Institutional Repository - Research Portal
Dépôt Institutionnel - Portail de la Recherche
researchportal.unamur.be
University of Namur

THESIS / THÈSE

MASTER IN COMPUTER SCIENCE

Authentication in mobile environment

Duchêne, A-S.

Award date:
2013

Awarding institution:
University of Namur

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 23. Jun. 2020

Facultés Universitaires Notre-Dame de la Paix, Namur
Faculté d'Informatique
Année académique 2012–2013

Authentication in mobile environment

Anne-Sophie DUCHÊNE

Maître de stage : M. Dirk BRODERSEN

Promoteur : _____ (Signature pour approbation du dépôt - REE art. 40)
M. Jean-Noël COLIN

Mémoire présenté en vue de l'obtention du grade de
Master en Sciences Informatiques.

Abstract

During the past years we have been witnessing an exponential increase in the use of smartphones. This increased use of smartphones has led to a growing demand and supply of mobile applications: some do not require authentication while others do. For users, having to remember all the passwords and usernames can become a real baffling problem. This problem could be solved by always using the same usernames and passwords for all applications, but it would cause a security problem for users. The challenge here is to allow the user to authenticate into various applications via another application. The use of this application to authentication would avoid some security problems that users might encounter. Indeed, it would reduce the number of aliases and so it would reduce the eventuality of his identity being usurped.
There are already several ways to use this technique as one-time passwords but also techniques of tokens. We will examine the benefits and disadvantages of these solutions and especially the OAuth authentication technique. We will discuss its growing use in the first place and try to understand why this technique is promising if neglected.

Lors de ces dernières années, on a pu voir l'augmentation exponentielle de l'utilisation des smartphones. Cette augmentation de l'utilisation de smartphone a entrainé une demande et une offre grandissante d'applications mobiles, certaines ne demandant pas d'authentification alors que d'autres demandent elles une authentification. Pour les utilisateurs devoir retenir tout les mots de passe et nom d'utilisateurs peut alors devenir un vrai casse-tête. Ce problème pourrait être combler en utilisant toujours le même pour toutes les applications mais cela entrainerait un problème de sécurité pour les utilisateurs. Le challenge ici est de permettre à l'utilisateur de s'authentifier à différentes applications via une autre application. Utiliser cette application pour l'authetification éliminerait une partie des problèmes de sécurité que l'utilisateur pourrait rencontrer. En effet, il réduirait le nombre d'alias qu'il aurait et ainsi il réduirait la possibilité que son identité soit usurpée. Il existe déjà différentes manière d'utiliser cette technique comme les mots de passe uniques mais aussi des techniques de tokens. Nous étudierons les bénéfices et les désavantages de ces solutions et tout particulièrement de la technique d'authentification OAuth. Nous discuterons de son utilisation grandissante dans un premier temps et essayerons de comprendre pourquoi cette technique si prometteuse n'est pas d'avantage utilisée.

Acknowlegdments

This thesis is the result of my four months internship at the Vodafone Group Compagny in Düsseldorf, Germany where I was part of the developement team. Many projects are developed in this team, I was assigned to one of them : the “Net Alert” project. The purpose of this project is to report bugs on the Vodafone network around the world. This project has already been developing for a year. This internship was supervised by Dirk Brödersen. The result of these four months was the development of a new feature in the application and also learning how authentication works with the one-time-password protocol and OAuth protocol.

This internship was a wonderful experience in many ways and I would like to thank some people for this. Jean-Nöel Colin, my supervisor in my faculty, gave my the opportunity of this internship but also his knowledge in the field which helped me during the writing of my thesis and during my internship. I also want to thank Birk Brödersen who was my supervisor in Vodafone during my internship, for his technical knowledge in mobile development but also for allowing me to discover the working place in such a great company. In other words, for everything I learned with him. Moreover, as a member of the solutions team, I want to thank all the members for every little thing they did that made a difference Andrei Abratzou, Mark Brady, Kamilia Bramska, Nicolas Burgos, Carmen Burmeister, Crhistoph Fassbach, Dominic Harrison, Ramunas Jurgilas, Matthew Langham, Zaixiang Meng, Rudy Norf.
I especially want to thank my team on the “Net Alert” project: Dominic Harrison, Kamilia Bramska and Ramunas Jurgilas.

For helping me to finding a place in Düsseldorf, and for helping me finding me my way around the city, I want to thank Denis Ledoux, the student who was the Vodafone intern from FUNDP the year before.

Contents

Abstract **2**

Acknowlegdments **3**

Glossary **8**

1 Introduction **12**
1.1 Problems description & Motivations 12
1.2 Discussion 13
1.3 Research questions 14
1.4 Thesis limits 14
1.5 Thesis structure 14

2 Presentation of the Android platform **16**
2.1 Android Global Architecture 16
2.1.1 The Linux kernel 17
2.1.2 The Android runtime 17
2.2 Android Security 19
2.2.1 Android security guidelines 19
2.2.2 Android platform security architecture 20
2.2.3 Permissions management 21

3 State-of-the-art in authentication **24**
3.1 Authentication systems 24
3.1.1 Authentication factors 24
3.1.2 Multi-factor authentication 25
3.1.3 Cryptographic keys 26
3.1.4 Symmetric keys 26
3.1.5 Asymmetric keys 26
3.1.6 Hash-Based Message Authentication Code (HMAC) 27
3.1.7 Secure Sockets Layers (SSL) and Transport Layer Security (TLS) . 28
3.1.8 Tokenization 29
3.1.9 One-time password 30
3.1.10 HOTP 32
3.1.11 SAML 2.0 33
3.1.12 Shibboleth 34

3.1.13 OpenID . . . 35
3.2 Known lacks of these systems . . . 37
3.2.1 Password breach . . . 37
3.2.2 Principle of least privilege breach . . . 38

4 The most common approaches: tokens and one-time passwords 41
4.1 What is tokenization? . . . 41
4.1.1 What is the interests in tokenization? . . . 42
4.1.2 How does it works? . . . 43
4.1.3 Token creation . . . 44
4.1.4 Token datastore . . . 45
4.1.5 Token storage in applications . . . 45
4.1.6 Authentication . . . 46
4.2 What are One-time passwords? . . . 47
4.2.1 Types and generations . . . 47
4.2.2 Vulnerabilities . . . 47
4.3 Implementation of OTP use on Android platform . . . 49
4.3.1 The two-factors authentication . . . 49
4.3.2 The Google Authenticator . . . 51

5 About Single Sign-On 53
5.1 What is Single Sign-On? . . . 53
5.2 How does it work? . . . 53
5.2.1 The components . . . 54
5.2.2 The flow . . . 54
5.2.3 Architecture of systems using SSO . . . 55
5.3 Security . . . 58
5.3.1 How to make the authentication? . . . 58
5.3.2 Transport of the authentication information . . . 58
5.4 Establishment of SSO . . . 58
5.5 Advantages and limits . . . 59
5.5.1 Benefits . . . 59
5.5.2 Drawbacks . . . 60

6 A way to implement SSO : OAuth 61
6.1 Main interests around OAuth . . . 61
6.2 OAuth 1.0 . . . 61
6.2.1 Why was OAuth 1.0 introduced? . . . 61

6.2.2 How does it works? . . . 62
6.2.3 Authenticating with OAuth 1.0 . . . 64
6.2.4 Advantages, limits and weaknesses of OAuth 1.0 . . . 67
6.3 OAuth 2.0 . . . 69
6.3.1 Why the need of OAuth 2.0? . . . 69
6.3.2 Protocol flow . . . 69
6.3.3 Types of authorization grant . . . 72
6.3.4 Types of client . . . 73
6.3.5 Knwon issues of OAuth 2.0 . . . 73
6.3.6 Is OAuth 2.0 a success? . . . 74
6.4 Implementation of an Android application using OAuth . . . 75
6.4.1 Method 1: Username and password . . . 76
6.4.2 Method 2: OAuth . . . 76
6.4.3 Method 3: Google+ . . . 80
6.5 Comparison between OAuth 2.0 and OpenID . . . 81
6.5.1 Introduction of both protocols . . . 81
6.5.2 Comparison of both protocols . . . 81
6.5.3 A way to combine the two protocols: OpenID connect . . . 82
6.6 Why do few people use OAuth? . . . 83

7 Conclusion **84**
7.1 What is tokenization? . . . 84
7.1.1 On wich conecpt it is based? . . . 84
7.2 How does the one-time password mechanism work? . . . 84
7.2.1 Which are the vulnerabilities? . . . 84
7.3 What is Single Sign-On? . . . 85
7.3.1 What are the interests of using this solution? . . . 85
7.4 Is it possible to do Sing Sign-On with the OAuth protocol? . . . 85
7.4.1 What are the interests of using OAuth? . . . 86
7.5 Remaining leads for future work . . . 86
7.6 What about OAuth in mobile environment? . . . 86

References **89**

Appendices **90**

A Application code **90**

List of Figures

1 Android architecture 16
2 Step of the Java compilation 18
3 Step of the Android compilation 19
4 Permissions ask during the installation 22
5 Android OS Persmissions 23
6 Exchange of messages using a symmetric key 26
7 Exchange of messages using asymmetric keys 27
8 Exchange of messages using HMAC 28
9 TLS handshake 29
10 Generation of OTP 31
11 SAML Roles 33
12 SAML authentication flow [12] 34
13 OpenID authentication flow 36
14 Basic Tokenization Architecture 43
15 Enable two-factor authentication 49
16 Sign in on Google Account 50
17 Enter the validation code 51
18 Simple SSO Architecture 56
19 Portal Web Architecture 57
20 Authentication in traditionnal way 63
21 Authentication on the behalf of someone 63
22 Authentication with a web application 63
23 Authentication flow with OAuth 1.0 67
24 Authentication flow with OAuth 2.0 71
25 First screen 75
26 My ClientID for web application 76
27 The user enters his credentials 78
28 The user allows the application to access his contacts 79

Glossary

The definitions of this glossary are mainly restated from several sources on the Internet.

- **Access Control:** Access control refers to exerting control over who can interact with a resource. Most of the time, this involves an authority, which does the controlling. The resource can be a given building, group of buildings, or computer-based information system.
- **Algorithm:** A set of mathematical instructions that must be followed in a fixed order, and that, especially if given to a computer, will help to calculate an answer to a mathematical problem.
- **Android:** Android is a software stack for mobile devices that includes an operating system, middleware, and key applications.
- **Android Manifest:** The AndroidManifest.xml file is the control file that tells the system what to do with all the top-level components in an application.
- **Application:** A computer program or piece of software designed to perform a specific task.
- **Authentication:** Authentication is the act of confirming the truth of an attribute of a datum or entity. This might involve confirming the identity of a person or software program, tracing the origins of an artifact, ensuring that a product is what its packaging and labeling claims to be.
- **Authorization:** Authorization is the function of specifying access rights to resources, which is related to information security and computer security in general and to access control in particular.
- **Backdoor:** A backdoor is a method of bypassing normal authentication procedures.
- **Basic Access Authentication:** In the context of an HTTP transaction, basic access authentication is a method for a web browser or another client program to provide a user name and password when making a request.
- **Component:** A component is a modular part of a system.
- **Context:** A context is an environment of an application in terms of constraints.

- **Constraint:** Condition that a problem must satisfy.
- **Database:** A large amount of information stored in a computer system in such a way that it can be easily looked at or changed.
- **Device:** Usually a constructed tool, but in this context refers to a smartphones, tablets or even computers.
- **Hash:** An irreversible mathematical function that scrambles data, producing a consistent and unique value.
- **Interoperability:** Being able to exchange and make use of information.
- **Machine Code:** Machine code or machine language is a system of impartible instructions executed directly by a computer's central processing unit (CPU). Each instruction performs a very specific task, typically either an operation on a unit of data (in a register or in memory, e.g. add or move), or a jump operation (deciding which instruction executes next, often depending on the results of a previous instruction).
- **Method (Java):** Name given in Java and other object-oriented languages to a procedure or routine associated with one or more classes.
- **Model:** A description of observed or predicted behaviour of some system, simplified by ignoring certain details
- **OAuth:** OAuth is an open standard for authorization. It allows users to share their private resources (e.g. photos, videos, contact lists) stored on one site with another site without having to hand out their credentials, typically supplying username and password tokens instead. Each token grants access to a specific site (e.g., a video editing site) for specific resources (e.g., just videos from a specific album) and for a defined duration (e.g., the next 2 hours). This allows a user to grant a third party site access to their information stored with another service provider, without sharing their access permissions or the full extent of their data.
- **OTP:** A one-time password (OTP) is a password that is valid for only one login session or transaction. OTPs avoid a number of shortcomings that are associated with traditional (static) passwords. The most important shortcoming that is addressed by OTPs is that, in contrast to static passwords, they are not vulnerable to replay attacks.

- **Policy:** Definite course or method of action to guide and determine present and future decisions.
- **Phishing:** Phishing is an attempt to acquire information (and sometimes, indirectly, money) such as usernames, passwords by masquerading as a trustworthy entity in an electronic communication.
- **Privacy:** Privacy is the state of being free from public attention.
- **Protected Resources:** A protected resource is a resource stored on (or provided by) the server which requires authentication in order to access it.
- **Ressource:** Virtual entity of limited availability that needs to be consumed to obtain a benefit from it.
- **Replay attack:** A replay attack is a form of network attack in which a valid data transmission is maliciously or fraudulently repeated or delayed.
- **Requirement:** A requirement is a need wished for a system.
- **Specification:** Documented detailed requirements with which a product or service has to comply.
- **Smartphone:** A smartphone is a mobile phone built on a mobile computing platform, with more advanced computing ability and connectivity than a feature phone.
- **SMS:** Short Message Service.
- **SMS Spoofing:** SMS spoofing is a relatively new technology which uses the short message service (SMS), available on most mobile phones to set who the message appears to come from by replacing the originating mobile number with alphanumeric text. Spoofing has both legitimate uses (setting the company name from which the message is being sent, setting your own mobile number, or a product name) and illegitimate uses (such as impersonating another person, company, product).
- **SSO:** Single sign-on (SSO)is a property of access control of multiple related, but independent software systems.
- **Tokenization:** Tokenization is the process of replacing some piece of sensitive data with a value that is not considered sensitive in the context of the environment that consumes the token and the original sensitive data.

- **Vulnerability:** In computer security, a vulnerability is a weakness which allows an attacker to reduce a system's information assurance.
- **URL:** A uniform resource locator (URL) is a specific character string that constitutes a reference to an Internet resource.

1 Introduction

The context of this thesis is presented here. The environment and the motivations are introduced, as well as the problems and issues I found. The limits of the works are stated, followed by the structure of the thesis.

1.1 Problems description & Motivations

Nowadays, more and more people own a smartphone (for private use, for work or both), whether to work, play or just stay connected at all times. According to a research of Gartner ([1]), in 2012, 812 million of smartphones were bought. This research proves that smartphones are omnipresent in the world and this trend is not ready to change. These phones support different operating systems like Android from Google, Blackberry OS, iOS from Apple and many others, all offering a range of various applications. These applications offer new functionalities, new experiences to enjoy, to help in everyday life, to work, to socialize, to organize events, to go somewhere and even more. In the end of 2012, Apple announced that we could find 700.000 applications on the Apple Store. A few weeks later, Google announced the same number for the Google Market[2]. According to statistics, 79% of data usage is used by iPhone users to download applications, whereas for Android users, the percentage is 74%[3]. In the end of 2011, Google published an infographic to celebrate the 10 billions of downloads on its Android Market[4]. We can see that the most popular download categories are games, followed by entertainment applications, tools, communication and productivity.

To use most of these applications, the customer needs to enter credentials - generally a username and a password - to access the application. Several ways to provide secure authentication are avaiable at the moment. I will partly discuss about the various solutions for authentication, but this thesis focuses on the work of the SSO property and OAuth protocol, as this is the subject I studied while working four months at Vodafone.

Like Anthony Plewes says -[5]- authentication is a big issue on the web or in the mobile environment. How can we trust the website or the application we are giving the information to. Being sure that you are dealing with a true party is a huge challenge. We need to find a balance between security, usability and cost.

> *"Traditional authentication techniques for online and telephone sessions between two parties are problematic. Typically, organizations ask questions and expect pre-defined answers from the calling party, such as their mother's maiden*

name, name of their first pet, or first school. These answers are static, and are easily available online in some cases, in others they can be guessed, phised or vished. When Sarah Palin's web mail account was famously hacked, for example, it was compromised by a college student who researched the answers to her security questions online."[5]

During my internship, I was constantly facing the authentication issues. The application created by Vodafone used a username/password system to authenticate the users. Thanks to this, I was able to understand the complexity, the strengths and the weaknesses of this type of protocol which is the mostly used. I asked Dirk why they did not use the OAuth protocol instead of the this system. He explained to me that the OAuth protocol was not old enough to be fully trusted. Few developpers understand all the subtleties of this protocol: since it is only seven years old. From then on, I understand better the reason why the OAuth protocol is so rarely chosen.

The difference between these two types of authentication will be explained later in this thesis. I will try to see if it would have been possible to use the OAuth protocol on an application like "Net Alert". I will also try to discover if the OAuth protocol can fix known issues of the existing authentication systems.

1.2 Discussion

What are we trying to fix with the new protocols like OAuth?

- Solving the known flaws of old authentication systems: passwords breach for example.
- Improving the user experience when he uses applications.
- Improving protection of the user information (username, password, email, and so on).

We can see through these points that the challenge for this new protocol is huge. Indeed, a lot of things need to be changed, not only the way to implement the application but also the behavior of the users. On the implementation side, it is important to try to remove the issue linked to the transport of password around the network. On the user side, it might be important to change the habits like using the same password all the time.

The question linked to the implement side might be solved with the use of OAuth, on the user issues the SSO property might be the answer.

1.3 Research questions

Based on the introduction above and on discussions, the purpose of my thesis is to learn the differents ways of authentication in the mobile environment. This thesis will have a big focus around the OAuth protocol: how does it work, why is it so innovative, and so on. Before talking about OAuth, some concepts will be explained as for instance the tokenization and the one-time password or even the Single Sign-On concept. It is important to introduce them to have a better understanding.

Here are some questions that need to be answered:

1. Question 1: What is tokenization?
    - On which concepts is it based?
2. Question 2: How does the one-time password mechanism work?
    - Which are the vulnerabilities?
3. Question 3: What is Single Sign-On?
    - What are the interests of using this solution in mobile environment?
4. Question 4: Is it possible to do Single Sign-On with the OAuth protocol ?
    - What are the interests of using OAuth?

1.4 Thesis limits

We know that the authentication problem has been a controversial subject for many years: there are so many ways to authenticate persons or even resources. Keeping this in mind and the fact that I had a limited time available to work on the subject, my work is focused on one protocol: OAuth and on one property: Single Sign-On. Nevertheless, some other protocols or ways to authenticate users will be exposed but only explained briefly.

1.5 Thesis structure

Based on the previous sections, the thesis structure can be described.

In section 2, I will introduce the background content which helps the readers to have a better understanding of the Android platform: global architecture, security, permissions

management. It is important to introduce this concept to help the readers when they will read the parts on the authentication methods on Android platform.

In section 3, the state-of-the-art in authentication and the known flaws of these sytems will be given. Concepts and systems of authentication will be explained: what are the authentication factors?, what is the two-factor authentication? and even some of cryptographic mechanisms. The second part will focus on the known lacks of these systems and more specifically those in direct link with passwords and users information. So, both sides will be treated: the secure authentication development and the associated failures. After this part the readers will be able to have a global view of authentication environment. The second part will introduce issues that I will next try to provide solutions for.

In section 5, the tokenization approach and the one-time password approach will be explained: in which way tokenization can solve authentication problems raised in section 3, how are tokens created or stored? One-time password will as well be explained: how are they generated?, what are the different types? In the end of the section, an example of an OTP implementation on Android platform will be given. This example presents a way to use the techniques described before to strengthen the security in authentication to the readers.

All the issues raised in section 2 cannot be solved by tokenization or one-time passwords. So I will introduce the property of Single Sign-On in section **??**: what is it?, how does it work?, what are its positive aspects and weaknesses?. Unfortunately, Single Sign-On is only a property: we need a protocol to implement it.

Section 6 will be about a way to implement SSO: OAuth. This section deals with the birth of the protocol, its interests and the reasons why it is so rarely used. An example of an Android application I created will be given and a comparison between an existing Single Sign-On solution - OpenID - and OAuth will be done.

The conclusion will summarize my research, list the answers I found and the remaining limitations.

2 Presentation of the Android platform

The introdiuction of the thesis purpose and the possible problems having been presented, this section will allow the reader to have a better understanding of the area in wich the work is confined. The first part is an explanation of the Android system. The second part is about security in Android.

2.1 Android Global Architecture

> *"Google's Android system is a comprehensive software framework for mobile devices (Smartphones, tablets, ...). Android includes an operating system, a middleware and a set of applications rounding it."*[6]

Android is a multi-process system based on a Linux kernel in which each application runs with its own user id. Applications are written in Java and compiled in a Dalvik Exectuable which is a custom byte-code linked to Android. Each application is distributed under the form of a package, with ".apk" extension, which is similar to ".jar" in Java.

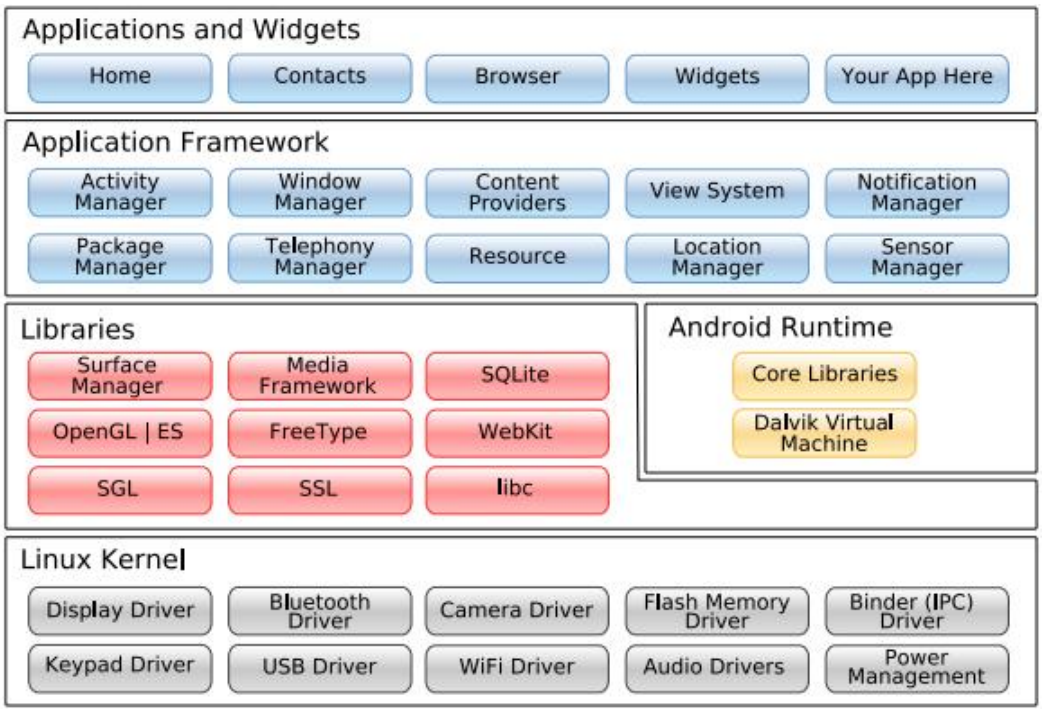

Figure 1: Android architecture

Figure 1 is the diagram of the Android architecture. The architecture is divided into levels of components which constitute the operating system. The lower component is the linux kernel, its is the most remote component for the user. Whereas, the applications are at the top of the Android architecture.

2.1.1 The Linux kernel

Android uses the Linux kernel, which allows to create the bridge between the hardware and the software. For example, the wifi pilotes allow to control the wifi chip. When the user wants to use the wifi, the chip needs to be activated, that is the reason why the software uses some functions like "turnwifion()" and this function will make the link with the wifi chip. There will be a unique function for all the chips but the content of the function will be specific to each hardware.

The Linux kernel used by Android, has been especially designed for the mobile area with a more effective management of the battery and a more particular management of the memory. This layer allows Android to be compatible with so many different supports.

On the plan, we can see that the Linux kernel is the only layer which handles the hardware. Android does not handle the hardware at all. Indeed, it is the kernel that handles the hardware. If one developer wants to add some material, he needs to work on the kernel and not on the Android specific layer.

2.1.2 The Android runtime

Because of this layer, Android is not a simple implementation of Linux but a specific implementation of Linux. Indeed, this layer has some basic java libraries with some specific Android librairies and also a virtual machine: the"Dalvik".

A *runtime system* is a program which allows the execution of other programs. For instance, the JRE (Java Runtime Environment) is used for Java applications.

Figure 2 presents the different steps needed for the compilation and the execution of a standard Java program.

The Java code is a line of instructions in a file (.java file) which is translated into another line of instructions in another language: the "bytecode". This one is in a .class file. The "bytecode" is the language that a virtual Java machine can understand and

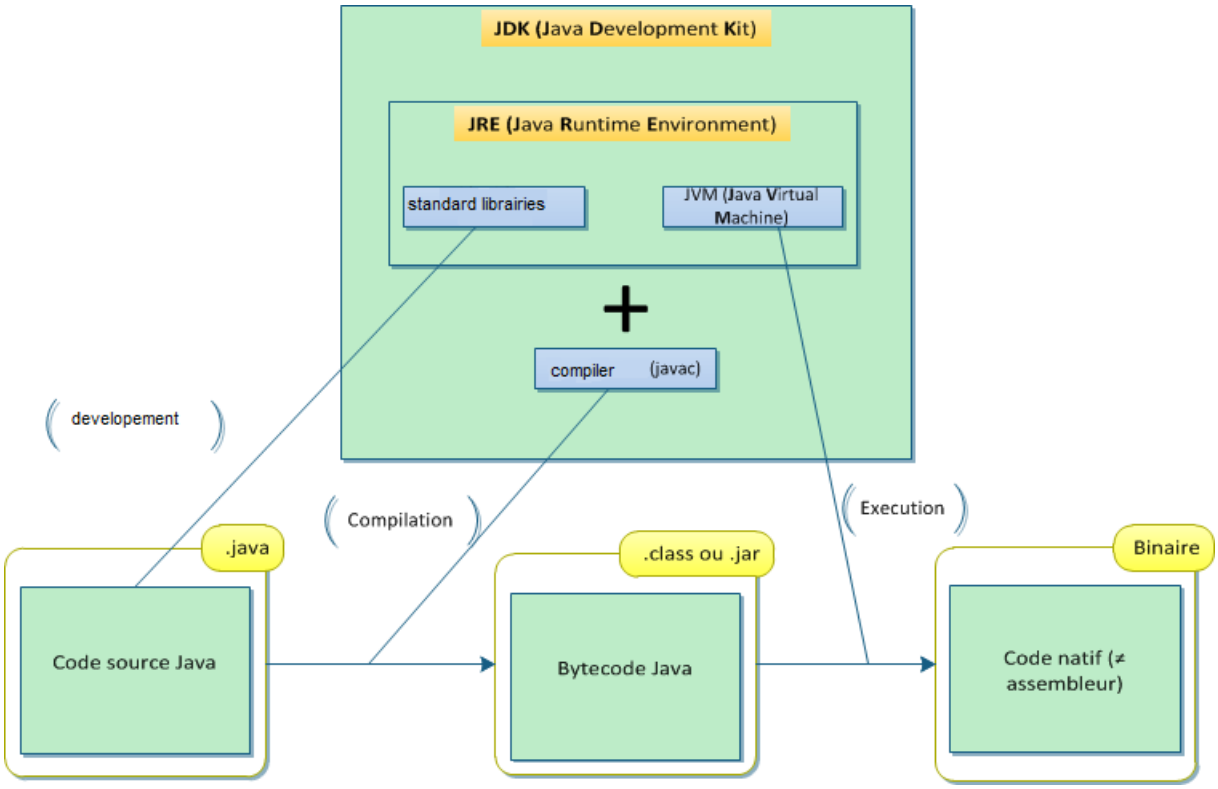

Figure 2: Step of the Java compilation

interprete. Later the different .class files are compiled in .jar file which is an executable file.

As far as Android is concerned, the process is different. The version of Java which is used for the Android development is an adapted version of Java for the mobile environment. Some of the fonctionnalities specific to Java are removed. For example, the graphic library specific to Java is removed but a specific to Android is used. Moreover, Android does not use the Java virtual machine: a machine for the embedded systems has been developed: the "Dalvik" machine. This machine is adapted and optimised to have a better management of the physical ressources. In Android, the memory space is a huge problem, so for example, this virtual machine is adapted to best handle the memory space and also to use less battery that a Java virtual machine.

The biggest characteristic of the "Dalvik" virtual machine is that it allows to create an occurrence of some objects, which is called the instantiate of the object. In Java when you create an object, one class is created. In Android when you create an object, an occurence of "Dalvik" is created so they can live with each others without any disturbance.

Figure 3 illustrates the steps needed for the compilation and the execution of a Android standard program.

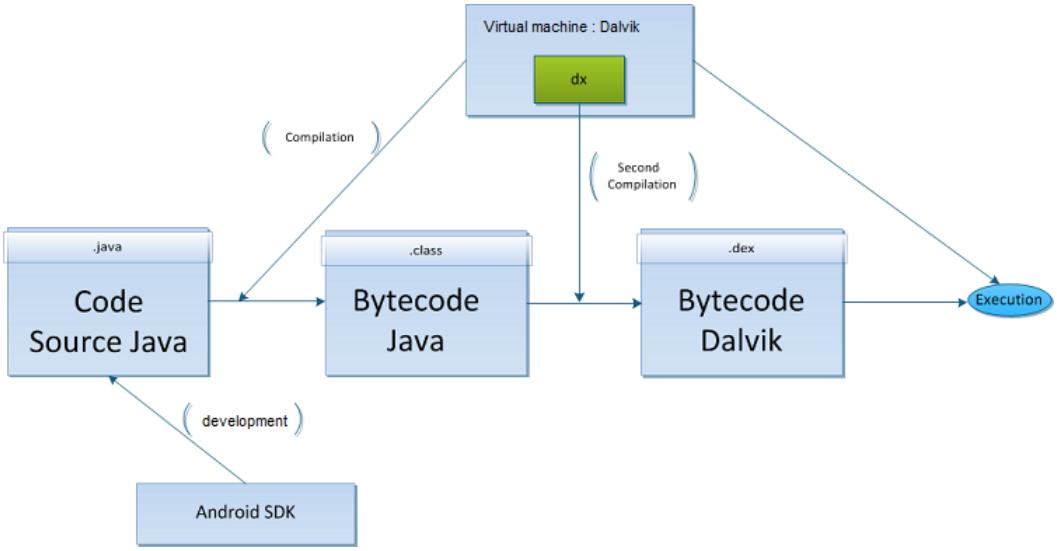

Figure 3: Step of the Android compilation

On the diagram, we can see that the Java code is converted in bytecode as before but we know that the Java bytecode can only be understood by a Java virtual machine. So we need to do another conversion between the Java bytecode and the Dalvik bytecode. The extension of those file is .dex which can be understood by the Dalvik virtual machine.

The Dalvik virtual machine is only for Android, but it is possible to develop a similar virtual machine which cannot work on Android. For example, the Nokia N9 can execute Android programs without the Dalvik virtual machine.

2.2 Android Security

The description given here is extracted from the Android security overview.[7]

2.2.1 Android security guidelines

Early in the Android development, the team responsible of the development recognized that a strong security model was requested to a great ecosystem of application or devices built on and around the Android platform. Moreover, the Android team had the opportunity to observe their competitors and took advantage of their weaknesses to build

a better program.

The team set guidelines to respect to improve the security in Android programs.

The components are:

- **Design Review:** The Android security process begins with the creation of a rich and configurable security model and design. Each major feature of the platform is reviewed by engineering and security resources, with appropriate security controls integrated into the architecture of the system.
- **Penetration Testing and Code Review:** Android-created and open-source components are subjects to strong security reviews all along the development. These reviews are performed by several people: the Android Security Team, Google's Information Security Engineering team, and independent security consultants. These reviews allow to identify weaknesses and possible vulnerabilities before the platform is open-sourced, and to simulate the types of analysis that will be performed by external security experts upon release.
- **Open Source and Community Review:** The Android project allows security review by any interested parts. Android also uses open source technologies which are subject to significant external security review, such as the Linux kernel.
- **Incident Response:** Sometimes, even with all of these precautions, security issues may appear after shipping, which is why the Android project has created a comprehensive security response process. Upon the discovery of legitimate issues, the Android team has a response process that enables a fast mitigation of vulnerabilities to ensure that potential risk to all Android users is minimized.

2.2.2 Android platform security architecture

Android seeks to be the most secure and usable operating system for mobile platforms by re-purposing traditional operating system security controls to:

- Protect the user data: his credentials, his phone number, and so on.
- Protect system resources (including the network).
- Provide application isolation.

To respect these goals, Android provides the following key security features:

- A strong security at the OS level thanks to the Linux kernel.
- A mandatory application sandbox for all applications to manage the permissions.
- A secure interprocess communication.
- An application signing.
- An application-defined and user-granted permissions.

Figure 1 summarizes the security components and considerations of the various levels of the Android software stack. Each component assumes that the components below are properly secured. With the exception of a small amount of Android OS code running as root, all code above the Linux Kernel is restricted by the Sandbox application.

2.2.3 Permissions management

2.2.3.1 Android Sandbox

In the context of my thesis, it is important to talk about the permissions management in Android. Indeed, the user needs to be able to control the access to his information. In Android, the Sandbox application fills this goal on the kernel level.

In Android, each applications runs with a unique user ID in a separate process. This approach is different from other operating systems, where multiple applications run with the same user permissions.

This sets up a kernel-level application Sandbox. The kernel enforces security between applications and the system. By default, applications cannot interact with each others and they have limited access to the operating system. If application X tries to do something malicious like reading application Y's data, then the operating system is protected against this because application X does not have the appropriate user privileges. The Sandbox is simple, auditable, and based on decades-old UNIX-style user separation of processes and file permissions.

Since the Sandbox is in the kernel, this security model extends to native code and to operating system applications. All of the software above the kernel in Figure 1 runs within the Sandbox application.

2.2.3.2 *The Android permission model*

The sandbox is the kernel security for the management permissions. By default, an application has access to a limited range of system resources. For example, an application cannot manipulate the SIM card. To make use of the protected APIs on the device, an application must define the capabilities it needs in its manifest. When preparing to install an application, the system displays a dialog to the user that indicates the permissions requested and asks whether to continue the installation. If the user continues the installation, the system accepts that the user has granted all of the requested permissions.

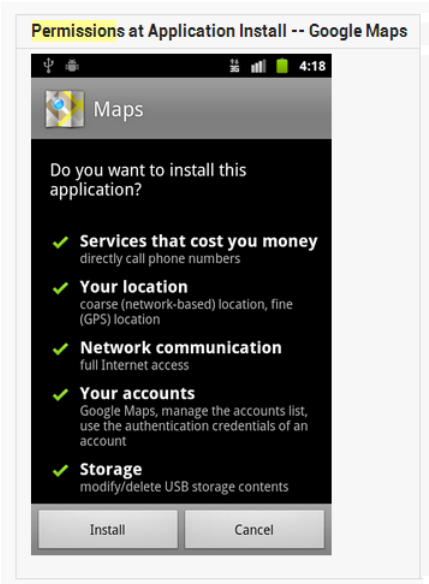

Figure 4: Permissions ask during the installation

Having the user review permissions at install time gives the user the option to not install the application if they feel uncomfortable with it. The required permissions by an application are in the Android manifest.

Regarding the personal information, Android places APIs that provides access to user information into the set of protected APIs. The third-party applications can also accumulate some personal information about the user. They can choose to share these information. If it is the case, they can use Android OS permission checks to protect the data from third-party applications.

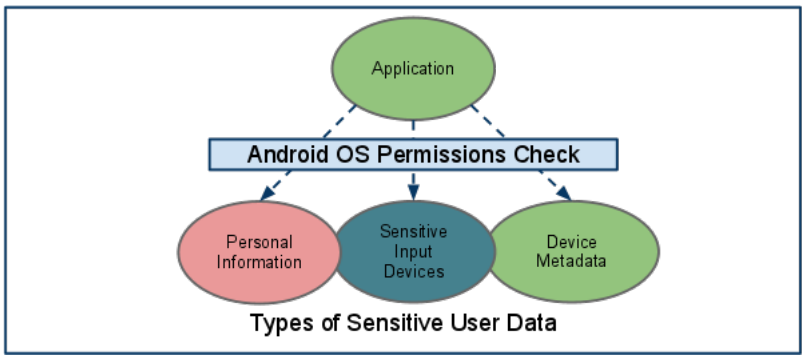

Figure 5: Android OS Persmissions

System content providers that are likely to contain personal or personally identifiable information - contacts or calendar - have been created with clearly identified permissions. This granularity provides the user with a clear indication of the types of information that may be provided to the application.

3 State-of-the-art in authentication

This section presents some of the possibilities of existing authentication systems. For a better understanding, I will explain the authentication factors before giving a list of the most known authentication systems. The second part of this section presents the known lacks of these systems.

3.1 Authentication systems

When we talk about authentication, in any field, we talk about an act which guarantees a piece of information to be true. Is that person or entity really what it claims to be? Authentication often involves verifying the validity of at least a form of identification. Once they are authentificated, users can do some operations to get or store some information.

We already know a lot of ways to authentificate, all having positive and negative aspects. The authentication process can be divided into three categories, based on what we called the factors of authentication: something you know, you have or you are.

3.1.1 Authentication factors

We always consider three types of factors for authentication:

- The inference factors: it is something that the user is or can do.
- The ownership factors: it is something that the user has.
- The knowledge factors: it is something the user knows.

The knowledge factors are by far the most popular factors. The common examples are passwords and personal information. We need to keep in mind that personal information is not necessarily secret, but is assumed to be unknown by anyone else. We use this sort of information to enter our mailbox, or create a passphrase for an encrypted key.

The ownership factors develop more and more as signet rings and passports reveal. Such objects are collectively called tokens. Some tokens perform sophisticated authentication functions, such as providing protected storage for cryptographic keys and performing cryptographic operations.

The inference factors are less used than the two others, at least for on-line authentication. This is either a physical (as with fingerprints) or behavioural (as with typing patterns) characteristic of a person. Authentication methods based on this factor are commonly called biometrics.

3.1.2 Multi-factor authentication

We talk about "multi-factor" authentication when the authentication uses a combination of more than one of the factors described earlier.

> *"Multifactor authentication is a great way to secure trusted computers and better secure remote access to networks. It does not solve all problems, but it does solve the problem of passwords being brute-forced and passwords being shared."*

Since we have three factors, it is obvious that we have three possibilities:

- Single-factor authentication: only factor is needed as for example a password.
- Two-factors authentication: two of the the three factors are needed as for example accessing your account through an ATM by using two things: the PIN (something you know) and the ATM (something you have).
- Three-factors authentication: all three of the factors of authentication are needed. In other words, if you want to access a secure site you might need to go through security: your face will be compared to a stored image (something you are), and swipe an access card (something you have), and enter a four-digit code (something you know).

Multi-factors is either two-factors or three-factors. The strength of authentication keys can vary even within a factor category. However, it is generally held that multi-factors authentication improves security. For example, it is more difficult to hack two factors than one. That is why, to get into a secure building, you need to steal or copy an access card, and find out the access code and fool the guard checking your face on their system.

If we want to keep using this system we need to do some adjustments.

> *"At the same time, the current approach that requires an end user to carry an expensive, single-function device that is only used to authenticate to the network is clearly not the right answer. For two-factor authentication to propagate on the Internet, it will have to be embedded in more flexible devices that can work across a wide range of applications."*[8]

3.1.3 Cryptographic keys

Cryptographic keys are used in cryptographic algorithms, such as digital signature schemes and message authentication codes. We need the keys to assume the secrecy and security of authentication. Encryption algorithms which use the same key for both encryption and decryption are known as symmetric key algorithms. The others are called asymmetric key algorithms.[9]

Digital signature is a way to demonstrate, via a mathematical scheme, the authenticity of a document or a message. A valid digital signature proves that a message was sent by a known person, and that the message was not corrupted during the transit.

3.1.4 Symmetric keys

As said before, symmetric key cryptography - also called private key or secret key - uses the same key to encrypt or decrypt something. Thanks to this definition we can explain, why it is so important to keep the key private but also to ensure that the distribution of these keys is secured.

The following picture illustrates an exchange between two people using a symmetric key. Alice must first transmit the symmetric key "XYZZY" to Bob via a secure channel. After the key is received, Alice can then encrypt the plaintext with the same key and transmit the ciphertext to Bob, who can then decrypt the ciphertext using the same key. [10]

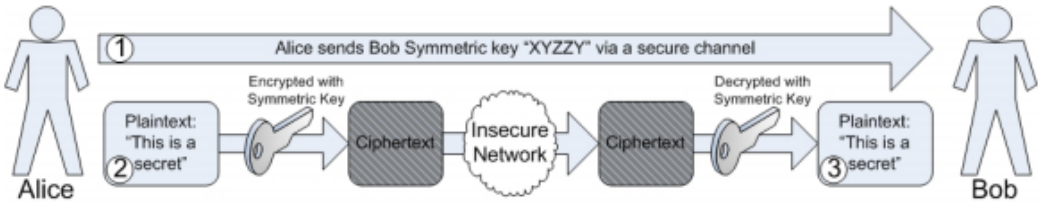

Figure 6: Exchange of messages using a symmetric key

All the security of this system holds on the exchange of the symmetric key. If the symmetric key is intercepted, the whole security is compromised.

3.1.5 Asymmetric keys

Asymmetric key encryption - also called public key encryption - uses two different keys: a public and a private. If a key is used to encrypt the data, only the other one can decrypt the data.

There are many uses of asymmetric key encryptions. Here are the most popular:

- Public-key encryption: Using an asymmetric key encryption is a good way to ensure the confidentiality of the message. Indeed, encrypting a message with a public key, ensures that only the owner of the secret key will be able to decrypt the message.
- Digital signatures : The public key can be used to check the identity of a sender. If he signed with his private key, his identity can be verified with his public key. This also ensures that the message has not been altered.

The diagram below explains how it works. Only Alice has access to her secret key. So, if Bob is able to decrypt a message with Alice's public key, he can be sure that Alice wrote it. [10]

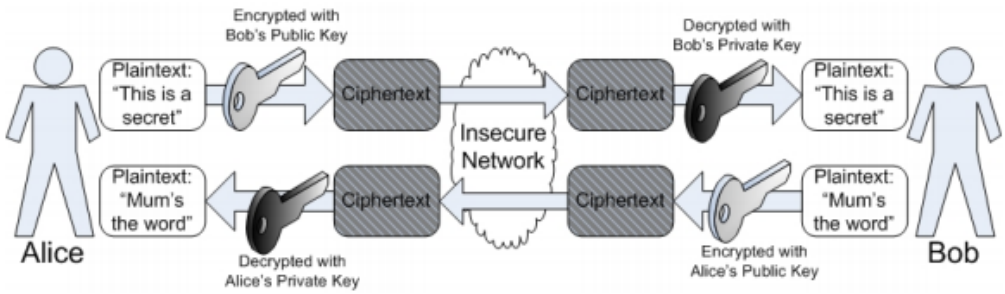

Figure 7: Exchange of messages using asymmetric keys

3.1.6 Hash-Based Message Authentication Code (HMAC)

In cryptography, we can also use keyed-hash message authentication code (HMAC) for authentication. As with any message authentication code (MAC), it may be used to simultaneously verify both the data integrity and the authentication of a message. The strength of HMAC depends on the properties of the underlying hash functions.

A MAC is produced from a message and a secret key by a MAC algorithm. One of the main properties of a MAC is that it is impossible to produce the MAC of a message and a secret key without knowing the secret key. A MAC of the same message produced by a different key looks unrelated. Even knowing the MAC of other messages does not help in computing the MAC of a new message.

For example, pretending if Alice keeps her secret key to herself and only uses it to compute MACs of messages that she stores on a cloud server or other unreliable storage media. If

she later reads back a message and sees a correct MAC attached to it, she knows that this is one of the messages that she stored in the past.

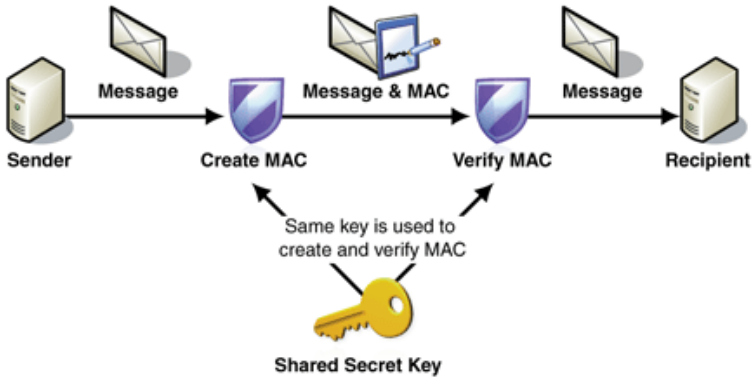

Figure 8: Exchange of messages using HMAC

3.1.7 Secure Sockets Layers (SSL) and Transport Layer Security (TLS)

Another use of public-key encryption is the Secure Sockets Layer (SSL) protocol. This security protocol can allow safe transmission of sensitive data for browsers and web servers. SSL is the precursor of the Transport Layer Security (TLS) protocol.

These two protocols use different techniques like asymmetric cryptography for authentication of key exchange, symmetric encryption for confidentiality or message authentication code for message integrity.

Once the client and the server decide to use TLS, they initiate a trustfully connection by using a handshaking procedure. There are some steps to follow:

- Initialy, the client goes to the server and asks it to authentify. At the same time, the client sends the server some information like his SSL version number, and so on.
- The server sends the certificate which is signed by a control authority back. Amongst other things, this certificate contains the public key of the server.
- Using the data from the handsake, the client can check the validity of the server's certificate. If the validity is verified, the client creates a random secret key. After that encrypts it with the server public key and finally sends the result to the server.
- When the server receives the secret key, he can decrypt it with its private key. After, booth client and server have a secure channel to talk on: they share a secret.

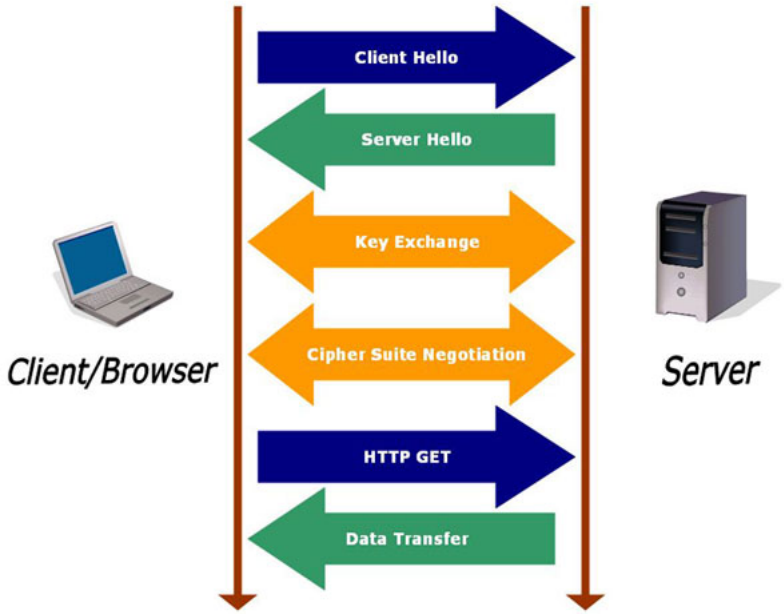

Figure 9: TLS handshake

3.1.8 Tokenization

Tokenization is the process of breaking a stream of text up into words, phrases, symbols, or other meaningful elements called tokens.

In data security this process can be used to substitute a sensitive data. To ensure safety, the tokens used, are often the ones with no direct mathematical relationship between the token and the original value associed. That way the value cannot be extracted from the token. The association between the token and the original value is maintaned in a database and there is no other way or place to connect them.

The process of tokenization can solve a lot of critical problems like protecting sensitive values within applications, database, storage, and so on. For example, token can be used to replace credit card numbers.

This mechanism his used in a lot of common applications, so it will be the center of another section of my thesis. In this section more aspects of this process will be developed like interests or tokens generation. ,

3.1.9 One-time password

One-time password (OTP) is a password which can be used a single time for a single transaction. Unlike static passwords, the one-time password changes each time the user logs in.

The main advantage of OTP is that, unlike static password, it is not vulnerable to replay attacks. Indeed, even if it has been discovered through a log or a packet capture, it can not be used twice, so it is useless. Unfortunately, OTP can be diffcult to memorize for a human being. Regarding to the security, the main problem with OTP is that it can not guarantee protection against Man-in-the-middle attacks: the attacker can use the one-time password before the user.

OTP can be generated and distributed in different ways.

The one-time password can be generated by counter-synchronized, based on a challenge or time-synchronized. The two first methods use mathematical algorithms.

> *"A counter-synchronized OTP solution synchronizes a counter between the client device and the server. The counter is advanced each time an OTP value is requested of the device. Just like with time-synchronized OTPs, when the user wants to log on, he enters the OTP that is currently displayed on the device."*[8]

> *"Challenge-based OTPs are a special case and also often use a hardware device. However, the user must provide a known value, such as a personal identification number (PIN), to cause the OTP to be generated. This type of OTP is currently being rolled out in Europe to add authentication to credit and debit cards. The OTP solutions in use today are all built on some sort of cryptographic processing to generate the current password from a synchronization parameter (that is, the time or counter value), a secret key, and possibly a PIN."*[8]

There are different ways to deliver the OTP:

- **Text messaging:** This is the easiest way to communicate the OTP. Indeed, text messaging is a ubiquitous communication channel, text messaging has a great potential to reach all consumers with a low total cost to implement. But if the user often needs a new OTP the cost can becom an issue. In addition to threats from hackers, the mobile phone operator becomes part of the trust chain. So anyone using this information can create a man-in-the-middle attack.

- **Mobile phone:** Using a mobile phone ensures a low cost since most of customers already have their own mobile phone. Moreover, the tools required for OTP are on the phone. Several OTP applications can run independently, and so the mobile phone can be used to authenticate with OTP to multiple resources.

- **Proprietary tokens:** Like all tokens, they can easily be lost, damaged or even stolen. A variant of the proprietary token was proposed by RSA in 2006 and was described as "ubiquitous authentication", in which RSA would partner with manufacturers to add physical SecurID chips to devices such as mobile phones.

- **Web-base methods:** Authentication-as-a-service providers offer some web-based methods to deliver OTP without using tokens. When first registering on a website, the user chooses several secret categories of things, such as dogs, cars, boats and flowers. Each time the user logs into the website they are presented with a randomly generated grid of picalphanumeric character overlaid on it.

- **Paper:** Sometimes paper is used to deliver OTP like in Germany for the web banking codes called TANs (transaction authentication numbers).

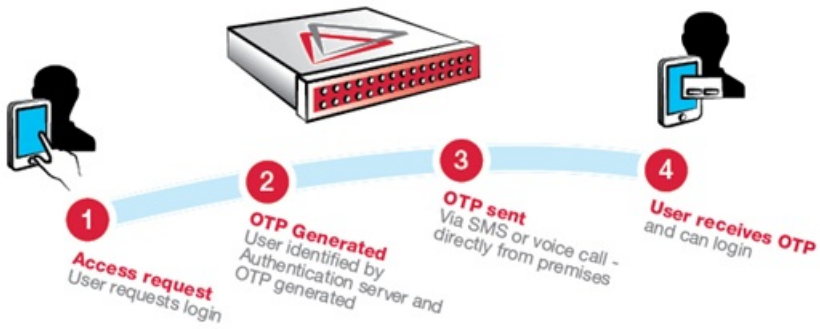

Figure 10: Generation of OTP

"One-Time Password is certainly one of the simplest and most popular forms of two-factor authentication for securing network access. [...] One-Time Passwords are often preferred to strongerforms of authentication such as Public-Key Infrastructure (PKI) or biometrics because an air-gap device does not require the installation of any client desktop software on the user machine, therefore allowing them to roam across multiple machines including home computers, kiosks, and personal digital assistants." [11]

3.1.10 HOTP

HOTP is a HMAC-based One-Time Password algorithm. To explain how this protocol works, we will extract some parts of the RFC4226 article - [11].

The algorithm has fundamental preriquisites:

1. The algortihm must be sequence-based or counter-based.
2. The algorithm should use minimal resources to implement in hardware by minimizing requirements on battery, number of buttons, and size of LCD display.
3. The algorithm must work with tokens that do not support any numeric input, but may also be used with more sophisticated devices such as secure PIN-pads.
4. The value displayed on the token must be easily read and entered by the user: this requires the HOTP value to be of reasonable length. The HOTP value must be at least a 6-digit value.
5. There must be user-friendly mechanisms available to resynchronize the counter.
6. The algorithm must use a strong shared secret. The length of the shared secret must be at least 128 bits. This document recommends a shared secret length of 160 bits.

The HOTP algorithm:

$$\text{HOTP(K,C)} = \text{Truncate(HMAC-SHA-1(K,C))}$$

With:

- C: 8-byte counter value, the moving factor. This counter must be synchronized between the HOTP generator (client) and the HOTP validator (server).
- K: shared secret between client and server, each HOTP generator has a different and unique secret K.
- Truncate: represents the function that converts an HMAC-SHA-1 value into an HOTP value.

3.1.11 SAML 2.0

The SAML 2.0 (Security Assertion Markup Language) is a standard to exchange authentication and authorization data between security domains. This protocol is based on XML which uses security tokens. These tokens are used to pass information from a SAML authority to a SAML consumer. By SAML authority, we are talking about an identity provider and for the SAML consumer we are talking about a service provider. SAML 2.0 enables web-based authentication and authorization scenarios including cross-domain Single Sign-On (SSO).

SAML defines a message format, a communication protocol that allows to exchange authentication information, attributes and autorisation in a secure way between partners. SAML has three types of statements: the authentication statement that confirms one user has been authenticated, the attribute statement that affirms one user has been given an attribute and the authorization descision statement that affirms one user has received the authorization to do an action.

3.1.11.1 The different components

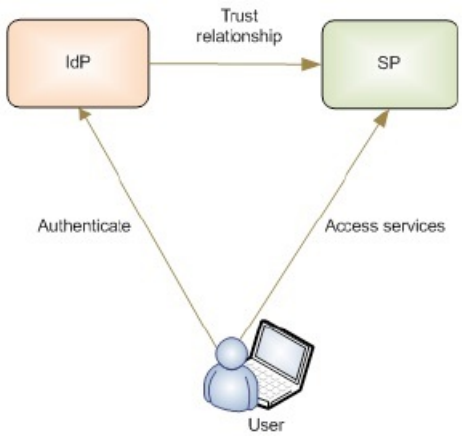

Figure 11: SAML Roles

The identity provider assumes the role of the asserting party and the service provider is used for the relying part.

3.1.11.2 Example of authentication with a web browser

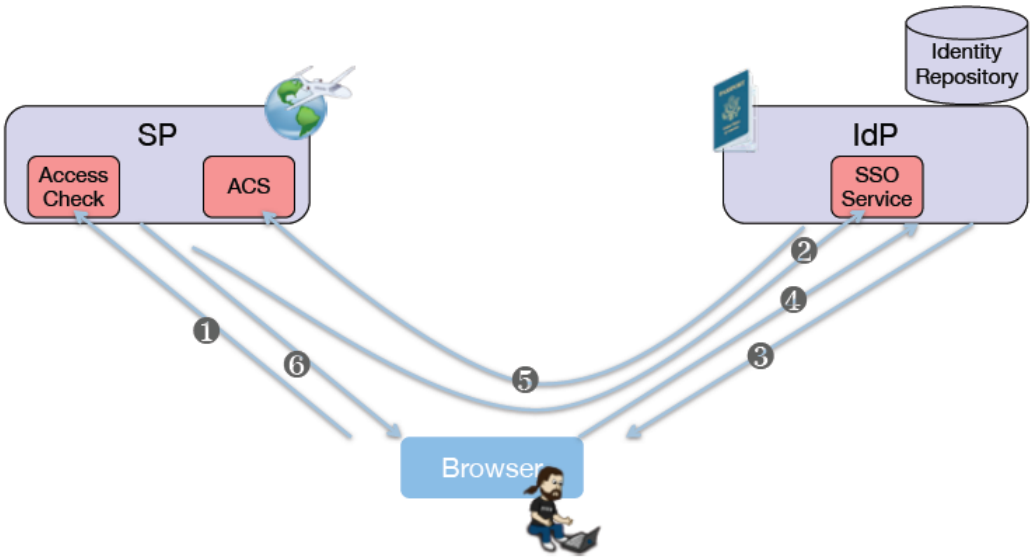

Figure 12: SAML authentication flow [12]

With :

1. Request of access
2. Request of authentication
3. Authentication
4. Authentication
5. Answer of the authentication
6. Access to the ressource

3.1.12 Shibboleth

> *"Shibboleth is among the world's most widely deployed federated identity solutions, connecting users to applications both within and between organizations."[13]*

Shibboleth is a logging-in system for the Internet and computer networks. It allows users to sign by using their username to various systems run by federations of different

organizations.

The Shibboleth middleware initiative created an architecture and open-source implementation for identity management and federated identity-based authentication and authorization (or access control) infrastructure based on Security Assertion Markup Language (SAML). This allows cross-domain single sign-on and removes the need for content providers to maintain user names and passwords. Identity providers (IdPs) supply user information, while service providers (SPs) consume the information and give access to secure content.

Shibboleth is a web-based technology which implements the HTTP/POST and attributes push profiles of SAML including the Identity Provider and the Service Provider.

Here are the steps of the standard use case:

1. The user accesses a ressource which enabled the Shibboleth content protection.
2. The service provider transmits an authentication request that is passed through the brower using the URL.
3. The user is redirected to the identity provider. After, he authenticates to an external acces control.
4. Shibboleth generates an authentication assertion with a temporary "handle" contained within in. This one allows the identity provider to recognize particular request.
5. The user is posted to the assertion consumer service of the service provider whiwh consumes the assertion and asks an attribute query to the identity provider.
6. The identity provider sends an attribute assertion containing trusted information.
7. The service provider either makes an access control decision based on the attributes or supplies information to applications which decide by themsleves.

3.1.13 OpenID

> *"Can't remember your passwords? Tired of filling out registration forms? OpenID is a safe, faster, and easier way to log in to web sites.[14]"*

OpenID is a decentralised authentication system which allows a unique authentication and the attributes sharing. A user who possesses an openID account, can access all the websites using OpenID with a unique id.

3.1.13.1 Actors

In the openID flow, four actors are involved:

- The user who wants to access the resource.
- The relying party (RP or consumer) which wants to prove the identity of the user. It can be a website, a webservice, a application, and so on.
- The OpenID provider (Identity Provider or IdP) which is the openId authentication server. It is contacted by the RP to obtain the validation of the identity of the user.
- The client who can be the web browser of the user.

3.1.13.2 Creation of an account

Before anything, the user needs to create an OpenID account. First, he needs to choose an OpenID identity provider such as VeriSign PIP. Then, he creates a username and password, and he can choose to add some usefull information about him. Finally, he has an identified URI on his IdP.
The algorithm should use minimal resources to implement in hardware by minimizing requirements on battery, number of buttons, and size of LCD display.

3.1.13.3 OpenID authentication flow

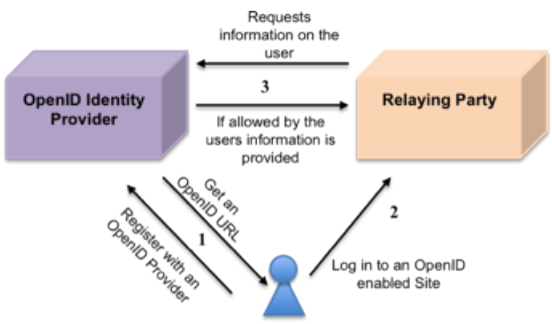

Figure 13: OpenID authentication flow

With :

1. The user creates his account on the chosen OpenId identity provider and retrieves his OpenID URI.
2. The user connects to the wanted OpenID site with his OpenID id.
3. The relying part needs to validate the identity of the user: it asks the OpenID IdP if the user is known. If the user allows the OpenId IdP to share his information, this one sends the information to the RP.

3.2 Known lacks of these systems

3.2.1 Password breach

In most of the systems explained before, the system stores the user password in databases which can be hacked. In the next paragraph we will expose a case of this type of issues.

This example is extracted from an article "Yahoo's password leak: What you need to know (FAQ)" [15].

> *"Yahoo has just become the latest big online service to suffer a major password breach. [...] the attack is a big black eye for Yahoo and a potential hazard to the 450,000 or so people whose log-in information is now flapping in the breeze.[15]"*

3.2.1.1 What went wrong ?

As David Hamilton explains -[15]-, a hacker published more than 450.000 login credentials (usernames and passwords) obtained from Yahoo's "Contributor Network" website. The hacker revealed that he could hack the website thanks to a SQL breach: an "union-based SQL injection", which is indeed a way to trick the database on a poorly secured website by divulging private information.

Thanks to the SQL injection, the hacker was able to retrieve a lot of usernames and passwords. But usually, the passwords are encrypted before storage. Unfortunately, here it was not the case: the passwords had been stored in plain text. This even shows a significant security failure from Yahoo's part.

After the publication of the file on a public website, this one has been widely distributed all over the Internet revealing thus the users's credentials. Moreover, this hack did not just affect poeple with Yahoo accounts. Indeed, the "Contributo Networ" allows users to sign in with their Gmail, Hotmail or other emails addresses through Facebook.

> *"It's estimated that fewer than half the total compromised accounts were Yahoo ones. Yahoo says it's changing the passwords of people on its own network, and it's "notifying the companies whose users accounts may have been compromised".[16]"*

3.2.2 Principle of least privilege breach

Sometimes with these systems, a lack in the principle of least privilege appears. Before going further, I will explain this principle.

3.2.2.1 Principle of least privilege

Definition of the principle of last privilege:

> *"In information security, computer science, and other fields, the principle of least privilege (also known as the principle of minimal privilege or the principle of least authority) requires that in a particular abstraction layer of a computing environment, every module (such as a process, a user or a program depending on the subject) must be able to access only the information and resources that are necessary for its legitimate purpose."*[17]

This principle is very important and means that any user or application should have only access to the resources they need and no other.

This principle of least privilege is widely important in enhancing the protection of user data. Indeed, an application should only access user information required for his good behaviour. For example, a backup user has no need to install software: he only needs to run the backup and backup-related applications. Any other privilege, such as installing new software are blocked. [17].

This principle leads to some of the benefits below:

- **Better system stability:** If the code is limited within the system, it is easier to test all of his actions and interactions with other applications.

- **Better system security:** If the code is limited in the system-wide actions it can perform, it reduces vulnerabilities. Indeed, a leak from the part of of the application can not be used on the rest of the applicaiton.

3.2.2.2 Example 1: Skype

This example may not be one of the latest but it symbolizes the importance of the principle of least privilege.

By the past, the user was able to connect to Skype with either a username and a password or Outlook credentials.
If the user decided to use his Outlook credentials, he gave Skype an access to his contact list and to everyl other information from his Outlook account. How could we be sure that Skype would not use the information badly? Even if the concept of retrieving more contacts on Skype via the Outlook contact list is tempting, it can be dangerous for your private information.

This example presents the implementation of the principle of least privilege as being important for the user.

3.2.2.3 Example 2: AppNexus

This example is extracted from the article *"Security Basics Part 1: Principle of Least Privilege."*[18]

> *"When users authenticate to our API, they get back a token that can either be stored in a cookie or passed in the header of subsequent requests so we know who they are and what they are allowed to do. When a user logs into Console, Console keeps track of this token and uses it to make calls for the user. That gives the AppNexus Console a lot of power; any call a user can make can be made for the user by Console. That makes sense for our own web application. If a client trusts our API, they may as well trust Console too."* [18]

At that point, the author wonders if it is a good idea to give so much power to a third part. As he explains:

> *"An app sits inside the AppNexus Console and makes calls to the API just like Console does. But should apps — which can be made by random third parties — be allowed to make any call to any service for the user like the AppNexus Console can?"* [18]

As far as the author is concerned, it can be dangerous to allow application to make calls to one or several services API provides. The need to restraint the information access is obvious. This principle is so fundamental that it is often forgotten.

On the one hand, the author decides to give the application a set of permissions. On the other hand, he warns the user that some applications will need access to his information while installing the AppNexus.

As Jon Avery writes, using the principle of least privilege :

> *"gives the platform the best combination of safety and usefulness, which in turn gives our clients the best combination of peace of mind and efficiency."* [18]

4 The most common approaches: tokens and one-time passwords

This section is about tokenization and one-time passwords. For both approaches, we will give their interests, their architecture, their functioning, their security and their points of vulnerability. They are not the only existing approaches but their are the main used in the authentication procedures.

4.1 What is tokenization?

> *"Tokenization is the process of replacing sensitive data with unique identification symbols that retain all the essential information about the data without compromising its security. Tokenization, which seeks to minimize the amount of data a business needs to keep on hand, has become a popular way for small and mid-sized businesses to bolster the security of credit card and e-commerce transactions while minimizing the cost and complexity of compliance with industry standards and government regulations."*[19]

The huge interest of tokenization relays in the fact that the token is not a primary information. Indeed, it cannot be used outside the context of a specific unique transaction. In the banking field, during a transaction with a credit card, the token typically contains only the last four digits of the card number. Concerning the rest of the token, it consists of alphanumeric characters that represent cardholder information and data specific to the transaction underway.

The tokenization process makes it more difficult for hackers to have access to the cardholder data. Indeed, in the older systems, the credit card numbers were stored in databases and exchanged freely all over the networks. Tokenization technology can, in theory, be used with sensitive data of all kinds including banking transactions, medical records, criminal records, vehicle driver information, loan applications, stock trading and voter registration.[19]

The explanations given in the following subsections are extracted from the paper *"Understanding and Selecting a Tokenization Solution"*[20].

4.1.1 What is the interests in tokenization?

The most common use is for tokenization to protect sensitive data. But the choice of using tokenization can be done for several different reasons:

- **Reduced compliance scope and costs:** As we know, tokenization replaces sensitive values with random values. Systems that use the token instead of the real value are mostly exempt from audits and assessments required by regulations for systems which access the original sensitive data.
- **Reduction of application changes:** Tokenization is used to protect sensitive values within legacy application environments where we might otherwise use encryption. Tokenization allows us to protect the sensitive value with an analogue using the exact same format, which can minimize application changes. Many of these changes might be avoided with tokenization, so long as the token formats and sizes match the original data. Tokens do not need to preserve formats and data types of the original values, but this capability is a key feature to reduce deployment costs. In our context, using a token can avoid the transmition of username and password on the network.
- **Reduction of data exposure:** The main asset of tokenization is that it requires data consolidation. Sensitive values are only stored on the tokenization server(s); on this server they are encrypted and highly protected. In encrypting systems, sensitive data tends to appear in many locations, increasing exposure risks.
- **Masking by default:** We know that value is random, so we can say that it acts as a mask for the data. The worry around masking sensitive data is less important, as the real value is never exposed or used in the application. Fully tokenized solutions provide greater security and less opportunity for data leakage or reverse engineering unlike classic masking.
- **Better performance** compared to encryption methods.
- **Reduction of the cost** is the main reason of using tokenization.

Tokenization seems to present good advantages in our goal to authenticate user without communicating his password or his username. Moreover, the OAuth protocol - explained in section 6 - uses tokens, so it is interessant for us to have a better understanding of this method.

4.1.2 How does it works?

In theory, tokenization is mostly simple: it involves substituting a marker of limited value for something of greater value. The token has limited value since outside the application environment or a subset of that environment it is useless. Useful data -such as the password- are taken and converted to a local token that is useless outside the application environment designed to accept it. If the token is stolen or if the token is inappropriately generated within the context environment thanks to the exploitation of some vulnerability, it can only de used it into this same environment. This significantly reduces risks , as well as the scope of compliance requirements for the sensitive data.

The architectural model is built by following some steps, as shown in figure 14:

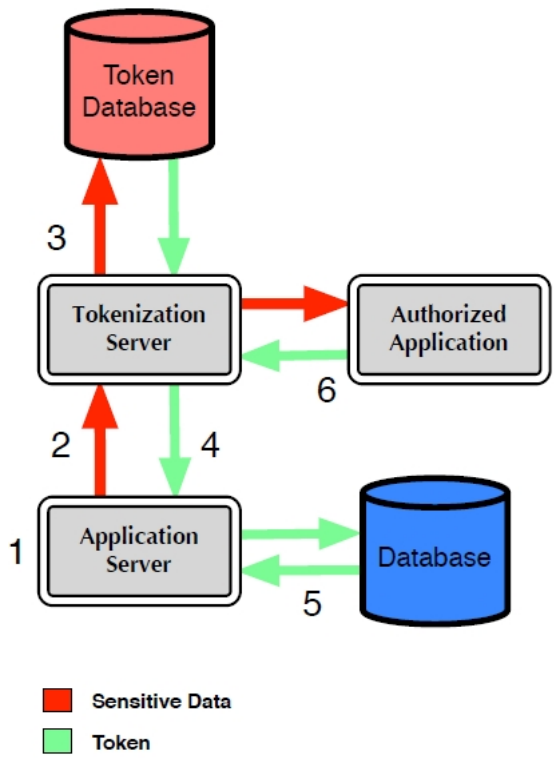

Figure 14: Basic Tokenization Architecture

1. The application collects or generates a piece of sensitive data.
2. The data is immediately sent to the tokenization server: the data is not stored locally.
3. The tokenization server generates the random or semi-random token. After, the sensitive value and the associated token are stored in a highly secure and restricted

database (usually encrypted).

4. The tokenization server returns the token to the application.

5. The application stores the token instead of the original value. This token will be used for most transactions with the application.

6. If the sensitive data is needed, an authorized application or user can request it. The sensitive data will never be stored in any local database, and in most cases access is highly restricted. This significantly limits potential exposure.

For this to work you must ensure a few things:

1. The original data cannot be reproduced without the tokenization server. This is different than encryption, where you can use a key and the encryption algorithm to recover the value anywhere.

2. All communications must be encrypted.

3. The application will never store the sensitive value: only the token.

4. Ideally, the application never even reaches the original value.

5. The tokenization server and database are highly secure.

4.1.3 Token creation

There are three main methods to create tokens, but only one is recommended:

- **Random Number Generation:** This method replaces data with a random number or alphanumeric value, and is the better method. Fully random tokens ensure the greatest security, as the content cannot be reverse engineered. This one is highly recommended.

- **Encryption:** This method generates a token by encrypting the data. Sensitive value is padded with a random salt to prevent reverse engineering, and then encrypted with the token server private key. The main advantage is that the token is reasonably secured from reverse engineering, but the original value can be retrieved if needed. Some of drawbacks are poor peformances and the risk of exposing the data if the server keys is compromised.

- **One-way Hash Function:** Tokens are created by running the sensitive value through an irreversible mathematical operation (the hash function). This offers reasonable performance but tokens are not reversible if the value is needed. Moreover, security is not as strong as fully random tokens.

4.1.4 Token datastore

Tokens - along their associated value - are stored within heavily secured databases with extremely limited access. The data is typically encrypted to ensure valuable data will not be lost in the event of a database compromise or stolen media. The token (database) server is the only point of contact with any transaction system, payment system, or collection point to reduce risk and compliance scope. The acess to the database server is extremely restricted, with administrative personnel denied read access to the data, and even authorized access to the original data limited to carrefully controlled circumstances.

Token are used to represent the same data for multiple events, possibly across multiple systems, so most token servers can issue different tokens for the same user data. For example, a credit card number may have a different unique token for each transaction. The token server not only generates a new token to represent the new transaction, but is responsible for storing many tokens per user ID. Many use cases require that the token database support multiple tokens for each piece of original data, a one-to-many relationship in the database. This provides better privacy and isolation, if the application does not need to correlate transactions by card number.

4.1.5 Token storage in applications

When the application receives the token from the token server, the application needs to store the token safely, and effectively erase the original sensitive data. This step is critical, not only to secure sensitive data, but also to maintain transactional consistency. Ideally, the application must never store the sensitive data, but pass it trough memory on its way to the tokenization server. An interesting side effect of preventing reverse engineering is that a token by itself is meaningless - it only has value in relation to some other information. The token server has the ability to map the token back to the original sensitive information, but it is the only place this can be done. Supporting applications need to associate the token with something like a username, the transaction Id, the merchant customer Id or some other identifier. This means applications that using token services must be resilient to communications failures, and the token server must offer synchronization services for data recovery.

The only moment weaknesses can occur is when the original data is collected or requested from the token server database: they might be exposed in memory, logs, or virtual memory. This is the most sensitive part of the architecture.

4.1.6 Authentication

Authentication at the center of the security of token servers, which need to authenticate connected applications as well as specific users. To rebuff potential attacks, token servers need to perform bidirectional authentication of all applications prior to servicing requests. The first step in the process is to set up a manually authenticated SSl/TLS session, and verify that the connection was started with a trusted certificate from an approved application. Any strong authentication should be sufficient, and some implementations may layer additional requirements on top.

The second step of the authentication process is to validate the user who is sending the request. This may be a system/application administrator using specific administrative privileges, or one of many service accounts assigned privileges to request tokens. The token server provides separation of duties through these user roles: serving requests only from approved users, through allowed applications, from authorized locations. The token server may further restrict transactions, perhaps only allowing a limited subset of database queries.

As we know, the authentication process is not so easy to set. As highlighted in the article [20], it is recommended to use SSL/TLS session started by an approved application with a trusted certificate. However, to recognize a trusted application is a challenge even when a certificate is used. In my opinion, we can never be totally sure that we complete this objective. Indeed, some of the factors used by the application to authenticate itself to the server - as the certificate, are included in its code or in its environment - and these factors can be exposed by reversing the application. While the authentication of the server is feasible as only the public interfaces are accessible and not the software files, ensuring the bidirectionnal authentication by proving the identity and the integrity of the application user-side is a real challenge.

4.2 What are One-time passwords?

The growth of new technologies and the rise of smartphones and tablets usage provides a new support for the use of one-time passwords. Indeed, in addition to the traditional physical token, it is now possible to use the mobile device to generate OTP's.

4.2.1 Types and generations

As explained in section 3.1.9, several types of OTP exist: time based, counter based and the challenge response based. This three types of OTP can be genereted in two ways. Regarding to the first one, the server generates the OTP, sends it through a specific communication means to the user and the user replies this OTP back to the server through a channel different from the one used by the server. The second approach is about the server and the device sharing a secret, like a password, a PIN or a seed token. When the authentication is needed, both generate the OTP resulted from the hash of a concatenation of this secret and depending on the type, the time or the counter. The server then compares its OTP with the one sent by the device.

4.2.2 Vulnerabilities

Nevertheless, the one-time password mechanism is useless without a well-designed process of the authentication. Multi-factor authentication scheme, including OTP authentication mechanisms are not always safe. It depends on the way the system was designed: they can have issues with attacks like SMS spoofing.

> *"The use of one-time passwords (OTP) as a second factor of authentication is growing in popularity, but some experts warn if they are not deployed smartly, they could actually leave organizations less secure than if they had not used an OTP at all."*[21]

To be more accurate, we can try to make the list of the one-time passwords vulnerabilities. According to Julian Lovelock, there is no black or white answer:

> *"Much of the concern around the security of OTP tokens stems from their underlying reliance on a symmetric key model. What that means in practical terms is that you need to load into the authentication server an exact copy of the key that's injected into the OTP token. These keys, often referred to as 'seeds', therefore need to be managed. And the processes and systems that manage those keys/seeds are great places for attackers to go after. When determining whether*

OTP tokens are secure enough, enterprises should take a look at how the keys are being managed."[22]

However, he listed six potential vulnerabilities of OTP:

- The manufacturing process that generates the seed file.
- The transport of that seed file to the customer website.
- The management of that seed file on the website, before being loaded into the authentication server.
- The secure storage of the seed file within the authentication server.
- The retention by the customer of that seed file subsequent to its being loaded into the authentication server.
- The retention of that seed file by the OTP token provider.

Furthermore, Julian Lovelock explains in [22] that if customers can initialize OTP tokens themselves from the admin console of the authentication server, the model is more secure, as it removes five of the six potential points of compromise.

Considering these vulnerabilities of OTP, here are some examples to try to steal authentication tokens:

- If the generation process is not truly random, the malicious person could try to copy it.
- A malicious person could catch the seed file during the submitting accross the network.
- A malicious person could also intercept the seed file between the generation and the storage into the authentication server.
- A malicious person could retrieve the seed file directly from the databse or from the customer device.
- A malicious person could request the token file while spoofing a customer identity.

4.3 Implementation of OTP use on Android platform

4.3.1 The two-factors authentication

The two-factors authentication is a process that enables to strengthen the security of the Google Apps connections. This is done via a validation code that users must enter in addition to their username and password. This validation cannot be sent by a third service provider but only by Google.

Before using the two-factors authentication, you need to veryfied your configuration. The prerequisite conditions to use of the two factors authentication are:

- A mobile able to receive the validation code through a SMS or a call.

or

- An Android, BlackBerry or iPhone. These smartphones use the Google Authenticator Apps, which is explained in the subsection 4.3.2, to genereting the validation code.

The main interest of using two-factors authentication is to strengthen the protection of the user account against non authorized access in case his password is lost or usurped. Even if the malicious person is able to decrypt, guess or steal the password, she cannot access the application without the validation code, which is on the phone of the user.

4.3.1.1 Set up the two-factors authentication for the domain

This activation is done in the Google administration console. To begin, you have to log in the Google administration console through `admin.google.com` with an administrator account.

After successfully login, you have to go to Security -> General settings and check the case "Allow users to turn on two factor authentication" as presented in figure 15.

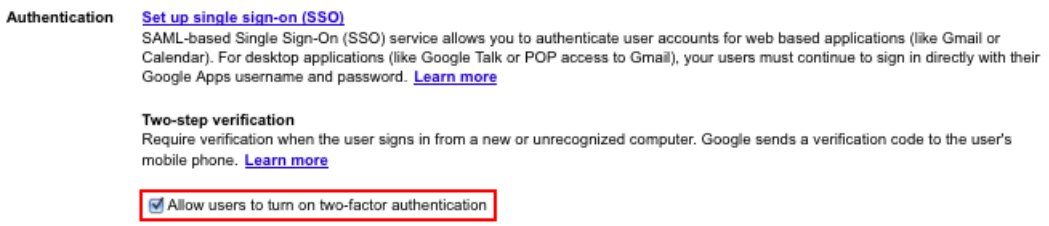

Figure 15: Enable two-factor authentication

4.3.1.2 Set up the two-factors authentication for the user

The user needs to activate the two-factors authentication of his account and choose a way to receive the validation code on his mobile: the Google Authenticator Apps, a SMS or a call. Google recommends to the smartphone users to use the Google Authenticator Apps (4.3.2) which can generate the codes without a network connection.

How to do that?

1. The user needs to sign on his Google Account settings page: figure 16
2. He has to click **Security** and edit the "2-step verification" which redirect to the "2-step verification settings page" (`https://accounts.google.com/SmsAuthConfig`).
3. Then he needs to follow the instructions.

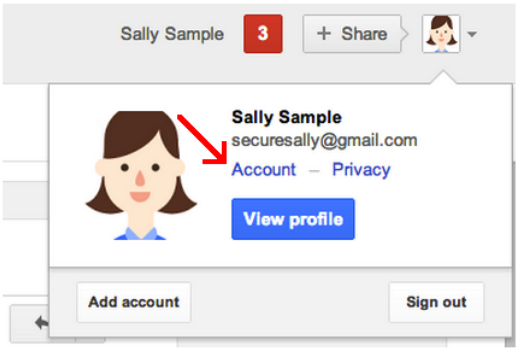

Figure 16: Sign in on Google Account

4.3.1.3 Use the two-factors authentication

If the two previous steps are successful the next time the user will connect on his Google Apps, he will enter his username and password like usual but a second page (17) will appear for him to enter his validation code.

If the user is on a public computer, he should not check the "Memorise the validation on this computer". On his own computer, he can check this option which allows him to enter the validation code only once every thirty days or when his browser's cookies are deleted.

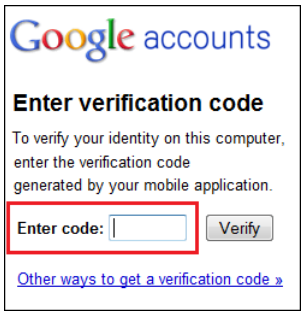

Figure 17: Enter the validation code

4.3.2 The Google Authenticator

The Authenticator App developed by Google, is a software based on two-factors authentication. This application provides a one-time password (a six digit number) that the user of Google services must provide in addition to his username and password. Moreover, the Authenticator App can be used by third-party applications to generate codes such as password managers.

A lot of websites use this application: wordpress.com, outlook.com, dropbox.com, and so on.

4.3.2.1 Technical description

To start with, the service provider needs to generate an 80-bit private key for each user. The secret key is provided as a 16 character base32 string or as a QR code. The client creates an HMAC-SHA1 using this secret key. The message that is HMAC-ed can be:

- the number of 30 second periods having elapsed since the Unix epoch.
- the counter that is incremented with each new code.

A portion of the HMAC is extracted and converted to a 6 digit code.

Pseudo code : time OTP

```
function GoogleAuthenticatorCode(string secret)

    key := base32decode(secret)
    message := current Unix time / 30
    hash := HMAC-SHA1(key, message)
    offset := last nibble of hash
```

```
    // 4 bytes starting at the offset
    truncatedHash := hash[offset..offset+3]
    // remove the most significant bit
    Set the first bit of truncatedHash to zero
    code := truncatedHash mod 1000000
    pad code with 0 until length of code is 6
    return code
```

Pseudo code : event/counter OTP

```
function GoogleAuthenticatorCode(string secret)

    key := base32decode(secret)
    message := counter encoded on 8 bytes
    hash := HMAC-SHA1(key, message)
    offset := last nibble of hash
    // 4 bytes starting at the offset
    truncatedHash := hash[offset..offset+3]
    // remove the most significant bit
    Set the first bit of truncatedHash to zero
    code := truncatedHash mod 1000000
    pad code with 0 until length of code is 6
    return code
```

5 About Single Sign-On

As explained in the previous section, tokenization and one-time password reduce the risks associated to the loss or hack of password. However, these methods do not solve the issue of needing a username and a password to access every services or applications the user needs. A few years ago, the Single Sign-On property - which allows user to connect only once and access all services or applications - appeared.

This section is centered on the Single Sign-On (SSO) concept, explaining his various interests, the architecture involved, the security concepts on wich the system which using SSO are based and the known points of vulnerabilities.

5.1 What is Single Sign-On?

Single Sign-On is a property of access control of multiple related software systems which are self-sufficient. This protocol allows a user to login once and get access to all systems without being prompted to log again to each of them.

As SSO allows different applications and ressources supporting different authentication mechanisms, it must internally translate it and store different credentials compared to what is used for initial authentication.

Once in your system the SSO property increase convenience and security. Indeed, applications or ressources that used the concept of SSO behave as if they were part of something bigger, leading to a better user experience. Morever, this property removes burden of maintaining and remembering another login and password and so minimizes the eventuality of losing one of them. The user login and password are stored in a single secured location, which is more secured than having the user creating a separate ID and password for each application or, even worse, re-using a password in multiple accounts.

A great example of the use of SSO is the Integrated Windows Authentication (IWA). This is a participating SAS server which accept users only if they have successfully authenticated to their Windows desktop. Thanks to this, IWA bypasses the initial logon and moreover, avoids the transmission of user's credentials.

5.2 How does it work?

This explanation of SSO is extracted from the *"SSO - Technique"*[23].

5.2.1 The components

On most systems using SSO, the following components can be found:

- **The client:** It can be a web browser or an application asking for application access.
- **The authentication server:** It is the central element of the system. The authentication server ensures the user authentication, the persitence of the connection and the propagation of the user identity across the applications.
- **The application server:** It ensures that before giving access to protected resources, the client asking resources is properly authenticated to the authentication server.
- **The agent:** His role is to verify that the user is correctly authenticated.

Eventually, one more component could be found:

- **The web portal:** It centralises the authorized applications access and gives an overview to the user for all applications he has access to.

5.2.2 The flow

Here are the steps of the application access in a system using SSO:

- The user needs to be registered in the directory of the authentication server.
- The authentication server must verify that the user can access the requested application.
- If booth steps are checked, the user is able to access the application.

This steps lead to some exchanges between the user and the authentication server and between the authentication server and the application server. These exchanges allow all the needed verifications to be done.

5.2.2.1 The directory side and rights management side

In this kind of system, the user needs to be registered in the access directory. Once it is done, he can access some applications.
Rights management is performed by a SSO tool which allows to manage the users rights. All the users rights are stored in a directory.

5.2.2.2 The user side

When the user wants to access the restricted application, he goes directly through or he is redirected to the authentication page where he needs to identify with his username and password.
If the authentication succeds, the user will be redirected to the application.

5.2.2.3 The application side

The application server is protected by a authentication agent:

- The agent intercepts each attempt to access the application and verifies the user identity.
- The agent verifies that the user has the permission to access the application with the authentication server.

5.2.3 Architecture of systems using SSO

Most commonly, a system using SSO has a n-tiers architecture which avoids the password transmission. Indeed, the password will be transmitted to the authentication server only on its first connection.

There are a lot of different because no architecture is whether good or bad for a system which want to do SSO. When developpers build a system like this, they need to ask themselves the right questions like the cost of developement, the cost of maintenance, the response time, and so on.

In the following paragraphs, some architectures will be explained.

5.2.3.1 The classic architecture

The following diagram illustrates the various messages existing between the components of a simple system.

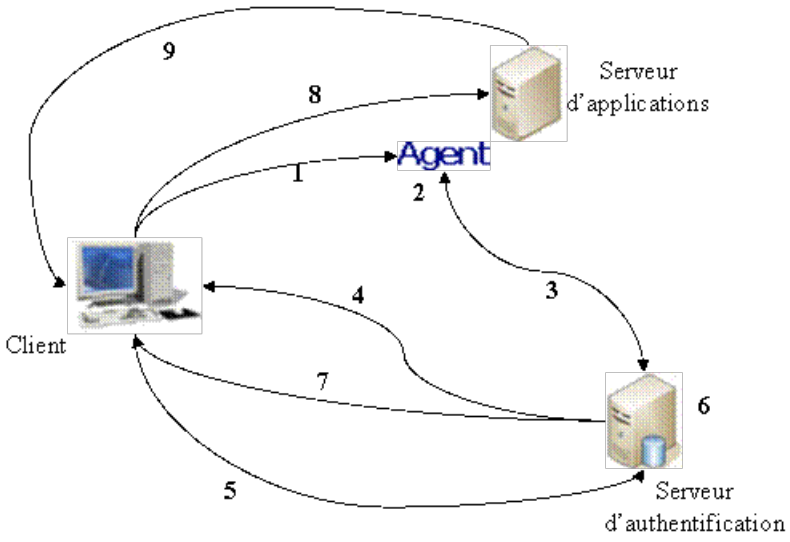

Figure 18: Simple SSO Architecture

Figure 18 requires several steps:

1. The client requests a restricted application without having been previously authenticated.
2. The request is intercepted by the agent.
3. The agent verifies with the authentication server if the user is identified and blocks the request if he is not.
4. The user needs to authenticate himself via a form sent by the authentication server.
5. The authentication server receives the user's username and password.
6. The username and password are verified with the directory. Most SSO systems have multiple ways to authenticate like passwords, certificates, and so on. Moreover, the authentication server can be connected to multiple databases.
7. If the authentication succeeds, a session token (cookie) is sent to the client.
8. Once the user is authenticated, the user session is maintained by the cookie. The browser of the user is redirected to the application, provided with identifiers. The safety data are inserted in the HTTP headers. The agent intercepts the request, it checks that the client is authenticated, then allows him to access. Once authenticated to an application, a typical application session is established.
9. The user acesses the requested application.

5.2.3.2 *Using a web portal*

The following diagram presents the various messages existing between the components of a sytem using a portal web.

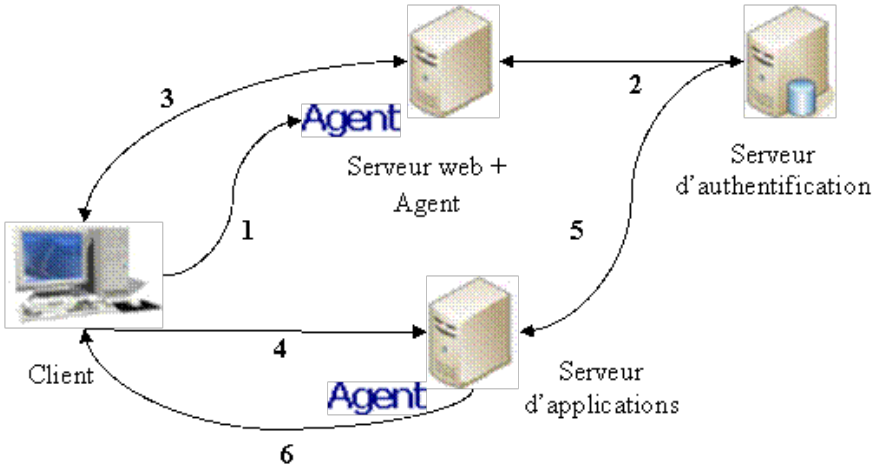

Figure 19: Portal Web Architecture

Figure 19 is made of following steps:

1. The client wants to connect to the website with his credentials.
2. The web server transmits the client authentication request to the authentication server.
3. The authenticated client receives the session cookie and a page with all the accessed applications.
4. The user can access any application on the page.
5. The agent intercepts the request and asks for the authentication of the client to the authentication server. If the authentication succeeds, the agent then passes the request to the application server.
6. The application server returns his requested application to the client.

5.3 Security

Even if the use of SSO presents many advantages for a system, there are some risks too. Strange enough, what makes this system so strong is also what makes it vulnerable: if the user username and password are stolen security is compromised. Securisation of the authentication is a key point. During the establishment of this type of system, security is definitely crutial.

5.3.1 How to make the authentication?

The user can authenticate himself with different methods such as the traditional method of username/password, a certificate, a ticket system.

- **The certificate:** The user certificate is like a numeric ID card. The certificate contains the user name and his public key. Moreover, this certificate is signed by the certificate authority (AC), a recognized organism. This signature certifies that the certificat's data were not modified. The verification can be done by decrypting the signature with the AC's public key.
- **The ticket:** It can be unique or not, unlimited or limited. It allows the management of the user's session. This system ensures more security and prevents usernames and passwords to be stolen. Tickets do not transport any critical information.

5.3.2 Transport of the authentication information

Usually, the information go through the HTTP or HTTPS protocol. The communication takes place via three channels: the POST and GET ports, the session cookies or the certificate.

To strengthen security, it is highly recommended to use HTTPS instead of HTTP. Indeed, the HTTP protocol has some known weaknesses like the interception of the requests. Evenry transfer method is secured with HTTPS who uses certificates.

5.4 Establishment of SSO

Even if the establishment of SSO is not so complex, some security aspects are expected.

The easy part is the authentication server implementation. The tough part lies in the addition of application, which are not a one time issue but a long time issue. Before going

further any application, it is needed to perform tests and to look into the users management.

The following steps needing to be checked:

- It is necessary to set up a centralised users directory (LDAP).
- The authentication server must be able to support a huge charge. Setting up duplicated server might reduce this weight.
- It is also needed to set up all the SSO softwares:
  - Depending on the chosen architecture, set up the authentication agent.
  - Deploying the authentication server which handles the users authentication.
  - Eventually, developping a web portail.

The integration of application into a system using SSO requires several stages:

- It is needed to make an analysis of the actuel authentication system. After that, we have to developp an agent into the application if needed.
- We have to integrate the application on a test server.
- We have to put the application in production.
- We need to manage the users authorizations (into the LDAP).

5.5 Advantages and limits

Like all processes or properties, SSO presents advantages and weaknesses at the same time.

5.5.1 Benefits

In this section, several benefits of using SSO will be explained. Since it is impossible to quote them all, I picked the most important.

- **Saving time and effort:** Even if logging may not seem to take a lot of time and effort, it can be a waste of time by the end of the day.
- **Fewer password to remember:** A lot of people have hard time remembering their password, so they write them down on paper, or often ask for a new password. This definitely increases the risks linked to the passwords transmition. Using SSO allows the user to have only one password to remember.

- **Reducing phishing:** Some types of phishing present users with a web page asking them to enter their credentials in order to verifiy them. If users are used to face a single screen when entering their credentials (as with Single Sign-On system), it may be easier for them to identify phishing attacks.

- **Speeding up development:** Developing the authentication system of application is not as easy it may seem. Authentication systems take time to implement, so using a SSO system reduces the amount of work.

- **Easier to secured:** In authentication systems security is essential. The systems need to be sure that password are transmitted and stored in a secure way. If there are ten way to log in, each of them needs to be perfectly secure. SSO allows to focus on making one spot absolutely secure.

- **Reducing costs:** SSO systems reduces servers costs due to a lower number of requests for authentication.

5.5.2 Drawbacks

In this section, the most important drawbacks of using SSO will be explained.

- **Using a single password:** The biggest benefit of SSO is also his biggest disadvantage: since the user accesses his services with a single password, if this one is lost or stolen, all the services are compromised.

- **The server:** If the SSO system in place uses a single server, it can introduce a single point of failure in network.

6 A way to implement SSO : OAuth

The Single Sign-On is a property, not a protocol. So you can use different protocols to implement an SSO environment. For example, Facebook uses a protocol called OAuth.

This section is centered on the OAuth protocol, explaining his various interests, the architecture involved, the creation process, the security concepts on which the system is based and the known points of vulnerabilities.

6.1 Main interests around OAuth

Now used by many popular services like Twitter or Facebook, OAuth is an authorization protocol which is now at his version 2.0. There are some pretty significant differences between the version 1.0 and the newer, the 2.0.

Why is everybody so interested about OAuth? Here are some of the reasons.

OAuth wants to be a simple and secure solution for the third-party mechanisms protocols. One part - the client - is the application that wants to use data or authenticate a user. The second part - the server - stores the data. The third part - the user - allows the application to use this data or not.
Indeed, various web services are increasingly interconnected through API, and it is not uncommon that I want to give permission to the X application to use some of my data from the application Y.
I did not have a lot of solutions before OAuth to achieve this. I usually had to deliver my login and password to X. The main interest of OAuth relies there: I do not have to communicate my credentials with a lot of application anymore.

6.2 OAuth 1.0

6.2.1 Why was OAuth 1.0 introduced?

With the huge expansion of the Internet, more and more websites appear, they provide many web services like social networks, website to book tickets, and so on. The problem with all these application, is that they need to access user data from other website so the user is obliged to communicate his username and password. This manipulation may compromise the efficience of the password, above all if the user uses the same password

for more than one application.

The goal of OAuth is to provide methods for users to grant third-part to access to their ressources without sharing their usernames and passwords. The main advantage for users is that are fully in control of what they allow applications to do or not to do, and they do not have to share their passwords with third-part applications.

The OAuth community, to a large extend, emerged from the OpenID community. This later was also looking for a better authentication method for the Twitter API, one that did not require Twitter users to share their usernames and passwords with third-part applications.

> *"We want something like Flickr Auth / Google AuthSub / Yahoo! BBAuth, but published as an open standard, with common server and client libraries, etc. The trick with OpenID is that the users no longer have passwords, so you can't use basic auth for API calls without requiring passwords (defeating one of the main points of OpenID) or giving cut-and-paste tokens (which suck). – Blaine Cook, April 5th, 2007"* [24]

In July 2007, a new working group was created around the problem of authentication.

6.2.2 How does it works?

OAuth is a protocol with three defined roles: the client, the server and the resources owner. These three roles are part of any use of the OAuth protocol. Sometimes, the client may be the resource owner.

For the comming explanations, we may use the term customer instead of the client, service provider instead of the server and user instead of the resource owner.

We know that in the traditional way, when we authenticate ourselves in a client server model, the client uses his own credentials to acces his resources on the server. In this case, the server do not care wether he is communicating with the client or if the client is acting on the behalf of someone else. All that matters for the server is that the shared secret matches the server's expectation.

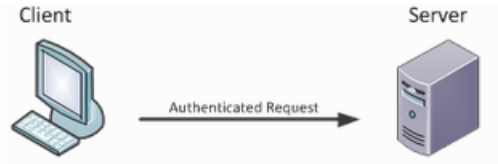

Figure 20: Authentication in traditionnal way

It is common for the client not to be acting for himself. What should we do then? The client can use the resources owner's credentials to make the request himself.

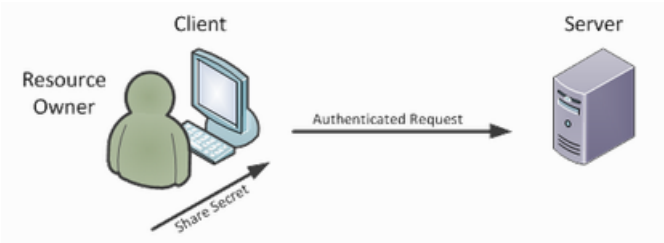

Figure 21: Authentication on the behalf of someone

Both models are simple. Whenever the client is a web application, the client can split into two components: a front-end and a back-end. The first one is running whithin a web browser on the client's desktop, while the other one is running on the client's server.

The request is executed in two parts. The resource owner interacts with the front-end of the client application while the server interacts with the back-end of this one.

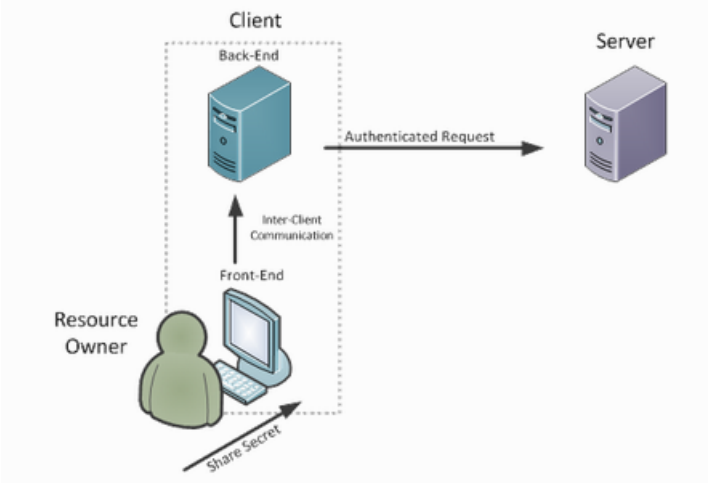

Figure 22: Authentication with a web application

6.2.2.1 Credentials and Tokens

In the OAuth protocol we can differentiate three kinds of credentials: client credentials, temporary credentials, and token credentials.

To authenticate the client, OAuth uses the client credentials. This allows the server to do some services for the client like collecting information, doing a special treatment for the client, and so on. The clients credentials can not be trusted blindly, in some case they are just used for informational purposes. Token credentials are used instead of the resource owner's username and password. Instead of having the resource owner sharing its credentials with the client, it authorizes the server to issue a special class of credentials to the client which represents the access grant given to the client by the resource owner.

Token credentials are divided into two parts: a token identifier which is usually a unique random string hard to guess and a secret to protect the token from being used by unauthorized parts. The token credentials are often limited in time and scope, they can also be revoked any time by the resource owner.

OAuth may use temporary credentials which allow to identify the authaurization request.

6.2.2.2 Specification

The OAuth specification is devided as below.

The first part defines a redirection-based browser process for end-users to authorize client access to their resources. This is made possible by having the users authenticate directly with the server, instructing the server to provide tokens to the client to use using the authentication method.

The second part defines a method to make authenticated HTTP requests using two sets of credentials, one identifying the client making the request, and the other identifying the resource owner on whose behalf the request is being made.

6.2.3 Authenticating with OAuth 1.0

The OAuth process for authentication allows users to grant access to their protected resources without sharing their credentials. In order to do this, OAuth need tokens which are generated by the service provider. There are two types of tokens (for a better under-

standing of tokens, refer to section 4.1):

- **The resquest token:** It is used by the consumer to ask the user to authorize access to the protected resources. After, the user authorized request token is exchanged for an access token which can be used only one time and which can not be used for any other purpose. For more security in the process, it is recommended for the request token to be limited in time.
- **The access token:** It is used by the consumer to access the protected resources on the behalf of the user. With this token, OAuth can limit the access to a part of the protected resources. Like the resquest token, the access token can have a limited lifetime. For more security, the service providers should allow the user to revoke the access token.

The OAuth flow can be described in three seperated steps:

1. The client need to obtain an unauthorized request token.
2. The service provider obtains the user authorization.
3. The client obtains an access token.

Now, we will explain the three steps in details.

6.2.3.1 Obtaining an unauthorized request token

The first step consists in the consumer asking the service provider to obtain an unauthorized token. This token can only be used to obtain an access token. This flow is devided into the following steps.

The consumer sends an HTTP resquest to the service provider. This request is specified in the documentation of the service provider. Generally, OAuth recommands an HTTP Post request. This request needs to be signed and contains some parameters like the oauth_consumer_key, oauth_signature_method, oauth_signature, oauth_timestamp, oauth_nonce, oauth_version. When this request is sent, the service provider checks the signature and the consumer key. If there is a match, the service provider generates a request token and a secret token. These tokens are sent to the consumer in the HTTP response body. The response contains the oauth_token and the oauth_token_secret. If the request falls for some reason, the service provider should respond to the consumer with an appropriate response.

6.2.3.2 Obtaining the user authorization

When the consumer gets the request token, he still needs to wait for an autorization from the user to use the token. Obtaining the user authorization includes three steps.

To get this authorization, the consumer needs to do an HTTP Get request to the service provider with some parameters : aouth_token which is obtain in the previous step, the oauth_callback. When he receives this request, the service provider checks the user identity and asks for his consent. OAuth does not specify how the service provider authenticates the user, it only defines a set of requeried steps. After, the user authenticates with the service provider and grants permission for consumer access, the consumer must be notified that the request token has been authorized and is ready to be exchanged for an access token.

6.2.3.3 Obtaining an access token

The consumer needs to exchange his request token with an access token which is capable of accessing the protected resources. Two steps have to be followed.
With an HTTP request to the service provider, the consumer is asking for an access token in exchange of his resquest token. The request must be signed and contains the following parameters: oauth_consumer_key, oauth_token, oauth_signature_method, oauth_signature, oaut_timestamp, oaut_nonce, oauth_version. When the service provider receives this request, he must ensure that the request signature is correct, that the request token has never been used and that the resquest token matches the consumer key. If this three verifications are successfull, the service provider generates an access token and a secret token. These two tokens return in an HTTP response. The access token and secret token are stored by the consumer.

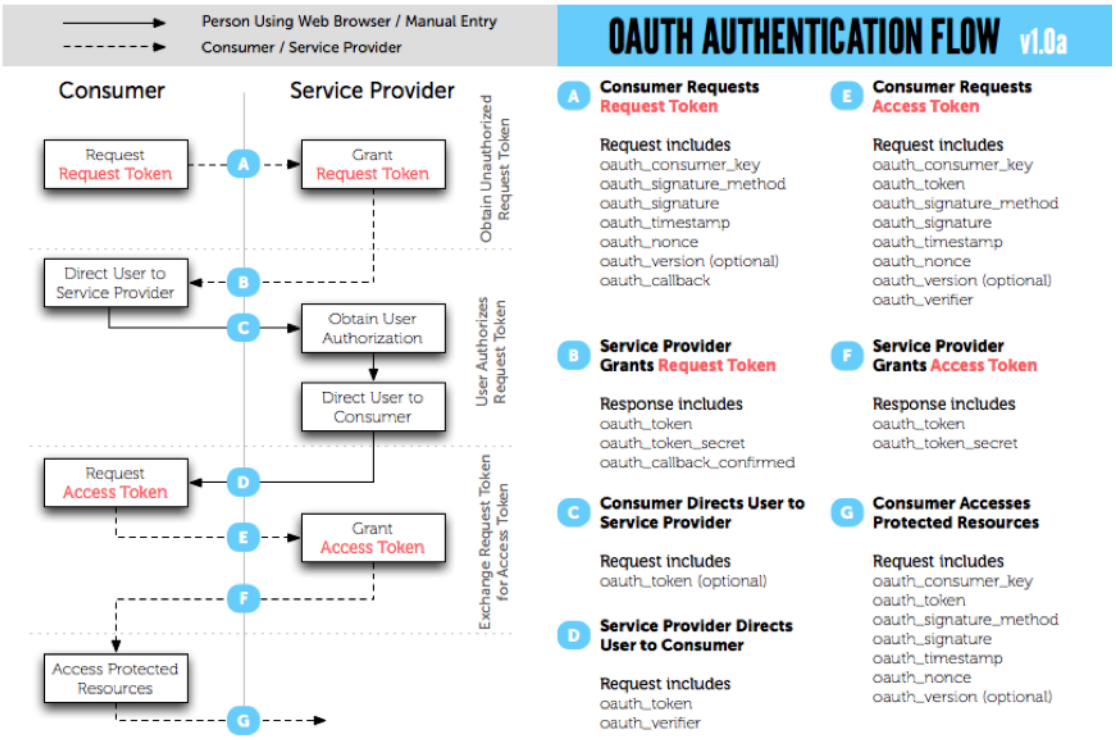

Figure 23: Authentication flow with OAuth 1.0

6.2.4 Advantages, limits and weaknesses of OAuth 1.0

OAuth is a popular mechanism which allows users to share protected resources without disclosing their username and password. In this part we will discuss the pros and cons of using OAuth for the authentication. We will begin with the advantages on the end user side and then the advantages on the organization side. After, we will discuss the disadvantages.

6.2.4.1 Advantages

From the end user point of view, login with OAuth presents some advantages as not having to create another profile, not having to create and remind a new password. In this way, users do not submit their username and password if they are not completely sure of they can trust the application. OAuth also allows the user to activate extra functionalities and synergies when taking advantage of the social graph and other data and features made available by the OAuth provider. This may provide a lot of added value.

From the organization point of view, login with OAuth presents some advantages too. Using OAuth can save time during the development. Indeed, the application no longer needs to

support passwords renewals, forgotten passwords, and the process of authentication, and so on. If the OAuth standard is extended with support of info cards or other functionality in the future, it might easily be supported in your application. At the same time, the organization has less data to store on its servers.

6.2.4.2 Drawbacks

Now, from the end user point of view, we can also find some disadvantages. Users can not tailor the profile of the application. It can be a bit confusing for the user having to create an account with OAuth providers if he/she does not have an account there already.

From the organization point of view, if the remote service is down you have no login. It requires some logic if you want to allow the user to log in with multiple OAuth providers. Maybe the end-user do not want to share his OAuth provider data with your application.

6.2.4.3 Weaknesses

The working group around OAuth has detected some weaknesses in some areas like the authentication and signature, performance at scale.

Concerning the authentication and signatures, the main problem occurs in the use of the cryptographic. In fact, the convenient and ease offered by simply using passwords is sorely missing in OAuth. For example, some developpers mainly writte little scripts on Twitter or Facebook by using their credentials. But with the use of OAuth, they can not do it anymore, they need to install and configure many librairies in order to accomplish what they could do in a single click before.

The group has also found issues with the users experience. We explained earlier that the OAuth process includes two parts: obtaining a request token and using it to have the access token. After a while, the developpers realized that this OAuth flow offered very limited and very basic experiences to users.

Developpers also dwelled on the fact that the increase use of OAuth, causes the poor scale of OAuth. A lot of management is required to implement on OAuth authentication. For example, OAuth 1.0 requires that the protected resources endpoints has access to the client credentials in order to validate the request.

6.3 OAuth 2.0

6.3.1 Why the need of OAuth 2.0?

OAuth 2.0 is a completely new protocol and is not backwards compatible with previous versions. However, it retains the overall architecture and approach established by the previous versions.

> *"Many luxury cars come with a valet key. It is a special key you give the parking attendant and unlike your regular key, will only allow the car to be driven a short distance while blocking access to the trunk and the onboard cell phone. Regardless of the restrictions the valet key imposes, the idea is very clever. You give someone limited access to your car with a special key, while using another key to unlock everything else. [...]*
>
> *The problem is, in order for these applications to access user data on other sites, they ask for usernames and passwords. Not only does this require exposing user passwords to someone else - often the same passwords used for online banking and other sites it also provide these application unlimited access to do as they wish. They can do anything, including changing the passwords and lock users out.*
>
> *OAuth provides a method for users to grant third-party access to their resources without sharing their passwords. It also provides a way to grant limited access."*[24]

The protocol OAuth 1.0 was principally based on two proprietary protocols: Flickr's API Auth and Google's AuthSub. After three years, the OAuth 1.0 was the best solution. But it was not enough.

After three years of work, the community has the benefit of hindsight and decides to improve the protocol in three areas where OAuth 1.0 proved some limits. These three areas are: the authentication and signatures, the user experience and the performance at scale. OAuth 2.0 will try to fix this issues.

6.3.2 Protocol flow

OAuth 2.0 defines four roles: the resource owner, the resource server, the client and the authorization server. In some cases, the authorization server is the same that the resource

server.

As explained before, the authentication with OAuth - 1.0 or 2.0 - involves the use of token. With OAuth 1.0 we discovered two types of token - the request token and the access token - with OAuth 2.0 one token disappears and one appears. OAuth 2.0 use these two types of tokens:

- **The access token:** Like with OAuth 1.0, the access token is used to access the protected resources. These tokens represent specific scopes and durations of access, granted by theresource owner. Access tokens can have different formats, structures, and methods of utilisation based on the resource server security requirements.
- **The refresh token:** Refresh tokens are credentials used to obtain access tokens, they replace the request token from OAuth 1.0. Refresh tokens are issued to the client by the authorization server and are used to obtain a new access token when the current access token becomes invalid or expires. Unlike access tokens, refresh tokens are intended for use only with authorization servers and are never sent to resource servers.

In the next part of this paragraph, we will explain the interactions between the four stakeholders. This explanation is inspired by the paper *"The OAuth 2.0 Authorization Framework"*[25]

1. The authorization request
2. The authorization grant by the resource owner
3. The authorization grant by the client
4. The access token by the resource owner
5. The access token by the client
6. The access to the protected resource

Figure 24 represents the authentication flow with OAuth 2.0. In the next subsections, we will explain the different steps of this authentication flow.

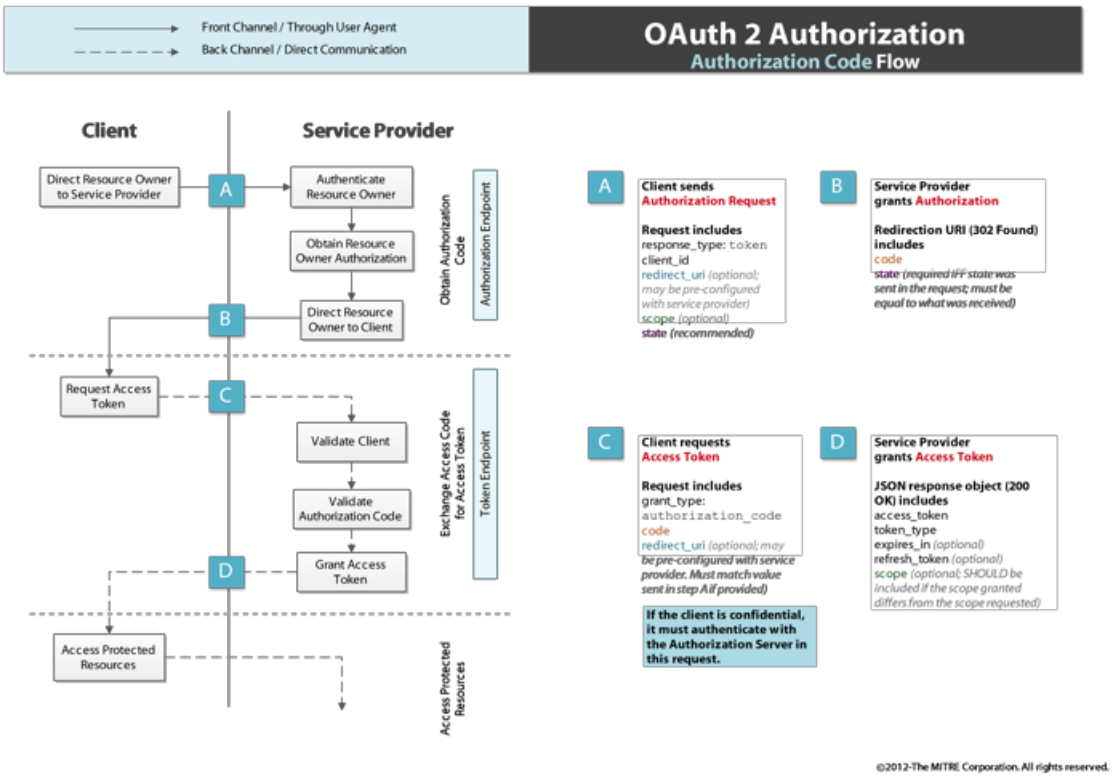

Figure 24: Authentication flow with OAuth 2.0

6.3.2.1 The authorization request

The first time, the client needs to ask authorization from the resource owner. This request can be done in two ways: either directly to the ressource owner or via the autorization server as an intermediary.

6.3.2.2 The authorization grant by the resource owner

The client receives an authorization grant, which is a credential representing the resource owner's authorization. The authorization grant type depends of the method used by the client to request authorization and the types supported by the authorization server.

6.3.2.3 The authorization grant by the client

The client requests an access token by authenticating with the authorization server and presenting the authorization grant.

6.3.2.4 The access token by the resource owner

The authorization server authenticates the client and validates the authorization grant, and if valid, issues an access token.

6.3.2.5 The access token by the client

The client requests the protected resource from the resource server and authenticates by presenting the access token.

6.3.2.6 The access to the protected resource

The resource server validates the access token, and if valid, serves the request.

6.3.3 Types of authorization grant

What is the authorization grant? It is the credential which represent the resource owner's authorization to access the protected resource. This credential is used by the client to obtain his access token. OAuth 2.0 specifies four types which is explained in the specifications.

- **Authorization code:** If we use an intermediary between the client and the resource owner, you receives an authorization code. Instead of direct request directly the resource owner, the client go through an authorization server which redirects the resource owner directly back to the client with the authorization code. This way, the resource owner's credentials are never shared with the client. Indeed, the authentication of the resource owner is done before the redirection to the client. This method provides important security features such as the ability to transmit directly the access token to the client without passing it through the resource owner's user-agent. This method decreases the exposition of the token. [25]

- **Implicit:** The implicit great is a particular case of the authorization code. In the implicit flow, instead of issuing the client an authorization code, the client is issued an access token directly. The grant type is implicit, as no intermediate credentials are issued. When issuing an access token during the implicit grant flow, the authorization server does not authenticate the client. Implicit grants improve the responsiveness and efficiency of some clients since it reduces the number of round trips required to obtain an access token but it is not recommanded to use it. [25]

- **Resource owner password credentials:** With OAuth 2.0, we can use the client

credentials, namely the username and the password, as an authorization grant to obtain an access token. This method must be used only if a high level of trust exist between the resource owner and the client or if other authorization grant are not available. Even though this method requires direct client access to the resource owner credentials, the resource owner credentials are used for a single request and are exchanged for an access token.This method does not eliminate the need of the client to store the client credentials. [25]

- **Client credentials:** The client credentials, whether his credentials or other thing, can be used as an authorization grant when the authorization scope is limited to the protected resources under the control of the client. Client credentials are used as an authorization grant typically when the client is acting on its own behalf (as explain before, the client can be in some case the resource owner) or is requesting access to protected resources based on an authorization previously arranged with the authorization server. [25]

6.3.4 Types of client

Every client needs to register with the authorization server at the beginning. This registration can be done by an HTML registration form for example. But this topic does not enter in the scope of my thesis. Two types of clients exist based on their ability to authenticate securely with authorization server:

- **Confidential client:** There are the clients which are able to maintain the confidentiality of their credentials or capable of secure client authentication using other means.

- **Public client:** In opposition, public client are not able to maintain the confidentiality of their credentials and incapable of secure client authentication via any other means.

6.3.5 Knwon issues of OAuth 2.0

Some issues of OAuth come from the changes made in the core architectural:

- **Unbounded tokens:** Like explain earlier, in OAuth 1.0n the client needs two credentials to access the protected resource: the token credentials and the client credentials. In OAuth 2.0, the client credetnials are no more used. This remove the link between the token and the client or instance. This fact introduces limits on the usefulness of access tokens and increase the possibility of security issues.

- **Bearer tokens:** Oauth 2.0 got rid of all signatures and cryptography at the protocol level. This means that 2.0 tokens are inherently less secure as specified.
- **Expiring tokens:** OAuth 2.0 tokens can expire and must be refreshed. This is the most significant change ffrom OAuth 1.0 as now the developpers need to implement token state management.
- **Grant types:** In OAuth 2.0, authorization grants are exchanged for access tokens. The "grant" is an abstract concept representing the end-user approval. It can be a code received after the user clicks "Approve" on an access request.

6.3.6 Is OAuth 2.0 a success?

After looking all over the Internet, I can say that OAuth 2.0 is not a huge success. Indeed, a lot of article talk about the huge disappointment which is OAuth 2.0. It is almost impossible to find an article which highlights the benefits of this new version.

Even if Oauth 2.0 is used by Google or Facebook, they made some changes to strengthen the security and other things before deploying the protocol. But even with this changes, Facebook had to deals with serious issues as explain in the article wrote by Grham Cluely: *"Facebook flaw allowed websites to steal users' personal data without consent"*.[26]

> *"I failed. We failed. - Eran Hammer"*[27]

Trough this quotation, Eran Hammer announced his resignation as leader of the OAuth 2.0 specification after three years of work. On his blog he explains that it was not an easy decision.

> *"Remove my name from a document that I somehow developed over three years, with more than twenty drafts, was not easy. Decide to throw in the towel in an effort that I drove for more than five years was dying."*[27]

According to him, OAuth 2.0 is not used because the specification failed to reach its two goals: the security and the interoperability. Moreover, some architectural changes have brougth useless complexity. For example, the token revocation in the first specification was replaced by illimited tokens.

Eran Hammer thinks that the specification of OAuth 2.0 give not enough details. According to him, it gives to much choice to the implementation which brings a lack of

interoperability.

The last advice from Eran Hammer is to stay at the 1.0 version if it works fine for you. Using the version 2.0 requires a strong security analysis before the use. Moreover, OAuth 2.0 does not fit entreprise culture either. The essential authentication design of OAuth 2.0 also clashes with the needs of enterprise level integration, making usage of a third party API authorized by OAuth 2.0 difficult if not outright impossible.

David Recordon, another founder of Oauth, also removed his name from the specifications for unspecified reasons.

6.4 Implementation of an Android application using OAuth

To have a better understanding of the OAuth flow on an application, I created a prototype which uses three methods to authenticate the user. The first one uses the classic connection with a username and a password created by the user for this specific application. The second one uses the OAuth protocol to connect the user and asks him the rights to access his google contacts. The third one uses the connection with Google+, the user cannot decide what the application can have access to: it is a class using the pre-installed libraries of Android.

The first screen of my application is common to the three methods. With this screen the user can choose which method he wants to use. Figure 25 presents it.

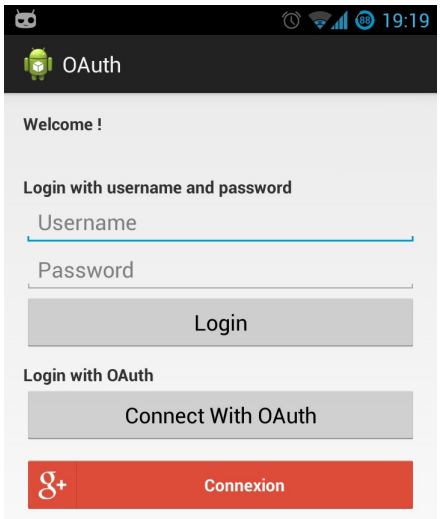

Figure 25: First screen

6.4.1 Method 1: Username and password

Method 1 is a classic method. The user needs to registrate himself to the server. After, he can access the application with his username and his password. This method is not in the interest of this section of my thesis, I want to present the benefits of the use of OAuth. So I did not implement it.

If I had chosen to do it, I would have needed to create in my application a little database to store the user credentials in my application: otherwise each time the user would have to access my application, he would have had to enter again his credentials again. After, my application have sent the credentials to my server and check if they matched credentials store in the database server. Only if the credetnials matches one in the database, the user would have been able to access my appliction. The incovenient of this method is the need to create a new set of username and password. Moreover the user cannot retreive any information of another account.

6.4.2 Method 2: OAuth

Method 2 is the most interesting. I choose to do it to explain how the OAuth 2.0 flow works into an Android application. I will explain my choices and the differents steps bellow.

To begin, I created a ClientID on Google console (`https://code.google.com/apis/console/`). Figure 26 presents my ClientID for web application which I need to implement OAuth on my application.

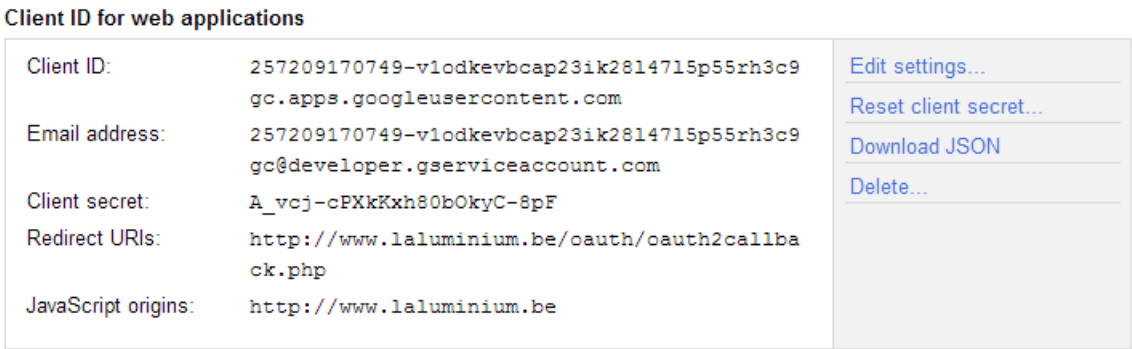

Figure 26: My ClientID for web application

My first decision was to use google contact to explain OAuth. In my application the user can choose to share his Google contacts or not.

After this, I had a choice to make. Indeed, I could put the information inside my application to retrieve the user access token but by doing so I took the risk of compromising the *"Client secret"*. I chose instead to create a server.

The server is implemented in php. To help me understand what I should do, I lookesato the google developers. Thanks to these explanations, I was able to properly implement my server. The code is bellow this paragraph.

```php
<?PHP
header('Content-type: application/json');
if ($_REQUEST['error']){
  return json_encode(array('Error' => $_REQUEST['error']));
}
else if(!isset($_REQUEST['code'])){
  return json_encode(array('Error' => 'No code provided'));
}
else{
  $url = 'https://accounts.google.com/o/oauth2/token';

  $fields = array(
      'code' => urlencode($_REQUEST['code']),
      'client_id' => urlencode("257209170749-v1odkevbcap23ik28l47l5p55rh3c9gc.apps.googleusercontent.com"),
      'client_secret' => urlencode("A_vcj-cPXkKxh80b0kyC-8pF"),
      'redirect_uri' => urlencode("http://www.laluminium.be/oauth/oauth2callback.php"),
      'grant_type' => urlencode("authorization_code"),
    );

  foreach($fields as $key=>$value) { $fields_string .= $key.'='.$value.'&'; }
  rtrim($fields_string, '&');

  $request = curl_init();

  curl_setopt($request,CURLOPT_URL, $url);
  curl_setopt($request,CURLOPT_POST, count($fields));
  curl_setopt($request,CURLOPT_POSTFIELDS, $fields_string);

  $result = curl_exec($request);
  curl_close($request);

  return json_encode($result);
}
?>
```

All the variables which I needed were given by the google developers specifications.

In the next step, I created the code I needed for my application.

The first change to do was to allow the application to access the Internet. This could be done thanks to the permission in the Android manifest.

```
<uses-permission android:name="android.permission.INTERNET" />
```

The fist time the user logs into my application, he needs to enter his google credentials, as shown in figure 27.

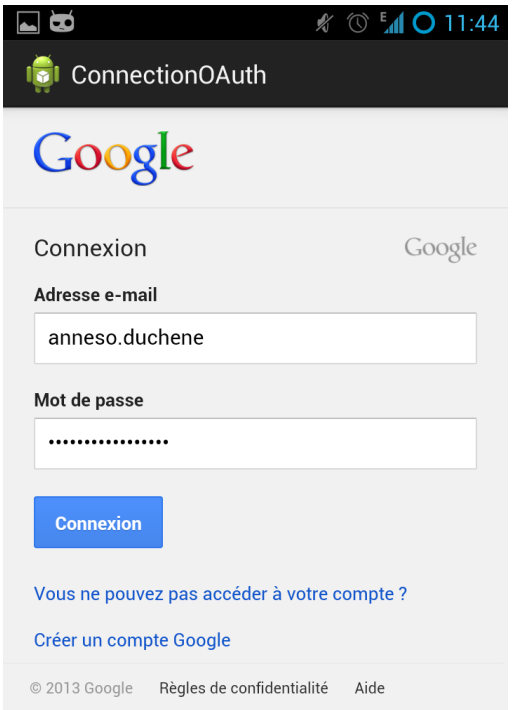

Figure 27: The user enters his credentials

After this, the user needs to accept to share his contacts with the application, as illustrated in figure 28.

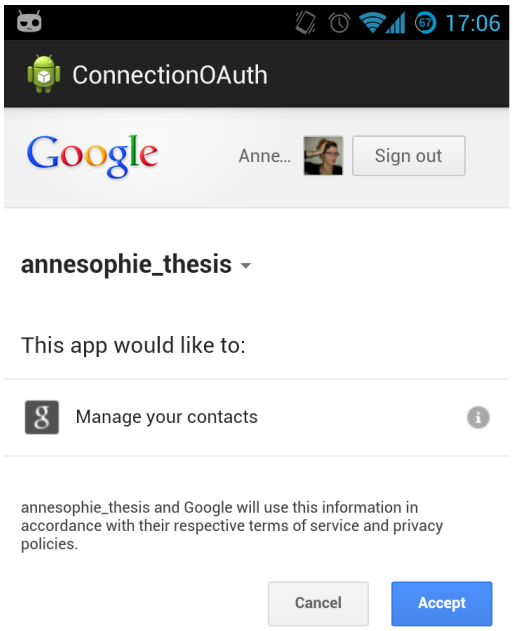

Figure 28: The user allows the application to access his contacts

If the user agrees, the application will create a request to get an access token from the server. The code bellow presents how the request works.

```
// prepare the url with all the information needed
    String response_type = "code";
    String client_id = "257209170749-v1odkevbcap23ik28l47l5p55rh3c9gc.
       apps.googleusercontent.com";
    String redirect_uri = "http://www.laluminium.be/oauth/
       oauth2callback.php";
    String scope = "https://www.google.com/m8/feeds";
    String access_type = "online";
    String approval_prompt = "auto";
    String state = "";
    String base_url = "https://accounts.google.com/o/oauth2/auth";

    String url = base_url + "?" + "response_type=" + response_type
        + "&client_id=" + client_id + "&redirect_uri=" + redirect_uri
        + "&scope=" + scope + "&access_type=" + access_type
        + "&approval_prompt=" + approval_prompt + "&state=" + state;

    // execute the url
    webview.loadUrl(url);
```

This code below show how the server response is extract of the webview.

```
webview.setWebViewClient(new WebViewClient() {
      @Override
      public void onPageFinished(WebView view, String url) {
        if (url.contains("laluminium.be/oauth/oauth2callback.php")) {
          webview.loadUrl("javascript:window.droid.print(document.
             getElementsByTagName('body')[0].innerText);");
        }
        super.onPageFinished(view, url);
      }
    });
```

Now the application has the *token_access*, it can use the google contacts from the user.

This application explains how a user can authenticate himself in a secure way and choose what he wants to share with this application.

6.4.3 Method 3: Google+

Method 3 allows the user to log into the application with his Google+ account. With this method, he cannot choose what he decides to share with the application.

Like I explained in section 2.2.3.2, I had to add two permissions in the manifest:

- GET_ACCOUNT: This permission allows my application to access to the accounts available on my phone.
- USE_CREDENTIALS: This permission allows my application to request authtokens from the AccountManager.
- INTERNET: This permission allows my application to access the Internet.

```
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.USE_CREDENTIALS" />
```

The code of this method is extracted from the Android libraries. The code can be found in the appendix A.

6.5 Comparison between OAuth 2.0 and OpenID

6.5.1 Introduction of both protocols

First of all, we will restate the typical user scenario for both protocols. Firstly, with the OpenID use case and secondly with the OAuth use case.

- **OpenID:**
  1. The user wants to access his account on a website which authorizes OpenID.
  2. The Relying Party asks the user his OpenID credentials. Once the RP has the credentials, it needs to verify them through the OpenID Identity Provider.
  3. If the validation is successful, the RP allows the user to access his account.
- **OAuth:**
  1. The user is on a website - SensCritique - and wants to connect himself without creating a new account. The website offers him to connect via a Facebook account. The user chooses to do so.
  2. The consumer - the website - redirects the user to a Facebook dialog which allows the user to authenticate himself with his Facebook credentials.
  3. Facebook asks the user which information he allows the consumer to access. The user decides : he can allow the website to access to all his information or only some of them.
  4. Facebook redirects the user back to the website which can access to some of the user information.
  5. The users is now connected to SensCritique.

6.5.2 Comparison of both protocols

Both scenarios indicate how the protocols set appart. OpenID protocol is about the authentication: a user can identify himself with an unique URL. OAuth is about authorization: the user grants permission on his data on some website to another website without providing this second website his credentials from the original website.

OAuth and OpenID are different in many ways. If we first take a look at their goals, we notice that OpenID was created to allow the federated authentication. What does it mean? This means let a third-party authenticate users for you. The term federated

is important in the understanding of OpenID: any provider can be used. Regarding to OAuth, it was created to avoid the need for users to share their passwords and their usernames with third-party. Both protocols can veryfied an assertion but OpenID is limited to confirm the assertion "who I am?" OAuth gives tokens which can be exchanged for an access.

To continue, both protocols have special caracteristics. Even if both protocols offer the same service - identify a user at one place and redirect him after having checked his identity - they have different functionalities . Concerning OpenID, the user uses a URI to authenticate himself, a lot of users do not understand this notion properly. Concerning OAuth, the main interest is the token which can be used many times. Unlike OpenID, Oauth is not limited to the authentication: OAuth allows access to protected resources.

If we now take a look at they implementation in a last time, both protocols share the same kind of architecture: use a third-party to verify the user identity. With OAuth, the user grants access to his protected resources. With OpenID, the user grants access to his identity. This architecture is all they have in common.

6.5.3 A way to combine the two protocols: OpenID connect

> *"OpenID Connect [...] is a standards-based authentication protocol built on top of OAuth 2.0 developed by the OpenID Foundation. Like TCP for networking or SMTP for email, OpenId Connect is meant to provide a fully specified protocol for authentication that lets any compliant implementation authenticate users from any other compliant system."*[28]

6.5.3.1 What is OpenID Connect

This paragraph is inspired by the article of Michael Calore: *"New 'OpenID Connect' Proposal Could Solve Many of the Social Web's Woes".*[29]

OpenID Connect essentially rebuilds OpenID on top of OAuth 2.0. It is a way to combine the two most popular protocols to authenticate user and to share data around the network.

> *"OpenID Connect is an attempt to pull the best pieces of two separate technologies together, to create a single technology stack that's simpler for everyone to use"* - David Recordon [29]

This proposal tries to combine the login interactions with the data sharing interactions

into a simple step.

Both OpenID and OAuth have been adopted by websites and applications in the last couple of years. Moreover, we know that each of them suffer from various problems. A lot of issues concerning OAuth 1.0 were solved with OAuth 2.0. Unlike OAuth, OpenID has not been updated since 2007,which is a very long time on the web, espacially in the mobile environement which has been developping so fast lately.

As explained earlier in my thesis, both technologies serve different purposes: OpenID allows the user to prove his identity while OAuth allwos the user to share his information with applications.

To conclude this paragraph, I can say that OpenID Connect takes the best part of OAuth and OpenID to create a simple identity layer.

6.6 Why do few people use OAuth?

OAuth is not so much used, whether the 1.0 version or the 2.0 version. Indeed, each version has his pros and cons.Version 2.0 should resolve all the weaknesses of former version 1.0, but it was not the case. This version brings his own weaknesses, which are bigger than the weaknesses of the 1.0 version.

In fact, people do not even use the version 1.0 but the version 1.0a. Indeed, the version 1.0 had a breach which has been fix in the version 1.0a. Most of users of OAuth do not use OAuth 2.0 because a lot of changes need to be done to ensure security with this version.

7 Conclusion

This section presents results of the research and conclusions that came up all along my thesis. It aims at summarizing the questions from Section 1.3 that have been answered. Every question will be associated to a subsection.

7.1 What is tokenization?

This subject is treated in the beginning of my thesis. The first part of the section dedicated to tokenization explains what is tokenization, the interests of tokens, how they work and stored (in general and in application), their use in the authentication process.

7.1.1 On wich conecpt it is based?

Tokenization is the process of substituting a sensitive data element with an easily reversible substitute. Tokenization can be used to safeguard sensitive data like passwords. To do so, three concepts exist: the random number generation, the encryption and the one-way hash function. Each concept transforms the original value in a token. Another important concept in this process, is that the application has no contact with the original value to improve security. Tokenization brings more security in the authentication process as explain in section 4.1.6.

7.2 How does the one-time password mechanism work?

In the section center on one-timepPassword, I explained the different types of existing one-time password and how can they be generated. After that I explained the potential vulnerabilites and some examples which explained how the OTP could be stolen.

7.2.1 Which are the vulnerabilities?

It is important to remember that one-time password are useless if they are not used properly. In section 4.2.2, I tried to summarize the vulnerabilities of the one-time password, but I could not give a full list of these vulnerabilities. The vulnerabilities that I explained came from the tokenization security requirements. Based on this list, I was able to identify some scenarios of potential breaches that a system using ont-time password has to prevent, like the issues linked to the generation of the one-time password.

7.3 What is Single Sign-On?

Section 5 introduces the concept of the single sign-on. This concept is a property of access control of multiple but independent systems. With this property, the user needs to authenticate himself only once: when he authenticate himself correctly he can access to all his sytems without needing to authenticate again. In this section I explained what is this concept, how it works, his advantages and his limits.

7.3.1 What are the interests of using this solution?

Concerning the issues raised in my thesis, single sign-on seems to be a good solution. Indeed it can solve some of these known issues: limited the transmition of passwords around the network, created too much credentials. In the mobile environment, single sign-on has a huge interest. Indeed, when this property is used the users do not have to create new credentials for each new applications they install. With the increase of the number of applications existing, this property can save a lot of time to the users and improve their experiences.

André Andrade explains very well interests of using SSO in his thesis:

> *"Single Sign-On (SSO) systems were developed to solve those issues. They enable users to access a multitude of services with one single login. This enhances usability and security in web authentication by restricting the number of passwords used to merely one, in an ideally wide set of services. Hence, users need to remember less passwords, which in turn avoids reusing and writing them down and makes changing them much more effortless. Service providers also benefit from SSO by possibly outsourcing identity management to other trusted third parties. SSO systems merely improve the characteristics of any particular authentication mechanism, which is why we consider them as good base for our strong authentication protocol."* [30]

7.4 Is it possible to do Sing Sign-On with the OAuth protocol?

Like single sign-on is a concept, protocols are needed to implement it. In the section 6, I introduced a possible protocol: OAuth. Indeed, OAuth can be used to implement the Single Sign-On property. A lot of entreprise use it, like Google or Facebook, in the web environment but also in the mobile environement.

7.4.1 What are the interests of using OAuth?

This protocol has a huge interest in the mobile environment regarding to the authentication of the users in applications. Moreover, his interest is also in the management of the user auhtorizations. Indeed, with OAuth, the user can not only authenticate himself with a single set of username/password but also choose which information he want to share with the application. The mobile environment, where the management of permission is developed in the kernel, allow the user to choose what he shared: it is a very important side.

7.5 Remaining leads for future work

What could be done for the next release of OAuth? Well, I think the biggest lack of OAuth 2.0 is clearly the specifications that are leaving too much freedom for the providers. At the moment, if you compare two provider which claims implementing OAuth 2.0 following its IETF specification,What could be done for the next release of OAuth? Well, I think the biggest lack of OAuth 2.0 is clearly the specifications which are leaving too much freedom to providers. At the moment, if you compare two providers which claim implementing OAuth 2.0 following its IETF specification, you will easily see strong technical differences between them. The spcefications should be more accurate, more restrictive. Another idea should be to add the service discovery for providers. A way to them to describe themselves. A kind of manifest, which can be read to know the endpoint url or where the users can manage the applications having access to their resources or even revoke accesses.

7.6 What about OAuth in mobile environment?

As explained before, when you use the OAuth protocol, you need a client secret obtained from the provider. Unfortunately, if you are doing this directly in your application, you must store this secret in your database or in a system file: is it the best way to do? Obvioulsy not: storing the secret in your application is the best way to compromise it. The best suitable solution is to obtain the access token from a web server (using a webview like my prototype does) and then forward the token to the mobile application. The application can then access the wanted resource using this precious token.

OAuth in the mobile environment is thus possible but, I would say that it does not perfectly fit it. A future work for OAuth would also be to adapt it for mobile applications.

References

[1] Gartner. http://www.gartner.com/technology/home.jsp.

[2] Google play : avec 700 000 applications, il rattrape l'app store. October 2012. http://www.atlantico.fr/atlantico-light/google-play-avec-700-000-applications-rattrape-app-store-530830.html.

[3] Go-Gulf.com Corporation. Smartphone users in the world. January 2012. http://www.go-gulf.com/blog/smartphone.

[4] Eric Chu. A closer look at 10 billion downloads. December 2011. http://android-developers.blogspot.com/2011/12/closer-look-at-10-billion-downloads.html.

[5] Plewes Anthony. New generation of phone authentication. September 2010. http://www.orange-business.com/en/blogs/enterprising-business/crm/new-generation-of-phone-authentication.

[6] George Pavlou Arosha K Bandara Emil C Lupu Alessandra Russo Naranker Dulay Morris Sloman Javier Rubio-Loyola Marinos Charalambides, Paris Flegkas. Policy conflict analysis for quality of service management. 2005. http://www.ee.ucl.ac.uk/~gpavlou/Publications/Conference-papers/Charal-05.pdf.

[7] Google Corporation. Android security overview. http://source.android.com/devices/tech/security/index.html.

[8] Dan Griffin. Safer authentication with a one-time password solution. May 2008.

[9] Wikipedia. Key (cryptography). Arpil 2010.

[10] Eric Conrad. Explanation of the three types of cryptosystems. January 2007.

[11] D. M'Raihi, M. Bellare, F. Hoornaert, D. Naccache, and O. Ranen. rfc4226 - hotp: An hmac-based one-time password algorithm. December 2005.

[12] Jean-Noël COLIN. Authentication. January 2012.

[13] Shibboleth. http://shibboleth.net/.

[14] http://openid.net/.

[15] David Hamilton. Yahoo's password leak: What you need to know (faq). June 2012. http://news.cnet.com/8301-1009_3-57471178-83/yahoos-password-leak-what-you-need-to-know-faq/.

[16] Lance Whitney. Millions of linkedin passwords reportedly leaked online. June 2012. http://news.cnet.com/8301-1009_3-57448079-83/millions-of-linkedin-passwords-reportedly-leaked-online/.

[17] Wikipedia. Principle of least privilege. http://en.wikipedia.org/wiki/Principle_of_least_privilege.

[18] Jon Avery. Security basics part 1: Principle of least privilege. http://techblog.appnexus.com/2011/security-basics-part-1-principle-of-least-privilege/.

[19] Margaret Rouse. Tokenization. May 2011. http://searchsecurity.techtarget.com/definition/tokenization.

[20] Tadd Axon, Mark Bower, Drew Dillon, Jacobs Jay, Ulf Mattsson, Martin McKeay, Gary Paigon, Branden Williams, Lucas Zaichkowsky, and Sue Zloth. Understanding and selecting a tokenization solution. September 2010.

[21] Ericka Chickowski. One-time passwords on the rise but come with some risks. October 2010.

[22] Julian Lovelock. Are otp tokens secure? August 2011.

[23] Sso - single sign on. http://www-igm.univ-mlv.fr/~dr/XPOSE2006/CLERET/techniques.html.

[24] Eran Hammer. Huenivers. 2012.

[25] D. Hardt Ed. The oauth 2.0 authorization framework. October 2012. http://tools.ietf.org/html/rfc6749.

[26] Graham Cluley. Facebook flaw allowed websites to steal users' personal data without consent. February 2011. http://nakedsecurity.sophos.com/2011/02/02/facebook-flaw-websites-steal-personal-data/.

[27] Tarik Benmerar. Le protocole libre oauth 2.0 est-il un échec ? July 2012. http://www.developpez.com/actu/46115/Le-protocole-libre-OAuth-2-0-est-il-un-echec-L-auteur-en-chef-de-sa-specification-demissionne-et-en-retire-son-nom/.

[28] Google. Openid connect. https://developers.google.com/accounts/cookbook/technologies/OpenID-Connect.

[29] Michael Calore. New 'openid connect' proposal could solve many of the social web's woes. http://www.webmonkey.com/2010/05/new-openid-connect-proposal-could-solve-many-of-the-social-webs-woes/.

[30] Andre Andrade. Strong mobile authentication in single sign-on systems. May 2011.

[31] Jean-Noël COLIN. Introduction à la cryptographie. 2010.

[32] Nirgoldshlager. How i hacked facebook oauth to get full permission on any facebook account (without app "allow" interaction). March 2013.

[33] Wikipedia. Single sign-on. http://en.wikipedia.org/wiki/Single_sign-on.

[34] Wikipedia. Integrated windows authentication. http://en.wikipedia.org/wiki/Integrated_Windows_Authentication.

[35] Android Developers. Authenticating to oauth2 services. http://developer.android.com/training/id-auth/authenticate.html.

[36] Authenticationworld.com - the buniness of authentication. http://www.authenticationworld.com/Single-Sign-On-Authentication/.

[37] Google. https://support.google.com/a/bin/answer.py?hl=fr&answer=175197.

[38] Wikipedia. Openid. http://fr.wikipedia.org/wiki/OpenID.

[39] Ens Hammer. The oauth 1.0 protocol. April 2010.

[40] Jon Avery. Oauth openid: You are barking up the wrong tree if you think they are the same thing. http://softwareas.com/oauth-openid-youre-barking-up-the-wrong-tree-if-you-think-theyre-the-same-thing.

[41] Welcome to openid connect. http://openid.net/connect/.

Appendices

A Application code

```
----------------------------------------
Main.java
----------------------------------------

package com.example.oauth;

import android.app.Activity;
import android.app.ProgressDialog;
import android.content.Intent;
import android.content.IntentSender.SendIntentException;
import android.os.Bundle;
import android.util.Log;
import android.view.Menu;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

import com.google.android.gms.common.ConnectionResult;
import com.google.android.gms.common.GooglePlayServicesClient.
    ConnectionCallbacks;
import com.google.android.gms.common.GooglePlayServicesClient.
    OnConnectionFailedListener;
import com.google.android.gms.common.SignInButton;
import com.google.android.gms.plus.PlusClient;

public class Main extends Activity implements View.OnClickListener,
    ConnectionCallbacks, OnConnectionFailedListener {

  // key : 80:16:63:48:07:B2:38:BB:74:EC:75:6D:23:07:D9:09:F4:46:4D:21
  private static final String TAG = "Main";
  private static final int REQUEST_CODE_RESOLVE_ERR = 9000;

  private ProgressDialog mConnectionProgressDialog;
  private PlusClient mPlusClient;
  private ConnectionResult mConnectionResult;

  private SignInButton google_signin;
  private Button deco;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);


    setContentView(R.layout.activity_main);

```

```
44      google_signin = (com.google.android.gms.common.SignInButton)
          findViewById(R.id.button_oauth);
45      deco = (Button) findViewById(R.id.button_deconnect);
46      google_signin.setOnClickListener(this);
47
48      mPlusClient = new PlusClient.Builder(this, this, this).
          setVisibleActivities("http://schemas.google.com/AddActivity", "
          http://schemas.google.com/BuyActivity").build();
49
50      // Progress bar to be displayed if the connection failure is not
          resolved.
51      mConnectionProgressDialog = new ProgressDialog(this);
52      mConnectionProgressDialog.setMessage("Signing␣in...");
53    }
54
55    @Override
56    protected void onStart() {
57      super.onStart();
58      mPlusClient.connect();
59    }
60
61    @Override
62    protected void onStop() {
63      super.onStop();
64      mPlusClient.disconnect();
65    }
66
67    @Override
68    public void onConnectionFailed(ConnectionResult result) {
69      if (mConnectionProgressDialog.isShowing()) {
70        // The user clicked the sign-in button already. Start to resolve
71        // connection errors. Wait until onConnected() to dismiss the
72        // connection dialog.
73        if (result.hasResolution()) {
74          try {
75            result.startResolutionForResult(this,
76                REQUEST_CODE_RESOLVE_ERR);
77          } catch (SendIntentException e) {
78            mPlusClient.connect();
79          }
80        }
81      }
82      // Save the result and resolve the connection failure upon a user
          click.
83      mConnectionResult = result;
84    }
85
86    @Override
87    protected void onActivityResult(int requestCode, int responseCode,
88        Intent intent) {
89      if (requestCode == REQUEST_CODE_RESOLVE_ERR && responseCode ==
          RESULT_OK) {
90        mConnectionResult = null;
```

```
91        mPlusClient.connect();
92      }
93    }
94
95    @Override
96    public void onConnected(Bundle connectionHint) {
97      String account_name = mPlusClient.getAccountName();
98
99      deco.setVisibility(View.VISIBLE);
100     google_signin.setVisibility(View.GONE);
101
102     Toast.makeText(getApplicationContext(), account_name + "␣is␣now␣
          connected␣!", 500).show();
103   }
104
105   @Override
106   public void onDisconnected() {
107     Log.d(TAG, "disconnected");
108   }
109
110
111
112   @Override
113   public boolean onCreateOptionsMenu(Menu menu) {
114     // Inflate the menu; this adds items to the action bar if it is
          present.
115     getMenuInflater().inflate(R.menu.main, menu);
116     return true;
117   }
118
119   @Override
120   public void onClick(View view) {
121     if (view.getId() == R.id.button_oauth && !mPlusClient.isConnected
          ()) {
122         if (mConnectionResult == null) {
123             mConnectionProgressDialog.show();
124         } else {
125             try {
126                 mConnectionResult.startResolutionForResult(this,
                      REQUEST_CODE_RESOLVE_ERR);
127             } catch (SendIntentException e) {
128                 // Try connecting again.
129                 mConnectionResult = null;
130                 mPlusClient.connect();
131             }
132         }
133     }
134   }
135
136   public void deco(View view) {
137     if (view.getId() == R.id.button_deconnect) {
138             if (mPlusClient.isConnected()) {
139                 mPlusClient.clearDefaultAccount();
```

```
                    mPlusClient.disconnect();
                    mPlusClient.connect();
                }
            }

    deco.setVisibility(View.GONE);
    google_signin.setVisibility(View.VISIBLE);
      }


  // launch the classic process
  public void connectionClassic(View view){
    // TODO
  }

  // launch the OAuth process
  public void connectionOAuth(View view){
    Intent intent = new Intent(this, ConnectionOAuth.class);
    startActivity(intent);
  }

}


----------------------------------------
ConnectionOAuth.java
----------------------------------------

package com.example.oauth;

import org.json.JSONObject;

import android.annotation.SuppressLint;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.Menu;
import android.webkit.JavascriptInterface;
import android.webkit.WebView;
import android.webkit.WebViewClient;

public class ConnectionOAuth extends Activity {

  @SuppressLint({ "SetJavaScriptEnabled", "JavascriptInterface" })
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    final WebView webview = new WebView(this);
    setContentView(webview);
    webview.getSettings().setJavaScriptEnabled(true);

    webview.addJavascriptInterface(new JIFace(), "droid");
```

```
193
194     webview.setWebViewClient(new WebViewClient() {
195       @Override
196       public void onPageFinished(WebView view, String url) {
197         if (url.contains("laluminium.be/oauth/oauth2callback.php")) {
198           webview.loadUrl("javascript:window.droid.print(document.
              getElementsByTagName('body')[0].innerText);");
199         }
200         super.onPageFinished(view, url);
201       }
202     });
203
204     // prepare the url with all the information needed
205     String response_type = "code";
206     String client_id = "257209170749-v1odkevbcap23ik28l47l5p55rh3c9gc.
          apps.googleusercontent.com";
207     String redirect_uri = "http://www.laluminium.be/oauth/
          oauth2callback.php";
208     String scope = "https://www.google.com/m8/feeds";
209     String access_type = "online";
210     String approval_prompt = "force";
211     String state = "";
212     String base_url = "https://accounts.google.com/o/oauth2/auth";
213
214     String url = base_url + "?" + "response_type=" + response_type
215         + "&client_id=" + client_id + "&redirect_uri=" + redirect_uri
216         + "&scope=" + scope + "&access_type=" + access_type
217         + "&approval_prompt=" + approval_prompt + "&state=" + state;
218
219     // execute the url
220     webview.loadUrl(url);
221   }
222
223
224   @Override
225   public boolean onCreateOptionsMenu(Menu menu) {
226     // Inflate the menu; this adds items to the action bar if it is
          present.
227     getMenuInflater().inflate(R.menu.connection_oauth, menu);
228     return true;
229   }
230
231   class JIFace {
232     @JavascriptInterface
233     public void print(String data) {
234       try {
235
236         // extract the values of the response
237         JSONObject output = new JSONObject(data);
238         String accessToken = (String) output.get("access_token");
239         String tokenType = (String) output.get("token_type");
240         Integer expires_in = (Integer) output.get("expires_in");
241
```

```
242         // launch the new activity
243         Intent intent = new Intent(getApplicationContext(),
              UseAccessToken.class);
244         intent.putExtra("accessToken", accessToken);
245         intent.putExtra("tokenType", tokenType);
246         intent.putExtra("expires_in", expires_in);
247         startActivity(intent);
248
249         // go back to the first screen if the user press the go back
              button
250         finish();
251
252       } catch (Exception e) {
253         e.printStackTrace();
254         Log.e("OAUTH","Authentication␣Server␣replied␣with␣a␣Non-JSON␣
              string");
255       }
256     }
257   }
258
259 }
260
261 ----------------------------------------
262 UseAccessToken.java
263 ----------------------------------------
264
265 package com.example.oauth;
266
267 import android.os.Bundle;
268 import android.app.Activity;
269 import android.view.Menu;
270 import android.widget.TextView;
271
272 public class UseAccessToken extends Activity {
273
274   @Override
275   protected void onCreate(Bundle savedInstanceState) {
276     super.onCreate(savedInstanceState);
277     setContentView(R.layout.use_access_token);
278
279     String accesToken = getIntent().getExtras().getString("accessToken
        ");
280     String tokenType = getIntent().getExtras().getString("tokenType");
281     int expires_in = getIntent().getExtras().getInt("expires_in");
282
283     ((TextView)findViewById(R.id.accessToken)).setText(accesToken);
284     ((TextView)findViewById(R.id.tokenType)).setText(tokenType);
285     ((TextView)findViewById(R.id.expires_in)).setText(String.valueOf(
        expires_in));
286   }
287
288   @Override
289   public boolean onCreateOptionsMenu(Menu menu) {
```

```
290     // Inflate the menu; this adds items to the action bar if it is
          present.
291     getMenuInflater().inflate(R.menu.use_access_token, menu);
292     return true;
293   }
294
295 }
296
297 ----------------------------------------
298 AndroidManifest.xml
299 ----------------------------------------
300
301 <?xml version="1.0" encoding="utf-8"?>
302 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
303     package="com.example.oauth"
304     android:versionCode="1"
305     android:versionName="1.0" >
306
307     <uses-sdk
308         android:minSdkVersion="11"
309         android:targetSdkVersion="17" />
310
311     <uses-permission android:name="android.permission.GET_ACCOUNTS" />
312     <uses-permission android:name="android.permission.INTERNET" />
313     <uses-permission android:name="android.permission.USE_CREDENTIALS"
          />
314
315     <application
316         android:allowBackup="true"
317         android:icon="@drawable/ic_launcher"
318         android:label="@string/app_name"
319         android:theme="@style/AppTheme" >
320         <activity
321             android:name="com.example.oauth.Main"
322             android:label="@string/app_name" >
323             <intent-filter>
324                 <action android:name="android.intent.action.MAIN" />
325
326                 <category android:name="android.intent.category.
                      LAUNCHER" />
327             </intent-filter>
328         </activity>
329         <activity
330             android:name="com.example.oauth.ConnectionClassic"
331             android:label="@string/title_activity_connection_classic"
                  >
332         </activity>
333         <activity
334             android:name="com.example.oauth.ConnectionOAuth"
335             android:label="@string/title_activity_connection_oauth" >
336         </activity>
337         <activity
338             android:name="com.example.oauth.UseAccessToken"
```

```
            android:label="@string/title_activity_use_access_token" >
        </activity>
        <activity
            android:name="com.example.oauth.ConnectionGoogle"
            android:label="@string/title_activity_connection_google" >
        </activity>
    </application>

</manifest>
```