MATRIX BARCODE BASED SINGLE-SIGN-ON WITHOUT TRUSTED THIRD PARTY

By

Valeriy Chevtaev

A DISSERTATION

Submitted to

The University of Liverpool

in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

27/12/2015

ABSTRACT

MATRIX BARCODE BASED SINGLE-SIGN-ON WITHOUT TRUSTED THIRD PARTY

By

Valeriy Chevtaev

This dissertation is a thorough research on the topic of using Matrix barcodes for authentication on existing web services in actual Internet infrastructure. In this study I decided to stick to most popular matrix barcode implementation which is Quick Response (QR) code because of its simplicity and popularity. On the other hand, the technical solution, which is made as a part of this study, is not limited to QR codes and could be used as a starting platform for authentication systems using other types of matrix barcodes.

In the past few years, several professionals has been researching and developing QR code based authentication services which in comparison to this study depend on third party service for storing user credentials. Implemented solution doesn't require any updates on existing web sites and web services. Multilayered encryption strategy including both asymmetric and symmetric cryptography methods make it possible to store user credentials on user's devices only omitting third party "big brother" credentials storage services.

A working technical solution has been developed as a part of this study which could also be treated as a prof of concept that the proposed solution has all reasons to become one of the standard authentication mechanisms alongside more traditional authentication methods like manual username and password typing, or certificate based authentication which recently became quite popular in the areas requiring highest level of security, like government and defence sectors, banking and money-related services.

DECLARATION

I hereby certify that this dissertation constitutes my own product, that where the language of others is set forth, quotation marks so indicate, and that appropriate credit is given where I have used the language, ideas, expressions, or writings of another.

I declare that the dissertation describes original work that has not previously been presented for the award of any other degree of any institution.

Signed,

Valeriy Chevtaev

Student, Supervisors and Classes:

Student name:	Valeriy Chevtaev
Student ID number:	H00022997
GDI name:	Taly Sharon
CRMT class ID:	LAUR-906-201462-3
DA name:	Yongge Wang
CAC class ID:	UKL1.CKIT.702.H00023862

ACKNOWLEDGEMENTS

Professor Yongge Wang who agreed to be my dissertation supervisor. Prof. Wang took crucial role in my dissertation, helping me to choice the topic and giving me support and advices needed to accomplish this challenging project.

I would also like to thank all the University of Liverpool and all the lecturers I met during my study who have given me the right directions in every module I studied.

A special thanks to my wife, Tatiana, who supported me during the whole process of implementation of this project and helped me to understand the topic from non-technical user perspective.

Finally and the most important, I would like to thank my family and friends, especially my mom and dad, without their support and words of encouragement I would have never finished my study.

TABLE OF CONTENTS

LIST OF TABLES

Page

LIST OF FIGURES

Page

1. INTRODUCTION

1.1. Scope

The dissertation aim is to proof that matrix barcodes could be used in authentication systems of existing web services without trusted third parties involved in the authentication process, therefore user credentials are being store on user's trusted device only, ex. smartphone or a tablet.

This might be a great benefit for users by making authentication process for them easier and more transparent, omitting remembering various credentials or even more complicated use-cases like providing an authentication solution within a company for users without even knowing plain text credentials.

QR codes were selected to be used in the implementation of this dissertation being most popular and casual matrix barcode implementation for typical Internet user. Moreover QR codes became so popular in the last few years that there are many open and free software libraries available in Internet to be used to generating and parsing QR codes, including JavaScript and Swift programming languages.

The study doesn't intend to be an industrial standard for authentication system and could be used in different variations and even more different application areas not related to authentication itself, for example to establish secure communication channel between desktop computer and portable device.

1.2. Problem Statement

The great benefit of the proposed solution in comparison to existing ones is that it doesn't require trusted third parties to be involved in the process which means that only a user himself is responsible on how safe his/her credentials are being stored.

Some of the related resources mentioned in this study give an insight on a possibility of QR code based authentication to existing web services, which on the other hand require not only a user's device with photo camera but also extra integration to be done on

existing web service's side, like it's described in Borchert & Gunther, 2013 and Gibson 2014. This complexity may not be an option for target customers (users of existing web services) as it's leads to extra development costs and potentially new breaches in an exists system.

Some of existing QR code based authentication implementations available in Internet, like mydigipass.com, 2015, eKaay.com, 2015 or TIQR.org, 2015, provide complete solution for integration of QR core based authentication mechanisms to any web service which already adds extra work and costs to business owners, i.e. existing web services. Moreover, as a part of this integration, a separate mobile application is required to be installed on user's device, which may be tricky for a target user in case different web services apply different technologies for QR core based authentication.

The idea and the main problem of this thesis is to design and implement technical solution for automating authentication process for a user in existing Internet services (ex. Google Accounts) based on generating and parsing QR codes on "the fly". The solution must be implemented in a way that no additional extra integration is required by existing web services as well as no trusted third parties should be involved, which is beneficials for all involved parties: target users and existing web services.

1.3. Approach

A research will be conducted to identify what existing matrix code based authentication systems are available nowadays, proving that all of them require trusted third party to be involved in the authentication process.

Potential libraries for QR code generating and parsing as well as libraries, providing cryptographic functions like MD5 hash code calculation and AES encryption, will be tested (making sure they work properly) and selected to be used in the final solution.

Security analysis will be conducted for the whole solution as well as for each implemented component separately paying greater attention on how securely user's credentials are stored and transferred on every step of inter-component communication.

Three main components will be developed as a part of the technical solution for this project, which are:

1. Trusted component: The Vault iOS application to store user credentials with an ability to scan and parse QR code generated by Desktop component.

2. Desktop component: Chrome web browser extension to start authentication process itself by generating QR code with meaningful payload in it.

3. Communication channel component: Lightweight file sharing service as a replacement for more complex services like Dropbox or Google Drive.

1.4. Outcome

Working technical solution, i.e. IT artefact, will be provided as an outcome of this study. This will include source code of the solution, deep analysis of its security and a demonstration of the running solution provided as a several screenshots. Working technical solution also means that it properly functions providing described functionality and meeting all defined criteria.

Main deliverables of this study will be:

1. Source code of The Vault iOS application for QR code scanning and credentials storing;

2. Source code of Chrome extension to be used on client's desktop machine for authentication;

3. Source code of lightweight file sharing service to be used as a communicational channel between iOS app and Chrome extension. Simple replacement for Dropbox and similar services;

4. Link to a source code on Github (https://github.com) for all implemented components;

5. Implemented lightweight file sharing service URL available in Internet working over SSL/HTTPS.

2. BACKGROUND AND REVIEW OF LITERATURE

2.1. Background

Nowadays millions of users daily face a problem of remembering their passwords to web sites and web services like social networks, e-mail, instant messaging and others. Some of those users keep their password in mind, others on a sticky paper on a monitor or in an electronic file, but all of them are required to enter user credentials to be able to access required functionality.

According to Bonneau et al (2012) in many cases this is a real critical breach to the privacy of a user: “The continued domination of passwords over all other methods of end-user authentication is a major embarrassment to security researchers”, but still being a major authentication method in existing IT systems: “As web technology moves ahead by leaps and bounds in other areas, passwords stubbornly survive and reproduce with every new web site”. Mentioned problems are also described in the work of Grosse et al (2013) who conclude in their work: “we feel passwords and simple bearer tokens, such as cookies, are no longer sufficient to keep users safe".

Moreover, login and password authentication method is also not the best solution for end users if we are talking about usability, this topic is partially raised in the work of Groose et al (2013) who say: “security and usability problems are intractable: it’s time to give up on elaborate password rules and look for something better”.

Thereby, in the last few years a major attention has been given to alternative way of authentication, such as 2-step verification and matrix barcode authentication, as well as authentication strategies using mobile devices. In this study we will investigate the possibility of using matrix barcodes (QR codes) for authentication purposes without trusted third parties involved and as a part of this project a proof of concept software solution will be implemented to demonstrate real application of proposed method.

2.2. Literature Review

There are several existing QR code based authentication software solutions on the market including commercial software like mydigipass.com, 2015 or ekaay.com, 2015 as well as free and open source solutions like SQRL, 2015 or tiqr.org, 2015.

Some of the mentioned products provide complete solutions for QR code based authentication to be integrated into existing web services including both client and server components. Solutions like ekaay.com, 2015 or mydigipass.com, 2015, for example, apart of a user device with photo camera also require extra integration which should be done on a server-side of implementing web service. This complexity may not be an option for existing web services providers as it's leads to extra development costs and potentially new breaches in the system. Moreover, some solutions like tiqr.org, 2015 require special software application to be installed on user's device which makes it even more tricky to provide transparent solution to target users.

Gibson, 2014 in his work, apart of client-side and server-side components, also develops new SQRL protocol to be used by any party adopting his solution which add even more complexities to the existing web services as well as to client software to add a support of that protocol. All of these makes the whole solution looks extraordinary hard to adopt in existing client and server systems.

A slightly better approach has been mentioned by Dodson et al, 2012 in their study they developed authentication software solution based on QR codes to extend OpenID protocol functionality. Besides of special client-side software is required to be installed, the proposed solution still requiring several updates on OpenID provider side, which is a trusted third party itself. The method proposed by Dodson et al, 2012 on the other hand limits a user to stick to OpenID authentication mechanism, on the other hand theoretically it's possible to extend the solution to work with other authentication systems similar to OpenID.

There are several patent applications which has been submitted in the last few years related to the topic of this work, i.e. authentication mechanisms using mobile devices based on matrix barcodes, like the one submitted by Cobos et al, 2012, which again

requires trusted third party to be involved: “TRUSTED SERVER 20: the purpose of the Trusted Server is to manage the creation and distribution of personal secrets to users (to their mobile devices)”. Very similar idea is also mentioned in another patent application by Desoto et al, 2013.

All mentioned authentication systems provide fairly user-friendly and secure solutions which on the other hard require a significant work to be done by the owners of existing web services, therefore they hardly become widely popular unless an industrial standard will be developed.

2.3. Theory

In this work QR code based authentication software solution will be developed to drop out server-side integration step which has been mentioned as a mandatory part of all of the previously mentioned existing solutions. This approach is expected to provide a great benefit for the owners of existing web services because no extra server-side integration will be required to adopt QR code base authentication mechanism to existing systems.

For this project, QR codes were selected to be used in the implementation of this dissertation being most popular and casual matrix barcode implementation for typical Internet user. The simplicity as well as the availability of a wide range of existing QR code generating and parsing libraries was the major factor of selecting QR code implementation of matrix barcode 2D pattern.

Typical use-case

A typical use-case of the developed system is to authenticate user on existing web service on public or personal computer using special software (which will also be developed as a part of this project) running on a trusted portable device, specifically smartphone like iPhone, which acts as a storage for user’s credentials.

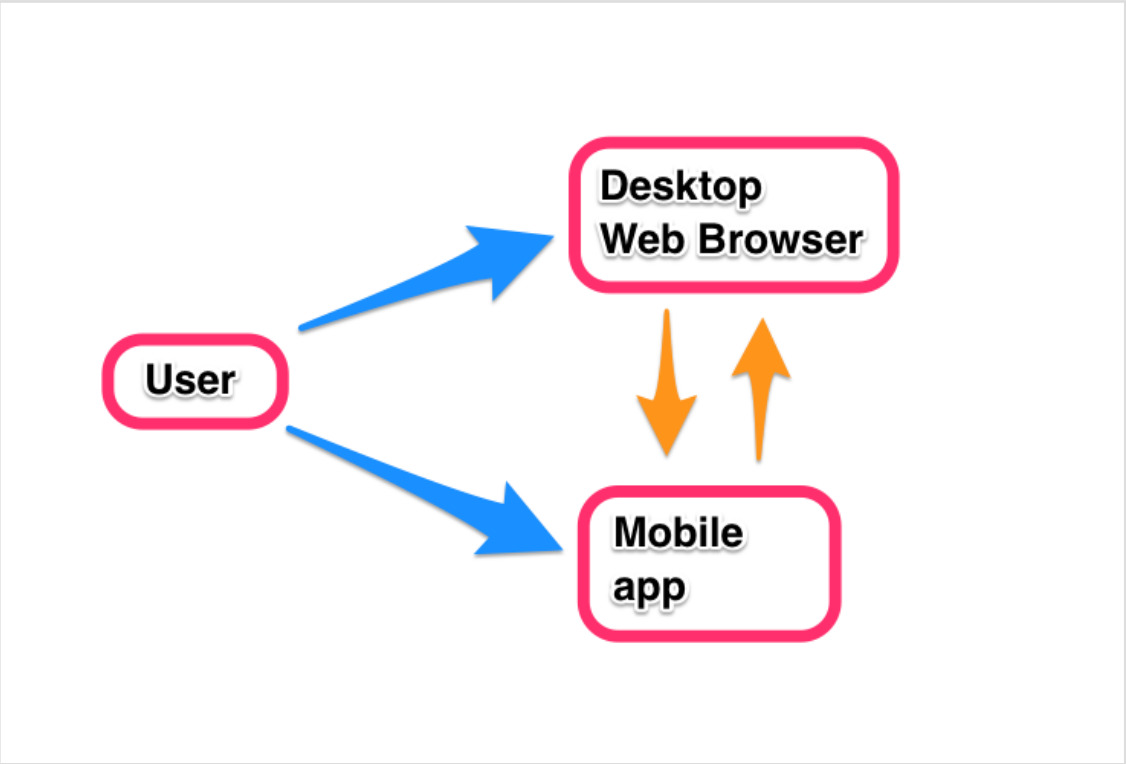


Figure 1. Typical use-case.

A user during the authentication process will interact with only two components: one running on a desktop public or personal computer and the other running on his trusted portable device. A trusted device is expected to be trusted by a user, keeping in mind that a device might have personal user information, but will require a user to authenticate on it prior to access private data. Authentication might be done by entering PIN code and/or by providing biometrical data like fingerprint, or any other applicable authentication method.

Moreover, desktop and mobile components are expected to open a communication channel to exchange encrypted data between each other. In theory, in should utilise Diffie-Hellman exchange algorithm in conjunction with secure connection between those two components via secured socket, very likely utilising SSL/HTTPS. According to Gibson, 2015: “it is generally more secure not to send all aspects of a secure communication through a single channel because the security of that channel may be compromised”, so multi-channel communication is considered as a very important attribute of authentication process implemented as a part of this study.

Expectations

The software solution to be developed should meet the following criteria but not limited to:

1. Be able to authenticate to an existing web service;
2. User credentials are stored on a user's trusted device only and in encrypted format in the communication channel;
3. User is required to enter his credentials only once on a trusted device in a special The Vault application;
4. QR code to be used to start authentication process;
5. There is no need for a user to enter his/her credentials manually on a web page (Google login page);
6. Special software, i.e. web browser predefined script, is expected to be installed on a public / personal desktop computer.

Probably, the most important part of the workflow is authentication process initiation which expected to be running on a public or a personal computer via Internet having web browser installed. There are several possible ways of running third-party scripts from web browser, for example running Java Applet, executing JavaScript on a page, running web browser's extension of plugin, opening bookmarklets and others. One of the methods is expected to be used to allow target user to start authentication initiation process on currently opened web page.

Developed system is expected to be highly secure considering the solution to be running on public desktop computers as well as personal computers. Also keeping in mind the privacy of developed communication channel between desktop and trusted components which is also expected to be implemented as a part of this project.

System components

The system to be developed will consist of three major components each of which is mandatory and cannot be dropped out from the system. As it already has been mentioned, the components are:

1. Desktop component. Web browser's predefined script acting as an initiator on the authentication process. The authentication process begins when a user starts predefined script which generates QR code containing special payload including web services URL, timestamp, random sequence string and asymmetric crypto algorithm information.

2. Trusted component. The Vault application to store user credentials and to scan QR code generated by the web browser's extension component, assuming that this software will be running on a trusted portable device. The Vault application will require a user to authenticate with fingerprints to access the encrypted data storage containing an information about suer's credentials as well as to be able to scan QR code itself. Afterwards, scanned QR code will be parsed by this software to retrieve required information for processed to next steps of the authentication process.

3. Communication channel component. File sharing service acting as a communication channel. The communication between desktop computer and a portable user's device will be conducted via file sharing service (a server), which on the other hand won't store user credentials but only encrypted information, moreover the information will be stored on the server only for specified time period counted in minutes so that the retrieval of that encrypted data won't be available after the authentication session has been closed to reduce the risk of compromising the data and so decrypting user's credentials.

2.4. Terms

The following terms has been introduced by the author of this work for future references in this study:

Communication channel component, also known as lightweight file sharing service, the service implemented in pure PHP to provide basic file sharing functionality via HTTP/ HTTPS protocol. To be more precise providing the following two methods: 1) upload a file, 2) download a file.

Trusted component, also known as The Vault app, iOS application developed in Swift acting as credentials vault for storing user credentials to Google accounts. Running on user's iPhone 6 or newer with fingerprint authentication enabled.

Desktop component, also known as The Vault Chrome Extension, Chrome Extension developed in JavaScript to be used as an initiator of the authentication process in the implemented solution. Generates QR code which is to be scanned by The Vault app on the next stages. As well as providing automatic authentication functionality for Google Accounts on later stages.

Trusted device, is a device which a user trusts, keeping in mind that a device might have personal user information, but will require a user to authenticate on it prior to access private data. Authentication might be done by entering PIN code and/or by providing biometrical data like fingerprint, or any other applicable authentication method.

The following terms are publicly known in IT though mentioning all of them would be good to avoid any ambiguities:

2D, two-dimensional. For example, QR code is a two-dimensional implementation of matrix barcode.

Bookmarklet, a bookmark in web browser which runs hardcoded JavaScript code, commonly used to add dynamics in web browsers but in most cases using web browser extension is preferable.

JavaScript, front-end scripting language, used is various areas in software engineering including but not limited to Chrome Extensions development and web user interface scripting.

JSON, which stands for JavaScript Object Notation, a simple format for storing and accessing structured data.

PHP, programming and scripting language, very popular for server-side development.

QR code, Quick Response code, one of the implementations of matrix barcodes.

Swift, programming language by Apple to be used for iOS/OSX applications development, but not limited to it.

Trusted Third Party, is an entity which facilitates interaction between two components of a software system. This could be a web service, a web site, a communication channel or any other type of software.

UML, Unified Modelling Language, general-purpose modelling language in software engineering.

User credentials, user authentication information, i.e. login and password, to access an application, web site or any other services requiring some kind of authorisation.

3. ANALYSIS AND DESIGN

3.1. Introduction

In the implemented solution a user is expected to use special software to authenticate to a web service skipping a step of manually submitting user credentials via HTML form on a target web service.

High-level authentication workflow described below in this section is considered as a basis for this study considering the following theoretical assumptions which will be clarified in more details in the next chapter (Chapter 5. Implementation (Realization)):

- Trusted device is a portable device which a user has an access to at any time at any place. Prior authentication is required by a device's operating system to access its functionality by asking a user's PIN code or a biometrical information like a fingerprint. A typical examples of this type of a device are a smartphone like iPhone 6 and Android Nexus 6.

- Desktop computer is public or personal desktop computer which a user has an access to.

- Desktop component is JavaScript script which is available in a user's web browser or could be installed via Internet. This script is publicly available via Internet and should be installed on a desktop computer to initiate authorisation process.

- Trusted component, also know as The Vault app, is an application developed to be running on a user's trusted portable device having fingerprint authentication functionality available.

- Communication channel component is a file sharing service deployed on a publicly available server and accessible via Internet. A typical examples of this type of a service are Dropbox or Google Drive.

- QR code is selected as an implementation of matrix barcode.

High-level authentication workflow

Figure 2. User opens web browser on desktop computer.

As it is shown on Figure 2 a user opens web browser on desktop computer with preinstalled Desktop component. Afterward a user opens a web service requiring to enter his/her user credentials.

Figure 3. User initiates authentication process.

Figure 3 demonstrates the next step of authentication initiation process which is basically running JavaScript script to produce session-specific QR code containing required information about current user authentication session including web service accessed and current session's random sequence as well as Diffie-Hellman encryption information, including desktop-side public key.

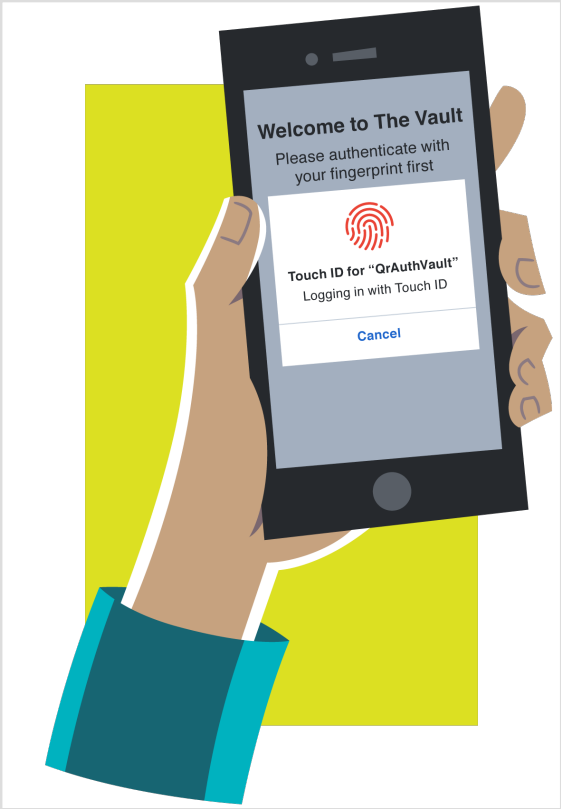


Figure 4. User authenticates on his/her mobile device.

On Figure 4 a user accesses his/her trusted portable device and starts special application, i.e. trusted component. User's fingerprint is required in order to access special functionality of this application like scanning QR code and managing user's credentials to all web services a user wants to access via developed solution, i.e. services to authenticate with QR code.

Figure 5. User scans previously generated QR code.

On the next step as described in Figure 5 a user scans QR code generated on previous steps by desktop component. An information contained in that QR code, including public key generated by desktop component, is used by the app to generate the second part of Diffie-Hellman encryption information, specifically client-side public key.

Afterward, if user credentials were found in the Vault app are being encrypted using symmetric cryptographic algorithm with private key calculated from public desktop component key and secret key generated on a side of trusted component, as well as other parameters of Diffie-Hellman exchange generated on desktop component side which are specific for this particular authentication session.

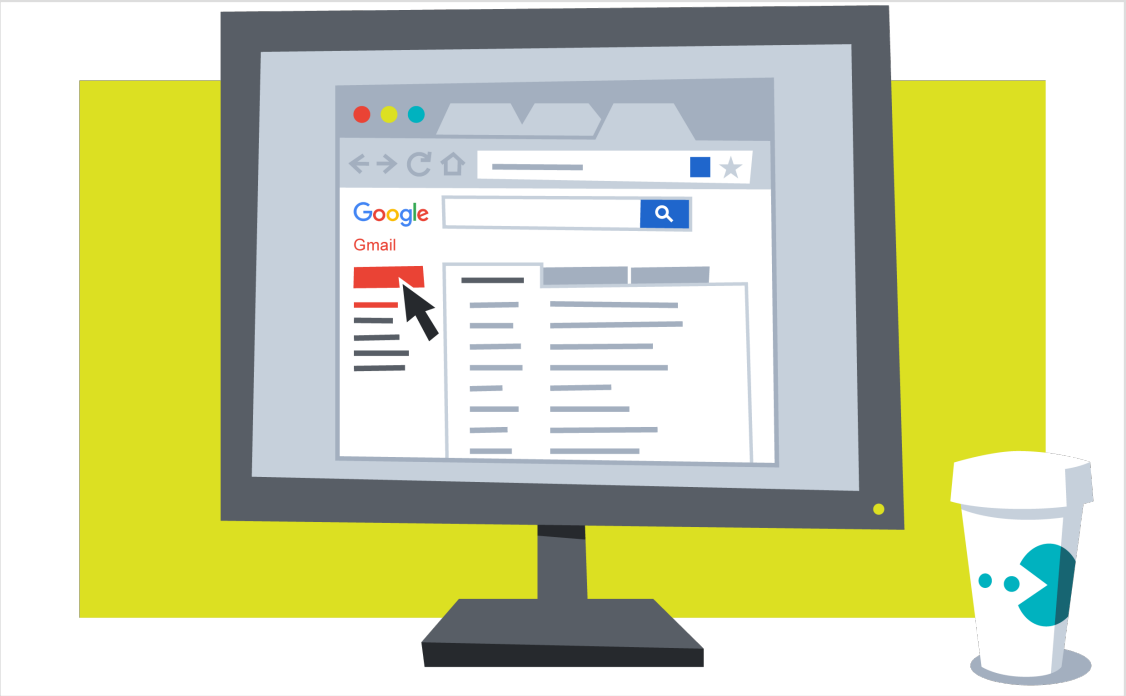


Figure 6. User has been authenticated on target web site.

On the final step of QR code based authentication process as shown in Figure 6, a user is authenticate to initial web service with credentials received from communication channel (component) provided in encrypted form from The Vault app. On this step decryption of encrypted data is being done with symmetric private key generated based on initially generate Diffie-Hellman parameters including local private secret as well as public key received from trusted component via secure communication channel.

Decrypted credentials are not being stored in web browser or desktop computer and are only used once to simulate real user actions on web page (actually, simulate authentication process itself) by filling HTML web form with user's login and password and clicking submit / login button in the end. This is done as a part of initial JavaScript script which also has been used on the second step to generate QR code.

3.2. Components

As per assumptions defined in the Introduction section of this chapter QR code based authentication system consists of three main components to be implemented, which are: desktop component, trusted component and communication channel component. The

latter on might be re-used a already existing file sharing service without a need to be implemented, in the developed solution though a lightweight file sharing service is being implemented to provide simple and limited set of functions.

Below, Figure 7 in comparison to Figure 1 define in theoretical part of this work, introduces the third component, i.e. communication channel or file sharing service, which is used by other two components in their communication for exchanging data.

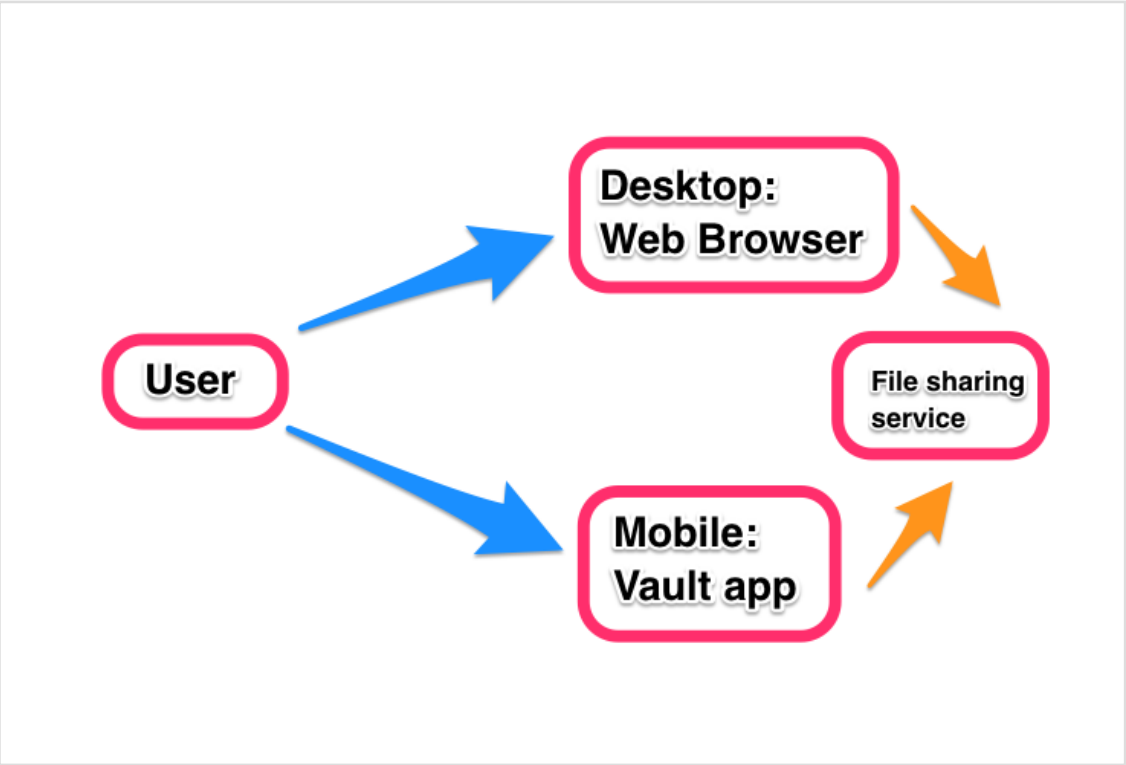


Figure 7. Authentication use-case, improved.

A combination of various security mechanisms will be used in the solution including symmetric (AES 128 bit) and asymmetric (Diffie-Hellman) algorithms as well as random generators and time based random sequence generator seeds. In the next sections of this chapter a detailed description of the developed solution including encryption algorithms will be given.

Figure 8 below gives an understanding on typical workflow of the developed solution as well as a detailed description of communication of all participating entities within the developed system.

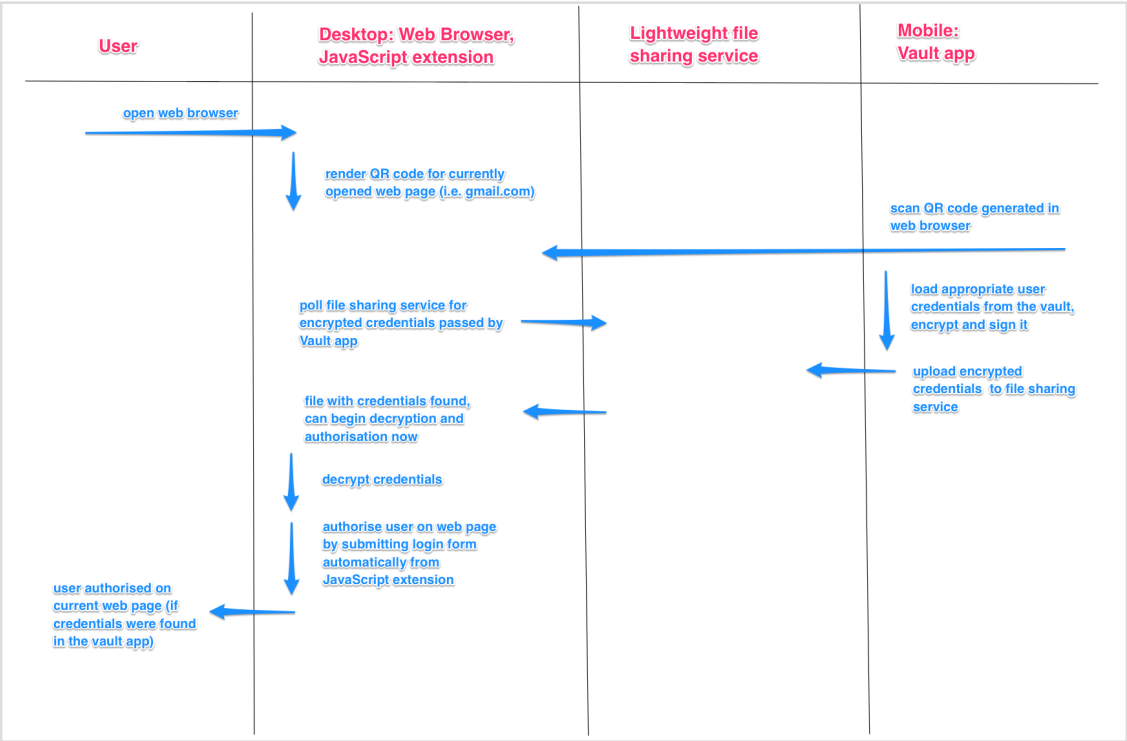


Figure 8. Workflow diagram of the developed solution.

3.3. QR code and its limitations

Limitations

QR code can store limited amount of data depending on a size of a generated barcode image. According to QRcode.com, 2015: “The symbol versions of QR Code range from Version 1 to Version 40. Each version has a different module configuration or number of modules. (The module refers to the black and white dots that make up QR Code.) "Module configuration" refers to the number of modules contained in a symbol, commencing with Version 1 (21 × 21 modules) up to Version 40 (177 × 177 modules). Each higher version number comprises 4 additional modules per side“.

The number of bits to be stored is being identified first and then and appropriate version of QR code is being chosen. To calculate a number of bit required to store the data the formula from Figure 9 can be used.

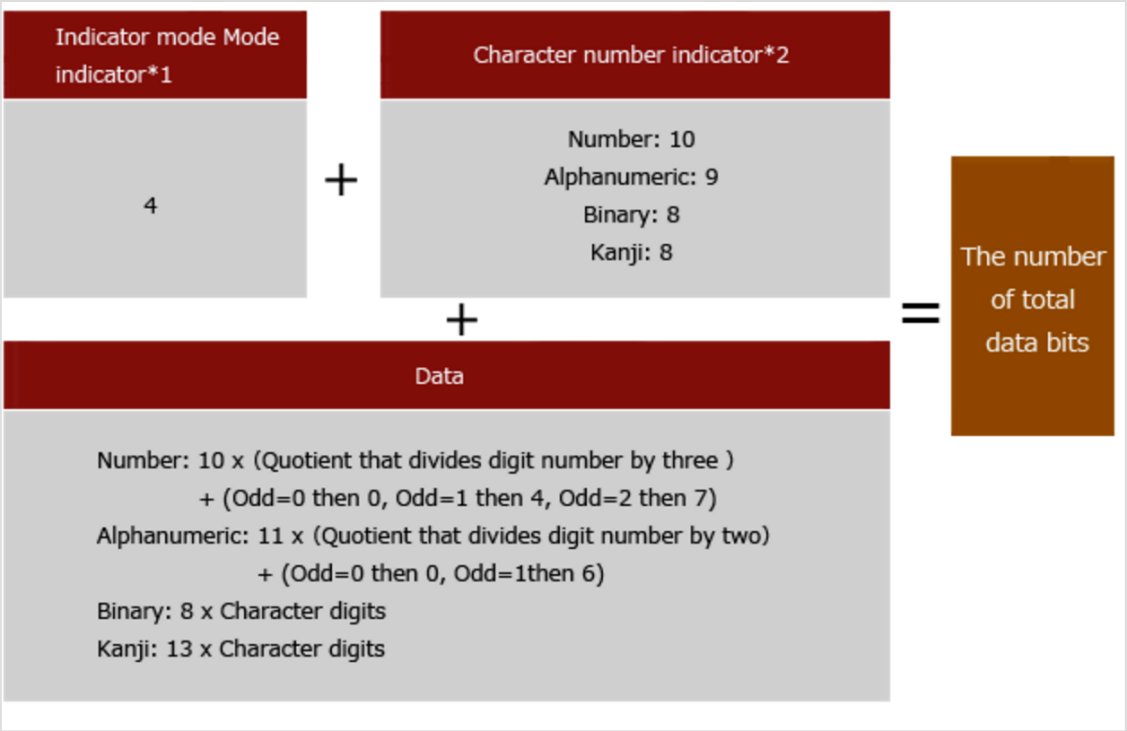


Figure 9. QR code number of bits formula (QRcode.com, 2015).

In worst case, the largest possible QR code should be generated which can store up to 23,648 bits of data (QRcode.com, 2015). Converting this value to bytes the number of 2956 is got which is basically 2.5 kilobytes of data.

Stored data

All QR code parameters are being stored as a part of JSON data structure which adds extra symbols to be coded as a part of input data. Specifically the overhead will be 6 bytes per parameter:

1. To wrap parameter key with quotes, 1 byte per quote, 2 quotes in total;
2. To wrap parameter value with quotes: 1 byte per quote, 2 quotes in total;
3. Separator between parameter's key and value, the symbol is a colon (':'), 1 byte;
4. Separator between parameters, the symbol is a comma (','), 1 byte.

Plus two bracket symbols of size 1 byte each.

Example JSON data to be used to generate QR code:

```
{
  "url": "https://accounts.google.com/AddSession",
  "ts": 1451107216,
  "seq": "ccvvccvcvcvcccvv",
  "dhP": "263794160103350151256067544128519380171",
  "dhG": "443267395101760144881989890559363738 51",
  "dhKey": "14786598240253872068320204041062808827 0",
  "secret": "206524392"
}
```

Table 1 below gives an overview on potential limitation of QR codes which in real situation hardly to be met, as it's limited to URL length which itself is being trimmed by the developed desktop component.

Parameter	Name, b	Overhead, b	Data, b	Total, b
timestamp	2	6	10	18
random sequence	3	6	16	25
Diffie-Hellman g	3	6	max. 40	49
Diffie-Hellman p	3	6	max. 40	49
Diffie-Hellman public key	5	6	max. 40	51
secret	6	6	max. 10	22
web service URL	3	6	min. 13	min. 22

Table 1. Data distribution between parameters.

According to the data provided in Table 1, it's obvious that all parameters except of web service URL are of constant size an in total produce a data of maximum size of 214 bytes, thereby in the worst case scenario the rest available 2742 or total 2956 bytes are available to be used for web service URL, which is real life is hardly achievable, considering that for example the maximum length of web page URL in Internet Explorer in limited to 2083 (Microsoft Support, 2007). On the other hand, in HTML 1.1 protocol, which is an industry standard these days, the size of URL is not limited and could be of any size (Fielding et al, 1999).

3.4. Data in communication channel

Simple special format for data transmitted in communication channel has been developed. Basically, the data is described as single line string with three entities separated by a colon:

<timestamp>:<trusted component public key>:<encrypted user credentials>

Where timestamp is an informational part of a message specifying the time when authentication process has been started by desktop component itself, and might be ignored by desktop component or used as an additional check that received data related to current authentication session.

Trusted component public key is a public encryption key generated on a side of trusted component as a part of Diffie-Hellman exchange. It is used by desktop component to evaluate private key required to decrypt received user credentials (more details on this topic could be found below in this chapter).

Encrypted user credentials is actually user credentials for requested web services but in an encrypted form. Evaluated private key is used to decrypt user credentials (more details are below in this chapter). The format of decrypted user credentials single line string is as follows:

<username>:<password>

So basically username and password are separated with a colon character, which add a limit that a colon cannot be used as a part of username. This format could be reviewed in the future and improve in case this limitation is practically limits the solution to work properly on some web services.

Data example:

1451113908213:110645926809613878557365111926517640355:U2FsdGVkX1+l1ZUVsg
AQ2J1leDnZqA4sWu7eXgBXjNh/r5dmciUgk8qliv9uRRHFxHKv/XBIPY7l+oLlc1suvQ==

Where:

- 1451113908213 is a timestamp;

- 110645926809613878557365111926517640355 is a public key provided by trusted component;

- U 2 F s d G V k X 1 + l 1 Z U V s g A Q 2 J 1 l e D n Z q A 4 s W u 7 e X g B X j N h / r5dmciUgk8qliv9uRRHFxHKv/XBIPY7l+oLlc1suvQ== is an encrypted user credentials, which when decrypted (knowing private key and other two DH parameters on desktop component side) looks like:

“uol.thesis.qr.auth.test@gmail.com:tmp1tmp2”, without wrapping quotes.

3.5. Desktop component architecture

This component is being developed considering that user credentials are not being stored on desktop computer and only available for the developed desktop component, i.e. JavaScript script, during the final stage, specifically after obtaining encrypted credentials from The Vault app and later decrypting it to use in HTML form on Google account login page.

The functionality desktop component provides is limited to the following three main functions: generate QR code, poll communication channel for data, authenticate user to target web site. Below, a detailed description of each step is given.

Generate QR code

During this step *Diffie-Hellman parameters* are being generated to be used as a part of an input data for generating QR code. Other parameters will be used to generate QR code, to be exact:

Web service URL which is basically currently opened web site URL including path part of it considering that login HTML form is available on this page. This URL is trimmed to include the path till ‘?’ (question mark) symbol is found. For example, URL like “https:// accounts.google.com/AddSession?sacu=1#identifier” will be trimmed to “https:// accounts.google.com/AddSession”. This limitation was introduced to filter out long URLs

and leave more space for other parameters. See previous section for more details on QR code limitations. On the other hand, trimmed URL is enough to provide an information on web service the authentication should be done for, actually in most cases domain name is enough for identify web service.

Timestamp, a numeric value, the number of seconds passed since 1970 Jan 1. This value represents a time when authentication process has been initiated by a user by running desktop component.

Random string sequence, randomly generated string of 16 characters. Is used in conjunction with timestamp to produce ID for current authentication session. Current authentication session ID is generated by running MD5 hash function on a string represented by leading random sequence string and subsequent timestamp number converted to string. This ID is used later on by both desktop and trusted components, first - to poll for user credentials provided by trusted component, second - to push encrypted user credential into communication channel component.

Diffie-Hellman (DH) key exchange parameters, which include three components generated “on the fly” during QR code generation process: random prime number g, random prime number p and a public key evaluated based on previously generated prime numbers plus a natural number also known as a secret key which is randomly generated at the same time. Detailed explanation of the algorithm in given below.

Poll communication channel for data

After QR code has been generated, desktop component starts to poll for a data for current authentication session in communication channel component. In the developed solution this is achieved by periodically (within few minutes until defined timeout reached) checking special file URL for existence via HTTP protocol utilising HTTP GET method. This URL is identified by current authentication session ID generated on previous step (described in previous section of this chapter).

When data is available desktop component stops polling special file URL and downloads data via HTTP protocol. After data has been downloaded, desktop component parses it,

decrypts received message and passed user credentials to a special authentication function which is described in details in the next section.

If data is not available after wait timeout is reached, desktop component notifies user that user credentials are not available, asking a user to check user credentials existence in The Vault app and Internet connection availability.

Authenticate user to target web service

Authentication to a target web site is the final step on the authentication process in the developed solution. On this step desktop component, specifically JavaScript script, starts a simulation of a real user actions by filling login HTML form on opened web page with parsed user credentials decrypted on previous step. Afterward the script submits HTML login form on a web page by simulating a click on a button.

Described functionality is possible with a help of JavaScript dynamic language which basically access web page content (HTML code) in context of currently opened web page. Described login automation functionality is specific for every supported web service, which adds reasonable level of complexity this the implementation requiring the developed software solution to be extended in case new web service should be supported. These improvements are briefly mentioned in Chapter 6.

Algorithms and data structures

Asymmetric Diffie-Hellman key exchange protocol is used to setup secure communication channel between desktop and trusted components. This protocol uses new encryption parameters every time an authentication session has been started, which means that brute force attacks are basically impossible.

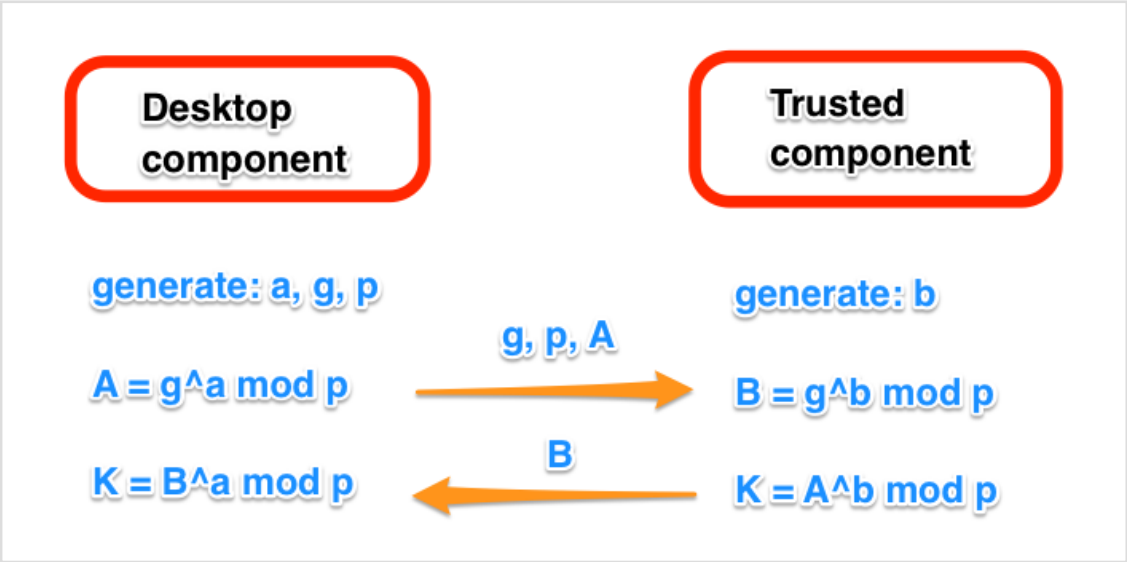


Figure 10. Explanation on Diffie-Hellman key exchange algorithm.

Mathematically, Figure 10 means that the same private key K could be calculated using special formula on both sides knowing shared DH parameters generated on both sides, which are: g, p, A, B. Where:

- g and p are randomly generated by desktop component shared 128-bit prime numbers, provided to trusted component in QR code payload;
- A is a public key of desktop component provided to trusted component in QR code payload;
- B is a public key of trusted component provided to desktop component via communication channel alongside encrypted user credentials (see details below in this chapter).

Then, symmetric AES 128 bit encryption algorithm is being used to encrypt and decrypt user credentials provided in the format like it was previously described in this chapter. AES algorithm has been selected and proposed for this project being an industrial standard nowadays. According to FIPS, 2001 “this standard may be adopted and used by non-Federal Government organizations. Such use is encouraged when it provides the desired security for commercial and private organizations“. The developed solution is not limited to use suggested algorithm and could be improved for example by using AES 256 bit algorithm to provide better security if required. On the other hand, AES 128 bit

algorithm should be enough for general purposes to provide high level of privacy for target user. More details on how AES algorithm works and could be implemented could be found in works of Moserware, 2009 and FIPS, 2001.

Simply, *AES encryption function* accepts two parameters which are 1) user credentials string, 2) private key, which is a variable K in the last formula in Figure 10.

AES decryption function on the other hand accepts similar two parameters which are: 1) encrypted user credentials string, 2) same private key, which has been used to encrypt the message (variable K in the last formula in Figure 10).

3.6. Trusted / Mobile component architecture

Trusted component also known as The Vault app is developed software application running on a user's trusted device providing the following three major functions:

- *Store user credentials* information in encrypted local database;
- *Scan QR code* generated by desktop component;
- *Deliver encrypted user credentials* to communication channel when requested.

Store user credentials

All user credentials are stored in local encrypted database which means that plain text credentials are not available unless a user unlocks the phone and runs The Vault app, which is only possible by typing device's PIN code and/or authenticating with fingerprint.

The type of data stored in local database is very simple consisting of one entity and three properties like it's described in Figure 11 below.

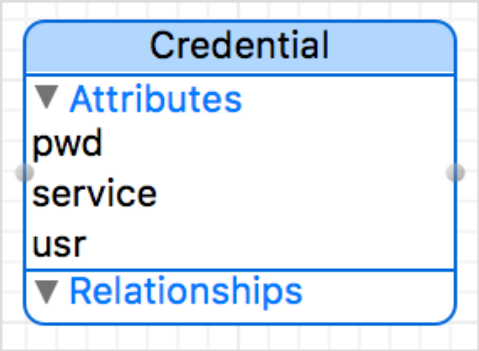


Figure 11. The Vault Local database schema.

Where:

- service is a string representing web service for which user credentials should be stored;
- usr is a string representing user login for this web service;
- pwd is a string containing user password which alongside user login provides full user credentials to specified by the first parameter web service.

This data structure might be improved in the future implementation to support multiple credential per web service, but in the developed solution single web service may have only one pair of user login and password.

Scan QR code

This functionality requires photo camera to be available on a user's trusted device. Photo camera is used to scan QR code generated by desktop component to retrieve meaningful payload from it, which is used on the next step.

Deliver encrypted credentials to communication channel

The Vault app provides an ability for its user to add and edit user credentials for any web service, but in the developed implementation for simplicity is limited to Google web service only.

Stored user credentials should be delivered to communication channel when required, which basically means when QR code has been scanned and parsed successfully. As it was previously described in this chapter, using Diffie-Hellman shared parameters private key K is being calculated based on a locally randomly generated private secret. This key is being used to encrypt user credentials in put them to communication in the format previously described in this chapter.

3.7. Securing communication channel and data

Apart of encryption algorithm used to secure transmitted data and Diffie-Hellman key exchange protocol, communication channel and data on trusted device are secured by additional methods.

Communication channel is designed to be secured with SSL, which practically means that communication requests from desktop and trusted components should be HTTPS protocol between components. This will prevent man in the middle attacks in the root (see detailed security analysis in Chapter 5). Also considering that, as it was previously mentioned, data in communication channel in encrypted using AES 138 bit algorithm.

Trusted component is also designed to be secure, which in case of this implementation means that data (user credentials) stored in The Vault app on a trusted device are encrypted. Technically, this is done be utilising standard or third-party libraries for encryption local persistent storage on a trusted device. For example, SQLite local database could be easily encrypted on iOS device by using standard Apple API (Apple Developer portal, 2015). Which practically means that a user won't be able to read plain text user credentials from local database unless he/she authenticates on the device using PIN code or biometrical data like fingerprint, even if a trusted device has been compromised, i.e. lost or theft.

4. IMPLEMENTATION (REALIZATION)

4.1. Introduction

The following assumptions should be applied to the developed solution considering that the implementation is not limited to it and could be extended in the future as a part of a separate project or research:

- Desktop component is implemented as Chrome web browser extension. This component should be installed on a public or personal desktop computer to initiate authorisation process, Operating System independent;

- Trusted component is implemented as iOS application developed to be running on iPhone 6 or above having fingerprint authentication functionality available and running iOS 8 or newer;

- Communication channel component is implemented as lightweight file sharing service implemented in PHP and running on Linux server which is available via Internet. It's implemented as a replacement for more complex file sharing services like Dropbox or Google Drive;

- Trusted device - iPhone 6 or above with fingerprint authentication functionality available, running iOS 8 or newer;

- Web service to authenticate to is limited to Google Accounts.

4.2. Requirements

The following set of skills is required to complete implementation of the proposed solution or/and to run the solution with provided deliverables:

- iOS development in Swift;

- Server-side development in PHP;

- Chrome extensions development in JavaScript;

- Apache2 server setup on Linux.

The project will require the following resources:

- Personal computer running Mac OS X;
- iPhone 6 or newer smartphone;
- Web server with PHP and Apache2 installed, available via Internet;
- Xcode IDE;
- Obj-c/Swift compiler;
- Intellij IDEA IDE;
- PHP interpreter;
- Chrome web browser;
- SSL certificate for communication channel component;
- Internet;
- Pages word processor.

No additional resources will be required to implement, run and test the defined software product.

Software libraries which has been used for implementation of this project are description in details in Appendix.C in Table 2.

Source code of all developed components is available in Appendix.E as well as configuration files are provided in Appendix.F.

4.3. Desktop component

Previously mentioned to be implemented as Chrome web browser extension, this component is running JavaScript code and utilising various JavaScript libraries available for public access in Internet. The implemented solution though could be improved to support other web browsers or to be running as a bookmarklet.

Chrome extensions development is a straightforward process with its own rules and definitions. According to Chrome Developer, 2015: “Extensions are small software programs that can modify and enhance the functionality of the Chrome browser. You write them using web technologies such as HTML, JavaScript, and CSS”, - exactly what is required by desktop component on this project.

Implementation details

Developed Chrome extension in its configuration file (manifest.json) requests permissions to peek into web pages and to be able run on every website (the most interesting parts of this file are highlighted with bold text):

```
  "background": {
    "page": "src/bg/background.html",
    "persistent": false
  },
//….
  "permissions": [
    "tabs",
    "https://*/*",
    "http://*/*"
 ],
  "content_scripts": [
    {
      "matches": [
        "https://gmail.com/*",
        "https://accounts.google.com/*"
      ],
      "css": [
        "src/inject/inject.css"
      ],
      "js": [
        "js/jquery/jquery.min.js",
        "src/inject/inject.js"
      ]
    }
  ]
```

On the other hand, according to this manifest file, the extension's icon will be available (and so clickable) only on Google Accounts page identified by two URLs: https://gmail.com and https://accounts.google.com having any address path, which basically means that URLs like https://accounts.google.com/AddSession?someparam or https://gmail.com/ will match and the extension's icon will be shown.

After web page were matched, a script from src/inject/inject.js source file will be executed which will send a message to the extension code defined in src/bg/background.js source file:

```
chrome.extension.sendMessage({"name": "ready"}, function(response) {
//…
}
```

Then this message will be received by an event listener defined in src/bg/background.js source file which will trigger an event to display this extension's icon:

```
chrome.extension.onMessage.addListener(
  function(request, sender, sendResponse) {

      if (request.name == "ready") {
         // display action_page icon in address bar if page is ready
         chrome.pageAction.show(sender.tab.id);
   //….
});
```

Then the interesting part of the developed extension starts, when a user presses extension's icon shown in the status bar (see Figure 12).

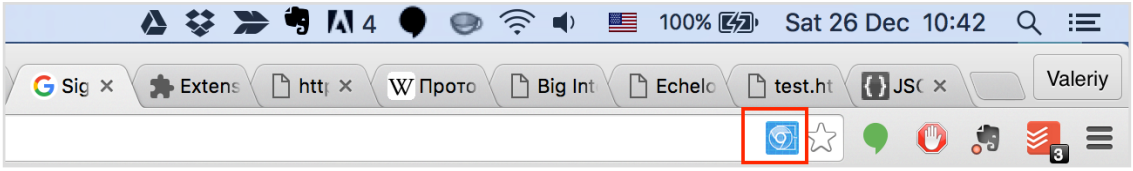


Figure 12. Chrome extension is being displayed.

User's click on the icon will trigger src/page_action/page_action.js script which on its own will request current web page URL and in on completion handle will start QR code generation process and its rendering:

```
chrome.tabs.query({'active': true, 'windowId': chrome.windows.WINDOW_ID_CURRENT},
    function(tabs){
        qrauth.tabUrlAvailable(tabs[0].url);
    }
);
```

After QR code has been rendered, communication channel starts to being polled for incoming user credentials, this is simply done by starting a timer every 3 seconds which it its handler checks expected data file for availability:

```
(function authFilePolling() {
    $.ajax({
        url: "https://chupakabr.ru/extra-test-qr-api/methods/get.php?id=" + id,
        cache: false,
        type: "GET",
        success: function (data, statusText, jqXHR) {
          //…
        error: function(data) {
          //…
```

In case the file is not available within 30 second an error message highlighted in red is being displayed right below rendered QR code suggesting a user to check credentials availability in The Vault app.

However, if the file is found in communication channel and read successfully then data parsing and decryption process is being started to retrieve user credentials in plain text:

```
        success: function (data, statusText, jqXHR) {
            qrauth.waiting = false;

            //console.info("response: " + statusText);
            //console.info("success - data: " + data);
            if (data) {
                var info = data.split(':', 3);
                if (info && info.length == 3) {
                    var ts = info[0];
                    var clientPubKey = info[1];
                    var cipher = info[2];
        //…
```

This workflow is described in details in Chapter 4. After received data is successfully parsed and decrypted, the message to start authentication process in being sent to the currently opened tab, which is expected to be the same tab when authentication process has been initiated:

```
chrome.tabs.query({active: true, currentWindow: true}, function (tabs) {
                chrome.tabs.sendMessage(
                    tabs[0].id,
                    {
                        "name": "qrauth.do",
                         "usr": creds[0],
                         "pwd": creds[1],
                    },
//….
```

The latter code will trigger event listener defined in src/inject/inject.js source file which will start authentication process simulation:

```
chrome.extension.onMessage.addListener(
        function(request, sender, sendResponse) {
          //…
                if (request.name == "qrauth.do") {
                        console.info("QrAuthVault - Authenticate on Google")
                        var usr = request.usr;
                        var pwd = request.pwd;
                        qrauth.auth.gmail(usr, pwd);
                }
         //…
```

Authentication simulation process itself is incapsulated in separate JavaScript function utilising jQuery library:

```
qrauth.auth.gmail = function(usr, pwd) {
        // enter username and press next button
        $('input[type="email"]').val(usr);
        $('input#next').delay(1000).click();

        // wait for 2 seconds and then enter password and proceed
        setTimeout(function() {
                $('input[type="password"]').val(pwd);
                $('input#signIn').delay(1000).click();
        }, 2000);
};
```

The whole authentication process itself, its pros and cons are described in details in Chapter 4. In worth though to mention that the implementation of qrauth.auth.gmail method fully depends on HTML code generated by Google Account service. This is a know limitation which is reviewed in more details in Chapters 4 and 6.

As a part of desktop component implementation, a lightweight JavaScript library (source file **js/crypto/qrvault-crypto.js**) has been developed for future use by trusted component to get rid of redundant implementation of the same functionality for working with cryptographic algorithms (both DH and AES) in trusted component. In is reviewed in more details in the implementation description of trusted component.

Chrome extension installation for testing

This section will be useful for developers and/or testers who will want to play this the developed solution trying to run it on the target desktop computer for testing. To make this happen a special URL should be loaded in Chrome web browser: **chrome://extensions/,**

which on the other hand will display Extensions configuration page like it is shown in Figure 13.

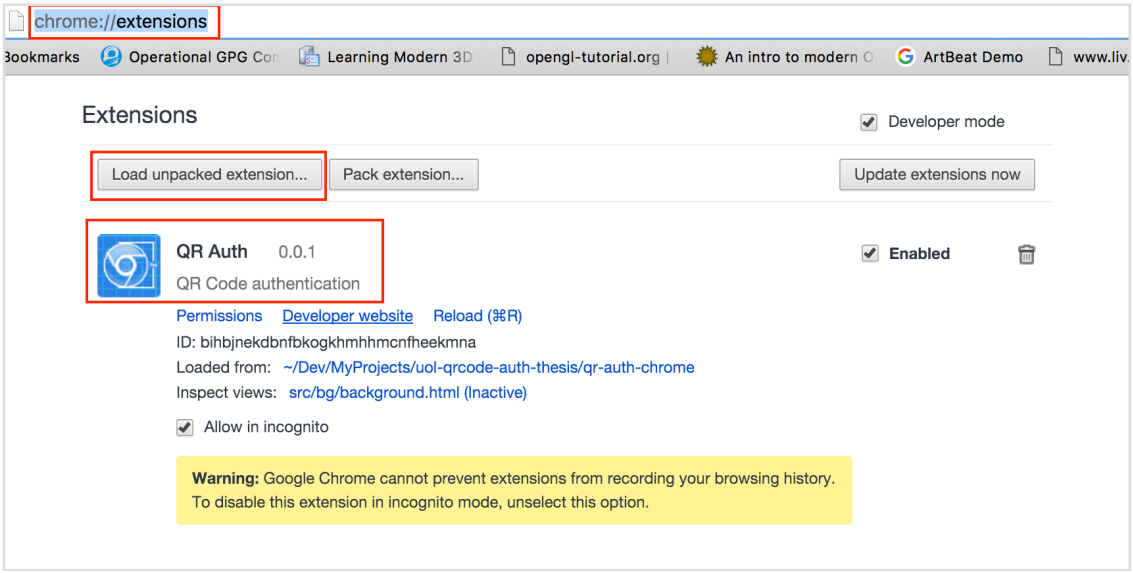


Figure 13. Chrome extensions configuration page.

By clicking on “Load unpacked extension…” it is possible to load most recent source code on the extension, in case it has been updated or wan’t loaded into Chrome web browser yet.

4.4. Trusted / Portable component

Previously mentioned to be implemented as iOS application in Swift programming language supporting iOS 8 and above.

Implementation details

An entry point of The Vault app is UIApplicationDelegate protocol implementation which is placed in AppDelegate.swift file:

```
@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?
//….
```

In this project’s implementation, Application Delegate file also contains a code for initialising local encrypted data storage and its persistent context:

```
    lazy var managedObjectContext: NSManagedObjectContext = {
      //…

    lazy var persistentStoreCoordinator: NSPersistentStoreCoordinator = {
      //…

    lazy var managedObjectModel: NSManagedObjectModel = {
      //…

    lazy var applicationDocumentsDirectory: NSURL = {
      //…
```

The important part in this Application Delegate persistent storage initialization code is SQLite local data storage encryption, which is achieved using standard Foundation framework provided by Apple:

```
  lazy var persistentStoreCoordinator: NSPersistentStoreCoordinator = {
     //…
      do {
          try coordinator.addPersistentStoreWithType(NSSQLiteStoreType,
              configuration: nil, URL: url,
              options: [NSPersistentStoreFileProtectionKey: NSFileProtectionComplete])
     //…
```

After Application Delegate has been initialized, application's main window in being created displaying the first view from the main storyboard which contains just one button "Authenticate" to start fingerprint local authentication process to access secured functionality provided by The Vault app (source file: ViewController.swift):

```
class ViewController: UIViewController {

    var context = LAContext()
    //…

    @IBAction func authenticate() {
        print("++ authenticate")

        if context.canEvaluatePolicy(LAPolicy.DeviceOwnerAuthenticationWithBiometrics,
error:nil) {

context.evaluatePolicy(LAPolicy.DeviceOwnerAuthenticationWithBiometrics,
              localizedReason: "Logging in with Touch ID",
              reply: { (success : Bool, error : NSError? ) -> Void in
                  dispatch_async(dispatch_get_main_queue(), {
                      if success {
                          self.performSegueWithIdentifier("auth_done", sender: self)
                      }

                      if error != nil {
    //…
```

On successful authentication the second and the last view is being displayed asking a user to add or edit his/her user credentials for Google Accounts as well as providing a button to scan QR code (source file: EditCredsViewController):

```
class EditCredsViewController: UIViewController {
    private let fileShareUrl = "https://chupakabr.ru/extra-test-qr-api/methods/put.php"
    //…
    private var jsContext: JSContext! = nil
    //…
    @IBAction func scan() {
    //…
    @IBAction func saveAndLock() {
    //…
```

In the implementation of this project the simplicity factor was taken into consideration thus the idea of reusing previously implemented shared JavaScript library overweighted native API implementation of encryption methods from scratch in Swift language, resulting in the running JavaScript engine in the context of the implemented application using JavaScriptCore standard framework provided by Apple:

```
import JavaScriptCore
class EditCredsViewController: UIViewController {
    //…

    override func viewDidAppear(animated: Bool) {
        super.viewDidAppear(animated)

        // Load JS libraries into JS context
        if jsContext == nil {
            do {
                jsContext = JSContext()
                try jsContext.evaluateScript(String(contentsOfURL: jsFile("core"),
encoding: NSUTF8StringEncoding))
                try jsContext.evaluateScript(String(contentsOfURL: jsFile("enc-base64"),
encoding: NSUTF8StringEncoding))
                try jsContext.evaluateScript(String(contentsOfURL: jsFile("md5"), encoding:
NSUTF8StringEncoding))
                try jsContext.evaluateScript(String(contentsOfURL: jsFile("evpkdf"), encoding:
NSUTF8StringEncoding))
                try jsContext.evaluateScript(String(contentsOfURL: jsFile("cipher-core"),
encoding: NSUTF8StringEncoding))
                try jsContext.evaluateScript(String(contentsOfURL: jsFile("aes"), encoding:
NSUTF8StringEncoding))
                try jsContext.evaluateScript(String(contentsOfURL: jsFile("BigInt"), encoding:
NSUTF8StringEncoding))
                try jsContext.evaluateScript(String(contentsOfURL: jsFile("qrvault-crypto"),
encoding: NSUTF8StringEncoding))

    //…
    private func jsFile(filenameWithoutExt: String) throws -> NSURL {
        guard let file = NSBundle.mainBundle().URLForResource(filenameWithoutExt,
withExtension: "js", subdirectory: "js") else {
            throw NSError(domain: "JsFileLoad", code: 1, userInfo: ["file":
filenameWithoutExt])
        }
```

```
    return file
}

//…
```

In short, JavaScriptCore framework allows to load and run JavaScript file and code “on the fly” while the application is running. The number of files and lines of code is only limited by the available memory of iOS device, which in most cases is not an issue. The code above loads a set of JavaScript files (same as we use in desktop component implementation) into memory of The Vault app so that all functions and data structures provided by it could be run from Swift code, like in the following example where symmetric private key is being calculated by Swift application using the same methods which are used in desktop component implementation:

```
private func uploadCredentials(id id: String?, loginInfo: LoginInfo?,
    timestamp ts: Int?, dh: DhInfo, lifetimeSec ttlSec: Int = 120) throws
{
    guard let loginInfo = loginInfo, id = id, ts = ts else {
        throw NSError(domain: "Login information, id or timestamp not found", code: 1,
userInfo: nil)
    }

    // Generate local secret, public and shared keys
    let secret = genSecretKey()
    let publicB = evalPublicKey(g: dh.g, secret: secret, p: dh.p)
    let sharedKey = evalSharedPrivateKey(publicA: dh.pubKey, secret: secret, p:
dh.p)
    let cipher = encryptMessage("\(loginInfo.username):\(loginInfo.password)",
sharedKey: sharedKey)

//…
private func evalSharedPrivateKey(publicA publicA: String, secret: String, p: String) ->
String {
    let jsFunc = getJsCryptoFunc("evalPrivKeyFromStr")
    let privateKey: JSValue = jsFunc.callWithArguments([publicA, secret, p])
    return privateKey.toString()
}

//…
private func getJsCryptoFunc(funcName: String) -> JSValue! {
    return
jsContext.objectForKeyedSubscript("qrauth").objectForKeyedSubscript("crypto").objectFo
rKeyedSubscript(funcName)
}
```

After credentials has been encrypted they are ready to be pushed to communication channel component for further delivery, which is done by the following code snippet:

```
private let fileShareUrl = "https://chupakabr.ru/extra-test-qr-api/methods/put.php"

//…
```

```
    private func uploadCredentials(id id: String?, loginInfo: LoginInfo?,
        timestamp ts: Int?, dh: DhInfo, lifetimeSec ttlSec: Int = 120) throws
    {
        //…
        let cipher = encryptMessage("\(loginInfo.username):\(loginInfo.password)",
sharedKey: sharedKey)
        let dataStr = "\(id):\(ts):\(publicB):\(cipher)"
        print("Upload credentials: \(dataStr)")

        if let data = dataStr.dataUsingEncoding(NSUTF8StringEncoding) {
            print("Uploading encrypted credentials to \(fileShareUrl)")
            Alamofire.upload(.PUT, fileShareUrl, data: data)
    //…
```

This is the final step on The Vault app side which basically transits the workflow back to desktop component to finish authentication process.

Apart of the mentioned classes and methods, a simple entity object was introduced as a part of persistent store implementation (source file: Credential.swift):

```
import CoreData

extension Credential {
    @NSManaged var usr: String?
    @NSManaged var pwd: String?
    @NSManaged var service: String?
}
```

This class is used by Managed Object Context to map the data stored in local database to its programmatic representation (see Figure 11).

The Vault app installation for testing

This section will be useful for developers and/or testers who will want to play this the developed solution trying to run it on the target trusted device for testing. It is required to have Xcode IDE to be installed and available on a development computer running on Max OS X.

The first step of The Vault app installation process is to navigate to the project's source directory and run CocoaPods command in from command line:

```
cd <replace with QR Vault app source code directory>
sudo gem install cocoapods
pod install
```

Then to install implemented The Vault app it's needed to open project file or source code directory from Xcode like it is shown in Figure 14 below.

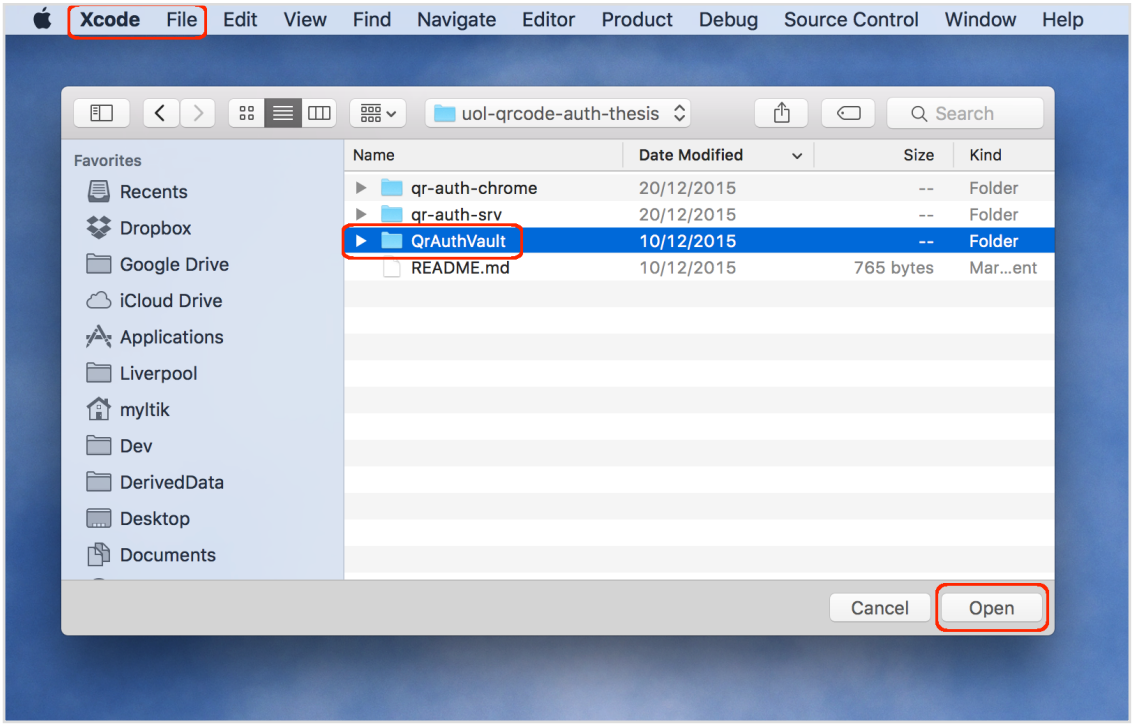


Figure 14. Opening The Vault app project.

This will open project in Xcode IDE which will make it possible to proceed to the final step on the installation process which consists of project compilation and runing the application in debug mode on connected iPhone device like it is shown in Figure 15.

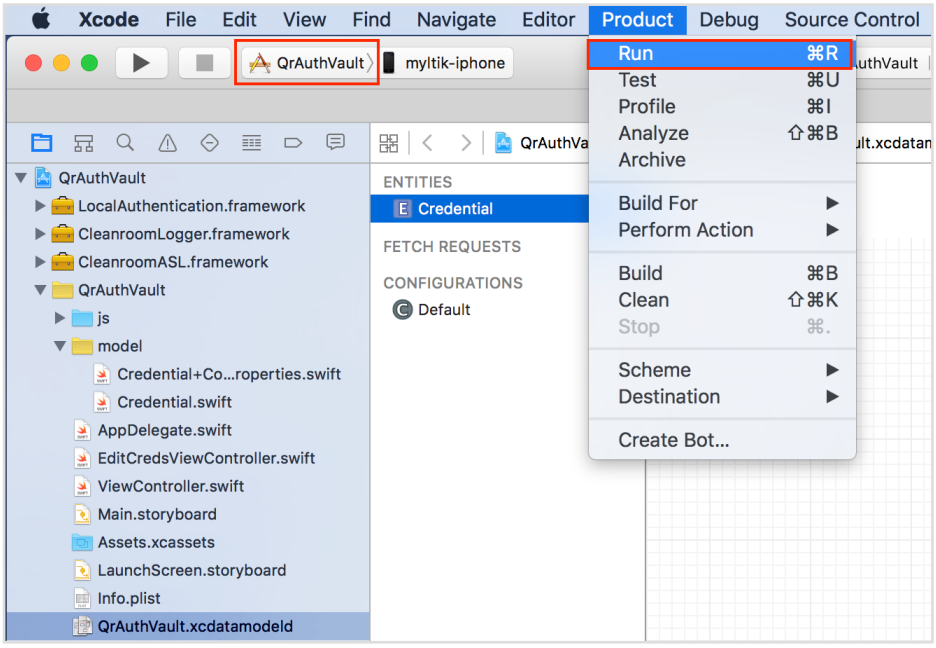


Figure 15. Running The Vault app in Xcode IDE.

4.5. Communication channel component

Previously mentioned to be implemented as lightweight file sharing service developed in PHP and is expected to be installed on Apache2 web service by copying root directory of lightweight file sharing service sources into /var/www directory on Linux server.

Implementation details

This lightweight file sharing service is able to store files up to 512 bytes. This limitation is configurable via a constant defined in the source code, file entry.php:

```
define(QA_MAX_FILE_SIZE, 512);
```

The file sharing service itself is implemented using standard PHP function providing two methods for pulling and pushing encrypted data streams. Both methods are expected to be ran over HTTP protocol while push (put.php) method should be used utilising HTTP PUT method and pull (get.php) method should be used utilising HTTP GET method. HTTP request type is being check by the implemented functions like in this example:

```
if ($_SERVER['REQUEST_METHOD'] != "GET") {
    echo "ERR invalid HTTP method\n";
    http_response_code(405);
    exit(1);
}
```

In case of invalid HTTP method an error HTTP response in being returned with a code of 405 meaning “illegal arguments” error.

Push method accepts input stream as an only one argument, which basically contains text file with encrypted credentials in the format described in Chapter 4.

Pull method on the other hand accepts ID parameter as its input, which is basically ID of an authentication session calculated with an algorithm described in Chapter 4.

The implementation operates with text files instead of a local database which on the other hand might struggle because of flow in/out file system operations, this implementation details should be considered as a simplification of the developed solution keeping in mind a possibility of improve the realisation of this component by using for example in-memory

credentials storage Map or any other suitable in-memory data structure. File operations are provided by standard PHP functions like the following:

```
fopen($filePath);
fgets($file, $bytesToRead);
fread($file, $bytesToRead);
fwrite($file, $data, $dataSizeBytes);
file_exists($filePath);
fclose($file);
```

Server setup

Every file in this service could be stored for up to 5 minutes, after that time the files are being removed from the filesystem. This automatic file removal functionality is achieved by running standard Linux utility which is called *cron*, the following two lines should be added to /etc/crontab file:

```
# Cronjob setup to remove credentials file every 5 minutes:
*/5 * * * *    www-data        find /mnt/qrvault/*.txt -type f -mmin +5 -exec rm -f {} \;
```

Having /mnt/qrvault directory to be created as a part of server deployment process. File permissions to this directory should be granted to www-data user which is a default Linux user for apache2 web server by running the following commands:

```
sudo mkdir -p /mnt/qrvault
sudo chown -R www-data /mnt/qrvault
sudo chgrp -R www-data /mnt/qrvault
```

Apache2 server is expected to be setup with PHP support which could be done by running the following command line script:

```
apt-get install php5 libapache2-mod-php5
```

Moreover Apache2 server should be setup to enable SSL support for the implemented lightweight file sharing service which could be done by adding the following lines into web site's configuration file (Comodo Group inc., 2015):

```
SSLEngine on
SSLCertificateFile /etc/ssl/certs/2_chupakabr.ru.crt
SSLCertificateKeyFile /etc/ssl/private/3_chupakabr.ru.key
SSLCertificateChainFile /etc/ssl/certs/1_root_bundle_chupa.crt
```

where the last three lines should contain valid paths to SSL certificate files received from trusted authority.

Trusted SSL certificate could be obtained via SSL certificate providers like GoDaddy.com, ssl.comodo.com and others. In the implementation of this project free SSL certificates provider comodo.com has been chosen to generate free less secure but nevertheless working SSL certificate (comodo.com, 2015).

In the end the following command needs to be run to restart Apache2 web server instance:

```
sudo service apache2 restart
```

Complete configuration files for Apache2 web service are provided in Appendix.F.

4.6. Putting it all together for testing

After every of all three components are installed on setup on appropriate device, the developed solution could be ran and tested with created test user credentials provided in Appendix.B. And example of such test run of the whole workflow is available in Appendix A. as screenshots.

5. RESULTS AND EVALUATION

In the evaluation process of this project only the author of this study took part which includes running of a complete workflow of the defined QR code based authentication process multiple times with and without valid credentials involved. The results of the evaluation process could be found in Appendix.A. The results of the evaluation of the developed solution should be considered as meeting all criteria predefined in theoretical part of this study in Chapters 1, 2 and 3. Produced IT artefact has been implemented considering all assumptions and boundaries defined in Chapter 4 (Analysis and Design).

This chapter gives a detailed description of security analysis provided during evaluation of the developed software solution and relevant conclusions are made.

5.1. Security considerations

In this chapter security analysis is given for the most popular and relevant types of attacks on the developed authentication solution.

Trusted device theft or loss

Affects: trusted component.

Third type of attack is expected to be one of the most frequently used attacks by malefactors, on the other hand, the developed software solution is securely protected unless a criminal knows trusted device's PIN code and/or has an access to the owner's fingerprint or other biometrical data required for authentication on the phone. Theoretically, it should be possible for a thief to access a trusted device's local file system to access download local database file, however the implementation provided in this study is safe to this type of attack as of using encrypted local database file.

A simple analysis of this attacking use-case has been analysed in this section. But first back to Chapter 4 which defines that trusted component software is expected to encrypt local database, the implementation should allow a user to access local database only if he/she is authenticated on a target phone and the app is running.

As a part of a test run of the developed solution, a security analysis has been conducted running Xcode with trusted device connected to development computer, so trusted device's local filesystem is accessible on development computer via utilities provided by Xcode like it is shown in Figure 16 below.

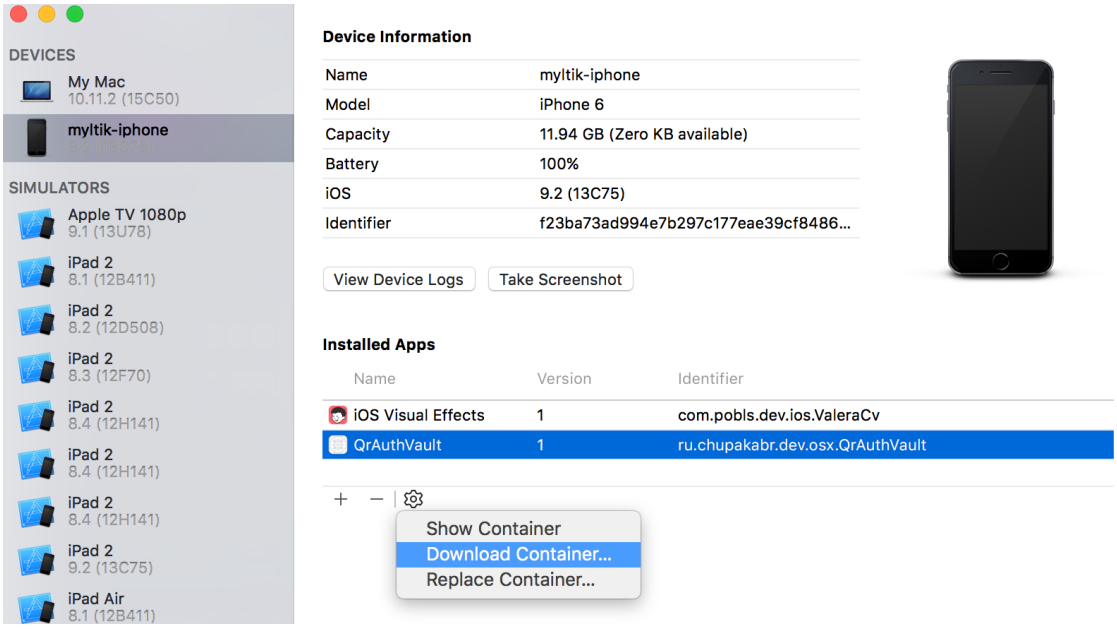


Figure 16. Filesystem export in Xcode from real iPhone device.

Filesystem export on locked phone:

```
[myltik:Downloads myltik$ ls -alh 1_locked_ru.chupakabr.dev.osx.QrAuthVault\ 2015-12-21\ 22\:30.16.876.xcappdata/AppData/Documents/
total 64
drwxr-xr-x+ 3 myltik  staff   102B 26 Dec 13:56 .
drwxr-xr-x+ 5 myltik  staff   170B 26 Dec 13:56 ..
-rw-r--r--+ 1 myltik  staff    32K 26 Dec 13:56 QrAuthVault.sqlite-shm
[myltik:Downloads myltik$
```

Figure 17. File system export of locked device.

Local database file is unavailable in filesystem export as it shown in Figure 17 on locked trusted device.

Filesystem export from unlocked phone with The Vault app running:

```
[myltik:Downloads myltik$
[myltik:Downloads myltik$ ls -alh 3_unlocked_with_app_ru.chupakabr.dev.osx.QrAuthVault\ 2015-12-21\ 22\:30.16.876.xcappdata/AppData/Documents/
total 152
drwxr-xr-x+ 5 myltik  staff   170B 26 Dec 13:59 .
drwxr-xr-x+ 5 myltik  staff   170B 26 Dec 13:59 ..
-rw-r--r--+ 1 myltik  staff    24K 26 Dec 13:59 QrAuthVault.sqlite
-rw-r--r--+ 1 myltik  staff    32K 26 Dec 13:59 QrAuthVault.sqlite-shm
-rw-r--r--+ 1 myltik  staff    16K 26 Dec 13:59 QrAuthVault.sqlite-wal
[myltik:Downloads myltik$
[myltik:Downloads myltik$ grep "uol.thesis.qr.auth.test" 3_unlocked_with_app_ru.chupakabr.dev.osx.QrAuthVault\ 2015-12-21\ 22\:30.16.876.xcappdata/AppData/Documents/*
Binary file 3_unlocked_with_app_ru.chupakabr.dev.osx.QrAuthVault 2015-12-21 22:30.16.876.xcappdata/AppData/Documents/QrAuthVault.sqlite-wal matches
[myltik:Downloads myltik$
[myltik:Downloads myltik$ grep "non.existing.string" 3_unlocked_with_app_ru.chupakabr.dev.osx.QrAuthVault\ 2015-12-21\ 22\:30.16.876.xcappdata/AppData/Documents/*
[myltik:Downloads myltik$
```

Figure 18. File system export of unlocked device.

Results displayed on Figure 18 demonstrate that local database is available for read on unlocked device, which means that it's theoretically possible to break down encrypted

local database, which on the other hand is very hard to be achieved considering high security standards defined by modern operating systems like iOS and Android.

The implementation of the current solution could be improved by initializing encrypted data store and its context only after a user authenticated in the app as well as he/she authenticated on a trusted device. At the moment this initialization on a local database is done during application start, which leaves a tiny security breach in the trusted component. This is one of the first to have improvement to be done in the future works or related studies.

Phishing attacks

Affects: desktop component, trusted component.

Nobody is protected against this type of attacks, which on the other hand are manageable by a user itself, but running the solution on a trusted websites only.

Another implementation of this type of attack for proposed solution is to force target user to scan QR code specially generated by a criminal, who receiving the data on the other side might emulate the whole process of desktop component and force The Vault app to push private user credentials to communication channel.

This type of attack is potentially very risky, but when users is fully controlling the websites he is visiting and QR code he is scanning via The Vault app, then this type of attacks are highly inefficient.

Man in the middle attacks

Affects: trusted components, desktop component.

This type of attack is hardly possible because of the nature of the developed solution, specifically because of using two separate secure connections to communication channel from both desktop and trusted components. This means that a criminal should substitute both secure connections running over SSL, however this type of attack could be implemented for example if a user running both desktop and trusted components on the

same WiFi hotspot controlled by a criminal, who can inject into both connections and changing inbound and outbound data in it. However, this is fairly hard to achieve because of a nature of SSL protocol.

Trusted third party

Affects: none.

This type of attack is not applicable to the implemented software solution as no trusted third parties are involved in authentication process. User credentials information is being stored on a user's trusted device which is considered to be trusted as per initial definitions.

Fingerprint forgery

Affects: trusted component.

This type of offline attack on authentication systems is not frequently used by malefactor but is possible to run against the solution developed in this study. However, there are several protocols available to protect against this type of attacks as mentioned in the work of Wang, 2012.

Key logger malicious software

Affects: desktop component, trusted component.

Desktop component is indirectly affect by this type of attack assuming that a malicious software is installed on a public or a personal desktop computer which has an access to web browser's JavaScript execution context of a Chrome extension.

On the other hand, trusted component is affected by key logger malicious software during the phase on entering / updating user credentials by a user himself. However, this type of software is not popular by criminals due to the fact that it's hard to avoid and chea pre-market approval process as well as high security standard provided by multilayered

security architecture and very limited APIs provided by mobile operating systems like iOS or Android.

5.2. Deliverables

In this section main deliverables of this study which has been identified in Chapter #1 will be provided.

Source code. Appendix.E. provides full source code for all components of the implemented technical solution as well a link to full source code on Github including projects' structures as well as Xcode and IntellijI DEA project files: https://github.com/chupakabr/uol-qrcode-auth-thesis.

Web server sample configuration files. Appendix.F. provides complete configuration files for Apache2 web server with SSL and PHP support running on Linux. Required to developed run lightweight file sharing service.

Lightweight file sharing service URL. Appendix.D. The server has been deploy on the author's personal server using publicly available domain, and is available at:

- Upload file: https://chupakabr.ru/extra-test-qr-api/methods/get.php
- Download file: https://chupakabr.ru/extra-test-qr-api/methods/put.php

The service will be available for few months since the submission of this work for reviewal by committee of University of Liverpool, which will happen on 2015 Dec 28.

5.3. Working solution

As the result of the project, the working technical solution was implemented as defined in Chapter 4 (Analysis and Design) of this work. The solution meeting the criteria which has been initially defined in the proposal for this dissertation and later on in extended format in the second chapter of this work as well as in Chapters 1 and 3.

All security and private prerequisites were met as proved in first section of this chapter. Deep security analysis were given to all of the developed components as well as to the whole solution itself.

The developed solution has been ran and tested with created test user credentials provided in Appendix.B. and screenshots of the working solution running the whole workflow is given in Appendix.A.

6. CONCLUSIONS

6.1. Lessons Learned

This study and the developed software solution proves that matrix barcodes, specifically QR codes, could be used for web services authentication on existing web services without trusted third parties involved.

The great benefit of the proposed solution is that existing web services doesn't require to be updated and the only work is to be done on client-side by installing special software on both desktop and personal (and portable, ex. smartphone) user's devices. Another major benefit of the developed solution is that user credentials are stored on user's personal trusted device only and are not being trusted to any third parties.

The major problem of the implemented solution is phishing attack in which a user is basically forced to scan QR code generated by a malefactor. This may lead to a leak of users credentials quite massively.

6.2. Academic Application and Limitations

This study provides new ideas to the existing authentication systems researches as well as proofs the possibility of adopting matrix barcode based authentication systems to existing Internet infrastructure requiring no trusted third parties to be involved.

The project might be used as a practical reference of asymmetric cryptographic algorithm adopting, specifically Diffie-Hellman algorithm implementation, used in a wide range of areas including but not limited to authentication systems.

The implemented solution is not limited to QR codes only and is applicable to a wide range of matrix barcode implementations. On the other hand, QR codes can contain data of a limited size depending on a size of generated image like it's described in Chapter 3. Analysis and Design.

6.3. Business Application and Limitations

This project might be used as a starting point for businesses to implement complete software solution for QR code based authentication. The solution is not limited to any specific web services and could be extended and improved to add a support for a wide range of existing web services as well as provide an ability to end users to created custom authentication schemes “on the fly” to provide easy and transparent way for end users to adopt the software for less know web services.

The developed solution is limited to a particular set of technologies which on the other hand could be easily replaced by a different set of programming languages, libraries and target device, or the solution could be just improved to support multiple platforms not limited to iPhone and Chrome web browser.

The implemented solution might become an industrial standard in the future if a complete product developed based on the ideas and/or software solution presenting in this work. The complete product is expected to provide public access to the developed Chrome Extension (https://chrome.google.com/webstore/category/extensions) as well as to the developed iOS application from Apple Store (https://appstore.com) considering communication channel has been deployed and setup properly on publicly available server.

6.4. Recommendations / Prospects for Future Research / Work

The developed authentication software solution is fully functional though requiring further improvements to become a complete product, but certainly the developed solution could be used as a starting point for a real product implementation.

The further research is suggested to be conducted to improve developed solution in terms of easier and more transparent support of different existing or even currently developing web services, for example by introducing new protocol for authentication schemes on web services server-side or by introducing user friendly authentication recording mechanism so that a user itself could setup authentication process “on the fly” just once to make QR code based authentication available on less known web service

and later on make it available for other users by sharing newly created authentication scheme.

Another, but very important suggestion for future research is to improve the developed solution's resistance to phishing attacks when a user if forced to scan QR code generated by a malefactor. This is a very challenging topic though, which requires a whole separate research to be conducted.

As an improvement of the developed software it's possible to replace web browser extension with publicly available bookmarklet or similar JavaScript script to allow user to access the functionality without additional software to be installed on a public or a personal desktop computer.

Usability of the implemented solution might also be analysed to have better understanding on the need of this type of authentication mechanism and to improve user experience in general.

REFERENCES CITED

Apple Developer portal, “NSFileManager”. Available at https://developer.apple.com/library/ios/documentation/Cocoa/Reference/Foundation/Classes/NSFileManager_Class/index.html#//apple_ref/c/data/NSFileProtectionCompleteUntilFirstUserAuthentication (accessed on 2015 Dec 22).

J. Bonneau, C. Herley, P. Van Oorschot, and F.Stajano. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In Proc. IEEE Security and Privacy, pages 553–567. IEEE, 2012.

B. Borchert and M. Günther. Indirect NFC-login. In ICITST, pages 204–209, 2013. Available at http://www.ekaay.com/press/Pressekopien/NFC-Login.pdf (accessed on 2015 Oct 20).

Chrome Developer. What are extensions? Available at https://developer.chrome.com/extensions (accessed on 2015 Dec 26).

J.J.L. Cobos and P.C. De La Hoz. Method and system for authenticating a user by means of a mobile device, September 2012. US Patent 8,261,089.

comodo.com. Available at https://ssl.comodo.com (accessed on 2015 Dec 15).

Comodo Group inc. “Certificate Installation: Apache & mod_ssl”. Available at https://support.comodo.com/index.php?/Default/Knowledgebase/Article/View/637/37/ (accessed on 2015 Dec 15).

D.B. Desoto and M.A. Peskin. Login using QR code, August 22 2013. US Patent 20,130,219,479.

B. Dodson, D. Sengupta, D. Boneh, and M.S. Lam. Secure, consumer-friendly web authentication and payments with a phone. In Mobile Computing, Applications, and Services, pages 17–38. Springer, 2012.

eKaay. Available at http://www.ekaay.com/ (accessed on 2015 Oct 22).

R. Fielding, UC Irvine, J. Gettys et al. Hypertext Transfer Protocol -- HTTP/1.1. 1999. Available at ftp://ftp.isi.edu/in-notes/rfc2616.txt (accessed on 2015 Dec 25).

FIPS. Advanced Encryption Standard (AES), 2001. Available at http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf (accessed on 2015 Dec 26).

S. Gibson. Secure quick reliable login SQRL. Available at https://www.grc.com/sqrl/sqrl.htm (accessed 2015 Dec 20).

E. Grosse and M. Upadhyay. Authentication at scale. Proc. IEEE Security & Privacy, 11(1):15–22, 2013.

Microsoft Support, 2007. Available at https://support.microsoft.com/en-us/kb/208427 (accessed on 2015 Dec 25).

Moserware, “A Stick Figure Guide to the Advanced Encryption Standard (AES)”, 2009. Available at http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html (accessed on 2015 Dec 15).

MyDigiPass. Available at https://www.mydigipass.com/ (accessed on 2015 Oct 27).

QRcode.com. Available at http://www.qrcode.com/en/about/version.html (accessed on 2015 Dec 25).

E. Rescorla. Diffie-Hellman Key Agreement Method. 1999. Available at http://tools.ietf.org/html/rfc2631 (accessed on 2015 Dec 15).

TiQR. Available at https://tiqr.org (accessed on 2015 Dec 14).

Y. Wang. Password protected smart card and memory stick authentication against off-line dictionary attacks. In SEC IFIP AICT 376, pages 489–500, 2012.

APPENDICES

Don’t forget to include the approved version of your DS Proposal in an appendix.

A. TECHNICAL SOLUTION REAL WORKFLOW IN SCREENSHOTS

This appendix provides a real-life example of the implemented solution based on screenshots taken while running the complete authentication workflow for Google Accounts (https://accounts.google.com/AddSession?sacu=1#identifier).

A.1. Initiation

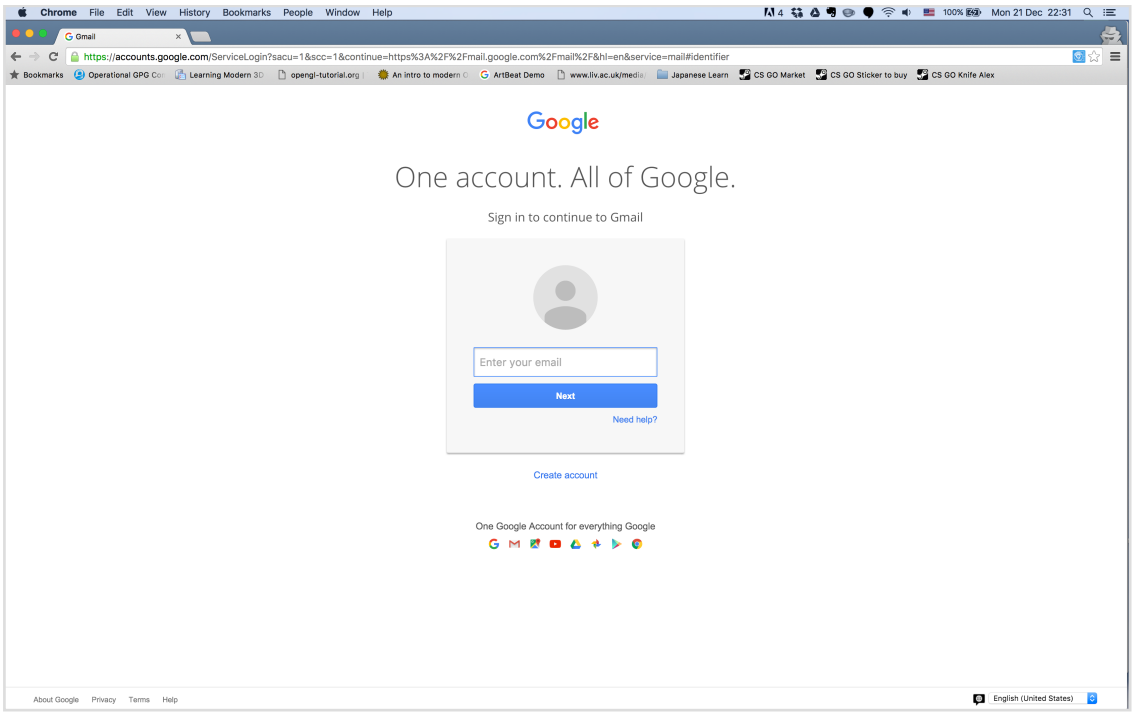


Figure 19. User opens Google login page.

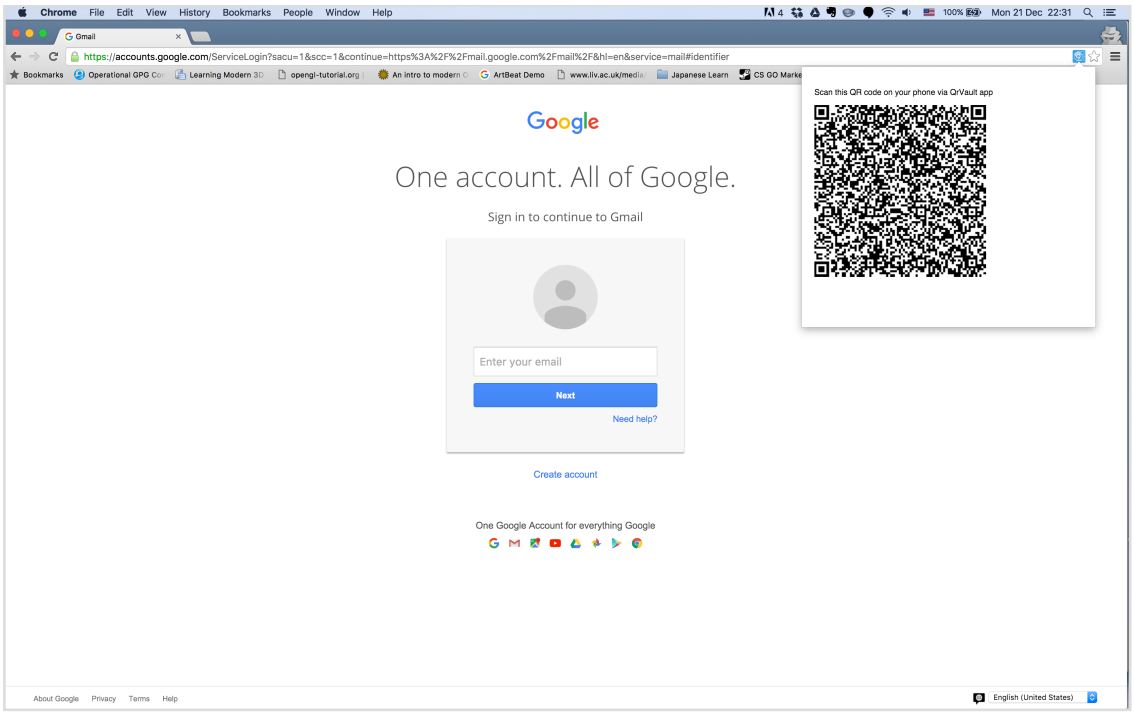
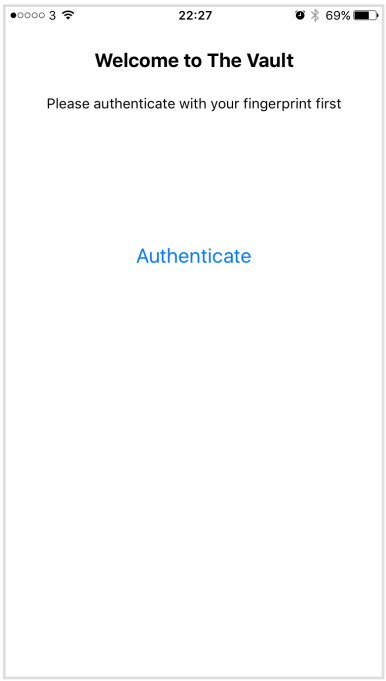


Figure 20. User clicks on QrVault Chrome extension icon in address bar and QR code is being generated and then displayed to be scanned by The Vault app.

A.2. Accessing credentials vault



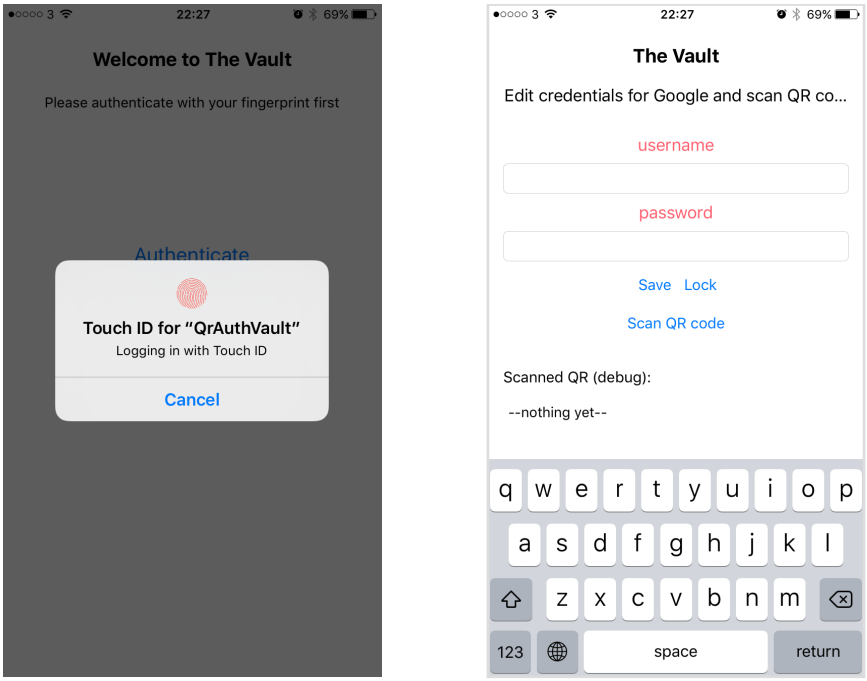


Figure 21. Use opens The Vault app and authenticates to it using his/her biometrical data, i.e. fingerprint.

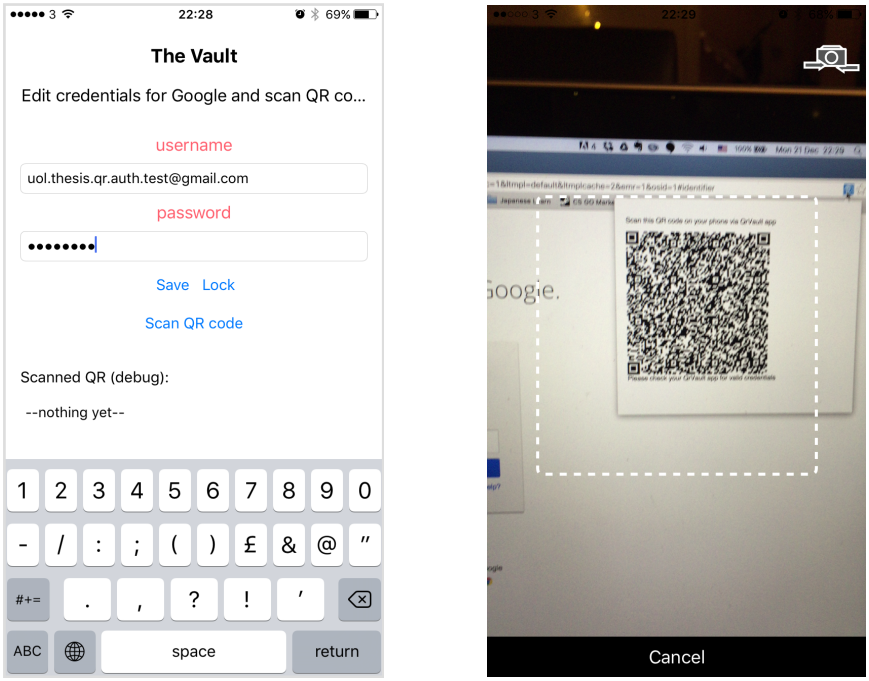


Figure 22. User enters his/her Google credentials if not exists in The Vault app yet and stores it in local database by pressing "Save" button. Then user scans QR code to proceed with authentication.

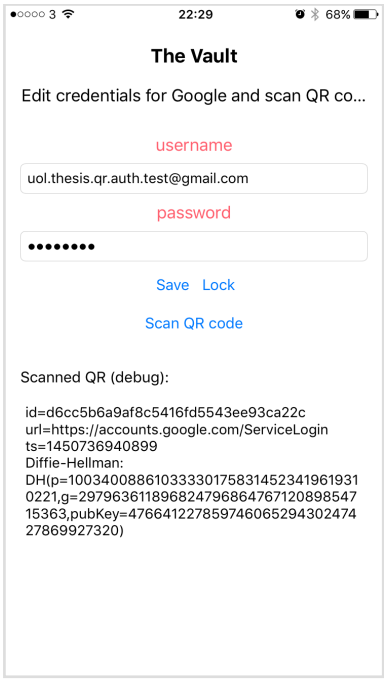
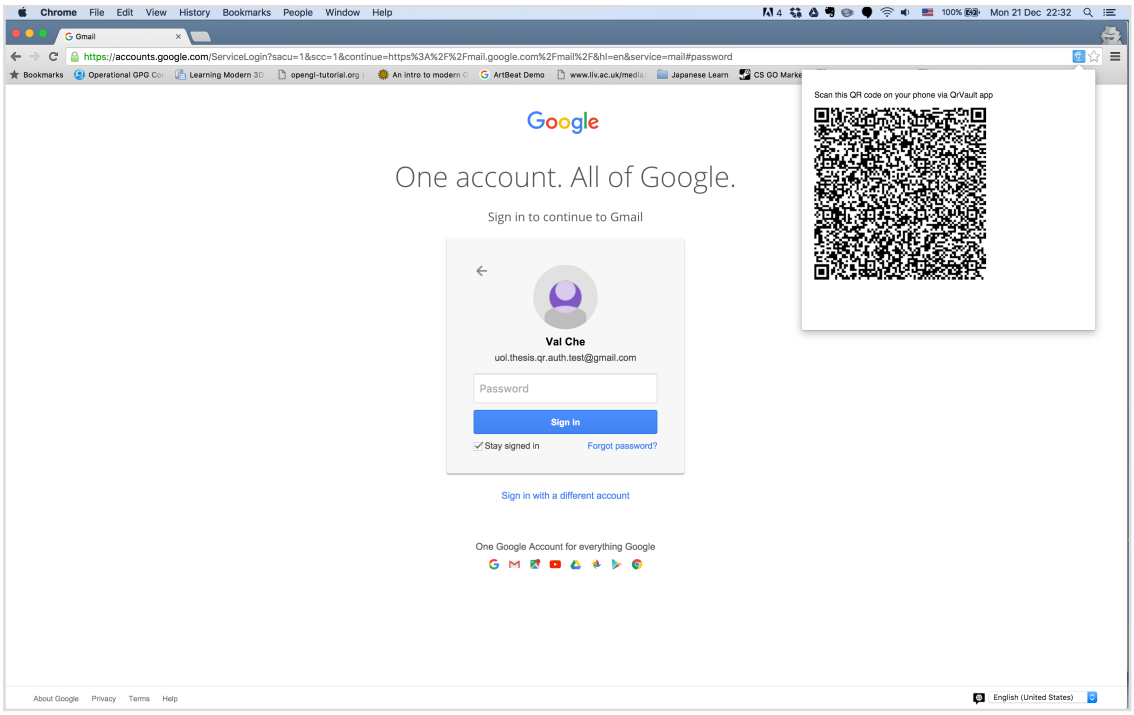


Figure 23. QR code has been scanned and user can see debug message in the app.

A.3. Authentication



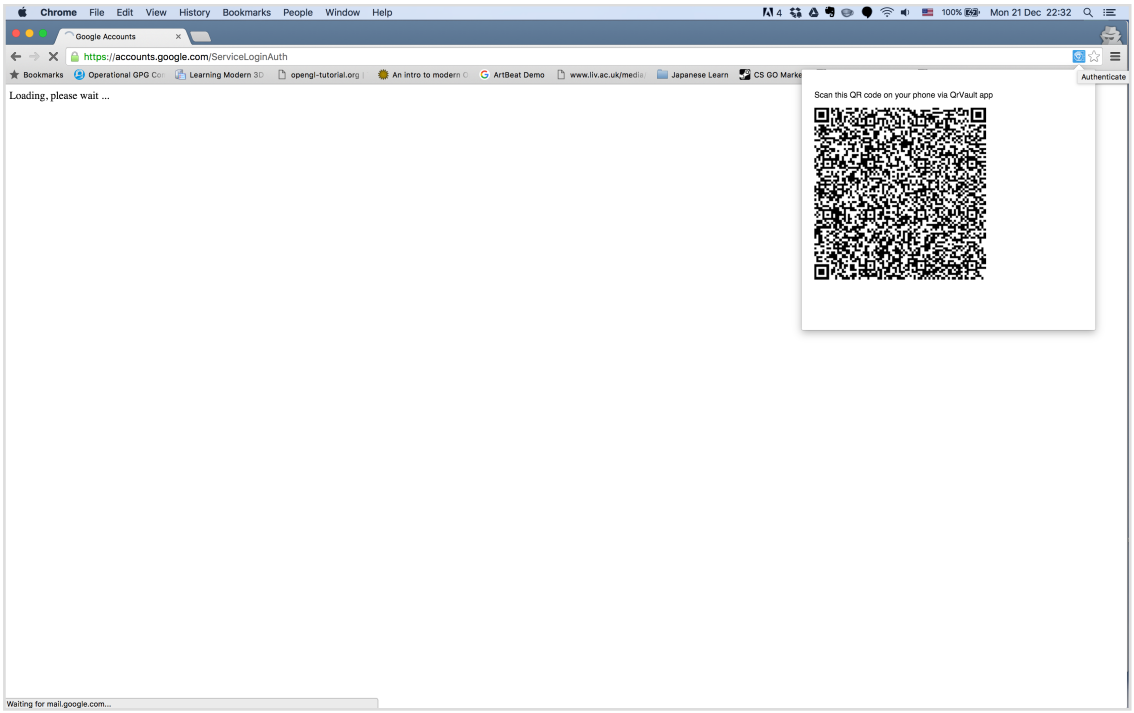


Figure 24. Authentication is being completed by Chrome extension after a data from The Vault app has been found in secure communication channel.

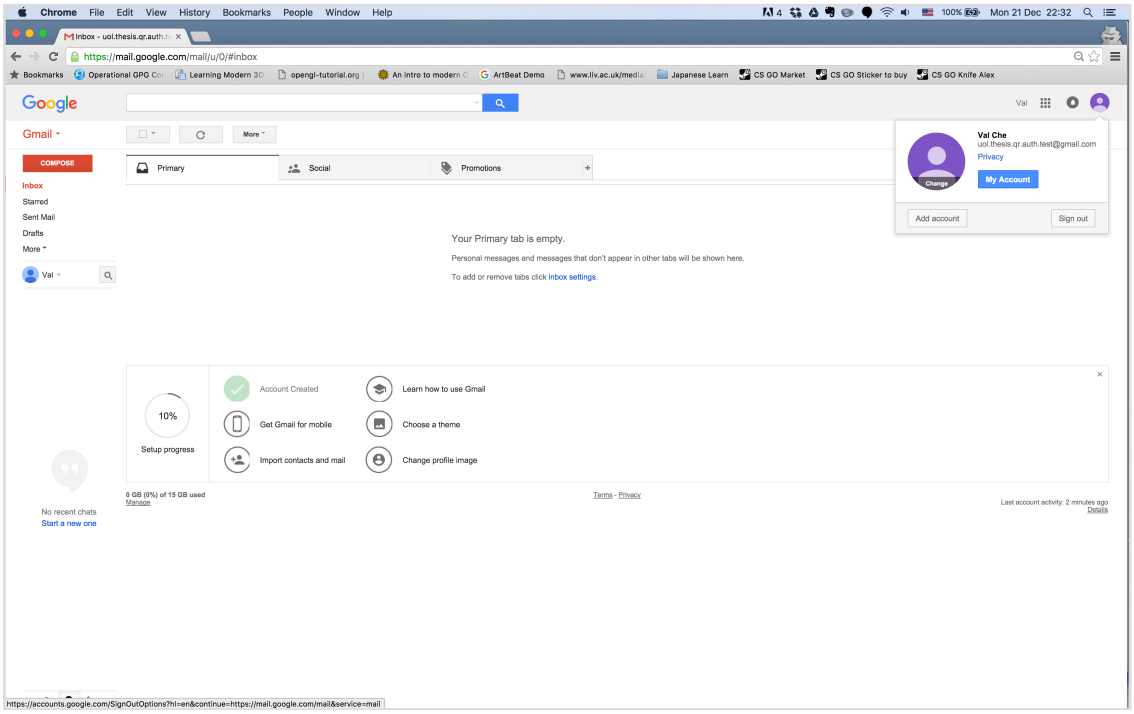


Figure 25. User has been successfully authenticated to Google service, i.e. Gmail.

B. User credentials for Google

The following user credentials has been used for testing the final software implementation:

- gmail #1: uol.thesis.qr.auth@gmail.com : pwd4UolQr!
- gmail #2 (developer account): uol.thesis.qr.auth.dev@gmail.com : pwd4UolQr!
- gmail #3 (test account): uol.thesis.qr.auth.test@gmail.com : tmp1tmp2

Google login page URL used for authentication: https://accounts.google.com/AddSession?sacu=1#identifier

C. Third-party libraries used in the project

This appendix provides detailed information on all third-party libraries utilised in implemented technical solution.

Library	Language	Licence	URL
CryptoSwift	Swift	Custom, but no restriction to use in commercial or educational purposes	https://github.com/krzyzanowskim/CryptoSwift
Alamofire	Swift	MIT	https://github.com/Alamofire/Alamofire
QRCoreReader	Swift	MIT	https://github.com/yannickl/QRCodeReader.swift
jQuery	JavaScript	MIT	https://jquery.com/
qrcodejs	JavaScript	MIT	https://github.com/davidshimjs/qrcodejs
BigInt	JavaScript	-none-	http://leemon.com/crypto/BigInt.js
crypto-js	JavaScript	MIT	https://github.com/brix/crypto-js

Table 2. Detailed libraries' information.

The major point about MIT licence is that the software could be used for any purposes including commercial. More information is available at https://opensource.org/licenses/MIT.

D. LINKS TO RESOURCES

Lightweight file sharing service server has been deploy on the author's personal server using publicly available domain, and is available at:

- Upload file: https://chupakabr.ru/extra-test-qr-api/methods/get.php
- Download file: https://chupakabr.ru/extra-test-qr-api/methods/put.php

Developed Chrome extension installer file (ZIP) as well as iOS The Vault app APK files could be found on download page at: http://chupakabr.ru/extra-test-qr-api/download/.

Keep in mind that APK file is built for Ad Hoc distribution using developer profile which means it may not work on some device. If this is the case then manual Xcode compilation and installation is required like it is described in Chapter 4 (The Vault app installation for testing).

Developed Chrome extension installer could be used as it is (ZIP file) to be installed on any computer with Chrome web browser available like it is describe in Chapter 4 (Chrome extension installation for testing).

E. SOURCE CODE OF THE IMPLEMENTED SOLUTION

This appendix provides full source code for all components of the implemented technical solution.

Full sources code including projects' structures as well as Xcode and IntellijI DEA project files also could be found at https://github.com/chupakabr/uol-qrcode-auth-thesis.

In case of GIT repository unavailability, full source code is also available in one ZIP file at: http://chupakabr.ru/extra-test-qr-api/download/assets/uol-qrcode-auth-thesis-master.zip.

E.1. Lightweight file sharing service

Server-side component written in pure PHP utilising standard library and its methods.

File: entry.php

```
<?php

define(QA_UPLOAD_PATH, "/mnt/qrvault");
define(QA_FILE_EXT, ".txt");
define(QA_MAX_FILE_SIZE, 512);
define(QA_ID_LEN, 32);

function QA_gen_fileid() {
    $randomFilename = "" . microtime(true) . "_" . QA_generateRandomString();
    return $randomFilename;
}

function QA_filepath_by_id($id) {
    // TODO Split files into directories, i.e. for id="ffacbcef1142acbde" directory path should
be "ff/ac/bc/ef/1142acbde"
    // TODO Check file name: only characters plus a dot
    return QA_UPLOAD_PATH . "/" . $id . QA_FILE_EXT;
}

function QA_generateRandomString($length = QA_ID_LEN) {
    $characters =
'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
    $charactersLength = strlen($characters);
    $randomString = '';
    for ($i = 0; $i < $length; ++$i) {
        $randomString .= $characters[rand(0, $charactersLength - 1)];
    }
    return $randomString;
}

function QA_normalize_id($id) {
    $id = preg_replace("/[^0-9_\-a-zA-Z]/", "", $id);
    return $id;
}

function QA_valid_id($id) {
    if (strlen($id) != QA_ID_LEN) {
        return false;
    }
    // TODO double preg_match check
    return true;
}
```

File: get.php

```
<?php

// TODO Better to use Nginx instead

require_once "../entry.php";

if ($_SERVER['REQUEST_METHOD'] != "GET") {
    echo "ERR invalid HTTP method\n";
    http_response_code(405);
    exit(1);
}
```

```
// Normalize file name: only characters plus a dot
$id = QA_normalize_id($_GET["id"]);
if (!QA_valid_id($id)) {
    echo "ERR invalid ID: [$id]\n";
    http_response_code(400);
    exit(2);
}

$filepath = QA_filepath_by_id($id);
if (@file_exists($filepath)) {
    $input = @fopen($filepath, "r");

    // Read the data 1 KB at a time and write to the file
    while ($data = @fread($input, QA_MAX_FILE_SIZE)) {
        echo $data;
    }
    @fclose($input);
    echo "\n";

    // TODO Delete the file, but maybe using cron job? with TTL of 2 minutes
    //@unlink($filepath);

    http_response_code(200);
} else {
    http_response_code(404);
}
```

File: put.php

```
<?php

require_once "../entry.php";

if ($_SERVER['REQUEST_METHOD'] != "PUT") {
    echo "ERR invalid HTTP method\n";
    http_response_code(405);
    exit(1);
}

// PUT data comes in on the stdin stream
$putdata = fopen("php://input", "r");

// Read the data 1 KB at a time and write to the file
$data = fgets($putdata, QA_MAX_FILE_SIZE);
fclose($putdata);
$info = explode(":", $data);
$cnt = count($info);
if ($cnt != 4) {
    echo "ERR malformed input ($cnt): [$data]\n";
    http_response_code(400);
    exit(2);
}

// Open a file for writing.
// Normalize file name: only characters plus a dot.
$id = QA_normalize_id($info[0]);
$timestamp = $info[1];
$clientPubKey = $info[2];
$credentials = $info[3];
```

```
if (!QA_valid_id($id)) {
    echo "ERR invalid ID: [$id]\n";
    http_response_code(400);
    exit(3);
}

$filepath = QA_filepath_by_id($id);
$fp = fopen($filepath, "w");

fwrite($fp, "$timestamp:$clientPubKey:$credentials", QA_MAX_FILE_SIZE);
fclose($fp);

http_response_code(200);
```

E.2. Shared JavaScript library

Shared JavaScript library utilising two JavaScript third-party libraries: Crypto-JS and BigInt. Used by both Desktop and Portable components.

File: qrvault-crypto.js

```
/**
 * QrAuthVault
 *
 * Crypto helpers library which uses 3rd party BigInt.js library
 *
 * 2015 (c) Valera Chevtaev
 */

if (typeof qrauth === "undefined") {
    qrauth = {};
}
qrauth.crypto = {};
qrauth.crypto.bits = 128;
qrauth.crypto.e = str2bigInt("65537", 10, 0);
qrauth.crypto.maxKey = 1073741824; // 2^30
qrauth.crypto.one = one; //int2bigInt(1, 1, 1);
qrauth.crypto.testStr = "asdasdasd";

// required to store latest value in the current context
qrauth.crypto.context = {};
qrauth.crypto.context.dhP = undefined;
qrauth.crypto.context.dhG = undefined;
qrauth.crypto.context.dhSecret = undefined;
qrauth.crypto.context.dhPubKey = undefined;
qrauth.crypto.context.dhPubKeyStr = undefined;
qrauth.crypto.context.dhPrivKey = undefined;
qrauth.crypto.context.dhPrivKeyStr = undefined;

// generate MD5 hash
qrauth.md5 = function(str) {
    return CryptoJS.MD5(str).toString();
```

```
};

// generate random prime number
qrauth.crypto.genRandomPrime = function(notCongruentToNum) {
    var primeNum;
    while (1) {
        primeNum = randProbPrime(qrauth.crypto.bits); // or randTruePrime()
        //the prime must not be congruent to 1 modulo e
        if (!equals(primeNum, notCongruentToNum) && !equalsInt(mod(primeNum,
qrauth.crypto.e), 1)) {
            break;
        }
    }
    return primeNum;
};

// generate DH p
qrauth.crypto.genP = function() {
    var prime = qrauth.crypto.genRandomPrime(qrauth.crypto.one);
    qrauth.crypto.context.dhP = prime;
    return prime;
};

// generate DH g
qrauth.crypto.genG = function(p) {
    var prime = qrauth.crypto.genRandomPrime(p);
    qrauth.crypto.context.dhG = prime;
    return prime;
};

// random number as a secret
qrauth.crypto.genSecret = function() {
    var secret = Math.floor((Math.random() * qrauth.crypto.maxKey) + 1);
    secret = int2bigInt(secret, qrauth.crypto.bits, 1);
    qrauth.crypto.context.dhSecret = secret;
    return secret;
};

// public key = g^secret mod p
qrauth.crypto.evalPubKey = function(g, secret, p) {
    var key = powMod(g, secret, p);
    qrauth.crypto.context.dhPubKey = key;
    return key;
};

// evaluate public key
// resulting value is converted from bigint to string
qrauth.crypto.evalPubKeyFromStr = function(g, secret, p) {
    var key = qrauth.crypto.bigint2str(qrauth.crypto.evalPubKey(qrauth.crypto.str2bigint(g),
qrauth.crypto.str2bigint(secret), qrauth.crypto.str2bigint(p)));
    qrauth.crypto.context.dhPubKeyStr = key;
    return key;
};

// private key = A^b mod p or B^a mod p
qrauth.crypto.evalPrivKey = function(A, b, p) {
    var key = powMod(A, b, p);
    qrauth.crypto.context.dhPrivKey = key;
    return key;
};

// generate private key
```

```
// resulting value is converted from bigint to string
qrauth.crypto.evalPrivKeyFromStr = function(A, b, p) {
    var key = qrauth.crypto.bigint2str(qrauth.crypto.evalPrivKey(qrauth.crypto.str2bigint(A),
qrauth.crypto.str2bigint(b), qrauth.crypto.str2bigint(p)));
    qrauth.crypto.context.dhPrivKeyStr = key;
    return key;
};

// convert bigint variable to string
qrauth.crypto.bigint2str = function(bigintNum) {
    return bigInt2str(bigintNum, 10);
};

// convert string variable to bigint
qrauth.crypto.str2bigint = function(bigintStr) {
    return str2bigInt(bigintStr, 10, 0);
};

// encrypt a message
qrauth.crypto.encrypt = function(text, sharedKey) {
    return CryptoJS.AES.encrypt(text, sharedKey).toString();
}

// decrypt a message
qrauth.crypto.decrypt = function(cipher, sharedKey) {
    return CryptoJS.AES.decrypt(cipher, sharedKey).toString(CryptoJS.enc.Utf8);
}
```

E.3. Chrome extension

Desktop component written in JavaScript utilising several third-party JavaScript libraries including Crypto-JS, BitInt, QrCode and jQuery. Apart of mentioned libraries, this component also uses shared JavaScript library.

File: manifest.json

```
{
  "name": "QR Auth",
  "version": "0.0.1",
  "manifest_version": 2,
  "description": "QR Code authentication",
  "homepage_url": "https://github.com/chupakabr/uol-qrcode-auth-thesis",
  "icons": {
    "16": "icons/icon16.png",
    "48": "icons/icon48.png",
    "128": "icons/icon128.png"
  },
  "default_locale": "en",
  /* is being called on tab open, i.e. match */
  "background": {
    "page": "src/bg/background.html",
    "persistent": false
  },
```

```
  /* is being shown on extension icon clicked */
  "page_action": {
    "default_icon": "icons/icon48.png",
    "default_title": "Authenticate",
    "default_popup": "src/page_action/page_action.html"
  },
  "permissions": [
    "tabs",
    "https://*/*",
    "http://*/*"
    //"https://accounts.google.com/*",
    //"https://accounts.google.com/AddSession*"
  ],
  "content_scripts": [
    {
      "matches": [
        "https://gmail.com/*",
        "https://accounts.google.com/*"
      ],
      "css": [
        "src/inject/inject.css"
      ],
      "js": [
        "js/jquery/jquery.min.js",
        "src/inject/inject.js"
      ]
    }
  ]
}
```

File: background.html

```
<!DOCTYPE html>
<html>
<head>
  <script type="text/javascript" src="../../js/jquery/jquery.min.js"></script>
  <script type="text/javascript" src="../../js/qr/qrcode.js"></script>
  <script type="text/javascript" src="background.js"></script>
</head>

<div id="mytest">
  <p>initial text</p>
</div>

</html>
```

File: background.js

```
/**
 * QrAuthVault
 *
 * Extension background scripts
 *
 * 2015 (c) Valera Chevtaev
 */
```

```
// extension events listener
chrome.extension.onMessage.addListener(
  function(request, sender, sendResponse) {

    if (request.name == "ready") {
       // display action_page icon in address bar if page is ready
       chrome.pageAction.show(sender.tab.id);
    } else if (request.name == "page_action.open") {
       // nothing
    }

    sendResponse();
});
```

File: page_action.html

```
<!doctype html>
<style type="text/css">
        #mainPopup {
                padding: 10px;
                height: 350px;
                width: 400px;
                font-family: Helvetica, Ubuntu, Arial, sans-serif;
        }
        h1 {
                font-size: 2em;
        }
        p.title {
                width: 300px;
                font-size: 1.1em;
        }
        #status {
                padding-top: 10px;
                color: deeppink;
                font-size: 1.1em;
        }
</style>

<script type="text/javascript" src="../../js/jquery/jquery.min.js"></script>
<script type="text/javascript" src="../../js/qr/qrcode.js"></script>
<script type="text/javascript" src="../../js/crypto-js/core.js"></script>
<script type="text/javascript" src="../../js/crypto-js/enc-base64.js"></script>
<script type="text/javascript" src="../../js/crypto-js/md5.js"></script>
<script type="text/javascript" src="../../js/crypto-js/evpkdf.js"></script>
<script type="text/javascript" src="../../js/crypto-js/cipher-core.js"></script>
<script type="text/javascript" src="../../js/crypto-js/aes.js"></script>
<script type="text/javascript" src="../../js/crypto/BigInt.js"></script>
<script type="text/javascript" src="../../js/crypto/qrvault-crypto.js"></script>
<script src="page_action.js"></script>

<div id="mainPopup">
        <p class="title">Scan this QR code with your phone via QrVault app</p>
        <div id="qrcode"></div>
        <div id="status"></div>
</div>
```

File: page_action.js

```
/**
 * QrAuthVault
 *
 * Main implementation file (basically it is a mediator):
 * - generate QR code
 * - poll for credentials file
 * - extract credentials
 * - authenticate user
 *
 * 2015 (c) Valera Chevtaev
 */

if (typeof qrauth === "undefined") {
    qrauth = {};
}
qrauth.retry = {};
qrauth.retry.cur = 0;
qrauth.retry.limit = 10;
qrauth.retry.millis = 3000;
qrauth.waiting = false;

// generate random string by pattern
qrauth.replacePattern = function(pattern) {
    var possibleC = "qwertyuiopasdfghjklzxcvbnm";
    var possibleV = "1234567890QWERTYUIOPASDFGHJKLZXCVBNM";

    var pIndex = pattern.length;
    var res = new Array(pIndex);
    while (pIndex--) {
        res[pIndex] = pattern[pIndex]
          .replace(/v/,randomCharacter(possibleV))
          .replace(/c/,randomCharacter(possibleC));
    }

    function randomCharacter(bucket) {
        var res = bucket.charAt(Math.floor(Math.random() * bucket.length));
        return res;
    }
    return res.join("").toLowerCase();
};

// generate QR code when current web page URL is available via callback
qrauth.tabUrlAvailable = function(currentWebsiteUrl) {
    // Only if not running yet
    if (qrauth.waiting) {
        return;
    }

    chrome.extension.sendMessage({"name": "page_action.open"}, function (response) {
        // Generate QR code
        var seq = qrauth.replacePattern("ccvvccvcvcvcccvv"); // 16 chars
        var now = Date.now();
        var id = qrauth.md5(seq + now);
        var curUrl = currentWebsiteUrl.substring(0, currentWebsiteUrl.indexOf('?'));
        var dhP = qrauth.crypto.genP();
        var dhG = qrauth.crypto.genG(dhP);
        var dhSecret = qrauth.crypto.genSecret();
        var dhKey = qrauth.crypto.evalPubKey(dhG, dhSecret, dhP);
        var data = {
            url: curUrl,
```

```
            ts: now,
            seq: seq,
            dhP: qrauth.crypto.bigint2str(dhP),
            dhG: qrauth.crypto.bigint2str(dhG),
            dhKey: qrauth.crypto.bigint2str(dhKey)
        };
        new QRCode(document.getElementById("qrcode"), {
            //text: '{"url":"' + curUrl + '", "ts":' + now + ',"seq":"' + seq + '"}',
            text: JSON.stringify(data),
            width: 256,
            height: 256,
            colorDark: "#000000",
            colorLight: "#ffffff",
            correctLevel: QRCode.CorrectLevel.H
        });

        //console.info("requested with seq=" + seq + ", ts=" + now + ", id=" + id);

        $("#status").text("");

        // Poll for http://qrauth.chupakabr.ru/methods/get.php?id=123
        qrauth.retry.cur = qrauth.retry.limit;
        qrauth.waiting = true;
        (function authFilePolling() {
            $.ajax({
                url: "https://chupakabr.ru/extra-test-qr-api/methods/get.php?id=" + id,
                cache: false,
                type: "GET",
                success: function (data, statusText, jqXHR) {
                    qrauth.waiting = false;

                    //console.info("response: " + statusText);
                    //console.info("success - data: " + data);
                    if (data) {
                        var info = data.split(':', 3);
                        if (info && info.length == 3) {
                            var ts = info[0];
                            var clientPubKey = info[1];
                            var cipher = info[2];

                            // decrypt credentials got from the client
                            var dhSharedKey =
qrauth.crypto.evalPrivKey(qrauth.crypto.str2bigint(clientPubKey), dhSecret, dhP);
                            var decrypted = qrauth.crypto.decrypt(cipher,
qrauth.crypto.bigint2str(dhSharedKey));

                            // extract credentials
                            var creds = decrypted.split(':', 2);
                            if (creds && creds.length == 2) {
                                // authorize on gmail.com
                                chrome.tabs.query({active: true, currentWindow: true}, function (tabs)
{
                                    chrome.tabs.sendMessage(
                                      tabs[0].id,
                                      {
                                          "name": "qrauth.do",
                                          "usr": creds[0],
                                          "pwd": creds[1],
                                      },
                                      function (response) {
                                          chrome.extension.sendMessage({"name": "qrauth.fail", msg:
""});
```

```
                            });
                        });

                        return;
                    }
                }
            }

            // Something went wrong?
            console.info("invalid data from the server");
            chrome.extension.sendMessage({"name": "qrauth.fail", "msg": "Server
error"}); // TOD Handle this error
        },
        error: function (data) {
            if (qrauth.retry.cur-- > 0) {
                setTimeout(authFilePolling, qrauth.retry.millis);
            } else {
                qrauth.waiting = false;
                console.info("error!");
                chrome.extension.sendMessage({
                    "name": "qrauth.fail",
                    "msg": "Please check your QrVault app for valid credentials"
                }); // TODO Handle this error
            }
        },
    }).done(function (data, statusText, jqXHR) {
        console.info("done");
    });
})();
});
};

// request current web page URL
chrome.tabs.query({'active': true, 'windowId': chrome.windows.WINDOW_ID_CURRENT},
    function(tabs){
        qrauth.tabUrlAvailable(tabs[0].url);
    }
);

// extension events listener
chrome.extension.onMessage.addListener(
    function(request, sender, sendResponse) {

        if (request.name == "qrauth.fail") {
            $("#status").text(request.msg);
        } else if (request.name == "qrauth.done") {
            $("#status").text("");
        }

        sendResponse();
});
```

File: inject.js

```
/**
 * QrAuthVault
 *
 * Extension code to be executed on page load plus web site authentication scripts
 *
```

```
 * 2015 (c) Valera Chevtaev
 */

chrome.extension.sendMessage({"name": "ready"}, function(response) {
	var readyStateCheckInterval = setInterval(function() {
	if (document.readyState === "complete") {
		clearInterval(readyStateCheckInterval);

		// ----------------------------------------------------------
		// This part of the script triggers when page is done loading
		console.log("Hello from QrAuthVault. This message was sent from
scripts/inject.js");
		// ----------------------------------------------------------
	}
	}, 10);
});


if (typeof qrauth === "undefined") {
	qrauth = {};
}
qrauth.auth = {};

// automated authentication to Google Account
// TODO move this kind of authorization steps to a server to be delivered on auth request
// TODO replace hardcoded delay with JS wait for an input field to appear
qrauth.auth.gmail = function(usr, pwd) {
	// enter username and press next button
	$('input[type="email"]').val(usr);
	$('input#next').delay(1000).click();

	// wait for 2 seconds and then enter password and proceed
	setTimeout(function() {
		$('input[type="password"]').val(pwd);
		$('input#signIn').delay(1000).click();
	}, 2000);
};

// extension events listener
chrome.extension.onMessage.addListener(
	function(request, sender, sendResponse) {
		console.info("hello from the other side");

		if (request.name == "qrauth.do") {
			console.info("QrAuthVault - Authenticate on Google")
			var usr = request.usr;
			var pwd = request.pwd;
			qrauth.auth.gmail(usr, pwd);
		} else if (request.name == "qrauth.fail") {
			console.info("QrAuthVault - ERROR: " + request.msg);
			// TODO Handle error by showing an alert or something like that
		} else if (request.name == "qrauth.log") {
			console.info("QrAuthVault - " + request.msg);
		}

		sendResponse();
});
```

E.4. iOS app (The Vault)

The Vault client-side / trusted component written in Swift for iOS platform utilising standard Foundation libraries as well as several third party libraries including CryptoSwift, Alamofire and QRCoreReader. Apart of mentioned libraries, this component also uses shared JavaScript library.

File: Podfile

```
platform :ios, '8.0'
use_frameworks!

source 'https://github.com/CocoaPods/Specs.git'
pod 'QRCodeReader.swift', '~> 5.2.1'

source 'https://github.com/CocoaPods/Specs.git'
pod 'Alamofire', '~> 3.0'

pod 'CryptoSwift', :git => "https://github.com/krzyzanowskim/CryptoSwift", :branch =>
"master"
```

File: Info.plist

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
	<key>CFBundleDevelopmentRegion</key>
	<string>en</string>
	<key>CFBundleExecutable</key>
	<string>$(EXECUTABLE_NAME)</string>
	<key>CFBundleIdentifier</key>
	<string>$(PRODUCT_BUNDLE_IDENTIFIER)</string>
	<key>CFBundleInfoDictionaryVersion</key>
	<string>6.0</string>
	<key>CFBundleName</key>
	<string>$(PRODUCT_NAME)</string>
	<key>CFBundlePackageType</key>
	<string>APPL</string>
	<key>CFBundleShortVersionString</key>
	<string>1.0</string>
	<key>CFBundleSignature</key>
	<string>????</string>
	<key>CFBundleVersion</key>
	<string>1</string>
	<key>LSRequiresIPhoneOS</key>
	<true/>
	<key>NSAppTransportSecurity</key>
	<dict>
		<key>NSAllowsArbitraryLoads</key>
```

```
                <false/>
                <key>NSExceptionDomains</key>
                <dict>
                        <key>chupakabr.ru</key>
                        <dict>
                                <key>NSExceptionRequiresForwardSecrecy</key>
                                <false/>
                                <key>NSIncludesSubdomains</key>
                                <true/>
                                <key>NSTemporaryExceptionMinimumTLSVersion</
key>
                                <string>TLSv1.2</string>
                        </dict>
                </dict>
        </dict>
        <key>UILaunchStoryboardName</key>
        <string>LaunchScreen</string>
        <key>UIMainStoryboardFile</key>
        <string>Main</string>
        <key>UIRequiredDeviceCapabilities</key>
        <array>
                <string>armv7</string>
        </array>
        <key>UISupportedInterfaceOrientations</key>
        <array>
                <string>UIInterfaceOrientationPortrait</string>
        </array>
</dict>
</plist>
```

File: Credential.swift

```
//
//  Credential.swift
//  QrAuthVault
//
//  Created by Valeriy Chevtaev on 15/12/2015.
//  Copyright © 2015 Valera Chevtaev. All rights reserved.
//

import Foundation
import CoreData


class Credential: NSManagedObject {

   override init(entity: NSEntityDescription, insertIntoManagedObjectContext context:
NSManagedObjectContext?) {
      super.init(entity: entity, insertIntoManagedObjectContext: context)
   }

   func save() {
      do {
         guard let moc = self.managedObjectContext else {
            throw NSError(domain: "Empty MOC", code: 1, userInfo: nil)
         }
         try moc.save()
      } catch let error as NSError {
         print("ERROR Could not save credentials: \(error), \(error.userInfo)")
```

```
        }
    }
}
```

File: Credential+CoreDataProperties.swift

```
//
//  Credential+CoreDataProperties.swift
//  QrAuthVault
//
//  Created by Valeriy Chevtaev on 15/12/2015.
//  Copyright © 2015 Valera Chevtaev. All rights reserved.
//
//  Choose "Create NSManagedObject Subclass…" from the Core Data editor menu
//  to delete and recreate this implementation file for your updated model.
//

import Foundation
import CoreData

extension Credential {

    @NSManaged var usr: String?
    @NSManaged var pwd: String?
    @NSManaged var service: String?

}
```

File: AppDelegate.swift

```
//
//  AppDelegate.swift
//  QrAuthVault
//
//  Created by Valeriy Chevtaev on 12/09/2015.
//  Copyright © 2015 Valera Chevtaev. All rights reserved.
//

import UIKit
import CoreData
import CleanroomLogger

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?


    func application(application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [NSObject: AnyObject]?) -> Bool {
        // Override point for customization after application launch.
        return true
    }

    func applicationWillResignActive(application: UIApplication) {
```

```
        // Sent when the application is about to move from active to inactive state. This can
occur for certain types of temporary interruptions (such as an incoming phone call or SMS
message) or when the user quits the application and it begins the transition to the
background state.
        // Use this method to pause ongoing tasks, disable timers, and throttle down
OpenGL ES frame rates. Games should use this method to pause the game.
    }

    func applicationDidEnterBackground(application: UIApplication) {
        // Use this method to release shared resources, save user data, invalidate timers,
and store enough application state information to restore your application to its current
state in case it is terminated later.
        // If your application supports background execution, this method is called instead of
applicationWillTerminate: when the user quits.
    }

    func applicationWillEnterForeground(application: UIApplication) {
        // Called as part of the transition from the background to the inactive state; here you
can undo many of the changes made on entering the background.
    }

    func applicationDidBecomeActive(application: UIApplication) {
        // Restart any tasks that were paused (or not yet started) while the application was
inactive. If the application was previously in the background, optionally refresh the user
interface.
    }

    func applicationWillTerminate(application: UIApplication) {
        // Called when the application is about to terminate. Save data if appropriate. See
also applicationDidEnterBackground:.
        // Saves changes in the application's managed object context before the application
terminates.
        self.saveContext()
    }

    // MARK: - Core Data stack

    lazy var applicationDocumentsDirectory: NSURL = {
        // The directory the application uses to store the Core Data store file. This code uses
a directory named "ru.chupakabr.dev.osx.QrAuthVault" in the application's documents
Application Support directory.
        let urls = NSFileManager.defaultManager().URLsForDirectory(.DocumentDirectory,
inDomains: .UserDomainMask)
        return urls[urls.count-1]
    }()

    lazy var managedObjectModel: NSManagedObjectModel = {
        // The managed object model for the application. This property is not optional. It is a
fatal error for the application not to be able to find and load its model.
        let modelURL = NSBundle.mainBundle().URLForResource("QrAuthVault",
withExtension: "momd")!
        return NSManagedObjectModel(contentsOfURL: modelURL)!
    }()

    lazy var persistentStoreCoordinator: NSPersistentStoreCoordinator = {
        // The persistent store coordinator for the application. This implementation creates
and return a coordinator, having added the store for the application to it. This property is
optional since there are legitimate error conditions that could cause the creation of the
store to fail.
        // Create the coordinator and store
        let coordinator = NSPersistentStoreCoordinator(managedObjectModel:
self.managedObjectModel)
```

```
        let url =
self.applicationDocumentsDirectory.URLByAppendingPathComponent("QrAuthVault.sqlite
")
        var failureReason = "There was an error creating or loading the application's saved
data."
        do {
            try coordinator.addPersistentStoreWithType(NSSQLiteStoreType,
                configuration: nil, URL: url,
                options: [NSPersistentStoreFileProtectionKey: NSFileProtectionComplete])
        } catch {
            // Report any error we got.
            var dict = [String: AnyObject]()
            dict[NSLocalizedDescriptionKey] = "Failed to initialize the application's saved
data"
            dict[NSLocalizedFailureReasonErrorKey] = failureReason

            dict[NSUnderlyingErrorKey] = error as NSError
            let wrappedError = NSError(domain: “DB_FAIL", code: 9999, userInfo: dict)
            // Replace this with code to handle the error appropriately.
            // abort() causes the application to generate a crash log and terminate. You should
not use this function in a shipping application, although it may be useful during
development.
            NSLog("Unresolved error \(wrappedError), \(wrappedError.userInfo)")
            abort()
        }

        return coordinator
    }()

    lazy var managedObjectContext: NSManagedObjectContext = {
        // Returns the managed object context for the application (which is already bound to
the persistent store coordinator for the application.) This property is optional since there
are legitimate error conditions that could cause the creation of the context to fail.
        let coordinator = self.persistentStoreCoordinator
        var managedObjectContext =
NSManagedObjectContext(concurrencyType: .MainQueueConcurrencyType)
        managedObjectContext.persistentStoreCoordinator = coordinator
        return managedObjectContext
    }()

    // MARK: - Core Data Saving support

    func saveContext () {
        if managedObjectContext.hasChanges {
            do {
                try managedObjectContext.save()
            } catch {
                // Replace this implementation with code to handle the error appropriately.
                // abort() causes the application to generate a crash log and terminate. You
should not use this function in a shipping application, although it may be useful during
development.
                let nserror = error as NSError
                NSLog("Unresolved error \(nserror), \(nserror.userInfo)")
                abort()
            }
        }
    }

}
```

File: ViewController.swift

```
//
//  ViewController.swift
//  QrAuthVault
//
//  Created by Valeriy Chevtaev on 12/09/2015.
//  Copyright © 2015 Valera Chevtaev. All rights reserved.
//

import UIKit
import CleanroomLogger
import LocalAuthentication

class ViewController: UIViewController {

    var context = LAContext()

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.

        if context.canEvaluatePolicy(LAPolicy.DeviceOwnerAuthenticationWithBiometrics,
error: nil) {
            // TODO Can do auth
        } else {
            // TODO Cannot do auth, display a message on UI
        }
    }

    override func viewDidAppear(animated: Bool) {
        super.viewDidAppear(animated)

        // Reset context so fignerprint is asked again
        context = LAContext()
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }


    @IBAction func authenticate() {
        print("++ authenticate")

        if context.canEvaluatePolicy(LAPolicy.DeviceOwnerAuthenticationWithBiometrics,
error:nil) {
            context.evaluatePolicy(LAPolicy.DeviceOwnerAuthenticationWithBiometrics,
                localizedReason: "Logging in with Touch ID",
                reply: { (success : Bool, error : NSError? ) -> Void in
                    dispatch_async(dispatch_get_main_queue(), {
                        if success {
                            self.performSegueWithIdentifier("auth_done", sender: self)
                        }

                        if error != nil {
                            var message : NSString
                            var showAlert : Bool

                            switch(error!.code) {
                            case LAError.AuthenticationFailed.rawValue:
```

```
                        message = "There was a problem verifying your identity."
                        showAlert = true
                        break;
                    case LAError.UserCancel.rawValue:
                        message = "You pressed cancel."
                        showAlert = true
                        break;
                    case LAError.UserFallback.rawValue:
                        message = "You pressed password."
                        showAlert = true
                        break;
                    default:
                        showAlert = true
                        message = "Touch ID may not be configured"
                        break;
                    }

                    let alertView = UIAlertController(title: "Error",
                        message: message as String, preferredStyle:.Alert)
                    let okAction = UIAlertAction(title: "Darn!", style: .Default, handler: nil)
                    alertView.addAction(okAction)
                    if showAlert {
                        self.presentViewController(alertView, animated: true, completion: nil)
                    }

                }
            })
        })
    } else {
        let alertView = UIAlertController(title: "Error", message: "Touch ID not available" as
String, preferredStyle:.Alert)
        let okAction = UIAlertAction(title: "OK", style: .Default, handler: nil)
        alertView.addAction(okAction)
        self.presentViewController(alertView, animated: true, completion: nil)
    }
  }
}
```

File: EditCredsViewController.swift

```
//
//  EditCredsViewController.swift
//  QrAuthVault
//
//  Created by Valeriy Chevtaev on 03/12/2015.
//  Copyright © 2015 Valera Chevtaev. All rights reserved.
//

import UIKit
import CleanroomLogger
import QRCodeReader
import AVFoundation
import Alamofire
import CryptoSwift
import CoreData
import JavaScriptCore

class EditCredsViewController: UIViewController {
```

```
    private let fileShareUrl = "https://chupakabr.ru/extra-test-qr-api/methods/put.php"
    private let defaultLoginUrl = "https://accounts.google.com/AddSession"
    private let defaultService = "google"

    private var jsContext: JSContext! = nil

    @IBOutlet var usernameTextField: UITextField!
    @IBOutlet var passwordTextField: UITextField!
    @IBOutlet var scannedQrText: UITextView!

    lazy var reader = QRCodeReaderViewController(metadataObjectTypes:
[AVMetadataObjectTypeQRCode])

    override func viewDidLoad() {
        super.viewDidLoad()

        // Load credentials from data store
        let savedModel = loadEntry(defaultService)
        if let savedModel = savedModel {
            self.usernameTextField.text = savedModel.usr
            self.passwordTextField.text = savedModel.pwd
        }

        print("Unlocked!")
    }

    override func viewDidAppear(animated: Bool) {
        super.viewDidAppear(animated)

        // Load JS libraries into JS context
        if jsContext == nil {
            do {
                jsContext = JSContext()
                try jsContext.evaluateScript(String(contentsOfURL: jsFile("core"), encoding:
NSUTF8StringEncoding))
                try jsContext.evaluateScript(String(contentsOfURL: jsFile("enc-base64"),
encoding: NSUTF8StringEncoding))
                try jsContext.evaluateScript(String(contentsOfURL: jsFile("md5"), encoding:
NSUTF8StringEncoding))
                try jsContext.evaluateScript(String(contentsOfURL: jsFile("evpkdf"), encoding:
NSUTF8StringEncoding))
                try jsContext.evaluateScript(String(contentsOfURL: jsFile("cipher-core"),
encoding: NSUTF8StringEncoding))
                try jsContext.evaluateScript(String(contentsOfURL: jsFile("aes"), encoding:
NSUTF8StringEncoding))
                try jsContext.evaluateScript(String(contentsOfURL: jsFile("BigInt"), encoding:
NSUTF8StringEncoding))
                try jsContext.evaluateScript(String(contentsOfURL: jsFile("qrvault-crypto"),
encoding: NSUTF8StringEncoding))

                print("JS loaded")
            } catch let err as NSError {
                // Go back to main/auth screen
                print("ERROR Cannot load JS scripts: \(err)")
                self.dismissViewControllerAnimated(animated, completion: nil)
                return
            }
        }
    }

    // MARK: - UI actions
```

```
    @IBAction func scan() {
        print("++ scan QR")

        // Or by using the closure pattern
        reader.completionBlock = { [weak self] (result: String?) in
            defer {
                if let vc = self {
                    vc.dismissViewControllerAnimated(true, completion: nil)
                }
            }
            if let result = result {
                print("Raw QR JSON: \(result)")

                let data: NSData = result.dataUsingEncoding(NSUTF8StringEncoding)!
                var json: AnyObject? = nil
                do {
                    try json = NSJSONSerialization.JSONObjectWithData(data, options:
NSJSONReadingOptions(rawValue: 0))

                    if let json = json,
                        seq = json["seq"] as! String?,
                        url = json["url"] as! String?,
                        ts = json["ts"] as! Int?,
                        dhP = json["dhP"] as! String?,
                        dhG = json["dhG"] as! String?,
                        dhKey = json["dhKey"] as! String?
                    {
                        let dhInfo = DhInfo(p: dhP, g: dhG, key: dhKey)
                        print("Parsed: seq=[\(seq)], url=[\(url)], ts=[\(ts), dh=[\(dhInfo)]]")
                        if let vc = self {
                            try vc.onSuccess(seq: seq, url: url, timestamp: ts, dh: dhInfo)
                        }
                    } else {
                        throw NSError(domain: "Unable to parse JSON", code: 1, userInfo: nil)
                    }

                } catch {
                    // error parsing JSON
                    print("ERROR QR JSON parsing: \(error)")

                    // Alert: parsing error
                    if let vc = self {
                        vc.onError(message: "Cannot parse QR")
                    }
                }
            } else {
                // Alert: nothing in QR code
                print("QR JSON is empty or cancel pressed")
                if let vc = self {
                    vc.onError(message: "Scan cancelled")
                }
            }
        }

        // Presents the reader as modal form sheet
        reader.modalPresentationStyle = .FormSheet
        presentViewController(reader, animated: true, completion: nil)
    }

    @IBAction func save() {
        print("++ save")
```

```
        do {
            try saveCredentials()
        } catch let err as NSError {
            print("ERROR Cannot persist credentials to database: \(err)")
        }
    }

    @IBAction func saveAndLock() {
        print("++ lock")

        // Save
        do {
            try saveCredentials()
        } catch let err as NSError {
            print("ERROR Cannot persist credentials to database: \(err)")
        }

        // Lock: Go back to main/auth screen
        self.dismissViewControllerAnimated(true, completion: nil)
    }

    //
    // Save credentials to the vault
    // TODO Perform on background thread
    private func saveCredentials() throws {
        print("saving credentials...")

        let credentials = loadCredentials(defaultLoginUrl)
        guard let credentialsGuarded = credentials else {
            throw NSError(domain: "Cannot read credentials", code: 1, userInfo: nil)
        }

        // try to load existing entry in DB first
        var model = loadEntry(credentialsGuarded.service)
        if model == nil {
            model = Credential(entity: getDataEntity(), insertIntoManagedObjectContext:
getMOC())
        }
        model!.usr = credentialsGuarded.username
        model!.pwd = credentialsGuarded.password
        model!.service = credentialsGuarded.service
        model!.save()
    }

    //
    // Handle QR parsing success
    private func onSuccess(seq seq: String?, url: String?, timestamp ts: Int?, dh: DhInfo)
throws {
        print("onSuccess: seq=[\(seq)], url=[\(url)], ts=[\(ts)], \(dh)")
        guard let url = url, ts = ts, seq = seq else {
            throw NSError(domain: "Empty parameters on success", code: 2, userInfo: nil)
        }

        // Calculate md5 of seq+ts to produce ID
        let id = "\(seq)\(ts)".md5()

        // Success: update UI
        dispatch_async(dispatch_get_main_queue()) {
            [weak self] in
            if let vc = self {
                vc.scannedQrText.text = "id=\(id)\nurl=\(url)\nts=\(ts)\nDiffie-Hellman: \(dh)"
            }
```

```
        }

        // Success: upload credentials file
        do {
            try uploadCredentials(id: id, loginInfo: loadCredentials(url), timestamp: ts, dh: dh)
        } catch {
            // TODO Handle error
            print("ERROR Can't upload credentials: \(error)")
        }
    }

    //
    // Handle QR parsing error
    private func onError(message error: String) {
        // TODO Alert?
        print("ERROR \(error)")
    }

    //
    // Upload user credentials to file sharing service
    private func uploadCredentials(id id: String?, loginInfo: LoginInfo?,
        timestamp ts: Int?, dh: DhInfo, lifetimeSec ttlSec: Int = 120) throws
    {
        guard let loginInfo = loginInfo, id = id, ts = ts else {
            throw NSError(domain: "Login information, id or timestamp not found", code: 1,
userInfo: nil)
        }

        // Generate local secret, public and shared keys
        let secret = genSecretKey()
        let publicB = evalPublicKey(g: dh.g, secret: secret, p: dh.p)
        let sharedKey = evalSharedPrivateKey(publicA: dh.pubKey, secret: secret, p: dh.p)
        let cipher = encryptMessage("\(loginInfo.username):\(loginInfo.password)",
sharedKey: sharedKey)
        //print("secret=\(secret)")
        //print("publicB=\(publicB)")
        //print("sharedKey=\(sharedKey)")
        //print("cipher=\(cipher)")

        // Encrypt data with shared key
        let dataStr = "\(id):\(ts):\(publicB):\(cipher)"
        print("Upload credentials: \(dataStr)")

        if let data = dataStr.dataUsingEncoding(NSUTF8StringEncoding) {
            print("Uploading encrypted credentials to \(fileShareUrl)")
            Alamofire.upload(.PUT, fileShareUrl, data: data)
        } else {
            print("ERROR cannot serialize credentials upload")
        }
    }

    //
    // Load user credentials for specified URL from the vault
    // TODO Evaluate service using URL passed as parameter
    private func loadCredentials(url: String) -> LoginInfo? {
        if let username = usernameTextField.text, password = passwordTextField.text {
            return LoginInfo(service: defaultService, username: username, password:
password)
        }
        return nil
    }
```

```
    //
    // MARK: - Database operations

    private func getSharedDelegate() -> AppDelegate {
        return UIApplication.sharedApplication().delegate as! AppDelegate
    }

    private func getMOC() -> NSManagedObjectContext {
        return getSharedDelegate().managedObjectContext
    }

    private func getDataEntity() -> NSEntityDescription {
        // cannot be nil
        return NSEntityDescription.entityForName("Credential", inManagedObjectContext:
getMOC())!
    }

    private func loadEntry(service: String) -> Credential? {
        let fetchRequest = NSFetchRequest(entityName: "Credential")
        fetchRequest.predicate = NSPredicate(format: "service == %@", service)
        do {
            let res = try getMOC().executeFetchRequest(fetchRequest)
            if res.count > 0 {
                return res[0] as? Credential
            }
        } catch let err as NSError {
            print("ERROR Cannot load entries from database: \(err)")
        }

        return nil
    }


    //
    // MARK: - JS manipulation and evaluation

    private func jsFile(filenameWithoutExt: String) throws -> NSURL {
        guard let file = NSBundle.mainBundle().URLForResource(filenameWithoutExt,
withExtension: "js", subdirectory: "js") else {
            throw NSError(domain: "JsFileLoad", code: 1, userInfo: ["file":
filenameWithoutExt])
        }
        return file
    }

    private func genSecretKey() -> String {
        let secret: JSValue =
jsContext.evaluateScript("qrauth.crypto.bigint2str(qrauth.crypto.genSecret())")
        return secret.toString()
    }

    private func evalPublicKey(g g: String, secret: String, p: String) -> String {
        let jsFuncEvalPubKey = getJsCryptoFunc("evalPubKeyFromStr")
        let publicKey: JSValue = jsFuncEvalPubKey.callWithArguments([g, secret, p])
        return publicKey.toString()
    }

    private func evalSharedPrivateKey(publicA publicA: String, secret: String, p: String) ->
String {
        let jsFunc = getJsCryptoFunc("evalPrivKeyFromStr")
        let privateKey: JSValue = jsFunc.callWithArguments([publicA, secret, p])
```

```
        return privateKey.toString()
    }

    private func encryptMessage(text: String, sharedKey: String) -> String {
        let jsFunc = getJsCryptoFunc("encrypt")
        let cipher: JSValue = jsFunc.callWithArguments([text, sharedKey])
        return cipher.toString()
    }

    private func getJsCryptoFunc(funcName: String) -> JSValue! {
        return
jsContext.objectForKeyedSubscript("qrauth").objectForKeyedSubscript("crypto").objectFo
rKeyedSubscript(funcName)
    }
}

//
// Login information plain immutable object
class LoginInfo {
    let service: String
    let username: String
    let password: String

    init(service: String, username: String, password: String) {
        self.service = service
        self.username = username
        self.password = password
    }
}

//
// Diffie-Hellman information holder
class DhInfo: CustomStringConvertible {
    let p: String
    let g: String
    let pubKey: String

    init(p: String, g: String, key: String) {
        self.p = p;
        self.g = g;
        self.pubKey = key;
    }

    var description: String {
        return "DH(p=\(p),g=\(g),pubKey=\(pubKey))"
    }
}
```

F. APACHE2 EXAMPLE CONFIGURATION

This appendix provides complete configuration files for Apache2 web server with SSL and PHP support running on Linux. Required to developed run lightweight file sharing service.

File: /etc/apache2/sites-enabled/010-chupakabr-ssl

```
<IfModule mod_ssl.c>
<VirtualHost *:443>
        ServerAdmin myltik@gmail.com
        ServerName chupakabr.ru

        DocumentRoot /var/www/chupakabr_ru
#       <Directory />
#               Options FollowSymLinks
#               AllowOverride None
#       </Directory>
        <Directory /var/www/chupakabr_ru>
                Options Indexes FollowSymLinks MultiViews
                AllowOverride All
                Order allow,deny
                allow from all
        </Directory>

        ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
        <Directory "/usr/lib/cgi-bin">
                SSLOptions +StdEnvVars
                AllowOverride None
                Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
                Order allow,deny
                Allow from all
        </Directory>

        ErrorLog ${APACHE_LOG_DIR}/chupakabr_ru-error.log

        # Possible values include: debug, info, notice, warn, error, crit,
        # alert, emerg.
        LogLevel warn

        CustomLog ${APACHE_LOG_DIR}/chupakabr_ru-access.log combined

        SSLEngine on
        SSLCertificateFile /etc/ssl/certs/2_chupakabr.ru.crt
        SSLCertificateKeyFile /etc/ssl/private/3_chupakabr.ru.key
        SSLCertificateChainFile /etc/ssl/certs/1_root_bundle_chupa.crt

    Alias /doc/ "/usr/share/doc/"
    <Directory "/usr/share/doc/">
        Options Indexes MultiViews FollowSymLinks
        AllowOverride None
        Order deny,allow
        Deny from all
        Allow from 127.0.0.0/255.0.0.0 ::1/128
    </Directory>

</VirtualHost>
</IfModule>
```

G. APPROVED VERSION OF DS PROPOSAL

Computing Project Proposal

Version 15 – January 2015

Student's Name: Valeriy Chevtaev

Student's Number: H00022997

Student's Email Address: valeriy.chevtaev@online.liverpool.ac.uk

Project Title: Matrix Barcode Based Single-Sign-On without Trusted Third Party

Proposal Submission Date: 23 Jul 2015

Version Number of the Proposal: 0.3

DA Class ID: UKL1.CKIT.702.H00023862

Name of DA: Yongge Wang

RMT Class ID: LAUR-906-201462-3

Name of GDI: Taly Sharon

Ethical Checklist Completed: (*Yes or No*)**?** Yes

Name of SSM: Anupama Menon

The Programme: MSc in Software Engineering

Domain: CKIT-515, CKIT-510

Proposal approved by: (*To be filled in by the DA*)

Date of the approval: (*To be filled in by the DA*)

Approval confirmed in MiTSA by the Lead Faculty (Dissertation):: (*To be completed by the Lead Faculty*)

Sponsor's Details: N/A

Sponsor's Background: N/A

Sponsor's Agreement: N/A

The Project Aims and Objectives:

The aim of the project is to demonstrate and to prove that it's possible to implement single sign on solution using QR code based application without any third parties involved.

The main objective is to design and implement mobile solution based on QR codes for automating authentication process for a user in existing Internet services, which won't require third parties involved.

Step	Short Description
Hypothesis	It should be possible to use user's device as credentials vault / passwords storage in QR code based authentication strategy, instead of trusting your credentials to 3rd party services like existing solution propose.
Research Methods	1. Research existing QR code based authentication mechanisms; 2. Investigate what open source and free to use libraries for QR code generation and parsing are available in JavaScript and Swift/Obj-C; 3. Investigate on how to automate authentication on gmail.com using JavaScript knowing user credentials; 4. Investigate how to securely setup communication channel between standalone mobile application and bookmarklet (JavaScript code) running in web browser on another device.

IT Artefact	1. There different gmail.com user accounts: uol.test.qrauth1@gmail.com, uol.test.qrauth2@gmail.com, uol.test.qrauth3@gmail.com. User credentials for created accounts should be provided as a part of IT artefact. 2. iOS application as trusted credentials vault with QR code scanning feature, this application will be used to scan QR code generated by a bookmarklet (see point #2) and to provide credentials to a bookmarklet (see point #2); 3. Chrome web browser JavaScript bookmarklet for Gmail web site to be used for QR code generation as well as for user authentication on target web site (gmail.com).
Evaluation	Manual testing of the following points: 1. Make sure that implemented bookmarklet generates QR code which is readable and parseable by implemented iOS application; 2. Make sure that implemented iOS application successfully parses QR code received from a bookmarklet; 3. Make sure that implemented bookmarklet successfully authenticates specified user on target web site (gmail.com); Automated testing of the following points: 1. Functional tests for QR code generation and parsing validation should be implemented as a part of iOS application (build phase).

Project Outline

The idea of this thesis is to design and implement mobile solution for automating authentication process for a user in existing Internet services based on generating and

scanning QR codes realtime. The solution must be implemented in a way that no additional extra integration is required and no third parties should be involved.

The main steps will be:

1. Research on how QR code could be used for authentication;
2. Investigate existing libraries for generating and scanning QA codes;
3. Implement QR code based application for iOS;
4. Implement bookmarklets to be used on client's desktop machine for authentication;
5. Validate that proposed implementation works. Validation criteria will be a proof that the solution authenticates test user on Gmail and Facebook.

Literature Survey / Resources' List:

Some of the related resources mentioned below give an insight on a possibility of QR code based authentication to existing web services, which on the other hand require not only a user's device with photo camera but also extra integration to be done on web service's side, like it's described in [2, Borchert & Gunther] and [9, Gibson]. This complexity may not be an option for target customers (existing web services) as it's leads to extra development costs and potentially new breaches in the system.

All of the rest resources from the list below, like [12, mydigipass.com], [8, eKaay.com] or [17, TIQR.org], provide complete solution for integration of QR core based authentication mechanisms to any web service. As a part of the integration, a separate mobile application is required to be installed on user's device, which may be tricky for a target user in case different web services apply different technologies for QR core based authentication.

Resources list:

1. J.Bonneau,C.Herley,P.VanOorschot,andF.Stajano.Thequesttoreplacepass- words: A framework for comparative evaluation of web authentication schemes. In *Proc. IEEE Security and Privacy*, pages 553–567. IEEE, 2012.

2. B. Borchert and M. Günther. Indirect NFC-login. available at http://www.ekaay. com/ press/Pressekopien/NFC-Login.pdf. In *ICITST*, pages 204–209, 2013.

3. S. Bugiel, A. Dmitrienko, K. Kostiainen, A. Sadeghi, and M. Winandy. Truwal- letm: Secure web authentication on mobile platforms. In *Trusted Systems*, pages 219–236. Springer, 2011.

4. J.J.L. Cobos and P.C. De La Hoz. Method and system for authenticating a user by means of a mobile device, September 2012. US Patent 8,261,089.

5. A. Czeskis, M. Dietz, T. Kohno, D. Wallach, and D. Balfanz. Strengthening user authentication through opportunistic cryptographic identity assertions. In *Proc. 2012 ACM CCS*, pages 404–414. ACM, 2012.

6. D.B. Desoto and M.A. Peskin. Login using QR code, August 22 2013. US Patent 20,130,219,479.

7. B. Dodson, D. Sengupta, D. Boneh, and M.S. Lam. Secure, consumer-friendly web authentication and payments with a phone. In *Mobile Computing, Applica- tions, and Services*, pages 17–38. Springer, 2012.

8. eKaay. http://www.ekaay.com/.

9. S.Gibson.SecurequickreliableloginSQRL,https://www.grc.com/sqrl/sqrl.htm.

10. E. Grosse and M. Upadhyay. Authentication at scale. *Proc. IEEE Security* & *Privacy*, 11(1):15–22, 2013.

11. H. Krawczyk. Hmqv: A high-performance secure diffie-hellman protocol. In *CRYPTO 2005*, pages 546–566. Springer, 2005.

12. MyDigiPass. https://www.mydigipass.com/.

13. K. Reinhardt and B. Borchert. Method and computer program product for pro- viding authorized access to online accounts, 2011. WO/2011/069492.

14. Geak Ring. http://www.igeak.com/Ring/.

15. SECG SEC. Sec2: Recommended elliptic curve domain parameters. *See http: //www.secg.org*, 2000.

16. M. Tanaka and Y. Teshigawara. A method and its usability for user authentication by utilizing a matrix code reader on mobile phones. In *Information Security Applications*, pages 225–236. Springer, 2007.

17. TiQR. https://tiqr.org.

Scholarly Contributions of the Project

In comparison to the literature mentioned above, the solution proposed in this thesis doesn't require extra integration with existing web services and acts as a standalone single sign-on solution running on trusted mobile device which is a great advantage for target users as well as for existing web service owners.

Another advantage of the proposed solution is that it doesn't require any 3rd party credentials vaults to be used as all credentials are stored on a user''s device which acts as a trusted credentials storage.

Implemented artefact and the whole research will introduce new method of authentication in Internet (and potentially not only in Internet) using QR codes.

Implemented artefact could be used later on to open source community or other developers to improve authentication process in their applications.

Description of the Deliverables:

Main deliverables will be:

1. iOS application for QR code authentication;

2. Bookmarklets to be used on client's desktop machine for authentication.

Evaluation Criteria:

Manual testing of the following points:

- Make sure that implemented bookmarklet generates QR code which is readable and parseable by implemented iOS application;
- Make sure that implemented iOS application successfully parses QR code received from a bookmarklet;
- Make sure that implemented bookmarklet successfully authenticates specified user on target web site (gmail.com);

Automated testing of the following points:

- Functional tests for QR code generation and parsing validation should be implemented as a part of iOS application (build phase).

Resource Plan:

The project will require the following resources:

- Personal computer;
- Xcode IDE;
- Obj-c/Swift compiler;
- Internet;
- Pages word processor;
- Numbers spreadsheet application.

All of mentioned resources are already available for using by the author.

No additional resources will be required.

Project Plan and Timing:

July 2015

- QR code possibilities research;
- QR code libraries investigation;

August 2015

- QR code authentication mechanisms research;
- QR code based iOS application implementation;
- Filling up dissertation document;

September 2015

- Initial testing of implemented solution;
- Bug fixing and tuneups of the solution;

October 2015

- Dissertation document detailed manuscript;

November 2015

- Last fixes and tune-ups of the solution;
- Finalising dissertation document.

December 2015

- Shipping final artefacts;
- Shipping final dissertation document.

Risk Assessment:

The following primary risks has been identified:

- Lack of time because of personal circumstances, illness or high work load. It's almost impossible to predict this kind of risks, so I will have to work harder later in case this happens. I can also take a holiday at work in worst case;
- Loss of the thesis work and information because of hardware failure or loss. To mitigate this risk I'm going to use cloud servers for daily/weekly backups;
- Hardware failure. To solve this issue if happened I'm going to buy new hardware and restore the system from a backup.

Additional risks and problems could arise during the implementation of the thesis.

Quality Assurance:

It will be a multilevel quality assurance plan, so it partially could be done in the following ways:

- QA by DA;
- QA by unit and functional testing of implemented software solution;
- Manual testing QA by myself.