UPPSALA
UNIVERSITET

IT 15 011
Examensarbete 15 hp
Feb 2015

KeySafe

The platform-independent password safe with external security

Olof Björklund

Institutionen för informationsteknologi
Department of Information Technology

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

KeySafe

Olof Björklund

Storing and accessing sensitive data has become an important task in today's society. Many different services require login credentials for users to remember in order to authenticate themselves. A common habit is to use the same password for several services. This is considered a security risk since if someone uncovers the user's password they will gain access to all of the different accounts using the same password. Hence users are encouraged to use different login credentials for different services, resulting in an increasingly large list of sensitive data the user needs to remember.
KeySafe provides a password storage service which makes all your digital keys available with the use of physical one. In this project, a platform independent service has been created with an Android application which implements external authentication using NFC tags. Using Google App Engine with the Endpoints API backend the service becomes available to a range of different devices such as iPhone, PC, Tablet or Mac. This resulted in a flexible, secure system that makes it an easy task to use strong and independent passwords for different login-services. All data stored by the application is encrypted using the AES encryption cipher and the AES key needed for decryption is stored on the external NFC tag.

Handledare: Johan Risch
Ämnesgranskare: Johannes Borgström
Examinator: Olle Gällmo
IT 15 011
Tryckt av: Reprocentralen ITC

Table of Contents

1. Introduction

The Internet grows larger and larger for each day. In the year 2010 there were 12.5 billion machines connected to the Internet [1] and the number keeps getting bigger. Something that has always been of great importance is the aspect of security and preserving integrity. For protection against viruses or harmful software certain anti-virus or anti-malware programs can be used. For protecting personal data online a user login system with password and username is a well spread method.

In general a user is discouraged from using the same login credentials for different services as this would pose a security risk. This quickly mounts a lot of information the user needs to remember in order to access her different services and can sometimes be a cumbersome task, who has never forgotten their password or username at some point?

A solution for this problem is to use an application to store and access this information.

1.1 The KeySafe project

This bachelor thesis is made for the software-company *Innocreate AB*[1] which is located in the city of Uppsala, Sweden. The mission of this project is to create a platform-independent password-storage service.

There are today many different password-storing programs in service. Some of them also implement platform independence. However, a common concern amongst users is the risk of getting ones password to the password-storage application stolen. If this happens the perpetrator would get access to all the sensitive data stored on the system.

What KeySafe aims to do is create an application that deals with storing sensitive data in a platform-independent way with external security. External security will be implemented using an access token which acts as a physical key to the digital safe making it inaccessible without possession of the physical token.

[1] www.innocreate.se innocreate AB (active 2015)

The application will provide:

- Platform-independent access
- User centred design
- External authentication with physical token

The following questions are investigated by the project:

- How can platform independence be implemented in an application?
- In what way should data be encrypted for maximal security?
- How can external authentication be made for smartphones?

1.2 Delimitation and methods

Since this project is done by a single student with limited knowledge in computer/network security and mobile application development, only a prototype for the application will be made. A web server will be implemented and communications to a Java-based client established. For the smartphone a *proof-of-concept* will be created showing how the finished application could work. The prototype should be functional in storing information in a secure way on a webserver using data encryption.

For the development of the project the IDE Eclipse will be used alongside Maven. Eclipse provides a good overview perspective as well as syntax-help and auto code-completion. Maven is mainly used for its ability to generate discovery documents and endpoint libraries to be used in the Google App Engine.

There is an SDK downloadable for free from the Google App Engine homepage (link in references) along with a neat plugin for Eclipse IDE.

2. Background

This section first lists abbreviations used in the report and then explains some of the more important background technologies and concepts used by the project. Second follows a look at related work and a peek at a problem that existing work is facing today.

2.1 Related Work

There are many different applications for storing passwords, user credentials and other sensitive data available to the public. To name a few examples: 1Password, Password Genie, mSecure and Keeper. All of these have one thing in common, one password to access all your passwords. Out of the examples mentioned *Keeper* [2] is in good relation to this project, since it offers platform independence and uses AES encryption. Keeper's smartphone application comes free of charge, with Android support and a web application interface for managing stored data making it suitable for comparison to KeySafe. In Keeper, the user can store passwords locally on the current device, or on a server using cloud services. There is one extra security measure, apart from the single master-password, which is called Two-factor authentication [3]. Two-factor authentication like the name implies requires the user to authenticate using two factors. In many applications, such as Keeper, the first factor is the master password and the second factor can be chosen out of three possible options:

- *Text Message*
  The application sends a SMS text-message to a user specified number that displays a short number code for authentication.
- *Voice Call*
  The application makes a phone call to a user specified number with an automatic voice-machine that tells you a number code for authentication.
- *Google Authenticator*
  The application displays a QR code that must be scanned with the Google Authenticator Application [4] which then displays a number code for authentication.

2.2 The problem with two-factor authentication

Two-factor authentication is a widely spread method for providing extra security. The method is used by many big companies such as Google, Microsoft and Apple. However, as hackers and cyber-attacks grow more sophisticated the

two-factor authentication could be circumvented, for example with the use of an in-system "spy-program" sending the authorization code to an outside source as it is received. A program of this type is referred to as *malware* and could find its way into your smartphone through a corrupted website or a compromised app.

A study made 2012 by *Eran Kalige* and *Darrell Burkey* for *Versafe* and *Checkpoint Software Technologies* [5] details a Trojan virus called "Eurograbber" which successfully stole over 36 million Euros from more than 30.000 bank customers with the use of smartphone-malware.

It is clear that two-factor authentication cannot guarantee safety. To quote the study:

"Enterprises as well as individuals need to exercise due care and ensure they conduct important online business, especially financial transactions in the most secure environments possible. (…) A computer that is current in OS and application updates and security protections combined with an office network that is protected with multiple layers of security will provide the most protection against attacks like Eurograbber."

2.3 Improvements

The KeySafe system would not be immune to malicious software or malware present on the system in use, but would provide an extra layer of security by the use of external authentication. Even if malware is present on the device, any attempts of decrypting stored data would require the NFC tag to be in contact. Since this only happens when the user is interacting with the system the user is given the opportunity to scan the device before each chance of data decryption.

Combining two-factor authentications with the external authentication of KeySafe would make it even more difficult to retrieve decrypted data as an unauthorized user. Although two-factor authentication will not be implemented for the prototype of the KeySafe system it could be done using one of the methods mentioned in section 2.2.

3. Project theory

This section presents the theory for the KeySafe project. First we look at the possibilities of external authentication, followed by theory on authentication implementation. The project aims to create a simple, yet secure way of managing sensitive data and access. Security and privacy will be provided by three-layer user verification:

1. External authentication
2. User authentication
3. Password

The external authentication method, i.e. the first layer, will be the most important one for this project since it distinguishes the project from similar applications already in service.

3.1 External authentication

There are several different methods of external authentication available today. One of these is the number authenticator, or disconnected security token.

These tokens typically have a built-in screen to display a generated authentication code that the user enters into the system manually. For security reasons a generated code typically has a life-span of about one minute before it becomes invalid and a new code must be generated. It is also common that a user defined PIN-code must be entered on the token before the authentication-code is displayed. An example of such a device is the RSA SecurID hardware authenticators [6].

Another method for external authentication is the plug-in token. Plug-in tokens must be physically connected to the system before authentication can be made. A token of this type automatically transmits the authentication information to the client once a physical connection has been established. The most common types of physical tokens are smart cards and USB tokens, which require a smart card reader or a USB port respectively.

By implementing external authentication any user attempting to gain access needs to be in possession of the external key before granted such. A useful technology for reading data from an external token is NFC.

3.1.1 NFC

Using a key card, or NFC chip, for identification could be implemented by setting such a token to contain a secret key. For shared access of sensitive data multiple keys could be configured to unlock the same data.

The token needs to be read somehow. A good method to do this is with the use of smartphones, some of which can be configured to read a token using *near field communication*, NFC [7]. NFC is a set of short-range wireless technologies, typically requiring a distance of 4cm or less to initiate connection. NFC makes it possible to share a small load of data between a key card and a smartphone.

NFC uses the same radio frequency which was originally used by Radio Frequency Identification, RFID [8]. RFID is capable of reception and transmission beyond a few meters while NFC is restricted to within very close proximity which makes it more secure.

When a smartphone is used to interact with a smart objects/key card in their environment an integration technique is used, which is the communication paradigm between the smart object and the user.

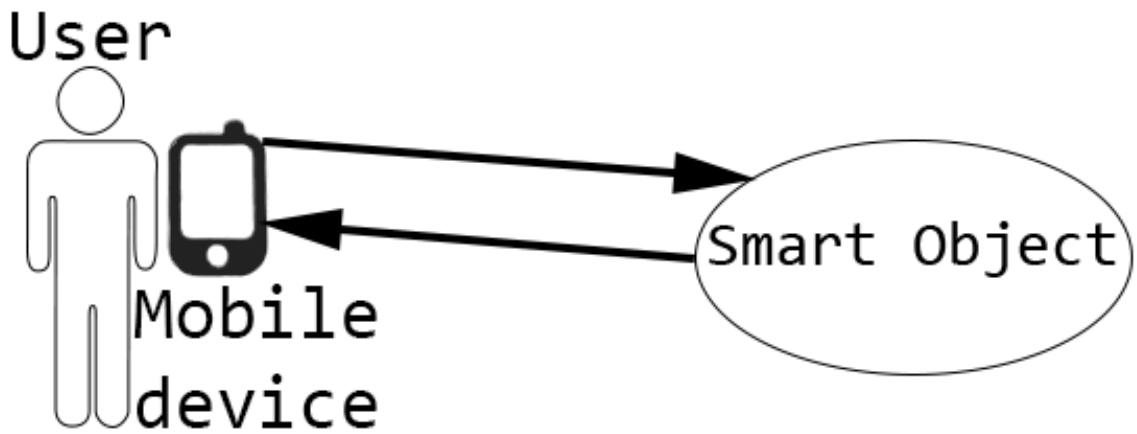

Figure 1 – Mobile interaction with NFC [9]

There are three interaction techniques in which a mobile device can interact with a smart object:

- Touching
  Touching the mobile device with the smart object, or having it within a very close range, typically between 0 and 10 cm.

- Pointing
  Pointing the device towards the smart object. Reading a QR-code is one example of this technique.

- Scanning
  Discover a smart object using wireless scanning of environment, for example Bluetooth technology.

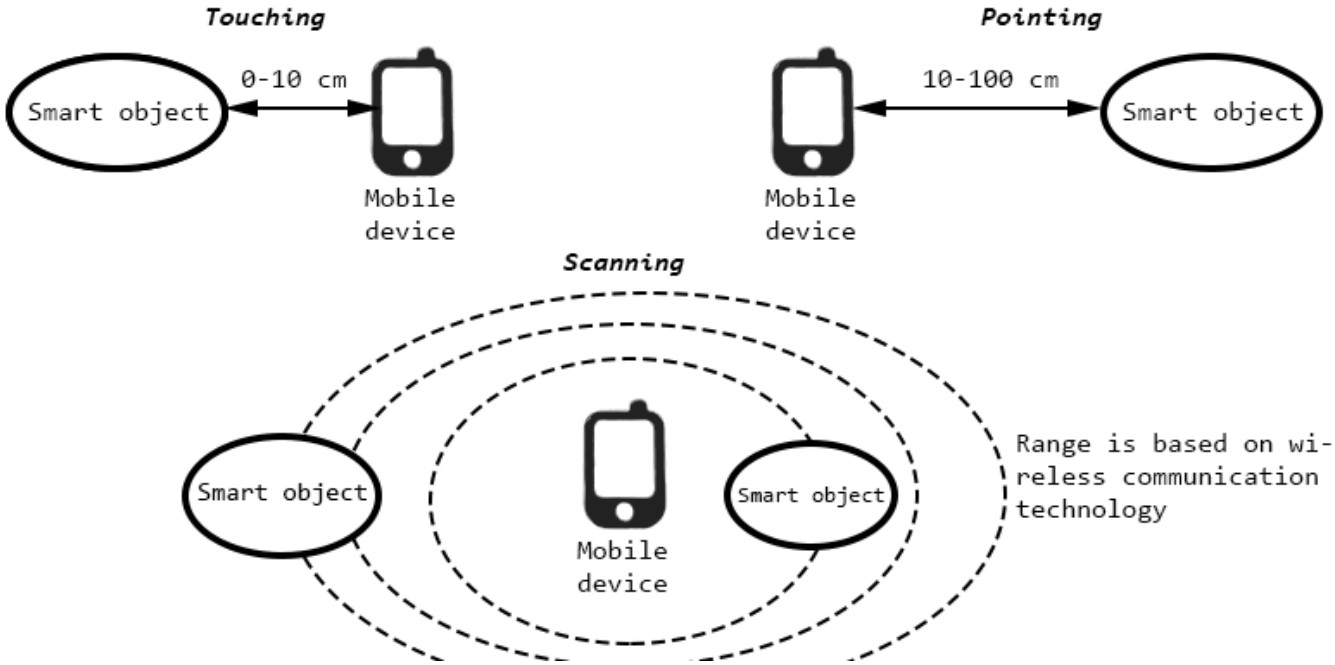

Figure 2 - Mobile Interaction-techniques with NFC [10]

3.1.2 Touch technique

For this project the method of NFC-authentication with touch interaction technique will be implemented for external authentication. It appears as the most user-friendly method and limits the risk of a *"man in the middle attack"* [11] if the authentication is done client-side. For NFC-authentication to work

with Touch-interaction the device must support the NFC framework. Today there are several operating systems which support NFC. Some examples include Android, BlackBerry and Windows Phone. Because of this, and availability, an Android smartphone will be used for the project prototype along with a NFC NTAG203 which can be bought at many computer/technology stores. The NTAG203 offers 144 bytes of on-chip memory and can be bought for around 10$. The technology was developed from the ISO/IEC 1443 proximity-card standard [12].

Some devices do not currently support NFC, for example iPhone. For these a Scanning-interaction option should be implemented, such as Bluetooth. However, this would require a different token then the NFC tag mentioned.

3.2 User authentication

The second layer of security consists of user authentication. Typically this is done with a login-procedure. In the KeySafe prototype user authentication is achieved by users authorizing the use of Google account for identification. This requires the user to have access to a Google account to be used with the system.

3.3 Password

The third and final layer of security is the master password for authentication. This layer consists of a user-defined password, which should be enforced to meet a standard *password policy*:

Password policy

- A password should be at least 15 characters long.
- A password should use both upper- and lower-case letters.
- A password should include at least one numerical character.
- A password should include at least one special character. (e.g. #,@,$ etc.)
- A password should not include any known phrase or word.
- A password should not match the format of a calendar date, license plate number, telephone number or other common numbers.

The two latter will not be enforced by the application, neither will the recommended length of 15 characters, mainly to simplify the password-authentication for smartphone users. These policies will be presented as recommendations to the user when choosing a password.

Furthermore, a user should be advised with the following *guidelines*:

- Never give your password to anyone, ever.
- Never use the same password for more than one account.
- Never write a password down.
- Never communicate a password via telephone, e-mail or instant messaging.
- Change password immediately if there is suspicion it may have been compromised.

The master password of the application would be used as key-seed for the AES cipher key needed for encryption and decryption of data. For maximal security the master password could be automatically generated by the application and then stored on a NFC tag, which in turn could be encrypted in order to prevent any other NFC reader from discovering the AES key. Figure 3 below illustrates the security concept of the KeySafe system with external authentication.

One way of encrypting the AES key on the NFC tag is to use asymmetric cryptography or RSA encryption [13]. RSA is implemented by generating a public/private key-pair. The private key is stored on the device on which it was generated and the public key on the NFC tag. In order to read data from the NFC tag the connecting device must have the corresponding private key of the public key stored on the tag.

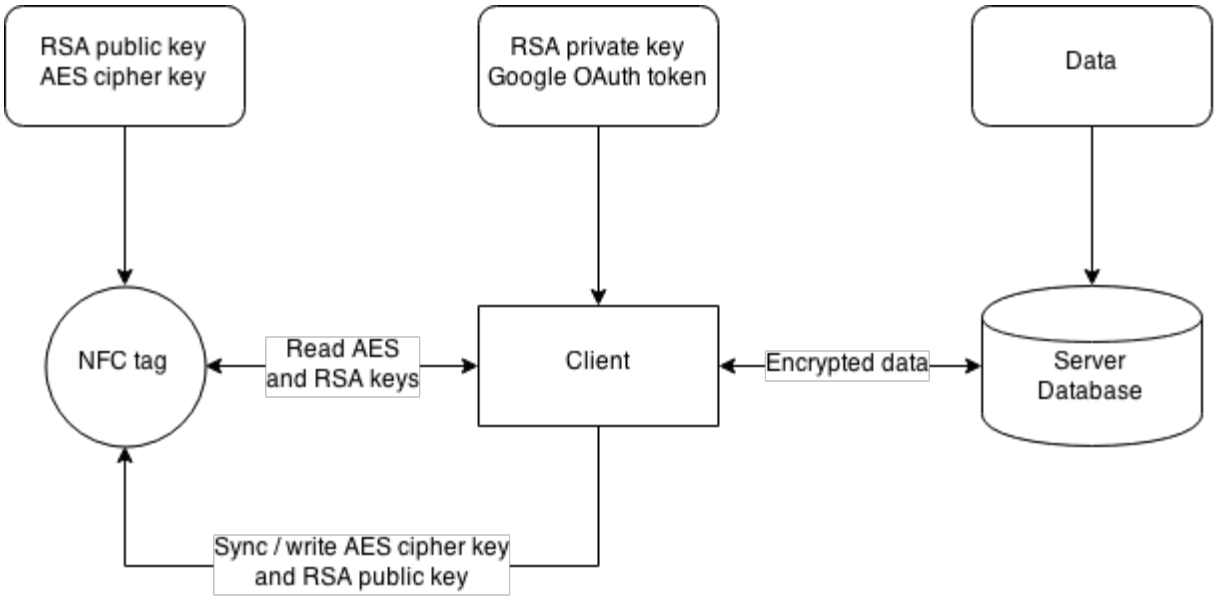

Figure 3 – KeySafe security diagram

3.4 Extra security

Additionally to the authentication system and data encryption extra security could be provided by only allowing access to certain high-value, locally stored, data if there is no network access active on a device. By doing so, it would in theory be impossible for an outside source to communicate with the device and steal the information. This would require Bluetooth, Internet connection and other network-communications to shut down temporarily before accessing the data but in return the data is shaded from network communications. This method would only work with locally stored data.

3.5 Platform Independence

Platform-independency, for this project, means that data stored on the server should be accessible from a range of devices, and not only those of the same platform as the server itself.

Google App Engine offers platform independency in a suitable way for the project. In GAE you can write applications in Java, Python, PHP or Go which can be run with separate runtimes in the Software Development Kit.

By the use of endpoint APIs connections can be established for a range of different devices and services. Endpoints can be generated for each of the connecting systems dynamically without alternations to the client or server application.

4. The KeySafe system

The KeySafe system relies on HTTPS for communication between client, endpoint and server. Google App Engine, or GAE for short, supports secure connections via HTTPS for custom domains.

GAE was chosen for backend communications because this was specified as a requirement by Innocreate. GAE is free of charge and can be used with somewhat limited resources without setting up a prescription plan for payment. For this project the free version of GAE will provide all the backend services and resources necessary.

Furthermore, Innocreate wants to evaluate GAE's technical aspects for potential future use with other applications.

As shown in figure 3 below, the KeySafe system consists of three major parts – an App engine server, Endpoints API and the different clients.

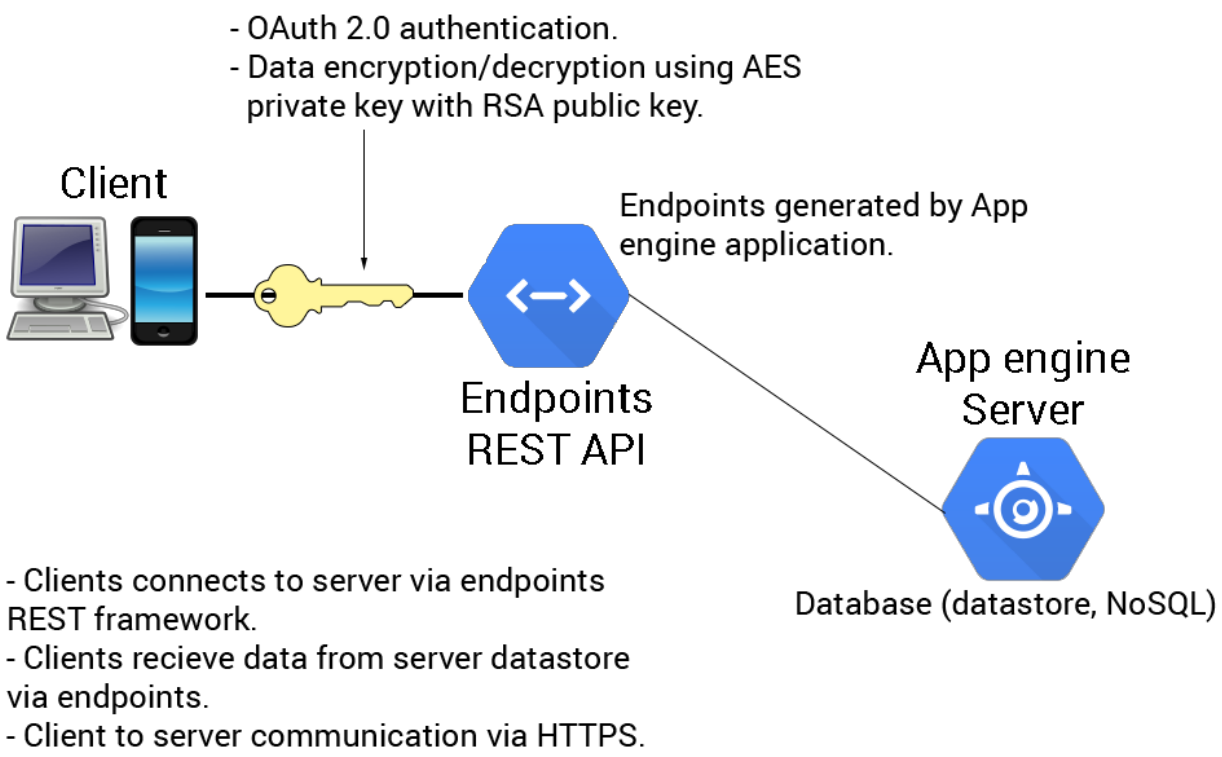

Figure 4 - System architecture

There are several security objects used by the KeySafe system. These are the OAuth access token, AES cipher key and RSA cipher key-pair. The figure below illustrates these objects and their use.

The AES cipher key is used for decryption and encryption of data. If a NFC tag is used the AES key is stored on the tag, otherwise it is stored locally.

The RSA cipher key pair is split up so that the private key of the device is stored locally, and the public key is stored to corresponding NFC tag.

The OAuth access and refresh tokens are stored locally and can be manually deleted by the user. The OAuth token is used for authentication and getting the users email address which is needed for datastore access.

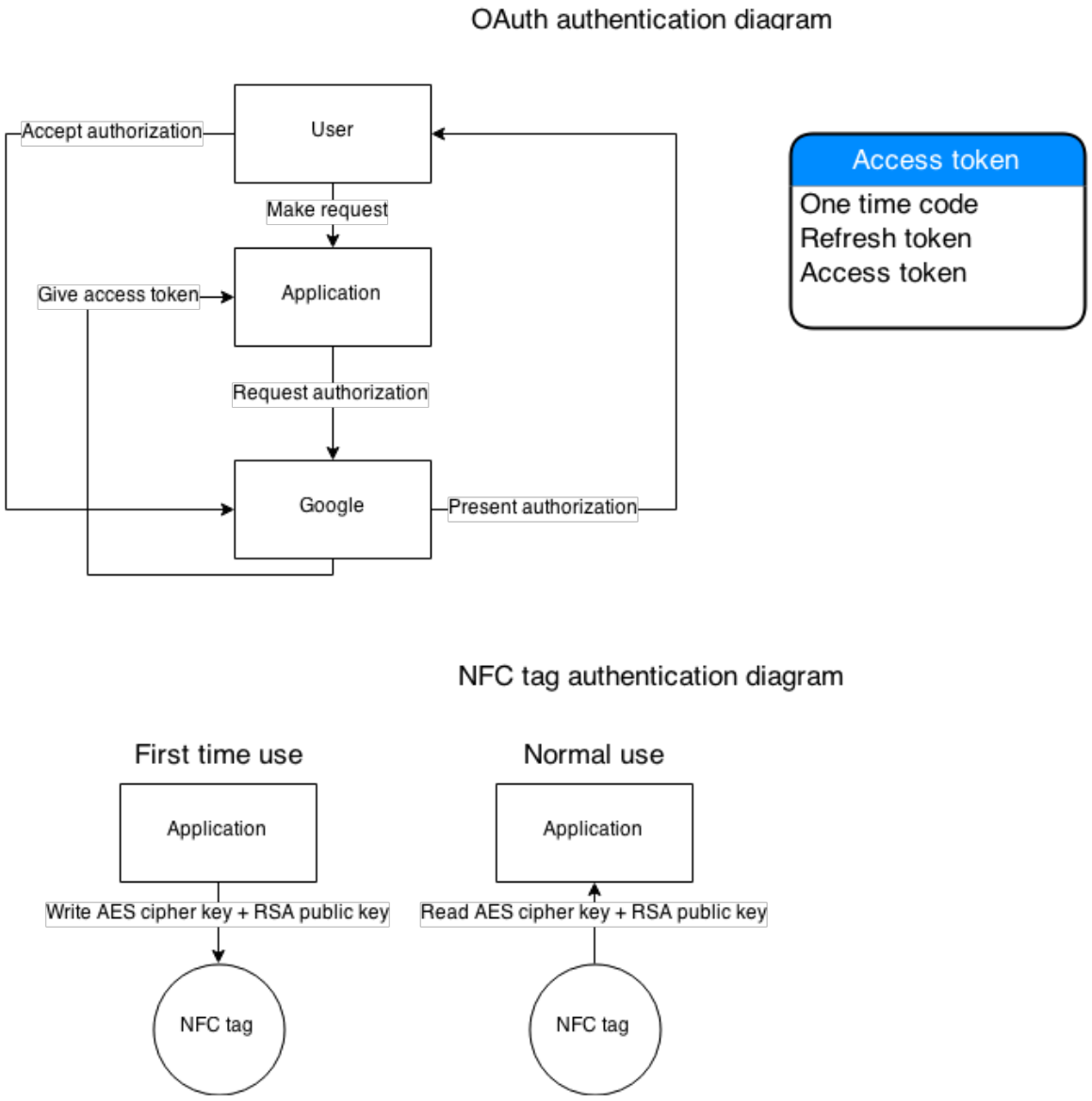

Figure 5 – System security diagram

The KeySafe system has the following workflow: Firstly, the client looks for and connects to a running app engine server using the client libraries and generated endpoints. Once connection is established the user is evaluated by the system. Once the user has been authenticated the application allows for a view of the available accounts. In order to edit, create or view the sensitive data of an account the user must provide the corresponding NFC tag or master password. Once the password or NFC tag is evaluated, if successful, datastore actions are granted and the user can now store, delete or get data. The workflow can be seen in figure 4 below.

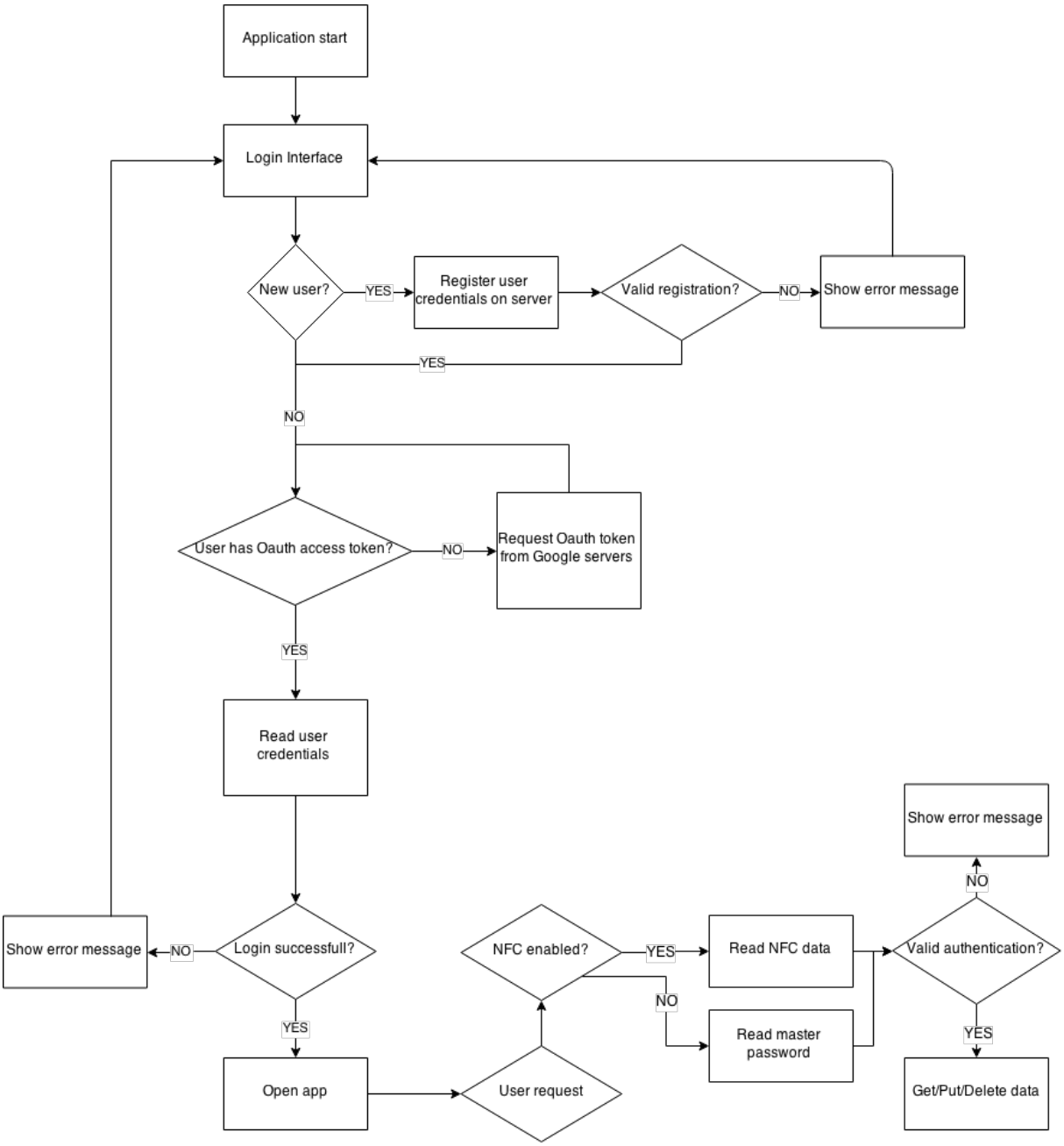

Figure 6 - A flowchart showing the login process

4.1 Authentication

Authentication in the KeySafe system is done by the use of account login. A user could log in to the systems own login system with username and password, or chose to use alternative authentication such as Google+ or Facebook. Instead of a master password a NFC tag could be used to unlock the account and make data accessible.

For the prototype of this project the user needs to provide a Google account and a NFC tag for authentication.

For the external authentication NFC tags are used. The standard Android API comes with a framework for handling NFC read and writes based around a NFC Forum [14] called NDEF (NFC Data Exchange Format). For this application only the read/write mode will be enabled and requested from the smartphone.

The amount of data written and read from the NFC chip will be small to minimize the interaction time. For external authentication only the master password will be read from the NFC along with the public key.

As mentioned in section 3.1.2 above the NFC chip used for this project is the NTAG203 that has on-chip memory of 144 bytes. This is with certainty sufficient memory to store a password and associated key pair. There are other NFC tags available on the market with higher security and more memory then the NTAG203. But for user availability (and price-related reasons) the NTAG203 was chosen for this project.

For convenience, a greater user experience and increased security the user will be given the option to use a NFC tag login-method. To login and authenticate the user would simply swipe a pre-configured NFC tag instead of manually typing the master password. This would completely remove the factor of having to remember a password to access the application since it would be stored on the NFC, to be retrieved and deciphered only locally on the smartphone. With this method the master password could be composed in a complicated manner without risking the user forgetting it or having to write it down. Furthermore using RSA encryption as mentioned in 3.2 above we can increase security by encrypting the master password kept on the NFC using RSA with the phones private/public key-pair. This would require anyone attempting to read the NFC stored data to possess both the NFC tag and the phone since the NFC tag would contain only the public key of the RSA key pair, while the phone associated with that specific tag holds the corresponding private key.

4.2 Authorization

Authorization in the KeySafe system will be given in the form of an OAuth authorization request. With the use of authorization the application will be given rights to retrieve user specific information using the OAuth authorization protocol. The KeySafe system will request access to the users email and username via a consent screen presented to the user. After the user accepts this, the application is given rights to retrieve user information with the scopes provided.

4.2.1 OAuth

The OAuth protocol [15] is the only method of user authentication for the prototype of this project and identifies a user with her Google account credentials. OAuth 2.0 is a well-spread protocol for authorization and can be used with many different login-methods such as Google-plus, Facebook or LinkedIn.

The complete interaction between a user and the server involve several steps:

- To initiate access on behalf of the user, the application calls a web service endpoint to get a *request token* for the app. This is a temporary token used solely for the authentication process.
- The user is presented with an authorization pop-up window (consent screen) which displays credentials required and request token. After the user signs in using her credentials she is redirected back to the application.
- The application calls a web service endpoint to exchange the request token for an *access token*.
- The application can now call the App Engine application's own web service endpoints using the access token until the user revokes access or the token expires.

The user only needs to register permission for the application once. The OAuth token will then remain within the application until access is revoked or the token expires. The lifetime of an OAuth token can be set using the Google API. The default expiration time of the refresh token is 60 days.

When authorized, the application can perform OAuth actions using a set of standard web service endpoints and the OAuth framework [16].

4.3 Endpoints

The communication between the KeySafe smartphone client and the backend system depends on the Google Cloud Endpoints feature. The endpoints feature consists of tools, libraries and capabilities that allow you to generate strongly-typed client libraries for Java (Android), Objective-C (iOS) and dynamically-typed libraries for JavaScript. With the help of Apache Maven, endpoints can be generated by invoking the Maven client libraries generation tool. This will create the necessary class and JAR-files.

The use of endpoints was chosen for this project because it provides a simple way to develop a shared web backend. Because the API backend is an App Engine application all the services provided in GAE becomes available. Furthermore, by using App Engine for the backend, work such as load balancing, scaling, server maintenance and system admin work comes free of charge. Although it is possible to create mobile clients for App Engine without endpoints, using endpoints makes the task easier because it frees you from having to write wrappers to handle App Engine communications. With the client libraries you can make direct API calls from the client.

Figure 5 below shows the specific client-to-server communications. Any data stored server-side has to be encrypted according to project goals and perquisites. Since encryption and decryption is done client side the backend does not implement encryption, it only deals with encrypted data and stores it or sends it to requesting client.

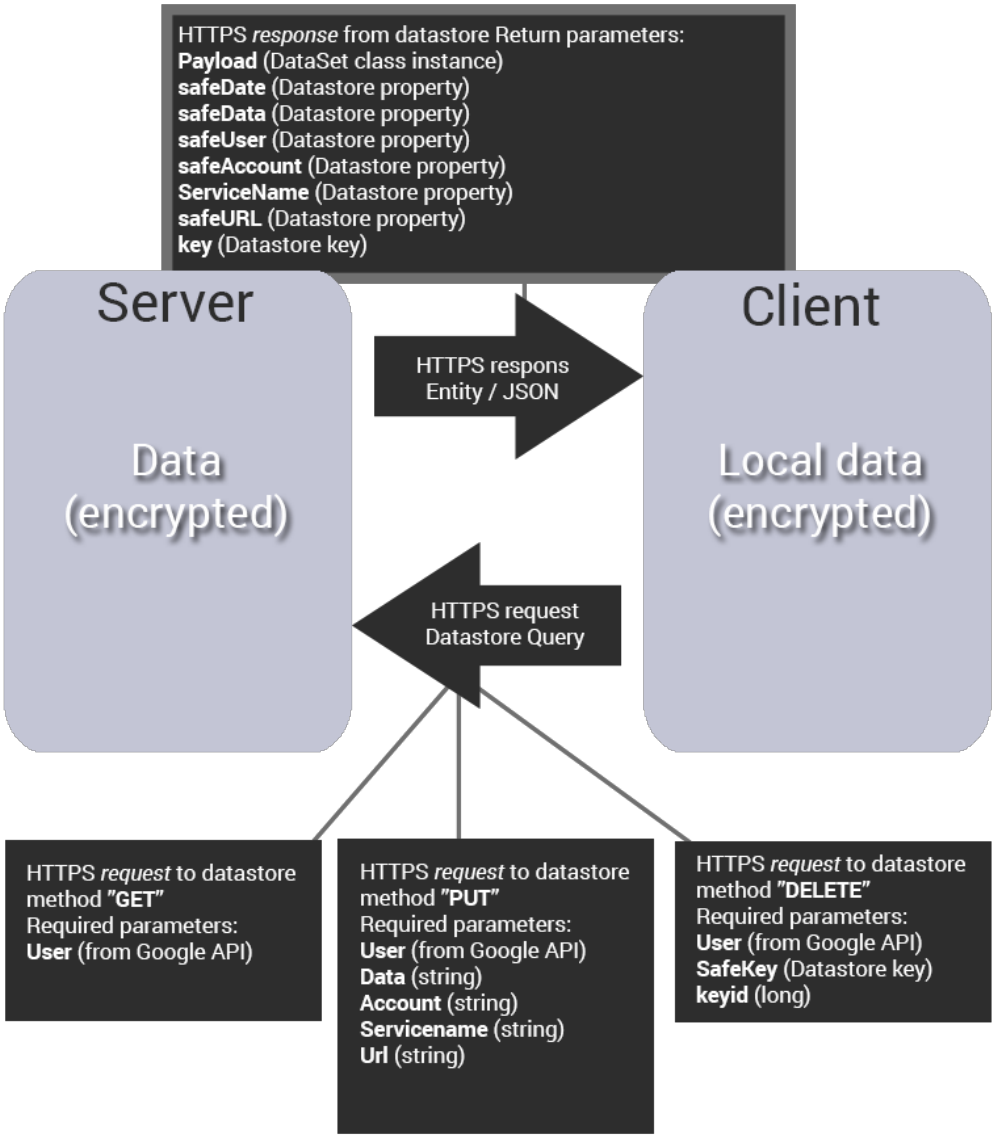

Figure 7 - The client-server communication implementation

4.4 Data storage

The storage of information is a vital part in this project. Unlike traditional web applications, such as those written in PHP, Perl/CGI or Ruby on Rails, which uses relational databases like MySQL for data storage, App Engine implements a persistent, schemaless (NoSQL) database called Datastore. [17]

The Datastore provides robust, scalable storage with the help of Bigtable [18], which is a data storage system built by Google. Datastore comes to good use

for web applications simply because it is better *equipped* for web applications then traditional databases, in the means of scalability. [19] As the number of user accounts and server requests increases this will be a useful feature for the KeySafe system. If there is a sudden spike in number of users the scalability of the Datastore would come in handy. For GAE the Datastore is the default database and therefore used in the project, in accordance with project specifications.

The different storage options for the system are presented in the figure below.

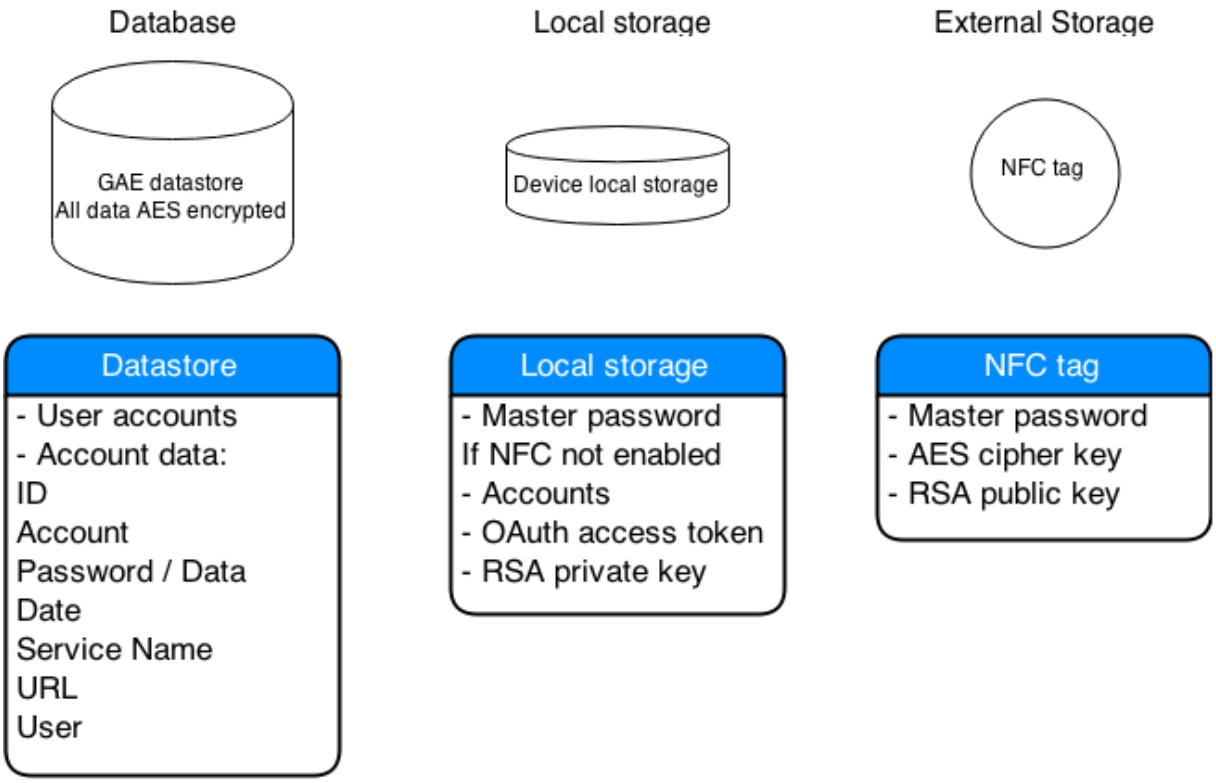

Figure 8 – Data storage

Any data stored locally or on the database must be encrypted. Accounts stored locally do not actually contain any data and are only used to reflect accessible accounts on the datastore.

4.5 Data encryption

For data encryption in the KeySafe system the method of Advanced Encryption Standard (AES) is used. [20]

The AES is a cryptographic standard which is used to protect electronic data by encryption. The standard was issued in 2002 by the National Institute of Standards and Technology (NIST). The AES algorithm is a symmetric block cipher that can encrypt and decrypt information [21].

The algorithm is capable of using cryptographic keys of 128, 192 and 256 bits to encrypt and decrypt data in blocks of 128 bits. For the KeySafe system the 128-bits version of AES will be used, since App Engine does not currently support the use of 256-bits AES.

AES is based on the *Rijndael algorithm*, developed by *Joan Daemen* and *Vincent Rijmen* and is a symmetric-key algorithm, meaning the same key is used for both encrypting and decrypting the data.

The KeySafe system utilizes AES by creating a secret key using the master password of the application as seed for the key. That means that the master password string is used, bite-wise, to generate a secret key. This key is then used for both encryption and decryption of the cipher.

There are several different encryption algorithms available today, such as triple-DES or CAST5. KeySafe uses AES because of its widely admitted security and the fact that it has been chosen by NIST to be a cryptographic standard. One could ask why not make your own encryption algorithm? Although this idea might seem interesting it is not recommended since the mathematics and engineering skills needed to make such an algorithm, in a secure way, can prove quite overpowering. Not to mention the amounts of tests that would need to be made to ensure stability and security. Therefore it is better to use an algorithm that has already been created and tested.

4.6 Surveillance

To detect unusual behaviour or attempts of forced entry a system log will be needed to record all client-server interactions. Thankfully, such a log is

provided by GAE. The GAE log is a series of messages that are updated in real time as HTTP requests are received and the program produces the results. For more information about the different features of GAE and how to use and program app engine see reference [22] [23].

4.7 Alternative input function

Passwords stored via the application should be made accessible so that they can be freely, and securely, transferred from the application to wherever they are needed, such as to a login interface in a web browser. Now, this can be considered somewhat tricky since the transfer of a password from the application to another instance is risky business and vulnerable for hacker attempts. One solution to this problem is to never transfer the data at all. On an Android smartphone this could be done by changing the input method when a password is to be provided from the standard keyboard input to the KeySafe application. An example to illustrate this idea further:

A user tries to access a website requiring login information, namely username and password, using a web-browser of an Android smartphone. If the user swipes a KeySafe-configured NFC tag a pop-up box will appear asking the user which account to select and which input method to use. Here the alternatives could be to use the KeySafe input-method or just go to the standard keyboard-input. Choosing the KeySafe input-method, an alternative view appears with buttons to input password or username corresponding to the account previously selected. The user then simply taps the button representing the authentication information she wants to provide, and this is sent as direct input by the application, and thus the information is only treated locally, and never visible to internet traffic.

4.8 Android Proof-of-concept

The proof-of-concept, or rather the prototype, for the Android application will be developed as far as possible with the time provided. It is likely that a finished prototype for storing and retrieving data will be ready when the project is to be presented. The application should be able to connect to a web-

based backend server running GAE providing the Android application with storage/database capabilities. Below is a mock-up of the smartphone app:

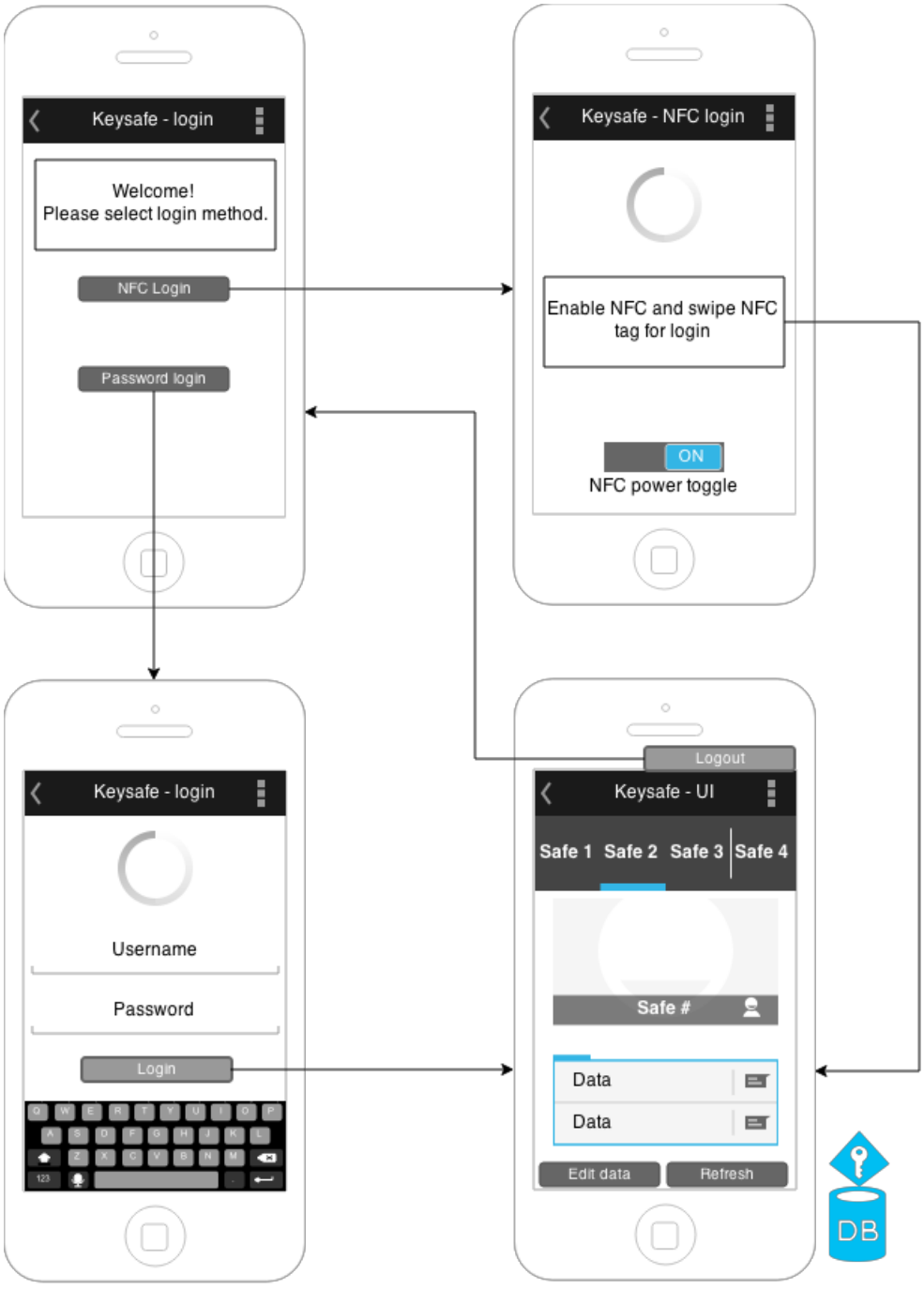

Figure 9 - Smartphone application mock-up'

5. Conclusions

It is our belief that the KeySafe application will become useful for private persons who want to store their passwords in a secure and easily accessible way and for companies where the employees have to remember and use different passwords or credentials for different services. With the use of external authentication and smartphones KeySafe provides a password storage service at a swipe, making sure the user may choose strong and sophisticated passwords without having to remember them.

Using NFC tags with an Android phone has proven to be both easy and effective. Reading or writing to the NFC chip with an HTC One is almost instant. It is somewhat cumbersome to present the NFC tag each time you want to access data, but this could be made even easier by, for example, purchasing a NFC ring and put it on the same hand as the phone is held. This way all the user needs to do is move her finger so that the ring is near enough to the NFC scanner to be read.

Using datastore for storage has proven most useful. The datastore comes free of charge and with a well design API and with the easy to work with Google developers console which gives direct access to databases, API commands and logs.

Since knowledge and previous experience of application development for Android was limited at the start of this project the prototype was built upon an existing application provided by Innocreate. The backend, that is the GAE server and endpoints libraries, where all created within the project and connected to the existing application. Naturally, some alternations and tinkering had to be made to the existing application in order to get it working properly with the new backend. We think that the most interesting function of the system is the KeySafe input method. Taking advantage of GAE platform independence this opens up for applying the same method to any device with configurable input methods.

6. Future work

The AES encryption algorithm needs a key seed to be used for encryption and decryption, a secret which can be shared between the server and the connecting client to ensure that nobody but the user can decipher the information. The user's chosen master password is in the prototype used as key seed. A good alternative to this could be to let the system generate a security strong password, following the password policies, and then storing the generated key on a NFC chip. This would only be done locally and only once, perhaps with the same *data shading* as described in section 3.4. This way no one, not even the user, would know the secret key-seed.

The possibility of storing GPS location of each client request was requested by the project specification. However, this was not implemented in the prototype of the project since it was considered to problematic and time consuming at this stage. For now, the surveillance provided by GAE per default will suffice. For a future release a GPS tracking system should be implemented allowing the system to map each client request to a GPS location and thus help detect and prohibit malicious use.

The encryption of data stored on the NFC tags mentioned in 3.3 will not be implemented for the prototype of this project. Tags will simply load and store data without encryption. For a future release this should be done to prevent an outside source from reading the AES key stored on the NFC chip. It was mentioned that RSA encryption could be used for the encryption of the NFC chip. This could prove particularly useful if digital signatures [24] were to be implemented in the future for use with multiple user access, since digital signatures works with asymmetric encryption just like RSA.

Considering the actual code there are several improvements that should be made before the application is released. As of now, the backend sends the client a JSON object that contains two fields: key (unique ID of each data post) and message. The message field is simply one long string containing all the Account data illustrated in figure 8 of section 4.4 above. It was implemented as such to simplify the encryption process. It became apparent at a rather late stage in the project that it is necessary to handle some specific fields of the

message field in order to parse the data properly. For such practise it would be better if the message field was given as a JSON object instead of a text string.

7. Discussion on Security

In Near Field Communication – *From theory to practise*, chapter 6 explains that there are some security risks to NFC authentication. The common attacks against NFC tags, which are in standby mode, can be categorized as follows:

- Tag cloning and tag impersonating
- Tag content changes
- Tag replacement and tag hiding

These attacks are aimed at NFC and RFID tags in general. In all operating modes of NFC technology short range communication is used. However, attackers with enhanced radio devices could communicate with NFC tags from a distance of several meters. The most common and serious threat of this form would be eavesdropping, meaning that an outside source attempts to read the data being transferred from the NFC tag to the reader, or vice versa. Other examples of attacks are data corruption, data modification, data insertion or MIM attacks. RSA encryption protects from content changing, however, encrypting the tag data does not prevent cloning. There are some proposals for providing protections against these kinds of attacks, such as *Active jamming* or *Faraday Cage* [25]. These methods are quite complex and requires a great deal from the user so the only realistic protection to be had against attacks of this nature would be to establish a secure channel and to keep the NFC communications to a minimal.

Since this application focuses on storing the AES key on a NFC chip instead of locally, it is assumed that the chip is handled with care and discretion by the user. The system security would be severely compromised if the NFC tag was cloned. There are some actions to prevent this from happening. NFC tags with built-in security measures could be used to prevent eavesdropping, data insertion or MIM attacks. Some of these more sophisticated NFC tags provide the option of using on-chip encryption which protects the data from attacks of the sort. One example of such a tag is the NTAG21x [26]. A more secure tag in would provide all the protection possible from outside attacks. Another way to

deal with problems of this nature could be to change the AES cipher key and re-encrypt data using the new key every once in a while. By doing so, even if someone has acquired the AES cipher key, it would be useless if a new key was put in action. If the user ever suspects that the application security have been compromised, or if the user was warned by the surveillance system described in section 4.6 the AES cipher key should be immediately changed and all encrypted data on server resynced to use the new encryption key.

When it comes to the choice of external authentication method, there are several things to take into consideration. In many cases, it is questionable if user experience could be cut back for the sake of security, especially if the system is intended for private use. To find a good balance is important for a good external authentication system. As previously mentioned with the RSA authenticators in section 3.1, the method of number-token authenticators is a widely spread method. However, this method can be quite cumbersome, especially if the user is in a hurry. The authentication-code usually consists of a longer number which is generated by a system-known algorithm and checked for authenticity once entered. This method is often used amongst companies and government agencies. However, an article in New York Times magazine [27] tells the story of a team of scientists who in as little as 13 minutes managed to break the security algorithm of such a device called *the SecurID 800* made by RSA. The scientists further claims that they could break similar tools produced by other companies. However this claim is accurate or not is still debated.

Another thing that one might ask is why AES 128-bit is used, and not AES 256. If there is a stronger encryption available, why not use it?

The simple answer to that question is because it is not allowed. Per default, JDK (or at least Sun's JDK) does not allow AES algorithms stronger than 128 bit. If stronger algorithms are needed, the JCE Unlimited Strength Jurisdiction Policy Files must be installed and this complicates any release of the product, since this might not be legal, depending on local regulations. [28]

We will conclude this report with a comparison to the application *Keeper* mentioned in section 2.2. Keeper stores passwords and sensitive data both locally and on the system server, in both cases stored data is AES encrypted. KeySafe also stores data on system server in encrypted format, but not locally. A feature for Keeper since it makes your data available even if no internet connection is available, but it makes the device a target for attacks. This could be made possible with KeySafe as well, but for the sake of greater security we chose to only keep sensitive data on the server and never store it locally.

Keeper implements two-factor authentication but no means of external authentication. This is where we believe KeySafe will be the more secure choice over Keeper and the reason why people would chose KeySafe rather than Keeper. The user will be required to obtain a NFC tag to be used with the application in order for external authentication to be possible. For a private person this could be too much to ask, but for a company it is simply a matter of acquiring a bunch of NFC tags which are reasonable cheap. Besides, when it comes to security it is our belief that all users, commercial as well as private, feels that it is worth the extra money and effort in order to obtain higher security for their sensitive data.

While KeySafe in its current state is not finished people who have been introduced to the project idea have expressed positive response and believe that they would find such an application useful. With more time we are confident that KeySafe can become a well-spread application for high-security storage of sensitive data.

8. References

[1]: *Dave Evans,* The Internet of Things, Cisco IBSG 2011, http://www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf

active 2015-02-11

[2]: Keeper Security Application, Version 2014.01.24 – 123, https://keepersecurity.com/

active 2015-02-10

[3]: *Fadi Aloul, Syed Zahidi, Wassim El-Hajj ,* Two-factor authentication using mobile phones*,* Department of Computer Science & Engineering American University of Sharjah, 2009

http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5069395&tag=1

active 2015-02-01

[4]: Google Authenticator Application

https://code.google.com/p/google-authenticator/

active 2015-02-10

[5]: *Eran Kalige and Darrell Burkey,* Head of Security Operation Center, Versafe 2012
https://www.checkpoint.com/products/downloads/whitepapers/Eurograbber_White_Paper.pdf

active 2015-02-11

[6]: RSA SecurID hardware authenticators

https://store.emc.com/Product-Family/RSA-SecurID-PRODUCTS/RSA-SecurID-Hardware-Authenticators/p/RSASecurIDHardwareAuthenticators

active 2015-02-11

[7]: *Vedat Coskun, Kerem Ok* and *Busra Ozdenizci,* Near Field Communication – *From theory to practise,* John Wiley & Sons Ltd , ISBN: 9781119971092, 2012

[8]: *Stephen A. Weis,* RFID MIT CSAIL

http://www.eecs.harvard.edu/cs199r/readings/rfid-article.pdf

active 2015-02-01

[9]: Picture inspired by: Near Field Communication – *From theory to practise,* page 117-118

[10]: Picture inspired by: Near Field Communication – *From theory to practise,* page 117-118

[11]: Man in the middle attack

http://en.wikipedia.org/wiki/Man-in-the-middle_attack

active 2015-02-11

[12]: ISO/IEC 14443-4 International standard 2001

http://jpkc.szpt.edu.cn/2007/sznk/UploadFile/biaozhun/iso14443/14443-4.pdf active 2015-02-07

[13]: *James F. Kurose, Keith W. Ross*, *Computer Networking - a top down approach* (6th edition), Pearson Ed Inc., ISBN-13: 978-0-13-285620-1, 2012

[14]: NFC Forum, a NFC data form

http://nfc-forum.org/

active 2014-08-21

[15]: *D. Hardt, Ed,* The OAuth 2.0 Authorization Framework, Internet Engineering Task Force (IETF) 2012

http://tools.ietf.org/html/rfc6749

active 2014-12-07

[16]: OAuth endpoints

https://developers.google.com/appengine/docs/java/OAuth/

active 2014-08-26

[17]: Java Datastore API

https://developers.google.com/appengine/docs/java/datastore/

active 2014-09-03

[18]: *Fay Chan, Jeffery Dean, Sanjay, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Candra, Andrew Fikes, Robert E. Gruber,* Bigtable, A distributed storage system for structured data*,* Google, Inc. 2008

http://doi.acm.org/10.1145/1365815.1365816

active 2015-03-18

[19]: *Charles Severance,* Using Google App Engine, *O'Reilly Media Inc.,* ISBN: 978-0-596-80069-7, 2009

[20]: Advance Encryption Standard (AES), Federal Information Processing Standards Publication 197, 2001

http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

active 2015-02-10

[21]: Federal Information Processing Standards Publication 197, 2001

http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

active 2014-08-26

[22]: *Dan Sanderson,* Programming Google App Engine*, O'Reilly Media Inc.* ISBN: 978-0-596-52272-8, 2009

[23]: *Charles Severance,* Using Google App Engine, *O'Reilly Media Inc.* ISBN: 978-0-596-80069-7, 2009

[24]: *James F. Kurose, Keith W. Ross*, *Computer Networking - a top down approach,* chapter 8 (6th edition), Pearson Education Inc., ISBN-13: 978-0-13-285620-1, 2012

[25]: *Vedat Coskun, Kerem Ok* and *Busra Ozdenizci,* Near Field Communication – *From theory to practise, chapter 6.4.4,* John Wiley & Sons Ltd, ISBN: 978111997109, 2012

[26]: NTAG

http://www.nxp.com/products/identification_and_security/smart_label_and_tag_ics/ntag/

active 2015-01-20

[27]: *Romain Bardou, Riccardo Focardi, Yusuke Kawamoto, Lorenzo Simionato, Grahm Steel, Joe-Kai Tsay,* Efficient Padding Oracle Attacks on Cryptographic Hardware, 2012

https://eprint.iacr.org/2012/417.pdf

active 2015-02-12

[28]: Sun's Java AES

http://docs.oracle.com/javase/7/docs/technotes/guides/security/SunProviders.html

active 2014-09-03

Downloads

GAE SDK

https://developers.google.com/appengine/downloads

Appendix 1: Glossary

AES	Advanced Encryption Standard. A symmetric key encryption algorithm. The Advanced Encryption Standard (AES) specifies a Federal Information Processing Standards approved cryptography algorithm that can be used to protect electronic data. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
API	Application Programming Interface. A specification of how software components should interact with each other. http://www.computerworld.com/article/2593623/app-development/application-programming-interface.html
Android	An operating system for smartphones. https://www.android.com/
Cloud	Cloud computing, a concept used to describe a large number of computers connected, through the Internet for instance. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf
Eclipse IDE	An Integrated Development Environment that is used for coding and development in this project. https://eclipse.org/
GAE	Google App Engine. Google's platform for building applications on Google's infrastructure. GAE is a Platform as a Service (Paas) offering that lets you build and run applications on Google's infrastructure. https://cloud.google.com/appengine/docs
Go	An open source programming language. https://golang.org/
HTTP	HyperText Transfer Protocol is an application protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communications on the World Wide Web. http://www.w3.org/Protocols/
HTTPS	HyperText Transfer Protocol Secure is a communications protocol for secure communications over a computer network. It can be described as HTTP on top of the SSL/TSL protocol. https://tools.ietf.org/html/rfc2818
Java	An object oriented programming language. https://www.oracle.com/java/index.html
JSON	JavaScript Object Notation. A human-readable standard for data exchange. KeySafe uses JSON because of its capability to work well with smartphones, and because of its similarities to XML which is a common text format in many mobile phone

	operating systems. http://www.json.org/
MAC address	Media Access Control address. A unique identifier for network interfaces such as network cards. http://standards.ieee.org/develop/regauth/tut/macgrp.pdf
Malware	Malicious software. Software used to disrupt computer operations, gather sensitive information or gain access to private computer systems. https://support.google.com/webmasters/answer/163633?hl=en
Maven	A software project and management comprehension tool developed by Apache http://maven.apache.org/
PHP	A server-side scripting language. http://php.net/
PIN	Personal Identification Number. A digital code combination used for authentication.
Python	A general-purpose, high-level programming language. https://www.python.org/
QR Code	Quick Response Code. A machine-readable barcode. http://www.nacs.org/LinkClick.aspx?fileticket=D1FpVAvvJuo%3D&tabid=1426&mid=4802
RSA	Asymmetric public-key cryptosystem. Public encryption key and private decryption key. RSA is a public-key cryptosystem. RSA consists of two interrelated components; the choice of public/private key and the decryption/encryption algorithm. http://searchsecurity.techtarget.com/definition/RSA
SSL/TSL	Secure Sockets Layer/Transport Layer Security. Cryptographic protocols which are designed to provide communication security over the Internet. http://tools.ietf.org/html/rfc5246
Trojan	A type of malware program. Often acts as a *backdoor*, granting unauthorized access to infected computers. http://techterms.com/definition/trojanhorse
XML	eXtensible Markup Language. A markup language used to describe data. Widely used in web services. http://www.w3.org/TR/REC-xml/
	All links in glossary active 2015-02-07