2011

An implementation of accountable anonymity

Leonardo Aguilera
Iowa State University

An implementation of accountable anonymity

by

Leonardo Aguilera

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Information Assurance

Program of Study Committee:

Yong Guan, Major Professor

Doug Jacobson

Lei Ying

Iowa State University

Ames, Iowa

2011

DEDICATION

This thesis is dedicated to my mother Dolores, who was always there for me when I needed encouragement, motivation and strength to continue with my education. It is also dedicated to my brother Jesus, who helped me unconditionally whenever I needed his help, and to my sister Claudia, who passed away while I was writing this thesis. Even though both my mother and brother passed away the same year in 2009 and my sister in 2011 they will always be in my heart. Finally, to my remaining sisters, Laura, Lolis, Rosa and Aide for their love and support. Mom...we did it!

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to thank all the people who helped me with this research directly and indirectly for their time, suggestions, discussions and disagreements that later became agreements. Especially, I would like to thank Dr. Yong Guan for believing in me and accepting me in his group without even knowing me. His guidance, patience, support, and his drive to push us to the limit to get the work done on time enabled me to complete this research. I would also like to thank my committee members Dr. Doug Jacobson and Dr. Lei Ying for agreeing to be in my committee. Dr. Guan and Dr. Jacobson thank you for giving me the opportunity to be your teaching assistant, I have learned a lot this past year. I would additionally like to thank my new friends Gang Xu, Wenji Chen and Yang Liu for those interesting conversations about our research and other things, also for laughing at my jokes in the lab when we were working late at night. Also, thanks to my other friends Victor Rios, Carlos Valles, Antonio Cordero, and Jose Martinez who also provided me with moral support to continue with graduate school. Thanks to my managers at Rockwell Collins for giving me the opportunity to work in their departments and friends for inviting me on their lunch groups. Ronald Zozaya, Curtis Topf, Mohammed Waheed, Scott Mickelson, Rob Dahl, Scott Conrad, Rob Bentz, Eric Berg, Paul Kelsen, Lee Keuter, Raj Patel, Stacey Sorowat, Herb Watson, Liz Lujan, Roberto Parga and other people that I missed here.

ABSTRACT

Complex well developed and established Anonymity systems lack *Accountability*. These systems offer unconditional anonymity to their users which can stimulate abusive behavior. Controlling abuse should be equally important as protecting the anonymity of legitimate users when designing anonymous applications. Current anonymity systems are promoted to family and friends, businesses, activists and the media. However, these same systems could potentially be used for: sending offensive email, spam, copyrighted material, cyber warfare, child pornography, pedophiles chatting with kids online or any other illegal activity performed on the Internet. Freedom of speech and the *First Amendment* allows people to express their opinions and choose any anonymity service and by no means will people be forced to use this system. In this thesis, a model that allows an anonymous yet accountable method of communication on the Internet is introduced. The design and implementation is based on a new proxy-re-encryption scheme and a modified onion routing scheme. Techniques for *Accountable Anonymity* are demonstrated by building a lightweight prototype. In this system, users are registered in the system's database in order to use it. In this research, the total network latency is significantly smaller when sending data over the network compared to The onion router (Tor) system which makes it deployable to use at a larger scale system. Also, the *Accountable Anonymity* system's digital forensic mode makes it easier to track the perpetrators.

CHAPTER 1. INTRODUCTION

Highly sophisticated hackers can rent Amazon.com Inc's servers for as low as three pennies an hour to execute cyber attacks and use the Tor anonymous system to hide their tracks in the layers of proxy servers that are spread all over the world. The most recent case is Sony Corp's PlayStation Network that became the second-largest online data breach in U.S. history. This attack compromised more than 100 million customer accounts and is the the largest data breach in the U.S. since 2009. The first-largest data breach was when hackers stole credit and debit card numbers from the Heartland Payment Systems [15].

"Anyone can go get an Amazon account and use it anonymously"

said Pete Malcolm, chief executive officer of Abiquo Inc. [15]. Recently, the defense industry has been a target of cyber attacks. Lockheed Martin, the largest U.S. defense contractor, Northrop Grumman, the second largest defense contractor, and L3 Communications are the most recent reports of cyberattacks; it appears that it all started when RSA Security was breached earlier this year. RSA Security admitted March 17th that cybercriminals had breached its network and obtained information relating to the SecureID technology. The SecureID lets remote users log in to the company with a cryptographic key secure token. The company has refused to publicly discuss exactly what was stolen or when the breach actually occurred, but anonymous sources say that clone SecureID tokens were used. The mentioned defense companies as well as other defense, industry, and healthcare customers use the RSA SecureID tokens. Mitigation plans for these defense contractor companies so far is to shut down their remote access and a companywide password reset. The Pentagon has announced that international computer intrusions are to be considered acts of war against the United States and will be answered with conventional military force [8].

A few years back, hackers were much easier to track since they launched their attacks from their parent's basement or back bedroom, but now with cloud computing and anonymous systems such as Tor, it is extremely hard to trace. These are the kind of problems that researchers are trying to solve; these are real world situations that affect millions of innocent people and leave the FBI, Law Enforcement Agencies and big corporations scratching their heads.

Malicious or criminal cyberattacks in the U.S. are escalating rapidly. According to a March report by the Ponemon Institute, they made up 31 percent of data breaches in 2010. These cyberattacks are up from 24 percent last year. Each event costs U.S. businesses an average of $ 7.2 million dollars. This study also found that about 85 percent of all U.S. companies have experienced one or more attacks [27]. This report coincides with a similar report conducted by the U.S Government Accountability Office (GAO). This government entity ensures that the U.S government spends the taxpayers money effectively. The study was formed by a compilation of existing reports, surveys and interviews with public and private officials. It identified the major government organizations that deal with cybercrime.

These organizations include: The Department of Justice (DOJ), The Department of Homeland Security (DHS), The Department of Defense (DOD) and the Federal Trade Commission (FTC). The estimated annual loss due to computer crime is $ 67.2 billion dollars for U.S. organizations according to the Federal Bureau of Investigations (FBI) [11]. Figure 1.1 illustrates a comparison between Traditional criminal techniques and Cybercrime.[1]

Now, to highlight the importance of *Accountability* in cyberspace, here are three recent good cases on cybercrime that serve as good examples of what happens when people loose all accountability on their actions. These are felony or criminal cases that will permanently taint an individual's record making it difficult for an him/her to get the simplest job possible. Please visit the Department of Justice website for more information on any particular case [7].

- ***Manhattan U.S. Attorney charges college student with creating and disseminating counterfeit online coupons over the Internet.*** On May 11, 2011, The U.S. Attorney for the southern District of New York and the FBI charged Lucas Townsend Henderson with wire fraud and trafficking in counterfeit online coupons. Henderson cre-

[1]Source: GAO

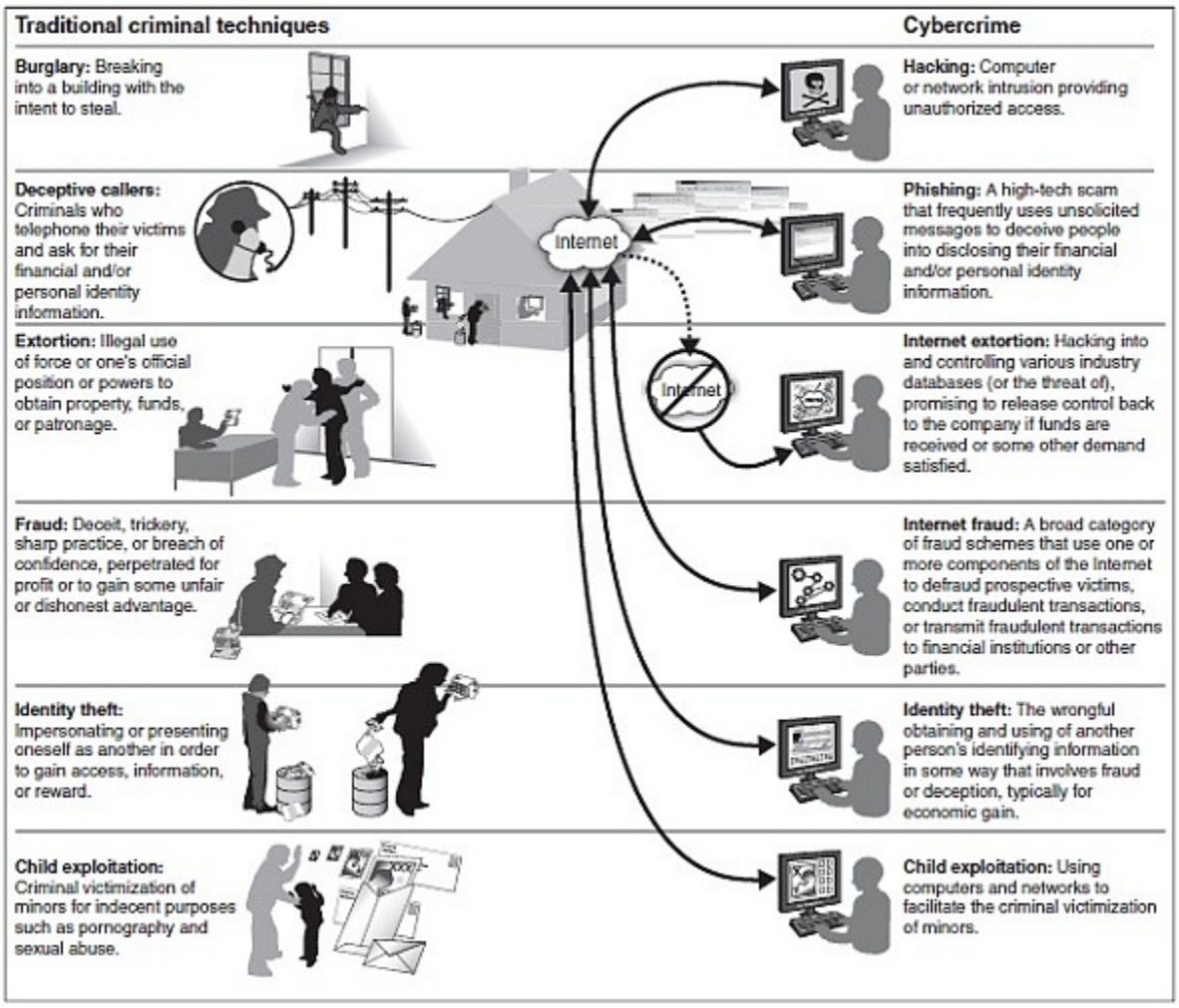

Figure 1.1 Comparison between Traditional Criminal Techniques and Cybercrime

ated these coupons between July 2010 and March 2011. Retailers and manufacturers lost thousands of dollars by paying out the coupons as a result of Henderson's behavior. Henderson posted these coupons on two message boards exclusive to the discussion of online coupons using the nicknames "Anonymous 123", "Anonymous234" and "Anonymous345" Henderson also wrote tutorials on how to create these online counterfeit coupons so others can create these coupons using their own computers. Henderson is 22 years old and lives in Lubbock Texas, he was charged with one count of wire fraud, which carries a maximum sentence of 20 years in prison, and one count of trafficking in counterfeit goods, which carries a maximum sentence of 10 years in prison. Henderson assumed responsibility for

his actions and surrender himself to the New York authorities.

- ***Hacker pleads guilty to identity theft and credit card fraud resulting in losses of more than 36 million.*** On Thursday April 21, 2011 Rogelio Hackett Jr. 26 of Lithonia Ga. pleaded guilty before U.S. District Judge Anthony Trenga in Alexandria Va. to trafficking in counterfeit credit cards and aggravated identity theft. U.S. secret agents searched Hackett's home armed with a search warrant and found 675,000 stolen credit card numbers and related information in his computers and email accounts. Hackett admitted in court that he has been doing this since 2002 either by hacking into business computer networks and downloading credit card databases, or purchasing information on the Internet using card forums which are online discussion groups used by "carders" to traffic credit card and personal information.

- ***Five domestic defendants linked to international computer hacking ring guilty of federal fraud charges which led to 46 people charged in Operation "Phish Phry" have now been convicted.*** March 25, 2011, five people were convicted of federal charges for being involved in an international "phishing" operation that used spam emails and bogus websites to collect personal information that was used to defraud American banks. A multinational investigation conducted in the U.S. and Egypt that led to the charges against 100 individuals, the largest number of defendants ever charged in a cybercrime case and 46 of them have been convicted in a federal court in Los Angeles as a result of Operation "Phish Phry". Operation "Phish Phry" revealed how Egyptian-based hackers obtained bank account numbers and related personal identification information from an unknown number of bank customers through phishing - a technique that involves sending e-mail messages that appear to be official correspondence from banks or credit card vendors. Bank customers who received the spam emails were directed to fake websites claiming to be linked to financial institutions, where the customers were asked to enter their account numbers, passwords and other personal identification information.

These are just three cases; there are many others where people are being charged for cybercrimes. One thing that was noticed while reading through these cases is that most of these

criminals are young and loose all accountability for their actions just for the thrill of hacking and making quick money or thinking they are going to outsmart the FBI. Some theories as to why these young cybercriminals get caught right away are because they are inexperienced, don't cover their tracks or just want to brag to their friends. Sophisticated hackers that have a good computer technical background, a computer science or mathematics degree or are self taught programmers might get away for a while but eventually get caught. We continue with the discussion on anonymity systems.

Unfortunately, complex anonymous systems such as Tor left out accountability features when the developers designed the software architecture. And as a consequence, it has caused many problems and headaches for the U.S. and many other countries around the world. An example of this is the controversial website Wikileaks [2]. In this example, sensitive data is stolen or captured from one of the exit routers in the Tor system. This is acomplished by performing network traffic analysis and is exposed to the world. The website is hard to shut down because many of the hosts are onion routers that are hard to trace and the uniform resource locator (url) is encrypted. In this research, the possibility of engineering powerful schemes that allow accountability in such a system is explored. A lightweight and efficient prototype system that might solve these exceptionally important issues was successfully implemented.

Here are the definitions and proposals of some important properties for the *Accountable Anonymity* System (A2S) that was designed:

Anonymity is derived from the Greek word anonymia, meaning "without a name" or "namelessness". Anonymity means that an individual's personal identity is publicly unknown. People choose not to disclose their identity and become anonymous for many reasons [40]. For example, some good reasons are charity contributions, crime witnesses, support groups. Some bad reasons are to commit cybercrimes and traditional crimes such as bank robberies although, some criminals are not careful enough and don't cover their faces when committing such crimes. Anonymity can be used for good or evil things. This property is important in the A2S system because it gives its users the power to communicate anonymously but with responsibility.

Accountability stems from the late Latin *accomplare* (to account), a prefixed form of *computare* (to calculate), which in turn derived from *putare* (to reckon). The word itself does not appear in English until its use in the 13th century Norman England, the concept of account-giving has ancient roots in record keeping activities related to governance and money-lending systems that first developed in Ancient Israel, Babylon, Egypt, Greece and later, Rome. Accountability is an obligation or willingness to accept responsibility or to account for our own actions. Often used synonymously with such concepts as responsibility, and liability [39]. This property is also important in the A2S system because if users abuse the system they will be held accountable for any offending message that is reported. As a consequence, they will be traced and all evidence will be handed over to a law enforcement agency.

Accountable Anonymity is the combination of both definitions given above. This concept is introduced for the first time in conjuction with an anonymous system. The notion of accountable anonymity is when people are free to choose not to disclose their identity, but must also be willing to accept any and all consequences for their own actions. *Accountable Anonymity* is believed to be a more *rational anonymity* as opposed to the *absolute anonymity*, which is already provided by current well designed anonymous systems. This property is the foundation of this research work.

Efficiency The system needs to be as efficient as possible in order to support a large number of users. This feature is of great importance because if the system grows in popularity, it can create bottlenecks on the network and if the system is slow many users won't use it.

Portability The system is currently developed in C++ for Linux/Unix systems with the g++ compiler and can be easily ported to a Macintosh or Windows system if necessary.

Usability The system needs to be user friendly.

Approach and contributions In this thesis, the *Accountable Anonymity* System (A2S) is presented. This system contains all the above properties and achieves anonymous communications on a network of peer to peer networked computers. The onion routing

structure of the Tor system was carefully examined and the A2S system was designed with a special circuit-less structure and a new proxy-re-encryption scheme that achieves multi-hop capabilities, a feature that has never been researched before with anonymity systems in the security research community. The A2S system will be described in further detail in the remaining chapters, as well as other related work.

Tor system's approach Tor stands for "The onion router". Tor is the most popular anonymous system available on the Internet. It is available to the general public and free to download from the Tor website [13]. Up to now the Tor system has about 1500 - 2000 onion routers, these onion routers are run by regular users, businesses, government agencies, activists and anybody that wants anonymity online around the world and that can spare a portion of their bandwidth in order to keep the anonymous communications [30]. The software is complex and contains thousands of lines of code written in the C language. In order to achieve anonymity with Tor, the user constructs a circuit where three onion routers are chosen by the user. The software comes loaded with a configuration file and a generic digital certificate that is used once the Transport Layer Security (TLS) [34] connection is made. Once the TLS connection is complete, a hybrid symmetric cryptographic operation is used to construct the onion via Diffie-Hellman protocol [36] and RSA cryptographic algorithm. If any of these steps fail, the entire process gets repeated again until the circuit is successfully constructed. The network latency on the TLS handshake was measured. It takes 669 ms to complete the TLS handshake, and other computations need to take place in order to construct the onion part before the user can successfully send encrypted data through the wire. The initial suspicions turned out to be correct and the findings showed that this system is slow compared with the A2S system. The reason is simple, all onion routers are constructing the TLS connection and negotiating the handshake, by multiplying by 2000, which is the number of onion routers currently on the network, the time it takes for it to complete some of these operations is greatly increased.

Threat Model A global passive adversary is the most powerful threat against any anonymity

system and as other systems the A2S system is not immune to such strong adversary [30]. An adversary can monitor parts of the network traffic as well as generate, modify, delete or delay such traffic. The adversary can also operate some of the onion routers and hack into other onion routers. But since the A2S system does not create circuits, and the proxy-re-encryption scheme re-encrypts the ciphertext at each hop with a special proxy-re-encryption key, the adversary will try to decrypt the ciphertext and will get a different ciphertext instead of the original message. Only the destination will be able to decrypt the ciphertext and get the correct message.

Attacks and Defenses The description of some of the most common attacks against any system anonymous or not and how well this implementation can mitigate them are discussed in detail. Some of these attacks can cripple any system that has no security mechanisms implemented or if existing vulnerabilities exist in the system.

Passive Attacks

- ***Network Analysis*** - The network can be analyzed using a simple tool such as the ping command and a map can be constructed on the network. This procedure is often called footprinting. Once this map is constructed, the attacker can create a complete profile of the network infrastructure before launching an active attack. Figure 1.2 illustrates a passive attack where Eve monitors the communication between Alice and Bob. To mitigate this attack we disabled ping fuctionality on our system.

- ***Host Traffic Analysis*** - The popular tool called Nmap can be used by an attacker to get specific information of a host system, this technique is called fingerprinting and is also a very popular for hackers because this utility too can tell wether the host system is a Unix, Machintosh or Windows System. To mitigate this attack our Unix/Linux systems don't respond to NETBIOS type 137 requests.

- ***Eavesdropping*** - This is the traditional method of spying with the intent to gather information. Email messages and instant messaging are vulnerable to eavesdropping because of their insecure design. One example is when something is bought online the email

account used for the purchase is opened and the exact same thing that was purchased is now being advertised on the right hand side of the screen. To mitigate this attack, the messages are re-encrypted several times and should be extremely hard to get the original message back without the proper key to decrypt it.

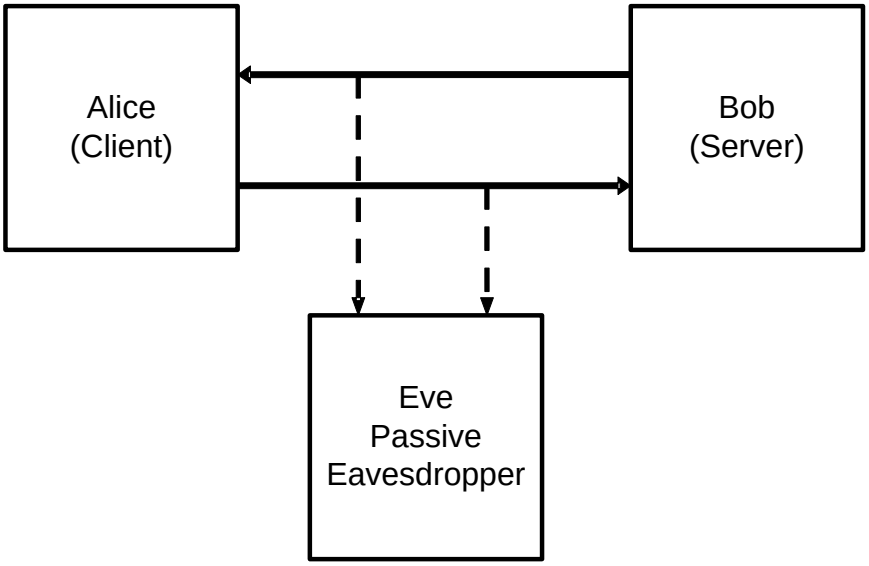

Figure 1.2 Passive Attack

Active Attacks

- ***Man in the middle*** - In this type of attack, the attacker can impersonate either the client or the server. It can also block, route and modify, insert data into the data stream that is being transmitted between both parties. Figure 1.3 illustrates an active attack.

- ***Denial of Service Attacks*** - Since the A2S system will probably be released to the public as a service, denial of service attacks could be launched against the system's network. But again, the A2S network does not create circuits or expensive cryptographic operations such as TLS connections so there is no way the attack can affect the network because the attacker won't know what computer to target.

- ***Running Malicious Node*** - An attacker can turn malicious and will try to attack the

rest of the nodes, but if detected the user's privileges will get revoked and possibly face criminal charges.

- ***Replay Attacks*** - Some anonymity systems are vulnerable to replay attacks. The Tor system and the A2S system are immune to this attack. For example in the Tor system, replaying one side of a handshake will result in a different negotiated session key, and so the rest of the recorded session can't be used. The A2S system does not execute a handshaking scheme so the replay attack won't work.

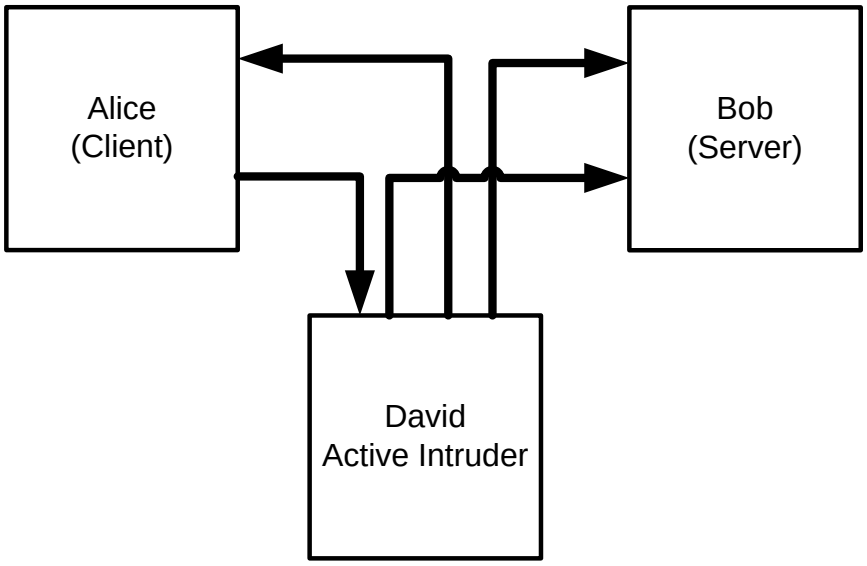

Figure 1.3 Active Attack

CHAPTER 2. LITERARY SURVEY

There are numerous systems that provide anonymity services. This Chapter describes the existing work done on anonymity systems. It Begins with single proxy systems and further elaborates on distributed systems and briefly describes various anonymous credential systems.

2.1 Single Proxy-based Anonymity Systems

In a centralized trust network, a single proxy is used to provide anonymity to its users. The user connects to the proxy which provides a different IP address to the end user. Some examples of this type of systems are the Penet Remailer and the Anonymizer.

2.1.1 Penet Remailer

Penet Remailer was a single proxy that provided email services only, where a user sends a message to the service, an alias is generated for the sender and the real email address of the user and the alias are saved in a database. The remailer strips off any identifying information from the message and forwards it to the intended user. When the remailer receives an answer from the intended user, then the remailer will look up the alias's real email and send the message to the pseudonymous user. This service was not truly anonymous, because even though the end parties didn't know each other, the anonymizer knew both of them, so it suffered many attacks and it was finally shut down by its creator in 1996 for not providing enough security and anonymity to its users, also because of controversy issues with the Church of Scientology [12].

2.1.2 Babel Remailer

Babel Remailer allows email users to converse electronically while remaining anonymous to each other and other hostile parties. This system was built with the Perl programming language and the Pretty Good Privacy (PGP) email cryptographic encryption software and the authors called this implementation, the Babel Mix. The password needed to access the secret key of a remailer is stored in cleartext [5]. This makes this system vulnerable to many attacks. Babel does not contain a single line of cryptographic code and relies entirely on PGP for crypto computations. The average time delay to send messages on this system is about 24 hours, which makes it unacceptable for mass use.

2.1.3 Mixminion Remailer

Mixminion Remailer is a message based anonymous remailer protocol with secure single-use reply blocks. It works in a real-world Internet environment. Ideas from this design are in collaboration with the Mixmaster development team. Mixminion aims to defeat even a global passive adversary, and address the end-to-end timing vulnerability. Mixmaster is vulnerable to the blending attack, where the attacker blends his own recognizable messages with the honest messages in the batch [10].

2.1.4 Mixmaster Remailer

Mixmaster Remailer is the most popular and widely deployed remailer. Mixmaster uses 3DES keys for each mix node in a chain between the sender and receiver; these 3DES keys are in turn encrypted with the RSA public keys of each mix node. All Mixmaster packets are the same length. When a message reaches a mix node, it decrypts the header, then decrypts the body of the message, and then places the message in a "message pool". Once enough messages have been placed in the pool, the node picks a random message to forward. The software works for Linux and Windows systems and can be downloaded from the Mixmaster website at [35].

2.1.5 Anonymizer

Anonymizer was founded by the creator of Mixmaster anonymous remailer and it is an anonymous proxy that connects the user to the Internet [35]. The company was bought in 2008 by Abraxas Corporation. *Anonymizer* is a computer that serves as a virtual private network or proxy to users that want to use the service. This allows the users to browse the Internet anonymously. The first problem with the *Anonymizer* is that the system knows the user's IP address and everything the user is sending through this server. The second problem is that the *Anonymizer* can add advertisements to webpages that the user is browsing. The third problem is that an adversary can take control of the *Anonymizer*, then all users loose anonymity and the adversary can take control of the data transmission. The fourth problem is that the adversary can also shut down the proxy [1].

2.2 Mix-net based Anonymity Systems

In order to make the systems more powerful against attacks, researchers began building distributed networks. David Chaum pioneered the popular idea called "Mix". It led to many other interesting designs because it was used as a building block.

2.2.1 Chaum Mixes

A *Mix* is an intermediate computer between senders and recipients. It achieves anonymity by scrambling and padding the messages with public key cryptography and sends them through the network in a constant size. The intention of this is to protect the data from traffic analysis.

How it works

Assume there is a *Mix* M with public-private keys and users A and B. M encrypts all communication to protect from traffic analysis by sending all messages in the same format and length. User A constructs a message for delivery to user B by appending a random value R to the

message, seals it with user's B public key K_B, appends B's address and then seals the result with the *Mix's* public key K_M.

So A sends: K_M (K_B(message, R), B's address) to M. Upon receipt, M decrypts the payload with his private key, and now he knows the destination address B.

So M sends: K_B(message, R) to B. Upon receipt, B decrypts the payload with his private key, and he can read the message.

An adversary can find the sender via two techniques: traffic analysis and man in the middle attack. He can easily compute the following: First, he analyses the packets being transmitted from A to M, then he can look at the packets being transmitted from M to B. Once this is examined, the eavesdropper will grab one of the messages and concatenate B's address. K_B(message, R) adds B's address (K_B(message, R), B's address) encrypts with M's public key K_M (K_B(message, R), B's address) once he computes this, he can compare the packets that are arriving at M and find A. To overcome this, a nonce or random number is added to the message that A sends to M and the nonce will be removed by the *Mix* as it reaches its final destination. To make the transmission more secure, a series of *Mixes* is used where the sender chooses a path of these *Mixes* and encrypts the message in an "onion format". Figure 2.1 shows an example of this procedure. To simplify this discussion in terms of the recipient replying to the anonymous message, we will illustrate this with a single *Mix*. In order for B to reply, we need to send A's address, but the address needs to be encrypted: K_1 (S_1, A), K_x where K_x is a public one-time session key, and S_1 is a random number that is used as a key. A will send this return address as part of the message described above. B uses the return address to form a response to A: B sends K_1 (S_1, A), K_x(S_0, response) to M and M transforms it and sends to A: S_1(K_x(S_0, response). Note that at each hop in the chain the message is being encrypted with some random value S_i [6].

Disadvantages Chaum *Mixes* use computationally expensive public key cryptography which is slow for simultaneous communication. Additionally, *Mixes* are vulnerable to timing attacks, this means that the first and last *mix* can join forces to de-anonymize the sender. Also, the nodes send constant fixed-length messages at regular intervals, including padded

random data when necessary, this creates a bottleneck on the network.

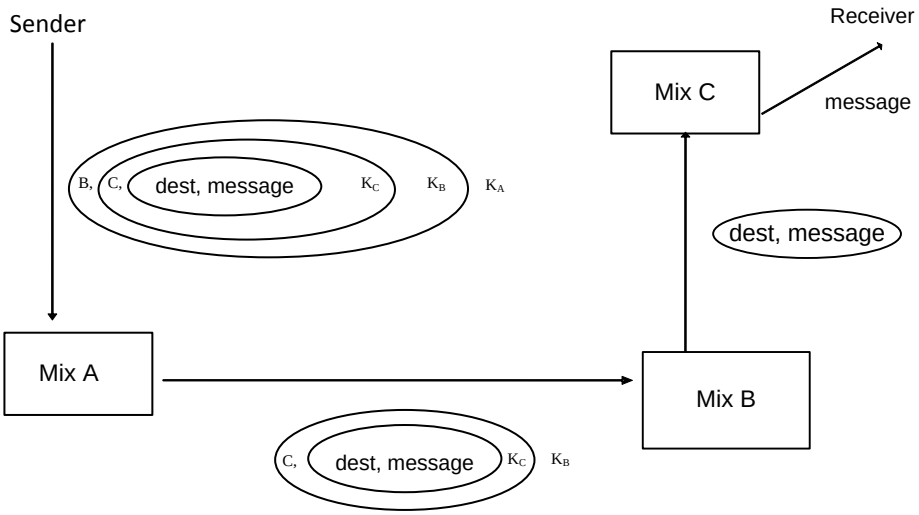

Figure 2.1 Chaum Mixes

2.2.2 Onion Routing

Onion Routing is the first generation anonymity system designed and architectured by the U.S. Navy and the Naval Research Lab. Tor was presented in Chapter 1, which is the second generation onion routing anonymity system that was released to the public and now run by an independent organization. The main objective of the onion routing project is to design, build and analyze anonymous communications systems. These systems are capable of resisting the following: traffic analysis attacks, eavesdropping, other attacks coming from Internet users, onion routers themselves. The main advantage is that the systems are convenient to use in a low latency Internet based environment [25].

How it works

Using onion routing can be explained in four steps and the steps are similar to Chaum Mixes in Figure 2.1:

1. ***Define the route*** - The sender connects to an entry onion router. This onion router constructs a chain of onion routers to the receiver and we call this an "onion".

2. ***Construct the anonymous connection*** - The onion moves between the onion routers, each onion router only knows about the previous node and the next node.

3. ***Move data through the connection*** - The entry onion router encrypts the data with the other onion routers public keys and this is how the layers of the "onion" are formed. At each hop each onion router decrypts one layer of the onion and sends the rest to the next onion router. The last router contains the plaintext data which is then forwarded to the receiver. If the receiver wants to send a response, the same process is applied in reverse order.

4. ***Destroy the anonymous connection*** - If any of the above steps or the connection fails for any reason, a destroy message is sent along the connection and any pertaining information regarding the connection is wiped out.

Onion routing uses public key cryptography to establish the communication between the onion routers and symmetric cryptography for the communication. Notice that this is different than the Chaum Mix implementation where they used public key cryptography only.

Disadvantages in onion routing the length of the message grows as the path length grows, and more keys are needed. The longer the path is, the more overhead it will cause. Research done by Nathan Evans from the University of Colorado discovered another vulnerability in the Tor system. This attack consisted of using a circular path with several onion routers and running malicious onion routers. With this attack, Nathan was able to perform traffic analysis and find the sender [24]. This vulnerability has been fixed in the Tor system. Another disadvantage is that if a user enables JavaScript, Java applets or ActiveX and is using onion routing or Tor his system is vulnerable to an attack.

2.2.3 Crowds

How it works

The anonymity system *Crowds* does not use public key cryptography, this system is for web browsing privacy only and the idea is to hide the user in a *crowd* of other users [33]. Crowds can

be compared to a distributed and chained *Anonymizer*, with encrypted links between crowd members. The notion of degrees of Anonymity is introduced and highlighted here.

- Absolute privacy
- Beyond suspicion
- Probable innocence
- Possible innocence
- Exposed
- Provably exposed

Disadvantages The mere fact that this system possesses a property that allows a member of a crowd to submit requests initiated by other users is ludicrous. Not only this property opens up more abuse to existing users but data is mishandled and can be modified in transit by malicious crowd users. The authors seem to acknowledge this but they still decided to implement the property on their system. This system makes no effort to defend against denial of service attacks by their own crowd members. This means that a crowd member could accept messages from other crowd members and refuse to pass them along. This system does not use SSL either and the published paper only mentions a stream cipher performing the cryptographic operations but it failed to mention which one. The stream cipher code was written in Perl 5 so it is a poor choice for the implementation because it creates a bottleneck on the network and it shows from their statistical data. The 5 kilobyte data packet that traversed a path length of 5 took 4508 milliseconds to complete.

2.2.4 Tarzan

Tarzan is a peer-to-peer anonymous IP network where the anonymity is achieved with layered encryption and multi-hop routing similar to Chaum mixes [21]. It is organized as a decentralized peer to peer overlay. This system was developed by Michael J. Freedman

and the legendary Robert Morris from MIT Lab for Computer Science who created the Morris worm, one of the first computer worms unleashed on the Internet back in 1988 [42].

How it works

Tarzan uses a network address translator (NAT) to bridge between Tarzan hosts and Internet hosts. Anonymity is provided to either clients or servers, without requiring that both participate.

All participants run software that:

1. Discovers other participating nodes.
2. Intercepts packets generated by local applications that should be anonymized.
3. Manages tunnels through chains of other nodes to anonymize these packets.
4. Forwards packets to implement other nodes' tunnels.
5. Operates a NAT to forward other participants' packets onto the ordinary Internet.

Disadvantages Tarzan operates at the IP layer, this means that applications that use the IP layer and need anonymity, will have to replace the IP layer for the Tarzan layer. Also this system uses a NAT, if the NAT fails, the connections that already set up through it will also fail. Tarzan is just a building block for anonymous systems.

2.2.5 Morphmix

Morphmix is another peer-to-peer system that uses symmetric key encryption and nested encryption and sends data encrypted through an anonymous tunnel. This system offers collusion detection that allows it to identify the compromised paths with high probability before they are being used [22].

How it works This system is a mirror copy of the onion routing system and added a collusion detection mechanism and is still a work in progress. The protocol design is incomplete.

Disadvantages The collusion detection mechanism consists of every node to maintain an internal table of each node selection it has received. This is a form of traffic analysis and a malicious user can use the data to de-anonymize the other users. Based on their statistics there is no way to determine exactly how many colluding nodes are on a network. The use of probability does not constitute that there are a certain amount of well behaved or colluding nodes in a determined system.

2.2.6 Freedom

Freedom is run by a for profit Canadian company called Zero Knowledge Systems. They have deployed two major systems, one for email and another for TCP/IP. The email system is similar to Mixmaster, and the TCP/IP system is similar to Onion Routing.

How it works The Freedom system consists of a set of Freedom server nodes that make up the Freedom network, and the Freedom core servers that provide basic services. The network transports encrypted IP traffic from one node to the next. Nodes on the Freedom network are called Anonymous Internet Proxies (AIP's). The number of nodes used in a route is chosen by the user by setting her security level in the Freedom client [26].

Disadvantages Freedom loses its anonymity for the primary reason that it is a commercial network operated for profit. Users must purchase the nyms used in pseudonymous communications. Purchases are performed via the online Web store, through credit-card or cash payments. A google search on this company turned out that they are no longer in operation.

2.3 Adding Traceability to Anonymity Systems and the limitations thereof

2.3.1 Nym

Nym Is a simple way to allow pseudonymous access to Internet services via anonymity networks such as Tor but can still block abusive users. Nym uses blind signatures to create a pseudonymity system that users can use. The blinded token is anonymously exchanged for an ordinary TLS client certificate [14].

How it works

To better understand how Nym works we highlighted the steps:

- Obtain a token in exchange for proving possession of a resource such as an IP address or email address to the token server. This is the only step in which the client might need to expose identifying information such as an IP address; the rest can take place via an anonymizing network.
- Unblind the token.
- Wait a random interval sufficient to foil transaction time correlations.
- Exchange one or more tokens for a CA's signature on an X.509 client certificate.
- Load the certificate into a web browser and use it to gain access to a TLS-based web service.

Limitations The main drawback here is that the client-side implementation is a Javascript application. Recall that the Tor system does not protect against Javascript attacks if the browser is Javascript-enabled. Also, the user has to create a TLS certificate which defeats the purpose of anonymity, after that, the user needs to wait a few hours to get the Certificate Authority (CA) signed certificate back and install the certificate before he can access Nym-enabled websites. In addition, some modifications to the server on the websites have to be done in order for this implementation to work. This is not realistic due to the millions of websites on the web. The nym system used on top of the Tor system is equally compared to browsing the Internet with two TLS connections. This is not a good idea because it will probably be extremely slow. It seems that this solution is geared towards only one website.

2.3.2 Nymble

Nymble is a system that allows websites such as the free user-built encyclopedia Wikipedia to selectively block anonymity users that use an anonymity network such as Tor. Abusive

users use the Tor network to cyber-vandalize useful websites such as Wikipedia and system administrators have no other choice but to block all incoming traffic coming from these anonymity systems blocking all users including the innocent ones. Nymble designers came up with a way to address this problem [28]. We describe it in the following five properties:

- Honest users remain anonymous and their request cannot be linked.
- Servers will have the ability to blacklist an anonymous user for future connections and put a complaint on this blacklisted user.
- This blacklisted user's accesses remains anonymous before the complaint.
- Next time the blacklisted user logs on; he will be notified of this status.
- Users are aware of their blacklist status before accessing a service.

How it works

The user connects to the Pseudonym Manager (PM) directly to get a pseudonym. This pseudonym or alias is mapped to the user's IP address. After the user gets his pseudonym he connects to the Nymble Manager (NM) by using the Tor system and requests nymbles which are a special type of pseudonyms for access to a particular website for example: Wikipedia. Nymbles are generated using the user's pseudonym and the server's identity. Websites can blacklist users by obtaining a seed for a particular nymble, allowing them to link future nymbles from the same user. Therefore, websites can blacklist anonymous users without the knowledge of their IP addresses while allowing behaving users to connect anonymously. Misbehaving users are aware of their blacklist status. They won't be able to connect for a period of one day or whatever the linkability window is set to if they are blacklisted. However, after the linkability window period expires they can access the system again.

Limitations The Nymble implementation is not feasible because the attacker will only be blocked for a period of time based on the linkability window. The period of time that is most commonly used is one day. The attacker will be back the next day and commit

more abuse or the attacker may have more than one disposable machines. Even though, the system ensures that the users are aware of their blacklist status before they present a nymble, there is no evidence to support how they make the user to disconnect immediately from their system if they are blacklisted. The authors claim that building a *credential system* such as this one will add a layer of accountability to any publicly known anonymity network, but we believe that this will only impede abusers temporarily or they can simply log on to their other computers or virtual machines and go about their business. There is also the possibility of the PM and the NM colluding to de-anonymize the user.

2.3.3 Nymbler

Nymbler is a proposed improvement on the Nymble system using zero knowledge proofs and a new cryptographic technique called verifier efficient restricted blind signatures (VERBS) [31]. They eliminate the use of trusted third parties referring to the PM and the NM described above that can easily collude to violate a user's anonymity. Users are permitted to construct their own nymbles using anonymous credentials. This approach uses RSA based signature similar to Chaum's combined with zero-knowledge proofs that allow the user to prove certain properties about the message before it is signed. They replace the PM with the Credential Manager (CM) and their scheme allows a user to construct his/her own set of nymbles in such a way that the NM is convinced of their validity without ever actually seeing them. Figure 2.2 depicts a Nymbler system. Then, the NM issues the user with VERBS on his/hers nymbles so that the SP can also be convinced of their validity.

In order for a user Alice to obtain service from an SP, the following steps need to be followed:

1. Alice connects directly to the CM to get a credential.
2. Alice connects anonymously to the NM to gain access to a particular website, in this case the (SP).
3. Alice computes a set of nymbles, she proves to the NM in zero-knowledge that this was done correctly to obtain a VERBS on each nymble.

4. Alice connects anonymously to the SP. She checks the blacklist and then presents the SP with a nymble together with a VERBS from the NM for that nymble.

5. The SP checks the linking list to ensure Alice is not banned, and then verifes that the VERBS is valid and stores the nimble in a log file and grants access to Alice.

6. If Alice misbehaves during her session with the SP, the SP reports her misbehavior to the NM by presenting it with the nymble used by Alice during that session.

7. The NM uses this nymble to compute Alice's subsequent nymbles for the remainder of the linkability window. Each of these nymbles is added to the linking list, and the final nymble is added to the blacklist. This allows the SP to detect any future connection attempts by Alice for the remainder of the linkability window.

Limitations The authors of Nymbler presented a new anonymous blacklisting scheme modeled after Nymble. They claimed to eliminate the trusted third parties used in Nymble but they just replaced the PM with a Credential Manager. This implementation is not scalable either because of the amount of computations added to the simple process of logging into a website. The service provider (SP) could be any website that the user wants to log on to. Any customer that wants to implement the Nymbler design will have to re-design their website. This will incur costly web design expense fees for a non-profit organization or anyone that adopts this system.

Recent blacklisting system

Jack is another blacklisting scheme. The authors recently published their work. The scheme is based on cryptographic accumulators and is secure against TTP collusion and provides improved scalability compared to previous schemes [43].

Overview

In this Chapter, previous research work on anonymity systems was presented and described. Obviously, all of the systems that have been developed were not covered but focus on the

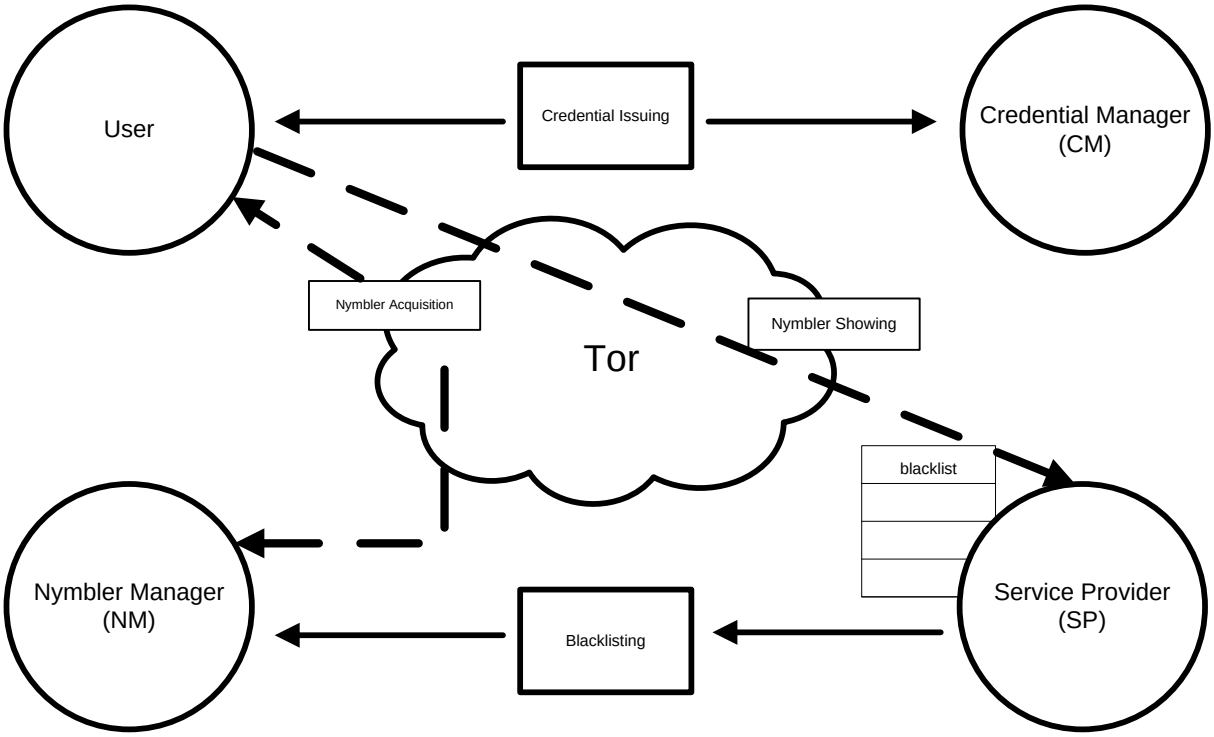

Figure 2.2 Nymbler Scheme Architecture

relevant ones to this research were introduced and described in detail. Some of these systems are theoretical and might not get deployed to a full working system. However, for the sake of emphasizing the research it is always good practice to include work done by other researchers. The limitations on the Nym based systems are that these blacklisting systems only "blacklist" or "block" a malicious user temporarily. The same malicious user can come back the next day and do more damage. The next Chapter will present the contributing work on anonymity.

CHAPTER 3. PRELIMINARIES

This Chapter begins with some background theory on bilinear mappings, tate pairings and elliptic curves. A brief background on the RSA algorithm is described, all that background is then used to explain the accountable anonymity scheme.

3.1 Bilinear Mappings

In order to describe bilinear mappings we must first define some basic group theory definitions.

Basic Definitions

3.1.1 Groups

One of the most familiar groups is the set of integers, $\mathbb{Z}$ which consists of the numbers:
..., -4, -3, -2, -1, 0, 1, 2, 3, 4, ...
A group is a set, G, together with an operation $\bullet$ that combines any two elements a and b to form another group element, denoted $a \bullet b$ or ab. To qualify as a group, the set and operation, (G,$\bullet$), must satisfy four requirements known as the group axioms [38].

Closure $\forall$ a, b $\in G$, the result of the operation, a $\bullet$ b, is also in G.

Associativity $\forall$ a, b and c $\in G$, (a $\bullet$ b) $\bullet$ c = a $\bullet$ (b $\bullet$ c).

Identity Element $\exists$ e $\in G$, $\forall$ $a \in G$, the equation e $\bullet$ a = a $\bullet$ e = a holds. The Identity element of a group G is often written as 1 or 1_G.

Inverse Element $\forall$ a $\in G$, $\exists$ b $\in G$ such that a $\bullet$ b = b $\bullet$ a = 1.

3.1.2 Abelian Groups

Abelian groups posess the same requirements mentioned above for a group, and contains the commutative property as well.

Commutativity $\forall$ a, b $\in G$, a $\bullet$ b = b $\bullet$ a.

3.1.3 Cyclic Groups

A cyclic group is a group in which there is an element x such that each element of the group may be written as X^k for some integer k. In additive notation, this translates to $k \bullet x$ where x is a *generator* of the cyclic group or that the group is generated by x. As an example, the *integers* under addition is a cyclic group. The number 1 is a generator. This is because for any n in the *integers* $n = n \bullet 1$. Note that -1 is also a generator. It follows that a cyclic group is an abelian group although not every abelian group is a cyclic group.

3.1.4 Bilinear Maps

Let G_1, G_2, *and* G_t *be cyclic groups of the same order.*

Definition

A bilinear map from $G_1 \times G_2$ to G_t is a function:
e: $G_1 \times G_2 \mapsto G_t$ such that $\forall\ u \in G_1, v \in G_2,\ a,\ b \in \mathbb{Z}, e(u,v)^{ab}$.

Bilinear maps are called pairings because they associate pairs of elements from G_1 and G_2 with elements in G_t where G_t is a cyclic group. More details are given on pairings on the next section. Here are the most common new problems that have been defined and assumed hard in the new bilinear context [4].

Bilinear Diffie-Hellman Given g, g^a, g^b, g^c, compute $e(g,g)^{abc}$ (something like a "three-way" CDH but across the two groups)

Decisional Bilinear Diffie-Hellman Distinguish $g, g^a, g^b, g^c, e(g,g)^{abc}$ from $g, g^a, g^b, g^c, e(g,g)^z$

k-Bilinear Diffie-Hellman Inversion Given $g, g^y, g^{y^2}, \ldots g^{y^k}$, compute $e(g,g)^{1/y}$

k-Decisional Bilinear Diffie-Hellman Inversion Distinguish $g, g^y, g^{y^2}, \ldots g^{y^k}, e(g,g)^{1/y}$ from $g, g^y, g^{y^2}, \ldots g^{y^k}, e(g,g)^z$

3.2 Elliptic Curve Cryptography

Elliptic curves

The use of elliptic curves in cryptography (ECC) was first introduced by Neal Koblitz a mathematics Professor at the University of Washington [17] and Victor S. Miller in 1985 an American mathematician at the Center for Communications Research (CCR) of the institute for Defense Analyses in Princeton, New Jersey [23]. The U.S. National Security Agency (NSA) has endorced ECC by including schemes based on it in its Suite B set of recommended algorithms and allows their use for protecting information classified up to the top secret with 384-bit keys. Full details of the ECC algorithms are not given, but we introduce the concept here because ECC is used in the A2S system.

An elliptic curve is the set of solutions (x,y) to an equation of the form:

$$y^2 = x^3 + x \tag{3.1}$$

together with an extra point O which is called the point at infinity.

How it works

This is a very simple example of an Elliptic curve group over F_p consider an elliptic curve over the field F_{23}. With a = 1 and b = 0, the elliptic curve equation is $y^2 = x^3 + x$. The point (9,5) satisfies this equation since:

$y^2 \bmod p = x^3 + x \bmod p$

$25 \bmod 23 = 729 + 9 \bmod 23$

$25 \bmod 23 = 738 \bmod 23$

$2 = 2$

The 23 points which satisfy this equation are: (0,0),(1,5),(1,18),(9,5),(9,18),(11,10),(11,13),(13,5) (13,18),(15,3),(15,20),(16,8),(16,15),(17,10),(17,13),(18,10) (18,13),(19,1),(19,22),(20,4),(20,19), (21,6),(21,17).

3.3 RSA Algorithm

The RSA algorithm is currently the most widely used public key scheme which was created by Rivest, Shamir and Adelman in 1978 at MIT and is based on the difficulty of factoring large numbers and that is how it gets its security[32]. The difficulty of getting the plaintext message back from the ciphertext and the public key is related to the difficulty of factoring a very large product of two prime numbers. Here is a simple example and then the A2S discussion continues with proxy re-encrytion.

Illustration

Suppose that two very large prime numbers are given, say 200 digits long and multiply them together, the result provides two properties:

1. The result is very large, about 400 digits in length.
2. It has two factors, both prime numbers, which are the two primes we multiplied together.

The product is easily found, given the two prime numbers. But finding the primes given only the product is more difficult, so difficult that once the numbers get larger, it is almost impossible to find them. There is not enough computing power to do so.

The RSA asymmetric algorithm uses this fact to generate public and private key pairs where finding the inverse, the factor finding operation is difficult.

3.3.1 Key Generation

Key Generation

1. Generate two large random primes, p and q of approximately equal size such that their product $n = pq$ is of the required bit lenght, e.g. 1024 bits.
2. Compute $n = pq$ n is used as the modulus for both public and private keys.
3. Compute $\varphi(n) = (p-1)(q-1)$, where φ is Euler's totient function.
4. Choose an integer e such that $1 < e < \varphi(n)$ and $\gcd(e,\varphi(n)) = 1$.

5. Compute the secret exponent d, $1 < d < \varphi(n)$, such that $ed \equiv 1 \bmod \varphi(n)$

3.3.2 Encryption

Encryption

Sender A does the following:

1. Obtains the recipient's B's public key *(n,e)*.
2. Represents the plaintext message as a positive integer m.
3. Computes the ciphertext c = $m^e \bmod n$.
4. Sends the ciphertext c to B.

3.3.3 Decription

Decription

Recipient B does the following:

1. Uses his private key (n,d) to compute $m = c^d \bmod n$.
2. Extracts the plaintext from the message representative m.

3.4 Proxy Re-Encryption

The concept of proxy re-encryption was first introduced by Blaze, Bleumer and Strauss in 1998 [20]. Proxy re-encryption converts the ciphertext under one public key to the ciphertext under another public key without decryption. However, the proxy cannot learn anything about the messages encrypted under either key. Although a number of proxy re-encryption (PRE) schemes have been proposed [3] [9] [29], none of them is feasible to build anonymous systems. To implement anonymity in a communication network, a new, discrete logarithm based, symmetric proxy re-encryption scheme is presented. This proxy re-encryption scheme was designed by Gang Xu, another Ph.D. student. This proxy re-encryption scheme has four algorithms **KeyGen**, **Encrypt**, **Re-Encrypt**, **Decrypt**.

Global Parameter

This cryptosystem operates over a multiplicative group G of prime order q. P is a generator of G.

KeyGen

Select separately at random two different numbers.

$$\text{x and y} \in \mathbb{Z}_q^* \tag{3.2}$$

$$\text{Alice's Key is } K_A = x \tag{3.3}$$

$$\text{Bob's Key is } K_B = y \tag{3.4}$$

$$\text{The proxy re-encryption Key is } rk_{A\to B} = (y - x) \pmod{q} \tag{3.5}$$

Encrypt

The ciphertext for Alice is (Anyone that knows Alice's key can do the encryption):

$m \in$ G is a message.

$$C_A = mP^{K_A} \tag{3.6}$$

$$= mP^x \tag{3.7}$$

Re-Encrypt

The Proxy re-encrypts the ciphertext simply by the raise of a power operation and multiplication operation.

$$OUTPUT_{re} = \{c_A\} * P^{rk_{A\to B}} \tag{3.8}$$

$$= (mP^x) * P^{(y-x)} \tag{3.9}$$

$$= mP^y \tag{3.10}$$

$$= mP^{K_B} \tag{3.11}$$

Notice that mP^{K_B} is exactly the ciphertext c_B encrypted by Bob's secret key.

Decryption

with Bob's key $K_B = y$, one can compute m from

$$C_{B/(P^{K_B})} \tag{3.12}$$

CHAPTER 4. A2S DESIGN AND ARCHITECTURE

This Chapter begins with the communication protocol phases in the A2S system and ends with the forensic feature and the steps to conduct a forensic investigation.

4.1 Design

A part of this protocol is based on the AFGH algorithm [9]. This scheme operates over two groups G_1 and G_2 of prime order q with a bilinear map $e : G_1 \times G_1 \mapsto G_2$. The system parameters are random generators $g \in G_1$ and $Z = e(g, g) \in G_2$.

The following problem is believed to be a hard problem. G is a group of prime order q , g is its generator. $a \in \mathbb{Z}_q^*, \mathbb{Z}_q^* = 1,2,\ldots, q$. $1/a$ is a's multiplicative inverse over $\mathbb{Z}_q^*$. If $g^{1/a}$ is known, it is a hard problem to get g^a. (Both $g^{1/a}$ and g^a are over G).

AFGH algorithm's parameters and functions are followed. G_1 and G_2 are groups of prime order q with a bilinear map.

4.2 Operation

There are four kinds of system nodes:

- *Directory Servers (DS)* play the same role as directory servers in the Tor system.
- *Registration Database (RD)* is a trusted third party.
- *Key Generator (KG)* is a trusted third party.
- *Peer onion routers (POR)* POR's forward messages as OR's in Tor. Also, every user node is a POR.

4.3 Protocol Phases

4.3.1 Setup Phase

The RD is the main part of the anonymity service provider. RD has a confidential database and has sufficient computing ability. It also maintains a website to provide a part of keys for anonymous communication. KG is a trusted third party and does not require any kind of storage mechanism. At the setup stage, KG generates a secret $SEC_{KG} = x$ and a public key pair $\{PK_{KG}, SK_{KG}\}$. The DS setup is similar to the Tor directory servers. Table 4.1 provides the key possession of each of the nodes.

Suppose user node A_i wants to send an anonymous message m to A_j. Figure 4.1 shows

Table 4.1 Key Possession

Node	Key Posession	Comments
A_i	sk_i, dk_2, ek_i	$\forall A_i \in S$
KG	$PK_{KG}, SK_{KG}, SEC_{KG}$	

a description of the onion and proxy re-encryption. Figure 4.2 shows the notations on our symbols used.

4.3.2 Registration Phase

1. User A_i selects at random $\theta_i \in \mathbb{Z}_q^*$, keeps $sk_i = \theta_i$ as his secret key. He calculates $\{g^{1/\theta_i}, g^{\theta_i}\}$. He also generates an AES key, this key is called *key* and encrypts it using *KG's* public key PK_{KG} and gets $(key)_{PK_{KG}}$, and then sends the tuple $\{ ID_i, g^{1/\theta_i}, g^{\theta_i}, (key)_{PK_{KG}}\}$ to KG through a secure TLS connection.

2. Upon receiving the tuple, RD verifies $e(g^{1/\theta_i}, g^{\theta_i}) = \mathrm{Z}$. If the equation holds, he will recognize the tuple is a valid registration information.

3. RD reandomly selects $u \in \mathbb{Z}_q^*$ and calculates $g^{u\theta_i}$.

4. RD sends $\{ g^{1/\theta_i}, g^{u\theta_i}, (key)_{PK_{KG}} \}$ to KG through a secure TLS connection.

5. KG calculates $\{g^{xu/\theta_i}, g^{u\theta_i/x}\}$. Secondly, KG decrypts $(key)_{PK_{KG}}$ using its secret key SK_{KG} and gets *key*. Thirdly, KG calculates $E_{key}(S(g^{x/\theta_i}))$.

6. KG sends $\{g^{xu/\theta_i}, E_{key}(S(g^{x/\theta_i})), g^{u\theta_i/x}\}$ to RD.

7. RD calculates $\{g^{x/\theta_i}, g^{\theta_i/x}\}$. He stores $\{ID_i, g^{1/\theta}, g^{x\theta_i}\}$ in the tracing key database.

8. RD publishes $\{ID_i, g^{x\theta_i}\}$.

9. RD forwards $E_{key}(S(g^{x/\theta_i})), g^{\theta_i/x}$ to A_i.

4.3.3 Communication Phase

This part describes the procedure of a communication session. Suppose user A_i wants to send an anonymous message $m \in G_2$ to user A_j. During the process of this communication, regular anonymity needs to be attained. Our system acts as follows:

Sender

This is what the sender does. The initial user A_i knows the routing network topology and selects a route, this is similar to the steps in the onion routing network. A_i generates a sequence of route information along with corresponding proxy re-encryption keys in the form of an onion. This is called the onion header. The message content part is called the payload. Figure 2 shows the structure of the onion header. Each layer contains the information for corresponding intermediate nodes.

Let the route be: $A_{r1}, A_{r2}, A_{r3}, \ldots, A_{rn}, A_j$. That is: the next hop from the intermediate node A_{rt} is to the node A_{rt+1}, *(t = 1,2,..., n)*, and A_j is the destination. The intermediate nodes acts as proxies during the message forwarding. They use corresponding proxy re-encryption key to re-encrypt the payload.

A_i randomly chooses $x_1, x_2, \ldots, x_n$ and y_1, y_2, y_n from $\mathbb{Z}_q^*$ and calculates $(x_1+x_2+\ldots+x_n)$ mod q and $(y_1 + y_2 + \ldots + y_n)modq$. A_i also calculates $g^{\theta_j/\theta i}$ from $g^{x\theta_j}$.

Generally the *(t - 1)th* layer contains the next hop A_{rt} and corresponding re-encryption key $rk_t^1 = xt$, $rk_t^2 = yt$. In particular, the most inner layer contains decryption keys $dk_0 = x_1 + x_2 + \ldots + x_n \bmod q, dk_1 = y_1 + y_2 + \ldots + y_n \bmod q, dk_2 = S(g^{x/\theta_i}), dk_3 = g^{x_j/\theta_i}$.

User A_i calculates the ciphertext $c_o = \{g^{\theta_i \cdot k/x, mZ^k}\}$ using the encryption key $ek_i = g^{\theta_i/x}$, where

k is a random number. User A_i send the encrypted payload $c_o = \{g^{\theta_i \cdot k/x}, mZ^k\}$ along with the onion header to the node A_{r1}.

Intermediate Nodes

This is what an intermediate node does. At an intermediate node A_{r1}, (t = 1,2,..., n) first the onion header is decrypted and the next hop A_{rt+1} is identified. Also from the onion header he learns the re-encryption key $rk_t^1 = y_t$, $rk_t^2 = y_t$. He then picks up the proxy key $rk_t^1 = y_t$, $rk_t^2 = y_t$ and re-encrypts the payload. A_{rt+1} gets:
$\{g^{x_1+x_2+...+x_t+(\theta_i \cdot k/x)}, mZ^{(k+y_1+y_2+...+y_t)}\}$ Then A_{rt} outputs the remained part of the onion header along with the payload c_t to the next hop A_{rt+1}.

Receiver

This is what the receiver does. At the node A_j, first A_{rn} the onion header is decrypted and therefore the decryption keys $dk_0 = x_1 + x_2 + \ldots + x_n \bmod q, dk_1 = y_1 + y_2 + \ldots + y_n \bmod q, dk_2 = S(g^{x/\theta_i}), dk_3 = g^{x_j/\theta_i}$ are known.
The payload received is denoted by $c_n = \{\alpha, \beta\}$. User A_j verifies whether dk_3 is valid. He calculates dk_3^{1/θ_j} and checks whether $(Sdk_3^{1/\theta_j}) = dk_2$ holds. If the verification fails, he throws out the received data. If the verification passes, user A_j decrypts the payload as follows: $m = \frac{\beta/Z^{dk_1}}{e(\alpha/g^{dk_0}, dk_3)^{(1/\theta_j)}}$. If the receiver finds out that this message is malicious, it is reported to a Law Enforcement Officer. The system will switch to the forensic investigation state from this point on.

4.4 Forensic Investigation

The general procedure is as follows: The victim sends a report along with collected evidence to a Law Enforcement Officer (LEO). LEO initiates the forensic investigation. LEO provides the evidence to *KG* and asks for cooperation. *KG* processes the evidence and gives converted evidence to LEO. LEO then sends it to the *RD* with a subpena. By searching *RD's* database, the source of the malicious message is found. Suppose the destination user node A_j finds out that the received message m is malicious and thus wants to find out the source of the message.

User node reports it to the Law Enforment Officer (LEO) with evidences which is acquired in step 7 on normal state. $EV_1 = m$, $EV_2 = mZ^k$, $EV_3 = \alpha/g^{dk_0} = g^{\theta_{ik/x}}$, $EV_4 = g^{x/\theta_i}$, $EV_5 = S(g^{x/\theta_i})$.

LEO investigates the case as follows:

1. LEO determines the message m is a malicious message and tracing the source is needed.
2. LEO verifies $S(EV_4) = EV_5$. If it does not hold, evidences are invalid and tracing terminates.
3. LEO verifies $EV_2/e(EV_3, EV_4) = EV_1$.
4. LEO takes $S(EV_4)$ to KG and asks for cooperation. If KG agrees, KG will compute $EV_4^{1/x}$ and gives it to LEO.
5. LEO takes $EV_4^{1/x}$ to RD with a subpena. With the cooperation of RD, the database is searched. If $EV_4^{1/x}$ appears in a tuple, the RD outputs corresponding *ID's*. If no tuple matches, it reports an error.

The tracing steps that are taken to trace the criminal is shown in Figure 4.2. In this Chapter, the A2S system design was described in detail and some mathematical background and theory was applied to construct the proxy-re-encryption scheme. The next Chapter will describe the A2S system implementation.

Table 4.2 A2S Notation

Symbol	Value	Comments
S		The set of system nodes
A_i		A system node
ID_i		The ID of node A_i
DS		The Directory Server
RD		The Register Database
KG		The Key Generator
$e()$		A bilinear mapping function
$E_k(m)$		AES encryption function with the key k
$S()$		KG's signature calculation function
PK_{KG}		KG's public key
SK_{KG}		KG's secret key
SEC_{KG}	x	KG's secret
ek_i	$g^{\theta_i/x}$	A_i's encryption key
sk_i	θ_i	A_i's secret key
rk_t^1	x_t	The first proxy re-encryption key for A_{rt} during the communication session between A_i and A_j
rk_t^2	y_t	The second proxy re-encryption key for A_{rt} during the communication session between A_i and A_j
dk_0	$x_1 + x_2 + \ldots + x_n \bmod q$	The first decryption key for A_j during the communication session between A_i and A_j
dk_1	$y_1 + y_2 + \ldots + y_n \bmod q$	The second decryption key for A_j during the communication session between A_i and A_j
dk_2	$S(g^{x/\theta_i})$	The third dencryption key for A_j during the communication session between A_i and A_j
dk_3	$g^{x\theta_j/\theta_i}$	The fourth dencryption key for A_j during the communication session between A_i and A_j
c_0	$\{\ g^{\theta/i*k/x}, mZ^k\ \}$	A_i ciphertext encrypted by the initiator A_i
c_t	$\{\ g^{x1+x2+\ldots+x_t+(\theta_i*k/x)}, mZ^{(k+y1+y2+\ldots+y_t)}\ \}$	A_i ciphertext re-encrypted by the intermediate node A_{rt}
EV_1	m	The first evidence provided to LEO
EV_2	mZ^k	The second evidence provided to LEO
EV_3	$g^{(\theta_i k/x)}$	The third evidence provided to LEO
EV_4	$g^{(x/\theta_i)}$	The fourth evidence provided to LEO
EV_5	$S(g^{(x/\theta_i)})$	The fifth evidence provided to LEO

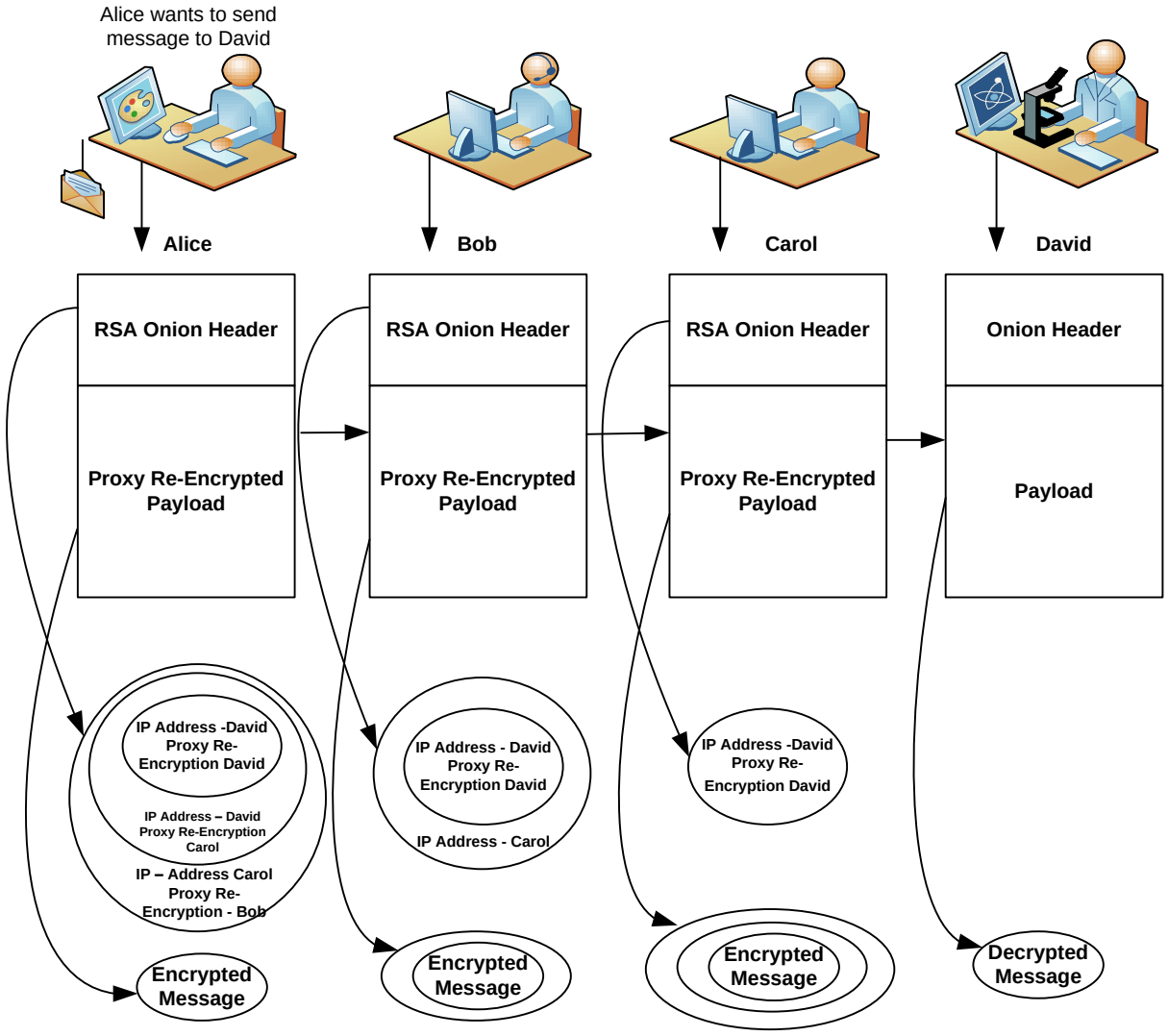

Figure 4.1 Message Routing

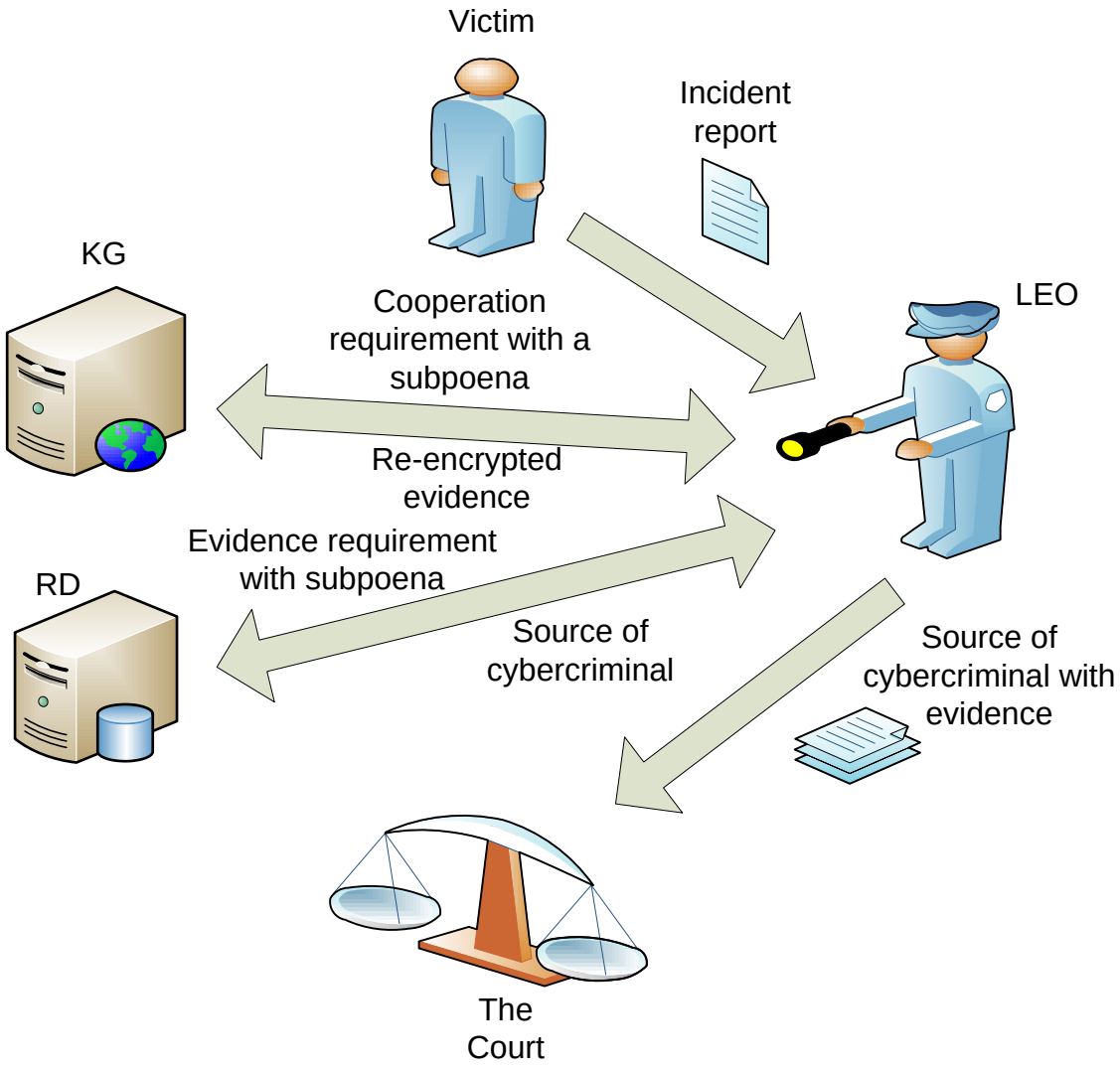

Figure 4.2 Tracing Process

CHAPTER 5. A2S IMPLEMENTATION

The implementation of the A2S system will be described. Details on the platform setup will be given, as well as some information on the evaluation of the system and a performance discussion.

5.1 Discussion

5.1.1 Programming Languages

Preferred Choice

C++ was chosen instead of Java for this implementation. C++ and Java are similar in syntax. This similarity makes it straightforward for C++ developers to learn Java. However, these two languages are different in many ways because of their different design goals.

C++ features are:

- Rapid development
- Security
- Portability
- Backward compatibility with C
- Memory Management
- Pointers
- Preprocessor

- C++ is compiled into native machine code
- Objects are passed by value
- Automatic type casting
- Methods must be declared virtual

Java features are:

- Garbage Collection
- Templates
- Multiple Inheritance
- Objects are passed by reference
- All methods are virtual

Java is compiled to virtual machine byte code and requires a virtual machine to run. So this implementation was chosen with C++ features in mind. The main reason for choosing C++ is that our crypto libraries are written in C and C++.

The Following Figures Illustrate some of the implementation's code:

Figure 5.1, Figure 5.2, Figure 5.3, Figure 5.4, Figure 5.5, Figure 5.6.

5.1.2 Object Oriented Programming

Benefits

What are the advantages of object-oriented programming? The most obvious advantage of object oriented programming's (OOP) is that it provides an excellent way to model the real world by associating data to the methods operating on them. This is accomplished by retaining the values of the member variable throughout the life of the object and preventing direct access to member variables from outside the class. Parts of the program outside the class may work

with the member variables only through methods that allow the caller to retrieve and modify the data. Large projects are more manageable because parts of the project are easily isolated and this is referred to as encapsulation.

Inheritance

Inheritance is another unique benefit of OOP. Inheritance encourages code reuse and it makes large libraries more manageable and easy to use. The A2S system takes advantage of this feature by inheriting the MIRACL library. This code reuse leads to faster development. Inheritance makes OOP a better way to program [18].

Polymorphism

Polymorphism means one name, many forms. With polymorphism multiple methods can have the same name but different functionality. It is achieved by method overriding and method overloading. Overriding, is also called run-time polymorphism. Overloading, is referred to as compile-time polymorphism. The difference is that for method overloading, the compiler determines which method will be executed, and this is decided when the code is compiled. For method overriding, the method that is beign used is determined at runtime [18]. The A2S system takes advantage of this feature by overloading the MIRACL library methods.

5.1.3 Latency and Bandwidth

Issues

What are the two major issues in networking performance? Any network can be measured by two major characteristics: latency and bandwidth. Latency refers to how long it takes a given bit of information to get through the network. In Chapter 1 we mentioned the service tool Ping which can be used to measure round trip latency. Bandwidth refers to the rate at which data moves through the network once communication is established. The perfect network would have infinite bandwidth and no latency. A pipe is a good analogy for a network. The time it takes for a water molecule to go through the whole pipe is determined by the length; this is analogous to the latency. The width of the pipe determines the bandwidth: how much

water can pass in a given time. Latency and bandwidth problems are often encountered when performing searches on the Internet. A good indication of good bandwidth but high latency is when a webpage takes a long time to display and then it appears quickly. On the other hand, if a webpage starts loading right away but takes a long time to load, this is a good indication of a low-latency, low-bandwidth connection.

5.2 Hardware

5.2.1 Computer Forensics Lab Computers

The implementation of the A2S system on a private computer network was done in the Computer Forensics lab. The system consists of five networked Dell OptiPlex GX280 2.8GHz Intel Pentium 4 Single Core, 80GB Hard Drive, 2GB RAM computers. Figure 5.7 shows a graphical description of the network setup. IP's were configured manually so every time one system was rebooted, the IP address does not get re-assigned. A hub and a new router with wireless capabilities was also used.

5.3 Software

5.3.1 Fedora Linux

Fedora Linux

The previous configuration on all the computers was erased and a fresh copy of Fedora Linux 13 was installed and all the software needed to run the programs was also installed.

5.3.2 Tor Test Software

Tor Test Software

At the beginning of the research, the question of how the Tor system worked was brought up by the research team. A successfull installation and configuration of the Tor software to run on all systems for a while was done.

5.3.3 Network Analysis

Network Analysis

Packets going through the network were analyzed with a tool called Wireshark.

5.3.4 MIRACL Crypto Library

The A2S system was implemented using a cryptographic library called MIRACL which is a big number library that implements all the primitives necessary to design big number cryptography in an application [19]. Some issues with our MIRACL library and the RSA decryption algorithm were evident in the testing phase of the implementation and as a consequence of this, it was decided to put the decryption function separately for the time being. The RSA decryption algorithm was conflicting with the proxy re-encryption algorithm.

5.3.5 OpenSSL Crypto Library

OpenSSL cryptographic library was used to create self signed certificates for the database and the clients and to develop software that creates a TLS connection between the RD and KG trusted third parties.

5.3.6 Kdevelop IDE

For the C++ development, a Kdevelop IDE and the g++ compiler were used and integrated subversion to keep track of the changes. For the graphical user interfaces, a tool called Qt Designer was used.

5.3.7 Debugger

For debugging purposes, gdb and a tool called data display debugger (DDD) was used.

5.3.8 Database Technologies

For the database, a MYSQL server with secure remote capabilities was implemented.

5.4 Evaluation

A good evaluation tool called sloccount that counts the number of lines of code was used and it was run in one of the system folders [37]. Figure 5.8 shows an estimation on how long it would take for a project this size, it also shows how many software developers are needed to do this job. Obviously, the code contains the library which contains a lot of code but it can give a good indication of how meaningful this project is.

5.5 Performance

The A2S system was run for a total of three months straight, every day including weekends to make sure all the operations done on the plaintext were correct and that the transmission was working properly, bugs were fixed along the way. Table 5.1 shows the performance of the system and the average of time it takes to compute the new proxy re-encryption algorithms. It is assumed that not all users have top of the line computers equipped with Dual Core processors and lots of memory so the system setup seems to measure good computing times due to this fact. According to the Tor website, there is no benchmark mode currently in the Tor system. They have an open ticket for this issue as part of a new enhancement. It is concluded that the performance of the various algorithms used (e.g. RSA, AES, number of circuits built, amount of memory, open connections) is not known at this time or if it is known it is only known to the Tor developers. One way of finding out how long the Tor software really takes is to modify the software, recompile it and run it with the new statistical modifications. It was mentioned earlier in Chapter 1 that a single TLS connection takes an enormous amount of time: it takes 669 ms to complete the TLS handshake.

Based on the amount it takes to complete a TLS handshake, a comparison is made to the A2S system and the software takes an average of 113 ms from the onion construction to proxy encrypting and sending the data. The reduced computation costs are an advantage to the A2S system. The A2S system performed well in a small test network and we are aware of the fact that the latency is greatly increased when the network grows in size as users join the network and it's hard to measure the time consumed. Overall, the A2S system performs better than

the current Tor system based on the facts mentioned previously.

Table 5.1 Performance Evaluation

Algorithm	**Time in milliseconds**
Key Generation	3 ms
RSA Encryption	10 ms
Proxy Encrption	10 ms
Proxy Re-encryption	10 ms
Proxy Decryption	20 ms
Onion Construction	60 ms
Source to Destination	452 ms

5.6 Deployment Considerations

We believe our system has a good chance of being accepted by the community. The system needs to be downloaded and installed without the need of many configuration settings and the lowest complexity for the end user. The more users there are on the network the more anonymity they will have. So basically, what we need to deploy it, is many people to test it and volunteers to help with the developing. We also need the following:

5.6.1 Scheduling Strategy

Best Route - Unlike the Tor system that the route is chosen for the user under what nodes are available at the time. The A2S system will display a user friendly map of the network to the user. This map will contain important information such as bandwidth, throughput and where the nodes are located. This will help the user decide what node to choose.

Scheduling

- Round robin - one of the simplest scheduling algorithms for processes in an operating system, which assigns time slices to each process in equal portions and in circular order. Round robin scheduling can be used as data packet scheduling in computer networks. The Tor network uses round robin.

- Fair queing - Since the size of data is varied on the network, some data can be very large and this can be favored over the other users. For example, one user could be using BitTorrent on Tor and other users on the network would be given lower priority. We plan to implement a Fair queuing scheduling algorithm for our system. Fair queuing takes into account data packet size to ensure each flow is given equal opportunity to transmit an equal amount of data. Current research shows that this method avoids Network congestion [41].

Software Development

The remaining issues have to be fixed and we need to port out the software to Windows systems, Mac systems and test it many times. A Unit Test procedure needs to be followed. A website needs to be developed so people know about this software project.
The deployment should be branched out in stages:

1. **Initial Stage** - The system will be launched as an email remailer.
2. **Middle Stage** - The system will be launched as a email remailer and chat service.
3. **Full Stage** - The system will be a full TCP/IP service capable of web browsing, chatting, and sending emails anonymously but with accountability features.

Security

- Penetration testers will be needed to attack the system to make sure there are no vulnerabilities.
- All crypto will need to be verified that is free of leaking data.

Other

MIT researchers were the last ones to produce a similar scheme but for one hop and bugs were discovered in their code, our scheme is multi-hop. The software needs to be licensed under a General Public License (GPL) or similar license. We need to start a Non-Profit Corporation.

We need to come up with a good name for the software, a name that does not confuse our system with the Tor project, a simple name that gets our point across to people. A small marketing campaign is needed to promote accountable anonymity in a positive way and we need to come up with a jingle. The system needs to be promoted in security and software conferences, and we need to give the software away for free so people can try it.

```
int
main()
{
    miracl *mip=mirsys(16,0);   // thread-safe ready. (32,0) for 1024 bit p
    ifstream common("publicparams.cfg");
    ifstream publickeysfile("public.keys");
    ifstream secretkeysfile("secret.keys");
    ifstream plaintext;
    ifstream ciphertext;
    ofstream ciphertextenc;
    ofstream decrypted;
    ofstream delegator;

    ECn P,Ppub;
    ZZn2 gid,cube,w, Z;
    Big p,q,r,x,y, paramsqsquared;
    int i,bits;
    int ZZn2Tochar (ZZn2 &z, char *c, int s);
    int BigTochar (Big &x, char *c, int s);

   common >> bits;
   mip->IOBASE=16;
   common >> p >> q;
   paramsqsquared = pow(q, 2);

  common >> x >> y;

#ifdef AFFINE
    ecurve(0,1,p,MR_AFFINE);
#endif
#ifdef PROJECTIVE
    ecurve(0,1,p,MR_PROJECTIVE);
#endif

    P.set(x,y);

    common >> x >> y;
    cube.set(x,y);
    mip->IOBASE=10;
    cout << "Cube: " << cube << endl;
    mip->IOBASE=16;

    common >> x >> y;
    Z.set(x, y);
    mip->IOBASE=10;
    cout << "Z: " << Z << endl;
    mip->IOBASE=16;

    ZZn2 Zpub1;
    ECn Ppub2;
    publickeysfile >> x >> y;
    Zpub1.set(x,y);
    publickeysfile >> x >> y;
    Ppub2.set(x,y);
    publickeysfile.close();
    common.close();
```

Figure 5.1 Sample Code 1

```
thetaiinv = inverse(thetai, q);
//g^(1/thetai) Ai idkey1
c11 = thetaiinv * P;
//g^thetai is Ai idkey2
c12 = thetai * P;

thetaj = rand(q);
thetajinv = inverse(thetaj,q);
c13 = thetajinv * P;
c14 = thetaj * P;

//KG idkey
kgx = rand(q);
kgxinverse = inverse(kgx, q);

//g^x*thetaj/thetai
Big kgxres = kgx * thetaj * thetaiinv;
key2 = kgxres * P;

Big te = thetai * kgxinverse;
Enckey_a = te * P;

Big k = rand(q);
 //this is c'
ciphertext1 = k * Enckey_a;


plaintext.open("testfile.txt", ios::in | ios::binary);
if (!plaintext)
{
  cout << "unable to open file" << plaintext << "\n"<<endl;
}

int size;
plaintext.seekg(0,ios::end);
size = plaintext.tellg();
plaintext.seekg(0,ios::beg);
buffer = new char[size];
memset(buffer,0,sizeof(buffer));
plaintext.read(buffer, size);
plaintext.close();
cout.write(buffer,size);
 // Compute res2 = plaintext * Z^k

 // Next, encode the plaintext as Big
Big msg = 0;
msg = from_binary(size, buffer);
cout << "msg : " << msg << endl;
// Compute res2 = plaintext * Z^k
zPlaintext.set(msg,0);
temp = pow(Z, k);
c22 = zPlaintext * temp;
cout << "encrypt: Z = " << Z << endl;
cout << "encrypt: temp = " << temp << endl;
cout << "encrypt: plaintext = " << zPlaintext << endl;
cout << "encrypt: c11 = " << c11 << endl;
cout << "encrypt: c22 = " << c22 << endl;
 delete[] buffer;
```

Figure 5.2 Sample Code 2

```
// Set the ciphertext2 file with (ciphertext1, c22)
mip->IOBASE=16;
ciphertext1.get(x,y);
ciphers << x << endl;
ciphers << y << endl;
c22.get(x,y);
ciphers << x << endl;
ciphers << y << endl;

ciphers.close();

////////////////////////////////////////////
////////////////////////////////////////////
//decrypting message here to test if we got it
//second level ciphertext, this decryption part
//wont be part of encryption function
//we are just checking here
////////////////////////////////////////////
////////////////////////////////////////////

ECn del;
ZZn2 temp4;
ZZn2 result2;
Big temp5 = thetaiinv * kgx;
del = (temp5 * P);

if (ecap(ciphertext1, del, q, cube, temp4) == FALSE) {
  cout << "Pairing computation failed.  Please try again with a different seed." << endl;
  exit(1);
}

result2 = c22 / temp4;
result2.get(msg);

////////////////////////////////////////////////////
//we need to open ciphertext2 file and get c11 and c22
////////////////////////////////////////////////////
ciphertext.open("ciphertexts2", ios::in | ios::binary);
 if (!ciphertext)
{
  cout << "unable to open file" << ciphertext << "\n"<<endl;
  exit(1);
}

ciphertext.seekg(0,ios::end);
size = ciphertext.tellg();
ciphertext.seekg(0,ios::beg);
buffer = new char[size];
memset(buffer,0,sizeof(buffer));
ciphertext.read(buffer, size);
ciphertext.close();
int n=0;
n = ZZn2Tochar(result2, buffer, size);
cout << "n = " << n << endl;

cout << "Decrypting Message\n";
```

Figure 5.3 Sample Code 3

```
decrypted.open("decrypted2.txt", ios::out | ios::binary);
if(!decrypted)
{
  cout << "Unable to open file" << decrypted << endl;
  exit(1);
}

int c;
 //Do whatever with buffer
for (c=0;c<n;c++)
{
  printf("%c", buffer[c]);
  decrypted << buffer[c];

}

decrypted.close();

delete[] buffer;

////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////
//transformation function () will generate c''1 & c''2
//take c'1 & c'2 ciphertext generated by Encryption function
//need to open ciphertext2 file
////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////

Big PREKey_b = rand(q);
cout << "PREKey_b: " << PREKey_b << endl;

ciphertext.open("ciphertexts2", ios::in | ios::binary);
if (!ciphertext)
{
  cout << "unable to open file ciphertexts2" << ciphertext << "\n"<<endl;
}

ECn ciph1;
ZZn2 ciph2;
ECn ciphII1;
ZZn2 ciphII2;

ciphertext >> x >> y;
ciph1.set(x,y);
ciphertext >> x >> y;
ciph2.set(x,y);
//close ciphertext2.txt file
ciphertext.close();

//c''1 = (c'1) ^ PREKey_b
ciphII1 = PREKey_b * ciph1;
//c''2 = (c'2) ^ PREKey_b
ciphII2 = pow(ciph2, PREKey_b);

// Set the transformation file with (ciphII1, ciphII2)
ciphertextenc.open("transformation.txt", ios::out | ios::binary);
if (!ciphertextenc)
{
```

Figure 5.4 Sample Code 4

```
cout << "unable to open file transformation.txt" << ciphertextenc << "\n"<<endl;
}

mip->IOBASE=16;
ciphII1.get(x,y);
ciphertextenc << x << endl;
ciphertextenc << y << endl;
ciphII2.get(x,y);
ciphertextenc << x << endl;
ciphertextenc << y << endl;

//close transformation file
ciphertextenc.close();

////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////
//decrypting here just to see we have this right
//we need to open ciphertext
////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////

//ECn del;
//ZZn2 temp4;
//ZZn2 result2;
ZZn2 temp6;
del = key2;
cout << "key2: " << key2 << endl;
//key1 is the product of all intermediate nodes proxy keys
//right now we have one so its prekey_b.

//forget about key3 ...only used for verification.

Big key1 = PREKey_b;
 cout << "key1 = PREKey_b: " << key1 << endl;
Big key4 = thetaj;
 cout << "key4: " << key4 << endl;
Big k4inverse = inverse(key4, q);
 cout << "k4inverse: " << k4inverse << endl;
Big k1inverse = inverse(key1, q);
 cout << "k1inverse: " << k1inverse << endl;
ECn Alphak1inverse;
Alphak1inverse =  k1inverse * ciphII1;
 cout << "Alphak1inverse : " << Alphak1inverse << endl;


if (ecap(Alphak1inverse, del, q, cube, temp6) == FALSE) {
  cout << "Pairing computation failed.  Please try again with a different seed." << endl;
  exit(1);
}

result2 = pow(ciphII2 , k1inverse) / pow(temp6, k4inverse);
//result2 = temp7 / temp8;
cout << "result2 get bigs: = " << result2 << endl;
result2.get(x,y);

//we need to get this number back
//7468697320697320612074657374210A,0
```

Figure 5.5 Sample Code 5

```
    ////////////////////////////////////////////////////
    //we need to open transformation.txt file and get c''1 and c''2
    ////////////////////////////////////////////////////
    ciphertext.open("transformation.txt", ios::in | ios::binary);
     if (!ciphertext)
    {
      cout << "unable to open file" << ciphertext << "\n"<<endl;
      exit(1);
    }

    ciphertext.seekg(0,ios::end);
    size = ciphertext.tellg();
    ciphertext.seekg(0,ios::beg);
    buffer = new char[size];
    memset(buffer,0,sizeof(buffer));
    ciphertext.read(buffer, size);
    ciphertext.close();
    //int n=0;
    n = ZZn2Tochar(result2, buffer, size);
    cout << "n = " << n << endl;

    cout << "Decrypting transformation Message\n";

    decrypted.open("decryptedtrans.txt", ios::out | ios::binary);
    if(!decrypted)
    {
      cout << "Unable to open file" << decrypted << endl;
      exit(1);
    }

    //int c;
     //Do whatever with buffer
    for (c=0;c<n;c++)
    {
      printf("%c", buffer[c]);
      decrypted << buffer[c];

    }

    decrypted.close();

    delete[] buffer;


    return 0;
}
```

Figure 5.6 Sample Code 6

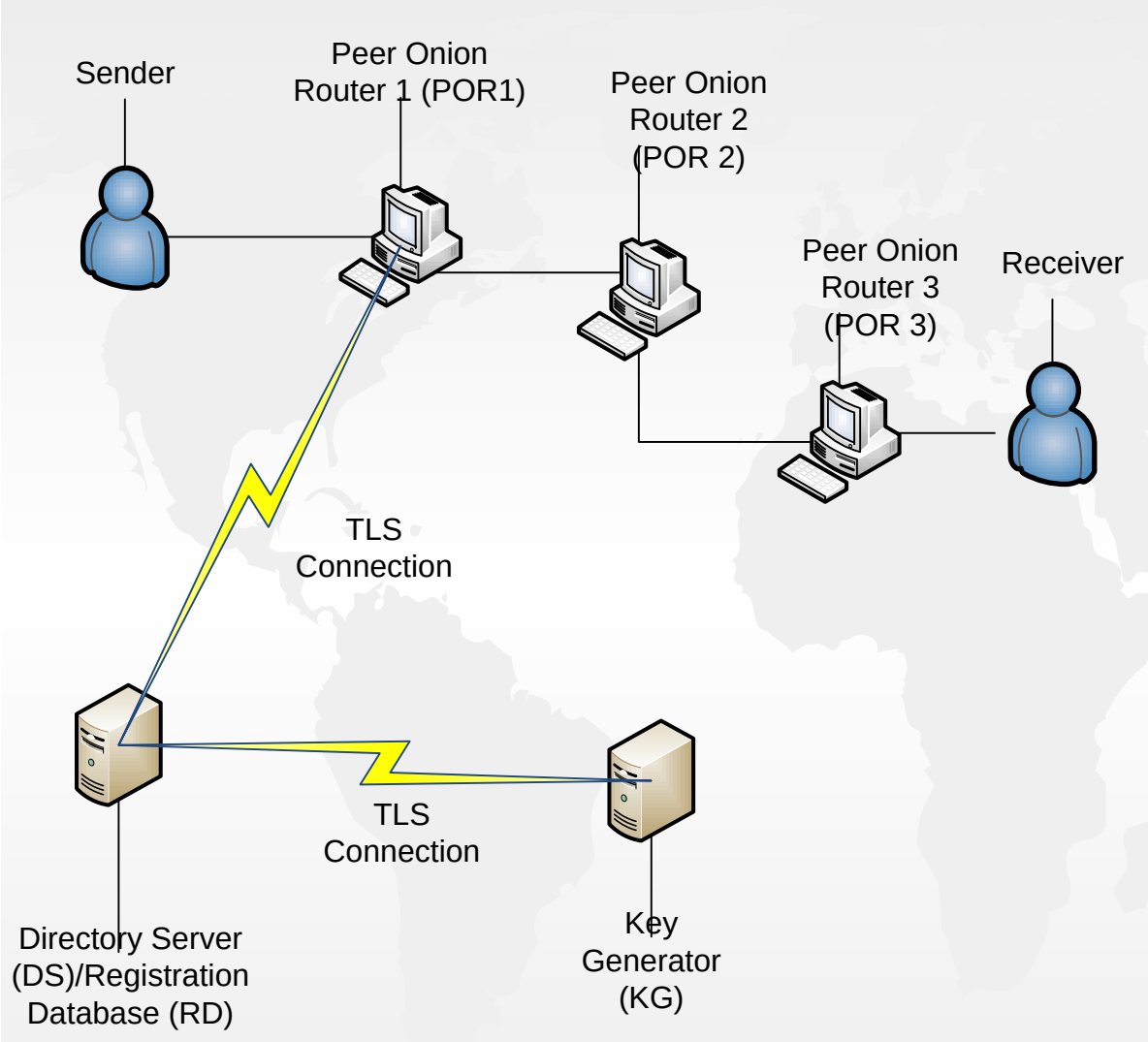

Figure 5.7 A2S Network

```
Creating filelist for 2
Categorizing files.
Finding a working MD5 command....
Found a working MD5 command.
Computing results.
SLOC    Directory       SLOC-by-Language (Sorted)
92260   2               ansic=46584,cpp=45611,asm=65
Totals grouped by language (dominant language first):
ansic:        46584 (50.49%)
cpp:          45611 (49.44%)
asm:             65 (0.07%)
Total Physical Source Lines of Code (SLOC)              = 92,260
Development Effort Estimate, Person-Years (Person-Months) = 23.14 (277.64)
(Basic COCOMO model, Person-Months = 2.4 * (KSLOC**1.05))
Schedule Estimate, Years (Months)                       = 1.77 (21.21)
(Basic COCOMO model, Months = 2.5 * (person-months**0.38))
Estimated Average Number of Developers (Effort/Schedule)  = 13.09
Total Estimated Cost to Develop                         = $ 3,125,401
(average salary = $56,286/year, overhead = 2.40).
SLOCCount, Copyright (C) 2001-2004 David A. Wheeler
```

Figure 5.8 SLOCCount Evaluation Tool

CHAPTER 6. SUMMARY AND FUTURE WORK

6.1 Summary

This thesis introduced the *Accountable Anonymity* concept and implemented it in the A2S system, a more *rational* anonymity system than current *absolute* anonymity systems. This is a lightweight system prototype that provides accountability features and proves that the new proxy re-encryption scheme works in a multi-hop environment, a feature that has not been implemented by the research community, to the best of the research team's knowledge. Examples were given of why *Accountability* is needed in an anonymous system. An overview of current work on *Anonymity* systems was given, as well as a brief mathematical background to understand the methods and protocols used for our new proxy re-encryption scheme. System implementations details were given and showed through our experiments that the A2S system is scalable and practical. This research was also the product of two technical papers submitted to the ACM Conference on Computer and Communications Security (CCS) 2011, and was part of a collaborative team effort with Gang Xu, another Ph.D. student.

6.2 Future Work

Most of our time was consumed in ensuring the algorithms were encrypting and decrypting correctly. This was done to make sure that the system is reliable. Therefore, the following, are minor changes that are needed for future work. Our Registration phase needs additional work, in the existing version, keys are created and used in order to save some time in the other phases. The NTRU [16] a new ring based public key cryptosystem can be used instead of RSA scheme to improve our performances and to avoid conflicting with the MIRACL library. Every intermediate hop needs to be a client, not just a server. To mitigate SQL injection attacks

to our Directory Server, we plan to use stored procedures and would not let any regular user access the database with administration rights.

BIBLIOGRAPHY

[1] Anonymizer. Anonymity service. Website. Accessed: May 23, 2011 available at https://www.anonymizer.com.

[2] J. Assange. Wikileaks. Website. Accessed: May 24, 2011 available at http://www.wikileaks.ch/.

[3] D. Vergnaud B. Libert. Unidirectional chosen-ciphertext secure proxy re-encryption. In *In PKC08, LNCS*.

[4] J. Bethencourt. Intro to bilinear maps. http://www.cs.berkeley.edu/~bethenco/bilinear_maps.pdf.

[5] G. Tsudik C. Gülcü. Mixing E-mail with Babel. In *Proceedings of the Network and Distributed Security Symposium - NDSS '96*, pages 2–16. IEEE, February 1996. http://www.freehaven.net/anonbib/cache/babel.html.

[6] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, February 1981.

[7] DOJ. Computer crime and intellectual property section. Website. Accessed: May 17, 2011 available at http://www.cybercrime.gov/.

[8] C. Drew. Stolen data is tracked to hacking at lockheed. Website. Accessed: June 04, 2011 available at http://www.nytimes.com/2011/06/04/technology/04security.html?_r=2.

[9] M. Green S. Hohenberger G. Ateniese, K. Fu. Improved proxy re-encryption schemes with applications to secure distributed storage. In *IN NDSS*, pages 29–43, 2005.

[10] D. Hopwood N. Mathewson G. Danezis, R. Dingledine. Mixminion: Design of a type iii anonymous remailer protocol. In *In Proceedings of the 2003 IEEE Symposium on Security and Privacy*, pages 2–15, 2003.

[11] GAO. Public and private entities face challenges in addressing cyber threats. Website, 2011. Accessed: May 15, 2011 available at http://www.gao.gov/new.items/d07705.pdf.

[12] S. Helmers. A brief history of anon.penet.fi - the legendary anonymous remailer. Website, 1997. Accessed: May 23, 2011 available at http://www.december.com/cmc/mag/1997/sep/helmers.html.

[13] Tor Project Inc. Anonymity online. Website. Accessed: May 24, 2011 available at https://www.torproject.org/.

[14] K. E. Seamons J. E. Holt. Nym: Practical pseudonymity for anonymous networks. Website, June 2006. Accessed: May 25, 2011 available at http://isrl.cs.byu.edu/pubs/isrl-techreport-2006-4.pd.

[15] P. Alpeyev J. Galante, O. Kharif. Sony attack shows amazon′s cloud service lures hackers at pennies an hour. May 2011. http://www.bloomberg.com/news/2011-05-15/sony-attack-shows-amazon-s-cloud-service-lures-hackers-at-pennies-an-hour.html.

[16] J.H. Silverman J. Hoffstein, J. Pipher. Ntru: A ring-based public key cryptosystem. In *Lecture Notes in Computer Science*, pages 267–288. Springer-Verlag, 1998.

[17] N. Koblitz. Elliptic curve cryptosystems. In *Mathematics of computation*, pages 203–209. American Mathematical Society, 1987.

[18] R. Lewallen. 4 major principles of object-oriented programming. http://codebetter.com/raymondlewallen/2005/07/19/4-major-principles-of-object-oriented-programming/.

[19] Shamus Software Limited. Crypto library. Website. Accessed: May 29, 2011 available at http://www.shamus.ie/.

[20] M. Strauss M. Blaze, G. Bleumer. Divertible protocols and atomic proxy cryptography. In *In EUROCRYPT*, pages 127–144. Springer-Verlag, 1998.

[21] R. Morris M. J. Freedman. Tarzan: A peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 2002)*, Washington, DC, November 2002. http://pdos.lcs.mit.edu/tarzan/docs/tarzan-ccs02.pdf.

[22] B. Plattner M. Rennhard. Introducing MorphMix: Peer-to-Peer based Anonymous Internet Usage with Collusion Detection. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2002)*, Washington, DC, USA, November 2002. http://www.freehaven.net/anonbib/cache/morphmix:wpes2002.pdf.

[23] V. S. Miller. Elliptic curves and their use in cryptography. In *DIMACS Workshop on Unusual Applications of Number Theory*, 1997.

[24] C. Grothoff N. Evans, R. Dingledine. A practical congestion attack on tor using long paths. In *Proceedings of the 18th USENIX Security Symposium*, August 2009. http://freehaven.net/anonbib/papers/congestion-longpaths.pdf.

[25] U.S. Navy. Onion routing. Website. Accessed: May 23, 2011 available at http://www.onion-router.net/.

[26] I. Goldberg P. Boucher, A. Shostack. Freedom systems 2.0 architecture. White paper, Zero Knowledge Systems, Inc., December 2000. http://freehaven.net/anonbib/cache/freedom2-arch.pdf.

[27] L. Ponemon. Cost of data breach clims higher. May 2011. Accessed: May 15, 2011 available at http://www.ponemon.org/blog/post/cost-of-a-data-breach-climbs-higher.

[28] C. Cornelius S. W. Smith P.P. Tsang, A. Kapadia. Nymble: Blocking Misbehaving Users in Anonymizing Networks. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, September 2009. http://www.cs.indiana.edu/~kapadia/papers/nymble-tdsc-preprint.pdf.

[29] S. Hohenberger R. Canetti. Chosen-ciphertext secure proxy re-encryption. In *Proceedings of the 14th ACM conference on Computer and communications security*, CCS '07, pages 185–194, New York, NY, USA, 2007. ACM.

[30] P. Syverson R. Dingledine, N. Mathewson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, August 2004. https://svn.torproject.org/svn/projects/design-paper/tor-design.html.

[31] I. Goldberg R. Henry, K. Henry. Making a Nymbler Nymble using VERBS (Extended Version). In *Proceedings of the 10th Privacy Enhancing Technologies Symposium (PETS 2010)*, Berlin, Germany, July 2010. http://www.cacr.math.uwaterloo.ca/techreports/2010/cacr2010-05.pdf.

[32] L. Adleman R. L. Rivest, A. Shamir. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 26:96–99, January 1983.

[33] Michael Reiter and Aviel Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1), June 1998. http://avirubin.com/crowds.pdf.

[34] E. Rescorla. *SSL and TLS: Designing and Building Secure Systems*. Addison Weley, INDIANA, 2001.

[35] P. Palfrader L. Sassaman U. Möller, L. Cottrell. Mixmaster Protocol — Version 2. IETF Internet Draft, July 2003. http://mixmaster.sourceforge.net/,.

[36] M.E. Hellman W. Diffie. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976.

[37] D.A. Wheeler. Linux kernel 2.6: It's worth more! Website. Accessed: May 29, 2011 available at http://www.dwheeler.com/essays/linux-kernel-cost.html/.

[38] Wikipedia. Groups(mathematics). http://en.wikipedia.org/wiki/Group_(mathematics).

[39] Wikipedia. Accountability. Website, 2011. Accessed: May 15, 2011 available at http://en.wikipedia.org/wiki/Accountability#cite_note-13.

[40] Wikipedia. Anonymity. Website, 2011. Accessed: May 15, 2011 available at http://en.wikipedia.org/wiki/Anonymity.

[41] Wikipedia. Fair queuing. Website, 2011. Accessed: June 7, 2011 available at http://en.wikipedia.org/wiki/Fair_queuing.

[42] Wikipedia. Morris worm. Website, 2011. Accessed: June 5, 2011 available at http://en.wikipedia.org/wiki/Morris_worm.

[43] N. Hopper Z. Lin. Jack: Scalable accumulator-based nymble system. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES2010)*. ACM, October 2010. http://www-users.cs.umn.edu/~hopper/Jack_wpes.pdf.