist and are enforced that prevent customs from detaining a person for more than a certain period of time
- Customs only has access to the items you carry on your person
- Customs may ask for your mobile phone and computer passwords
- Customs is less likely to seize your device if you grant them access
- Suspicious files and locked folders are likely to raise red flags for customs agents
- Customs may download all of your data on the spot. Following this, they have the ability to run a forensic search on your data. They are supposed to delete it after 21 days, but this is not guaranteed [1].
- Customs is required to return a seized device after 5 days. However, they have the ability to extend this for 7 days at a time indefinitely [1].
- If a foreign citizen declines compliance with customs, they may be barred from entering the country [1].

4 Threat Model

In our design, we assume the server is malicious and very vulnerable. We assume that a country like the U.S. is able to issue a subpoena to get files from the server. Also, we assume that both an attacker and the owner of the server can get files from the server.

Users can be one of two types, a normal, honest user and a malicious one. We assume the average honest user has a strong incentive to follow the instructions very carefully. Any mistake may cause a definite lose of his/her files. A malicious user would try to gain knowledge about files on the server using the client. We assume polynomial computation time for all possible adversaries.

5 Design Goals

In a recent Wired article, Andy Greenberg explained how to get through customs without allowing the customs officers to get access to one's private data. His proposed ideas are complicated and often prevented the traveler from having access to his data for a long period of time. We propose a new mechanism that prevents customs from accessing private data with minimum burden on the traveler. The three most important aspects to a system the average traveler would use for keeping their data private are usability, security, and anonymity. We decided to prioritize these when designing our system.

5.1 Usability

The average traveler is more likely to have his or her personal device seized or searched than ever before. We focused on protecting the average travelers because they are the least likely to be familiar with methods to encrypt and protect their data. Because they are not familiar with encryption methods, the average user will only use a system that they can easily use. We wanted to make sure these kinds of users will actually use our system, so we prioritized usability when designing our system.

5.2 Security

The security of our users' data is top priority. The system needs to ensure and to protect the privacy of traveler's private information from customs. Because customs seizes devices frequently, we must design our encryption scheme so that it is impervious to an attack by customs officials. In particular, we have the following goals:

- Gaining access to the server should not reveal any information about users or their files.
- Encryption keys (asymmetric or symmetric) should not be accessible nor stored anywhere.
- Obtaining the software source code or access to a user's computer should not enable access to the user's protected data.
- There should not be any evidence of using the software.

5.3 Anonymity

Many of our assumptions describe situations in which a customs official might become suspicious of a traveler and seize their devices or download their data. Thus, when a user encrypts their files using our system, we want to make sure the other users in the system, as well as anyone on the server, do not learn his/her identity.

6 Soteria System

Soteria takes a user's files, encrypts them and sends them to a server to be stored. It then deletes the files and replaces them with fake files. Finally, it deletes itself from the user's computer. Soteria decrypts the user's files once he or she is ready to re-download them. The overall design of Soteria is demonstrated in the figure below.

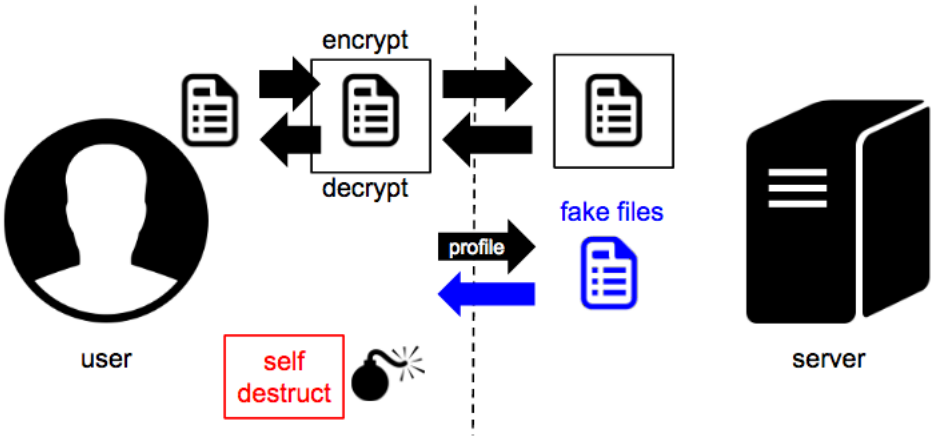

Figure 1: Soteria System Design

6.1 Client

The client side of the system is the software that runs on the client's machine and interacts with the server. It includes three main parts: cryptography, fake files, and self destruct.

6.1.1 Cryptography

All encryption and decryption takes place on the client side. By keeping it local, the server does not even see the keys, much less store them. This design choice protects from adversaries looking to break into the system through the server side.

Key Generation
Passwords provide low security. Since good passwords, which contain not human friendly combinations, are hard to memorize, most people use very simple ones. Recent analysis of over 10M passwords shows that the most common password is still 123456 [5]. On the other hand, encryption keys provide the highest

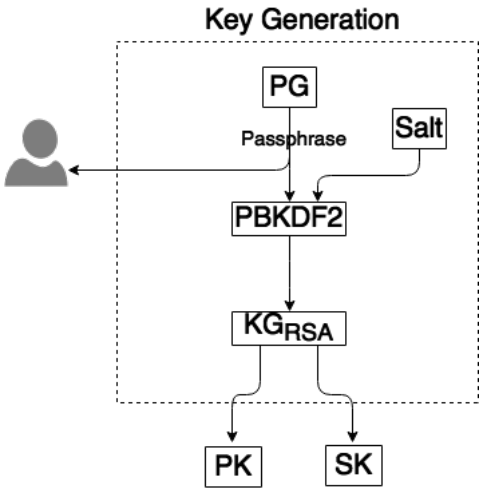

Figure 2: Key Generation process. We first generate a random passphrase, then we extract a unique random key using a PBKDF2 Key dervivation function according to the passphrase. Lastly, we seed the RSA key generator with the unique key and generate a pair of public and private key that are unique to the passphrase.

security, but are complex to the average user. In Soteria, we decide to find the middle ground and generate our RSA keys out of passphrases. A passphrase is a sequence of n English words that are used as a password. It has two main advantages over regular password. The first, it has far more entropy and therefore more secure. The second, it has no unnatural combination of characters and symbols and is therefore easier to memorize.
In order to generate a pair of keys (PK, SK), we generate a random passphrase, seed it to a RSA key generator through key derivation function and get a key pair that is unique to the passphrase (See figure 6.1.1). The generation happens "on-the-fly" and the keys are never stored on hard-drive.

Passphrase generation: We generate a passphrase by selecting 12 words with length of 5-8 charcters uniformly at random from a collection of 65,000 English words and phrases. The length of the passphrase is configurable. We decided to use 12 words because it gives entropy of $\log_2(65000^{12}) = 191$ bits, which is sufficiently high enough to guarantee a high level of security.

Key Derivation: In order to seed the RSA key generator, we use the key derivation function - PBDFK2. Key derivation function derives a unique psuedorandom key from an arbitrary input such as a passphrase and a salt. We chose to use a KDF to add another layer of security. KDF

makes it harder to enumerate over all passphrase. First by adding salt and second, by being inherently (and intentionally) slow function so attackers cannot enumerate over all possibilities fast enough.

Encryption

Asymmetric encryption schemes, such as RSA, have two main properties that do not allow the encryption of large data. The first, since we mod every operation, asymmetric encryption schemes cannot encrypt any arbitrary amount of data. The second, asymmetric encryption schemes are substantially slower than symmetric encryption schemes. To get the benefits of the two schemes, we combine them. We encrypt the user files using an AES block cipher and encrypt the AES symmetric key using the RSA key pair. (See Figure 6.1.1)

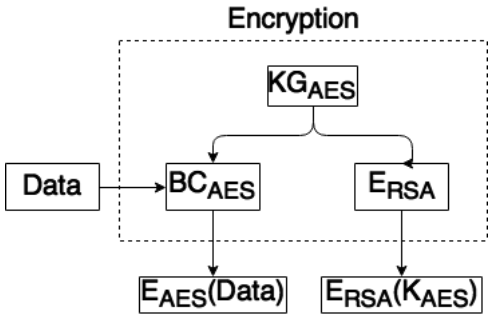

Figure 3: Encryption process. We generate a fresh AES key and encrypt the data using an AES Block cipher in GCM-Mode. Then, we encrypt the AES key using the RSA encryption key.

AES Block cipher: On every new encryption, we generate a new AES symmetric key. We then use AES Block cipher in GCM mode to encrypt the data.

RSA: After encrypting the data, we use the RSA public key PK to encrypt the symmetric key.

Decryption

In order to decrypt the data, we generate the same pair of RSA keys from the user's passphrase. Then, we decrypt the symmetric key and using it to decrypt the rest of the data.

6.1.2 Fake Files

The user inputs several pieces of information in order to create a profile. The profile is used to generate a set of fake files that will replace the files the user wishes to encrypt and remove from his or her computer. The fake files are generated in order to avoid suspicion of an empty folder from a customs officer looking through a user's computer.

6.1.3 Self Destruct

Soteria deletes itself from the user's computer so as not to raise suspicion at customs. If a customs officer saw that a user was using encryption software, he or she might be more likely to confiscate the user's personal device.

6.2 Server

The server is in charge of storing the client's encrypted files. When it receives the encrypted files from the client, it stores them in a look up table. The index of the look up table is hash of the first two words in the client's passphrase. There is a low probability of collision because each passphrase is generated randomly on a collection of 65,000 English words and phrases. This means that probability of collision is $\left(\frac{1}{65,000}\right)^2 = 2.37 \times 10^{-10}$–negligible. Although it uses a part of the secret passphrase, it does not reveal enough of the passphrase to help an adversary. This index scheme also means the files can be retrieved easily when the client enters the passphrase to the server and does not require additional input. By storing the files only using part of the passphrase, the client is able to retain anonymity on the server because the look up is never based on the client's identity.

7 Evaluation

In order to evaluate our system, we revisit our deign goals.

7.1 Usability

- Our design is, indeed, simple. the design doesn't require any knowledge of cryptography from the user or any other technical knowledge. The user only needs to specify a profile and the folder he/she wants to protect. The user must also write down his/her passphrase in order to later retrieve his/her files. All the rest is done automatically.

7.2 Security

- Gaining access to the server should not reveal any information about users or their files.
  We store only encrypted files on the server. That way, even if the server is compromised, no data is readable. With that, the hash is SHA256 of the first two words in the passphrase. Even if an adversary would enumerate all 65000^2 different possibilities, it reveals just the first 2 out 12 words. which has still sufficient entropy.

- Encryption keys (asymmetric or symmetric) should not be accessible nor stored anywhere.

All keys are being generated on-the-fly and not stored anywhere. The only key that is being stored is the symmetric key but this one is being stored encrypted with the public key on the server.

- Obtaining the software source code or access to a user's computer should not enable access to the user's protected data.
  Since everything is encrypted and nothing (including the encrypted data) is not stored on the user's computer, knowing the source code or having access to the user's computer doesn't reveal any data.

- There should not be any evidence of using the software.
  By the end of every run, the program completely removing itself from the system.

7.3 Anonymity

In order to keep the data anonymous, we do not store any user's information on the server. All the private data, including the user's profile is never being stored on the server.
In the current implementation, we do send the dummy files unencrypted over the Internet, which expose the user's (chosen) profession and (chosen) sex. With that, there is always an option to let the client and the server agree on a symmetric key using DH protocol and send these files encrypted using a symmetric key encryption scheme (Such as AES).

7.4 Implementation

We implement a prototype of Soteria in Python 2.7.11, consisting of approximately 400 lines of code. We use *Crypto* library for the cryptography primitives (RSA for asymmetric encryption, AES for Symmetric, PBDFK2 for key derivation function, and HASH256 as hash function), *Flask* library for server side, and *xkcdpass* library for generating passphrases.

7.5 Client side

Upon loading, Soteria requires the user's profile, target folder to protect, and whether the user wants to encrypt or decrypt the data. After getting the information, Soteria generates a 12-word passphrase, where every word has between 5-8 characters, and presents it to the user. Then, a 128-bit key is derived using a 1000 rounds PBKDF2 with the passphrase and 16 byte salt. We seed the RSA key generator with the 128 bit and generate 2048bit key asymmetric key pair. In addition, Soteria generates a 256-bit long AES symmetric key.

Then, Soteria zips the user's data, encrypts it a AES block cipher in GCM mode and encrypts the AES key with the public key. Once, the system finishes to encrypt the data, it hashes $H_{data} = H(passphrase[1,2])$, the first two words in the passphrase and submits the tuple $(H_data, Enc(data), salt$ to the server.

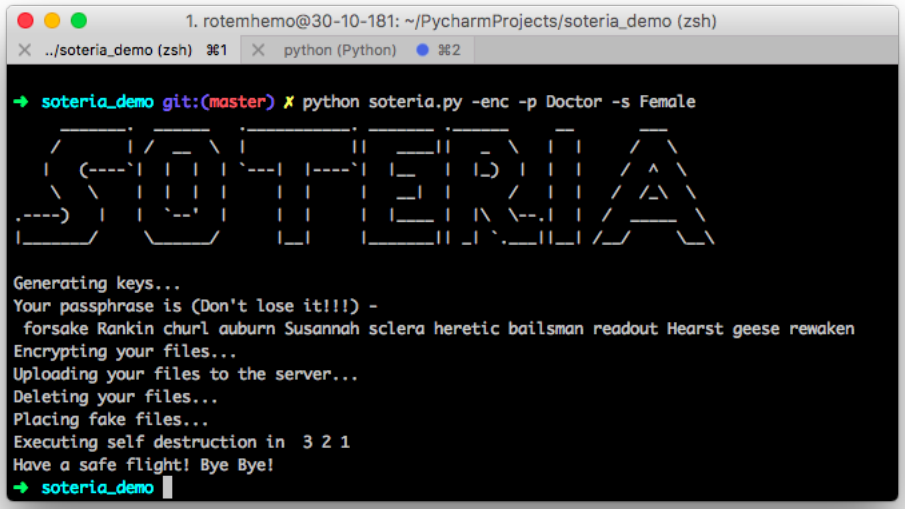

Figure 4: Soteria user interface

After submitting the data, the systems removes the zipped files and sends a GET request to the server for the relevant fake files, downloads them and puts them in the protected folder.

Finally, the system injects a command into an external process and the latter removes all Soteria files.

The user never interfaces with any of the steps above, except providing the information about his/her profile and the command to encrypt/decrypt. Thus, after providing a one line command, the user can sit back and let Soteria do the rest.

7.6 Server

We keep the server implementation for the prototype simple. The encrypted data is held in a Map $< H_data, (Enc(data), salt) >$ and the interaction with the user happens with with GET and POST commands. The connection with the user isn't encrypted because any private information never leaves the user's computer unencrypted.

8 Future Work

Ideally, we would have no shortage of time to implement all the features that could truly make this a great platform. We consider some of the main improvements that could be made in the future.

- **GUI:** Right now, the program works entirely from the command line, and does not provide the user any kind of GUI. Ideally, we would have a program that walks the user through the usage of the program with a well-designed interface. A GUI would help the user understand what the program is capable of, and make it easier to use.

- **Cross platform support:** Since the goal is to protect a user's information, we would want to protect this information on any platform that a user keeps their documents, including mobile phones and tablets.

- **Smarter fake files generator:** We currently support four different "profiles" of dummy files to present to the user. Since each individual person has a unique background, there should be more dummy content to draw from so as to construct well-tailored (Using machine learning for example) dummy data for each person. This aids the user in avoiding suspicion. If a user's devices are searched by a customs agent, a better matching dummy data profile will stand out less.

9 Conclusion

Data security is an issue that is becoming a big concern to the average person. As we rely more on digital devices to function in today's society, most of the important information in our lives lives on our portable digital devices.

Many people do not want to give customs official access to their personal data, even when asked. To date, there does not appear to exist a platform that is easy for the average person to use and can keep this data safe while not arousing suspicion from customs officials who request access to devices.
Soteria addresses this gap in the market for personal data security. Using Soteria, a user can safely protect all of his/her sensitive documents while going through customs.

Acknowledgments

We would like to thank Professor Rivest, Professor Kalai, and all of the TAs for their advice throughout the project.

References

[1] Esha Bhandari, Nathan Freed Wessler, and Noa Yachot. *Can Border Agents Search Your Electronic Devices? It's Complicated.. (English)*. 2017. URL: https://www.aclu.org/blog/free-future/can-border-agents-search-your-electronic-devices-its-complicated.

[2] Sophia Cope et al. *Digital Privacy at the U.S. Border: Protecting the Data On Your Devices and In the Cloud*. 2017. URL: https://www.eff.org/wp/digital-privacy-us-border-2017.

[3] U.S. Customs and Border Protection. *Inspection of Electronic Device*. 2017. URL: https://www.cbp.gov/sites/default/files/documents/inspection-electronic-devices-tearsheet.pdf.

[4] Andy Greenberg. *A Guide to Getting Past Customs With Your Digital Privacy Intact*. 2017. URL: https://www.wired.com/2017/02/guide-getting-past-customs-digital-privacy-intact/.

[5] Keeper Security. *The Most Common Passwords of 2016*. 2016. URL: https://keepersecurity.com/public/Most-Common-Passwords-of-2016-Keeper-Security-Study.pdf.