Secure Gmail

Aisling Connolly
B.A.I. Engineering
Final Year Project April 2012
Supervisor: Dr. Hitesh Tewari

School of Computer Science and Statistics
O’Reilly Institute, Trinity College, Dublin 2, Ireland
Declaration

I hereby declare that this thesis is entirely my own work and that it has not been submitted as an exercise for a degree at any other university.

_______________________________ April 2012
Aisling Connolly
Permission to Lend

I agree that the Library and other agents of the College may lend or copy this thesis upon request.

_______________________________ April 2012
Aisling Connolly
Acknowledgements

I would like to thank Dr. Hitesh Tewari for supervising my final year project and offering guidance throughout. I would also like to thank Karl Reid, postgraduate student, for providing the platform upon which I have built my project, and offering help and advice when needed.

I would like to thank my parents for the support they have shown and the relentless proof reading they have done for this report. I thank my brother and sister for sitting through the many demonstrations and for acting interested.

I also thank my friends and classmates for keeping me sane throughout this year, particularly Gen and Kelly for removing me from the fluorescent lights of LG12 to see daylight every now and again.
Abstract

The use of web-based electronic mail communications has increased rapidly in the past decade. The security of email communication has not grown at the same rate, with only a small number of users encrypting their email messages in comparison.

This report presents a design for an Identity-Based Cryptographic application for webmail service “Gmail”, implemented via a Firefox extension. The resulting application is called ‘Secure Gmail’ and provides users with secure email communication. Within this report the motivation behind the project is discussed and the current state of the art of cryptography is examined.

Also contained within the report is a description of the design and implementation of ‘Secure Gmail’. The real world implementation of the project and the potential barriers to its success are outlined. The report concludes by presenting a number of possible future prospects for the application of Identity-Based Cryptography, and plans for the expansion of the ‘Secure Gmail’ service.
Table of Figures

FIGURE 2.a: ILLUSTRATION OF MAIL TRANSMISSION FROM ALICE TO BOB, INTERCEPTED BY TRUDY.........................4
FIGURE 2.b: SIMPLE REPRESENTATION OF A PUBLIC-KEY (ASYMMETRIC) CRYPTOGRAPHIC SYSTEM ......................6
FIGURE 2.c: REPRESENTATION OF AN IDENTITY-BASED CRYPTOGRAPHIC SYSTEM ...........................................8
FIGURE 2.d: GRAPH SHOWING THE NUMBER OF GMAIL USERS WORLDWIDE OVER THE PERIOD 2009 TO 2012 ......11
FIGURE 2.e: GENERIC FORMAT OF A UNIFORM RESOURCE IDENTIFIER .............................................................13
FIGURE 3.a: COMPANY LOGO OF EMAIL ENCRYPTION PRODUCT PROVIDER VOLTAGE SECURITY INC ..................26
FIGURE 4.a: INSTALLING AN ADD-ON (EXTENSION) FROM FILE USING THE FIREFOX ADD-ON MANAGER ..........28
FIGURE 4.b: THE ADD-ON MANAGER AFTER THE EXTENSION “FYP HTTP OBSERVER 1.0” HAS BEEN INSTALLED.....28
FIGURE 4.c: REPRESENTATION OF THE FILE HIERARCHY IN THE PACKAGE ‘EXTENSION.XPI’ ............................29
FIGURE 4.d: XUL FILE ‘OVERLAY.XUL’ ........................................................................................................30
FIGURE 4.e: MANIFEST FILE ‘CHROME.MANIFEST’ ..........................................................................................31
FIGURE 4.f: INSTALL MANIFEST RDF FILE ‘INSTALL.RDF’ .............................................................................32
FIGURE 4.g: GENERIC FORMAT OF A UNIFORM RESOURCE IDENTIFIER ..........................................................35
FIGURE 4.h: SAMPLE OF THE CONTENTS OF THE POST DATA WITHIN THE RELEVANT POST URI WHEN AN EMAIL WITH SUBJECT LINE: “SUBJECT LINE” AND MESSAGE BODY: “MESSAGE BODY” IS SENT TO EMAIL ADDRESS: FINALYEARPROJ@GMAIL.COM ........................................................................................................35
FIGURE 4.i: REGULAR EXPRESSION FOR THE GET BIND URI WHICH ESTABLISHES THE CONNECTION BETWEEN SERVER AND CLIENT, AND KEEPS IT OPEN TO LISTEN FOR REQUESTS AND RESPONSES .................................................................36
FIGURE 4.j: A SAMPLE OF THE GET BIND URI RESPONSE WHEN AN EMAIL WITH SUBJECT LINE: “SUBJECT SUBJET SUBJECT” AND MESSAGE BODY: “MESSAGE MESSAGE MESSAGE” IS RECEIVED FROM EMAIL ADDRESS: FINALYEARPROJ@GMAIL.COM .................................................................37
FIGURE 4.k: THE REGULAR EXPRESSION OF THE POST URI WHICH CONTAINS THE REFERENCE NUMBER FROM THE PREVIOUS GET URI RESPONSE SEGMENT (FIGURE 4.i), CORRESPONDING TO AN OUTBOUND EMAIL......38
FIGURE 4.m: XMLHTTPREQUEST SENT TO THE LOCAL IBE CLIENT, THROUGH THE SPECIFIED PORT TO ENCRYPT A PLAINTEXT STRING USING AN EMAIL ADDRESS AS THE RECIPIENTS IDENTITY..............................................................................40
FIGURE 4.n: XMLHTTPREQUEST SENT TO THE LOCAL IBE CLIENT, THROUGH THE SPECIFIED PORT, TO DECRYPT A CIPHERTEXT STRING........................................................................................................40
FIGURE 5.a: SCREENSHOT OF THE TERMINAL DISPLAYING THE NECESSARY THREE INPUT PARAMETERS FOR ‘SERVER.SH’ ..........................................................................................................................43
## Contents

Acknowledgements............................................................................................................................ ii  
Abstract ............................................................................................................................................. iii  
Table of Figures ................................................................................................................................. iv  
1. Introduction .................................................................................................................................... 1  
   1.1 Project Objective ...................................................................................................................... 1  
   1.2 Project Motivation ................................................................................................................... 1  
   1.3 Report Structure ....................................................................................................................... 2  
2. Background ..................................................................................................................................... 4  
   2.1 Cryptographic Schemes ............................................................................................................ 4  
      2.1.1 Symmetric Key Cryptography ............................................................................................ 5  
      2.1.2 Public (Asymmetric) Key Cryptography ............................................................................. 5  
      2.1.3 Identity-Based Cryptography ............................................................................................ 7  
      2.1.4 Hierarchical IBE ............................................................................................................... 10  
   2.2 Gmail ...................................................................................................................................... 10  
      2.2.1 Privacy and Security Issues .............................................................................................. 12  
      2.2.2 Hypertext Transfer Protocol (HTTP) ................................................................................ 12  
      2.2.3 Transmission Control Protocol (TCP) ............................................................................... 15  
   2.3 The Importance of Email Encryption ...................................................................................... 16  
      2.3.1 Data Loss Prevention ....................................................................................................... 16  
      2.3.2 Why Don’t Users Encrypt Email? ..................................................................................... 17  
      2.3.3 Legislative Motivation for Email Encryption ................................................................. 19  
3. State of the Art ............................................................................................................................. 20  
   3.1 The Start of Identity-Based Cryptography ............................................................................. 20  
   3.2 Advances in IBE ...................................................................................................................... 21  
   3.3 The Boneh-Franklin Scheme ................................................................................................... 22  
      3.3.1 Mathematical Foundation ............................................................................................... 22  
      3.3.2 Operation ........................................................................................................................ 22  
      3.3.3 Level of Use ..................................................................................................................... 23  
   3.4 Existing Email Cryptographic Products ................................................................................... 23  
      3.4.1 Email Client Encryption ................................................................................................... 24  
      3.4.2 Webmail Encryption ........................................................................................................ 25  
      3.4.3 Voltage Security Inc ......................................................................................................... 25
1. Introduction

This report outlines the progress and completion of the Final Year Project titled ‘Secure Gmail’, carried out under the School of Computer Science, Trinity College Dublin. The project was carried out under the supervision of Dr. Hitesh Tewari.

1.1 Project Objective

The aim of this project was to develop a Firefox add-on for Gmail. The add-on intercepts any emails that are sent to or from a Gmail account and encrypts or decrypts the body of the message accordingly.

As opposed to traditional public key cryptography, an Identity Based Cryptography (IBC) technique is employed, where a unique identity associated with the recipient, such as their email address, can be used to generate their public key. A further aim of this project was to develop the add-on so that operation is as transparent as possible once installed and intuitive when user interaction is required.

A personal aim in developing the add-on was to extend my knowledge of cryptographic schemes, building on material learnt in my Junior Sophister year. Becoming familiar with JavaScript coding and web based applications was another goal of mine when deciding to undertake the project.

The main objective of this project was to confirm that Identity-Based Encryption (IBE), another term used for IBC, can be applied to a webmail service efficiently, with a negligible amount of disruption to the operation of the service.

1.2 Project Motivation

Email has become an exceedingly popular means of communication for the exchange of corporate and personal data. The majority of email users, specifically those using a personal email account, do not encrypt their email, or even consider that their email messages may be vulnerable to attack.
Unaware of the dangers of sending confidential data through email communications, users may include bank details, social security numbers and other personal details in an email and send it unencrypted, open to attack.

There are many products available for email cryptography and yet many users do not employ these applications. Although the reason for this failure to use such applications is uncertain, studies suggest that there is a need for a transparent, intuitive cryptographic service for email users, who have little knowledge of the process of encryption. Reducing the need for user interaction in encrypting email can also reduce the error associated with user confusion and oversight, providing a more secure data protection service.

This project is also motivated by the demand for a lightweight, low cost cryptographic service for webmail. During a workshop run by the National Institute of Standards and Technology (NIST) in 2008 [8], it was estimated that IBE is three to five times less expensive to operate than PKC schemes. The operational overhead associated with IBE is relatively small and so the implementation of IBE is less time consuming than the implementation of other schemes.

1.3 Report Structure

Chapter 1 presents the objective of this project and the motives that have prompted the development of the project, ‘Secure Gmail’.

Chapter 2 outlines the background of cryptographic schemes and the history of Gmail. It also discusses some reasons why people are reluctant to encrypt their emails and outlines HTTP and TCP protocols, both important to the design of this Firefox extension.

Chapter 3 explores, in further detail, the evolution of Identity-Based Encryption, with particular emphasis on the 2001 Boneh-Franklin scheme. The email cryptographic applications currently available on the market are also listed and discussed.
Chapter 4 describes the design process involved in developing the ‘Secure Gmail’ extension. The process of building a Firefox extension, writing a JavaScript file to intercept HTTP traffic and implementing an IBE client is explained in this chapter.

Chapter 5 describes the method by which the ‘Secure Gmail’ system is set up and implemented. Measures taken to test the extension are also outlined in the chapter. The potential real world implementation of the system is discussed and possible barriers to its success identified.

Chapter 6 concludes this report. In this chapter the lessons learned and difficulties encountered are presented. Some future prospects for IBE and email encryption are proposed as a closing point to this report.
2. Background

2.1 Cryptographic Schemes

The main purpose of cryptography in communications is to prevent data from being viewed by people, other than the intended data recipient. Looking at the example below Alice wishes to send a message to Bob. Trudy is waiting to intercept the message as it is transmitted over the channel.

![Illustration of mail transmission from Alice to Bob, intercepted by Trudy](image)

If the message is sent in plaintext, Trudy can view the contents and use it maliciously. If the message is encrypted and sent as ciphertext over the network, Trudy cannot view the true content of the message.

In “Introduction to Identity-Based Encryption” [1], Luther Martin lists the five objectives that an effective information security solution should meet. The first is **Confidentiality**, keeping information safe and secret from those who should not be allowed to view it. The second is **Integrity**, ensuring information has not been altered by unauthorized means. An information security solution should ensure that information is readily available to the appropriate users, the objective of **Availability**.
The solution should be able to verify the identity of a user, known as the Authentication objective of a security system.

The final objective which should be met is Non-Repudiation, to prevent the denial of previous commitments or actions. It is a property of security solutions that provides proof of the integrity and origin of data. In a security system which supports non-repudiation, a user cannot deny data that they have sent using the system.

There have been many attempts to solve the problem of establishing secure communication from Symmetric Key Cryptography to Public Key Cryptography, and more recently Identity-Based Cryptography.

2.1.1 Symmetric Key Cryptography

In the early 1970s symmetric key cryptographic (SKC) schemes were being implemented in military networks, interbank communication and academic systems [10]. In an SKC system the sender and receiver use the same public key to encrypt and decrypt messages. The public key is available to all system users.

Although this type of cryptography was useful for contained networks, where all users could know the public key and decrypt messages accordingly, the scheme could not be used for bigger networks, in which users may not necessarily know or trust each other. This problem was further highlighted when the “dot com” boom hit in the mid 1990s [10].

2.1.2 Public (Asymmetric) Key Cryptography

To overcome the obstacles faced by symmetric key cryptography, the RSA public-key cryptographic (PKC) algorithm was proposed by Rivest, Shamir and Adleman in 1978 [3]. RSA employs the use of asymmetric keys. Instead of the same key being used for encryption and decryption, the recipient’s public key is used to encrypt the message and the recipient’s private key is used to decrypt the message [1].
Public keys are most commonly distributed using certificates, issued by a Trusted Third Party (TTP) or Certificate Authority (CA) [10]. A user’s private key is generated randomly by a Private Key Generator (PKG). The corresponding public key is generated from the private key and is added to a public register by the user [44]. Users must pre-enroll with the CA before their public key and private key pair can be generated.

The systems based on the RSA scheme, commonly known as Public-Key Infrastructure (PKI), grew in popularity in the late 1990s [10]. More suited to the larger networks than SKC, PKI is the basis for the two most widely used email encryption schemes of today, S/MIME (Secure Multipurpose Internet Mail Extensions) and PGP (“Pretty Good Privacy”) [21].

PGP was released as an encryption scheme by Phil Zimmerman in 1991 [18]. Although it is similar to PKI, PGP employs a “web of trust” where a user must trust that they have been given the correct public key for a certain recipient, rather than using a certificate authority to distribute public keys [21]. For this reason PGP is limited to smaller networks, where all users know each other and therefore trust that all public keys are correct.
S/MIME is closely based upon the Public Key Cryptography Standard (PKCS #7) [21] and is the PKC scheme included in the major email applications of today. It is a set of protocols for securely sending data, encoded using the MIME format [19]. Although the cryptographic extension is included in most mail systems, users are still reluctant to encrypt their messages. The reasons for this will be discussed in Section 2.3.2 of this report.

Although the asymmetric public-key scheme has proven to be more secure than the original SKC schemes, it does have its shortcomings. The cost associated with distributing certificates, and the effort involved in keeping key revocation lists, means PKI is not well suited to wide network communication [10].

The user pre-enrolment requirement means that a message intended for a user cannot be encrypted until that user has requested a private key, generating the necessary public key simultaneously [1]. Until these problems are resolved, PKI cannot be considered a perfect cryptographic scheme.

### 2.1.3 Identity-Based Cryptography

Identity-Based Cryptography is a public-key technology in which a sender can calculate a recipient’s public key from an arbitrary string that represents the recipient’s identity, such as an IP address or an email address [7]. In 1984, Adi Shamir [4] described an Identity-Based Cryptographic scheme:

> “An identity-based scheme resembles an ideal mail system: If you know somebody’s name and address you can send him messages that only he can read, and you can verify the signatures that only he could have produced”

In Figure 2.c sender Alice encrypts a message intended for Bob, using his email address (bob@gmail.com in this case). The encrypted message is sent across a channel to Bob, who in turn uses his private key, generated by the PKG, to decrypt the message before reading.
The history and mathematical foundation of IBE is discussed in more detail in Chapter 3, with particular focus on the Boneh-Franklin IBE scheme [5] in Section 3.3.

The advantages of using IBE over PKC are worth reiterating. In 2008 it was estimated that IBE systems are approximately three to five times cheaper to operate than traditional PKI [7]. This is attributed to the low user support costs associated with the simple IBE system. Luther Martin [8] of Voltage Security Inc reported approximately six million users of IBE worldwide as of June 2008.

In an IBE system users need not pre-enrol with a CA in order to make a public key available to other users. Alice can simply encrypt a message using Bob’s identity and a set of public parameters from a public parameters server, even if Bob has not yet acquired a private key. If Bob wants to decrypt the message at any time he can authenticate to the PKG and receive his private key. This allows for offline operation at both the sending and receiving ends.
This property of IBE can prove very useful during times of natural disaster relief. A government body cannot always predict the time or location of a natural disaster, and so cannot know who they will need to communicate with for future disaster relief plans. With IBE, the government body can generate a public key for a certain branch of the Red Cross, for example, and encrypt important instructions quickly and easily. Also, since the need to look-up a user’s public key is not an issue in IBE, the system can run more transparently than other cryptographic systems.

Regular renewal of IBE key pairs can be arranged by appending a time period onto the end of a users ID and generating a public key accordingly. For example, using string “bob@gmail.com || January 2012” to generate a public key for Bob creates a key that is valid only for the month of January in the year 2012.

If Bob’s key is compromised at any point, Bob need not acquire a new email address as the short-lived property of the keys implies that a new key will be generated, in this case, at the start of the next month. Keys can be renewed even more regularly, on a daily or indeed an hourly basis, if needed. A standard format for date and time must be agreed by all users so that the same ID string is used to generate a user’s public key and private key [6].

Users can utilize this property of IBE to “encrypt into the future” [6]. By appending a future date to recipient’s identity and generating a public key, the recipient cannot decrypt the message correctly until the specified date. This could be useful in sending a message out for a product release on a certain date in the future that cannot be decrypted until that date, for example.

Although there are many advantages associated with IBE, in terms of straightforward implementation and lower costs, it does not offer any new capabilities compared with PKI. IBC also lacks the non-repudiation objective of information security solutions [1], discussed in Section 2.1.
At the NIST workshop ‘Applications of Pairing-Based Cryptography: Identity-Based Encryption and Beyond’, held in June 2008, it was concluded that “IBE as a technology, compliments PKI rather than replacing it” [8].

2.1.4 Hierarchical IBE

The original IBE system employed a single PKG to generate private keys for all users. In these systems it is not possible to create a hierarchy of PKGs, where a high-level PKG controls the keys granted by the lower-level PKGs, subordinate to it. A hierarchical structure is valuable in terms of security as an IBE system with just one PKG is easier to attack, compromising all private keys in the system.

In 2002 Horwitz and Lynn [9] introduced the concept of Hierarchical Identity-Based Encryption (HIBE), in which the operation of a PKG at a particular level in the system depends on those PKGs located in higher levels of the hierarchy. The system consists of a root PKG and a number of levels of PKGs operating under the control of the root PKG.

The key advantage of using this system over regular IBE is that if the security of one PKG is compromised, it does not necessarily affect the security of the whole system [1]. Similarly, when recovering from a security compromise it is easier for the system to recover as it need only restore the affected PKGs in the hierarchy.

The concept of HIBE can increase the level of security in an IBE system, through the employment of additional hierarchical PKGs.

2.2 Gmail

On the 1\textsuperscript{st} of April 2004 Google released the beta version of its free webmail service “Gmail” [28]. With a storage capacity of 1GB and the ability to receive 10MB per email message, the 2MB storage capacity of Hotmail and 4MB of Yahoo! Mail were no match for Gmail [27].
Such a large storage capacity meant that service users would never have to file or delete their email messages. Google also promised improved spam filters and efficient mail search mechanisms for all Gmail users [27].

The ease with which a user could search for old messages and the impressive memory capacity led to the almost instant popularity of the new Google webmail service. By 2009, just 5 years after its release, Gmail had 150 million users worldwide [26]. This number continued to grow in more recent years and the current number of active Gmail users is reported to have reached 350 million, as of January 2012 [25].

Trends indicate that Gmail is set to overtake the number of active Hotmail users, last reported to be approximately 350 million users in October 2011 [26] and has already outnumbered that of Yahoo! Mail. Figure 2.d below shows an increase of 200 million users from 2009 to 2012. The trend indicates that the increase in the number of Gmail users will continue in future years.

![The Number of Gmail Users Worldwide from 2009 to 2012](image)

As the increasing popularity of the service is bound to continue in the future, developing a cryptographic product for Gmail seems a worthwhile venture at this time. Especially after reviewing the significant media attention Gmail has received since its release in 2004, with particular focus on users’ privacy and email security.
2.2.1 Privacy and Security Issues

Upon the release of the beta Gmail product, criticism over the lack of user privacy sparked almost instantly. The Guardian NY [27] reported Google’s plan to raise money from its webmail service, “by programming its servers to pick up key words in emails and deliver related advertising in the messages”.

A BBC article from April 5th 2004 [29] listed concerns raised by privacy campaigners about Google’s content targeted advertisement plans, reporting that campaign group “Privacy International” had filed a complaint with the UK Information Commissioner regarding same. Campaigners argued that the scanning of emails was invasion of a user’s privacy, despite assurances from Google that all email content would be closely guarded and only ever monitored by a machine [27].

Another concern raised by “Privacy International” referred to the permanent storage of emails, both sent and received. The Google terms of use reads: “residual copies of email may remain on our systems, even after you have deleted them from your mailbox or after termination of your account” [29]. The idea that Google could keep a record of a user’s email messages, after that user has supposedly permanently deleted the messages, or ceased use of the webmail service, sparked further controversy among privacy campaigners.

Although Gmail has managed to compete with its rivals in spite of these valid concerns, the provision of secure email transmission, in the form of a browser add-on, could somewhat remedy the reputation of the service for lack of user privacy and security which is still of note at present.

2.2.2 Hypertext Transfer Protocol (HTTP)

An understanding of HTTP was necessary in order to develop the Firefox Gmail extension, in order to observe and intercept the incoming and outgoing messages during a Gmail session.
HTTP is an application-level protocol used by browsers to communicate with web servers [33]. It is a stateless request/response protocol that employs extensible semantics and message payloads, similar to Multipurpose Internet Mail Extensions (MIME), to provide flexible interaction with network based hypertext information systems [32]. The first version of the protocol, HTTP 0.9, was adopted for implementation by the World Wide Web (WWW) global information initiative in 1990 [34]. HTTP 1.0 and HTTP 1.1 have now become standard protocols for accessing the internet [43].

HTTP operates through HTTP requests and HTTP responses. The protocol communicates between a HTTP client, also known as a user agent, and a HTTP server, commonly known as the origin server [32]. The user agent establishes a connection to a server with the aim of sending one or more HTTP requests to the server. The origin server accepts connections in order to receive and process HTTP requests and to return the corresponding HTTP responses [32].

In the case of Gmail, the web browser (Firefox) behaves as the user agent communicating with the Gmail web server, which can be viewed as the origin server. They communicate through a series of HTTP requests and corresponding HTTP responses. As HTTP is a memoryless protocol, it processes each new command completely independent of previous commands that have come across the channel.

In HTTP, a Uniform Resource Identifier (URI) indicates the target resource and relationship between resources [35]. The generic format of a URI is shown in the figure below.

```
<scheme name> : // <authority> <path> [ ? <query> ] [ # <fragment> ]
```

*Figure 2.e: Generic format of a Uniform Resource Identifier*

In the case of Gmail the **scheme name** is ‘https’, the **authority** is the host identifier ‘mail.google.com’ and the **path** is ‘/mail/’. All URIs between the Firefox web browser and the Gmail web server begin with ‘https://mail.google.com/mail’.
The query and fragment components of the URI are optional [35], unlike the first three components. Gmail does not make use of the fragment component. The query string is usually a string of key value pairs, joined by ‘=’ and separated by ‘&’. The query string of a URI can vary depending on the method of the URI.

The method of a URI can be one of five: GET, POST, PUT, DELETE and HEAD [32]. The two methods most relevant to this report are GET and POST. The URI format of these methods, along with a more detailed view of HTTP requests and responses are discussed in Chapter 4 of this paper, in Section 4.2 specifically.

In 2008, Google introduced an option to encrypt Gmail sessions using HTTPS (Secure Hypertext Transfer Protocol) [24]. On January 12th, 2010, HTTPS became the default for all Gmail sessions [31]. This announcement came very soon after news that several Gmail accounts, belonging to human rights activists, had been hacked in China [31]. This was bad publicity for a webmail service that had received significant criticism over email privacy in the past.

HTTPS provided two assurances for Gmail users. Firstly, a user could be sure that the page they were looking at was sent in encrypted form and so could not be read by eavesdroppers [31]. Secondly, a Certificate Authority has ensured that the appropriate organisation (namely Gmail) owns the page domain [30]. Google claimed that although HTTPS encryption was known to slow down the speed at which users could send and receive emails, the security benefits associated with HTTPS made this performance issue almost negligible [31].

In 2010, Gmail’s competitors, namely Microsoft’s Hotmail and Yahoo! Mail, did not yet offer HTTPS as an option, let alone use it as default [24]. Although user’s webmail messages remained unencrypted and open to attack, the adoption of HTTPS was a significant step in Gmail’s journey towards becoming a more secure webmail service.
2.2.3 Transmission Control Protocol (TCP)

An understanding of the Transmission Control Protocol (TCP) was needed for this project. TCP is a connection-oriented, end-to-end protocol, fitting into a layered protocol architecture, above the basic Internet Protocol (IP) [36].

The TCP has two interfaces, on one side it communicates with an application process and on the other side it communicates with a lower level protocol, such as IP [36] to send and receive information over the network. The default TCP port for HTTP is 80 [32] if no other port is specified.

HTTP 1.1 employs persistent connections by default. A persistent connection is a connection which is ‘kept alive’ so that multiple HTTP requests can be sent and responses received over the one connection, rather than opening a new TCP connection for each HTTP request and response [32]. The use of persistent connections can greatly reduce internet congestion by cutting down on the number of connections opened and closed, decreasing the load on HTTP servers.

The pipelining of requests and responses provided by persistent connections allows for a number of requests to be sent at a time before the response of each request is returned. Although this is advantageous in terms of efficient communication, it must be ensured that responses are sent in the same order as the corresponding requests [32].

Persistent connections are used by the Gmail webmail service. When a page is loaded, a TCP connection is established between the user account and the Gmail web server, via HTTP requests and the Firefox browser. The GET URI request which establishes this connection is discussed in more detail in Section 4.2.2. The persistent connections are monitored for HTTP traffic between the origin server and user agent. By monitoring the persistent connections between the Gmail service and the Gmail web server, incoming messages, in particular, can be detected, intercepted and decrypted before reaching a user’s inbox. It is noted that persistent connections in the Firefox browser time out after 115 seconds (1.92 minutes) of inactivity, by default.
2.3 The Importance of Email Encryption

Despite the considerable amount of information that is sent over email on a daily basis, some of which is sensitive and supposedly confidential data, very few users apply encryption to their outbound emails. There has been no definitive conclusion made as to why users refrain from using cryptographic applications. Are users unaware of the services available to secure email communication? Are they deterred by the effort and skill required to use the services that are available for utilisation? Section 2.3.2 examines some suggestions that have been made to account for this lack of security measures taken by webmail users in particular.

2.3.1 Data Loss Prevention

In June 2010 the data security firm ‘Proofpoint’ commissioned its seventh annual survey, on outbound messaging and content security issues, fielded by Osterman Research [23]. The online survey had a total of 261 participants, all decision-makers at larger US enterprises, with at least 1,000 employees. The results revealed substantial concerns among large US companies over confidential data lost through email.

The study was designed to investigate the level of concern over content of outbound email in large US companies, the technologies these companies employ to mitigate the risk of data loss, the message-related company policies in place and the frequency of data security breaches experienced by these large organisations.

Compared to previous years, results showed a higher level of concern about data lost through web-based email and the physical loss of mobile devices. 60% of respondents reported concerns over the use of personal web-based email services in the workplace while only 40% prohibit the use of personal webmail during working hours. 67% of respondents said that it was “very important” to reduce the legal and financial risks associated with outbound HTTP Traffic, particularly webmail traffic, in the next 12 months [23].
31% of the companies surveyed admitted that their business had seen a negative impact from the exposure or theft of customer information in the previous 12 months, while 29% had been impacted by improper exposure of intellectual property. One in ten outbound email messages were found to contain risky content or highly confidential information [23].

The results of the survey revealed that manual human monitoring of email messages was the most common method of reducing data loss through email. 37% of respondents employ staff to read or analyse outbound emails, 89% of which employ staff solely to monitor email messages [23]. The failure to use security applications may be due to a lack of trust in existing technologies or lack of drive to educate employees in the use of these technologies.

A notable 66% of the 261 companies surveyed gave high priority to investing in improving the ability to prevent sensitive content from leaving the organisation through email in an unauthorized manner [23]. There is a gap in the market for a product which both monitors email content and encrypts messages accordingly. The development of such a product could aid in reducing the significant amount of data loss experienced by larger companies in today’s business world.

2.3.2 Why Don’t Users Encrypt Email?

The two standard encryption schemes currently employed in email encryption are S/MIME and PGP, discussed in Section 2.1.2 of this report. There have been many studies carried out to determine the main reasons why people do not encrypt their personal and corporate email using these readily available end-to-end security schemes.

Stephen Farrell [22] makes a point that most modern encryption applications are designed upon the same principles as used for information security years ago. In previous years the features of Privacy Enhanced Mail (PEM) were driven by enterprise and legislation, rather than usability.
Farrell suggests that the lack of usability associated with email encryption applications to be one of the main reasons for the low use of these applications by email users [44]. Many studies have explored the usability factor of such applications as PGP 5.0 [18] in 1999 and S/MIME [19] in 2005. The results of these studies further enforce the argument presented by Farrell.

In [18], twelve participants were given use of an email client program with PGP 5.0 preinstalled and told to send sample confidential data through encrypted email using the PGP application, under isolated laboratory conditions. Despite the 90 minutes they had to complete this task, only one third of the participants were capable of encrypting and signing a message correctly, while a quarter of the group exposed their private key accidentally. The conclusion of this study was that the user interface of PGP 5.0, although considered one of the best of the time, was not sufficiently intuitive so as to enable a user to encrypt and decrypt messages correctly [18].

The later studies of S/MIME in 2005 [19] and PGP 9.0 2006 [20] reached similar conclusions. Although some improvements had been made since the study of PGP 5.0, the two applications were found to be lacking in usability. Garfinkel and Miller [20] suggest that email cryptographic applications should be automated and more transparent, reducing the problems caused by user error in key management and basic encryption decisions.

Although these studies were carried out in the earlier years of email encryption applications, there is still a significantly low use of personal, more so than corporate, email encryption in internet communications today. Despite the clear conclusions from the studies [18, 19, 20], few applications have been able to increase the usability of their systems. The Gmail extension presented in this report is almost completely transparent to the user, encrypting and decrypting messages with no need for user interaction once the extension is installed in the web browser.
2.3.3 Legislative Motivation for Email Encryption

In addition to the desire to keep confidential data secure for the sake of protecting business, there are also legal motivations for keeping company and client data secure from attack.

In 1974 the Privacy Act was enacted in the US, followed by the Electronic Communications Privacy Act in 1986 [44]. The latter prohibits unauthorised and intentional interception of electronic communications during the transmission phase [44]. Although this Act is in place, it is difficult to stop attackers from intercepting electronic communications, and so encryption is necessary to protect sensitive data sent via email, that may be vulnerable to attack.

In 1998 the European Union (EU) Directive of Data Protection was issued [44]. Although it was not implemented as law, EU member states were advised to build information security legislation using the Directive as a framework. Under the Security section of the Directive it stated that organisations must provide adequate security to protect the integrity and confidentiality of personal information [44]. This refers to personal information belonging to clients but also to the intellectual property of the company.

The number of business transactions performed via email is on the rise and there has also been an increase in personal transactions completed online, such as the transmission of credit card details or social security numbers. Many users are sending sensitive data via email without taking any security measures [23]. These users are exposing themselves to the risk of being in violation of legislation in the US, Europe and indeed worldwide. The legislation discussed above should motivate users, especially in the corporate world, to use cryptographic email applications on a daily basis.
3. State of the Art

3.1 The Start of Identity-Based Cryptography

In 1984, Adi Shamir, of the RSA public key cryptography standard [3], proposed the concept of Identity-Based Cryptography [4]. A scheme which “enables any pair of users to communicate securely without exchanging private or public keys, without keeping key directories and without using the services of a third party” [4].

The scheme assumes the existence of a trusted key generation centre to distribute a personal smart card to each user joining the network. The smartcard allows the user to encrypt, decrypt, sign and verify messages.

The proposed system resembles the simple IBE systems explained in Section 2.1.3 previously. Instead of generating a random pair of keys, as in PKC, the recipient’s public key is chosen as an arbitrary string which uniquely identifies the recipient in some way [4]. The corresponding private key is generated by a key generation centre (a PKG) and distributed as a smartcard.

There are four algorithms associated with an IBE scheme, first outlined by Shamir. They are; Setup, Extract, Encrypt and Decrypt [1]. **Setup** involves the initialisation of all system parameters, including the PKG master secret. **Extract** involves the calculation of the IBE private key from the master secret and an identity, using system parameters.

**Encrypt** employs the use of the IBE public-key, generated from an identity and system parameters, to encrypt information. **Decrypt** uses the IBE private key extracted to decrypt the encrypted information. The four algorithms are covered in more detail, with reference to the Boneh-Franklin IBE scheme, in Section 3.3.
Shamir could describe the concept of IBC in detail but was unable to construct a practical implementation of an IBC scheme. The first practical and secure IBE scheme was not seen for many years after this initial proposition paper. Although Shamir failed to implement an IBE scheme, he did succeed in constructing a practical Identity-Based Signature (IBS) system.

3.2 Advances in IBE

There were very few advances in the field of IBC between Shamir’s paper in 1984 and schemes proposed in the early 2000s, with no viable concrete solutions for IBC provided for 15 years following the introduction of the concept. There were, however, new mathematical concepts developing in other fields of cryptography in this time that should be noted, as they contribute to the mathematical foundations of many modern IBC schemes.

In 1998, Law et al. [45] proposed a two-pass protocol for authenticated key (AK) agreement, to be used in PKC. It was an AK agreement system which could be modified to operate in an arbitrary finite group, such as an elliptic curve group. The scheme was based on the Diffie-Hellman key agreement as described for the Diffie-Hellman public-key cryptographic scheme [2].

In 2000, Certicom Corporation [46] released a paper reviewing the use of arithmetic operations on an elliptic curve, over a finite field in public-key cryptography schemes. Although Elliptic Curve Cryptography (ECC) was used solely in the area of public-cryptography at this time, the research of the mathematical methods carried out contributed heavily to the construction of the first practical IBC scheme, the Boneh-Franklin IBE scheme.
3.3 The Boneh-Franklin Scheme

In 2001, Dan Boneh and Matt Franklin published their paper “Identity-Based Encryption from the Weil Pairing” [5], said to be the first successful implementation of an IBC scheme. The Boneh-Franklin (BF) implementation is the basis of the IBE system applied in this project and detail of the scheme is outlined in the following sections.

3.3.1 Mathematical Foundation

The Boneh-Franklin scheme is based on elliptic curves over a finite field [1]. The security of the system is based on a natural analogue of the Computational Diffie-Hellman (CDH) assumption defined in [2]. Boneh and Franklin [5] claim that their IBE system can be built from any bilinear map as long as the CDH problem is hard. This system uses the Weil Pairing on an elliptic curve.

The mathematical foundation of the Boneh-Franklin IBC scheme will not be covered in detail in this report.

3.3.2 Operation

The operation of the BF scheme consists of the four randomised algorithms briefly outlined in Section 3.1; Setup, Extract, Encrypt and Decrypt. A brief description of each of these algorithms is included below, with reference to the BF scheme.

Setup: The inputs to the algorithm are a security parameter, $k$, an elliptic curve $E$ and plaintext of bit length $n$. The algorithm outputs a set of public parameters, $\text{BFParams} = (G_1, G_T, G, e, n, P, sP, H_1, H_2, H_3, H_4)$ and a master secret $s$.

Extract: This algorithm takes as input a string $ID$, representing an identity, and the set of public parameters, $\text{BFParams}$. The output of the algorithm is the private key $sQ_{ID}$.
Encrypt: The inputs to this algorithm are a plaintext message $M$, of length $n$ bits, the string ID and the set of public parameters BFParams. The output is ciphertext $C = (C_1, C_2, C_3)$.

Decrypt: Takes as input ciphertext $C = (C_1, C_2, C_3)$, the set of public parameters BFParams and private key $sQ_{ID}$. The output of the algorithm is a plaintext message $M$, or an error condition.

The BF paper [5] also suggests the use of HIBE, explained in section 2.1.4, as an additional measure to protect the integrity of the master secret. In HIBE the Setup algorithm becomes two algorithms, Root Setup, to generate the public parameters for the PKG at the highest level of the PKG hierarchy, and Lower-Level Setup, to generate the public parameters required by all other PKGs in the hierarchy.

3.3.3 Level of Use

The Boneh-Franklin scheme has been the basis for many IBE schemes which have been developed since its publication in 2001. Dan Boneh went on to publish several papers with Xavier Boyen [1], including “Efficient Selective-ID Secure Identity Based Encryption Without Random Oracles ”, building upon the IBE scheme first developed in his paper written with Franklin.

A real world application of the scheme is discussed in Section 3.4.3 when examining the data protection products supplied by Voltage Security Inc, based upon the Boneh-Franklin IBE implementation.

3.4 Existing Email Cryptographic Products

There are many email encryption products on the market, available for implementation at this time. Customers can decide between transparent products, that encrypt in the background of an application and user interactive products that require configuration of settings and generation of keys performed by the user.
3.4.1 Email Client Encryption

Encryption applications installed into email clients such as Microsoft Outlook and Windows Mail have been available for purchase for many years now, for use in corporate environments in particular. Email client applications currently available include ‘eSecure Mail’ [11], ‘FortiMail’ [12] and ‘Comodo SecureMail’ [13].

The operation of both ‘SecureMail’ and ‘eSecure Mail’ is built upon a PKC scheme, as outlined in Section 2.1.2. The first uses an S/MIME framework for encrypting and decrypting email, while the second offers an option for both PGP and S/MIME. ‘eSecure Mail’ intercepts messages at the gateway to the server [11] and encrypts or decrypts accordingly. It is transparent to the user. However, access to the mail server is required for interception, an access that is not granted in webmail systems such as Gmail.

The overhead associated with PKC is not as significant in a corporate environment where the duty of generating key pairs for all email users can be delegated to a single trusted body within the company. For this reason PKC operates more efficiently when applied to email client communications in large organisations, than it would in webmail encryption.

The third product listed is ‘FortiMail’. This application is based on an IBE scheme. The suppliers of the application, ‘FortiNet’, offer a service with “no need for certificate or key management for end-users, and no need to install additional hardware or software” [12]. The suppliers offer an encryption scheme with transparent operation. The absence of key management requirements is an appealing aspect for corporate customers wishing to automate the email security process when employing email client communications.
3.4.2 Webmail Encryption

There are several web browser plug-in products available for webmail encryption. Three such applications are ‘GPG4Browsers’ [14], ‘FireGPG’ [15] and ‘Freenigma’ [16], all built upon a PGP cryptographic scheme.

‘GPG4Browsers’ was developed by German security firm Secure Labs [14]. A Google Chrome extension for Gmail, it is based on the OpenPGP standard. As of November 2011 the product was not yet ready for use by the general public, though beta versions are now available.

‘Freenigma’ is a Firefox plug-in for Hotmail, Gmail and Yahoo! Mail, first released in August 2006. ‘FireGPG’ is also an application for Firefox web browsers. In both cases PKC is employed for message encryption and decryption.

Weaknesses of these schemes include the need for public and private keys to be stored in memory on the user’s computer, keys which have to be manually generated by the user. Additionally, the “web of trust” concept used by PGP applications is not well suited to webmail environments where users do not know, and therefore cannot trust, all users on the network. This report suggests that PGP is not a sufficient standard for webmail cryptography and proposes a more secure means of webmail encryption.

3.4.3 Voltage Security Inc

In 2002, Dan Boneh, of the Boneh-Franklin IBE scheme [5], founded the data security company Voltage Security Inc [10]. Providing IBC solutions for email clients and webmail, Voltage Security has produced a service adopted by numerous corporate bodies. Representatives at international company Kodak state: “It’s amazing that even though usage is quite high, we’ve had fewer than 10 calls to the help desk, worldwide, since we went live” after implementing Voltage SecureMail in 40,000 employee email accounts [10].
The interface of the Voltage IBE Toolkit is programmed in C language, providing a high level interface for security implementation. Solutions are designed for use in Linux and Windows operating systems, with no support for MAC OS X [17]. Voltage Security products are FIPS (Federal Information Processing Standard) 140-2 Level 1 certified [17].

![Voltage Security Logo](image)

**Figure 3.a: Company logo of email encryption product provider Voltage Security Inc**

IBC has allowed the company to provide transparent, efficient data encryption for enterprise, and webmail encryption for personal and corporate use. Voltage Security is well known for email client solutions more so than the webmail products it provides. The company’s IBC services implement encryption at the gateway, at the network edge [17] rather than within the web browser being used. There is currently no Voltage solution for Gmail users, prompting the development of this Final Year Project.
4. Design and Implementation

The ‘Secure Gmail’ system was designed as two separate system components. A Firefox extension was first built and programmed to observe the behaviour of Gmail HTTP traffic.

Once the behaviour was understood and verified, the JavaScript code was extended to intercept the HTTP traffic to modify the contents of incoming and outgoing Gmail messages in some manner (reversing the message text for example).

Upon achieving this level of operation, the code which simulates an IBC scheme, programmed in C++, was amalgamated with the JavaScript code to encrypt and decrypt data as required. The separate design components are discussed in more detail in the following sections.

4.1 Building a Firefox Extension

The first step in designing the ‘Secure Gmail’ product was to develop a Firefox Add-On. In Mozilla applications, including the Firefox internet browser, add-ons are installed as .XPI files, using the Add-On Manager [41]. Certain files must be included in a package in order for it to be installed and the required format of these files is specified by the Mozilla Developer Network [41].

In modern versions of Mozilla Firefox, the Add-On Manager is opened by ‘Ctrl+Alt+A’, when the web browser is open. An add-on can then be installed using the drop down menu shown in the Figure 4.a below and selecting “Install Add-On From File”.

27
Once the extension is installed, it will be listed in the Add-On Manager. The extension can be removed or disabled at any time as shown in Figure 4.b below. Following installation, the browser must be restarted before the changes can take place.

When the extension is installed, the package software seamlessly encrypts and decrypts Gmail messages sent and received through the Firefox browser.

4.1.1 XPI file

A Cross-Platform Installer (XPI) Module is a ZIP file used to install packages, utilising the XPInstall Mozilla technology [41]. When an XPI file is passed to the Add-On Manager, XPInstall automatically interacts with the installation instructions contained in the XPI to install the software.

The XPI file containing the software of the Firefox extension for Identity-Based Encryption of Gmail is named ‘extension.xpi’. This file contains a number of other files and folders essential to the operation of the Firefox extension. The location and hierarchy of these files within ‘extension.xpi’ is shown in Figure 4.c.
The purpose of each of these files is described in the following sections, with focus on the required content within the files and the appropriate coding languages.

### 4.1.2 XUL file

XUL (XML User Interface Language) is the XML-based language used to build Mozilla applications [37]. XUL files specify the layout of XUL application elements such as windows, labels and buttons on a User Interface (UI). In this particular XUL file the UI is transparent and so is not visible to users.

The first line required in a XUL document is the XML declaration. This information indicates the version of XML used and must be included in order for the file to be valid. The second line applies a Cascading Style Sheet (CSS), overlay.css in this case, defining the appearance of XUL elements when applied to a web page.

The third line is a DOCTYPE statement, defining the XML elements to be applied when the extension is installed. Graphical elements such as ‘window’, can be included in the DOCTYPE statement. In ‘overlay.xul’, the XML element to be applied is an overlay.

Overlays are used to describe extra content for the UI and are applied while the XUL is being loaded [37]. The overlay “helloworld-overlay” identifies two JavaScript files to be included in the UI, ‘observe.js’ and ‘jquery-1.7.1.min.js’.
The overlay adds an Event Listener to register the object HttpRequestObserver, within ‘observe.js’, upon a page load. HttpRequestObserver implements a series of functions in the JavaScript file ‘observe.js’, in order to examine all HTTP requests associated with the page load.

```xml
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://helloworld/skin/overlay.css" type="text/css"?>
<!DOCTYPE overlay SYSTEM "chrome://helloworld/locale/overlay.dtd">
<overlay id="helloworld-overlay" xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
  <script src="jquery-1.7.1.min.js"></script>
  <script src="observe.js"/>

  <![CDATA[
    document.addEventListener('load', onPageLoad, true);
    function onPageLoad() {
      HttpRequestObserver.register();
    }
  ]]>}
</script>
</overlay>
```

Figure 4.d: XUL file ‘overlay.xul’

The fifth line in ‘overlay.xul’ indicates that there is only one XML namespace (xmlns) to be used in this extension. Separate namespaces are important to define when more than one XUL file is used in a particular XPI package.

### 4.1.3 Manifest file

A manifest file describes a package and maps its location on disk to a chrome URL [40]. When Firefox starts up, the manifest files in the chrome directory are examined to determine what packages are due to be installed. To install a package the corresponding manifest file is placed in the chrome directory.
The following code specifies the package name ‘helloworld’ and the file path ‘/content’. The next two lines define the chrome URLs used to include the two XUL files, ‘browser.xul’ (implements appropriate browser layout) and ‘overlay.xul’ (described in Section 4.1.2).

```
content helloworld content/
overlay chrome://browser/content/browser.xul
chrome://helloworld/content/overlay.xul
```

**Figure 4.e: Manifest file ‘chrome.manifest’**

The ‘chrome.manifest’ file is located in the top level of the XPI file.

### 4.1.4 RDF file (Install Manifest)

An Install Manifest is the file that the Add-on Manager uses to determine information about an add-on as it is being installed [38]. It is written in RDF/XML (Resource Description Framework/Extensible Markup Language) format [39], must be named ‘install.rdf’ and must be located in the top level of the XPI file. This RDF file gives details about a Firefox add-on such as the name of the add-on, the versions of Firefox that the add-on is suited to and the name of the creator.

Similar to the XUL file, the install manifest starts with an XML declaration, followed by the listing of relevant XML namespaces. The file then lists the ID of the relevant Mozilla application (Firefox ID = {ec8030f7-c20a-464f-9b0e-13a3a9e97384}), the current version of the add-on and ‘<em:type>2</em:type>’ specifies that the add-on is in the form of an extension.

The Mozilla application for which the add-on is intended (Firefox) is then specified again, along with the minimum and maximum versions of the Firefox that the add-on is suited to. Finally, details of the add-on are provided including the name of the add-on, a description of the add-on, the name of the creator and a homepage URL for user support.
The information provided by the Install Manifest gives users a brief description of an add-on before they decide if this is the add-on they are looking to install.

```xml
<?xml version="1.0"?>
<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    <Description about="urn:mozilla:install-manifest">
        <em:id>{ec8030f7-c20a-464f-9b0e-13a3a9e97384}</em:id>
        <em:version>1.0</em:version>
        <em:type>2</em:type>
        <em:targetApplication>
            <Description>
                <em:id>{ec8030f7-c20a-464f-9b0e-13a3a9e97384}</em:id>
                <em:minVersion>1.5</em:minVersion>
                <em:maxVersion>7.0.*</em:maxVersion>
            </Description>
        </em:targetApplication>
        <!-- Front End MetaData -->
        <em:name>FYP Http Observer</em:name>
        <em:description>IBE extension for Gmail</em:description>
        <em:creator>Aisling</em:creator>
        <em:homepageURL>http://www.tcd.ie/</em:homepageURL>
    </Description>
</RDF>
```

**Figure 4.f: Install Manifest RDF file ‘install.rdf’**

### 4.1.5 JavaScript files

JavaScript is a cross-platform, object-oriented scripting language. JavaScript is a small, lightweight language; it is not useful as a standalone language, but is designed for easy embedding in other products and applications, such as web browsers [42].

There are two JavaScript files implemented in this extension. The first, ‘jquery-1.7.1.min.js’, allows the second JavaScript file to avail of the JQuery JavaScript library of functions. The need for this library will be explained in Section 4.3.1, where the function of ‘observe.js’ is discussed in more detail.
The second JavaScript file, the most important file in terms of this project, is ‘observe.js’. This file contains the code which performs the interception of HTTP traffic, the communication with an IBE client for encryption and decryption of messages, and the subsequent replacement of HTTP traffic between the Firefox Browser and Gmail Web Server. The approach taken to all three of these tasks, implemented by the JavaScript code, is described in Sections 4.2 and 4.3.

### 4.2 Intercepting HTTP Traffic

As mentioned previously the encryption and decryption of email messages was achieved through the interception and modification of HTTP requests and responses. HTTP traffic associated with incoming emails differs from that of an outbound email. For this reason the methods of interception are discussed separately below.

Every HTTP Uniform Resource Identifier (URI) observed by this function is seen as having three parameters that provide information about the URI. These parameters are aTopic, aSubject and aData. The first gives information about the motivation of the URI, the second gives information about the URI structure itself, while aData contains the data within the HTTP requests and responses associated with the URI.

In ‘observe.js’, the HttpRequestObserver is governed by four JavaScript functions, used to intercept HTTP traffic; register, observe, onModifyRequest and onExamineResponse.

```javascript
register : function()
```

The function ‘register’ initiates the HttpRequestObserver when the aTopic parameter of an observed URI is either ‘http-on-modify-request’ or ‘http-on-examine-response’. That is, HttpRequestObserver is called to observe any URI which indicates a new request or changing response. A call to the ‘register’ function is made in the XUL file ‘overlay.xul’, Figure 4.d, upon a page load.
The function ‘observe’ examines the aTopic parameter passed to it and calls the appropriate function to further investigate the URI. If the aTopic of the URI is ‘http-on-modify-request’ it may indicate a new outbound Gmail message and so the aSubject of the URI is passed to the function ‘onModifyRequest’. If aTopic is ‘http-on-examine-response’, it may indicate a new incoming message and the aSubject, from which the URI can be found, is passed to the function ‘onExamineResponse’.

The task for the ‘onModifyRequest’ function is to find URIs specifically associated with outbound email messages. The method used to find such URIs and manipulate the information accordingly is described in Section 4.2.1.

The task of the ‘onExamineResponse’ function is to find URIs specifically associated with inbound email messages. The method used to find such URIs and manipulate the information accordingly is described in Section 4.2.2.

The HttpRequestObserver also employs several Trace Listeners and HTTP URI content-related functions to modify the HTTP URI traffic and to replace the original data with the modified data. These additional functions will be briefly discussed in the following sections 4.2.1 and 4.2.2.

### 4.2.1 Outgoing Webmail Messages

POST is one of the five URI methods discussed in Section 2.2.2. It is used to petition the origin server to accept the entity enclosed in the request, as a new subordinate of the resource identified by the Request-URI [32]. The POST method can perform many actions involving the posting of data. The function performed is specified by the Request-URI and determined by the server [35].
Any URI associated with an outbound email is intercepted and modified within function ‘onModifyRequest’. The POST URI associated with outgoing Gmail messages sends the message data to the Gmail web server, in addition to posting the message data to the ‘Sent’ folder of the relevant Gmail account. The regular expression in Figure 4.g illustrates the structure of the URI of interest. IT is compared to all POST URLs observed to find the relevant message data among a large amount of HTTP traffic.

```
hits://mail.google.com/mail/.*\?ui=2&ik=f.&rid=mail%3Asd.* &at=AF6bup.*&view=up&act=sm&jsid=*&pcd=1&mb=0&t=c
```

Figure 4.g: Regular expression for POST URI corresponding to an outbound email

Once this URI is observed, the information contained in the post data of the HTTP request is extracted, similar in format to the data shown in Figure 4.h.

```
to=%22fyp%2002%22%3Cfinayearprojb%40gmail.com%3E%2C%20&cc=&bcc=&subject=Subject%20Line&body=Message%20Body%3Cbr%3E&ishtml=1&nowrap=0&draft=undefined&bwd=&rm=undefined&cans=&ctok=&ac=%5B%5D&adc=
```

Figure 4.h: Sample of the contents of the post data within the relevant POST URI when an email with Subject Line: “Subject Line” and Message Body: “Message Body” is sent to email address: finalyearprojb@gmail.com

The string of data is first split into key/value pairs (split at each ‘&’). The keys are then stored in one array and the corresponding values in another. The keys array is searched for fields ‘to’ and ‘body’. The corresponding string value for key ‘to’ is found and URL decoded. URL decoding converts URL encoded strings into a legible string. The decoded string is the identity (email address) of the email recipient and is stored for the email encryption process.

The value corresponding to key ‘body’ is also URL decoded. This is the message body of the email and must be encrypted before being sent to the Gmail web server. The encryption of the message is explained in Section 4.3.2.
Once encrypted the new message is URL encoded, the post data string in Figure 4.h is reconstructed with the new message body and the post data is uploaded back into the HTTP channel. The channel remains unaware of the changes that have been made to the HTTP request URI.

4.2.2 Incoming Webmail Messages

The GET method of a URI retrieves the entity from a resource, specified by the Request-URI [32]. The persistent connection between origin server and user agent, as described in Section 2.2.3, is initialised by a particular GET request, the URI of which fits the regular expression shown in Figure 4.i. The request is made by the Gmail user agent to establish a connection, also known as a comet-type connection [32], with the web server. Data is sent asynchronously from the server to the web browser, through this connection.

https://mail.google.com/mail/u/0/channel/bind\?.*VER.*

Figure 4.i: Regular expression for the GET bind URI which establishes the connection between server and client, and keeps it open to listen for requests and responses

Once the initial request to open the connection is made, the contents of the GET URI response expands over time. If there is no data to be delivered from server to browser, a ‘noop’ is sent, the format of which is shown in the first four lines of Figure 4.k. The first integer value indicates the length of the segment of data. The second integer keeps track of the number of segments sent over the connection.

When an email is sent from the Gmail web server to a user’s account, the GET URI response grows to include data similar to the segment shown below, beginning with integer length 185 in the case of Figure 4.k. The data string includes the subject line of the email and the email address of the sender. It also includes a reference number, shown in bold font, which becomes important in the interception of the incoming email messages.
Interception of incoming webmail is performed in the 'onExamineResponse' function. Upon observing the relevant HTTP GET request, a tracing listener is placed upon the response of the URI. The tracing listener contains three functions; onStartRequest, onDataAvailable and onStopRequest. When the connection is first established ‘onStartRequest’ is called and when that connection is closed, ‘onStopRequest’ is called to notify the listener.

When data is added to the response, the ‘onDataAvailable’ function is called. Within this function the new data is tested to see if it is the relevant data segment associated with an incoming message. If the data passes this test, the reference number and subject line of the message are stored in variables and the data is sent to the browser, unchanged.

```plaintext
18 [[20,['noop']
 ]
 ]
185 [[21,['m'],['nm','0','13606a2b01906db9','new','fyp 02','0',['^cob-processed
 gmr','^i','^all','^smartlabel_personal','^u']
 ,"Subject Subject Subject"],"","finalyearprojb@gmail.com"]
 ]
 ]
18 [[22,['noop']
 ]
 ]
```

Figure 4.j: A sample of the GET bind URI response when an email with Subject Line: “Subject Subject Subject” and Message Body: “Message Message Message” is received from Email Address: finalyearprojb@gmail.com

Upon the expansion of the HTTP GET response, if an incoming email is pending, a POST URI is observed, containing the reference number corresponding to the most recent GET response data segment. This POST request sends the message body to the Gmail user’s ‘Inbox’ folder. The regular expression used to find this URI is shown in Figure 4.k.
The information about the incoming email can be found in the response of the POST URI. When this URI is detected by the ‘onExamineResponse’ function, another tracing listener is added to the response. Once again it is the ‘onDataAvailable’ function, associated with this particular tracing listener, which examines the data of the response. In this case data is pushed into the response only once.

From the large amount of response data, the string shown in Figure 4.1 is extracted, located using the same reference number as before. The message body of the email is extracted and stored as a string. The decryption of the message body is explained in Section 4.3.3. Once decrypted the new message body is uploaded to the browser using a HTTP URI identical to the original POST URI apart from the message body. This concludes the description of HTTP traffic interception and modification.

Figure 4.1: A sample of the contents the POST URI response when an email with Subject Line: “Subject Subject Subject” and Message Body: “Message Message Message” is received. The Reference Number: “13606a2b01906db9” corresponds to that contained in the previous GET URI response content sample

```
["ms","13606a2b01906db9","",4,"fyp 02 \u003cfinalyearproja@gmail.com\u003e","fyp 02",finalyearproja@gmail.com ,1331551187000,"Message Message Message", ["^all","^i","^iim","^io_im","^io_imc2","^smartlabel_personal","^u"] ,0,1,"Subject Subject Subject",["13606a2b01906db9",["fyp 01 \u003cfinalyearproja@gmail.com\u003e"] ,[] ,[] ,[] ,"Subject Subject Subject","Message Message Message\n",[]]
```

Figure 4.k: The regular expression of the POST URI which contains the Reference Number from the previous GET URI response segment (Figure 4.i), corresponding to an incoming email.

https://mail.google.com/mail/.*\?ui=2&ik=f.*&rid=.*&view=cv&th ='+reference+'&prf=1&_reqid=.*&nsc=1&mb=0&rt=j&search=inbox
4.3 Communicating with the IBE Client

An offline PKG, developed by Karl Reid, of the School of Computer Science, Trinity College Dublin, was used for this project. The IBE Client was developed for Karl’s final year project in 2011, “Applying identity-based encryption to secure DNS updates” [43].

In this C++ implementation the “MIRACL” (Multiprecision Integer and Rational Arithmetic C/C++ Library) software library is utilised [43]. This software was published by Shamus software, developed by a professor at Dublin City University (DCU) and is free for educational use [6]. MIRACL is a cryptographic library which is an example implementation of the Boneh-Franklin IBE scheme.

The interface developed for Karl’s project [43] allows a user to run an offline PKG to generate the necessary IBE parameters and also allows the user to set up a local server to act as an IBE client for the purpose of encrypting and decrypting strings. The PKG tool and server initialisation is described in Section 5.1.

4.3.1 XML HTTP Requests and Responses

Communication between the ‘Secure Gmail’ extension and the IBE client is accomplished using the XMLHttpRequest (XHR) object in the ‘observe.js’ file. XHR is used to send HTTP requests directly to the Firefox web server and to load the server response data directly back into the script [32]. The initial set up of the IBE client servers is described in Section 5.1 of this report.

XHR employs asynchronous communication, which caused problems when used in the JavaScript code as the message body, to be encrypted or decrypted, was returned to the browser before the XHR response was returned with the new message body. This lead to the message body remaining unchanged when sent and received. This problem was solved by utilising the JavaScript library jQuery.
The JavaScript file ‘jquery-1.7.1.min.js’ was added to the extension folder and ‘jQuery.ajax’ used within ‘observe.js’ to set the asynchronous property to ‘false’. When the state of the XHR reaches state 4, the response is read.

### 4.3.2 Encryption

The XMLHTTPRequest sent to the local IBE client server to encrypt a message is shown in Figure 4.m.

```
http://localhost:<port>/encrypt?message=<plaintext>&id=<email_address>
```

*Figure 4.m: XMLHTTPRequest sent to the local IBE client, through the specified port to encrypt a plaintext string using an email address as the recipient’s identity*

The port parameter corresponds to the port specified for the IBE client server of the user sending the message, instigated in the initial setup of the server (Section 5.1). It is the port through which the extension communicates with the IBE client. The plaintext parameter is the message body string to be encrypted and the email_address parameter is the email address of the intended email recipient.

The IBE client uses the email address and the public parameters generated by the PKG tool, also described in Section 5.1, to generate the recipient’s public key. Using this key the message body is encrypted into ciphertext. The response to this XHR, sent from the local server (IBE client) is the encrypted message body, in the form of ciphertext.

### 4.3.3 Decryption

The XMLHTTPRequest sent to the local IBE client server to decrypt a message is shown in Figure 4.n.

```
http://localhost:<port>/decrypt?ciphertext=<ciphertext>
```

*Figure 4.n: XMLHTTPRequest sent to the local IBE client, through the specified port, to decrypt a ciphertext string*
The port parameter corresponds to the port specified for the IBE client server of
the message recipient, instigated in the initial setup of the server (Section 5.1). The
ciphertext parameter is the encrypted message body received from the Gmail server
and extracted by the function ‘onExamineResponse’.

The IBE client uses the recipient’s email address and the master secret
generated by the PKG tool, described in Section 5.1, to generate the recipient’s
private key. Using this key the message body is decrypted into the original plaintext
message. The response to this XHR, sent from the local server (IBE client) is the
decrypted message body.
5. Implementation and Evaluation

The ‘Secure Gmail’ Firefox extension can be used with Firefox browsers from version 1.5 to 10.0, suited to operating systems Windows OS and Linux. However, the IBE client used for encryption and decryption was designed for Linux operating systems and so most implementation and evaluation was completed using the Linux OS distribution, Ubuntu.

The majority of the implementation and testing of the extension was carried out on a Dell™ Latitude™ E5410 notebook computer. Testing was executed in Firefox version 8.0 originally, and more recently in version 10.0.

5.1 Initial Setup

To run ‘Secure Gmail’, the extension must be installed in the Firefox browser. This is done using the Add-On Manager (Ctrl+Alt+A), mentioned briefly in Section 4.1. Once ‘extension.xpi’ is added, ‘Secure Gmail’ is in place to encrypt incoming emails and decrypt outgoing emails during all Gmail sessions which take place within the Firefox profile where the extension is installed.

Before encryption and decryption can take place, the PKG must generate a keyfile for each identity and the public parameters required to generate the public key from the recipient’s identity (email address). The generation of the keyfile and naming of that keyfile is done via the Terminal, running command ‘./pkgTool64’.

The generation of public parameters performed by the PKG tool can be seen in the screenshot of the Terminal labelled Appendix A. The prompt by the PKG to enter an identity in order to generate a keyfile for the user can be seen in the last line of this screenshot. Appendix B displays a screenshot of the Terminal after keyfiles have been generated for both email addresses (finalyearproja@gmail.com and finalyearprojb@gmail.com), including the renaming of the keyfiles (‘a’ and ‘b’). At this stage the PKG has finished generating the parameters needed for the IBE system.
As this project requires an offline IBE client, two servers are initiated to act as an IBE client for each of the Gmail users (finalyearproja@gmail.com and finalyearprojb@gmail.com). If these servers are not running, the Firefox extension cannot make a call to the IBE client to encrypt or decrypt data.

The parameters required by the IBE client in order to perform cryptographic operations are shown in Figure 5.a below. The second line shows that ‘server.sh’ takes three inputs; user identity (email address), keyfile (as generated by PKG) and port (for communication between the IBE client and the Firefox extension).

![Screenshot of the Terminal displaying the necessary three input parameters for ‘server.sh’](image)

Appendix C shows a screenshot of the Ubuntu Terminal with an IBE client server running, given parameters identity ‘finalyearproja@gmail.com’ with corresponding keyfile ‘a’ and the port through which the user will communicate with the IBE client, ‘4242’. Appendix D displays the server running for ‘finalyearprojb@gmail.com’, keyfile ‘b’ through port ‘4243’.

The launch of the two servers is the last step in the set up, before implementation can take effect. The Firefox add-on is now capable of communicating with the web browser on one side and with the IBE client on the other. An outbound email from a Gmail account will be encrypted and inbound messages decrypted by the ‘Secure Gmail’ extension.

### 5.2 Testing and Evaluation

In order to test the extension, and consequently the IBE C++ code, two Gmail accounts were created (‘finalyearproja@gmail.com’ and ‘finalyearprojb@gmail.com’). The accounts were opened in separate Firefox profiles, both with the ‘Secure Gmail’ extension installed.
Email messages were sent from one account to the other to test the encryption of the 26 alphabetic letters and numbers 0-9. Once the operation of the extension was verified as correct for these characters, testing of new lines, commas and periods began, as the structure of most emails contain a greeting, a comma and then a new line for the remainder of the message.

Special characters such as ‘!’, ‘&’ and especially ‘%’, as it is repeatedly used in the URL encoding of a message body, were then tested. The symbols were included in email messages to verify that the ‘Secure Gmail’ extension worked correctly upon encountering such characters, and also to ensure that the IBE client could process the special characters in order to encrypt and decrypt the message.

The extension was found to operate correctly on computers running Windows operating systems and Linux operating systems. The IBE client, designed for Linux could not be tested on a Windows machine, but was found to work correctly in Ubuntu when passed correct input parameters from the extension. If passed an unexpected input, asked to decrypt plaintext for example, the local server will crash.

If the server crashes the ‘Secure Gmail’ extension cannot decrypt the incoming message and place the original message text into the message body. The most recent testing of the completed application, however, has shown 100% consistently correct operation and zero server crashes.

Testing during development was carried out using the Mozilla Web Developer tool Error Console (Ctrl+Shift+J).

5.3 Regular Use and Real World Implementation

In the real world implementation of the “Secure Gmail” service, the PKG would operate online, rather than operating locally on the user’s computer. It would be implemented by a trusted body on the internet. The services of the PKG could then be obtained by all ‘Secure Gmail’ users, through an XMLHTTPRequest to an online server, rather than a local host.
As the PKG is controlled by an online body, users must authenticate themselves to the PKG server before receiving their private key. A simple interface asking for a username (the user’s email address in this case) and password (chosen and stored when the user first installs the extension) can achieve this authentication.

5.4 Potential Problems/Barriers

The Private Key Generator, considered to be more susceptible to attack than the Certificate Authority of a PKC scheme [7], must be kept secure at all times. Any compromise of the PKG master secret would compromise the security of all past and future private keys generated. Employing the model of the Hierarchical Identity-Based Encryption (HIBE) scheme, outlined in Section 2.1.4, could ensure higher protection of the master secret and therefore better security for the service users.

The need to authenticate to the online PKG may seem tedious to users and may dissuade them from using the extension. This requirement may also act as a positive attribute to the ‘Secure Gmail’ application. If the add-on operates completely transparently once installed with no need for authentication, users may not feel it is actually providing secure email communication.

Douglas Maughan of US Department of Homeland delivered results of a trial of both PKI and IBE [8]. It was found that participants found it easier to use IBE than to use than PKI but gave a higher rating to PKI when asked which scheme made them feel more secure in sending encrypted email. Maughen attributes this to the fact that it is simpler to implement IBE, and so users may feel the process is too easy to be encrypting their messages correctly.

A potential barrier to the operation of the extension is recognised. The JavaScript code for ‘Secure Gmail’ is highly dependent on the structure of the HTTP traffic between the web server and the browser. If Gmail were to change the structure of the URI that HTTPRequestObserver listens for, the extension may not function correctly.
In an ideal situation Google would adopt the ‘Secure Gmail’ service for all its Gmail users and take on the duty of modifying the source code when changes to the HTTP traffic layout are to be made. At this point, the only realistic solution to the problem of varying URI structures is to monitor any changes made by Google and modify the source code of the extension accordingly.

When a message is sent to a Gmail mailing list, the routing of the message occurs at the Gmail web server, rather than in the web browser. This causes a problem when trying to encrypt the message individually for each recipient on the mailing list, as the email addresses are not available in the web browser. Unless web server access can be granted to the ‘Secure Gmail’ extension, to retrieve all relevant email addresses, the emails cannot be encrypted using IBE. This is a major barrier to the system and will prevent it from being implemented in real world applications in the near future.
6. Conclusion

Upon completing the ‘Secure Gmail’ project, a Firefox extension exists that, when installed into the web browser, employs Identity-Based Encryption to successfully encrypt and decrypt Gmail messages. The extension works transparent once installed by the user. It does not require complicated user interaction.

6.1 Learning Outcomes

This project gave me the opportunity to learn how to write code in JavaScript, a programming language I had no experience in using, prior to the project. I also obtained a useful skill in learning how to create and debug a Firefox extension, gaining exposure to the web developer tools available for Mozilla applications.

Having had a vague idea of the function of the Hypertext Transfer Protocol and the Transmission Control Protocol, this project allowed me to become more familiar with web based applications, and the role these protocols can play in running them.

The project allowed me to extend my knowledge of cryptographic systems, specifically IBC, and implement a practical IBC system, to observe cryptography in practice. I have become more aware of the importance of keeping data secure in transmission and would like to continue working on projects such as this in the future.

One of the most important lessons I have learnt from undertaking this project is that it is good coding practice to program and test different components of a system separately before joining all components together for final testing. Although this was always the case with smaller projects I have undertaken, working on a project with many different components has illustrated how important it is to test each component in a system separately and thoroughly.
6.2 Difficulties Encountered

Some of the difficulties encountered during the development of the ‘Secure Gmail’ system have been mentioned in earlier sections of this report. The asynchronous nature of XMLHTTPRequest and the corresponding responses lead to some time wasted in debugging the code, as I was unfamiliar with the asynchronous property of XHR communication. It was under the guidance of Karl Reid, a postgraduate student in the School of Computer Science, that the problem was eventually resolved.

A challenging aspect of the project was obtaining a consistent HTTP structure upon which to base the code for HTTP traffic interception and modification. The HTTP URIs and content could vary from one Gmail session to another, sometimes depending on what mail folder the user is in at the time (eg. the Sent folder). Developing code that could detect the various URIs was challenging but ultimately achievable.

The initial stages of testing the Firefox extension’s communication with the IBE client, lead to the local server crashing every second or third message received. The reason for the crash became clear when the data being sent to the server was observed. As mentioned in Section 5.2, the IBE client is very sensitive and will crash if passed incorrect data, differing from the structure of the input parameters it expects to observe.

In this case the Firefox extension was sending data other than the appropriate ciphertext, to be decrypted by the IBE client, causing the local server for that user to crash as the string could not be decrypted correctly. This further highlighted the need to test components of the system separately, even during the process of combining the system components.

There were no difficulties experienced that severely hindered the progress of the project and all difficulties were overcome in time.
6.3 Future Work

There are many potential applications of IBC to be considered in the area of email and other means of communication on the internet. Extending the functionality of ‘Secure Gmail’ to include the encryption of email attachments, such as PDF documents and images, could give it a significant edge over other cryptographic email products currently on the market.

From the background reading completed during this project, it has become clear that a system that scans the contents of an outbound email and either stops sensitive data from being sent or encrypts the data, using a higher authority ID as a public key, before sending would be well received by email users, especially in the area of corporate email communication. Such a system could be developed by combining the existing IBE functionality of the ‘Secure Gmail’ extension with code to assess the content of an email. The new system could implement semantic study of the language in the email or simply search for strings of a certain format, such as that of a Social Security Number or Credit Card number.

Although email is the most common form of online messaging, social networking sites account for a large amount of non-corporate online communication. An IBE scheme could be applied to social networking messaging services, in a similar fashion to email, to encrypt and decrypt messages sent between users. The arbitrary string representing the recipient’s ID could be their unique username for the site or the email address the user to create their account on the site.

Another future prospect for this particular project would to be to develop similar services for other web browsers such as Internet Explorer and Google Chrome. Firefox is currently considered the best web browser for web development, with Google Chrome not too far behind. Developing an extension for Internet Explorer may be more difficult but would be worthwhile as it is the web browser which has the most usage worldwide [26].
References


Electronic Resources

The CD accompanying this Final Year Project Report includes all relevant project code, the report in PDF format and a text file explaining the contents of the CD (information.txt).

Appendices

Appendix A: Screenshot of the generation of public parameters and master secret by the PKG tool ‘pkgTool64’, used to create public and private keys for an identity

Appendix B: Screenshot of the generation of key files for each user ID (email address), running ‘pkgTool64’ via the Terminal

Appendix C: Screenshot of ‘server.sh’ for ID (email address):
finalyearproja@gmail.com, keyfile: ‘a’ and port: ‘4242’

Appendix D: Screenshot of ‘server.sh’ for ID (email address):
‘finalyearprojb@gmail.com’, keyfile: ‘b’ and port ‘4243’
Appendix A: Screenshot of the generation of public parameters and master secret by the PKG tool ‘pkgTool64’, used to create public and private keys for an identity
Appendix B: Screenshot of the generation of key files for each user ID (email address) running ‘pkgTool64’ via the Terminal

```
asling@ubuntu$:~$ cd Desktop/IBLEDeploy
asling@ubuntu$:~/Desktop/IBLEDeploy$ ./pkgTool64

# SETUP

Random seed used = 816734152
Prime p = 12265572525435727899888094738498578036978512669418591054953923624998044988946190487592852236624827596074877575268502326571443149095015
38637651198396983760187
Prime q = 738758818654514959184241035814151692879690492501
Master private key, s = 145467452770412902623534548996103196736753267912
Point P = (21264646005431449408171708099840376476160944626759333235297578477663069925563549637234710575819056817797546943764109977894487299144200
5323213601943457289995, 34791210006730309219265771505061151768884049952438509979302167756022555223262239601841558833220183128
634162743208166787559209083287990847)
Master public key, Ppub = (4122772679955527442297222689661457638212757434671609480431863323579656787217302233918818641396792199219578391042710
235840369668678201862389229210548519076, 379374502725538947343150699187757697668312328458569692905085679561150360791807428724135133391586125477
947741459525730013182745352360813719035501438400551)
Cubic root of unity = [61313286262176389494445243629429285015489256334709255274769618126499022494739502379662611831241379803741387382315161628572
1575457507692879859198492180093, 10228846772273302275677024651180651980343189631184138852856785552397904572043987853356019418737285131143965424
612254886981283362977769555657837095895939734692]
PKG successfully initialised.
PKG tool running, <CTRL-C> to quit.
Enter an identity to generate a keyfile for the client using that identity: finalyearproja@gmail.com

# EXTRACT

Private key = (0517332993938555645285054634696893976924255321249055119235808432919168063431971613061855111585591424253483262826237944428459678960
329811156842448747088002899158240915721236388352743623161728397651609575863004118404306883528026845429811448488165752961149655642600063389389964
937637455563449309829653174146741961)
key is 5102249915792213039852374263616172839765160957586300411840430688352206240542981440498165752961149655642600063389389964
39296537414146741961
Enter the name of the user to download the keyfile for: finalyearproja@gmail.com
written key file to a
Extract complete.

Enter an identity to generate a keyfile for the client using that identity: finalyearproja@gmail.com

# EXTRACT

Private key = (1142292062138203562318027947616144286234906911064016666038236147914464654616565895868648840824026506979955401390816738093835392142639170
03444449799335226767676767947131, 129646324689528360314053464754361482758548023037377826421517994143410308512224109476212646613412448734014226068087161
17507858985599964704655121651349929063)
key is 12984632468952836031405346475436148275854802303737782642151799414341030851222410947621264661341244873401422606808716175078589855999647046551216513
Enter the name of the user to download the keyfile for: finalyearproja@gmail.com
written key file to b
Extract complete.

Enter an identity to generate a keyfile for the client using that identity: ~C
asling@ubuntu$:~/Desktop/IBLEDeploy$
```

56
Appendix C: Screenshot of `server.sh` for ID (email address): finalyearproja@gmail.com, keyfile: `a` and port: `4242`

```
alsling@ubuntu:~$ cd Desktop/IBEDeploy
alsling@ubuntu:~/Desktop/IBEDeploy$ ./server.sh finalyearproja@gmail.com a 4242
line(key) = 51024991579212308852374236316728397651609575630841814843088352920
0454298144484816575291911649564260806903303989964937634555663449300939295374711463
line(params) = 122665725525435272789988990473484986780097521266941859510459323632
4998044988941904875928223624282759607487752682922057144314989958153805756119899
693760187; 73975851866454591501982416358141589827966462561; 21264468008543194448
8171990840374676169494207593553232207587477636099252653496372470578519850681
77594649376199899447299114429853313904523459728995; 3347991210066739039179220557
71558061151768684840962542308097932012677680230867767979822552325233681941
588332218123288641627382081678579298882780998847; 6133826267271638949944452369249
28501548925633740799255274696181624990224944739052079662618131213798073487632
5110628572157545705708278095919841980093; 10228846772273302725770246511800519
08343896131884285268532954720439875353601941877285131341365482481254648
981238326272773655657837095895709349629; 12; 1; 1601, 12227672973595274422972226896
8463145638211757343462180468341868317276565787271072289918816684139670291922185783
912047102358480389666670168238922921548519067; 3703475077253598473431500908911877
576970668312324855968920065675901150303797807428724213513393186524279477414595
25736013812745352368613713965561430460551
####### SETUP ########
Random seed used = 269091828
Prime p = 1248386631744780845435085591883118160324874357453806911378417638911651878
212836892042246543311545842353603681569579754094919122707198187765835894225220
94971
Prime q = 730570818665451459101842416358141509827966402561
Master private key, s = 4478792782649941168496954658816925662131949227899
Point P = (39535792875965378578579575552348377991687866712174661156511239896299
2486690813029626614239323272343685782936947019126574996049638694112929748768873
6343; 90374961998630391848170308909565301737930316266987382672431684815067557
610995241565015268997633575843008680494909373213129738693969922943782065313
91593071516903345345721314345080554338625470929541153270906617
50223866325852094602, 114366906272527312608379525216914655612593943291656201699
55988934332325536188084877894617731018937216679576780435977699047998849680255242
04287205075046900
Cube root of unity = [e2419431585739842271572597954159555451243672854080658920881
945585259391096418544662112827165577741296553196784778987784769955551530839988882917
92971126104785, 3854383488094281185897193323839786382762659968519282376759463
539311377618639795812457974014503833729678138712661129477186601073323925529397
3262829962]
PKG successfully intitalised.
Server started, press enter to quit...
```
Appendix D: Screenshot of `server.sh` for ID (email address): `finalyearprojb@gmail.com`, keyfile: `b` and port `4243`