Secure Instant Messaging

Master Thesis

Royce Lanson Pinto

Supervisors: Dr. Michael Spahn, Prof. Dr. Christian Baun and Prof. Dr. Matthias Deegener

Frankfurt am Main, 2014

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Department of Computer Science and Engineering

in cooperation with

SAP RESEARCH

SECURE
INSTANT MESSAGING

by
Royce Lanson Pinto

A thesis submitted in partial fulfillment
of the requirements for the degree
of

MASTER OF SCIENCE

in

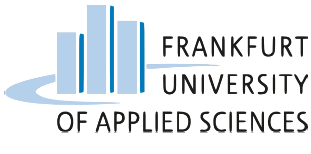
High Integrity Systems

Under the guidance of

Dr. Michael Spahn
Senior Researcher
SAP Research CEC Darmstadt

Prof. Dr. Christian Baun
Supervisor 1
Computer Science and Engineering
Frankfurt University of Applied Sciences

Prof. Dr. Matthias Deegener
Supervisor 2
Computer Science and Engineering
Frankfurt University of Applied Sciences



The work in this thesis was supported by SAP SE. Their cooperation is hereby gratefully acknowledged. This thesis, submitted in partial fulfillment of the requirements for the award of Master of Science in High Integrity Systems, Faculty of Computer Science and Engineering, Frankfurt University of Applied Sciences. The document has not been submitted for qualifications at any other academic institution.

Secure Instant Messaging

Royce Lanson Pinto
Matr.-Nr: 1029035

30 ECTS thesis submitted in partial fulfillment of a
Master of Science degree in High Integrity Systems.

External Supervisor: Dr. Michael Spahn
Thesis supervisor: Dr. Prof. Christian Baun
Co-supervisor: Dr. Prof. Matthias Deegener

Date of registration: 9.04.2014
Date of submission: 9.10.2014

Contact the author:

University address:
Frankfurt University of Applied Sciences
Nibelungenplatz 1
D-60318, Frankfurt am Main
http://www.frankfurt-university.de/

Tel: +49 069/1533-0
Fax: +49 069/1533-2400
post@fh-frankfurt.de

The purpose of art is to lay bare the questions
which have been hidden by the answers
— James Arthur Baldwin

To my parents Lancy and Leena Pinto.
All I have and will accomplish are only possible
due to their love and sacrifices....

Preface

This thesis would not have been possible without the guidance and the help of several individuals who in one way or another contributed and extended their valuable assistance in the preparation and completion of this study.

First and foremost, my utmost gratitude to Dr. Michael Spahn, my advisor and mentor at SAP SE, who encouraged me to tackle challenges which, at first, seemed insurmountable. I have learned much from him and am grateful for it. I also want to thank Dr. Sebastian Wieczorek at SAP SE, for his guidance and support at every point of the way.

I am also grateful to my supervisor- Professor Dr. Christian Baun at the Frankfurt University of Applied Sciences. I thank him from the bottom of my heart for sharing valuable insights in the relevance of the study. I also want to thank my co-supervisor Professor Dr. Matthias Deegener for his help and good feedback. I also thank Simon Hoffmann and Johannes Stehlin, Gobal Mobile IT CoE, SAP SE for their assistance and help on SAP Wire. Without their assistance I would not be able to understand the workings of SAP Wire.

I am grateful to our renowned institution, The Frankfurt University of Applied Sciences with its very ideals and inspirations for having me provided with the facilities which were required for completion of this thesis. I also want to thank my friends for sharing their insights and their support. And last but certainly not the least, I would like to thank my parents and my brother. They have always supported and encouraged me with their best wishes.

Frankfurt am Main, 9 October 2014 Royce Lanson Pinto.

Abstract

With the rise of the internet-connected always-on smart phones, instant messaging (IM) applications have become more and more ubiquitous, and for some people even act as their main center of private communication. These kinds of applications are easy to use, but they usually do not technically ensure that sent messages can only be read by the intended recipient. One reason is due to the technical limitations of current technology like sending plaintext across or using HTTP and another is that some countries enforce operators of communication services to enable tapping interfaces for according authorities.

Thus, if technical ways of tapping communication exists, sooner or later they will be used and as we learned from Edward Snowden [27], these are widely used. The awareness of communication tapping leads to a changed perception of communication. The change is not only about privacy, it's a change in behavior. People act and share differently when they know that a photo or video will live forever. As no one can guarantee that the message will never be re-viewed in the future or if the political climate changes to the extreme then this can be used against the user. A possibility to lower the risk of information leakage is to encrypt the content of sent messages in a way only decryptable by the intended recipient. To foster more secure communication, one not only needs to tackle the challenge of finding ways of making message exchange itself more secure, but also to make it as easy to use as possible. To foster its adoption the best encryption technology won't be used if its usage is too complex for the masses.

How can IM-based communication, especially in an enterprise context, be made more secure? How to exchange messages if one can neither trust the communication channels (i.e. internet;HTTP) nor the server processing the data to not reveal the data to unwanted third

parties? As a concrete challenge, the most important security-related challenges should be identified, best practice solution to tackle them found, a concept for enhancing IM systems with additional security created, as well as prove the concept by doing a concrete implementation enhancing a real enterprise IM system. The IM system should be significantly improved in at least one of the identified problem domains.

This thesis describes how this challenge was solved. In the following, First, the security-related challenges to IM are presented along with the related research into the domain and current solutions to come to a best-practice solution. Then a concept is proposed on how to create a Secure IM system which can overcome the challenges described. Then, the concept is proved by implementing the solution to a real IM system. Finally, findings on the IM concept are documented and how and in which way it was improved and where future work is still necessary.

Declaration

I hereby declare to have written this master thesis entirely by myself and used no other sources than those listed in the resources part at the end of this thesis. Furthermore, I affirm that this thesis has never been published in Germany or in any other country.

Frankfurt am Main, October 9 2014

Royce Lanson Pinto

Contents

List of Figures

List of Tables

1 Introduction

In this chapter the topic- Instant Messaging (IM) is introduced, the questions that will be discussed are presented and also the motivation behind the research and scope are outlined. The chapter also includes reading directions, which describe the organization of report.

1.1 Background

> **Instant Messaging** (sometimes called IM or IMing) is a type of online conversation which offers real-time text transmission over the Internet.

Instant Messaging (IM) is extremely popular worldwide. What began as a simple application for consumers just to chat with friends, IM systems have grown to be an important communication platform for both the Internet community and the business world. Globally, Informa forecasts that "...SMS traffic will total 9.4 trillion messages by 2016, up from 5.9 trillion messages in 2011"[2]. However, SMS's share of global mobile messaging traffic will fall from 64.1% in 2011, to 42.1% in 2016. At the same time, global mobile Instant Messaging traffic will increase from 1.6 trillion messages in 2011 to 7.7 trillion messages in 2016, doubling its share of global messaging traffic from 17.1% in 2011 to 34.6% in 2016[2]. Worldwide IM accounts are expected to grow from over 3.4 billion in 2013 to over 4.4 billion by year-end 2017, representing an average growth rate of about 7%[10]. UK based institute Juniper Research recently forecasts

that global mobile messaging traffic will reach 28.2 trillion annually by 2017, nearly double the 14.7 trillion messages which will be sent in the year 2012. [12].

Historically, IM systems are present since the applications `talk` and `write` where developed for the UNIX operating system. However since the explosion of the use of the internet coupled with the massive increase in use of the mobile devices the popularity of IM applications has exploded. In essence it is an integral part of ones social life. As per the study conducted by Radicati[10] Instant Messaging can be roughly divided into the following segments: Public IM Networks, Enterprise IM Platforms and Mobile IM.

Public IM Networks provide IM services primarily for the consumer market, but are also used by business users. Vendors include Facebook, Google, Microsoft / Skype, Yahoo! and others. Enterprise IM Platforms are designed for businesses, they have improved security and privacy features that cannot typically be attained through Public IM networks. Vendors include: Cisco, IBM, Microsoft, Novell and others.

There has been a move from desktop devices to mobile devices. Since the advent of the iPhone and Android devices there has been a considerable shift to portable devices. There has been a move from desktop computers to touch based tablets. Even the most widely used operating system- Windows is now designed for touch based portable devices. So naturally Instant Messaging applications have become more popular on these devices. IM services that are developed specifically for use on mobile devices, and typically serve as alternatives to SMS messaging, The Mobile IM segment includes: Apple iMessage, BlackBerry Messenger, WhatsApp Messenger, and others.

Being such a popular method of communication over the internet, IM is becoming a replacement for e-mail. Unlike e-mail, Instant Messaging allows users to see whether a chosen friend or co-worker is connected to the Internet. Typically, the Instant Messaging service will alert a user if somebody on the user's list of correspondents is on-line. Instant Messaging also differs from e-mail in that messages are exchanged directly almost instantly, allowing for a two-way communication in real-time.

Because of the almost immediate two-way nature of communication, many users feel that the use of Instant Messaging in the workplace leads to more effective and efficient workplace com-

munications and, therefore, to higher productivity. As a result, IM is increasing in popularity in both professional and personal applications. However, as with most things Internet based, the increasing use of Instant Messaging has led to an associated increase in the number of security risks.

Though Instant Messaging competes with social media and SMS as a means of communication, Mobile IM segments have shown solid growth in 2013.[10] IM remains a popular form of communication with consumers and business users, despite popular alternatives like Facebook or SMS / text messaging. While this competition has in some ways limited growth, it is also somewhat revealing about the blurred boundaries between these technologies. IM is becoming far more than text-only communication and is more properly understood as a multimedia messaging and sharing platform with a robust feature set.

1.2 Motivation

Instant Messaging is in today's time ubiquitous and is used by many employees at home and at the work space. In many companies, therefore find themselves next to Enterprise Instant Messaging services (EIM) as Sametime, Lync and other a multitude number free communication tools mainly in private use for example Skype, AIM, Hangout and WhatsApp. [35]. For personal use the latter are reasonable, but when they are used in a company in daily communication it implies a security risk. Often they become the target of malware and virus developers not only by phishing communication data can be routed.

One increasingly finds Instant Messaging in the corporate communications, this is thanks to the many chat programs for smartphones and tablets and can be integrated quickly and used easily internal and external employees.

Many users and media worry since the acquisition of the "top dog" WhatsApp by Facebook has led them to expose their data. The fear being that services of this type in addition to the users telephone numbers also have access to the complete address book contained therein. Considering that Instant Messaging applications add a lot of convenience, but mostly security concerns are not addressed in great detail. Most Instant Messaging services often trade speed for security. A simple example being WhatsApp in which the same encryption key is used in

both directions [26]. Hackers are trying to gain access to conversations. Thus, some of the major issues that plague the IM world are: Client Vulnerabilities, Insecure Network Traffic, Open Connections, Identity Theft, Data Theft, Loss of Privacy, Absent Authentication and Social Engineering [25]. The many security mechanisms designed for web based applications are not sufficient for IM. There is no complete collective suite to solve all security problems of the IM system and simple SSL is not enough.

The number of interested parties eager to listen in on online conversations, including what is typed through Instant Messaging, has never been higher. Also, the provider of the service might be coerced or forced to turn over the un-encrypted messages to a third party. It's not just since the disclosure of the extent of National Security Agency's (NSA) spying activities[3], that security and privacy of message exchange matters for users of all kinds may it be a company that wants to avoid leakage of company secrets, people living in countries without freedom of speech that want to communicate freely without having to fear punishment, or people that just want to ensure that their private communication really stays private. Also, if a third party listens onto conversations these can be used to target advertisements.

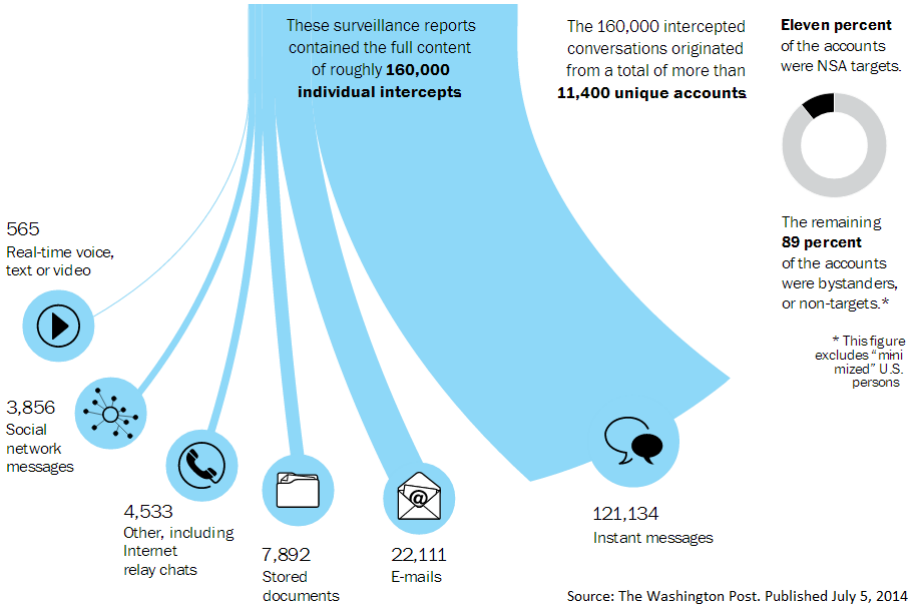


Figure 1.1: Communication surveillance breakdown

The above figure details the communication surveillance breakdown according to the Washington Post, former NSA contractor Edward Snowden who obtained and shared with The

Washington Post a large volume of e-mails, messages, photos and documents intercepted by the NSA from online accounts and network links in the United States. In this cache of communications, the NSA's foreign targets were far outnumbered by ordinary Web users whose content was intercepted "incidentally" as a collateral effect of the surveillance [27]. For example the Washington Post also estimated NSA intercepted and collected about 22,000 surveillance reports from 2009-2012.

If we look at this scenario, why would a user want any third party to read their personal communication? Considering the above figure of a sample set of 160,000 individual intercepts from 11,400 unique accounts, in this sample set, 121,134 messages are intercepted from IM clients! Especially when a large chunk is from the general populace? It blows privacy entirely down the drain. This is like George Orwell's Nineteen Eighty-Four [63], we are tending to a world with omnipresent government surveillance.

The change is not only about privacy, it's a change in behavior. People act and share differently when they know that a photo or video will live forever. A possibility to lower the risk of information leakage is to encrypt the content of sent messages in a way only decryptable by the intended recipient, for example using PGP (Pretty Good Privacy) to encrypt e-mail content to the recipient's public key before sending the e-mail. But such kind of extra steps are hard to understand for average users and kill the ease of use, causing the vast majority of users to continue using more insecure ways of communication because of convenience.

Considering all factors it boils down to a matter of trust. Who do we want to trust? The end users? The server? End users are generally trustworthy because people are generally extremely careful of their devices. The server or administrator but can normally see all conversations that pass through it and can turn them over to a third party or the government when demanded. So the question is what if we cannot trust the server itself?

With all the above threats to IM highlighting the danger of using present IM systems and to improve the state of security we aim to provide a well defined security add-on for current IM system.

1.3 Objective

The majority of Instant Messaging services for consumers in the consumer environment has not been specifically secured. Meanwhile, the tools available generally use standardized methods of encryption, SSL / TLS, but also protect the conversations with private / public key method. The WhatsApp Alternatives Threema, myENIGMA and Telegram prefer self-development in which the particular is praised by Threema for their safe source.

As of now, however anyone who does not want to give up the ability to communicate with the rest of the world, must make compromises in terms of safety.

The aim of this thesis is to analyze the current scenario of instant messaging and identify the security related problems, challenges and state of the art solutions. Using the results obtained, a concept is proposed which can be show how security may be improved on the IM platform and thus achieve Secure IM. This concept is applied to the existing corporate enterprise environment. A look is taken on how a communication system should be composed, that is as secure as possible and at the same time as easy to use which is an open challenge involving a multitude of unanswered research questions.

1.4 Approach Methodology

The basic approach was to understand the pre-requisites, gather information regarding the problem try to address problems in a systematic manner then try to give solutions by means of proof of concepts and demo prototypes.

1.5 Reading directions

The parts of the methodological approach are described in the following chapters:

Chapter 1 provides a basic introduction and what motivates this work. It describes Why Secure Instant Messaging is important to address and why current security solutions do not suffice in the current corporate scenario and need to be complemented. Moreover, it states the goals of this work.

Chapter 2 discusses the state of the art of the current affair of things in the domain of Secure Instant Messaging. It shows that many security ideas exist depending on the given scenario or technique and presents our definition of the Secure Instant Messaging. It details the current trends in applications that use secure concepts in Instant Messaging. Moreover, it states the direction in which way this work is to follow from previous work.

Chapter 3 describes the concept of the Secure IM system proposed along with the requirements. This is tune with the goal of the thesis.

Chapter 4 outlines the implementation of the concept onto a prototype in SAP's corporate environment. It explains technical terms used in the domain of SAP's Cloud offering-The HANA Cloud platform, encryption and the android platform and the libraries used and also details the challenges faced during implementation. It specifies how the concept's principles were applied to the prototype.

Chapter 5 details the test results of the developed app. It explains the tests performed on the app and the testing methodology used.

Chapter 6 is the final chapter of this thesis and details the summary of the entire thesis and explains the scope of future enhancements.

The next section consists of the appendices, glossary of terms and the bibliography of references.

2 State of the Art

This chapter lists the current knowledge and state of research in the field of Secure Instant Messaging and its usages in today's trending applications. We give a close look at existing mobile IM applications and their security offerings. This chapter also details the most significant threats to mobile IM systems.

2.1 Working of Mobile IM Applications

In a bare essence almost all IM clients contain: a contact list, a block list, presence information, email address / telephone number or a unique ID, ability to send and receive files and of course messages (one to one or group).

Communication generally follows a client-server model, each user shares a detail with the server. This can be a mobile number or email address, in some cases this is ofent coupled with a user chosen password with the server. The ID and if present, the password is used for authentication with the server. Messages are mostly sent through the server, but true peer to peer (P2P) communications can also take place. Communication takes place over TCP and is sometimes secured generally

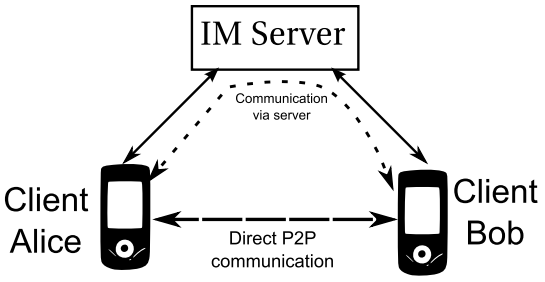


Figure 2.1: Single server (Central) IM model

with SSL.

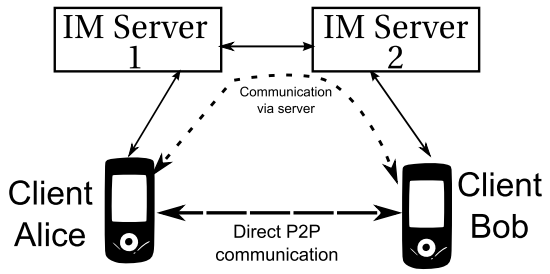


Figure 2.2: Multiple (Distributed) server IM model

An IM server generally appears to be a single entity to a client, however it may be a group of servers controlled by a single IM service provider. Different IM client providers cannot communicate with each other because of different in-compatible protocols or servers. For example if Alice wants to talk to Bob, they must use the same IM service. Alice's messages will be sent to Bob with help of the server. For P2P communication, the server provides information to each party.

Below is a table which shows generally used IM terms:

Contacts	The list of user IDs who use the same IM service and who the user can contact
Blocked contacts	The list of user IDs explicitly barred from contacting the user or seeing his details.
IM user	A human user successfully logged in to an IM server.
IM client	A mobile app that enables a user to use the IM service.
IM server	A server which allows IM clients to access IM features in an IM service.
Presence	Presence information reveals whether or not a user is logged in to an IM server.
One-to-One chat	A user sends or receives messages from another user, generally through the IM server.
Group chat	More than two users exchange messages at a time.
Chat room	A virtual room, generally consisting of many users who exchange Instant Messages.

Table 2.1: IM-related definitions

2.2 Threats to Instant Messaging

This section lists the most significant threats to public IM systems. The list is formulated from known attack forms, and IM protocol and implementation flaws that may allow attacks. The aim of this list is to acquire insight to aid in designing a robust security protocol for IM systems.

2.2.1 General Threats

Insecure Connections

The greatest threat to IM networks is in their open, insecure connections. Because IM connections are mostly based on the client-server architecture. Authentication first takes place, most requests lack authentication (except in the login message), confidentiality and integrity.This may lead to other security vulnerabilities including impersonation, denial of service (DoS), man-in-the-middle, replay and more.

Impersonation

Attackers may impersonate valid users. Attackers can snoop and take over client-to-server connections. It is thus quite easy to impersonate any connection via man-in-the-middle attacks. This allows the attacker to send and read messages meant for another user this is a major risk to IM applications which do not offer encryption.

Denial of Service (DoS)

A DoS attack may simply cause the client never to connect to the server and crash. Flooding with unwanted messages is possible when users can receive messages from everyone. However, IM clients generally support user blocking. A victim can block the attacker's account ID easily; however, attackers may get through this barrier by using many compromised accounts simultaneously.

DNS Spoofing

An attacker or a malicious program can modify the TCP / IP settings of a clients system to point to a different DNS. The attacker will setup a duplicate or rogue IM server. The client has no way of knowing or verifiying if the server is legitimate. A server verifies the user through his username and password. So the server is susceptible to man-in-the-middle attacks.

Spam sent via IM Systems

The default security settings in IM clients are usually at the lower end of the client's security capability. Most IM clients allow anyone from the same IM service to contact a user by default. Allowing message reception from everyone opens the door to a nuisance – `spim` – i.e. spam sent via IM systems.

This is because most Instant Messaging apps do not have an option enabled by default that allows only users added to a persons contact list to be authorized to message the user.

Malicious Hyperlinks

Links to web pages containing malicious content can be sent within normal Instant Messages. These are meant to mislead the user receiving the hyperlink having an innocent visible text to visit a web site corresponding to a deceitful link. Also, malicious hyperlinks can vector users to phishing web sites.

2.2.2 Attacks

> **Cryptanalysis** is the art of deciphering encrypted communications without knowing the proper keys.
> *Mathematics Department of the University of Colorado Boulder*

Since we are dealing with Secure Instant Messaging, at the most basic the system uses encrypted messages. The attacker might try to break the encryption and read or change these messages. There are the following attack forms.

Replay Attacks

A replay attack is a network attack in which data is maliciously transmitted or delayed or fraudulently repeated. The attacker who intercepts data and sends it again can do a packet substitution or change the data.

Man-in-the-middle attacks

A man-in-the-middle attack is an active eavesdropping in which the adversary intercepts and reads the transmission between the two clients. It basically establishes individual connections between the users and make them believbe they are communicating directly with each other. The attacker can read all messages and inject new ones as well.

Known-ciphertext Attacks

The known-ciphertext attack (KCA) is an attack model for cryptanalysis where the attacker has only access to a set of ciphertexts. In practice it is possible to make guesses of plaintext, as the messages generally have fixed format headers. Most modern cryptosystems are strong enough agains ciphertext-only attacks.

Known-Plaintext Attacks

The known-plaintext attack (KPA) is an attack model for cryptanalysis where the attacker has a sample of both the plaintext i.e the unencyrpted text, and its encrypted version. The attacker can potentially use these to reveal secret information such as secret keys.

Chosen-Plaintext Attacks

A chosen-plaintext attack (CPA) is another attack model for cryptanalysis where that the attacker has the capability to choose arbitrary plaintexts to be encrypted and obtain the corresponding ciphertexts.

Chosen-Ciphertext Attacks

A chosen-ciphertext attack (CCA) is another attack model for cryptanalysis in which the attacker gathers information, at least in part, by choosing a ciphertext and obtaining its decryption under an unknown key.

In the attack, the attacker has a chance to enter one or more known encrypted texts into the system and return back the resulting plaintexts. Using this the attacker can obtain the secret key.

2.3 Enterprise Instant Messaging (EIM)

In a corporate environment when searching for an Instant Messaging service, companies have a wide range to choose from. Today's platforms that exist overall partly as a separate product are IBM Lotus Sametime, Microsoft Exchange Instant Messaging (Now known under the name Microsoft Lync), Oracle Beehive or something more extensive like Cisco Jabber and Salesforce Chatter.

There also exist as a vacuum variant of established consumer Messenger Services such as Qtalk Secure, which is about the known Messenger myENIGMA [50] originates. Another example is the BlackBerry Enterprise Messenger which focuses on business environments, even though BlackBerry Messenger exists for regular consumer as well. Previously, Instant Messaging platforms used proprietary technology concepts as a base and offered no inter-operatability. Today both IBM and Microsoft offer inter-operability between their EIM systems and some public IM networks.

So IBM Lotus Sametime offers a WebSphere-based Sametime Gateway. It allows a plug-in connection of the Third Party messengers with different Protocols such as SIP (Session Initiation Protocol) and XMPP (Extensible Messaging and Presence Protocol). On this based employees can chat with external Suppliers, whether this one Sametime client, Microsoft Lync, the AOL Instant Messenger, Yahoo messenger or even use Google Hangout.

If EIM systems are only operated within the company the risks to safety, security and privacy

are noticeably reduced. But if the services are outsourced in the cloud then these could be threatening.They are normally customized with a corporate design and are usually compatible with other products of the company. However, their biggest drawback is the lack of the ability of communicating with the outside world, one of the advantages of non-corporate IMs.

The EIM management server dominate substantial features such as content filtering, encryption, message archiving and are legally compliant. The employees use their IM clients and the IM network provides all the data including contact lists and internal and external colleagues can communicate freely in these conditions.

When considering Enterprise products we have to consider international server use and different compliance options. However, from various reasons not always easy to set. Considering a scenario of a small business, with a limited budget, the cost of initial installation, operation and maintenance is a significant investment.

For these cases cloud based products are offered, the initial price is comparatively less expensive to set up and no to little investment in server hardware or software is required.

In the selection of cloud-tools however, dealing with privacy aspects is mandatory. So it is a must that the selected tool complies to the country specific data protection requirements.

2.4 Related work in Secure Instant Messaging

With the massive number of users of instant messaging today, one server is not enough to handle all the client requests, so multiple servers are used to handle the client requests. But, even though there are multiple servers, conceptually, it can be thought of as a single server as they perform the same action, bound by the same rules and monitored/controlled by the same entity. The server is trusted. Hence, many SSL connections between the server and all clients are required. But SSL has a few flaws when used here, it gives a privacy problem. Each message is received by the server, this message is then decrypted at the server, and is encrypted again and sent to the recipient.

The problem here is the server can read all the messages to and from a person. This data can be mined and obtain private knowledge. This is a very big risk. An example where this is a risk

is Nielsen / NetRating Japan [28], is considering Instant Messages as a possible advertisement method, and offers a service that reports on Instant Messaging user populations and their attributives.

Sometimes the server may sometimes may store the unencrypted messages and allow them to be synced across all devices. A direct consequence of having unencrypted messages on server may cause leakage of personal information, or even identity theft.

There are many commercial Instant Messaging applications that claim to be secure. The most used secure way is use of SSL based encryption. SSL is used to provide confidentiality and security by establishing a secure connection between client and the server.

Digital certificates can also be used for identification. Although these provide a high level of security, these solutions may be are more expensive, for example a digital certificate must be bought from VeriSign for public domain users, and typically puts the entire responsibility of certificate distribution, verification, expiry, renewal,revocation etc. on end-users.

An issue with SSL is if a third party asks the certificate authority itself for a certificate of another website/person and the certificate authority is either coerced or compelled to do so, Then the entire concept of security using SSL breaks down. The server or the client have no way of knowing who can read the information sent between them.

Kikuchi et al [32] proposed a new protocol for group key management that is directed towards IM services. The protocol is an extension of Diffie-Hellman (DH) key agreement protocol.

It makes use of a distributed process in DH protocol. On starting the service, a user communicates with the server to calculate and get his DH public key defined by a combination of both the user and the server's secret. When a user begins a new conversation, he picks up a random number, on which a single message is computed and sent to the server.

Then, the server modifies the message so that a recipient can perform the DH key exchange protocol with the sender's public key every time a new on-line user comes. The advantage with this scheme is to reduce the cost of initiator and to leave the rest computation before a new user gets on-line. Unless the user's secret key is revealed to the server, the encrypted messages exchanged between members can't be seen by the server. In order for real-time

processing, a symmetric block cipher algorithm can be used for message encryption and the message encryption key is then encrypted with the key established by the DH protocol.

This being a strong protocol however gives rise to the issue of its compatibility with the existing consumer Instant Messaging service.

Xue Sun, Zhenjun Du and Rong Chen [33] proposed a hybrid encryption algorithm to secure the IM system using the AES, SHA-1, and RSA algorithms to implement a hybrid encryption policy. This was implemented over an Extensible Messaging and Presence Protocol (XMPP) based IM server and Java based clients.

The IM system uses the client-server architecture. The server is made up of two parts: A XMPP server and a backend database. The client connects to the server, then sends the user's requests to the server after the XML packaging program transforms the messages into the XML format defined by XMPP. The server analyzes the requests, saves it to the database, and then sends the message to the receiver that is specified by the sender. The mobile receiver client receives the XML message and calls the XML parsing program to transform the XML format to the data format that the mobile application could process, and finally displayed on the screen.

This system provides a method of security it does not seem viable as XMPP has disadvantages for example it does not support offline chats: if a user is offline, normal chat messages are dropped. Also since XMPP uses XML. Currently there is a trend to use JSON in place of XML. JSON is more compact and can be easily loaded in JavaScript. It seems superior in every way - it's flexible, faster and more compact and in many cases easier to use (especially when working with JavaScript).

Tsai-Yeh Tung et al. [34] proposed a method called Pandora messaging. This is presented as an enhanced Secure Instant Messaging architecture complete with a self destructing feature. It is also implemented over XMPP. When the transmitted messages constraints are satisfied, the ephemeral key used for encryption will be deleted. Thus, the encrypted messages become unrecoverable.

However this affects the ease of use and does not provide a proper method to appropriate validate the keys. Again as all messages pass through the server and the Ephemeral key server

is also controlled by the provider it can spoof keys which is a potential risk.

Chang-Ji Wang, Wen-Long Lin and Hai-Tao Lin [4] proposed an Instant Messaging System Using Identity Based Cryptosystems. In an identity-based cryptosystem, the public key of an entity can be easily computed from his identity information (e.g. e-mail address, IP address, IM account, etc.), and the corresponding private key is generated by a trusted third party named as private key generator (PKG), the private key is transferred from the PKG to the user through a secure channel. One issue that this has is that the Private keys pass through the server and the server could store them or read messages which is very undesirable.

A very important model used is off-the-record messaging. Its follows a real world scenario in which two people are talking without any hidden recording medium. The model is designed to achieve perfect forward secrecy [37] and repudiability.

> **Perfect forward secrecy** ensures that if an attacker was to compromise a users private key, they would only be able to access data in transit protected by that key and not any future transmitted data, as future data would not be associated with that compromised key.
> *Scott Helme, Information Security Consultant*

AT&T in the year 2001 proposed and implemented a protocol for off-the-record email to enhance security of emails [38]. **Borisov et al.** [13] proposed "off-the-record communication" in the purview of Instant Messaging. The proposed "Off-the-record messaging" protocol provides authentication and confidentiality of messages and claims that when an IM conversation is over, no one, including the communicating parties, can produce a record of the messages exchanged. Users publish their public keys, in basic terms their digital signatures, beforehand and this is also used to identify themselves.

An encryption key and a Message Authentication Code (MAC) key is established using the Diffie-Hellman (DH) key agreement protocol. Every message uses a different encryption key. A HMAC[39] key is used for authentication. Gaim now known as Pidgin [5] has a plug-in which implements this protocol. However, this has a few shortcomings: Long term signature keys

need to be maintained and casual users wont understand and might not maintain the keys and since the encryption key and MAC key negotiation is a continuous process and this will lead to a large overhead which is undesirable for a mobile client in which most clients use mobile data.

2.5 Security in current Instant Messaging Applications

Some applications use key based encryption to prevent the server from reading the messages, for example **WhatsApp** uses an RC4 key for encryption and a HMAC[39] key for authentication. However, as a Dutch Computer Science and Mathematics student Thijs Alkemade [29] at Utrecht University pointed out "...not only does WhatsApp use the same (RC4) encryption key for the messages in both directions, but also the same HMAC key to authenticate messages." However, MAC is not strong enough to detect all forms of tampering: an attacker could drop specific messages, swap them or even transmit them back to the sender. This is a major problem, considering that majority of the world as of now uses WhatsApp, he claims the users are not very safe, messages can be sniffed out and decrypted!

The **BlackBerry Messenger (BBM)**[40] uses the Session Initiation Protocol (SIP) for communication and is based on the cloud architecture. All communications are wrapped in a SSL / TLS communication protocol and contacts are added either with the help of a QR code. Few disadvantages include the contacts are not visible and does not offer any end-to-end encryption.

Blackberry released another version called BBM Protected [41] for enterprises which offers end-to-end encryption. BBM Protected introduces a new layer of encryption to the existing BBM security model.[41].

The keys are generated on the client device, by the FIPS 140-2 [43] certified cryptographic library, and are controlled by the enterprise. Each message uses a new random symmetric key for encryption. A Triple DES 168-bit BBM scrambling key encrypts messages on the sender's smartphone, and is used to authenticate and decrypt messages on the recipient's phone. All communications are wrapped within the SSL / TLS communication protocol. This is targeted strictly for enterprises but as of now is implemented only for Blackberry devices and lacks

critical support for Android or iOS.

Telegram [30] is another Instant Messaging application that claims to fix some of the flaws of WhatsApp. Telegram is a cloud-based and encrypted tool. It supports end-to-end encryption, self destructing messages and a multi-data center infrastructure. The architecture has a few servers all around the world routing messages. The MTProto [44] protocol is used in telegram. However the protocol used in telegram has a flaw. Encryption takes place between the client and the server, but not using TLS but using MTProto. Encryption takes place end to end between clients, but lacks authentication, the server is thus capable of performing a Man-in-the-middle (MITM) attack [31]. The threat model followed by Telegram is simple- Trust the server. Messages that go through the network are encrypted, however nothing is known of the server to server communication done, nor about their data storage system. But whatever goes through the server is available in clear.

ChatSecure [45] is another app offered by the Guardian Project. It makes use of XMPP and has a similar architecture of Facebook or Google's IM apps. It is based on the cloud and its own XMPP server. It offers an optional to use off-the-record messaging in addition to SSL / TLS. Thus it claims to have end-to-end encryption and because of use of different protocols an additional layer of security. However, the registration procedure is cumbersome and costly which affects the ease of use.

TextSecure [14] offers a wide set of features and uses the Curve25519, AES-256, and HMAC-SHA256 encryption algorithms. Messages are end-to-end encrypted and each user has a unique fingerprint, and these fingerprints are shared when a user is contacted by another user and a user will be warned if a user's fingerprint ever changes. Only if an identity is verifiable then communication is successful. Again this puts too much trust on one server.

Hoccer XO [36] is a German app which uses a custom communication protocol and is cloud based as well. All communications are wrapped in SSL / TLS and uses RSA as well. However it does not offer any local feature set.

Microsoft Lync [48], the successor from Microsoft Office Communicator, this uses the SIP, TLS, Secure Real-time Transport Protocol (SRTP) and HTTPS communication protocols. It has 3 varied architectures: Cloud or On-Premise or Hybrid. It is designed to work within

an organization. The use of SRTP basically provides encryption, message authentication and integrity, and replay protection to the real-time transport protocol (RTP) data in both unicast and multicast applications. A disadvantage with SRTP is that the protocol cannot setup encryption keys on its own [46]. Connections must be encrypted so that data cannot be intercepted by an eavesdropper, but this is complicated because both clients must agree on an encryption method before a secure communication can exist [47]. Also another possible lack of feature in Lync is that it does not offer end-to-end encryption.

Skype [49] has been a major player in the web conferencing domain. Targeted first for general users it is widely being used now in organizations as well for conferencing. It is cloud based and all communications are secure using SSL / TLS, however it is still more a desktop based application rather than a mobile based one and its IM features are looked over for its strong VoIP features. Added to the fact it offers no end-to-end encryption makes it a little weak in the domain of Secure IM.

MyEnigma [50] is a Secure Instant Messaging app that offers end-to-end encryption with the help of keys. It also wraps all its communications using 256-bit SSL / TLS. However, it again tells the user to trust the server and end users can never truly verify if the public key they have is actually the public key of the person they are communicating with and not one that is given by the server.

Surespot [51] also offers an end-to-end encryption and all communications are wrapped in SSL. However in a general scenario When a user is created and its public keys uploaded to the server, the server signs the public keys. Clients that download the public key then validate the signature of the key against the hardcoded server public key in the client. As with myEnigma the server can then change the key as it has control over it which is a big risk.

SIMSme [53] is a new Secure Instant Messenger by Deutsche Post AG. It offers an end-to-end encryption system. I also provides an additional layer of security by using the self-destruct feature. This causes messages to be automatically deleted from the recipient's device. are symmetrically encrypted on your phone using AES-256 and All messages transmitted to the recipient via the SIMSme servers using SSL encryption. Contacts can then be verified by using a QR code to confirm their identity. However, it does not offer forward secrecy.

Threema [52] basically as of now offers the best-in-practice method. It uses an end-to-end encryption and it uses a self developed communication protocol based on "Salt" which is the Networking and Cryptography library (NaCl). In addition it offers an optional encryption based on AES-256 on android devices. This allows the users to verify that the public key on the recipients device is really their public key by scanning a QR code of the keys fingerprint. This presents the fact the server can be trusted and is not sending spoofed keys. In addition to this it claims to offer forward secrecy by allowing the users to change their key pairs at periodic instances of time or at will.

3 Concept

The purpose of this chapter is to communicate the concept proposed by taking into consideration the challenges and problems presented in the previous chapter. A clear description of the requirements is given along with the proposed concept.

3.1 Initial Requirements

From the research done and results obtained in chapter 2, a Secure Instant Messaging app must:

- Not send messages in clear text form.
- Never log and save any information regarding any message or its contents or session or event on the server.
- Not rely on third party servers for message security and handling.

Thus conclusion that arose from the analysis is that a secure Instant Messaging app should provide at bare minimum the following features as described in chapter 2, taking into account all the above features provided by today's trending applications and the features and concepts proposed by the various knowledge exchanges on the topic it comes down to this:

- **Authentication**

There should be a proper way of verifying that only a valid user is logged in and using the service and there should not be any misuse. Most normal way would be with help of a username and password. This coupled with a way to verify the public keys on the device as well.

- **Confidentiality**
  Confidentiality is the ability to keep a message unreadable by anyone other than the intended recipient. This can be achieved through end-to-end encryption.

- **Forward secrecy**
  An attacker who has captured the network traffic from the device will not be able to decrypt the data even if he finds out the private key of the client.

- **Repudiability**
  The ability of the receiver of something to prove to a third party that the sender really did send the message. This can be done by signatures or signing the message.

- **Trustworthiness**
  The ability of the user to trust the server that their messages are not read and that their messages are not tampered with.

An application which offers these key features is said to follow the best practices in the industry right now. The aim is to use these findings as a base and apply them to the corporate scenario.

3.2 Detailed Requirements

From the current state of research and development into the domain of secure IM, considering all the positives and negatives of the systems in implemented and on paper.

3.2.1 Confidentiality

Message data passes between a client and the Wire Web service. All communication to and from the server is wrapped in SSL / TLS. The messages are kept in the database to provide the

multi-platform feature. Some of the data within the messages is considered to be sensitive in nature.

Even though the data stored on the database is encrypted, the server has access to it and can decrypt and read it. This is therefore a risk. Also, even though all communication on the server is encrypted there is a risk that an attacker can gain access to sensitive data, either by eavesdropping on the network or accessing the database.

End-to-End encryption is possibly the best solution to maintain confidentiality. The messages can and should be encrypted on the client devices so only clients can decrypt them and the server under no circumstances can decrypt them. This is paramount. So the next question that arises is which method of encryption should be used? We have three possible options:

- Symmetric Encryption
- Asymmetric Encryption
- PGP Encryption

Symmetric Encryption

> **Symmetric encryption** is the oldest and best-known technique. A secret key is applied to the text of a message to change the content in a particular way.
> *Microsoft*

A simple example of symmetric encryption might be achieved by just shifting each letter by a number of places in the alphabet. As long as both sender and recipient know the secret key, they can encrypt and decrypt all messages that use this key. In other words, the algorithms used for cryptography that use the same keys for both encryption of plaintext and decryption of ciphertext.

The main advantage of this method is its relatively fast because ecrypting and decrypting symmetric key data is relatively easy to do, giving a good performance.

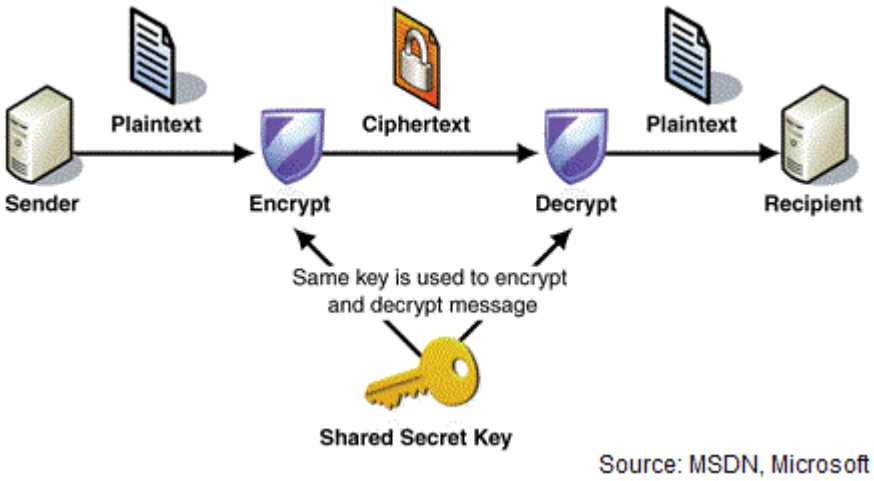


Figure 3.1: Symmetric Encryption

It is secure, but have been susceptible to known-plaintext attack and chosen plaintext attacks. Also because the key has to be shared a safe way must be used to give the key to the other party. Giving rise to more chance of stealing of the key.

Asymmetric Encryption

> **Asymmetric Encryption** has the main advantage that it is relatively fast because ecrypting and decrypting symmetric key data is relatively easy to do, giving a good performance.
> *Hitachi*

In asymmetric encryption there is a key pair. A public key is made freely available to anyone who might want to send you a message. A private key is kept secret. Any message that has to be to a user is first encrypted using the public key can only be decrypted by applying the same algorithm, but by using the matching private key. This means that public keys can be passed easily over the Internet as they are meant to be "public".

A problem with asymmetric encryption, however, is that it is slower than symmetric encryption. It requires far more processing power to both encrypt and decrypt the content of the message.

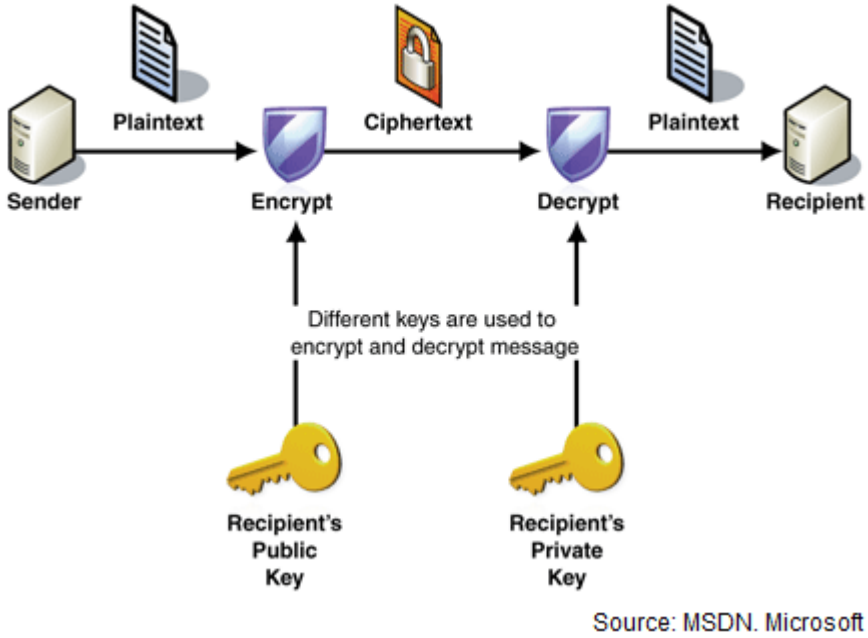


Figure 3.2: Asymmetric Encryption

PGP Based Encryption

PGP stands for Pretty Good Privacy. It is the most widely used software in the the world for the encryption of electronic mail.

It basically combines the advantages of asymmetric and symmetric encryption, downplaying the disadvantages of both.

It uses public key cryptography to let you communicate securely with people who've never met, without the prior exchange of keys over secure channels. [22]

A basic working can be described as follows: If user Alice wishes to send a message to Bob, PGP first takes plaintext and compresses it. This is done so it reduces the patterns found in languages. PGP creates a session key , this is done by taking a random number and is run through a symmetric encryption algorithm such as Triple DES, Twofish, CAST, or AES, this is basically a one time secret key. This key is symmetric so the encryption is very fast and secure to get a ciphertext. The session key is then encrypted to the receiver (Bob's) public key, using assymetric key encryption for example the Diffie-Hellman or RSA. The entire data is then transmitted to the recipient.

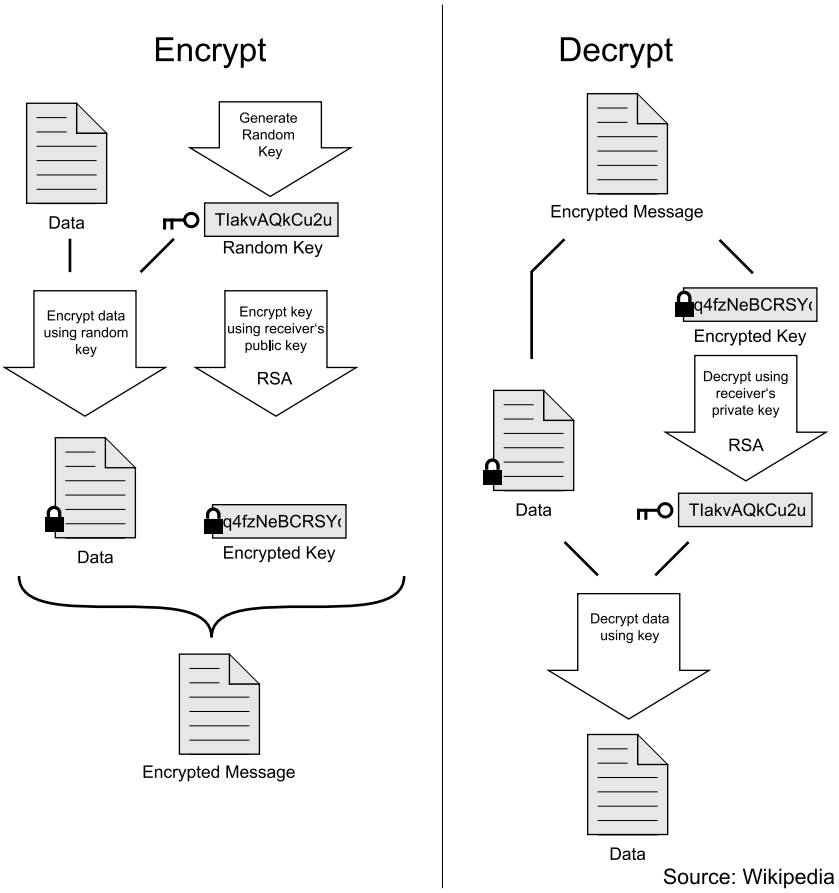


Figure 3.3: Working of PGP

When Bob receives this message, the reverse process takes place. The session key is obtained by decrypting using Bob's private key the session key is then used to decrypt the encrypted data to get the original message. [23]

Advantages of PGP

Some of the plus points of PGP include:

- PGP ties together the advantages of public key and symmetric cryptography.
- The combination of asymmetric and symmetric encryption methods combines the convenience of public-key encryption with the speed of conventional encryption and provides a feasible solution to the disadvantages of both.
- Using conventional symmetric encryption is about 100-1,000 times faster than public-

key encryption.

- Public-key encryption provides a solution to key distribution and data transmission issues when using symmetric encryption.
- Performance and key distribution are improved without any cut back to security.

Disadvantages of PGP

Though PGP has many advantages it has few flaws too. Some of them include:

- PGP is slightly complex and difficult to understand for a layman.
- Both parties should use the same version of PGP.

Thus, symmetric though faster requires sharing the key, and since a client has to share his key with the recipient and the key may get intercepted. This inadvertently becomes a risk.

Asymmetric encryption is powerful and the concept of public and private key is appealing, as the private key remains only on the client devices while the public key can be shared and used for encryption. The server will not be able to decrypt the messages on its end and only the intended clients will be able to decrypt the messages. However, it has efficiency issues. So the encryption method needed should have public and private key pairs and have relatively good efficiency, thus the solution should be PGP. Since PGP uses advantages of both symmetric and asymmetric encryption and cuts on the disadvantages, PGP is a good practice to follow and this seems to be the approach that was decided.

3.2.2 Authentication

Considering that we use PGP and make use the concept of public and private keys one possible risk scenario is in which the server cannot be trusted. For example, Alice wants to use the IM services and generates a key pair. Alice sends her public key to the server so other users who wish to communicate with her can get the public key from there. However, the server creates another key pair and says that the new public key is Alice's public key.

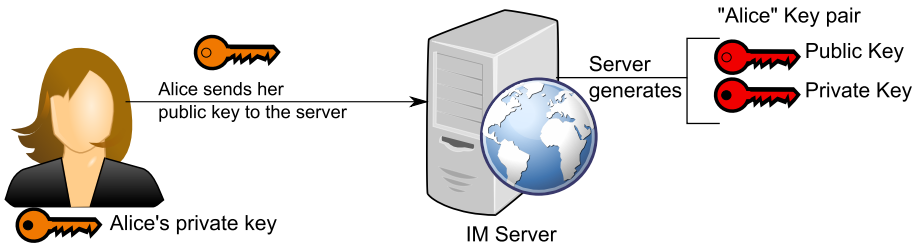


Figure 3.4: Server spoofs Alice's public key

Now of Bob wants to communicate with Alice, he requests Alice's public key from the server. The server sends the key it generated and not Alice's actual public key.

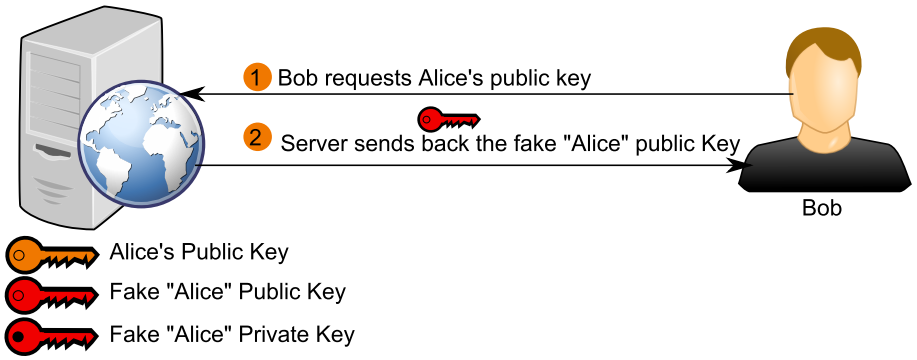


Figure 3.5: Server sends spoofed public key

Now Bob needs to send a message to Alice so he encrypts a message with this key and sends it to the server, the server gets this message, decrypts it and reads it and possibly saves it somewhere. Then it encrypts the message again with Alice's actual public key and sends it to Alice. Alice is able to decrypt the message and read it. However both Bob and Alice are clueless that the message was decrypted and read by the server! This is a form of man-in-the-middle attack. It is illustrated in figure 3.6.

So steps have to be taken to counter this. One of the methods is to make it possible for the users to verify their public keys. But public keys are massive collection of "junk" characters for the normal eye. They are long and even when ASCII armored are not very easy for humans to understand or compare. Thus here the concept of a fingerprints comes into play.

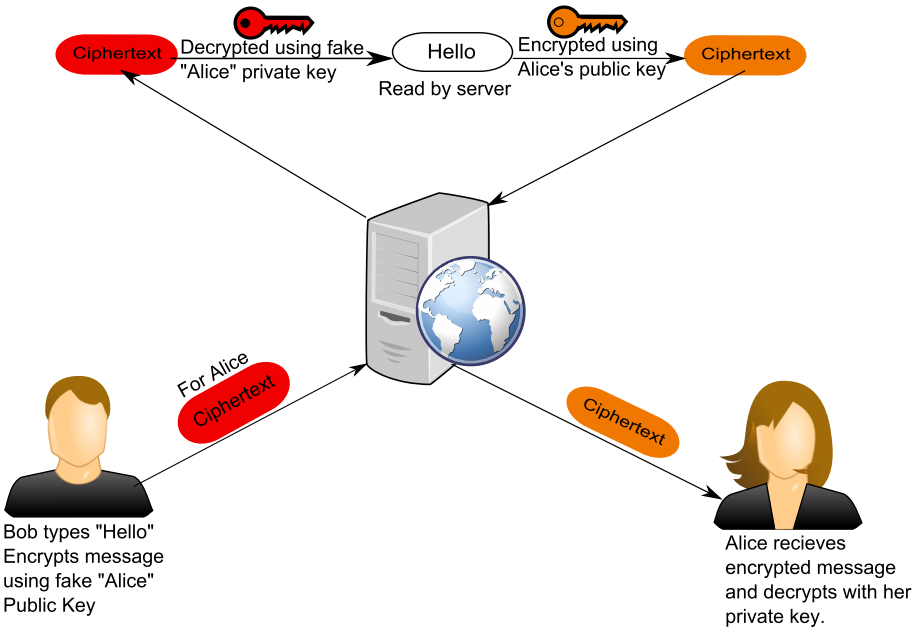


Figure 3.6: Man-in-the-middle attack

> A **fingerprint** is a shorter digest of the key formatted in a way that makes it easier for humans to read and compare.
>
> *Apache*

Thus allowing users to verify fingerprints will allow users to be sure the server is not spoofing the public keys and the public keys are indeed genuine and valid. This is an important key point that has to be considered.

3.2.3 Forward Secrecy

The concept of forward secrecy is that even if a user gets the private key of another user he should not be able to decrypt any older messages or any future messages. An attacker who has captured the network traffic will not be able to decrypt it even if he finds out the long-term secret key of the client or the server after the fact.

In the section 3.2.1 it was showed that PGP is suitable approach to follow for the Secure Instant Messaging system, but since it uses long-term public keys it makes it a risk. The use of SSL /

TLS makes sure that forward secrecy is maintained on the network connection but not on the end-to-end layer.

One solution to this problem is to not use long-term keys. New public keys can be generated at regular intervals or at the user's preference if he ever thinks his key is compromised. This is another point that has to be considered.

3.2.4 Repudiability

Once a message is sent by a user how do we know it is genuinely from that user? Here the concept of non-repudiation comes into the picture.

> **Non-repudiation** refers to a state of affairs where the purported maker of a statement will not be able to successfully challenge the validity of the statement or contract.[57]
> *Wikipedia*

So how can we achieve repudiability? The answer is through a digital signature or signing the message. The question is why not use a digital certificate. If a third party wants to send a message by spoofing another user, they can get a "valid" certificate from a certificate authority but this is spoofing the user. So digital certificates have a few flaws when trying to maintain confidentiality against the administrator.

For signing a message the concept of digital signatures comes into play.

Digital Signatures

> A **digital signature** (not to be confused with a digital certificate) is an electronic rather than a written signature that can be used by someone to authenticate the identity of the sender of a message or of the signer of a document.
> *Margaret Rouse, SearchSecurity, TechTarget*

Digital signature is the process by which a message is authenticated that is proving that a message is actually coming from the given sender. Digital signatures enable the recipient of information to verify the authenticity of the information's origin, and also verify that the information is intact. [13] For instance, suppose that Alice wants to digitally sign a message to Bob. To do so, she uses her private-key to sign the message.

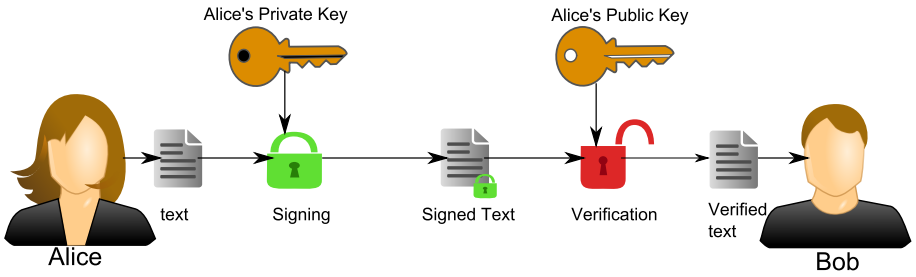


Figure 3.7: Working of Digital Signatures

Since Alice's public-key is the only key that can verify that message, a successful Digital Signature Verification can be performed, meaning that there is no doubt that it is Alice sent the message as her private key was used to sign the message.

A digital signature is used to sign the message and is part of the PGP key and this can then again be verified and validated. Thus, this coupled with a PGP key based encryption should provide another layer of authentication as well as satisfy the non repudiability.

3.2.5 Trustworthiness

Since one issue is trying to trust that the server will not tamper or spoof the users public key or "fake" the public key.

To achieve security against the server thus another precaution that can be taken to keep a users public keys secure is to de-centralize the servers. In other words keep the server which stores the public keys separate in other words an independent public key repository. Users can verify their digital fingerprints in real time and this adds another layer of security.

3.3 Secure IM Concept

In the earlier section we detailed the requirements that Secure IM system must provide and use this and the research done on the current technologies and propose a concept which can be applied to the current enterprise IM environment.

For confidentiality, PGP seems the best bet. Each and every user should have a key-pair. The public key is to be stored on a server so it can be accessed by other devices. Now the key server should be kept independent from the IM server. This is to add the additional layer of security.

So first considering authentication, each and every message sent is signed and verified by the recipient. This should provide a strong basis for authentication.

Now there has to be a way in which the public key must be verified by the user itself as genuine has his or her own. This can be done using fingerprints. But how can fingerprints be verified? Sending them across the same channel as IM messages can lead to manipulation or spoofing by the server which is undesirable, so it is better people meet and verify fingerprints or send them across another channel. Email seems to be a good way of sending the fingerprint which could work, but one problem could be that people don't like comparing long strings. For example a fingerprint looks like this

```
5A967C04A8F33C9D4F60145248B8AA50C7C384E2
```

Now comparing this manually can be hard, so an easier way to this is through a quick response (QR) code. A QR code can easily be scanned by a camera and compared by the app itself. This seems to be a good approach to follow.

Now for maintaining forward secrecy there should be an option to allow the user to regenerate his key pair and upload it to the key to the key server.

For maintaining authentication since we are using PGP, the public key has a encryption key and a signature key. Each and every message should be signed so it can be verified and thus checked and we can thus also provides non-repudiation, which means that it prevents the sender from claiming that they did not actually send the message.

4 Implementation

This chapter details on how the design principles were applied to the SAP Wire prototype. It details the libraries used and also details the challenges faced during implementation.

4.1 Application of concept

In the previous chapter a concept of the Secure IM system was proposed. Now the focus is to implement the concept. We first look at the Instant Messaging scenario at SAP SE and provide a detailed feature set to be applied to it.

4.2 Instant Messaging at SAP SE

SAP SE being a massive global organization spread across many countries world wide communication between different clients and employees is a must. For most office communications Microsoft Lync is used while for conferencing Cisco WebEx is used.

With the development of the HANA Cloud Platform, a new product of SAP that offers a Java Platform which you can use to develop applications "on the cloud", SAP worked on a project called Twaddle which later became SAP Wire and was released as an internal Instant Messaging app. One of its main features is to keep conversations in sync with all the devices.

The advantage of using Wire over Lync or WhatsApp is both WhatsApp and Lync cannot be

used on multiple devices. WhatsApp is a 3rd party app which is not permitted on SAP devices because all of your chats are not 100% secure. Lync for one does not provide persistent chats like Wire does. The target was to give Wire the best of WhatsApp and Lync and then some more.

Wire currently offers a web based desktop client and a web based mobile client, a stand alone JAVA based desktop client,an iOS client and an android app (in-development) allows the user to keep track of his conversation and requirements across all media.

4.3 Technology

4.3.1 The SAP HANA Cloud Platform

The SAP HANA Cloud Platform is the in-memory Platform-as-a-Service offering from SAP, enables customers and developers to build, extend, and run applications on SAP HANA in the cloud. [7]

Definition

SAP HANA Cloud Platform is an in-memory cloud platform based on open standards. It provides access to a feature-rich, easy-to-use development environment in the cloud. The platform includes a comprehensive set of services for integration, enterprise mobility, collaboration, and analytics.

SAP HANA Cloud Platform enables customers and partners to rapidly build, deploy, and manage cloud-based enterprise applications that complement and extend solutions, either on-premise or on-demand.

As a Platform-as-a-Service operated by SAP, it frees the end user from any infrastructure, costs and offers state-of-the art quality of service - availability, scalability, multitenancy.

Features

The HANA Cloud platform offers a lot of features:

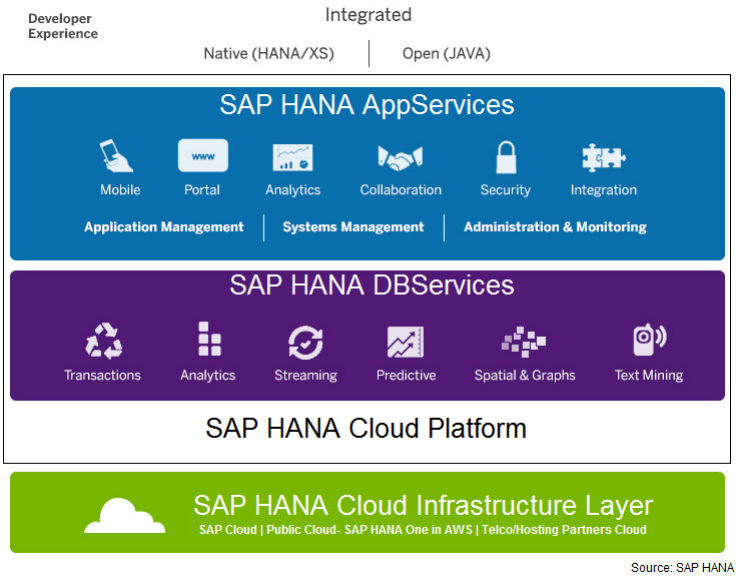


Figure 4.1: SAP HANA Cloud Platform

- **Runtime container for applications:** Applications developed on SAP HANA Cloud Platform run in a modular and lightweight runtime container. The platform provides a secure, scalable runtime environment with reusable platform services.

- **Integration with SAP and non-SAP software:** SAP HANA Cloud Platform facilitates secure integration with on-premise systems running software from SAP and other vendors. Using the platform services, such as the connectivity service, applications can establish secure connections to on-premise solutions, enabling integration scenarios with your cloud based applications.

- **In-memory persistence:** SAP HANA Cloud Platform includes persistence powered by SAP HANA. SAP HANA DB Services delivers fast provisioning of SAP HANA and hardware in the cloud. This means you can quickly build real-time analytic apps using SAP HANA development capabilities.

- **SAP HANA App Services:** SAP HANA App Services builds on SAP HANA DB Services, and allows you to create, deploy, and extend real-time consumer-grade apps in the cloud. Leverage comprehensive services for mobility, collaboration, Big Data, and more.

- **Secure data:** Comprehensive, multilevel security measures have been built into SAP HANA Cloud Platform. This security is engineered to protect your mission critical

business data and assets and to provide the necessary industry standard compliance certifications.

Services

SAP HANA Cloud Platform provides the following services:

Service	Description
Connectivity Service	The connectivity service provides a secure, reliable and easy-to-consume access to business systems, running either on-premise or in a cloud. SAP HANA Cloud provides a trusted channel to your business systems, while at the same time your IT has complete control and auditability of what is technically exposed to the on-demand world.
Persistence Service	The persistence service provides in-memory and relational persistence. All maintenance activities, such as data replication, backup and recovery, are handled by the platform.
Document Service	The Document service provides a content repository for unstructured or semi-structured content. Applications access it using the OASIS standard protocol Content Management Interoperability Services (CMIS). The applications consume the service using the provided client library.
Identity Service	The identity service is responsible for identity management and authentication on SAP HANA Cloud. It also enables Single Sign-On (SSO) between applications running on SAP HANA Cloud, based on the Security Assertion Markup Language (SAML) 2.0. It has already more than 2.5 million users which could access applications hosted on SAP HANA Cloud.
Feedback Service	The feedback service provides developers, customers, and partners with the option to collect end user feedback for their applications. The feedback service also delivers detailed text analysis of user sentiment (positive, negative, or neutral). The feedback service consists of a client API exposed through the HTTPS REST protocol and of administration and analysis user interface. The feedback service is a beta functionality and is available on the SAP HANA Cloud Platform trial landscape.

Table 4.1: SAP HANA Cloud Platform Features

Accounts

SAP HANA Cloud Platform provides free and paid accounts, the capability to create additional accounts on a self-service basis, and a member management feature for setting up teams.

Development

Developers can build highly scalable applications using one of the following programming models:

- **Java EE** - SAP HANA Cloud Platform is Java EE 6 Web Profile certified. Java EE applications can be developed just like for any application server. Also existing Java applications easily run on the platform.
- **SAP HANA** -The SAP HANA development tools can be used to create comprehensive analytical models and build applications with SAP HANA programmatic interfaces and integrated development environment.

4.3.2 The Android Operating System

> **Android** was built from the ground up with the explicit goal to be the first open, complete, and free platform created specifically for mobile devices.
> *Ableson F. et al., Unlocking Android, page 4.*

What is Android?

Android is an open system. Any handset manufacturer can use Android if they follow the agreement stated in the Software Development Kit (SDK). There are no restrictions or requirement for the handset manufacturer to share their extensions with anyone else, as there are in other open source software, if they leave the Linux kernel as is. The Linux kernel is under a different and more restricted license than Android. [17]

Android is a software environment and not a hardware platform, which includes an OS, built

on Linux kernel-based OS hosting the Dalvik virtual machine. The Dalvik virtual machine runs Android applications as instances of the virtual machine. Android contains a rich user interface, application framework, Java class libraries and multimedia support. Android also comes with built-in applications containing features such as short message service functionality (messaging), phone capabilities and an address book (contacts). [17]. Every Android application runs on its own instance of the Dalvik virtual machine.

Android Versions

The version history of the Android mobile operating system began with the release of the Android beta in November 2007. The first commercial version, Android 1.0, was released in September 2008. Android is under ongoing development by Google and the Open Handset Alliance (OHA), and has seen a number of updates to its base operating system since its initial release.[18] The current version is 4.4 KitKat.

Development

The Android SDK makes use of the Java programming language similar to Java Standard Edition (J2SE) called the Java Android library. The syntax is the same as Java in terms of operands, selections, iterations, file handling and more. The more specific Android classes and packages use other names that are not similar to Java editions, such as the Activity Class and the View Class.[1]

Android SDK

The Android SDK provides you the API libraries and developer tools necessary to build, test, and debug apps for Android.[6] The SDK is mandatory to android developers. It is a collection of all packages, application framework and classes libraries the developer needs to develop an android application.

Android studio

Android Studio (Beta) is a new Android development environment based on IntelliJ IDEA. It provides new features and improvements over Eclipse ADT and will be the official Android

IDE once it's ready.[1]

4.4 SAP Wire

According to Martin Lang, Manager at Global IT Mobile in charge of development of SAP Wire is "*SAP Wire is quite similar to WhatsApp, one of the most successful consumer chat apps ever with 300 million users. However it's built entirely with SAP technology and with the enterprise in mind. So in a way it's more than just a chat client, it's a platform, that doesn't just facilitate chats, but it can also connect to other apps and business processes to get a time sensitive question answered.*"

SAP Wire began as a thesis project called Twaddle developed by SAP Global IT Mobile Center of Excellence (CoE). It was designed to be a mobile chatapp that can be used to send short messages between two or more users via their mobile devices. The HANA Cloud is used as the back-end: It stores and provides all the messages, users and chatrooms. The application was initially only available for iPhones and iPads.

The project then built and redesigned into a internal product called SAP Wire. Its functionality was extended to serve desktop users as well and a web based client, a desktop client and the iOS client were re-designed and rolled out. An android app is developed but it is still in beta and is not yet released for general use.

This makes Wire multi-platform. This means one might start a chat on their iPhone, continue it on an iPad and then finish it on their desktop.

It is available on http://sapwire.hana.ondemand.com

4.4.1 Features

SAP Wire provides a wide feature set which includes:

- Send text and images, share your location on a map.
- **Group Chats**: Allows for Group conversations with multiple persons. Add or remove

group participants, change group subject and set a group icon.

- **Public Chats**: Discover and create public chats based on events, locations, topics inside SAP.
- **Persistent Chats**: Access to all chat history on any device.
- **Cloud Storage**: Ability to access all of ones Wire messages and media from multiple devices.
- **Secure**: All of ones chats are fully encrypted on the server side and stored on the HANA Cloud
- **Single-Sign-On**: With Single-Sign-On, a secure login service is provided without never have to worry about security even without having to remember any long passwords.
- **Always connected**: With push notifications, Wire is always ON and always connected.
- **Contact Synchronization**: Ability to find other Wire users automatically.
- Other features include the ability to set profile photo, set custom wallpaper, Email chat history and much more.

4.4.2 Functionality

The app has a login screen where you enter your ID and your password. This is linked to the users SCN account and uses SAP Single Sign On (SSO).

If a new user logs in for the first time, the system detects a new username and a user will be created on the database. When a user wishes to communicate with another user he searches and selects his / her (the recipient) contact and a chatroom is created with the two users as the participants. A chatroom can also have multiple participants or a group chat. All messages between the two users are stored in that chatroom. These chatrooms are available across all devices and all the messages are encrypted at the server and stored in a backend encrypted database.

4.4.3 Data Model

The data model used is quite simple to understand. There are of users. Many users are in many chat rooms and the other way round. A new message belongs to the user who sent it and to the chat room where it was posted. Thereby there is the possibility to get all messages of one chat room but also to know who sent it.

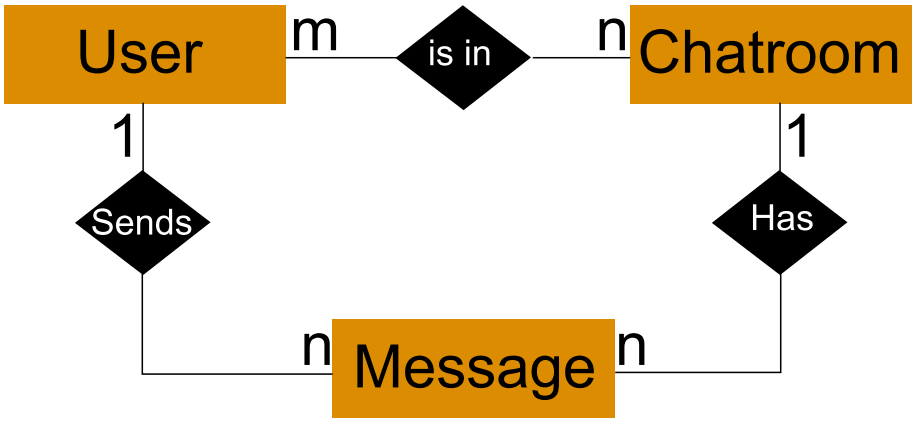


Figure 4.2: Data model of SAP Wire

4.4.4 Architecture

A high level overview of the architecture of SAP Wire is shown below. In the middle of the figure beneath you can see the SAP HANA Cloud with a servlet and a database. The devices on the left-hand side sends a message to our servlet. All communications are wrapped in SSL / TLS authenticated by the HTTP-Basic-Authentication or the SAP SAML 2.0 authentication.

When the message arrives at the server it is encrypted and stored in the database belonging to HANA Cloud in this case the MaxDB database. This database also stores the users, chat rooms and all messages.

The devices on the right-hand side asks the servlet for messages, the message is retrieved from the database and decrypted and sent to the recipient device which displays the new message on the screen. Also a push notification service is implemented so that the recipient is notified for a new message.

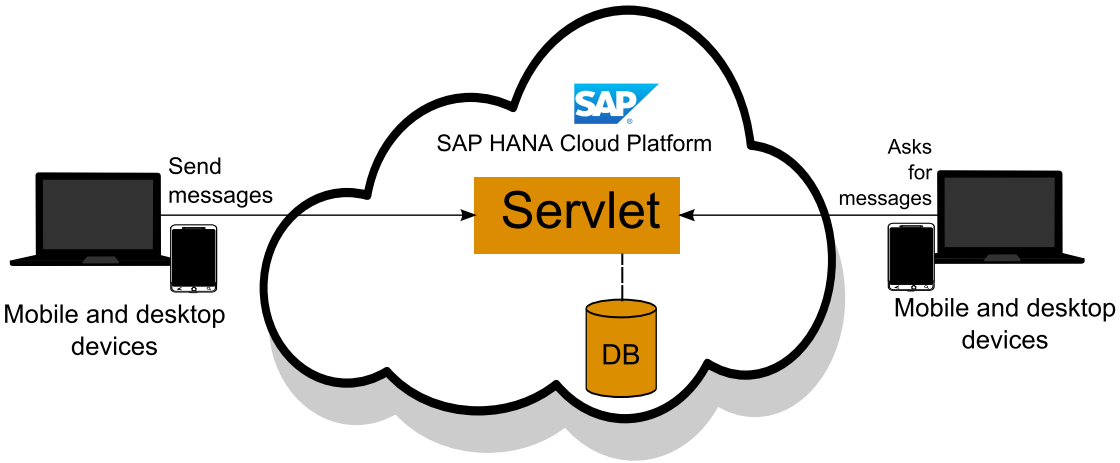


Figure 4.3: SAP Wire Architecture

4.4.5 Wire's Web Service

REST calls are used for retrieving and posting data to the server. The data is pushed into the database with the servlet. This servlet acts like a web service: the app can ask for or send data and gets a response. To exchange the data between app and servlet we decided to use JSON. The great advantage of JSON is that its overload is very low (especially compared to XML). The request-JSON is transported to the servlet via an HTTP-POST. The response-JSON is then returned to the app via the HTTP-response.

Every request-JSON includes a string parameter called "function". In this parameter you specify the task of the web service: e.g. return all users- `"getusers"` or return messages- `"getmessages"` for a given chat room. Also a function that creates resources, e.g. post a message- `"postmessage"` or create a chat room- `"createchatroom"`. Depending on the function there have to be other parameters in the request-JSON, for example when using `"getmessages"` one of the paramters should be the chat room ID. Nearly every function has to know for which user this function is called. This is achieved through the SAP SSO.

Depending on the requested function the other parameters of the response-JSON can include e.g. a list of users or messages in a chat room. The data storage on the server side is achieved using the Java Persistence API (JPA) [54] to save all the data on the backend database which is necessary for the web service.

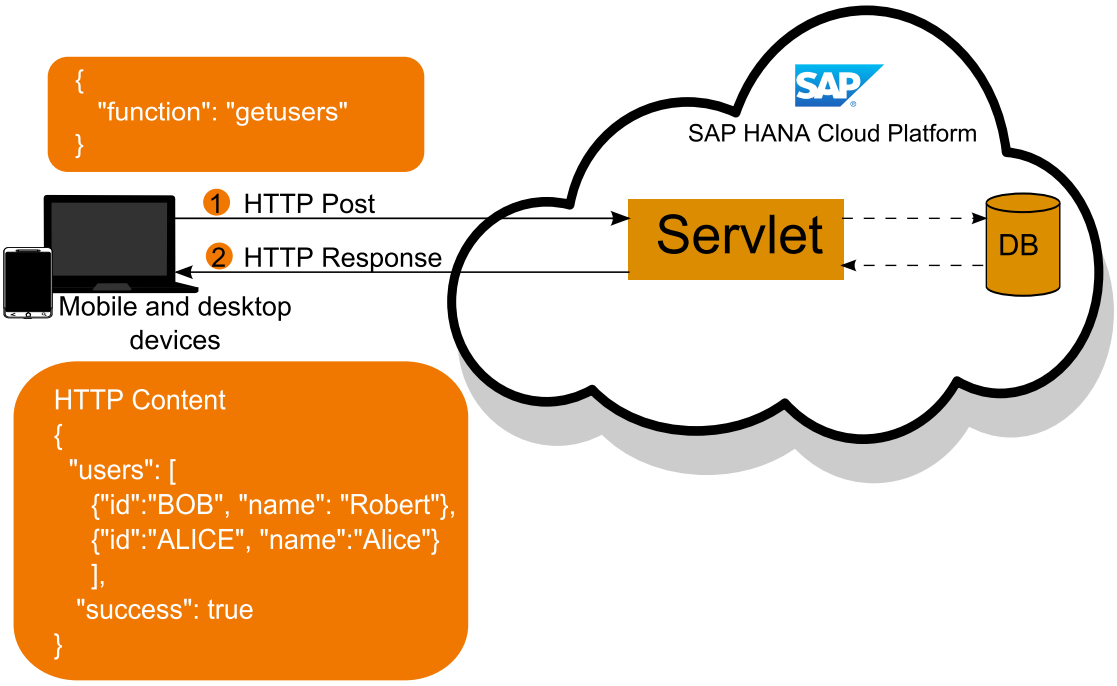


Figure 4.4: SAP Wire Web service working

4.4.6 The Web Based App

The SAP Wire app is built on SAP UI5 [55], a HTML5-based front-end framework from SAP which helps you to build desktop and mobile web apps. It is available on https://sapwire.hana.ondemand.com/desktop for the desktop web client and is available on mobile devices using the mobile web browser's https://sapwire.hana.ondemand.com/mobile. The Web App are communicates with HANA Cloud by using a RESTful webservice via HTTP. This webservice is based on a request-response-architecture. This is coupled with a HTML5 WebSocket through which a consistent connection is opened with the backend which stays open over the whole time the client is running.

After a WebSocket connection is opened the client can send requests to the server, and also the server can send requests to the client. Only one connection handshake per session is necessary. The client doesn't have to check for new messages repeatedly, it just waits until the server notifies when new messages have arrived.

Once a client sends a message, the backend forwards this messages to all corresponding clients which are currently online right away - in real-time.

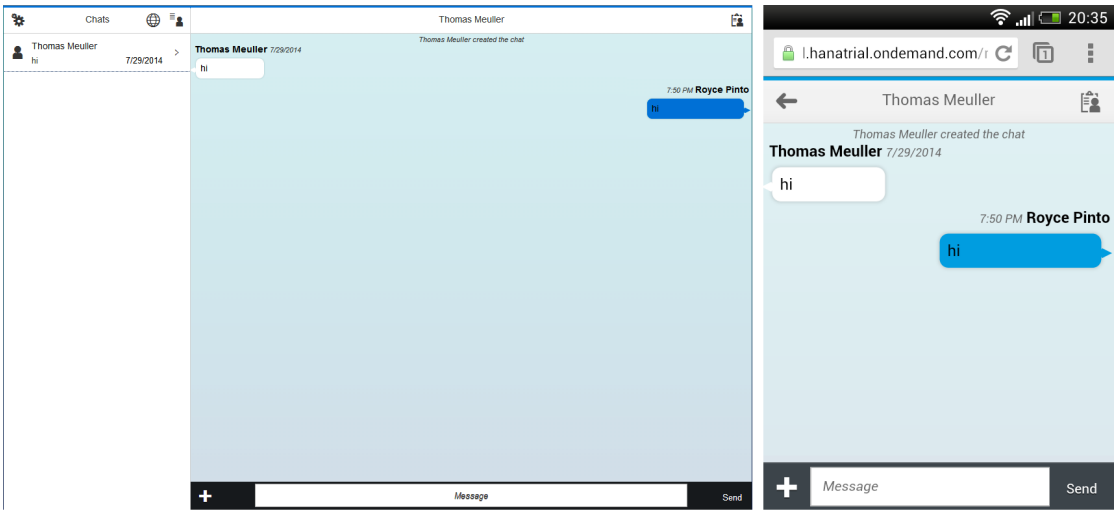


Figure 4.5: SAP Wire Desktop Web App and Mobile Web App

4.4.7 The Wire Android App and iOS App

A native android app is developed but not released for general use. It is still in beta and continuously worked upon and improved. The app makes use of the Wire REST calls to perform all actions.

The App is built using the SAP Mobile UI toolkit and theme so it matches the UI5 design and feel. The App interface is quite simple and is made easy to use. On starting the app a login screen is displayed. The user can login by manually using his SCN username and password or the Single Sign-On with a client side device certificate.

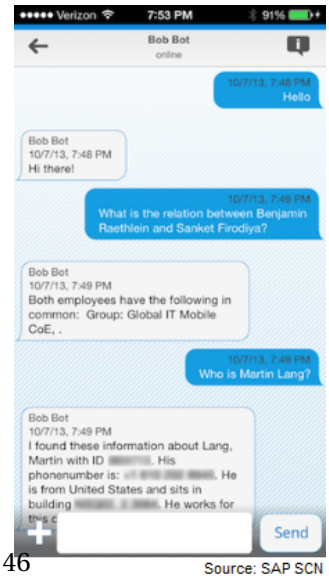


Figure 4.6: The iOS App

On logging in, if the user already exists his chatrooms, messages and settings are synced from the server. If a user logs in for the first time, then a user is created in the backend.

On opening a chatroom a user can continue an existing conversation or start a new conversation or group chat with other users. The app also allows the user to create a public group chat.

The working is very similar to the web based app as they both use the same REST calls and WebSocket connections. The app makes use of the content wrapper and content manager and

Google's account manager for maintaining and saving the SAP Wire account. This saves the user from logging in every time.

A native App is developed for Apple devices basically the iPhone and the iPad. The user is authenticated uses native SAML2 authentication. Initially when the app was still called Twaddle it used to use push notifications to notify the clients that a message had arrived on the client. But as the app was re-invented to SAP Wire, consideration was taken to the load the server will have and it was modified to work like the web app or the android app, WebSocket connections are used.

Figure 4.7: The Android App

SAP Wire is a good tool. Developed in-house it is continuously worked upon. It has a decent and ever increasing user base at SAP. The android app seems to be an ideal start point for development of a Secure Instant Messaging app as being still in beta its feature set can continuously being expanded upon.

Since Wire is based on the HANA cloud its security mechanisms include SSL / TLS and the messages are stored on the database are encrypted.

So for adding the Secure IM concept and its security feature set the, android app seems a great place to start.

4.5 Features to be added

First a server must be created to hold the public keys. This is called the key repository. The server has a database with a simple table having one table with the field for ID (E-mail) and public key. Along with this there should be REST methods to retrieve the public key based on ID, another to post the public key and finally one which retrieves all public keys.

Since PGP has to be added to the IM system, when the app is started it should check if keys are present, if not it should generate a key pair. Once this is done, the public key must be uploaded to the key repository server.

Before each message is sent to the server to be delivered to another user, the client must first request the recipients public key from the server, the second step is the message is signed by the user and is encrypted with the public key of the recipient and it is then delivered to the recipients device. The recipient then decrypts the message with his own private key and verifies the message. If all is valid the message should be shown or else dropped.

In addition to this the ability to check public key fingerprints is to be added:

1. A QR code generator to convert the fingerprint to a QR Code
2. Ability to scan and verify QR codes
3. Ability to send fingerprint through e-mail

These design features are in sync with the requirements and should satisfy the best practice used in Secure IM in a corporate environment.

4.5.1 Assumptions

A few assumptions are made:

- The best features available on the android app have been picked up and added to the basic prototype of this application
- Libraries are used to implement the PGP based encryption and decryption
- The new features implemented do not break any existing functionality is designed in such a way that the security features can be deactivated or activated as required

4.5.2 Target Audience

The target audience of this application are SAP Employees. The app is now just released for internal use and is in its pilot phase. With the growing need of privacy the app will be productive for private conversations and add another additional level of security for office conversations.

4.6 Use Cases

4.6.1 Actors

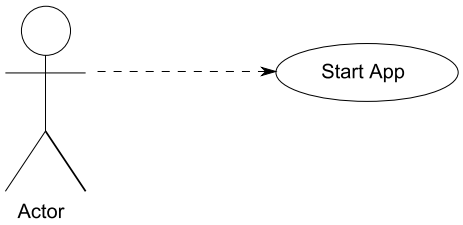


Figure 4.8: Actor and a use case

An actors is a user of the system. It includes humans as a process. He is the one who initiates the event that leads the use case to begin. The arrow also is an indicator that the interaction starts between the user and the use case. The scope of their action and their overall role in the system defines the set of Use Cases an actor can access.

4.6.2 Use Case Models

A use case is a single unit of meaningful work. It is built with a specific functionality for the proposed system. The keyword UseCase can also be used to define a use case. The actors are mainly related to the usecases. Through them the progress of the application could be well known. It describes the basic functionality in a layman's understanding. It also shows us the requirements of the applications and the constraints associated with it as to what are the functionality a user can access.

There are 4 use cases considered:

- UseCase for Starting the App
- UseCase for Sending a Message
- UseCase for Receiving a Message
- UseCase for Verifying a Public Key

UseCase for Starting the App

The following UseCase shows the work-flow of the procedure of starting the app and invariably generating the key-pairs.

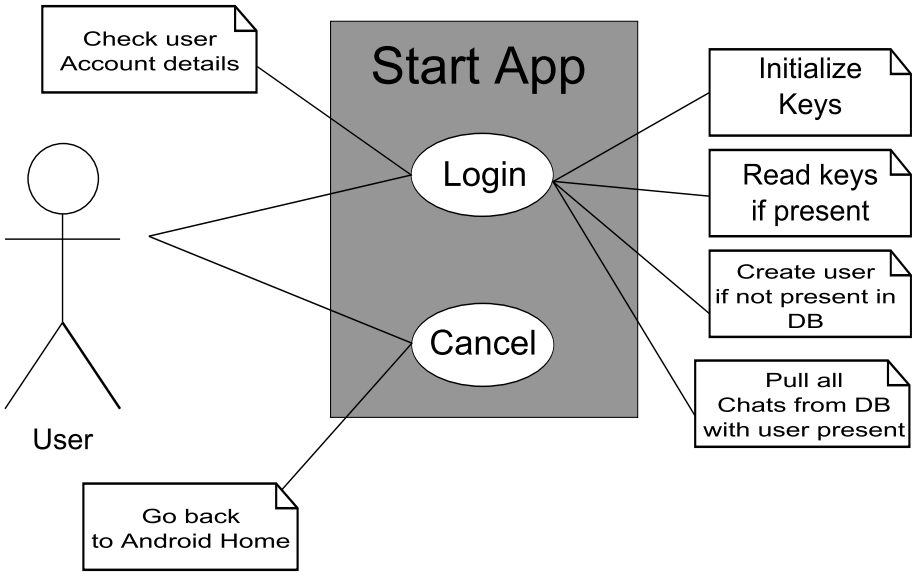


Figure 4.9: UseCase for Starting the App

UseCase for Sending a Message

The following UseCase shows the work-flow of the procedure of sending messages to a recipient.

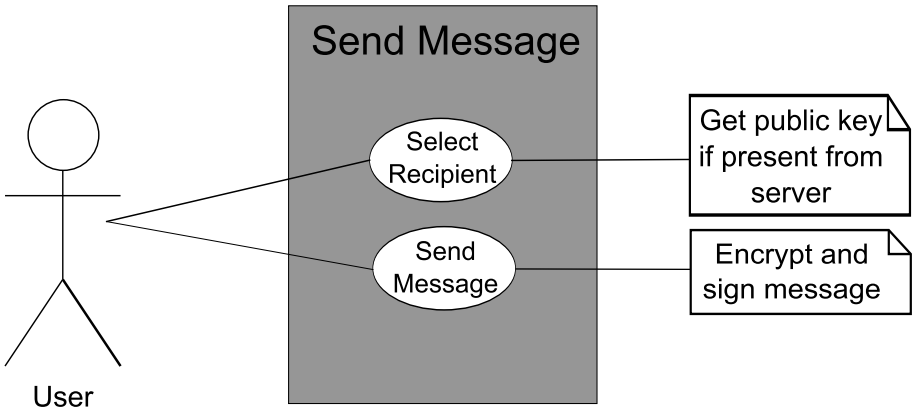


Figure 4.10: UseCase for Sending a Message

UseCase for Receiving a Message

The following UseCase shows the work-flow of the procedure of receiving messages from a sender.

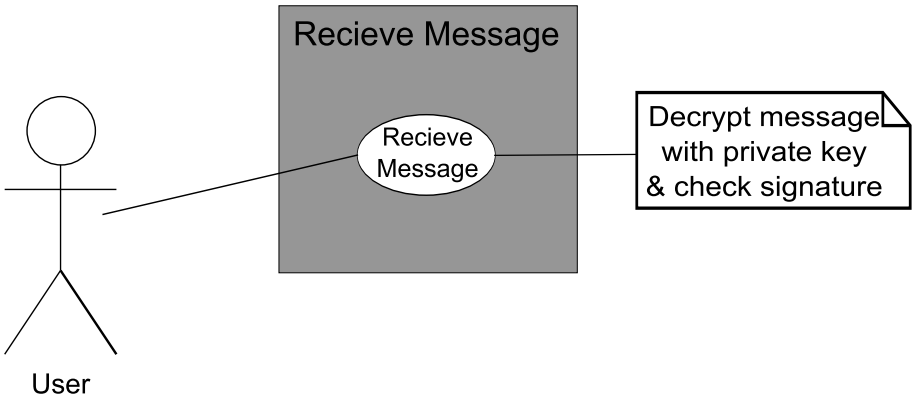


Figure 4.11: UseCase for Receiving a Message

UseCase for Verifying a Public Key

The following UseCase shows the work-flow of the procedure of verifying the public key of a user on a recipient's device.

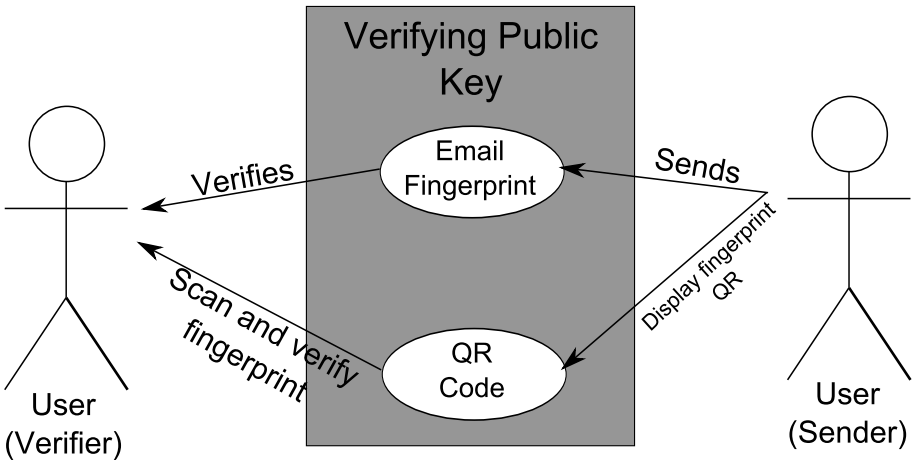


Figure 4.12: UseCase for Verify Public Key

4.7 Programming languages and tools

Since the product being enhanced is SAP Wire which is based on the HANA Cloud and also since the client device is an android app the programming language used is JAVA. For all applications developed on the HANA Cloud the official IDE supported is Eclipse with the HANA SDK Toolkit. For the android app the IDE used is IntelliJ because of its powerful visualization tools and official Android support.

The devices used for development are:

1. HTC One S, running Android version 4.2 (Jelly Bean)
2. Samsung Galaxy S2, running Android version 4.1 (Ice Cream Sandwich)
3. Google Nexus 7, 2012, running Android version 4.4 (KitKat)

The IM service and the key repository developed are hosted on the SAP HANA Cloud Platform Trial account available on http://hanatrial.ondemand.com. Also the local run-time of the SAP HANA Cloud platform is used for local testing.

4.8 PGP Encryption and Decryption

Since PGP has to be implemented and is a core part of the development, the first step is to find a library that provides PGP encryption and decryption features. For this purpose we use Bouncy Castle[58].

Bouncy Castle is a collection of APIs used in cryptography. It includes APIs for both the Java and the C# programming languages.

According to the Bouncy castle website, The Bouncy Castle APIs currently consist of the following[58]

- A lightweight cryptography API for Java and C#.
- A provider for the Java Cryptography Extension and the Java Cryptography Architecture.
- A clean room implementation of the JCE 1.2.1.

- A library for reading and writing encoded ASN.1 objects.
- Lightweight APIs for TLS (RFC 2246, RFC 4346) and DTLS (RFC 4347).
- Generators for Version 1 and Version 3 X.509 certificates, Version 2 CRLs, and PKCS12 files.
- Generators for Version 2 X.509 attribute certificates.
- Generators/Processors for S/MIME and CMS (PKCS7/RFC 3852).
- Generators/Processors for OCSP (RFC 2560).
- Generators/Processors for TSP (RFC 3161 & RFC 5544).
- Generators/Processors for CMP and CRMF (RFC 4210 & RFC 4211).
- Generators/Processors for OpenPGP (RFC 4880).
- Generators/Processors for Extended Access Control (EAC).
- Generators/Processors for Data Validation and Certification Server (DVCS) - RFC 3029.
- A signed jar version suitable for JDK 1.4-1.7 and the Sun JCE.

It is distributed under the MIT License and is used as a base for implementing PGP based encryption and decryption.

The implementation is explained more in detail in the following sections.

4.9 SAP Wire Key Repository

The SAP Wire server as described in the earlier chapter maintains all the users and the respective chat rooms and other details, but does not contain a place to store key data. So a simple key repository was created on the HANA Cloud platform. The persistence service is used for data storage. The persistence service makes available both in-memory and relational database storage to applications that are hosted on SAP HANA Cloud Platform. The database used is SAP HANA. Since all communication to and from web applications hosted on the HANA Cloud are wrapped in SSL/TLS the communication is considered secure.

For simplicity reasons JDBC is used. A table Person is used with the following fields: ID, PUBLICKEY of the type Varchar and having the sizes 255, 2555 characters respectively.

Three classes are created for this purpose as described below.

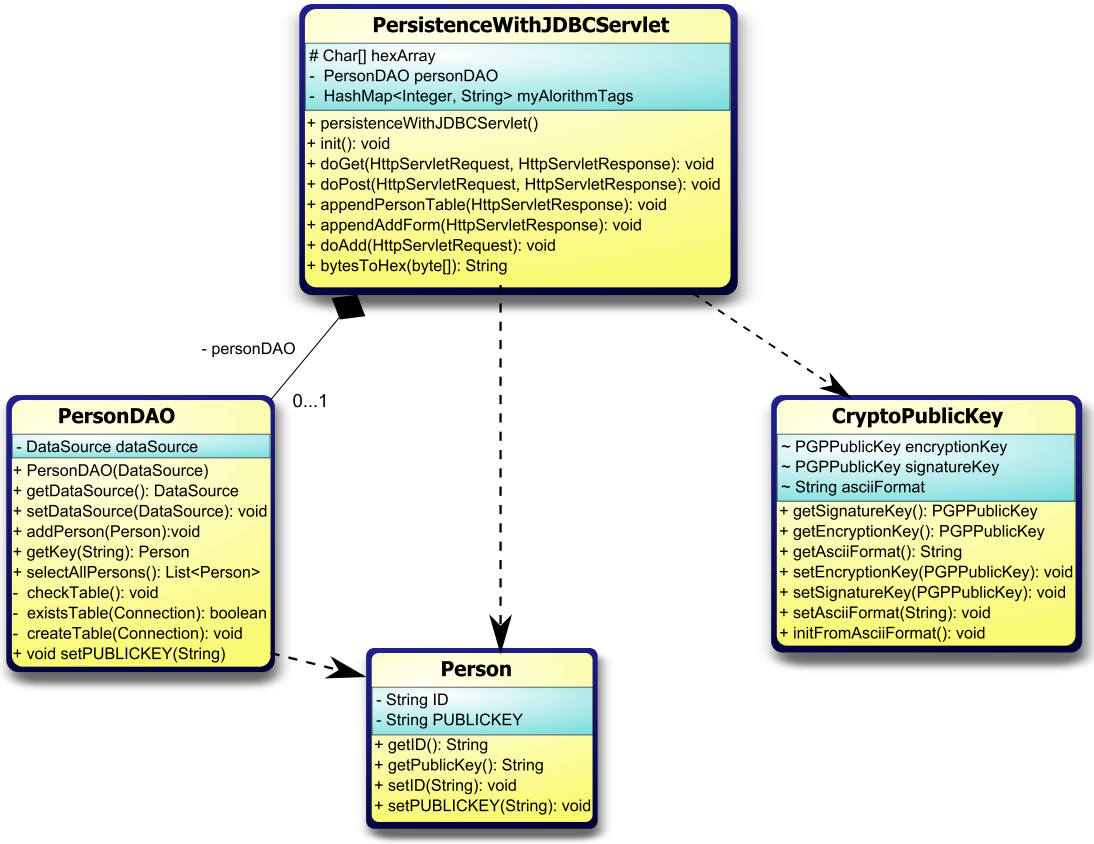


Figure 4.13: SAP Wire Key Repository UML

4.9.1 Person

The person class is the class file with two fields ID and PUBLICKEY of the type string. The other methods this class has are accessors and modifiers for ID and PUBLICKEY.

4.9.2 PersonDAO

This is the person data access object class. It contains the SQL queries which are needed to access the data from the database. It has methods which create the table PERSONS, one which checks if the table PERSONS exists, a select all persons method which selects all the data from the table, a method called get key which retrieves the key based on the ID and also a method to add a person to the database.

4.9.3 PersistenceWithJDBCServlet

This class is the servlet class and it extends the HttpServlet. It has a GET method implemented which returns all the entries if an email is not passed as a parameter or it sends the public key of the ID passed.

Also a POST method is implemented which will add the a person (email and public key) to the database. The servlet checks if an entry already exists, if it does then the value is updated or a new entry is created in the database. The servlet makes use of an SAP UI5 to display all the entries. But its not feasible to show the entire public key, so we make use of the bouncy castle library to read the public key. Just the signature key and few of its characteristics are displayed. The fields include: Email, Public key ID, Algorithm used (e.g. DSA, RSA, Elgamal, ECDSA etc.), Key size in bytes, Creation date and time of the key and the Public key fingerprint.

4.10 New Functionality

Confidentiality, Repudiability and Forward Secrecy

For implementing confidentiality, we make use of encryption and decryption of messages. For this purpose we make use of the Bouncy Castle libraries. Now the Android system comes with the bouncy castle in-built. But it is a stripped down version of Bouncy Castle and using the APIs might be a little complicated and it might not have the functionality desired. Also it makes installing an updated version of the libraries difficult due to classloader conflicts[59].

Therefore we have make use of Spongy Castle [59]. Due to class name conflicts, this prevents Android applications from including and using the official release of Bouncy Castle as-is. Spongy Castle is the same stock Bouncy Castle libraries repackaged with a couple of small changes to make it work with Android apps. According to the Spongy Castle website, the changes made are:

- All package names have been moved from **org.bouncycastle.*** to **org.spongycastle.*** - to avoid classloader conflicts
- The Java Security API Provider name is now **SC** rather than **BC**
- No class names change, so the BouncyCastleProvider class remains Bouncy, not Spongy,

but moves to the **org.spongycastle.jce.provider** package.

We make use of 4 libraries for this purpose:

- **core (jar)** - Core lightweight API
- **prov (jar)** - JCE provider (requires core)
- **pkix (jar)** - PKIX, CMS, EAC, TSP, PKCS, OCSP, CMP, and CRMF APIs (requires prov)
- **pg (jar)** - OpenPGP API (requires prov)

Authentication

For authentication, SAP Wire already provides a good authentication using SAML2 and SAP SSO. This allows users to sign in using their SCN Account details or their SAP certificate. The account is linked to the Android accounts interface.

Now for verification of public keys, it was proposed to verify digital signatures. Now since we need to verify public keys the two methods used are sending the fingerprint via e-mail or through scanning the fingerprint as a QR code.

For the purpose of generating and scanning the QR code the ZXing ("zebra crossing")[60] library. According to the website, "The ZXing ("zebra crossing") is an open-source, multi-format 1D/2D barcode image processing library implemented in Java."

The provide the API to generate bar codes and also the ability to scan and read bar codes using their Barcode Scanner app which is available on Google Play.[61]

4.11 Basic Cryptography Prototype

To start implementing the app, first it is a pre requisite to understand the working of the Spongy Castle API.

To achieve this it was decided to allow to run the basic PGP encryption/decryption process on a test string. Thus, to begin with an app was created with a limited UI which just implements a basic function: It generates a key pair on start-up and based on user input string, the app

will first sign the data with the public key generated and encrypt the data. Then it will decrypt the data and verify the signature and the decrypted string is compared with the input string. The match proves that the process is working.

This implementation can therefore be easily ported to the wire app as it can run in isolation.

4.12 The Wire Android App

The Wire Android app is built using native android code. Now since the existing functionality must not break, the new features must be added in such a way they stand independent from the app. So it would be like addition of a layer of security to the app.

An overall activity diagram of the entire process is shown below.

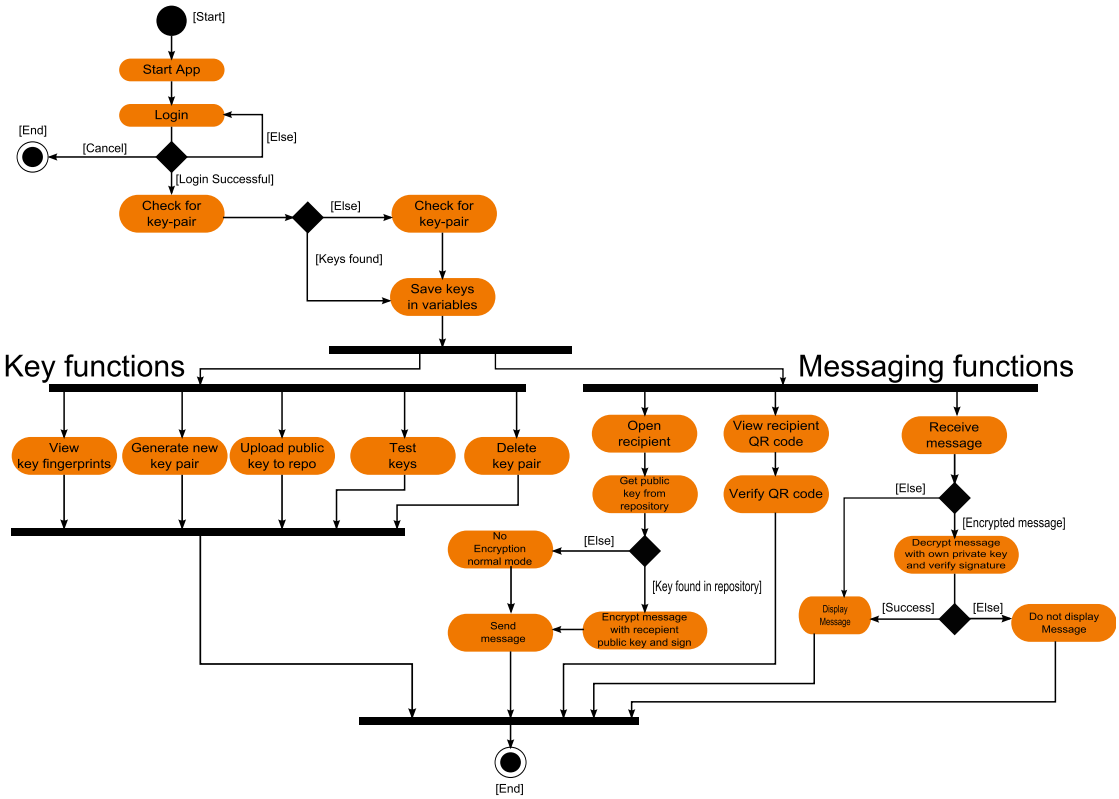


Figure 4.14: SAP Wire Activity Diagram

Since work has to be done on SAP Wire's Android App, it makes no sense trying to make changes to the Wire API and the Cloud application. But since the Wire application is hosted for public use on the official SAP HANA Cloud platform it makes no point trying to modify it

there. So another instance of SAP Wire was setup on the trial HANA Cloud Platform to act as a server for the android app. The Wire app is hosted with a MaxDB database used to the store data.

Any changes to be made to the API because of the app will be revisited later on.

4.12.1 Class Descriptions

To implement the criteria, the package com.sap.encryption is created and in it eight classes are created and used, which are:

1. DataHolder
2. CryptoEncryptedContent
3. CryptoUtils
4. CryptoPGP
5. CryptoPublicKey
6. CryptoSignedContent
7. EncDencMessage
8. NetEncSignMessage

What follows is a UML Relationship diagram between the classes followed by a detailed description of the classes. All classes are in the package **com.sap.wire.encryption**.

- **DataHolder.java**

  The DataHolder class is used to hold public and static variables and data that is shared between the classes. It is done to organize the data in a proper readable manner.

  The variables it contains are the directory in which the key files are stored, the PGP Private Key, the PGP Public key and a PGP Private Encryption Key. It has a method to read a file and save it as a string, also another method which saves an input stream as a string.

  The DataHolder class also has methods to test the key function methods.

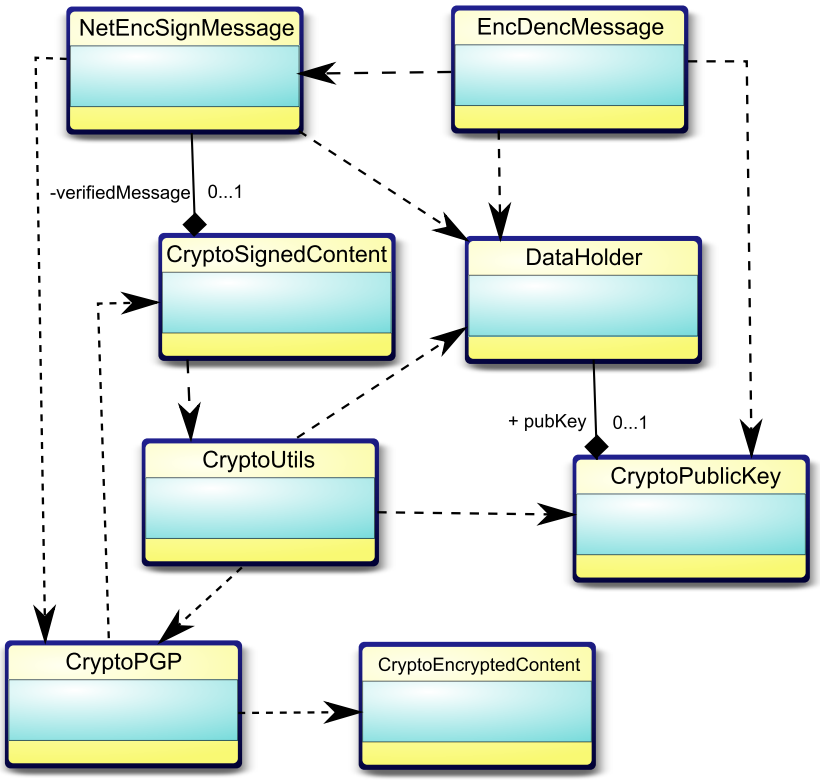


Figure 4.15: Wire Android Encryption Class Relationship

- **CryptoUtils.java**

  This class is used to initialize or read keys if they are already present. It creates 2 files: a private key file and a public key file with the file extension .asc. If the keys are not found in the app files directory, it can generate a PGP key pair and write the keys which are ASCII armored to files in the app files directory.

  The private key will be written to file 'secret.asc' and the Public key will be written to file 'public.asc'. If the keys exist, they are read and verified if their user IDs in the key match the client's hashId. The ElGamal key is used for encryption while the DSA key is used for signing. There is also a method to delete the keys.

- **CryptoPGP.java**

  The CryptoPGP class is taken and modified from https://subversivebytes.wordpress.com/category/cryptography-2/. This class has the technical implementation of signing and encrypting bytes based on the Spongy Castle libraries. This class is where the actual implementation takes place. Strings and data

files are converted to a byte stream and then encrypted with the PGP public key ring and the encrypted byte stream can be decrypted and verified with the PGP private key.

- **CryptoPublicKey.java**

  This class is a wrapper class for PGPPublicKeys. The DSA key that can be used verify a signature of the partner and ElGamal key can be used to encrypt data to the partner. The CryptoPublicKey can either be initialized from a file, or from a String containing the ASCIIarmored public key ring.

- **CryptoSignedContent.java**

  This class is a wrapper class for processing raw encrypted messages. After the raw message and public key have been set in constructor, it can decoded and verified through its signature against the public key. This is used for comparing signature against public key if the message has been signed by the owner of the public key. A valid signature will result in the field valid of the class has been set to true.

- **NetEncSignMessage.java**

  This class is a wrapper class for performing the main actions of encryption and a wrapper for decryption. Sign and then encrypt content of raw data string with set keys. After operation, the signed and encrypted content is available in encrypted data. It also has a wrapper method for decrypting and verifying messages.

- **EncDencMessage.java**

  This class is the re-usable wrapper class which has to be called for encryption and decryption. The encryption method takes in a string as input, then using the NetEncSignMessage class method, first the sender private signing key and passphrase is set. The reciever encryption key and passphrase are set. Then the message is signed and encrypted. For decryption, first the recipient private key and passphrase are set and the sender's signing key is obtained from the public key of the sender. Then the decryption and verification method is called. If any error, then null is returned.

4.12.2 Connecting new functionality to the app's existing working

Now the task is linking the new functionality to the app's existing working framework.

First pre-requisite of the app is the key-pair. So the key initialization/read method is added to the section when the app starts or resumes. This allows for the app to always have a key-pair at the ready.

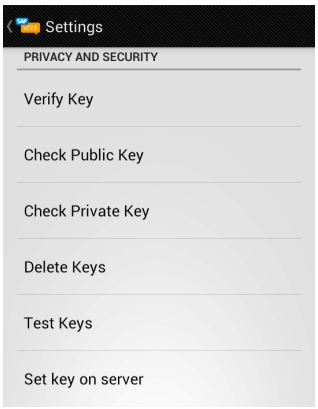


Figure 4.16: The app preferences menu

In the preferences menu, a set of key functions are added to allow for working with the keys. The ability to view key fingerprints, delete keys, test the basic working of the keys- this is done by taking a fixed string, signing it, encrypting it with ones own public key and decrypting it again with ones own private key and verifying the signature to make sure the keys are working fine. Also options include ability to delete keys and upload ones public key to the key repository.

Now for sending an encrypted message, the sender must have the recipient's public key. So a REST call is made. When a user selects another user to chat with, the app makes a rest call to the key repository passing the email ID of the recipient. The repository checks if the public key of the recipient is present and it returns it. The key is returned in form of an ASCII string. The key returned is then taken and read.

From this string, the encryption key portion is extracted. This will be used for encrypting the text. The key is stored in a temporary variable and not on the device. So every single time a chat is opened the key is pulled from the database. When the key is successfully returned from the repository a toast notification is shown notifying the user if his chat will be encrypted or not. This keeps in line with the existing functionality that even if a user doesn't have a public key he can still have a chat with a user.

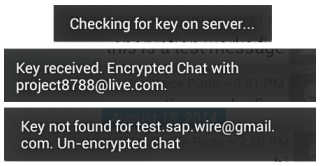


Figure 4.17: Toast notifications displayed by the app

Now is the process of adding the encryption method at the right point. The right point is just before the REST request is made. There is a method in the program which basically encodes the URL before making the REST call. Just before the URL is encoded, the message is encrypted and signed.

The second step of the process is when a user receives a message. The app saves all messages on the device's database. The point where the decryption method is called just before the message is stored in the local database of the android device. The message is first checked if it is encrypted, if not, it is displayed as it is, or else it is decrypted using the private key saved on the device. The decrypted message is then verified. If it is valid, then it is displayed. If there is any error in the decryption then the message is not displayed.

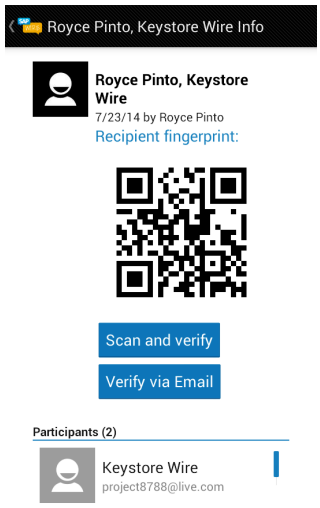


Figure 4.18: App fingerprint verification screen

Now since Wire is multi-platform, the messages can be accessed on the web based app as well. But that is out of the scope of this thesis. But if the messages are viewed on the web app, the encrypted message is show as there is no way to decrypt it on the browser. But this adds as a proof of concept that the server cannot decrypt the end-users messages.

Now since the public key of the recipient is pulled from the key repository when a recipient is selected, so the verification section is added to the chat information display screen of the app. If the key is received successfully then the QR code is displayed and the option to send the code by email and also an option to scan ones own public key on another device automatically.

4.13 Challenges Faced

During the development cycle of this product a number of challenges and roadblocks were faced.

The challenges were more to do with limitations of the libraries used (Spongy Castle and Zxing) and the understanding of the clear working of SAP Wire and the android app.

What follows is a brief description of the challenges faced and how they were overcome.

4.13.1 Message size in the SAP Wire Database

One of the earliest challenges faced was after a message is encrypted, it is sent to the server and stored in the database.

In the "stock" version of SAP Wire there is a limit on the size of the message placed is 900 characters, so any message greater than 900 characters will be truncated to 900 characters. Now this is an issue as an encrypted message can sometimes go over 900 characters and because of this on the recipients device the entire message would not be received and hence could not be decrypted.

For overcoming this flaw, a change was made to the SAP Wire application. Since the application uses JPA for persistence on the database, it is designed to hold up to 2500 characters, so the statement which used to truncate the message length is removed. This allows the entire encrypted message to be saved in the database.

4.13.2 Running of Encryption/Decryption as a background task

The encryption takes place just before the REST call is made to send it to the Wire server and the messages are decrypted when they are received from the server.

On android all network tasks have to be run in the background and cannot run in the UI thread as they should not deadlock the UI. This design principle works and is the right way to go, but however these do not take into account the spongy castle encryption and decryption methods as the methods when run in background and at times, before one decryption method finishes another method starts and there is a conflict and corruption. This leads to complications as even a valid message won't be decrypted successfully and thus not shown.

To overcome this flaw we make use of synchronized methods.[62]

> A **synchronized method** ensures that only one a single thread can execute an object's synchronized method at a given time.

This allows the threads to wait for resources to become available and also notify the thread that makes resource available to notify other threads are on the queues for the resources.

With making both the encryption and decryption methods synchronized the problem was overcome successfully.

4.13.3 Wire message PULL feature

The Wire app will pull messages from the database at regular intervals. When it does this, it generally updates the entire chat room every single time. The app then updates all the messages on the app.

This leads to a problem as messages sent to the user are encrypted with his public key so are successfully decrypted, but the messages sent by the user are encrypted with the recipients public key so they cannot be decrypted. This causes messages which are sent earlier to have a problem.

To overcome this an additional check was made to just add only the new messages retrieved into the database and leave the other messages as they were. This allows for a lesser load and better efficiency also in the end result.

5 Testing

This chapter details the test procedures and results of using the Secure IM system. It defines the test plan followed and the results obtained. The scope is defined. Also the product end results are defined and specified in comparison with the requirements.

5.1 Testing Plan

Testing is one of the most important part of a software development life cycle. There are different ways of approaching an application to test. The items listed below will focus on identifying the scope of testing this application.

- Items / functionality to be tested
- Testing Strategies and approach to be followed
- Identifying the roles and responsibilities
- Test Deliverables
- Define the Entry n Exit Criteria for each phase of testing
- Test Environment-Hardware / Software
- List Risks and contingencies / mitigation plans to overcome

5.1.1 Scope

The scope includes development and execution of the test suit consisting of the test scenarios for Unit Testing, Integration Testing, and System Testing features. They are as follows:

- Unit Testing needs to be done at the development stage
- Test cases are based on Use case and Requirement specifications
- Test execution will be carried out mostly through manual approach

5.1.2 Test Approach

The different levels of testing that are carried out during the Test Execution are shown below.

Test Types	**Test Levels (Risk Level)**	**Strategy / Methods**	**Techniques for identification of test cases**
Unit Testing	Low	Black Box Testing	Basic function testing.
UI / Usability Testing	White	White Box Testing	Basic ease-of-use and function testing.
Functionality Testing	High	Black Box Testing	Test scenario shall be identified and based on that test cases shall be developed.
Usability Testing	High	Black Box Testing	Specify the functional areas to regression needed.
Regression Testing	Low	Black Box Testing	Specify the functional areas for which regression is needed.
Data Integrity	High	Database Testing	This will validated by using a set of database scenarios with set of database scenario with set of available or prepared data.

Table 5.1: Test Types

5.1.3 Entry Criteria

The following conditions should be met for test plan entry criteria:

- All functionality described in requirement specification has been implemented
- All unit and Integration Test cases should have been successfully executed
- Test environment includes test database with the master data operational

5.1.4 Exit Criteria

The testing phase can conclude when the following criteria has been met:

- All the test cases and Scenario including Unit, Integration, and System Testing should be executed
- All open defects have been properly fixed and verified
- All the planned test scenarios / test cases are executed and these cover all the system requirements functional, non functional, system / business requirements-functional and non functional.

5.2 Test Environment

5.2.1 Test Hardware

The test devices used as mentioned before are the HTC One S, Samsung I9100 Galaxy S II and the Google Nexus 7 (2012). The technical details of the devices are shown in the table that follows.

	HTC One S	**Samsung I9100 Galaxy S II**	**Google Nexus 7**
Type	Smart Phone	Smart Phone	Tablet
Android version	4.1.1	4.1.2	4.4
Chip	Qualcomm Snapdragon S4 Plus MSM8260A	Samsung Exynos 4210	NVIDIA Tegra 3 T30L
Processor	Dual core, 1500 MHz, Krait	Dual core, 1200 MHz, ARM Cortex-A9	Quad core, 1300 MHz, ARM Cortex-A9
RAM	1024 MB	1024 MB	1024 MB
Internal Storage	16 GB	32 GB	8 GB
Display	540x960 pixels, 4.3 inches	480x800 pixels, 4.3 inches	1280x800 pixels, 7.0 inches
Pixel Density	256 ppi	218 ppi	216 ppi
Mobile Data	HSDPA, 42 MBps; HSUPA, 5.76 MBps, EDGE, GPRS	HSDPA, 21 MBps; HSUPA, 5.76 MBps, EDGE, GPRS	HSDPA 21 Mbps, HSUPA 5.76 MBps, EDGE, GPRS
WiFi	802.11 b/g/n	802.11 a/b/g/n	802.11 b/g/n
Dimensions (in mm)	130.9 x 65 x 7.8	125.3 x 66.1 x 8.49	198.5 x 120 x 10.45
Weight	120 g	116 g	340 g

Table 5.2: Test Devices

5.2.2 Test Software

For the purposes of unit testing and initial development, the applications are first tested on the HANA Cloud local run-time. This allows to run the applications locally, but for testing the app this is not enough as the mobile app works over the internet.

For this purpose, the Wire Key Repository and SAP Wire are hosted on the trial HANA Cloud Platform and can be found on the following addresses as of October 9, 2014:

Wire Key Repository: https://personp1940720813trial.hanatrial.ondemand.com/

SAP Wire: https://wired060155trial.hanatrial.ondemand.com/

The android app on the other hand is first run on the Android emulator, included in the Android SDK in harmony with the IDEA intelliJ IDE. However an emulator is good for checking functionality but for tests on usability and real-time tests, the above hardware devices are used.

5.2.3 Setting up the Test Environment

Setting up the server

As explained in section 5.2.2 the servers are setup on the HANA Cloud trial platform. The test devices are connected to WLAN and mobile data.

Installing the app

The native app when compiled has the file format "apk". The file must be transferred to the mobile device first and installed. This is done via USB. But since android does not allow to installation of non market apps by default, the option has to be set to allow it for that. Once this is done, three dummy trial accounts are created for testing.

5.3 Testing Methods Overview

5.3.1 Functional Testing

Here the basic functionality of the app is tested first one at a time individually then after integrating it with other modules continuously and performing functionality tests. It includes:

- Functionality of each module will be covered based on requirements specifications
- Check for valid and invalid data
- Tests carried on mobile data as well as WLAN
- Checks to ensure interdependence of modules

5.3.2 Data and Database Testing

Here the data stored on the database is checked for consistency and validity. It includes:

- Data checks will be done on all types of database in scope
- Data checks like insert, update, retrieval will be done for all possible transactions
- Check whether database logging of data is proper or not
- Data validity check

5.3.3 UI / Usability Testing

This section deals with testing the basic "feel" and usability of the app from the end-users' point of view. It includes:

- Tests include look and feel aspects
- Running the app in landscape as well as portrait
- Ease-of-use tests

5.3.4 Performance Testing

This section deals with testing the performance of the app during regular use. It includes:

- Check response time, transaction times and other time sensitive requirements to ensure that it is consistent with the requirements
- Memory usage and CPU usage tests

5.3.5 Failure and Recovery Testing

This section deals with testing the app's functionality on failure of any service or the app itself and their recovery mechanisms. It includes:

- Check for abnormal shutdowns for the application, system failures / network malfunctions.
- Check for for data loss due to data corruption, system failures, and database failures.

5.3.6 Regression Testing

This section deals with regression tests which are done after bug or defect fixes. A regression test will be performed after each phase to ensure that:

- No impact on existing functionality
- To ensure that there is an increase in the functionality and stability of the software

5.4 Test Results Overview

The tests were executed in process and the results were documented. Most tests were completed successfully and a few defects were encountered. The most troublesome defect was that the App crashes when internet connection is too slow because it cannot connect to the SAP SAML connection.

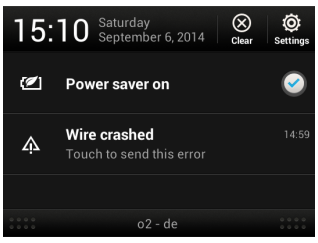


Figure 5.1: Wire crashed notification

This is a problem and defect with the API and little can be done to fix it. In addition to this possibly, since the servers are hosted on the HANA Cloud trial platform, the performance is not up to par with the customer HANA Cloud platform. This is a minor defect and considered out of scope of this development.

In addition to this, few defects were encountered and fixed. The code was continuously updated and kept track of using GIT repository. This distributed revision control and source code management (SCM) gives a clear overview of the changes was made and a proper version tracking enabled to track changes and affect bug fixes.

6 Conclusion

This chapter details the product developed and compares them with the requirements detailed in chapter 3. A short comparison with control approaches used in previous works is also done. The scope for future enhancements is also detailed and finally accomplishments of this work are summarized in this chapter.

6.1 The Concept Proposed

Taking into account the security challenges faced by IM applications, a prototype was suggested and this prototype overcame the challenges faced. The requirements were taken into consideration and the concept was proved by implementing the same in a prototype built on top SAP Wire. It offers the same level of performance and ease-of-use, in addition to this it offers the feature set for providing privacy and another additional level of security.

This coupled with the way a key can be verified using a QR code basically offers the best-practice methods to the corporate structure.

6.1.1 Requirements Re-visited

In section 3.2 a clear set of requirements was specified. In this section we compare the requirements with the end results of the product developed.

The requirements are divided into 4 principal sections:

Authentication

As specified, there should be a proper way of verifying that only a valid user is logged in and using the service and there should not be any misuse.

The prototype offers authentication through the SAP SSO and also provides the method of signing and verifying the signatures on all encrypted messages sent and received. In addition to this, the feature to verify the public key fingerprints successfully covers the man-in-the-middle attack and provides a trustworthy authentication service.

Confidentiality

As specified, the message should be unreadable by anyone other than the intended recipient.

A PGP based end-to-end encryption is implemented to the end product. This ensures, through the PGP encryption method, that confidentiality is maintained. The encrypted messages cannot be read by the server even if they were compelled to turn over the messages to a third party.

Forward Secrecy

As specified, even if network traffic has been captured, the messages should not be decipherable even if the private key of the client is found.

The product developed has the ability that the user can change his key pair at regular intervals or as per will, this prevents the attacker from deciphering any new messages. This allows forward secrecy.

Repudiability

As specified, the ability that the receiver of a message is able to prove to a third party that the sender really did send the message.

In the end product, this is achieved through digital signatures. This is an add-on to the authentication feature and provides an additional level of security and trust.

6.2 Comparison with other IM Products

Considering the popular messaging apps in the market today, the wire app developed with the encryption package has a lead.

Because of use of PGP, the app has a clear advantage of Whatsapp, BBM, Skype and Lync. Also because of the ability to verify the user as genuine by method of verifying public keys, the app thus gets an advantage over Hoccer XO, MyEnigma and Surespot. Splitting up of the key repository and the server, again put the app over the apps SIMSme, ChatSecure, TextSecure and Telegram.

Threema on the other hand offer a feature set that is currently best in the business when considering security in mobile IM apps. The product developed in course of this thesis offers a feature-set that matches this development.

6.3 Future Enhancements

The app developed though rich in in its features can be enhanced in a number of ways. Some of them include extending the functionality to include multimedia messages and location messages. In addition to this functionality of encryption can be extended to group chats as well.

Features can also include the improvement of the way user contacts are shared to make them more "private" and also add a more stronger way of verifying fingerprints.

Also a method in which communication with the key server can be made more secure than just using SSL may be a possible extension to this application.

In addition to this a further extension can be made to port the encryption features to the web app, iOS and desktop versions of SAP Wire.

6.4 Summary

The aim of this thesis was to develop a a concept based on best in practice methods that could be implemented on "normal insecure" OM platform to allow usage if IM in a secure way in which one doesn not trust the communication channel (e.g. HTTP/TCP traffic) or the server routing messages but just the recipient.

This was achieved by first identifying the challenges and problems faced by Instant Messaging applications and the current security technologies in them and accordingly find the best-in-practice solutions. Using the research done, a concept was worked out which matched the requirements of authenticity, confidentiality, non-repudiability, forward secrecy and trustworthiness. It was designed to be simple for use and could be added on to existing applications. The feasibility of this concept was then demonstrated by implementing it on top if an existing corporate IM prototype- SAP Wire and thus presenting a working prototype.

Although there is nothing like 100% security, the concept was successfully developed based on a proven best-practice and successfully implemented on to of a real IM shows that it is possible to use the developed concept to put security and privacy on the next level.

A Appendix

A.1 Screenshots

Figure A.1: Wire Android App Login Screen

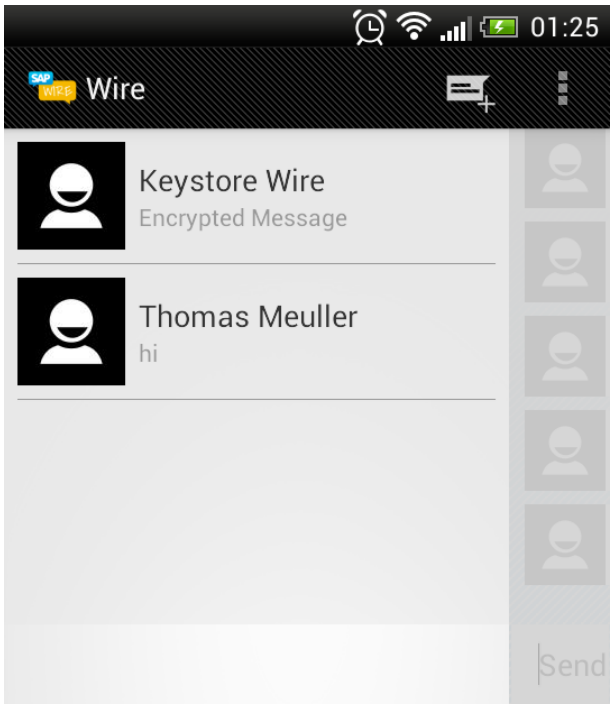


Figure A.2: Wire App Chatrooms

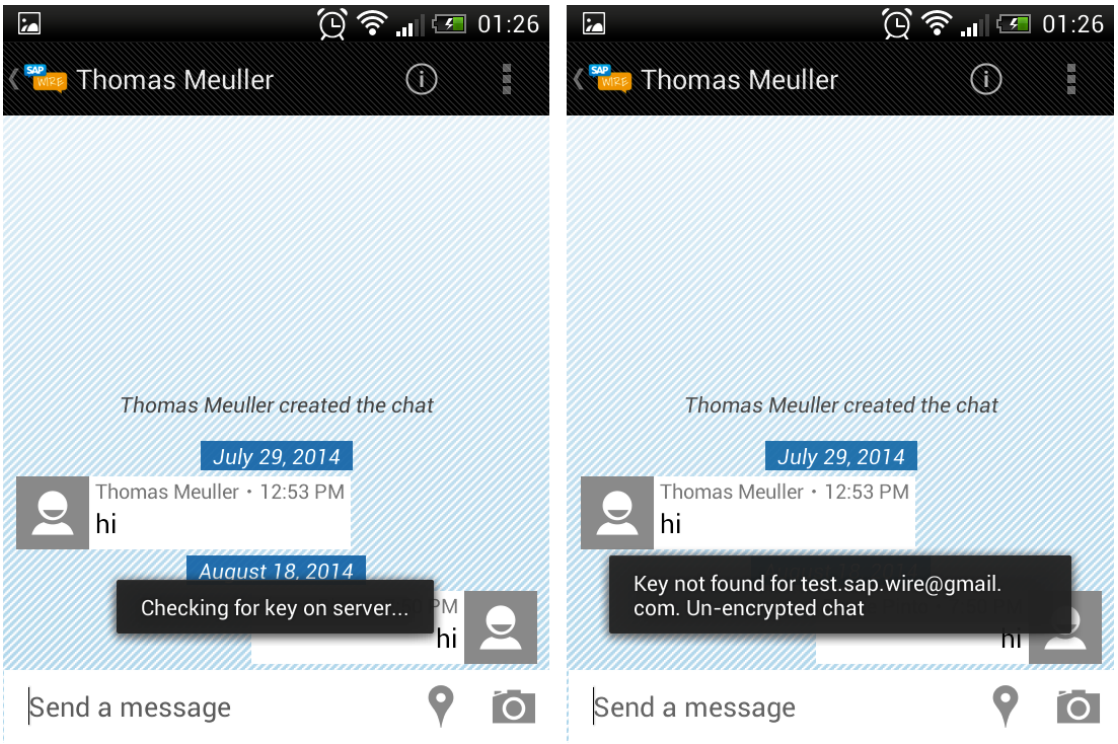


Figure A.3: Wire Un-encrypted chat

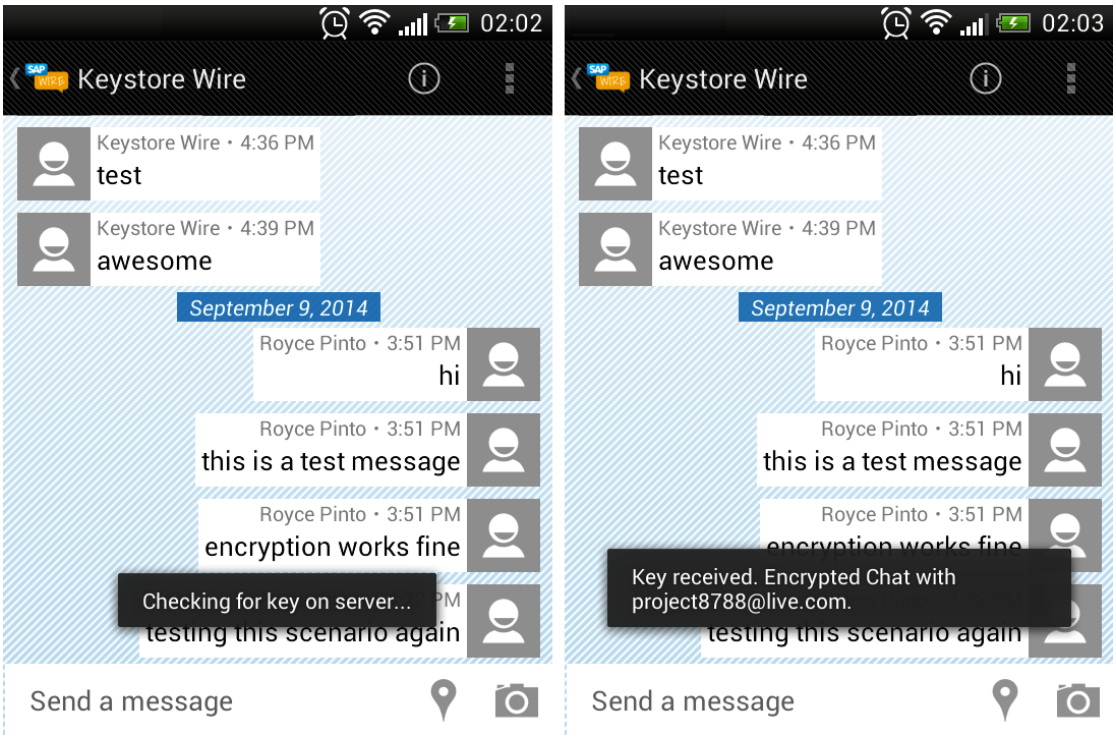


Figure A.4: Wire Encrypted chat

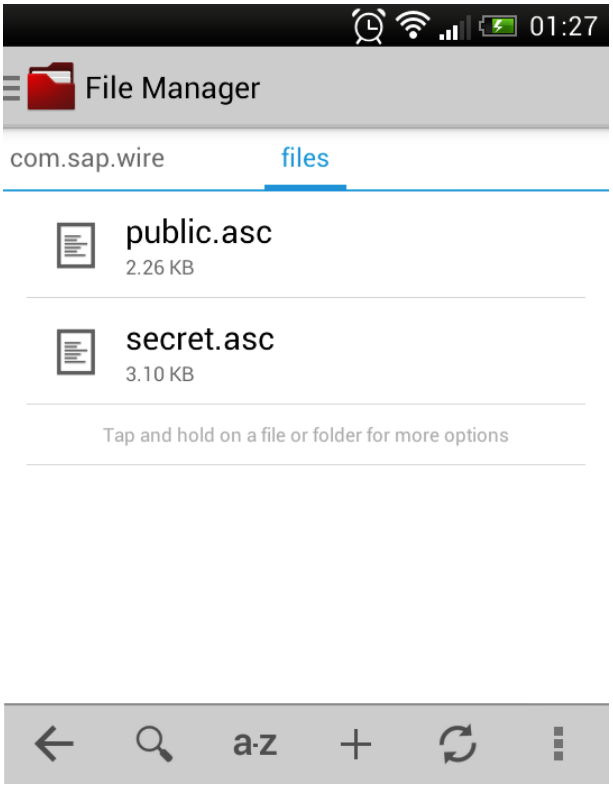


Figure A.5: Wire Key files

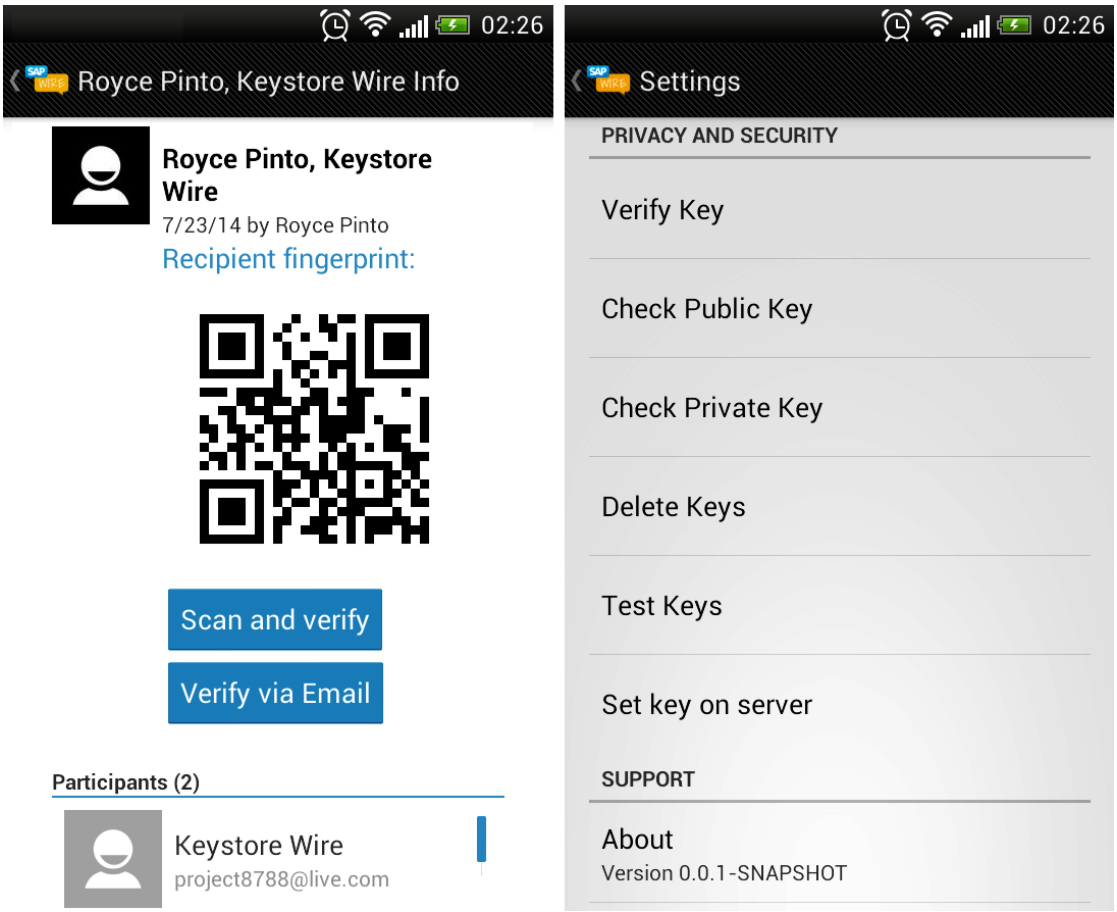


Figure A.6: Wire Key preferences and Verification options

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: BCPG v@RELEASE_NAME@

mQGiBFPOeosRBADEvSdbItohmTXIDWYGlGY7vIPGwrDQ6wLmoeT5016hoVVfzUe8
IFCeRXLDwQwkptfo78RyfpFMW/Gv0kIbh7t8KGdOpT8hhCxqzFZdPv/iijEhlSm2
OOg6GaqjRXT8S1wIlJJ6u4i9dKLkmJq6nMYvPWUitL5g5xU0Z+UfD0h58wCgrWcL
+0CiwzNNu5qwJKkDqX8qJ+kD/RHSOqEQ8l+gcdVvYK/rsk/GTnLUzwE278/cTWr9
n8aZioQISPZdQGF4K2grfm6D/0oTf468nqM9zpgy/fXk+Ew9XZ0XtBEzsNZKLuVf
Q8y+AFCI8NEgiCij89Tfmq1aQTYDSAUfxynMZHlqoGiDG36uKaeZUI5+hQcastPG
vXRsA/4qihDIvEZ2vzxfBLxpMI3aNBNXe4eXuWTSIJhDVk971OMMZjMu4t/JoGDs
DqCdeCFet9gX1icebczCmAENq5+yTrmohqQsxRlSkyEkClXYo0eykdQFypRpyG50
Yc97dHUEjY6U/NSxh91yNhSUvL1FbBxFVOdU6y75Ikm5qAlYQbQVcm95Y2UucGlu
qILH/3UU8woAnMHNlfe+QYEnP+kOtc8cF4xJ9m+iBjED/ieWZZ4Tt4JnmNBcj3VR
mJh3F9nRhD9rOcFEohESbEL96LiIUFqJF7a4fUQmh6dRReUi091iHMbWPKy36s80
hqN2fsgPVxi6/CidtzCybaqlhYPxnrrmr+Me7hj9rEUWhffQqMqNatTTL040NFKT
TVCOBu/VrAl3eM6XKAp+wSu1VD3exj5GIUeU+NBRj25mGpKATqKDTe2gB0pJXob8
X3b4fN5qLUnk46X0x0gkd+wzP6SIRgQYEQIABgUCU856iwAKCRCy2BYs4/VfEA+1
AJoD/eYtKHT82VLyPFkXUsa2BWCrFQCfRrZD0+lv2jRcBrRo3MZiSPNkkIg==dtab

-----END PGP PUBLIC KEY BLOCK-----
```

Figure A.7: A wire generated PGP Public Key

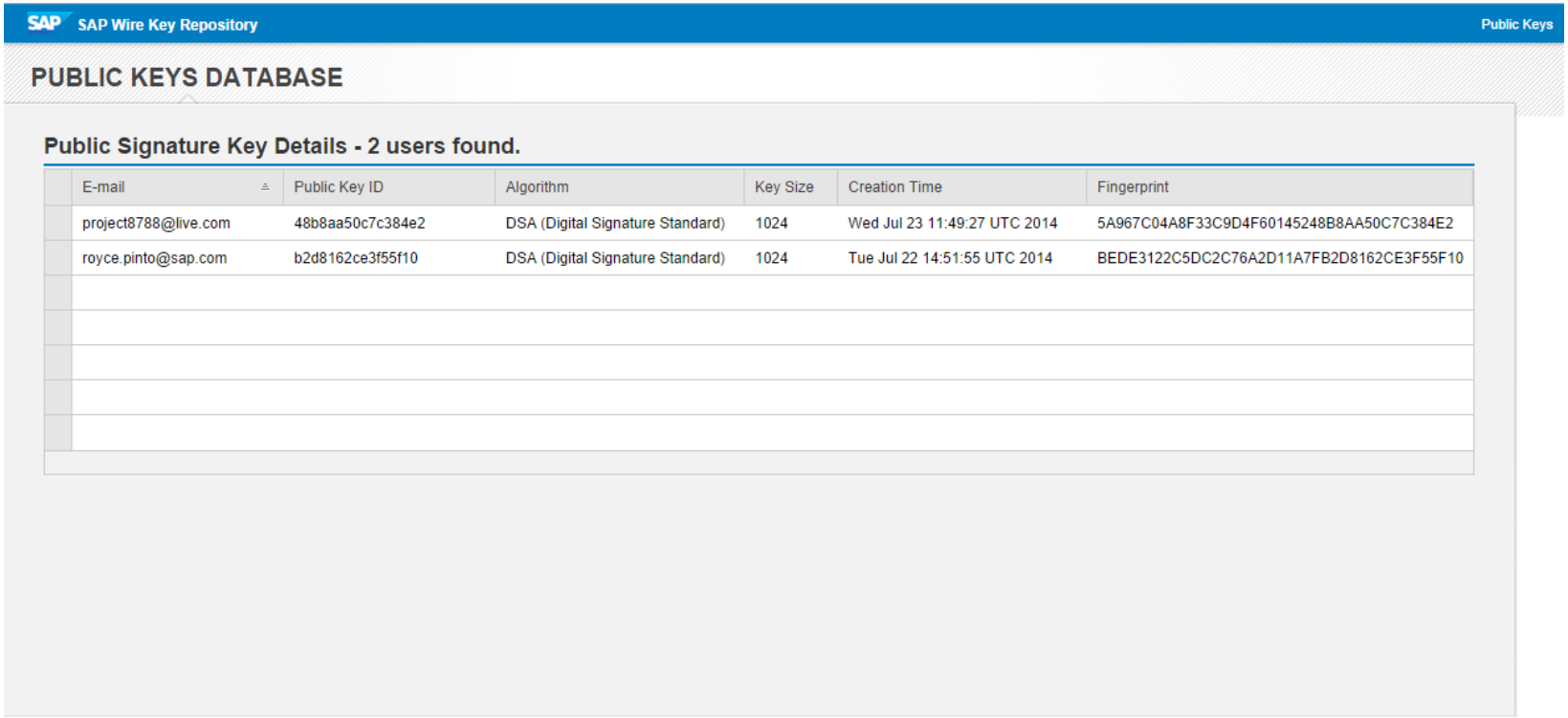


Figure A.8: Wire Key Repository

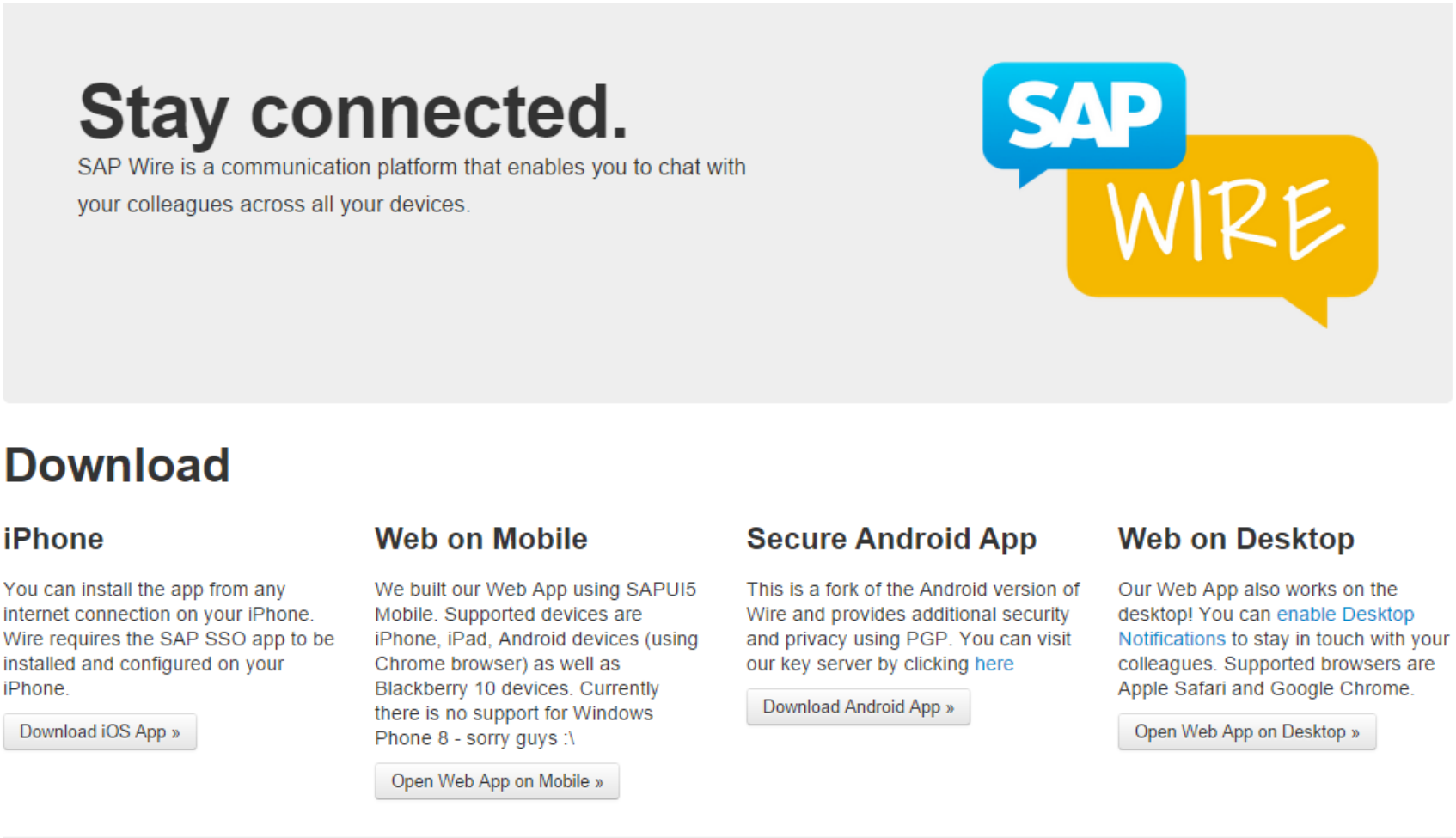


Figure A.9: SAP Wire website

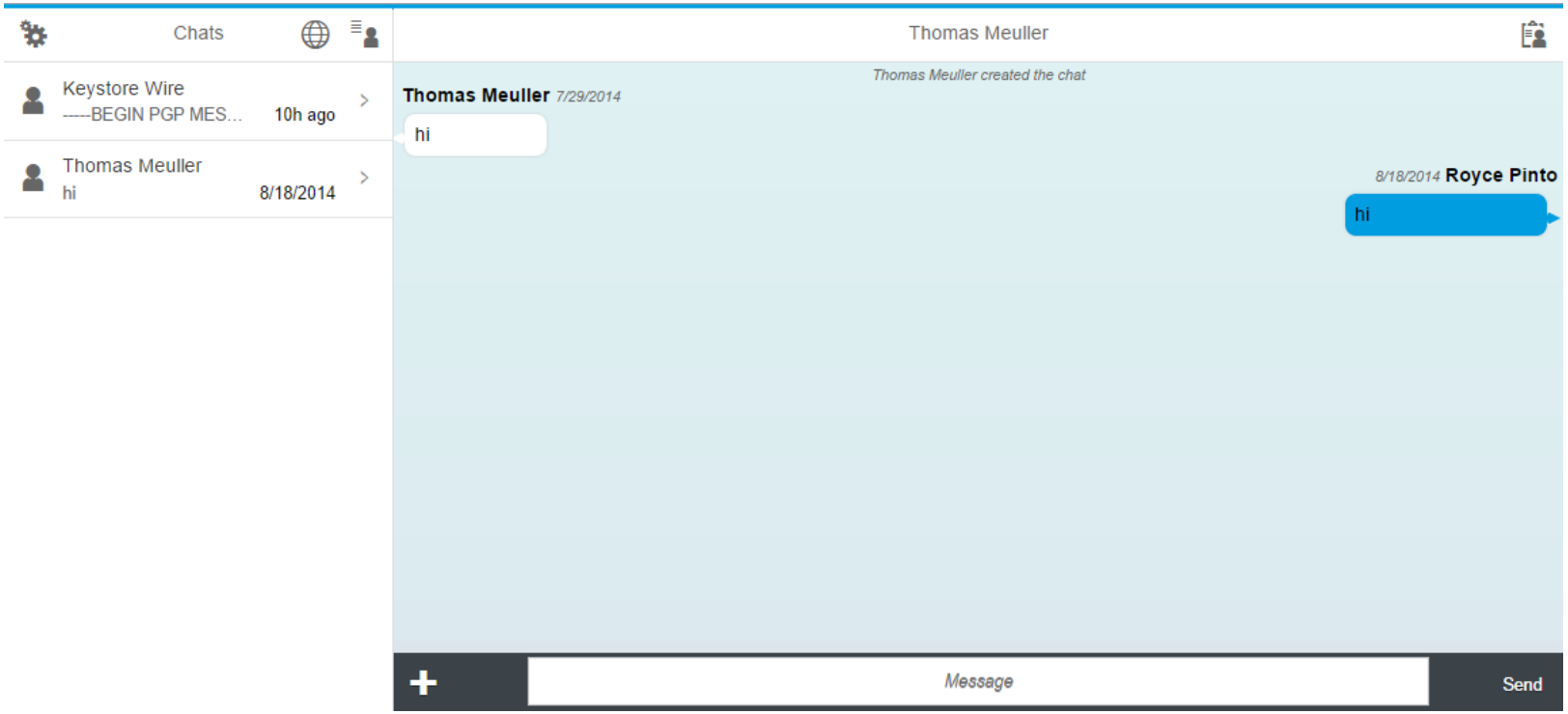


Figure A.10: Un-encrypted chat on Wire Web Application

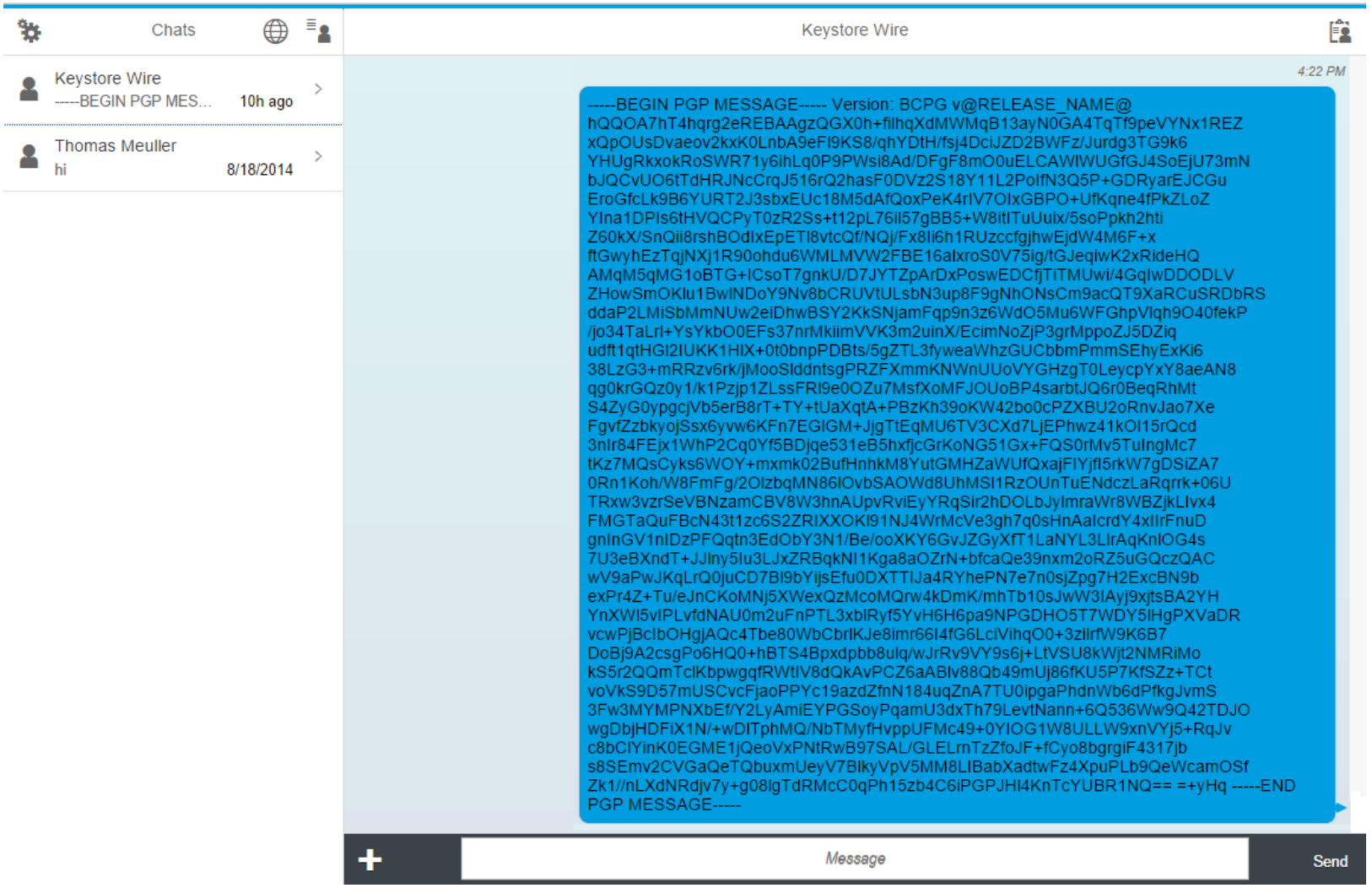


Figure A.11: Encrypted chat on Wire Web Application

B Glossary

3G The third generation standards family for Mobile Communication.

AES Advanced Encryption Standard

API Application Programming Interface

AVD Android Virtual Device

CAST Carlisle Adams, Stafford Tavares encryption algorithm

CCA Chosen-Ciphertext Attacks

CPA Chosen-Plaintext Attacks

DES Data Encryption Standard

DH Diffie-Hellman

DoS Denial of Service

ERP Enterprise Resource Planning

FIPS Federal Information Processing Standards

IDE Integrated Development Environment

J2SE Java Standard Edition

J2EE Java Enterprise Edition

JDBC Java Database Connectivity

JSON JavaScript Object Notation

HANA High-Performance Analytic Appliance

HMAC Hash-based message authentication code

IM Instant Messaging

KPA Known-Plaintext Attacks

MAC Message authentication code

MITM Man-in-the-middle

NaCl Networking and Cryptography library ("Salt")

OHA Open Handset Alliance

OS Operating System

PaaS Platform as a Service

PGP Pretty Good Privacy

QR Quick Response

RC4 Rivest Cipher 4 or Ron's Code 4

RTP Real-time Transport Protocol

SaaS Software as a Service

SAML Security Assertion Markup Language

SIP Session Initiation Protocol

SMS Short Message Service

SRTP Secure Real-time Transport Protocol

SSL Secure Socket Layer

SSO Single Sign On

XML Extensible Markup Language

XMPP Extensible Messaging and Presence Protocol

Bibliography

[1] Conder S. and Darcey L., *Android Wireless Application Development*, 2009. Addison-Wesley, ISBN 978-0-321-62709-4.

[2] Cellular News: SMS will remain more popular than mobile messaging apps over next five years, 29th May 2012

[3] Savage, Charlie. *N.S.A. Said to Search Content of Messages to and From U.S.* (8 August 2013).

[4] Chang-Ji Wang, Wen-Long Lin and Hai-Tao Lin, *Design of An Instant Messaging System Using Identity Based Cryptosystems*, Fourth International Conference on Emerging Intelligent Data and Web Technologies, 2013

[5] Pidgin (http://www.pidgin.im/)

[6] The Android SDK- http://developer.android.com/sdk/

[7] SAP HANA Cloud Portal- http://hana.ondemand.com

[8] Whatsapp- www.whatsapp.com

[9] M. Fabri, D. Moore and D. Hobbs, *Empathy and Enjoyment in Instant Messaging*, IEEE International Workshop on Human-Computer Interaction, Sept. 2005

[10] Radicati, Sara, *Instant Messaging Market, 2013-2017* (23 September 2013)

[11] Jaakko Kangasharju and Marko Saaresto, *Instant Messaging and Presence: Research and Challenges*, Seminar on instant messaging and presence architectures in the internet.

[12] Juniper Research, *Mobile Messaging Markets* (19 February 2014)

[13] Nikita Borisov, Ian Goldberg and Eric Brewer *Off-the-record Communication, or why not to use PGP*, Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society (WPES '04), Washington DC, USA, 2004. DOI 10.1145/1029179.1029200.

[14] Whisper Systems- www.whispersystems.org

[15] Marlinspike, Moxie. *The Difficulty Of Private Contact Discovery* (03 January 2014).

[16] ICT facts and figures, The International Telecommunication Union (May 2014)

[17] Skogberg, Benny. *Android Application Development- A guide for the Intermediate Developer* (6 September 2010).

[18] *Android version history-Wikipedia*

[19] Twister- www.twister.net.co

[20] PingPal- www.pingpal.io

[21] Snapchat- www.snapchat.com

[22] Why I created PGP- Philip Zimmermann (www.philzimmermann.com/EN/essays/WhyIWrotePGP.html)

[23] *Introduction to Cryptography*, PGP 6.5.1 documentation, 1990-1999 Network Associates, Inc.

[24] SAP Wire (www.sapwire.hana.ondemand.com)

[25] Instant Messenger Security (http://www.technicalinfo.net/papers/IMSecurity.html)

[26] Piercing Through WhatsApp's Encryption (https://blog.thijsalkema.de/blog/2013/10/08/piercing-through-whatsapp-s-encryption/)

[27] Communication breakdown (http://apps.washingtonpost.com/g/page/world/communication-breakdown/1153/)

[28] Nielsen/NetRatings, *A report on the instant messaging services*, 2002.

[29] WhatsApp encryption flaw revealed, POC code published (http://www.net-security.org/secworld.php?id=15745)

[30] Telegram (http://www.telegram.org)

[31] Telegram, AKA “Stand back, we have Math PhDs!” (http://unhandledexpression.com/2013/12/17/telegram-stand-back-we-know-maths/)

[32] Hiroaki Kikuchi, Minako Tada, Shohachiro Nakanishi. *Secure Instant Messaging Protocol Preserving Confidentiality against Administrator*, 18th International Conference on Advanced Information Networking and Applications, 2004. DOI 10.1109/AINA.2004.1283749

[33] Xue Sun, Zhenjun Du, Rong Chen. *A Secure Cross-platform Mobile IM System for Enterprise Applications*, International Conference on Uncertainty Reasoning and Knowledge Engineering, 2011

[34] Tsai-Yeh Tung, Laurent Lin, D. T. Lee. *Pandora Messaging: An Enhanced Self-Message-Destructing Secure Instant Messaging Architecture for Mobile Devices*, 26th International Conference on Advanced Information Networking and Applications Workshops, 2012

[35] WhatsApp-Alternativen fürs Business, IX Magazin für Professionelle Informationstechnik pages[76-83], July 2014

[36] HoccerXo (http://hoccer.com/)

[37] D. Park, C. Boyd and S.-J. Moon. *Forward secrecy and its application to future mobile communications security*. In Proceedings of the 3rd International Workshop on Practice and Theory in Public Key Cryptography (PKC 2000), volume 1751 of LNCS, pages 433–445. Springer-Verlag, Jan. 2000.

[38] P. Henry and H. Luo. *Off-the-record email system*. In Proceedings of the IEEE INFOCOM, pages 869–877, Apr. 2001.

[39] H. Krawczyk, M. Bellare, and R. Canetti. *HMAC: Keyed-hashing for message authentication*, Feb. 1997. RFC 2104, Status: Informational. (http://www.ietf.org/rfc/rfc2104.txt)

[40] BlackBerry Messenger (http://us.blackberry.com/bbm/bbm-chat.html)

[41] BBM Protected (http://us.blackberry.com/business/products-services/e-bbm.html)

[42] Security Note- BBM Protected, 16 Jun. 2014.

[43] Security Requirements For Cryptographic Modules, Federal Information Processing Standards Publication, May 25, 2001.

[44] MTProto (http://core.telegram.org/mtproto)

[45] ChatSecure (http://www.chatsecure.org)

[46] Feng Cao and Malik, S., *Security analysis and solutions for deploying IP telephony in the critical infrastructure*, Workshop of the 1st International Conference on Security and Privacy for Emerging Areas in Communication Networks, 2005.

[47] McNelly, Brian. VoIP Security Vulnerabilities. Boulder, December 6, 2007.

[48] Microsoft Lync (http://products.office.com/lync/)

[49] Skype (http://www.skype.com/)

[50] myENIGMA: Whitepaper, Qnective AG, 2013

[51] Surespot-encrypted messenger (https://www.surespot.me/)

[52] Threema- Seriously secure mobile messaging (https://threema.ch/)

[53] SIMSme (http://www.sims.me/)

[54] Java Persistence API (http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html)

[55] SAP UI5 (https://sapui5.hana.ondemand.com/)

[56] SAML 2.0 and SAP GUI Single Sign-On in one and the same scenario, Donka Dimitrova, SAP, June 2, 2014

[57] Non-repudiation (http://en.wikipedia.org/wiki/Non-repudiation)

[58] Bouncy Castle (https://www.bouncycastle.org/)

[59] Spongy Castle by rtyley (http://rtyley.github.io/spongycastle/)

[60] Official ZXing ("Zebra Crossing") Project Home (https://github.com/zxing/zxing)

[61] Google Play: Barcode Scanner (https://play.google.com/store/apps/details?id=com.google.zxing.client.android)

[62] The Java™ Tutorials- Synchronized Methods (http://docs.oracle.com/javase/tutorial/essential/concurrency/syncmeth.html)

[63] Orwell, George. *1984 by George Orwell*, Ed. Erich Fromm. New York: Harcourt, 1949.