1997

Democratic key escrow scheme

Chor Wah Man
University of Wollongong

Follow this and additional works at: <https://ro.uow.edu.au/theses>

University of Wollongong

Copyright Warning

You may print or download ONE copy of this document for the purpose of your own research or study. The University does not authorise you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site.

You are reminded of the following: This work is copyright. Apart from any use permitted under the Copyright Act 1968, no part of this work may be reproduced by any process, nor may any other exclusive right be exercised, without the permission of the author. Copyright owners are entitled to take legal action against persons who infringe their copyright. A reproduction of material that is protected by copyright may be a copyright infringement. A court may impose penalties and award damages in relation to offences and infringements relating to copyright material.

Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.

Unless otherwise indicated, the views expressed in this thesis are those of the author and do not necessarily represent the views of the University of Wollongong.

Recommended Citation

Man, Chor Wah, Democratic key escrow scheme, Master of Science (Hons.) thesis, Department of Computer Science, University of Wollongong, 1997. <https://ro.uow.edu.au/theses/2796>

DEMOCRATIC KEY ESCROW SCHEME

A thesis submitted in partial fulfilment of the
requirements for the award of the degree

Master of Science (Honours)

from

UNIVERSITY OF WOLLONGONG

by

Chor Wah Man

Department of Computer Science
1997

Acknowledgement

I am grateful to my supervisor Dr. Rei Safavi-Naini for giving me an opportunity to undertake this work. Her constant guidance and encouragement enabled timely completion of this thesis.

Abstract

Cryptography is a dual-edged sword. On the one hand, it allows secure electronic communications by legitimate users. On the other hand, it can be used by criminals to thwart law enforcement efforts and by foreign countries to prevent security agencies from gathering intelligence information about these countries.

However, the widespread use of cryptography is critical to the establishment of the Information Superhighway. In order to prevent illegal users from enjoying the same convenience, the most straight forward solution is to escrow every user's private key.

Governments' prime concern focuses on law enforcement and national security. However, users' major worries are the security and potential abuses of these systems by the authorities. This is particularly important because, with the explosive growth of the Internet, a user's private key can be used to access a wide range of information about him/her.

In recent years, numerous key escrow schemes have been proposed but their success will ultimately depend on the trust and acceptance by users.

Unfortunately, all proposed systems suffer from one common weakness: concentration of trust. Even if the private key is split among several trustees, it is still too concentrated and thus too insecure.

Another subtle yet equally important weakness is the contradiction to the spirit of democracy. In these schemes, user's fundamental right of privacy is at the mercy of the governments and a few trustees. It is unfair for most legitimate users to hand over their private keys to the authorities (or their agents), even before they have started any electronic communication. Given the fact that governments are strong and users are weak, this is unfair and undemocratic.

This thesis proposes a key escrow scheme which is not only secure but also fair and democratic by introducing the concept of "electronic jury" similar to the jury system in the common law. This is a new mechanism to achieve proper balance of power and mutual trust by both users and authorities by distributing the key shares in the hands of electronic jury until they are required for lawful recovery.

Publication arising out of this thesis

Chor Wah Man, R. Safavi-Naini. "Democratic Key Escrow Scheme," in *Second Australasian Conference on Information Security and Privacy (ACISP'97)*, July 1997, published by Springer-Verlag in the Lectures Notes of Computer Science series.

I hereby declare that I am the sole author of this thesis.

I also declare that the material presented within is my own work, except where duly acknowledged, and that I am not aware of any similar work either prior to this thesis, or currently being pursued.

I certify that this thesis has not been submitted for a degree in any other university or institution.

Chor Wah Man

May 1997

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Nature of the Problem	2
1.3	Objectives	3
1.4	Scope of Study	3
1.5	Outline	4
2	Importance of Cryptography and Information Superhighway	5
2.1	Privacy Issue	5
2.2	Development of Cryptography	6
2.2.1	Symmetric Cryptography and Its Weaknesses	7
2.2.2	Public key Cryptography	8
2.2.3	Importance of Cryptography	8
2.3	Potential Threats of Cryptography	9
2.3.1	Threats	9
2.3.2	Crypto Anarchy	10
2.4	Government Policy	11
2.5	Information Superhighway	12
2.5.1	Cryptographic Services	12
2.5.2	Key Management Infrastructures	12
3	Challenge	14
3.1	Problems with Existing Key Escrow Schemes	14
3.2	Orwellian World	15
3.3	Dilemma	16
3.4	This Work	17
4	Related Works	19
4.1	Escrowed Encryption Standard (EES)	19

4.1.1	Brief Description of EES	19
4.1.2	Operation	20
4.1.3	Criticism	21
4.1.4	Clipper II & III	22
4.2	Fair Public Key Cryptosystems	22
4.3	Encapsulated Key Escrow	23
4.4	Verifiable Partial Key Escrow	23
4.5	A Proposed Architecture for Trusted Third Party Services . .	24
4.6	Oblivious Key Escrow	24
4.7	Translucent Cryptography	25
4.8	Binding Cryptography	25
4.9	A Matter of Trust	26
5	Jury System	27
5.1	Introduction	27
5.2	Origin	28
5.3	Early History	29
5.4	Characteristics	29
5.5	Selection Procedure	30
5.6	Merits	30
5.7	Trials in Continental Law	31
5.8	Bulwark of Civil Liberty	32
6	Democratic Key Escrow Schemes	34
6.1	Electronic Jury System	34
6.1.1	New Mechanism	34
6.1.2	Abstract Model for Electronic Jury System	35
6.2	System Description	36
6.2.1	Entities	36
6.2.2	Requirements	37
6.3	Scheme 1	38
6.3.1	Key Registration	38
6.3.2	Key Recovery for Law Enforcement	40
6.3.3	Efficiency Consideration	40
6.4	Scheme 2	41
6.4.1	Key Registration	41
6.4.2	Key Recovery for Law Enforcement	43

6.4.3	Efficiency Consideration	43
6.5	Scheme 3	43
6.5.1	Key Registration	44
6.5.2	Key Recovery for Law Enforcement	45
6.5.3	Efficiency Consideration	46
7	Security Analysis	47
7.1	Strengths	47
7.2	Possible Attacks	49
7.3	Subliminal Channel	50
8	Practical Consideration	53
8.1	Key Recovery	53
8.2	Selection Criteria of Jurors	53
8.3	Key Renewal	54
8.4	Share Transfer	54
8.5	Implementation	54
8.6	Wider Applications	54
9	Conclusion	56
A	Source Code for Implementation of Scheme 3	57

The only sure bulwark of continuing liberty is a government strong enough to protect the interests of the people, and a people strong enough and well enough informed to maintain its sovereign control over the government [37].
President Roosevelt of USA

Chapter 1

Introduction

This chapter firstly explains the origin of the problem and the motivation of this thesis. It then discusses the nature of this problem. Objectives are then set out as a roadmap for the remaining parts of this thesis. As key escrow is a very complex issue, this chapter places a scope of study in this thesis.

1.1 Motivation

Rapid advances in computer and telecommunication technologies have revolutionarized people's ways of communications, and have become an essential element of modern life. They have removed distance as a geographical barrier and have turned the world into a big global village.

As open communication networks are insecure, users need to use cryptography to protect their privacy in modern communications. However, cryptography is a dual-edged sword. While strong cryptography provides absolute privacy unavailable before, it can be abused by criminals to avoid electronic surveillance so that it also presents a serious threat to effective law enforcement and national security. In fact, cryptography has upset the traditional balance of power between individuals and authorities.

Government's past strategy was to impose strict control in the use of cryptography (e.g., by limiting the length of the encryption key). This seriously reduces the protection to legitimate users and thus cannot be accepted as a long-term solution for secure communications in the emerging information society. Realising this, government's new approach is to allow the widespread use of cryptography subject to the condition that every user must escrow his/her decryption key with a trusted party to enable emergency key recovery for law enforcement purposes. Besides other inherent security problems,

such key escrow systems may be abused by the authorities to intrude into users' privacy in the name of law enforcement by obtaining the decryption keys to conduct wiretaps. Moreover, it is conceptually wrong to force users (either by law or by creating a substantial market [26]) to hand over their decryption keys to the authorities even before they have started any electronic communications. Instead, they should be required to do so only if they are suspected of committing a serious crime.

Thus, before one can fully relax and enjoy the fruit of the proximity and accessibility provided by the emerging Information Superhighway, one must guard against barbarism due to the abuse of privacy by users or tyranny due to the abuse of power by authorities in the emerging electronic community. This dilemma is the motivation behind this thesis.

1.2 Nature of the Problem

In order to strike a new balance of power between individuals and authorities, one must fully appreciate the nature of this problem:

- This is an ethical problem because it involves social values and human judgement of what is good and what is bad about privacy. If one believes that privacy is evil and intrusion of one's privacy by government is meritorious because it only shows that the government really cares about him/her, then the discussion can end here. Also, one believes that a democratic society is better than a totalitarian one.
- This is a political problem because there is power struggle between users and authority. It is the new battle field between ruling class and the ruled class which should ideally lead to a rational compromise or balance of power between two classes.
- This is a legal problem because it involves enforcement of law and order (or policing) extended to the electronic media. A suspect must be assumed innocent until proven guilty and the process must not only be fair but also seen to be fair. In case of doubt, the suspect should gain the credit. These legal principles should also be held in the cyberspace.
- This is a cryptographic problem because a cryptographic solution must be sought to support the best compromise at national and international levels.

Most important of all, the solution to this problem must be trusted and accepted by users because their participation is paramount to the ultimate success of any key escrow system.

1.3 Objectives

This thesis proposes a new approach to the key escrow system. The crux of the whole issue is the mutual trust by both users and authorities. The jury system in common law has earned the trust of both the general public and authorities as an independent and impartial tribunal for settling disputes and it has a long history of protecting individual's civil liberty from the encroachment of the authorities. This thesis investigates how this implicit trust system can be applied to the key escrow problem. As a result, the objectives of this thesis are as follows:

- Evaluate the impact of cryptography on the emerging Information Superhighway.
- Evaluate current key escrow schemes.
- Explore the jury system and its characteristics.
- Abstract the trust model behind the jury system.
- Apply this trust model to the key escrow situation.
- Illustrate how to build a secure, fair and democratic key escrow scheme using this model.
- Analyse the security of this new scheme.
- Consider some practical issues related to this new scheme.

1.4 Scope of Study

This thesis is mainly concerned with large-scale national and international key escrow systems for confidentiality purpose only and it will not discuss the following topics:

- Commercial key escrow systems within the user's organization which serves primarily to protect against the loss of data within the organization, e.g., Nortel's Entrust or Bankers Trust.
- Cryptographic keys for data integrity and authentication because they do not encrypt messages and do not interfere with law enforcement investigation and intelligence gathering. In fact, they tend to bind the source and message together and help to enforce accountability.

1.5 Outline

The outline of this thesis conforms very closely with the above objectives.

Chapter 2 starts by defining privacy and then briefly describes the development and application of cryptography to protect privacy in the emerging Information Superhighway. Chapter 3 highlights some problems and potential dangers with existing key escrow schemes and summarises the contribution of this thesis to solve these problems. In chapter 4, a review of some of the related works and their drawbacks is provided. It points out the importance of mutual trust by users and authorities in these systems and leads to the investigation of the jury system in chapter 5. Chapter 5 briefly summarises the development and features of the jury system and its similarities with the counterparts in continental law. Based on the trust model implicit in the jury system, chapter 6 proposes a new approach to key escrow by the introduction of "electronic jury system" and shows how to design three different democratic key escrow schemes. Chapter 7 evaluates the strengths and possible attacks of the three proposed schemes. Some counter-measures are proposed to prevent or minimize these attacks. Chapter 8 explores some practical issues related to the implementation of these schemes. Chapter 9 draws a conclusion about democratic key escrow schemes. In the appendix, the source code for implementation of scheme 3 in Java language is also included.

Chapter 2

Importance of Cryptography and Information Superhighway

This chapter briefly describes the development and application of cryptography to protect privacy in the emerging Information Superhighway.

2.1 Privacy Issue

This section gives an informal review of the privacy issue because the protection of privacy is the central theme in this thesis.

Privacy lies at the heart of freedom and it is the right most valued. Privacy is essentially the right to be left alone. A more comprehensive definition of privacy as claim or right is given by A. Westin [22]:

Claim of individuals, groups, or institutions to determine for themselves when, how, and to what extent information about them is communicated to others.

Privacy can be conceived roughly as a cluster of three related but independent components [22]:

- *Secrecy: information about an individual.*
- *Anonymity: attention paid to an individual.*
- *Solitude: access to an individual.*

A breach in any of the above components constitutes an intrusion into one's privacy.

The value of privacy is associated with the following ideas [41]:

- Personal autonomy: this means that individuals are free from interference by authorities and they can act according to their own judgement and not have collective values forced upon them.
- Self-fulfilment: it is argued that privacy provides the best conditions under which individuals may flourish. The dropping of the public mask, the communion of the intimates and the expression of the deepest emotions are crucial to maintain the identity and well-being of each individual.

The means to maintain one's privacy include control over both one's personal life and information. Privacy is also closely implicated in the notions of respect and self-respect, of love, friendship, trust, freedom and protection of personal liberty.

Protection of right to privacy is so fundamental that it can be found in many enacted laws and treaties, such as:

- The Fourth Amendment of the American Constitution.
- The Article 12 of the United Nation's Declaration of Human Rights.
- The Article 8 of the European Convention of Human Rights.

Equally important is the privacy of commercial enterprises. Commercial privacy protects intellectual property, corporate secret and financial transactions and it is one of the cornerstones in capitalistic economy and financial stability.

In this thesis, privacy refers to both personal and commercial privacy without distinguishing them unless expressly stated otherwise. Likewise, "user" refers to individual, group or institution whose privacy is the main concern in this thesis.

2.2 Development of Cryptography

This section briefly reviews the development of cryptography and its importance to electronic communications and most of the materials in this section can be found in [12] and [14]. Before moving on, the following provides a

working definition of cryptography and encryption:

Cryptography is the art or science of keeping a message secure. It can be used to hide its information content, establish authenticity, prevent undetected modification, prevent repudiation, and/or prevent unauthorised use.

Encryption is a branch of cryptography that uses a mathematical algorithm to scramble data in such a manner that renders it unintelligible to anyone who does not possess cryptographic key necessary for it and can be unscrambled only with the knowledge of the key.

2.2.1 Symmetric Cryptography and Its Weaknesses

Cryptography used to be the exclusive domain of government especially in military and diplomatic circles. However, with the advent of modern computers and communications, there has been growing commercial and academic interest in cryptography for civilian applications.

In 1975, U.S. government proposed the Data Encryption Standard (DES) for the protection of sensitive but unclassified information by government agencies. DES is a symmetric cryptosystem in which the same secret key is used to both encrypt and decrypt the message and it is now extensively used by the financial sector to protect financial transactions. The public disclosure of the DES algorithm provided inspiration for much research and new designs. National Security Agency (NSA) in USA has made numerous attempts to either stop or to control the research work by the academic community without success (see chapter 6 in [14]): the genie is out and it cannot be put back in bottle again.

Symmetric cryptosystems suffer from the following drawbacks:

- The secret key needs to be agreed in advance of any communication.
- It is difficult for the sender to distribute the secret key securely to the receiver.
- The security lies completely in the secrecy of the secret key. Compromising the secret key allows anyone to decrypt the message or to tamper with the messages.

- As the number n of participants increases, the number of secret keys required increases in the order of n^2 .
- In the case of dispute, it is difficult to decide which party has cheated because both parties know the secret key.

Public key cryptosystems attempt to address these problems. However, any public key used in public key cryptography must be properly authenticated.

2.2.2 Public key Cryptography

In 1976, Diffie and Hellman introduced the concept of public key which allows parties to exchange encrypted data without communicating a shared secret key in advance. Rather than sharing a single secret key, a public key cryptosystem uses two keys: a public key that can be disclosed to the public and used to encrypt data and a corresponding private key that is kept secret and used to decrypt the data. Currently, RSA is the most famous and widely used public-key cryptosystem.

Public key cryptography offers the following unique features:

- Secure communication is possible without the need to share a secret key in advance.
- The authenticity of a message by a sender can be verified by the receiver.

These properties are vital for establishing large scale information infrastructures (e.g., electronic commerce).

2.2.3 Importance of Cryptography

As the volume of sensitive information flow across insecure networks is growing rapidly, strong cryptography is an essential tool for protecting users' privacy.

The importance of cryptography to users includes:

- Protection of privacy such as personal records or financial data or business transactions from eavesdropping.

- Protection of financial assets from sabotage, fraud, theft or commercial espionage of proprietary data or intellectual property.
- Protection of stored information from unauthorised access or disclosure.

Indeed, cryptography is critical to the development of the Information Superhighway and electronic commerce. Now so much economic activities occur through electronic networks that restriction in deploying strong cryptography is dangerous. However, due to severe government control, the widespread use of cryptography has not materialised.

2.3 Potential Threats of Cryptography

This section investigates some threats of cryptography to society and users. In an extreme situation, abuses of cryptography may create chaos and social disorder known as “Crypto Anarchy”.

2.3.1 Threats

Cryptography presents a conflict between privacy and law enforcement and intelligence gathering. Strong cryptography can virtually lock out all stored files and communications and thwart investigation by making legally intercepted messages unreadable. Thus, criminals can be immune from lawful interception so that they can act and conspire with impunity. The use of cryptography seriously hampers this important law enforcement tool and poses a serious social threat to public safety and national security. The potential threats of cryptography to society include:

- It can be used by terrorist groups to further terrorism (e.g., a plot to bomb a building or to assassinate a political figure).
- It can be used by criminal rings to further organized crime (e.g., extortion, kidnapping, drug trafficking, child pornography, murder or money laundering).
- It facilitates fraud and encourages corruption without fear of being discovered.
- It can interfere with foreign intelligence operations.

Cryptography can also be a threat to users themselves. Potential threats to users themselves include:

- Lost keys by accident (e.g., by carelessness or sudden death of key-holder).
- Lost keys by intent (e.g., by disgruntled former employee or the key is held in ransom for blackmail).
- Keys may be used to cover up fraud or espionage (e.g., by employee).

Loss of key means that valuable data becomes inaccessible. Huge financial loss may result if the lost key cannot be recovered and the encrypted information will be lost forever.

2.3.2 Crypto Anarchy

Cryptography allows absolute freedom of communications unprecedented before in history. If cryptography is not properly harnessed, the widespread deployment of cryptography may lead to social disorder and chaos. A horrible vision of crypto anarchy in abusing cryptography is depicted in [29].

In crypto anarchy, it is no longer possible for governments to control information (e.g., to compile dossiers) and to regulate economic activities (e.g., collection of tax). Governments crumbled and disappeared. Instead, they are replaced by virtual communities of individuals doing as they wish without interference.

Crypto anarchy becomes the safe havens for many criminal activities such as tax evasion, money laundering, espionage, contract killing and data havens for storing and marketing illegal material. This brings the civilization back to a new dark age of barbarism history never witnessed before. In Dorothy E. Denning's own word,

It is like an automobile with no brakes, no seat belts, no pollution controls, no licence, no way of getting in after you've locked your keys in the car [12].

Is this desirable? Is this inevitable?

To prevent this phenomenon from happening, the new cryptographic right must be properly matched by the corresponding new responsibility.

2.4 Government Policy

This section investigates the fundamental change in government policy in cryptography from strict control of use to strict access to decryption key.

Encryption technology used to be tightly controlled by governments to protect public and national security. Control included export control, limiting the length of the encryption key or licensing the use. However, stringent control created numerous problems:

- Strict control weakens the protection of user's privacy.
- Strict control harms export competitiveness because the affected industry is at a disadvantage to compete globally.
- Industry needs two sets of products for domestic and international applications and this creates extra cost and interoperability problem.
- Strong encryption algorithms and softwares (e.g., PGP) are readily available in many computer sites. Strict control can only hurt most legitimate users but certainly cannot prevent criminals from using them.

Recently, there is a fundamental change in governments' policy: from strict control of usage in cryptography to mandatory escrowing of decryption key.

To relax the use of strong encryption with long keys, an emergency decryption capability by government must be provided. The most straight forward solution is by key archive through a trustee or key escrow authority appointed or licensed by the governments.

An encrypted message normally contains a header with the session key encrypted under the public key of the recipient. Access is then possible using the escrowed private key to recover the session key and then to decrypt the encrypted data with the session key.

The key archive services provided by key escrow authorities will be available to the government agencies upon court order and to the owners of the keys (called self-escrow).

Although criminals might obtain encryption products from some underground servers and bulletin boards or they can develop their own non-compliant products, they have to face interoperability problem with licensed ones, so their use will be limited. At minimum, the key escrow system deters

criminals from the embarrassing situation of using the convenience of the legitimate key management infrastructure for illegal purposes.

2.5 Information Superhighway

The Information Superhighway is in the making. This is possible due to the availability of global communication networks and strong cryptography. This section examines the factors and ingredients contributing to the success of the Information Superhighway.

2.5.1 Cryptographic Services

In order to effectively use the Information Superhighway, it is crucial that a public key system is available and a user has a reliable way of verifying the authenticity of public keys.

In general, the following cryptographic services are required:

- Authentication of users.
- Issuing and distribution of signed certificates for public keys.
- Revocation and expiration of public keys.
- Time-stamping and notarization of electronic documents.
- Resolution of disputes.
- Private key escrow management and lawful access to private keys.

2.5.2 Key Management Infrastructures

Management and certification of public keys can be performed by a certification authority while maintaining private key archives is done by a key escrow authority (or a trustee) though they can be combined together.

The certification authority establishes proof of identity of the person owning the corresponding private key and then signs a certificate containing the user's public key. A certificate for a public encryption key will not be issued unless the corresponding private decryption key is archived by the key escrow authority. The certificate can be used to verify user's public key over a network. Separate keys can be used for encryption (for confidentiality) and

digital signature (for authentication and nonrepudiation) and separate certificates will be issued for each key. With the help of a certification authority, confidentiality, integrity, authenticity and nonrepudiation cryptographic services can be provided. A hierarchy or network of certificate authorities can be established to form an infrastructure for managing and certifying public keys.

Such a key management infrastructure can be extended to form a national and global Information Infrastructure (NII and GII). With a key escrow facility, the Information Superhighway will not become a safe haven for illegal activities. A global key management infrastructure with key escrow facility also requires international agreements. Thus, government initiative and involvement is critical to make NII and GII a reality.

Users can then access strong and globally interoperable encryption systems and key management services.

As certification authorities and key escrow authorities play crucial roles in NII and GII, strict standards must be developed by legislation to ensure key integrity, confidentiality, accessibility, auditability, recovery and use. In particular, key escrow authorities must ensure the following:

- Authorised access is possible in a timely fashion.
- Proper access procedures and legal operational safeguards are strictly followed.
- Proper audit records of key handling events are kept.
- A high level of assurance that there is no unlawful compromise or abuse of escrowed keys.

Some early suggestions of key escrow authorities included post office, bank or clerks of the Federal Courts [7]. In general, they should be certified or accredited according to their qualification, responsibility, liability and be in the private sector rather than the government agencies.

From the security point of view, it is desirable to split the keys into a number of shares and each share can be escrowed by a separate key escrow authority so that the keys cannot be misused without collusion.

Chapter 3

Challenge

This chapter briefly highlights some problems and potential dangers with existing key escrow schemes and summarises the objectives of this thesis.

3.1 Problems with Existing Key Escrow Schemes

Existing large-scale key escrow schemes have numerous conceptual and technical problems. Typical problems are:

- It is mandatory for users to hand over their private keys to a key escrow authority as a prerequisite to certification of their public keys by a certification authority. This is analogous to assuming everyone is guilty until proven innocent which nurtures fear in being wire-tapped at any moment and this may create a chilling effect in communication [25].
- Government may suddenly become malicious: for example in the case that a dictator gains power, by changing the escrow policy overnight, the government can then start massive recovery of the escrowed keys and embark upon mass wire-tapping.
- Trustees are the weak link of the system: they can be easily attacked because they are well-known and their number is small.
- If the key escrow database is compromised, the security is lost. Every user of that key escrow authority can then be eavesdropped electronically.

- Users do not have confidence in these schemes because they are in favour of law enforcement agencies at the price of sacrificing users' right of privacy.

Hence, under existing schemes, users are completely unprotected from the abuse of power by authorities or compromise of the security in the system yet they have no other options but to faithfully and unconditionally trust that the system will work properly for them.

3.2 Orwellian World

The government is powerful while the individual is weak. In its attempt to ensure the safety of its citizens, the government can easily overstep the privacy of the individuals. Increasingly more activities (political, social, cultural, financial) are occurring electronically, too much government control can result serious erosion of the rights of privacy.

The following is an abstract taken from George Orwell's 1984 which depicted the horrible scene of complete intrusion of individual's privacy by the government:

Outside, even through the shut window-pane, the world looked cold. Down in the street little eddies of wind were whirling dust and torn paper into spirals, and though the sun was shining and the sky a harsh blue, there seemed to be no colour in anything, except the posters that were plastered everywhere. The black-moustachio'd face gazed down from every commanding corner. There was one on the house-front immediately opposite. BIG BROTHER IS WATCHING YOU, the caption said, while the dark eyes looked deep into Winston's own. Down at street level another poster, torn at one corner, flapped fitfully in the wind, alternately covering and uncovering the single word INGSIC. In the far distance a helicopter skimmed down between the roofs, hovered for an instant like a bluebottle, and darted away again with a curving flight. It was the police patrol, snooping into people's windows. The patrols did not matter, however. Only the Thought Police mattered. Behind Winston's back the voice from the telescreen was still babbling away about pig-iron and the overfulfilment of the Ninth Three-Year Plan. The telescreen received and transmitted simultaneously. Any sound that Winston made, above the level of a very low whisper, would be picked up by it; moreover, so

long as he remained within the field of vision which the metal plaque commanded, he could be seen as well as heard. There was of course no way of knowing whether you were being watched at any given moment. How often, or on what system, the Thought Police plugged in on any individual wire was guesswork. It was even conceivable that they watched everybody all the time. But at any rate they could plug in your wire whenever they wanted to. You had to live – did live, from habit that became instinct – in the assumption that every sound you made was overheard, and, except in darkness, every movement scrutinized [34].

This vision of a Big Brother government should serve as a warning to the design and choice of any key escrow schemes.

3.3 Dilemma

Strong encryption enables absolute privacy of communication with the potential of denying legitimate government access. This is unprecedented and not available in other areas of civilized lives, for example, one's speech is subject to slander, libel, obscenity and other legal restraints. Human society is based on an implicit social contract: individuals sacrifice absolute freedom in exchange for an orderly society and limited freedom. One must protect privacy as well as public safety and national security. Cryptography has upset the traditional balance of power between individuals and government. A new balance must be sought to restore the status quo.

This involves balancing competing interests in a way that ensures effective law enforcement and intelligence gathering, while protecting users' privacy.

To sum up this dilemma, Dorothy E. Denning wrote a letter [13] and it is reproduced here as follows:

My position on encryption policy, April 2, 1997.

I do not know what is the best long-term U.S. or international policy regarding encryption. I recognize the need for encryption to protect information, but I also appreciate the adverse effects encryption can have on public safety, law enforcement, and national security.

I do not advocate domestic regulations mandating key recovery. Neither do I advocate that cryptography necessarily be free of all regulation, including export controls. I constantly struggle with the issues and do not see easy

answers. I believe that we should seriously discuss and evaluate a wide range of options, and that our decisions should be as informed as possible. There may not be a single approach that is best in all contexts.

I believe that organizations and individuals should be able to use strong, robust encryption. I also believe that key recovery is good business policy.

I support the program of the Clinton Administration to liberalize export controls for key recovery products, to adopt key recovery within federal agencies, and to promote key recovery technologies. I believe that key recovery can be done without compromising privacy and security.

I support open, public discussions of encryption policy. I also recognize that pertinent information relating to national security, law enforcement, and proprietary interests will not be made public. Encryption policy must be based on all available information.

Dorothy E. Denning

This dilemma of electronic privacy versus lawful access by government may in extreme cases create crypto anarchy or the Big Brother world. The paramount question in this thesis is: is there a middle ground or compromise between users and authorities that is mutually acceptable to and trusted by both parties?

3.4 This Work

This thesis proposes an optimal solution to this intractable dilemma based on the jury system and presents an improved key escrow scheme that is acceptable to all parties. This scheme has the following properties:

- The government cannot embark upon mass wire-tapping.
- The number of trustees is so large that it is practically infeasible to manipulate them.
- Even if the key escrow database is compromised, the escrowed keys are still safe.
- Users' private keys are not handed over to the authorities prematurely until they are actually alleged of committing a crime.

- This scheme provides a fair balance of power between users and authorities. It balances the user's right of privacy without sacrificing the need for law enforcement.

These properties are achieved by retaining the bulk of the escrowed keys in the hands of the users themselves (conceptually) until the moment they are required for law enforcement purposes.

Chapter 4

Related Works

This chapter briefly reviews some of the related works. A taxonomy for key escrow systems can also be found in [10]. Most of these schemes suffer from the drawbacks as pointed out in section 3.1.

4.1 Escrowed Encryption Standard (EES)

After over twenty years of service, DES was believed to be close to the end of its useful life. A stronger algorithm with longer key is needed to replace DES. On April 16, 1993, the U.S. Government announced [33] a proposal for a new federal standard encryption system with key escrow capability which is called Escrowed Encryption Standard (EES) [23].

4.1.1 Brief Description of EES

EES is a voluntary Federal standard for encryption of voice, fax and computer information transmitted over circuit-switched telephone systems. It is based on a tamperproof chip (also known as Clipper chip) which implements the classified Skipjack encryption algorithm designed by the National Security Agency (NSA). Skipjack is a symmetric block cipher with a key length of 80 bits which is much stronger than DES with 56 bits. Both the plaintext and ciphertext lengths are 64 bits.

In EES, a user's Clipper chip key is escrowed with two government escrow agents (namely, the National Institute of Standards and Technology and Automated Systems Division of the Treasury Department) and hence allows key recovery by law enforcement authorities when served with a lawful warrant. To identify the sender and receiver, a field, called Law Enforcement Agency

Field (LEAF) is attached to the ciphertext. An excellent account of EES can be found in [11].

4.1.2 Operation

For two parties to communicate using EES, both parties must have a communication security device with a Clipper chip. Every chip will have its chip identifier and key registered with the Federal government. Key registration will occur during manufacturing process and NSA licences the manufacturers of the chip. The protocol is as follows:

- The devices agree on an 80-bit session key separately (not included in EES).
- Each device passes the session key to its Clipper chip.
- Clipper chip encrypts the session key with its chip key.
- The encrypted session key and the chip's 32-bit identifier together with a 16-bit checksum are encrypted again by the family key known to the government only and this forms the LEAF that is transmitted to the other device.
- If the LEAF is valid, encrypted communications can begin.

Recovery by law enforcement agencies such as Federal Bureau of Investigation (FBI) is as follows:

- FBI intercepts the encrypted message.
- FBI extracts the LEAF from the header of the message.
- LEAF is decrypted by using the family key and the chip's identifier is determined.
- Escrow agents return the two halves of the corresponding chip's key.
- The session key can be recovered by decrypting again with this chip key.
- The plaintext can be recovered by decrypting with this session key.

Note that, once the chip key is obtained, all encrypted calls made using this chip can be similarly decrypted without time-bound.

4.1.3 Criticism

Public response to EES has been overwhelmingly negative and it has sparked off a lot of criticisms. Criticisms include:

- Skipjack algorithm is classified and not open to public scrutiny.
- Because Skipjack is designed by NSA (a military organization), there is a suspicion that a backdoor might have been installed.
- EES requires inflexible and special tamper-proof hardware.
- User's secret keys are escrowed by government agencies and this allows ease of collusion and unlawful wiretapping of private communications of the users.
- There is fear that there will be a total ban by government in unescrowed encryption systems.
- There is no self-escrow facility and hence it offers no benefit to the users because users of lost key cannot use EES to recover their lost keys.
- Although the exact method of checksum is classified, the 16 bits checksum is too small to be secure against exhaustive search.
- Once an investigation is authorized by the court, the authority will be able to decrypt every message transmitted by the suspect, without any time-bound.
- Only information about a message sender is included in the LEAF. One cannot trace the destination of a message.
- It requires a limited number of licensed EES manufacturers to produce the Clipper chips and security is a problem.
- EES adds complexity and cost to the communication system.
- It is unlikely that foreign countries will adopt Clipper chip for secure communications.
- Interoperability problem between two different systems requires dual products to support domestic and overseas applications.

One security attack comes from the work of Blaze [5]. He pointed out that it is possible for rogue applications to use the Skipjack algorithm integrated inside a Clipper chip to do encryption/decryption without using the LEAF. This technique replaces the LEAF containing the current session key by one containing an unrelated key that allows one participant in a communication to construct unilaterally a LEAF that denies law enforcement, but which will be accepted as "valid" by a communicant using EES-compliant technology. Some more technical criticisms can also be found in [24].

4.1.4 Clipper II & III

In the Clipper chip II scheme, the U.S. government recognized the importance of independent trustees and agreed to use trusted third parties (TTP) outside the government to escrow users' private keys but still refused to unclassify the Skipjack algorithm. However, Clipper II was soon replaced by a more open and comprehensive proposal [32] in May 1996 which was soon dubbed as "Clipper III".

4.2 Fair Public Key Cryptosystems

This is an important scheme which was proposed by Micali in 1992 [28] and was partially adopted by Clipper III. In this scheme, there are a fixed number of predesignated trustees and there exists an arbitrary number of users. A user chooses his/her public and private key pair and then splits the private key into pieces and gives each trustee one piece. Each trustee can individually verify that the piece he/she received is indeed part of the private key without combining the pieces. Unlike EES, there is no need for a tamperproof chip and a classified algorithm. The scheme is based on public key cryptosystems and can be implemented in software alone.

The contribution of this paper includes:

- It introduces a software-based public-key key escrow scheme.
- This scheme is fairer to the users because it strikes a better balance of power between users and authorities in comparison with EES.
- Different schemes for both RSA and Diffie-Hellman based systems are proposed.

- It incorporates verifiable secret sharing protocol so that the user can choose his/her key but he/she cannot cheat the trustees.
- It discusses algorithmic-chosen session key generation.
- It discusses time-bounded court-authorized eavesdropping.

However, this scheme suffers from the drawbacks mentioned in section 3.1. Later on, this thesis will show how to make this scheme democratic.

4.3 Encapsulated Key Escrow

This scheme [1] improves [28] and makes it possible for authorities to only selectively wiretap a small number of individual users but makes it computationally prohibitive to launch large scale wiretapping. This is achieved by imposing a computational time delay (or time capsule) between obtaining of the escrowed information of a user and recovering the user's private key. The time capsule can be set arbitrarily depending on the computational strength of the authorities.

In this scheme, it is true that massive recovery of private keys by the government is computationally hard, but some problems can still be noticed:

- There is no concrete construction of the time capsule function is given and it is difficult to assess the strength of such a function.
- Even the time capsule can be found, it is difficult to set the time capsule correctly because the government's computational strength is not static but highly dynamic (e.g., due to technological breakthrough). This may create worries and frequent re-escrowing by users.
- If the time capsule is set too high, it may create too much delay to the law enforcement agencies to render it useless after the key is recovered.

4.4 Verifiable Partial Key Escrow

In this scheme [2] only part of the private key of a user is escrowed in a verifiable manner with the trustees of the government and it does not suffer from early recovery attacks. Even if the government obtains legal authorization for wiretapping the user and gets hold of the part of the private key which

was escrowed, it will still need to compute for a non-negligible but feasible time, before it can recover the entire private key. This prevents the government from embarking upon large scale wiretapping. This scheme is similar to [1] and shares similar problems with the other scheme.

4.5 A Proposed Architecture for Trusted Third Party Services

This scheme [31] is adopted by the UK government for secure email and it is based on Diffie-Hellman's key distribution protocol. It introduces a licensed trusted third party (TTP) to escrow users' private keys.

In this scheme, consider a pair of users A and B where A wishes to send B a confidential message and needs to be provided with a session key to protect it. Suppose that A and B have associated different TTPs TA and TB respectively. TA and TB need to agree on a number of parameters during the setup phase.

Prior to any communication, A needs to generate the private send key and obtain the public receive key for B and B needs to have the private receive key. These keys are unique for each pair of participants A and B.

There are numerous problems with this scheme [30]. One serious problem (Problem 6 on p.9) is that the compromise of the interoperability key between the two domains will be catastrophic, as all traffic between users in those domains will be vulnerable to attack.

4.6 Oblivious Key Escrow

In this scheme [6], the Internet as a whole acts as an escrow agent. This is made possible by splitting the key into a very large number of shares (e.g., 5000). These shares are then escrowed randomly by different sites independently in such a manner that no one knows which sites are escrowing whose keys. The trustees are very decentralised and their number is huge. This scheme gets rid of many concerns associated with conventional schemes but there are some problems :

- It is extremely inefficient and requires a lot of memory for a large-scale system because each key has to be split into a large number of shares.

- It is very chaotic and unreliable (e.g., the identities of shareholders are not known).
- It is difficult for lawful recovery of the escrowed key (the author suggested pleading the case in television in order to convince these sites to return their shares).
- It lacks structure and details for a serious implementation.
- It cannot prevent a malicious user from cheating by escrowing some random bits rather than his/her true private key.

4.7 Translucent Cryptography

This scheme [3] offers an alternative to key escrow and does not require key escrow trustees at all. It was proposed that a probability can be used as a parameter such that a particular intercepted message can be decrypted by law enforcement agencies with this probability. However, it has two serious disadvantages:

- Law enforcement may be frustrated because when it has an authorised wiretap, it is not getting decryption of all the messages.
- Individuals may be frustrated that this scheme does not provide absolute privacy for their messages; law enforcement can read some fraction of their messages.

Hence, this scheme should not be used for sending sensitive information.

4.8 Binding Cryptography

This scheme [20] is another alternative to key-escrow schemes and it allows data recovery by law enforcement agencies as before. Data recovery is possible without the need to recover the private key. It avoids the need to escrow the private key by the concept of “binding data”. In this scheme, users’ messages should consist of the following fields:

- The intended message encrypted with a symmetric system by using a session key.

- The session key encrypted by the public key of the receiver so that it can be recovered by the receiver only.
- The session key encrypted by the public key of a trustee so that it can be recovered by the trustee only.

It is essential to prove that the two encrypted fields correspond to the same session key without disclosing it. An efficient implementation based on ElGamal encryption scheme [19] is also proposed. In case of lawful data recovery, the trustee only needs to decrypt the session key with his/her private key. In this manner, this scheme gets rid of the problem of escrowing user's private key altogether. However, it shares the same problem with [31]: the potential risk of compromising the trustee's key. In addition, the last field is redundant and it is not necessary for secure communication between two parties. Hence, it is difficult to enforce illegal users from omitting it unless all communication providers cooperate to prevent this from happening.

4.9 A Matter of Trust

The ultimate problem with existing key escrow schemes is a matter of trust. Without user's trust, key escrow systems cannot become widespread as the governments would like to see. The failure of Clipper chip initiative clearly supports this point. Realising this problem of trust, governments now agree to use TTP as escrow agents. While this is a drastic improvement in boosting confidence among users, the revised schemes are still insecure, unfair and undemocratic. The old problem remains: whom can the users trust? Can they trust a handful of trustees appointed and regulated by the governments?

Perhaps, the real solution does not lie in inventing new cryptographic technology alone but also in finding the new delicate balance point between users' and authorities' rights which must be mutually trusted by both parties. However, a measure of trust is something which is difficult to establish or to maintain.

The jury system in common law has a high reputation of being impartial, independent and representative. It has been implicitly trusted by individuals and authorities alike as a tribunal for settling disputes over several hundreds of years. The implicit trust model in the jury system may have modern relevance to key escrow scheme today.

Chapter 5

Jury System

The jury system provides impartial administration of justice to protect individuals' civil liberty. It allows lay participation by individuals and this can be used to express the community conscience through its members of the jury. The concept of lay participation and community conscience is the most crucial part of the jury system. This chapter briefly summarises the development and features of the jury system and its similarities with the counterparts in continental law. Most of the materials in this chapter can be found in [9] and [18].

5.1 Introduction

There are broadly two kinds of justice systems in criminal trials in the world : common law system (also known as adversary system) and continental system (also known as inquisitorial system). The former one is mainly practised by English speaking countries such as Britain and USA while the latter is typical of European countries such as France and Germany. The adversary system relies mainly on oral argument and presentation while the inquisitorial system concentrates more on written evidence. In both trial systems, a suspect is assumed innocent until proven guilty.

Crime in the common law system can be classified into two types - summary offences and indictable offences. The first category of offences are mainly minor offences and not criminal in the full sense (e.g., driving offences which can be mostly settled by fines) so that it is not worthwhile to undergo prolonged trial by jury. In case of indictable offences (e.g., murder cases which normally impose severe imprisonment sentences), the accused has a right to trial by jury. Trial by jury is of utmost importance in criminal

trial of common law system. It is conducted in a fashion similar to a contest between prosecution and defence lawyers and the prosecution must prove to the jury's satisfaction that the suspect is guilty beyond reasonable doubt. The role of the judge is to act as an umpire in order to make sure both sides act properly according to the prescribed rules.

The origins of trial by jury is deeply rooted in the political struggles of medieval England. It is not what some lawgiver so decreed but evolved gradually over several hundreds years. By this system, twelve men and women are selected at random from a large population of jurors chosen from the common folks to form a jury in a criminal trial; they have never before had any judicial experience. Their role is to judge the facts presented to them. At the end of the case they will deliberate in order to arrive at a verdict of whether the accused is guilty or not. The jury are free to reach whatever verdict seems just, taking the case as a whole. It is not required that the jurors have to understand all directions given by the judge; they deliberate secretly; they need not give explanations for their verdict. They are therefore free from strict application of the law to the facts and this gives them some flexibility in applying the law.

5.2 Origin

Originally a juror was just a man who was compelled by the King to take an oath in telling the truth. The Normans brought over this idea of jury to Britain during the Norman Conquest. Subsequently, the King used it for obtaining information which he wanted to know about matters of local administration. People living in the place where an inquiry was being held were compelled to answer because they must know the local facts. Thus the jury originated as a body of men used in an inquest decreed by the King. At the beginning, the inquest was not related to the administration of justice. Disputes were settled simply by one of the disputants proving himself by some means to be the better man, e.g., trial by battle. However, trial by ordeal was the most popular way to settle a dispute because people at that time believed that it represented the divine acceptance of a claim by the claimant.

5.3 Early History

A jury which gave the King information for administrative purposes was later used by King Henry II to give information which would enable him to decide a dispute. When a party got twelve oaths from the jury in his favour, he won. This is the origin of the trial jury, of the rule that the trial jury consists of twelve and of the rule that the verdict must be unanimous. Twelve was chosen because it was thought to be large enough to create favourable opinion of the side that won.

In November 1215 Pope Innocent III prohibited trial by ordeal which was practised in criminal trials for more than fifty years because it was thought to be too cruel. As a result, new ways must be invented and it was at this moment that judges started to use the jury for criminal trials. Gradually, the role of jury changed from providing local knowledge in an inquest to the sole adjudicators of facts in modern jury system. This was due to the fact that courts in feudal society required local knowledge of jurors for the settlement of disputes. Due urbanization, this was no longer possible. Jurors, instead of deciding the case on what they themselves knew, relied on the submitted evidence provided by both parties to reach the verdict.

Bushells Case in 1670 was a landmark in the history of jury system because it ended the judge's power to force juries to convict a suspect, by threatening to punish them if they did not comply. In that case, the Quakes, Penn and Mead, were set free by the jury of participating in an unlawful assembly and the trial judge subsequently put the jury in jail. However, the whole body of judges gathered together and concluded that as a matter of law the trial judge did not have any power to do this. The case is often hailed as a triumph for the independence of juries.

5.4 Characteristics

The following are the main characteristics of the jury system:

- The composition of jurors should be representative of the community at large.
- The juror represents the reasonable man or the man in the street.
- The juror should be a freeman. Most civil servants are exempt from jury service.

- The jurors are not responsible for anything said or done in carrying out of their duty.
- The jury is the sole judge of facts.
- Twelve jurors are randomly selected from the juror population to form a jury in a criminal trial.
- The jury has only to return yes or no without any need to give any reasons.
- In most cases, ten-to-two majority verdict is sufficient. In rare cases, unanimity is required.

5.5 Selection Procedure

The selection of a jury is done in two stages. The first stage is the selection of a small subset of jurors from the juror population by the sheriff. The second stage is the selection of twelve jurors from the subset to form the jury and that is done by the clerk of the court randomly. A juror can be challenged either peremptorily or for cause. The peremptory challenge is the one for which no reason need to be assigned and the number is now limited to seven. The party challenging for cause must show grounds to support the challenge. The challenged juror cannot serve in that trial and must be replaced by other jurors.

5.6 Merits

The Jury system presents an image of independence, impartiality and representativeness. The jurors are selected randomly from citizens who have no conflict of interest in the case, their judgement is unbiased by the legal and administrative bureaucracy (e.g., courts and police). Thus, it convinces members of society to accept the impartial nature of the judicial institutions.

In summary, the merits are :

- The jurors prevent collusion between judges and the police, and prevent police influence in the courts from becoming dominant.

- The jurors prevent private citizens from exerting improper influence over judges (e.g., by bribery).
- The jurors prevent the state from manipulating justice to curb its political opponents.
- The jurors prevent unjustifiable prosecutions and safeguard against repugnant laws which are harsh and oppressive.
- The jurors act as adjudicators whose view and experience will be those of the man-in-the-street.
- The jurors are free from prejudice of professional judges and prevent judges from imposing the views from their social class.
- The jurors ensure the independence and quality of the judges.
- The jury system educates the jurors with a sense of fairness and propriety of the judicial processes.

In 1768, William Blackstone enshrined the jury as the palladium, the bulwark of liberty [4]. In USA and Canada, the jury system is generally acclaimed as a fundamental guarantee of individual liberty. Trial by jury is therefore enshrined as a constitutional right (the Six Amendment of U.S. Constitution and the Canadian Bill of Rights).

5.7 Trials in Continental Law

Many countries practising continental law employ a system in which judges and laymen sit together as a combined bench to decide a criminal case. Below are just a few examples:

- France : seven laymen (also called jurors) are required to sit together with three judges as one bench to decide questions of guilt and punishment jointly.
- Germany : a court of six lay members and three judges is established to try the most serious criminal offences, and for the medium-range offences a smaller version comprising one judge and two laymen is created.

- Sweden : in serious crime, a professional judge and from seven to nine lay members known as "namndeman" form a panel. These are drawn from a large body of local citizens chosen for their position of responsibility and their record of service in the community. The judge acts as chairman and his/her vote is given special weight. A case can only go against his/her opinion if at least seven of the lay members decide it the other way. For the middle range of crimes, a judge and three namndeman is used.

Here the judge has greater influence over the lay members by taking part in the joint deliberations (in common law, jury deliberates alone). However, the judge has to justify his/her opinion to the laymen, for they in the end have the power to outvote him. Thus, the presence of lay judges, being totally independent, offers advantages similar to the common law's jury system.

5.8 Bulwark of Civil Liberty

The jury system is often hailed as the bulwark of civil liberty. Below are just a few famous quotations from legal sources to support this view:

- *no free man shall be taken and imprisoned or disseised of any free tenement or of his liberties or free customs or outlawed or exiled, or in any other way destroyed, nor will we go upon nor send upon him, except by the lawful judgment of his peers or by the law of the land [8].*
- *The first object of any tyrant in Whithall would be to make Parliament utterly subservient to his will; and the next to overthrow or diminish trial by jury, for no tyrant could afford to leave a subject's freedom in the hands of twelve of his country-men. So that trial by jury is more than an instrument of justice and more than one wheel of the constitution: it is the lamp that shows that freedom lives [15].*
- *What makes juries worthwhile is that they see things differently from the judges, that they can water the law, and that function which they filled two centuries ago as a corrective to the corruption and partiality of the judges requires essentially the same qualities as the function they perform today as an organ of the disestablishment [17].*

- *In a democracy law is made by the will of the people and obedience is given to it not primarily out of fear but from goodwill. But just as important as the frame of the law is its application. The jury is the means by which the people play a direct part in the application of law. It is a contributory part. The interrelation between judge and jury, slowly and carefully worried out over several hundred years, secures that the verdict will not be demagogic; it will not be simply uninhibited popular reaction. But it also secures that the law will not be applied in a way that affronts the conscience of the common man. Constitutionally it is an invaluable achievement that popular consent should be at the root not only of the making but also the application of the law. It is one of the significant causes of our political stability [16].*

Chapter 6

Democratic Key Escrow Schemes

In the following, this thesis proposes a new approach to key escrow by the introduction of “electronic jury system”.

6.1 Electronic Jury System

The jury system represents the middle ground or compromise in the traditional balance of power between individuals and the government and this trusted compromise is exactly what is lacking in the existing key escrow systems. The implicit trust model can be borrowed and applied in this new situation of key escrow system.

6.1.1 New Mechanism

The proposed electronic jury system is a new mechanism to achieve proper balance of power in the new era of information revolution by distributing the escrowed key shares to users who are members of the electronic jury rather than the authorities. The selection of electronic jurors and electronic jury must be fair and seen to be fair. Each escrowed key is kept collectively by the electronic jury using a secret sharing scheme [38]. The specific combination of the electronic jury should be kept secret and known to the authorities alone. In the case of an alleged crime, the authorities will request the electronic jury to return their shares of the suspect’s private key in order to recover his/her private key. Optionally, it may be required that the authorities have to submit the case to the electronic jury and to convince them for the return

of their shares. Each electronic juror may decide individually or together. If the number of shares exceed the threshold, the authorities can recover the private key and decrypt the necessary information.

One distinct advantage of this scheme is that it allows a better international agreement in establishing a global key escrow infrastructure because there is a common thread in using laymen in the judicial process in both common law and continental law.

In the remaining parts of the thesis, jury/juror are not distinguished from electronic jury/juror. This can be easily understood from the context.

6.1.2 Abstract Model for Electronic Jury System

Based on the traditional jury system, the following abstract model can be established:

- The jurors must be chosen fairly from the users to form the juror population.
- The juror population must be large enough to prevent possible manipulation of jurors.
- A jury consisting of a small number of jurors (say seven) must be randomly selected from the juror population by escrow authority.
- Each user's private key is then shared among the jurors in the jury.
- Each user should not know his/her jury to prevent collusion.
- Each juror must not know the user for the same reason.
- The actual secret key shares must not be known to escrow authority to prevent compromise or abuse by authorities.
- The list of jury population must be public to foster openness and trust.
- The key escrow information (the mapping of users to juries) is known to the escrow authority only in order to allow lawful recovery of private keys.

Optionally, jury can be used to evaluate police's request (e.g., by requiring a brief summary of the case) and vote by returning/not returning individual shares.

Further benefits are the same as traditional jury system mentioned in Section 10.6.

6.2 System Description

A key escrow scheme is *democratic* if it employs the above electronic jury system as key escrow trustees. It is democratic because the traditional jury system is one of the cornerstones of modern democracy.

This section shows how to design democratic key escrow schemes. Three different schemes are then proposed in the following three sections. Each scheme prevents a user from cheating: that is escrowing some random bits rather than his/her true private key. The first one attempts to solve this problem by using a tamperproof smart card while the second and third ones do the same but eliminate the need for special hardware and can be fully implemented in software.

6.2.1 Entities

The system consists of the following entities:

- User: the user wants to (or is required to) escrow his/her private key.
- Escrow authority (EA): EA stores the information about each user's private key but does not actually store the private key. Instead, EA only needs to know the jury who actually hold the shares of a given user.
- Certification Authority (CA): CA signs user's public key and binds user's identity to his/her public key. To obtain such a certificate, the user needs to escrow his/her private key with EA beforehand.
- Legal authority (LA): LA is responsible for registration, selection and maintenance of juror population. LA is also responsible for recovering user's private key.
- Law enforcement authority (LEA): LEA wants to wire-tap a suspect's encrypted message. LEA will need to seek help from LA.
- Juror: a juror is selected from the pool of users themselves according to some known criteria. Each juror independently escrows user's shares

of his/her private key. The juror population is publicly known and the population size is large.

- Jury: a jury is a body of jurors (say 7 members) randomly selected from the juror population. Each jury is responsible for escrowing one private key according to a secret sharing scheme.

6.2.2 Requirements

A democratic scheme must have the following properties:

1. It must allow the user to securely choose his/her private key.
2. EA must be convinced that the escrowed private key is correct and not just some random bits; yet, EA cannot learn anything about the private key. EA also has to ensure that the escrowed private key correctly matches user's public key.
3. User must be prevented from learning who his/her jurors are and the jurors must not learn who the user is.

The first property prevents the authorities from selecting a weak key for the user.

The second property requires either employing a tamperproof smart card, a verifiable secret sharing scheme or a “cut-and-choose” protocol to be discussed below.

The third property can be achieved by dis-allowing direct communications between the user and the jury so that all communications are routed through EA. However, EA will then know the private key too. One solution is to allow each juror to send an encryption key to the user through EA so that the user can encrypt his/her share with this key for this juror. This share is then forwarded to the juror through EA and the juror can then decrypt the message with his/her secret decryption key to recover the share. By this way, the user can send the shares to the jurors securely without EA learning the escrowed key. However, EA can cheat by secretly creating a pair of keys itself. EA then gives one of them to the user and pretends that the key comes from the juror. User will then encrypt juror's share using this key. This enables EA to recover user's secret key easily after obtaining enough shares. This can be prevented by allowing the user to *challenge* the jury. A

challenged jury will be *opened* by EA and become known to the user and the user can then check with the jury whether EA has cheated by verifying the encryption key. The challenged jury will then be replaced by a new one by repeating the key registration scheme again. More information about the security of this mechanism can be found in 4 of section 7.2.

It is assumed that all communication channels are secure. This may be easily achieved by sharing a secret key in advance by the participants who need to communicate directly.

6.3 Scheme 1

This scheme employs a tamperproof smart card for secure key generation and storage to prevent the user from cheating. It presumes the availability of such a trusted hardware device with some cryptographic capabilities.

6.3.1 Key Registration

Let M be the number of jurors in each jury per user.

1. User requests key escrow service from EA.
2. While User is not satisfied with the jurors selected by EA, repeat the following steps:
3. EA randomly selects M jurors from the juror population provided by LA to form a jury.
4. EA calls the M selected jurors.
5. Each juror i , $i = 1, \dots, M$, does the following:
 - Generates two primes p_i, q_i and computes a modulus $n_i = p_i * q_i$.
 - Chooses $K_{1,i}, K_{2,i}$ such that:

$$K_{1,i} * K_{2,i} = 1 \text{ mod } (p_i - 1)(q_i - 1).$$
 - Sends $K_{1,i}, n_i$ to EA but keeps p_i, q_i secret.
6. EA sends all $K_{1,1}, \dots, K_{1,M}, n_1, \dots, n_M$ to User.
7. User challenges the jury or exits the while loop from here if he/she accepts this jury selected by EA to escrow his/her shares.

8. EA opens the jury to User and public.
9. User checks that $K_{1,i}$ is indeed sent by juror i .
10. End of while loop
11. User's smart card generates one key pair (pk, sk) for RSA.
12. EA randomly generates a mask m which is a random number used to hide the private key of the user.
13. EA does the following:
 - Notifies the user to insert the smart card.
 - Authenticates user's smart card and verifies the key pair is correct.
 - Sends all $K_{1,1}, \dots, K_{1,M}, n_1, \dots, n_M$ and m to the smart card.
14. The smart card does the following:
 - XORs sk with m to produce skm . That is, $skm = sk \oplus m$.
 - Splits skm into M shares $s_1, \dots, s_M$ using Shamir's threshold scheme. The share s_i is intended for juror i .
 - Encrypts each share of skm for the juror i with his/her corresponding $K_{1,i}$. Let the result be denoted by S_i . That is, $S_i = E_{K_{1,i}}(s_i)$.
 - Sends $S_1, \dots, S_M$ to EA.
15. EA sends $S_1, \dots, S_M$ to the jurors $1, \dots, M$.
16. Juror i does the following:
 - Decrypts S_i with $K_{2,i}$ to recover his/her share. That is, $s_i = D_{K_{2,i}}(S_i)$.
 - Stores his/her share.
17. EA accepts the registration.
18. CA publishes the user's pk .

6.3.2 Key Recovery for Law Enforcement

1. LEA presents the case to LA.
2. LA assesses any prima facie evidence to proceed or reject the claim.
3. Should the case be accepted, LA will notify the jury through EA. EA will also send the mask m to LA.
4. Each juror i returns his/her share s_i to LA.
5. If the number of received shares exceeds the threshold, LA can recover the XORed private key skm .
6. LA XORs the mask m to recover the private key sk .

The shares are returned directly to LA in order to prevent EA from knowing the private key of the suspect (EA does not need it).

6.3.3 Efficiency Consideration

This scheme requires a lot of computations but they are distributed in such a way that makes it feasible in practice. Below is a breakdown of the computation for each party :

- User's smart card needs to prepare one key pair, XORs the private key with the mask and splits the result into M shares. It also has to encrypt M times using jurors' keys. User then only has to wait for EA's notice for acceptance. Typically, M is a small integer (say 7). Computation is not excessive because a user needs to register his/her private key occasionally and non-interactively. Hence, efficiency is not a critical issue.
- EA needs to generate one mask. Also, EA needs to save user's information including one mask and the jury information for each user. Given that EA is a dedicated organization, this is not excessive either.
- LA has to recover the private key in case of law enforcement.
- Each juror needs to generate 2 keys for communication and decrypts once to recover the share. Also, he/she has to save only one share per user. This is considered acceptable because it does not occupy a lot of jury's computing time and memory.

Unlike Clipper chip, the key pair is generated by the smart card and not embedded during manufacturing. However, the security of this scheme still heavily depends on the security of the smart card itself.

However, a software solution offers a flexibility and versatility that hardware does not have. A family of compatible products is an excellent way to sell new technology. Vendors will often offer capability of beginning with low-cost software, with the option of upgrading to higher-performance and more expensive hardware when needed.

A software solution that prevents the user from cheating without resorting to a smart card is as follows.

6.4 Scheme 2

6.4.1 Key Registration

This is based on the subset method for RSA in [28] (p.127&128). In the subset method, the set of trustees is divided into different subsets such that each subset has a unique composition. The same secret is then shared repeatedly among trustees in the subset. To recover the secret, it is sufficient to have all shares of only any one subset. Similarly, M jurors are first randomly selected by the EA to form a jury. The user's private key is then repeatedly shared among different subsets of the same jury. To recover the private key, LA needs to have all shares of any one subset from the jury. Consider the following parameters for secret sharing: $n = M, T = t + 1$.

1. User requests key escrow service from EA.
2. Go through the jury challenge protocol in step 2 to 10 of section 6.3.1 in scheme 1.
3. User does the following:
 - Chooses two primes p and q congruent to 3 *mod* 4 as his/her private key, and computes $n = p * q$.
 - Generates all T -subsets of the jury consisting of M selected jurors. where a subset is represented by a T -tuple $J^s = (J_1, \dots, J_T)$ and $J_1, \dots, J_T$ represent different jurors from the chosen jury. There are C_T^M such subsets. For each tuple J^s , the user repeats the following steps.

- Chooses randomly T Jacobi symbol $+1$ integers $X_{1,s}, \dots, X_{T,s}$ in Z_n^* , where Z_n^* is the multiplicative group of the integers between 1 and n which are relatively prime to n .
- Computes the following:
 - $X_s = X_{1,s} * \dots * X_{T,s} \bmod n$.
 - $X_{k,s}^2 = X_{k,s} * X_{k,s} \bmod n$ for $k = 1, \dots, T$.
 - $Z_s = X_{1,s}^2 * \dots * X_{T,s}^2 \bmod n$.
Note that Z_s itself is a square. One square root of $Z_s \bmod n$ is X_s which has Jacobi symbol $+1$.
 - Computes Y_s , which is one of the Jacobi symbol -1 roots $\bmod n$ of Z_s . $X_{1,s}^2, \dots, X_{T,s}^2$ will be the public pieces and $X_{1,s}, \dots, X_{T,s}$ will be the corresponding private pieces.
- To each juror k in J^s , sends the private piece $X_{k,s}$ and n through EA as in scheme 1.
- Sends EA the value of Y_s and n .

4. Juror k in J^s does the following:

- Checks that $X_{k,s}$ has Jacobi symbol $+1$.
- Computes $X_{k,s}^2 = X_{k,s} * X_{k,s} \bmod n$.
- Sends $X_{k,s}^2$ to EA.
- Stores $X_{k,s}$.

5. EA does the following:

- Checks whether $Y_s^2 \bmod n = X_{1,s}^2 * \dots * X_{T,s}^2 \bmod n$. In this way, it is possible to verify whether the user has cheated.
- Checks whether n is in fact a product of 2 prime numbers.
- Stores the user and jury information, Y_s and n .
- Accepts the registration (e.g., by notifying CA and jury).

6. CA publishes the user's pk .

6.4.2 Key Recovery for Law Enforcement

1. LEA presents the case to LA.
2. LA assesses any prima facie evidence to proceed or reject.
3. Should the case be accepted, LA will notify the jury through EA.
4. EA returns Y_s and n to LA.
5. Each juror returns his/her all $X_{i,s}$ to LA.
6. If LA receives all T values of $X_{i,s}$ in at least one tuple J^s , LA then computes the product of the received pieces to obtain X_s . Knowing X_s , Y_s and n , it is easy to factorise n ([36]) and to recover the private key.

6.4.3 Efficiency Consideration

In this scheme, the user has to generate one key pair but he/she has to split repeatedly this single private key into T shares for all possible combinations of subsets in the same chosen jury (total C_T^M ways). He/she has to generate T Jacobi symbol +1 integers and to compute Z_s and its square root Y_s of Jacobi symbol -1 for each subset. Computation by EA involves simple verification and factorization procedures. For the jurors, computation is trivial but the memory requirement is higher. Instead of escrowing one share per user, each juror has to escrow C_{T-1}^{M-1} shares per user because each juror belongs to C_{T-1}^{M-1} subsets of the same jury in the secret sharing scheme. Since M and T are typically small values (say 7 and 4) and each juror escrows only a few users (say 10), it is believed that this is still acceptable.

6.5 Scheme 3

In this scheme, a user can be prevented from cheating by using the “cut and choose” protocol ([35]). Here the user arbitrarily generates many private and public key pairs (e.g., 100 pairs) and sends shares of each of them to the juries according some secret sharing schemes (see [40]). EA then opens all except one and verifies that they are all correct. The remaining unopened private key is taken to be correct and will be actually used by the user and

to be escrowed by the jurors. If the number of key pairs is large and the penalty for cheating is high, it is in the user's interest not to cheat.

6.5.1 Key Registration

Let N be the number of key pairs generated by the user in the “cut and choose” protocol.

1. User requests key escrow service from EA.
2. Go through the jury challenge protocol in step 2 to 10 of section 6.3.1 in scheme 1.
3. EA generates N masks m_j , for $j = 1, \dots, N$. Each mask is a random number which is used to hide the private key of a user.
4. EA sends the N masks to the user.
5. User does the following :
 - Creates N pairs of RSA public and private keys (pk_j, sk_j) , for $j = 1, \dots, N$.
 - XORs each sk_j with the mask m_j . Let the result be skm_j . Thus, $skm_j = sk_j \oplus m_j$.
 - Splits each skm_j into M shares, $s_{1,j}, \dots, s_{M,j}$. The share $s_{i,j}$ is intended for juror i .
 - Collects the shares of all skm_j intended for juror i , encrypts it with public key $K_{1,i}$ (let's call the result S_i). That is, $S_i = E_{K_{1,i}}(s_{i,1}, \dots, s_{i,N})$
 - Sends $S_1, \dots, S_M$ and $pk_1, \dots, pk_N$ to EA.
6. EA does the following :
 - Generates a random number r between 1 and N .
 - Sends $S_1, \dots, S_M$ and r to juries $1, \dots, M$.
7. Each juror i does the following :
 - Decrypts S_i with $K_{2,i}$ to recover the N shares. That is, $(s_{i,1}, \dots, s_{i,N}) = D_{K_{2,i}}(S_i)$.

- Sends back the $N - 1$ shares, except $s_{i,r}$, to EA.
8. EA does the following :
- Recovers the $N - 1$ skm 's. Each skm_j can be recovered from shares $s_{1,j}, \dots, s_{M,j}$.
 - XORs each with its mask. That is, $sk_j = skm_j \oplus m_j$. In this way, sk_j except sk_r can be recovered.
 - Verifies that all recovered $N - 1$ pairs of $(pk_1, sk_1), \dots, (pk_N, sk_N)$, excluding (pk_r, sk_r) , are all correct.
 - Stores the user information, the jury list and the remaining mask m_r securely.
 - Accepts the registration (e.g., by notifying CA and the jurors). The corresponding public key will be pk_r .
9. CA publishes the user's pk_r .
10. Jurors $1, \dots, M$, securely store the remaining shares $s_{1,r}, \dots, s_{M,r}$. These shares are used to recover sk_r , if necessary.

6.5.2 Key Recovery for Law Enforcement

1. LEA presents the case to the legal authorities.
2. LA assesses any prima facie evidence to proceed or reject the claim.
3. Should the case be accepted, LA will notify the jurors through EA. EA will also send the mask m_r to LA.
4. Each juror i returns his/her share $s_{i,r}$ to LA.
5. If the total number of shares exceeds the threshold, LA can recover the XORed private key skm_r .
6. LA XORs the mask m_r to recover the private key sk_r .

6.5.3 Efficiency Consideration

This scheme is very simple but computationally most inefficient. Below is a breakdown of the computation for each party :

- Each user needs to prepare N key pairs, XORs each private key with the mask and splits the result to M shares. He/she also has to encrypt M times using jurors' keys. He/she then only has to wait for EA's notice for which one key pair to use. Depending on the speed of his/her CPU and values of M and N , this is not excessive because a user needs to register his/her private key occasionally and non-interactively. Hence, efficiency is not a critical issue.
- EA needs to generate N masks, encrypt M times using jurors' keys, recover $N - 1$ keys and verify that they are correct key pairs. Also, EA needs to save user's information including one mask and the jury list for each user. Given that EA is a dedicated organization, this is not excessive either.
- LEA has to recover the private key in case of law enforcement.
- Each juror needs only to generate two keys for communication and decrypt once to recover the shares. Also, he/she has to save only one share per user. This is considered acceptable because it does not occupy a lot of jury's computing time and memory.

Chapter 7

Security Analysis

This chapter evaluates the strengths and possible attacks of the three proposed democratic key escrow schemes. Some counter-measures are proposed to prevent or minimize these attacks.

7.1 Strengths

The three proposed schemes provide better protection of privacy from abuse of power and collusion of different parties. This section gives a brief summary of the strengths in these schemes and they are as follows:

1. These schemes can be easily understood by all parties because they are very similar to the well established jury system.
2. Shares are stored independently by each juror. This is democratic because the escrowed key is conceptually in the hands of user's peers (though randomly and in a scrambled way). Key recovery is possible only at the moment of suspected crime. This is fairer to the users and matches our perception of democracy.
3. Jurors can stay offline until EA requests key registration service. Jurors can also stay offline after storing the shares until notified by LA to return their shares.
4. Government cannot embark upon a mass wire-tapping because the keys are completely decentralized. Should the government become malicious (e.g., due to sudden change of government), all jurors just delete their shares!!

5. Even if EA's database is compromised, the private keys cannot be recovered because the shares are not stored in EA's database.
6. Using jurors' keys to forward shares from user to jury through EA, jurors cannot learn who the user is and the user cannot learn who the jurors are because they only deal with EA. Thus, it is very difficult for the user to prevent (by bribe, threat, etc) the jurors from returning their shares.
7. Even if all jurors in a jury collude together, the private key cannot be recovered without the mask kept by EA in schemes 1 & 3. In scheme 2, one private key may be compromised but the bulk of private keys are still secure. Also, if the jurors do not know among themselves before key recovery, it is infeasible for them to collude.
8. In scheme 3, using "cut and choose" protocol, the user can cheat only 1 in N cases. By making N sufficiently large and the penalty for cheating sufficiently high, the chance of cheating is minimal.
9. The threshold can be adjusted according to the society's need and/or applications. For example, the Parliament can renew and adjust the threshold level annually after review in order to balance the protection of privacy and the need for law enforcement. If the government becomes too aggressive or it is too difficult for LEA to recover the private key, the threshold can be adjusted accordingly.
10. The private key is chosen by the user. This prevents authorities from using weak keys to facilitate wire-tapping.
11. Separation of duty between EA and LA provides the required check and balance of power. LA is responsible for the juror population but the jury for each user is chosen by EA. Neither LA nor EA can manipulate the jury.
12. These schemes provide better opportunity to obtain international agreements because the electronic jury system is an independent and impartial system similar to the judicial trial systems both in common law and continental law.

7.2 Possible Attacks

This section explores some possible attacks to these schemes and explains how they can be avoided or minimized. These attacks are:

1. In scheme 2, assume LA is corrupt. Instead of admitting jurors fairly, LA systematically admits jurors who collude with LA. Some safeguard can be provided by disqualifying certain classes of people (e.g., civil servants, etc) but still cannot prevent a malicious LA from doing this. This problem can be better dealt with by establishing a special committee to guarantee that fair selection procedures are observed.
2. EA selects (not randomly) specific subsets of the jurors from the juror population provided by LA. A malicious government can then force them to hand over their shares, the private key can then be recovered. However, this scenario is extremely unlikely. Also, the same committee can check that EA indeed randomly selects the jury by periodic inspection.
3. Consider the case that the government is only interested to recover a small set of private keys from the whole set of users. EA can provide the required juries to the government and the government can then force the juries to give up their shares to the government to enable wire-tapping. Even if this were true, the bulk of private keys remain secure. Again this is highly unlikely. This situation can be largely avoided by requiring EA to be an independent third party.
4. As mentioned in 6.2.2, EA can cheat by secretly creating a pair of keys itself. EA then gives one of them to the user and pretends that the key comes from juror i . User will then encrypt juror i 's share using this key. This enables EA to recover user's secret key easily after obtaining enough shares. This can be prevented by allowing the user to challenge the jury. A challenged jury will be exposed to the user and the user can then check with the jury whether EA has cheated by verifying the encryption key. The challenged jury will then be replaced by a new one by repeating the key registration scheme again. This challenge protocol can be repeated until the user is satisfied that EA has not cheated (different users may have different needs). The probability that EA can cheat after N rounds is $(1/2)^{N-1}$ and can be theoretically made as small

as possible but practical consideration limits the number of rounds to a small value (In fact, it is also feasible perform the challenge protocol in parallel rather than in series). Of course, as EA is assumed to be an independent third party, EA can be heavily penalized if it is caught in cheating (by law or by market force). Also, if EA is dishonest, it can be easily discovered because it has to register many users and any one of them can reveal EA's dishonesty. The opened jurors can also be used to indicate whether EA has fairly selected jurors because this information can be made public (unless EA can collude with a large number of jurors).

5. In scheme 3, each juror cannot verify his/her share is correct and not just random bits, so there is always a chance one in N that the user can cheat.
6. It might be possible that the number of shares returned is not enough to recover the private key even there is a genuine need by LEA. This is also true in our jury system where a criminal is walked free. Of course, it is possible to require the jurors for the mandatory return of their shares if their sole responsibility is to escrow user's shares.

7.3 Subliminal Channel

The notion of subliminal channel was first introduced by Simmons in his Prisoners' problem [39]. Basically, a subliminal channel (or covert channel) is a covert communication channel that cannot be read by those for whom it is not intended.

It was pointed out [27] that an attacker can set up a covert channel (or shadow public key cryptosystem) by choosing his/her key pair in the following manner.

Instead of generating a key pair (P, S) as normal, the dishonest user generates two key pairs (P, S) and (P', S') , where (P, S) is a proper public and private key pair. (P', S') is a shadow key pair and $P' = f(P)$ where f is an easily computed and publicly known function. The attacker uses (P, S) in the same way as would an ordinary user, but keeps S' reserved as his/her shadow secret key. In order for someone to send a secret message through this covert channel without the fear of being wiretapped, the sender

computes $P' = f(P)$ and then encrypts the message using P' . The receiver of the message then decrypts it using S' . In an attack for RSA cryptosystem [21], the attacker does the following:

- Generates primes $p > p'$ and $q > q'$.
- Computes $n = p * q$ and $n' = p' * q'$.
- Sets $K_e = n'$
- Finds K_d such that $K_e * K_d = 1 \bmod (p - 1)(q - 1)$.
- Sets $K'_e = (n \bmod n') = (n \bmod K_e)$.
- Finds K'_d such that $K'_e * K'_d = 1 \bmod (p' - 1)(q' - 1)$.
- Hands over p, q, n, K_e, K_d to EA but keeps p', q' secret.

The accomplices of the attacker encrypt a message m using K'_e, n' derived from K_e, n which can be obtained from a CA legitimately. The attacker can decrypt it by K'_d . In this way, the attacker can escrow his/her private key and yet the authorities cannot decrypt the actual messages sent by his/her accomplices.

Five properties are stipulated to provide a failsafe key escrow system:

- Each user can be sure that the secret key is chosen securely, even if all the trustees and central authorities are malicious.
- The central authority will be guaranteed that the secret key for each user is chosen securely even if the user does not have access to a good random generator.
- Each user will be guaranteed that his/her secret key will remain secret unless a sufficient number of trustees release their shares of the key to the central authority.
- The central authority is guaranteed to be able to recover the escrowed key if necessary.
- The subliminal channel is avoided.

It is noted that one way to meet the above criteria is to require the key generation to be through the collaboration of the user and the EA. For simplicity, this thesis does not include this scenario in the above schemes. However, it is not difficult to extend the schemes to prevent this type of attack. For example, instead of letting a user to randomly choose his/her public key, EA can select a prime number as the user's public key. The user then generates two primes p, q as before and calculates the corresponding private key secretly. As the public key is a prime number, it is relatively prime to $(p - 1)(q - 1)$ so that a unique private key exists. In this way, it is impossible for the attacker to set $K_e = n' = p' * q'$ because K_e is now prime.

Also, by selecting the public key for the user, EA has no way of learning the user's private key as long as it does not know p and q . In this way, the problem of subliminal channel can be avoided.

Chapter 8

Practical Consideration

This chapter explores some practical issues which need to be considered.

8.1 Key Recovery

Key recovery is often more urgent. By properly selecting the jury and setting the threshold value for secret sharing of the private key, the process of key recovery can be made faster for law enforcement purposes. For example, it might be acceptable to have 7 jurors to form a jury and requires agreement of any 4 to recover an escrowed key. Also, jurors' shares can be stored in some portable device such as a mobile phone or an electronic wallet in order to facilitate urgent recovery.

8.2 Selection Criteria of Jurors

Ideally, every responsible user can be a juror. However, it is recommended to have a stricter criteria first and then gradually relax the requirement to allow more users to become jurors. The following are just a few guidelines :

- Jurors should be professionals with high personal integrity (e.g., lecturers, doctors, accountants, etc).
- Jurors must have good character (e.g., no previous criminal records).
- Jurors must not be civil servants (e.g., policeman).

8.3 Key Renewal

It should be possible for a user to periodically renew his/her escrowed key. A user who feels threatened can change his/her private key more frequently while others can stick to the same private key for a long time.

8.4 Share Transfer

For some reasons, it is necessary to transfer jury's shares (e.g. if he/she is retired). This can be easily performed without key re-registration by the user through the following steps:

- EA selects a new juror.
- The new juror generates 2 keys using public key cryptography and distributes them in the same way as in step 14 of section 6.3.1 in scheme 1.
- The old juror sends his/her shares to the new juror through EA as before.
- The new juror recovers the shares.

8.5 Implementation

One possibility is to modify existing PGP software to incorporate a new escrow key ring for each juror. This requires the establishment of a key escrow agent(s). Thus, if a user wants to register his/her public key in the public key server, he/she has to escrow his/her private key with the key escrow agent beforehand. This thesis has implemented scheme 3 as a prototype and the source code in Java language is available in Appendix.

8.6 Wider Applications

The proposed schemes can shed light into the possibility of having a net criminal trial. Suppose the future net police arrests a net suspect who has allegedly committed a net crime with net evidence only (not impossible in the near future). The net suspect is notified and he/she can elect either to

have a traditional jury trial or a net jury trial. If he/she elects the latter, the net court will conduct a net trial. The net judge will call for both net prosecutor and net defence lawyer to submit net evidence. A panel of net jurors is summoned to give a net verdict. This idea may sound like a science fiction now but only time will tell.

Chapter 9

Conclusion

Advances in cryptography has created an opportunity to effectively use the power of the new emerging Information Superhighway. However, governments are hastily rushing through legislation to implement undemocratic key escrow schemes in order to regulate these fast growing electronic media. However, as pointed out in this thesis, these systems are insecure, unfair and undemocratic. Perhaps, eventually, one cannot escape from having some kind of key escrow system because not all individuals cannot be fully trusted. On the other hand, individuals cannot unconditionally trust authorities either. This thesis believes that the proposed democratic key escrow schemes can offer a democratic, practical and socially acceptable solution to this problem. It is hoped that these schemes can be a starting point for further research in this direction.

Appendix A

Source Code for Implementation of Scheme 3

INTRODUCTION

This appendix includes details of a prototype implementation for Scheme 3 as outlined in page 43. The prototype serves as a proof of the concept and requires further enhancement to become a realistic application. For simplicity, the following assumptions are made:

1. Only one user is considered;
2. All jurors are online;
3. Small key values are used(both p and q are smaller than MAX which is set to 1000);
4. The number of jurors in one jury is set to 5 only;
5. Any 3 of jurors in one jury can recover user's private key;
6. In the cut-and-choose protocol, only 10 key pairs are generated;
7. All jurors return their shares to LEA without deliberation;
8. Key recovery is set at shares 1, 3 and 5 in the program;
9. No consideration is given to multi-threading of server and synchronization of methods when accessing common data;
10. No detection of deadlock situation is considered;
11. No optimization of data structures is provided;
12. No optimization of message size is considered when a message is sent across the network;
13. Some servers' hosts and ports are set in the program;
14. No error recovery procedures are included;
15. Jury registration is not included;
16. Assume all communication channels are secure.

The prototype consists of EA, LA and 10 juror servers and user and LEA clients that are implemented in Java language of version 1.0. All the servers must run simultaneously but the user client is needed only in key registration phase and LEA client is needed only in key recovery phase.

In order to represent all entities effectively, an Entity class is defined. This class can be used to create objects for EA, LA and LEA. For user and juror, classes User and Juror are defined separately as subclasses of Entity class because they both share some commonalities of Entity class yet they have some other unique features.

In addition, some utility classes are also defined. RSA class is responsible for key and share operations such as generation of RSA key pairs or recovery of private key from the given shares. Classes U_InfoDialog, LEA_InfoDialog, Info and interface ResultProcessor are used to provide a user-friendly input dialog box.

Based the above classes, it is possible to define the following classes:

1. U
This class represents the user client program which is used for key registration;
2. EA
This class represents the EA server program which interacts with user and LEA clients;

3. LA

This class represents the LA server program which interacts with EA and J servers;

4. J

This class represents the juror server program which interacts with EA server and user and LEA clients.

Interactions among various clients and servers follow closely the protocol of Scheme 3. Instructions for compiling and running of these programs can be found in the README section below. Output of some example sessions can be found at the end of this appendix.

CLASS DESCRIPTION

This section provides the specification of all classes.

Let

w = number of jurors in one jury
t = number of jurors needed to recover a user's private key
cac = number of key pairs a user has to generate in the cut-and-choose protocol

1. Entity classes

class Entity

Data:

public String name, host (entity name and host name)
public int port (port number of the host)
public Socket socket (socket connection)
public PrintStream ps (print stream for the socket connection)
public DataInputStream dis (input stream for the socket)

Note: LA, EA and LEA are objects of class Entity

class User inherits Entity

Data:

public int ID (user ID)
public String address (user's address)
public int tel (user's telephone number)
public int chosen_nr (chosen key by EA NOT to be opened)
public long[] pks, sks, nmod (RSA key arrays of size w)
public long[] rmod (prime number array of size w for Shamir's scheme)
public RSA[] RSAKeys (array of size w of RSA key objects)
public Juror[] jury (array of size w of selected jury)

class Juror inherits Entity

Data:

public long pk, sk, p, q, nmod (juror's RSA key pair)
public int UserID (user ID)
public int chosen_nr (chosen key by EA NOT to be opened)
public long[] shares (array of size cac for holding shares)

2. Utility classes

class RSA

Methods:

```
public long Rand(long n)
generates random numbers less than n
private long Rand()
generates random numbers less than MAX (a predefined value)
private long LRand()
generates random numbers between MAX*MAX and MAX*MAX*MAX

public long Fastexp(long a, long z, long n)
calculates fast exponentiation
private long gcd(long a, long n)
calculates common greatest divisor
private boolean TestPrime(long n)
tests whether n is prime
private long Prime()
generates a prime number less than MAX
private long LPrime()
generates a prime number between MAX*MAX and MAX*MAX*MAX
private long Inverse(long n2, long n1)
finds the inverse of a number

private long Public_Key()
generates a public key
private long Private_Key()
generates a private key
public void Gen_UKey()
generates an RSA key pair for user
public void Gen_JKey()
generates an RSA key pair for juror
public long Encrypt(long X)
encrypts X
public long Decrypt(long Y)
decrypts Y

public void Split_key()
splits a private key into w shares using Shamir's scheme
public void Encrypt_Share(Juror[] jury)
encrypts each share with the corresponding juror's public key
public void Decrypt_Share(long[] sks, long[] mods)
decrypts each share with the corresponding juror's private key
public long Recover_key(int[] pos, long[] shadow, long modulus)
recovers the private key from any t shares
```

the following methods allow changing of private data

```
public void SetPK(long pk)
public void SetSK(long sk)
public void SetN(long modulus)
```

the following methods allow access of private data

```
public long PK()
public long SK()
public long N()
public long P()
public long Q()
public long M()
public long R()
public long[] SHARES()
public long SHARE(int pos)
```

Data:

```
long p, q, m, e, d, n (for RSA key pair)
long r, a[], share[] (for Shamir scheme)
static final int PRIME_WITNESS (maximum value for Lehman's test)
static final int MAX = 1000 (arbitrary set constant)
static final int t = 3, w = 5 (threshold values)
```

class U_InfoDialog inherits Dialog

```
private TextField field1, field2, field3
```

these fields are used to capture input data from the dialog box by user

class LEA_InfoDialog inherits Dialog

```
private TextField field1, field2, field3
```

these fields are used to capture input data from the dialog box by LEA

class Info

```
public String field1, field2, field3
```

these fields are used to store the input data from the dialog box

interface ResultProcessor

```
public void processResult(Dialog source, Object obj)
```

this method allows the implementor program to obtain data from dialog box source

3. User client program

class U inherits Frame and implements ResultProcessor

Buttons:

User Registration Information

Display an input dialog box and capture user information

EA Connection Information

Display an input dialog box and capture EA information

Request Jury

Request EA to send jury's encryption keys to user

Challenge Jury

Requested jury is challenged

EA has to send juror's connection information to user

User then connects to each juror of the challenged jury

Each juror sends p and q to user

User checks if $p \cdot q = n$ for each juror

If this is true for all w jurors, then the EA is honest

Accept Chosen Jury

User accepts the selected jury

User proceeds to generate key pairs

Generate Keys

Generates all cac RSA key pairs for cut-and-choose protocol

Write all keys to a file called "Ukey.dat"

Generate Shares

Reads cac key pairs from "Ukey.dat" file

Generates w shares for each of all cac private keys

Send Shares to Jury

Sends all shares to jury via EA

Waits for the reply from EA

If EA accepts registration, EA sends which key pair is chosen

Write selected key pair to the same file

Overwrites the previous content of the file

Label:

Status Bar

TextField:

status

gives the instruction for the next step to follow

notifies the result of each step

Data:

TextField status
Entity LA
Entity EA
User user
FileOutputStream fos (write to file "Ukey.dat")
PrintStream f
FileInputStream fis (read from file "Ukey.dat")
DataInputStream rf
String fname (file name)
int w, cac

4. EA server program

class EA

Methods:

public void EA_listen()
Starts the EA server
Reads the first message from the client
Classify each client
Handles each client according to its type

public void handle_LEA(Socket sock, DataInputStream cin, PrintStream cout)
Handles LEA request for recovery suspect's private key
by sending all w jurors connection information to LEA

public void handle_U(Socket sock, DataInputStream cin, PrintStream cout)
Handles user's registration by recording user information and
then sends back user's ID

public void Summon_J(Socket sock, DataInputStream cin, PrintStream cout)
EA requests jury from LA
LA sends w randomly selected jurors
LA sends connection information of this jury to EA
EA then summons the jury
Each juror sends the encryption key to EA
EA pass all w encryption keys to user

public void Challenge_J(Socket sock, DataInputStream cin, PrintStream cout)
User challenges the selected jury
EA sends connection information of the challenged jurors to user

```

public void Accept_J(Socket sock, DataInputStream cin, PrintStream cout)
User sends all cac public keys to EA
User sends w encrypted shares of corresponding private key to EA
EA sends the shares to corresponding jurors
EA generates a random number less than cac
EA sends cac to each juror
Each juror returns w-1 decrypted unchosen shares to EA
EA recovers the w-1 private keys
EA checks whether these keys are genuine
EA notifies user the result

```

Data:

```

int next (next user ID)
RSA aRSA
Entity LA
Entity EA
User[] (array of all users)
User (current user)
ServerSocket listener (server socket)

```

5.LA server program

class LA

Methods:

```

public int Rand(int n)
Generates a random number less than n

public void readFile()
Reads in the juror population kept in the file "Jlist.dat"

public void LA_listen()
Starts the LA server
Reads the first message from the client
Classify each client
Handles each client according to its type

public void handle_J(DataInputStream cin, PrintStream cout)
Accepts new juror registration
public void handle_EA(DataInputStream cin, PrintStream cout)
Randomly selects w jurors from the juror population
Sends connection information of these w jurors to EA

```

Data:

int next (next user ID)
Entity[] jurors (array of jurors)
ServerSocket listener (server socket)
FileInputStream fis (input stream to read juror file)
DataInputStream rf
String fname ("Jlist.dat")

6. J server program

class J

Methods:

public void J_listen()
starts the J server
reads the first message from the client
classify each client
handles each client according to its type

public void handle_EA(DataInputStream cin, PrintStream cout)
generates a key pair
sends the encryption key to EA
if user does not challenge this juror, receives all cac shares from EA
decrypt all shares and returns all unchosen shares to EA

public void handle_U(DataInputStream cin, PrintStream cout)
user challenges this juror
juror sends the values of p and q to user

public void handle_LEA(DataInputStream cin, PrintStream cout)
LEA requests share for recovery of user's private key
this juror sends the share to LEA

Data:

RSA aRSA
Entity LA
Entity EA
Juror juror
ServerSocket listener (server socket)

7. LEA client program

class LEA extends Frame implements ResultProcessor

Buttons:

User Information

display an input dialog box and capture user information

EA Connection Information

display an input dialog box and capture EA information

Request Jury's Shares

requests jury's connection information from EA

sends request to each of w jurors

recovers the suspect's private key

recover User's Key

Label:

Status Bar

TextField:

status

Gives the instruction for the next step to follow

notifies the result of each step

Data:

TextField status

Entity EA

User user

int w

long r, sk

long[] shares

README

This section explain how to install, compile and run these programs.
Follow the steps below:

1. Create a new directory;
2. Copy all programs into this directory;
3. Change the host name and port number of the servers if necessary;
4. Compile all modules using the following command:
`javac -deprecation *.java`
5. Run the clients and servers in the following order:
 - 5.1 LA by typing `java LA`
 - 5.2 EA by typing `java EA`
 - 5.3 J by typing `java <port number>` (port number from 1 to 10)
 - 5.4 U by typing `java U`
 - 5.5 LEA by typing `java LEA`
6. Instructions for user client:
 - 6.1 push button "User Registration Information" and enter user information;
 - 6.2 push button "EA Connection Information" and enter EA information;
 - 6.3 push button "Request Jury", user can either accept or challenge the jury;
 - 6.4 push button "Challenge Jury" to challenge selected jury;
 - 6.5 repeat the steps 6.3 and 6.4 until user is satisfied that EA is honest;
 - 6.6 push button "Accept Chosen Jury" to accept selected jury;
 - 6.7 push button "Generate Keys" to generate cac key pairs which are written to a data file called Ukey.dat;
 - 6.8 push button "Generate Shares" to generate w shares for each private key;
 - 6.9 push button "Send Shares to Jury" to send shares to jury via EA;
 - 6.10 if user does not cheat, the chosen key will be rewritten to the same data file;
 - 6.11 user can try to cheat by changing some private keys in the data file before share generation. User may be caught cheating by EA and a warning message will be displayed.
7. Instructions for LEA client:
 - 7.1 push button "User Information" and enter user information;
 - 7.2 push button "EA Connection Information" and enter EA information;
 - 7.3 push button "Request Jury's Shares" to request suspect's shares from the jury;
 - 7.4 push button "Recover User's Key" to recover the suspect's private key which will be displayed in the screen.

SOURCE CODE

1. Entity.java

```
import java.net.*;
import java.io.*;

class Entity
{
    public String name, host;
    public int port;
    public Socket sock;
    public PrintStream ps;
    public DataInputStream dis;
}
```

2. User.java

```
class User extends Entity
{
    public String address;
    public int tel;
    public int ID, chosen_nr;
    public long[] pks, sks, nmod, rmod;
    public RSA[] RSAKeys;
    public Juror[] jury;
}
```

3. Juror.java

```
class Juror extends Entity
{
    public long pk, sk, p, q, nmod;
    public int chosen_nr, UserID;
    public long[] shares;
}
```

4. Info.java

```
class Info
{
    /*
    this class contains fields the are used to store the input data from dialog box
    */
    public String fd1;
    public String fd2;
    public String fd3;
    public Info(String f1, String f2, String f3)
    { fd1 = f1; fd2 = f2; fd3 = f3; }
}
```

5. ResultProcessor.java

```
import java.awt.*;

interface ResultProcessor
{
    /*
    this interface allows the implementor program to obtain data from dialog box
    */
    public void processResult(Dialog source, Object obj);
}
```

6.RSA.java

```
public class RSA
{
    public RSA()
    {
        p = q = m = e = d = n = -1;
        a = new long[t];
        share = new long[w];
    }

    //generate random numbers less than n and MAX
    public long Rand(long n)
    {
        long a = (long) ((Math.random()*MAX) % n);
        return a;
    }

    //generate random numbers less than MAX (a predefined value)
    private long Rand()
    {
        long a = (long) (Math.random() * MAX);
        return a;
    }

    //generate random numbers between MAX*MAX and MAX*MAX*MAX
    //this is used to generate the prime number for Shamir's scheme
    private long LRand()
    {
        boolean done = false;
        long a = 0;
        while(!done)
        {
            a = (long) (Math.random() * MAX * MAX * MAX);
            if(a > MAX * MAX)
                done = true;
        }
    }
}
```



```

        return a;
    }

//compute a^z mod n
public long Fastexp(long a, long z, long n)
{
    long x = 1;
    while (z != 0)
    {
        while (z % 2 == 0)
        {
            z /= 2;
            a = ( ( a % n ) * ( a % n ) ) % n;
        }
        z--;
        x = ( ( x % n ) * ( a % n ) ) % n;
    }
    return x;
}

```

//compute the common greatest divisor by using
//Euclidean algorithm

```

private long gcd(long a, long n)
{
    if(a > n)
    {
        long tmp = a;
        a = n;
        n = tmp;
    }
    long g0 = n;
    long g1 = a;
    long g2 = 0;

    while(g1 != 0)
    {
        g2 = g0 % g1;
        g0 = g1;
        g1 = g2;
    }
    return g0;
}

```

//test whether n is prime by using Lehman's test

```

private boolean TestPrime(long n)
{
    if(n % 2 == 0) return false;

    int i = 0;
    long a, result;

```

```

        while (i < PRIME_WITNESS)
        {
            a = Rand(n);
            if(gcd(a, n) > 1) return false;
            result = Fastexp(a, (n-1)/2, n);
            if((result == 1) || (result == n-1)) i++;
            else return false;
        }
        return true;
    }

//generate a prime number less than MAX
private long Prime()
{
    boolean done = false;
    long prime = 1;
    while((!done) || (prime == 1))
    {
        prime = Rand();
        done = TestPrime(prime);
    }
    return prime;
}

//generate a prime number between MAX*MAX and MAX*MAX*MAX
private long LPrime()
{
    boolean done = false;
    long prime = 1;
    while((!done) || (prime == 1))
    {
        prime = LRand();
        done = TestPrime(prime);
    }
    return prime;
}

//find the inverse of n2 mod n1 by using
//extended Euclidean algorithm
private long Inverse(long n2, long n1)
{
    if(n2 < 0) n2 = n1 + n2;
    long inverse;
    long tmp = n1;
    long a1 = 1;
    long b1 = 0;
    long a2 = 0;
    long b2 = 1;
    long t = 0;
    long q = n1 / n2;
    long r = n1 - q * n2;

```

```

        while (r != 0)
        {
            n1 = n2;
            n2 = r;
            t = a2;
            a2 = a1 - q * a2;
            a1 = t;
            t = b2;
            b2 = b1 - q * b2;
            b1 = t;
            q = n1 / n2;
            r = n1 - q * n2;
        }
        inverse = (b2 > 0) ? b2 : tmp + b2;
        return inverse;
    }

//generate a public key
private long Public_Key()
{
    long aRand = 0;
    boolean done = false;
    while(!done)
    {
        aRand = Rand(m);
        if (gcd(aRand, m) == 1) done = true;
    }
    System.out.println("public key: " + aRand);
    return aRand;
}

//generate a private key
private long Private_Key()
{
    long aPKey;
    aPKey = Inverse(e, m);
    System.out.println("private key: " + aPKey);
    return aPKey;
}

//generate an RSA key pair for user
public void Gen_UKey()
{
    while(true)
    {
        p = Prime();
        if(p < 200) break;
    }
    System.out.println("p: " + p);
}

```

```

        while(true)
        {
            q = Prime();
            if((q < 200) && (q != p)) break;
        }
        System.out.println("q: " + q);
        n = p * q;
        System.out.println("n: " + n);
        m = (p-1) * (q-1);
        System.out.println("m: " + m);
        e = Public_Key();
        d = Private_Key();
        long unity = (e * d) % m;
        System.out.println("e * d mod m = " + unity);
    }

```

//generate an RSA key pair for juror

```

public void Gen_JKey()
{
    while(true)
    {
        p = Prime();
        if(p > 200) break;
    }
    System.out.println("p: " + p);
    while(true)
    {
        q = Prime();
        if((q > 200) && (q != p)) break;
    }
    System.out.println("q: " + q);
    n = p * q;
    System.out.println("n: " + n);
    m = (p-1) * (q-1);
    System.out.println("m: " + m);
    e = Public_Key();
    d = Private_Key();
    long unity = (e * d) % m;
    System.out.println("e * d mod m = " + unity);
}

```

//encrypt X

```

public long Encrypt(long X)
{
    long Y = Fastexp(X, e, n);
    return Y;
}

```

```

//decrypt Y
public long Decrypt(long Y)
{
    long X = Fastexp(Y, d, n);
    return X;
}

//split a private key into w shares using Shamir's scheme
public void Split_key()
{
    r = LPrime(); //r must be greater than n or MAX*MAX
    while((r <= d) || (r <= w))
        r = Prime();
    a[0] = share[0] = d; //d is the secret
    System.out.println("r = " + r);
//generate w different numbers
    for(int i=1; i<t; i++)
    {
        boolean done = false;
        while(!done)
        {
            int collision = 0;
            a[i] = Rand(r);
            for(int j = 0; j < i; j++)
                if(a[i] == a[j])
                    collision++;
            if(collision == 0) done = true;
        }
        share[0] += a[i];
    }
//generate shares
    share[0] %= r;
    for(int i = 0; i < t; i++)
        System.out.println("coeff " + i + " = " + a[i]);
    System.out.println("share " + 0 + " = " + share[0]);
    for(int i = 1; i < w; i++)
    {
        share[i] = a[0] + (i + 1) * a[1];
        for(int j = 2; j < t; j++)
            share[i] += a[j] * Fastexp(i + 1, j, r);
        share[i] %= r;
        System.out.println("share " + i + " = " + share[i]);
    }
}

//encrypt each share with the corresponding juror's public key
public void Encrypt_Share(Juror[] jury)
{
    for(int i = 0; i < w; i++)
        share[i] = Fastexp(share[i], jury[i].pk, jury[i].nmod);
}

```

//decrypt each share with the corresponding juror's private key

```
public void Decrypt_Share(long[] sks, long[] mods)
{
    for(int i = 1; i < w; i++)
        share[i] = Fastexp(share[i], sks[i], mods[i]);
}
```

//recover the private key from any t shares

```
public long Recover_key(int[] pos, long[] shadow, long modulus)
{
    long sum = 0;
    for(int i = 0; i < t; i++)
    {
        long numerator = 1;
        long denominator = 1;
        for(int j = 0; j < t; j++)
        {
            if (j != i)
            {
                numerator *= -pos[j];
                denominator *= (pos[i] - pos[j]);
            }
        }
        if(numerator < 0) numerator = modulus + numerator;
        sum += shadow[i] * numerator * Inverse(denominator, modulus);
    }
    sum %= modulus;
    return sum;
}
```

//the following methods allow changing of private data

```
public void SetPK(long pk)
{
    e = pk;
}
```

```
public void SetSK(long sk)
{
    d = sk;
}
```

```
public void SetN(long modulus)
{
    n = modulus;
}
```

//the following methods allow access of private data

```
public long PK()
{
    return e;
}
```

```
public long SK()
{
    return d;
}
```

```
public long N()
{
    return n;
}
```

```
public long P()
{
    return p;
}
```

```
public long Q()
{
    return q;
}
```

```
public long M()
{
    return m;
}
```

```
public long R()
{
    return r;
}
```

```
public long[] SHARES() //return all shares
{
    return share;
}
```

```
public long SHARE(int pos) //return one specified share
{
    return share[pos];
}
```

```
private long p, q, m, e, d, n, r, a[], share[];
static final int PRIME_WITNESS = 99;
static final int MAX = 1000;
static final int t = 3;
static final int w = 5;
```

```
}
```

7. Jlist.dat

```
peter1
vivaldi
6001
peter2
vivaldi
6002
peter3
vivaldi
6003
peter4
vivaldi
6004
peter5
vivaldi
6005
peter6
vivaldi
6006
peter7
vivaldi
6007
peter8
vivaldi
6008
peter9
vivaldi
6009
peter10
vivaldi
6010
```

8. U.java

```
import java.awt.*;
import java.io.*;
import java.net.*;

/*
this class contains fields that are used to capture input data from dialog box
*/

class U_InfoDialog extends Dialog
{
/*
this constructor creates a GUI dialog box for a user to enter his personal data
*/
    public U_InfoDialog(U parent, Info u, String fD1, String fD2, String fD3)
    {
```



```

        super(parent, "Input Info", true);
        Panel p1 = new Panel();
        p1.setLayout(new GridLayout(3, 2));
        p1.add(new Label(fD1));
        p1.add(FD1 = new TextField(u.fd1, 8));
        p1.add(new Label(fD2));
        p1.add(FD2 = new TextField(u.fd2, 8));
        p1.add(new Label(fD3));
        p1.add(FD3 = new TextField(u.fd3, 8));
        add("Center", p1);

        Panel p2 = new Panel();
        p2.add(new Button("Ok"));
        p2.add(new Button("Cancel"));
        add("South", p2);
        resize(350, 150);
    }

    /*
    this method handles action event and captures user input after a user
    pushes the "Ok" button
    */
    public boolean action(Event evt, Object arg)
    {
        if(arg.equals("Ok"))
        {
            dispose();
            ((ResultProcessor)getParent()).processResult(this,
                new Info(FD1.getText(), FD2.getText(), FD3.getText()));
        }
        else if (arg.equals("Cancel"))
            dispose();
        else return super.action(evt, arg);
        return true;
    }

    /*
    this method destroys the dialog box
    */
    public boolean handleEvent(Event evt)
    {
        if (evt.id == Event.WINDOW_DESTROY)
            dispose();
        else return super.handleEvent(evt);
        return true;
    }

    /*
    these fields are used to capture input data from the dialog box
    */
    private TextField FD1;
    private TextField FD2;
    private TextField FD3;
}

```

```

/*
User client program
*/

public class U extends Frame implements ResultProcessor
{
/*
This constructor creates entity objects and a GUI frame so that a
user can perform key escrow procedure and check the status easily
*/
    public U()
    {
        LA = new Entity();
        EA = new Entity();
        user = new User();
        fname = "Ukey.dat";
        w = 5;
        cac = 10;
        setTitle("User Registration");
        setLayout(new FlowLayout());
        add(new Button("User Registration Information"));
        add(new Button("EA Connection Information"));
        add(new Button("Request Jury"));
        add(new Button("Challenge Jury"));
        add(new Button("Accept Chosen Jury"));
        add(new Button("Generate Keys"));
        add(new Button("Generate Shares"));
        add(new Button("Send Shares to Jury"));
        Label st = new Label ("Status Bar");
        add(st);
        status = new TextField("Start: enter User Registration Information", 35);
        status.setEditable(false);
        add(status);
    }
/*
This is an implementation of the interface processResult
which actually pass data from a dialog box to this program
*/
    public void processResult(Dialog source, Object result)
    {
        if(source instanceof U_InfoDialog)
        {
            Info info = (Info)result;
            if("EA".equals(info.fd1)) //it is an EA input dialog box
            {
                EA.host = info.fd2;
                EA.port = Integer.parseInt(info.fd3);
            }
        }
    }
}

```

```

        else //it is a user input dialog box
        {
            user.name = info.fd1;
            user.address = info.fd2;
            user.tel = Integer.parseInt(info.fd3);
        }
    }
}

/*
this method destroys the user client window and exits the program
*/
public boolean handleEvent(Event evt)
{
    if(evt.id == Event.WINDOW_DESTROY) System.exit(0);
    return super.handleEvent(evt);
}

/*
this method captures action event and takes appropriate action
depending the type of event
*/
public boolean action(Event evt, Object arg)
{
    try{
        /*
        this event handles user's input of personal data
        */
        if (arg.equals("User Registration Information"))
        {
            System.out.println("User Registration Information");
            Info in1 = new Info("name", "uow", "8888");
            U_InfoDialog pd1 = new U_InfoDialog(this, in1, "name",
                                                "address", "tel no");
            pd1.show();
            status.setText("next step: enter EA information");
        }

        /*
        this event handles user's input of EA's connection information
        */
        else if(arg.equals("EA Connection Information"))
        {
            Info in2 = new Info("EA", "vivaldi", "3838");
            U_InfoDialog pd2 = new U_InfoDialog(this, in2, "name", "host",
                                                "port");
            pd2.show();
            System.out.println(EA.name + " " + EA.host + " " + EA.port);
            status.setText("Trying to connect to EA...");
            System.out.println("Trying to connect to EA...");

            /*
            connect to EA's socket
            */
            EA.sock = new Socket(EA.host, EA.port);
            EA.dis = new DataInputStream(EA.sock.getInputStream());
            EA.ps = new PrintStream(EA.sock.getOutputStream());
        }
    }
}

```

```

        status.setText("Connected to EA, request jury now!!!");
        System.out.println("Connected to EA, request jury now!!!");
    /*
    send request for key escrow service to EA
    */
        EA.ps.println("U REQ REG EA");
    /*
    send user data to EA
    */
        EA.ps.println(user.name);
        EA.ps.println(user.address);
        EA.ps.println(String.valueOf(user.tel));
    }
    /*
    this event handles user's request to EA for jury
    */
        else if(arg.equals("Request Jury"))
        {
            System.out.println("Request Jury");
            EA.ps.println("U REQ EA"); //send the request
            status.setText("Requesting jury...");
            System.out.println(EA.dis.readLine());//display EA's reply
        }
    /*
    read in encryption keys of w jurors from EA
    */
        user.jury = new Juror[w];
        for(int i = 0; i < w; i++)
        {
            System.out.println("pk of juror " + i);
            Juror juror = new Juror();
            juror.pk = Long.parseLong(EA.dis.readLine());
            System.out.println(String.valueOf(juror.pk));
            juror.nmod = Long.parseLong(EA.dis.readLine());
            System.out.println(String.valueOf(juror.nmod));
            user.jury[i] = juror;
        }
        status.setText("Challenge or Accept jury now!!!");
        System.out.println("Challenge or Accept jury now!!!");
    }
    /*
    this event handles challenge of jury by user
    */
        else if(arg.equals("Challenge Jury"))
        {
            System.out.println("Challenge Jury");
        }
    /*
    user sends challenge to EA
    */
        EA.ps.println("U CHALLENGE J EA");
        int honest = 0;
        System.out.println(EA.dis.readLine());//display EA's reply

```

```

/*
user reads connection information of w jurors from EA
and check with each juror whether EA is honest
*/
        for(int i = 0; i < w; i++)
        {
            System.out.println("connect to juror " + i);

/*
user reads connection information of each juror from EA
*/
            Juror juror = user.jury[i];
            juror.name = EA.dis.readLine();
            System.out.println(juror.name);
            juror.host = EA.dis.readLine();
            System.out.println(juror.host);
            juror.port = Integer.parseInt(EA.dis.readLine());
            System.out.println(String.valueOf(juror.port));

/*
user reads connects to each juror
*/
            juror.sock = new Socket(juror.host, juror.port);
            juror.dis = new
                DataInputStream(juror.sock.getInputStream());
            juror.ps = new PrintStream(juror.sock.getOutputStream());

/*
user sends challenge to juror
*/
            juror.ps.println("U CHALLENGE J");
            juror.ps.println(String.valueOf(user.ID));
            System.out.println(juror.dis.readLine());

/*
user reads p and q from each juror
*/
            juror.p = Long.parseLong(juror.dis.readLine());
            juror.q = Long.parseLong(juror.dis.readLine());
            System.out.println(String.valueOf(juror.p));
            System.out.println(String.valueOf(juror.q));
            juror.sock.close();

/*
user checks whether EA is honest with respect to this juror
*/
            if(juror.p*juror.q == juror.nmod) honest++;
            System.out.println("honesty value is: " +
                String.valueOf(honest));
        }

/*
notify user whether EA is honest
*/
        if(honest == w) status.setText("EA is honest!!!");
        else status.setText("EA is NOT honest!!!");

```

```

        if(honest == w) System.out.println("EA is honest!!!");
        else System.out.println("EA is NOT honest!!!");
    }

    /*
    this event handles user's acceptance of jury
    */
    else if (arg.equals("Accept Chosen Jury"))
    {
        status.setText("next step: Generate Keys!!!");
        System.out.println("next step: Generate Keys!!!");
    }

    /*
    this event handles generation of cac key pairs
    */
    else if (arg.equals("Generate Keys"))
    {
        System.out.println("Generate Keys and write to file");
        try{
            fos = new FileOutputStream(fname);
            f = new PrintStream(fos);
        }catch(IOException ioe) { }
        user.RSAKeys = new RSA[cac];
        for(int i=0; i< w; i++)
            user.jury[i].shares = new long[cac];

    /*
    generate cac key pairs and write all keys to a file
    */
        for(int i=0; i< cac; i++)
        {
            user.RSAKeys[i] = new RSA();
            user.RSAKeys[i].Gen_UKey();
            f.println(String.valueOf(user.RSAKeys[i].PK()));
            f.println(String.valueOf(user.RSAKeys[i].SK()));
            f.println(String.valueOf(user.RSAKeys[i].N()));
        }
        fos.close();
        status.setText("done!!! next step: Generate Shares now!!!");
    }

    /*
    this event handles generation of w shares for each of cac private keys
    Note: a user can try to cheat by tampering with the private keys in the file
    before this step is performed but he/she may be caught by EA
    */
    else if (arg.equals("Generate Shares"))
    {
        System.out.println(" Read file and Generate Shares");
        try
        {
            fis = new FileInputStream(fname);
            rf = new DataInputStream(fis);

```

```

        }catch(IOException ioe) { }

/*
read the keys back from the file and split each private key into w shares
*/
        for(int i = 0; i < cac; i++)
        {
            long pk = Long.parseLong(rf.readLine());
            user.RSAKeys[i].SetPK(pk);
            long sk = Long.parseLong(rf.readLine());
            user.RSAKeys[i].SetSK(sk);
            long n = Long.parseLong(rf.readLine());
            user.RSAKeys[i].SetN(n);
            user.RSAKeys[i].Split_key();

/*
encrypt each share with corresponding juror's encryption key
*/
            user.RSAKeys[i].Encrypt_Share(user.jury);
            for(int j = 0; j < w; j++)
            user.jury[j].shares[i] = user.RSAKeys[i].SHARE(j);
        }
        fis.close();
        status.setText("done!!! next step: Send Shares now!!!");
    }

/*
this event handles user's sending of shares to each of w jurors via EA after
user has accepted the jury
*/
        else if (arg.equals("Send Shares to Jury"))
        {
            System.out.println("Send encrypted shares to Jury");

/*
notify EA to receive shares
*/
            EA.ps.println("U SEND SHARES EA");
            status.setText("Sending shares!!!");

/*
send cac public keys to EA first
*/
            System.out.println("send cac public keys to EA first");

            for(int i = 0; i < cac; i++)
            {
                System.out.println("pk, n, r for i = " + i);
                EA.ps.println(String.valueOf(user.RSAKeys[i].PK()));
                System.out.println(String.valueOf
                    (user.RSAKeys[i].PK()));
                EA.ps.println(String.valueOf(user.RSAKeys[i].N()));
                System.out.println(String.valueOf(user.RSAKeys[i].N()));
                EA.ps.println(String.valueOf(user.RSAKeys[i].R()));
            }
        }
    }
}

```

```

        System.out.println(String.valueOf(user.RSAKeys[i].R()));
    }

/*
send encrypted shares to jurors via EA
*/
    System.out.println("send encrypted shares to jurors via EA");
    for(int i = 0; i < w; i++)
    {
        System.out.println("shares to juror " + i);
        for(int j = 0; j < cac; j++)
        {
            EA.ps.println(String.valueOf
                (user.jury[i].shares[j]));
            System.out.println(String.valueOf
                (user.jury[i].shares[j]));
        }
    }

/*
EA accepts key registration
*/
    if("EA ACCEPT KEY U".equals(EA.dis.readLine()))
    {
        System.out.println("EA accepts key registration");

/*
read the chosen key
*/
        String s = EA.dis.readLine();
        System.out.println("chosen key no: " + s);
        user.chosen_nr = Integer.parseInt(s);
        try
        {
            fis = new FileInputStream(fname);
            rf = new DataInputStream(fis);
        }catch(IOException ioe) { }
        long pk = 0, sk = 0, n = 0;

/*
overwrite the file with this chosen key
*/
        for(int i = 0; i <= user.chosen_nr; i++)
        {
            pk = Long.parseLong(rf.readLine());
            sk = Long.parseLong(rf.readLine());
            n = Long.parseLong(rf.readLine());
        }
        fis.close();
        try{
            fos = new FileOutputStream(fname);
            f = new PrintStream(fos);
        }catch(IOException ioe) { }
        System.out.println("pk: " + String.valueOf(pk));
        System.out.println("sk: " + String.valueOf(sk));
    }

```



```

        System.out.println("n: " + String.valueOf(n));
        f.println(String.valueOf(pk));
        f.println(String.valueOf(sk));
        f.println(String.valueOf(n));
        fos.close();

    /*
    notify user the result
    */

        status.setText("chosen key: " +
            String.valueOf(user.chosen_nr) + " ,key escrow done!!!");
        System.out.println("key escrow done!!!");
    }
    else
    {
        status.setText("You are caught cheating!!!");
        System.out.println("You are caught cheating!!!");
    }
    }
    else return super.action(evt, arg);
} catch(IOException ioe) { }
    repaint();
    return true;
}

public static void main(String[] args)
{
    Frame f = new U();
    f.resize(400, 200);
    f.show();
}

    TextField status;
    Entity LA;
    Entity EA;
    User user;
    FileOutputStream fos;
    PrintStream f;
    FileInputStream fis;
    DataInputStream rf;
    String fname;
    int w, cac;
}

```



```

//user challenges jury
                                Challenge_J(sock, cin, cout);
                                else
                                    break;
                                s = cin.readLine();
                                System.out.println(s);
                                }
                                System.out.println("call Accept_J");
                                Accept_J(sock, cin, cout); //user accepts jury
                                sock.close();
                                next++; //increment user ID
                                }
                                }

}

/*
this method handles LEA's request
*/

public void handle_LEA(Socket sock, DataInputStream cin, PrintStream cout) throws
IOException
{
/*
return the prime number used in secret sharing
*/
    System.out.println("send r : " +
                        String.valueOf(user.rmod[user.chosen_nr]));
    cout.println(String.valueOf(user.rmod[user.chosen_nr]));
    System.out.println(cin.readLine());
    cout.println("EA SEND JURY INFO LEA");
    System.out.println("EA SEND JURY INFO LEA");

/*
send suspect's jury to LEA
*/
    System.out.println("send suspect's jury");
    for(int i = 0; i < w; i++)
    {
        Juror juror = user.jury[i];
        cout.println(juror.name);
        System.out.println(juror.name);
        cout.println(juror.host);
        System.out.println(juror.host);
        cout.println(String.valueOf(juror.port));
        System.out.println(String.valueOf(juror.port));
    }
}

/*
this method handles registration of user's personal data
*/

```

```

public void handle_U(Socket sock, DataInputStream cin, PrintStream cout) throws
IOException
{
    user = new User();
    user.sock = sock;
    user.ID = next;
    user.dis = cin;
    user.ps = cout;
/*
record user info
*/
    System.out.println("record user info");
    user.name = user.dis.readLine();
    System.out.println(user.name);
    user.host = user.dis.readLine();
    System.out.println(user.host);
    user.port = Integer.parseInt(user.dis.readLine());
    System.out.println(String.valueOf(user.port));
    System.out.println(String.valueOf(next));
    users[next] = user;
}

/*
this method summons the jury
*/
    public void Summon_J(Socket sock, DataInputStream cin, PrintStream cout)
    throws IOException
    {
        System.out.println("summons the jury");
/*
this port is fixed and must be changed and recompiled again if this program
is run in another host other than "vivaldi"
*/
        LA.sock = new Socket("vivaldi", 8383);
        LA.dis = new DataInputStream(LA.sock.getInputStream());
        LA.ps = new PrintStream(LA.sock.getOutputStream());
/*
EA requests jury from LA
*/
        LA.ps.println("EA REQ J LA");
        System.out.println("EA REQ J LA");
        users[next].ps.println("EA SEND JKEYS U");
        System.out.println("EA SEND JKEYS U");
        System.out.println(LA.dis.readLine());

        users[next].jury = new Juror[w];
        System.out.println("read jury's connection information");
        for(int i = 0; i < w; i++)
        {
            System.out.println("Juror " + i);
/*
read jury's connection information
*/

```

```

        Juror juror = new Juror();

        juror.name = LA.dis.readLine();
        System.out.println(juror.name);
        juror.host = LA.dis.readLine();
        System.out.println(juror.host);
        String s = LA.dis.readLine();
        System.out.println(s);
        juror.port = Integer.parseInt(s);

        System.out.println(juror.name);
        System.out.println(juror.host);
        System.out.println(String.valueOf(juror.port));
    /*
    connect to each juror's socket
    */

        juror.sock = new Socket(juror.host, juror.port);
        juror.dis = new DataInputStream(juror.sock.getInputStream());
        juror.ps = new PrintStream(juror.sock.getOutputStream());
    /*
    summon each juror
    */

        juror.ps.println("EA SUMMON J");
        System.out.println("EA SUMMON J");
        juror.ps.println(String.valueOf(next));
        System.out.println(String.valueOf(next));
    /*
    each juror sends encryption key to EA
    */

        System.out.println(juror.dis.readLine());
        juror.pk = Long.parseLong(juror.dis.readLine());
        juror.nmod = Long.parseLong(juror.dis.readLine());
    /*
    EA send each juror's encryption key to user
    */

        user.ps.println(String.valueOf(juror.pk));
        System.out.println(String.valueOf(juror.pk));
        user.ps.println(String.valueOf(juror.nmod));
        System.out.println(String.valueOf(juror.nmod));
        user.jury[i] = juror;
    }
}

    /*
    this method handles challenge of jury
    */

    public void Challenge_J(Socket sock, DataInputStream cin, PrintStream cout)
    throws IOException
    {
        System.out.println("challenge of jury");
    }

```

```

/*
EA sends jury's connection information to user
*/
    user.ps.println("EA SEND JINFO U");
    System.out.println("EA SEND JINFO U");
    for(int i = 0; i < w; i++)
    {
        Juror juror = user.jury[i];
        juror.ps.println("EA CHALLENGE J");
        System.out.println("EA CHALLENGE J");
        user.ps.println(juror.name);
        System.out.println(juror.name);
        user.ps.println(juror.host);
        System.out.println(juror.host);
        user.ps.println(String.valueOf(juror.port));
        System.out.println(String.valueOf(juror.port));
    }
    LA.sock.close();
    for(int i = 0; i < w; i++)
    {
        user.jury[i].sock.close();
    }
}

/*
this method handles acceptance of jury by user
*/
public void Accept_J(Socket sock, DataInputStream cin, PrintStream cout)
throws IOException
{
    System.out.println("U SEND SHARES EA");
    user.pks = new long[cac];
    user.nmod = new long[cac];
    user.rmod = new long[cac];

/*
read cac public keys and prime numbers for secret sharing scheme of private keys
*/

    System.out.println("read cac pk, n and r");
    for(int i = 0; i < cac; i++)
    {
        System.out.println("user key: " + i);
        user.pks[i] = Long.parseLong(user.dis.readLine());
        user.nmod[i] = Long.parseLong(user.dis.readLine());
        user.rmod[i] = Long.parseLong(user.dis.readLine());
        System.out.println(String.valueOf(user.pks[i]));
        System.out.println(String.valueOf(user.nmod[i]));
        System.out.println(String.valueOf(user.rmod[i]));
    }

/*
read all shares of user's private keys from user
*/
    System.out.println("read all shares from user and send all shares to
jurors");

```

```

for(int i = 0; i < w; i++)
{
    System.out.println("juror: " + i);
    Juror juror = user.jury[i];
    juror.shares = new long[cac];
    juror.ps.println("EA SEND SHARES J");
    juror.ps.println(String.valueOf(user.ID));
    for(int j = 0; j < cac; j++)
    {
        System.out.println("encrypted share: " + j);
        juror.shares[j] = Long.parseLong(user.dis.readLine());
        juror.ps.println(String.valueOf(juror.shares[j]));
        System.out.println(String.valueOf(juror.shares[j]));
    }
}

/*
EA chooses a random number between 1 and cac and notify jury
*/

user.chosen_nr = (int)aRSA.Rand(cac);
System.out.println("chosen no: " + String.valueOf(user.chosen_nr));
for(int i = 0; i < w; i++)
{
    Juror juror = user.jury[i];
    juror.chosen_nr = user.chosen_nr;
    juror.ps.println("EA CHOOSE KEY J");
    juror.ps.println(String.valueOf(user.chosen_nr));
}

/*
jury send unchosen keys to EA
*/

System.out.println("jury send unchosen keys to EA");

for(int i = 0; i < w; i++)
{
    System.out.println("juror: " + i);
    Juror juror = user.jury[i];
    for(int j = 0; j < cac; j++)
    {
        System.out.println("decrypted share: " + j);
        if(j != user.chosen_nr)
        {
            juror.shares[j] =
                Long.parseLong(juror.dis.readLine());
            System.out.println(String.valueOf(juror.shares[j]));
        }
    }
}

/*
verify shares are correct
*/

System.out.println("verify shares are correct");
int[] pos = {1,3,5};// recover shares at these positions

```

```

        long[] shadow = new long[t];
        int honest = 0;
        for(int j = 0; j < cac; j++)
        {
            if(j != user.chosen_nr) //open all keys except the chosen number
            {
                for(int i = 0; i < t; i++)
                {
                    shadow[i] = user.jury[pos[i]-1].shares[j];
                }
            }
        }

/*
recover each private key
*/
        long sk = aRSA.Recover_key(pos, shadow ,user.rmod[j]);
        System.out.println("recovered sk: " + String.valueOf(sk));

/*
simple test to check whether each recovered private key is genuine
*/
        long ptext = aRSA.Rand(user.nmod[j]);
        long ctext = aRSA.Fastexp(ptext, user.pks[j],
user.nmod[j]);

        long rtext = aRSA.Fastexp(ctext, sk, user.nmod[j]);
        if(rtext == ptext) honest++;
        System.out.println("honesty value: " +
String.valueOf(honest));
    }

/*
notify user whether registration is accepted
*/
    if(honest == 9)
    {
        user.ps.println("EA ACCEPT KEY U");
        user.ps.println(String.valueOf(user.chosen_nr));
        System.out.println("ACCEPT KEY");
    }
    else
    {
        user.ps.println("EA not ACCEPT KEY U");
        System.out.println("not ACCEPT KEY");
    }
    LA.sock.close();
    for(int i = 0; i < w; i++)
    {
        user.jury[i].sock.close();
    }
}

public static void main(String[] args) throws IOException
{
    EA aEA = new EA();
}

```



```

        aEA.EA_listen();
    }

    int next, w, t, cac;
    RSA aRSA;
    Entity EA, LA;
    User[] users;
    User user;
    ServerSocket listener;
}

```

10. LA.java

```

import java.io.*;
import java.net.*;
import java.util.*;

```

```

class LA
{
    LA()
    {
        jurors = new Entity[99]; //juror population set to 99
        listener = null;
        fis = null;
        rf = null;
        fname = "Jlist.dat"; //file containing juror population
        w = 5;
    }
}

```

```

/*
this method generates a random number between 1 and n - 1
*/

```

```

    public int Rand(int n)
    {
        int a = (int) ((Math.random()*10000) % n);
        return a;
    }
}

```

```

/*
this method reads in connection information of all jurors in the whole population
kept in a file
*/

```

```

    public void readFile() throws IOException
    {
        System.out.println("read juror file");
        fis = new FileInputStream(fname);
        rf = new DataInputStream(fis);
        String s = null;
        while((s = rf.readLine()) != null)
        {

```

```

        jurors[next] = new Entity();
        jurors[next].name = s;
        jurors[next].host = rf.readLine();
        jurors[next].port = Integer.parseInt(rf.readLine());
        next++;
    }
    for(int i = 0; i < next; i++)
    {
        System.out.println(jurors[i].name);
        System.out.println(jurors[i].host);
        System.out.println(jurors[i].port);
    }
}

/*
this method starts the LA server
*/
public void LA_listen() throws IOException
{
    listener = new ServerSocket(8383); //port number is fixed
    while (true)
    {
        Socket sock = listener.accept();
        PrintStream cout = new PrintStream(sock.getOutputStream());
        DataInputStream cin = new
            DataInputStream(sock.getInputStream());
        String s = cin.readLine();
        System.out.println(s);
        if("J REQ REG LA".equals(s)) //it is a juror
        {
            handle_J(cin, cout);
            next++;
            sock.close();
        }
        else if("EA REQ J LA".equals(s)) //it is EA
        {
            handle_EA(cin, cout);
            sock.close();
        }
        else
            System.out.println("error");
    }
}

/*
this method handles juror registration
*/
public void handle_J(DataInputStream cin, PrintStream cout) throws IOException
{
    cout.println("LA ACK REQ J"); //acknowledge request
    Entity juror = new Entity();

    /*
    read in new juror's connection information
    */

```

```

        juror.name = cin.readLine();
        juror.host = cin.readLine();
        juror.port = Integer.parseInt(cin.readLine());
        jurors[next] = juror;
        cout.println("LA REG OK J");
        System.out.println(juror.name);
        System.out.println(juror.host);
        System.out.println(String.valueOf(juror.port));
    }
}
/*
this method handles request from EA to select a jury of w jurors
*/

public void handle_EA(DataInputStream cin, PrintStream cout) throws IOException
{
    System.out.println("send jury to EA");
    cout.println("LA SEND J EA");
    int[] a = new int[w];

    /*
    randomly select w different numbers less than the size of the juror population
    */
    a[0] = Rand(next);
    for(int i = 1; i < w; i++)
    {
        boolean done = false;
        while(!done)
        {
            int collision = 0;
            a[i] = Rand(next);
            for(int j = 0; j < i; j++)
            {
                if(a[i] == a[j])
                    collision++;
            }
            if(collision==0) done = true;
        }
    }
    for(int i = 0; i < w; i++)
    {
        System.out.println(jurors[a[i]].name);
        System.out.println(jurors[a[i]].host);
        System.out.println(jurors[a[i]].port);
    }

    /*
    send the selected jury to EA
    */
    for(int i = 0; i < w; i++)
    {
        cout.println(jurors[a[i]].name);
        cout.println(jurors[a[i]].host);
        cout.println(String.valueOf(jurors[a[i]].port));
    }
}

```

```

public static void main(String[] args)
{
    try{
        LA aLA = new LA();
        aLA.readFile();
        aLA.LA_listen();

    }catch(IOException ioe) { }

    int next, w;
    Entity[] jurors;
    ServerSocket listener;
    FileInputStream fis;
    DataInputStream rf;
    String fname;
}

```

11. J.java

```

import java.io.*;
import java.net.*;
import java.util.*;

```

```

public class J
{

```

```

    J(Juror j)
    {

```

```

        juror = j;
        cac = 10;
        aRSA = new RSA();
        LA = new Entity();
        EA = new Entity();
        listener = null;
    }

```

```

/*
this method starts the juror server
*/
    public void J_listen() throws IOException
    {
        System.out.println("juror port: " + String.valueOf(juror.port));
        listener = new ServerSocket(juror.port);
        while (true)
        {
            Socket sock = listener.accept();
            PrintStream cout = new PrintStream(sock.getOutputStream());
            DataInputStream cin = new DataInputStream(sock.getInputStream());
            String s = cin.readLine();
            System.out.println(s);
        }
    }

/*
check the type of client
*/

```

```

        if("EA SUMMON J".equals(s)) //it is EA
        {
            handle_EA(cin, cout);
            sock.close();
        }
        else if("U CHALLENGE J".equals(s)) //it is a user
        {
            handle_U(cin, cout);
            sock.close();
        }
        else if("LEA RECOVER J".equals(s)) //it is LEA
        {
            handle_LEA(cin, cout);
            sock.close();
        }
        else
            System.out.println("error");
    }
}

```

```

/*
this method handles summon from EA
*/
public void handle_EA(DataInputStream cin, PrintStream cout) throws IOException
{
    juror.UserID = Integer.parseInt(cin.readLine()); //get user ID
    aRSA.Gen_JKey();
    cout.println("JUROR SEND KEY EA");
    System.out.println("JUROR SEND KEY EA");

    /*
    send encryption key to user via EA
    */
    cout.println(String.valueOf(aRSA.PK()));
    System.out.println(String.valueOf(aRSA.PK()));
    cout.println(String.valueOf(aRSA.N()));
    System.out.println(String.valueOf(aRSA.N()));
    if(!("EA CHALLENGE J".equals(cin.readLine()))) //user accepts this juror
    {
        System.out.println("user accepts this juror");
        System.out.println("EA SEND SHARES J");
        juror.UserID = Integer.parseInt(cin.readLine());
        juror.shares = new long[cac];

        /*
        read in cac shares from the user via EA
        */
        System.out.println("read in cac shares from the user via EA");
        for(int i = 0; i < cac; i++)
        {
            System.out.println("share " + i);
            juror.shares[i] = Long.parseLong(cin.readLine());
        }
    }
}

```

```

        System.out.println("encrypted: " +
            String.valueOf(juror.shares[i]));
        juror.shares[i] = aRSA.Decrypt(juror.shares[i]);
        System.out.println("decrypted: " +
            String.valueOf(juror.shares[i]));
    }
    System.out.println(cin.readLine()); //EA CHOOSE KEY J
/*
read in the chosen number from EA and then send cac - 1 unchosen shares to EA
*/
    juror.chosen_nr = Integer.parseInt(cin.readLine());
    System.out.println("chosen no: " +
        String.valueOf(juror.chosen_nr));
    for(int i = 0; i < cac; i++)
    {
        if(i != juror.chosen_nr)
            cout.println(String.valueOf(juror.shares[i]));
    }
}

/*
this method handles challenges from the user
*/

    public void handle_U(DataInputStream cin, PrintStream cout) throws IOException
    {
        System.out.println("user challenges this juror");
        if(juror.UserID == Integer.parseInt(cin.readLine()))
        {
/*
send p and q to user directly
*/
            cout.println("J SEND SECRET KEY U");
            System.out.println("J SEND SECRET KEY U");
            cout.println(String.valueOf(aRSA.P()));
            System.out.println(String.valueOf(aRSA.P()));
            cout.println(String.valueOf(aRSA.Q()));
            System.out.println(String.valueOf(aRSA.Q()));
        }
    }

/*
this method handles key recovery request from LEA
*/

    public void handle_LEA(DataInputStream cin, PrintStream cout)
    throws IOException
    {
        System.out.println("key recovery request from LEA");
//        if(juror.UserID == Integer.parseInt(cin.readLine()))
        {
            cout.println("J SEND SHARE LEA");
            System.out.println("J SEND SHARE LEA");
/*
send user's share to LEA

```

```

*/

        cout.println(String.valueOf(juror.shares[juror.chosen_nr]));
        System.out.println(String.valueOf(juror.shares[juror.chosen_nr]));
    }
}

/*
this method reads in an integer from the command line and adds this integer to
6000, the resulting integer is then used as the port number of this juror
*/

    public static void main(String[] args) throws IOException
    {
        Juror aj = new Juror();
        aj.port = 6000 + Integer.parseInt(args[0]);
        J aJ = new J(aj);
        aJ.J_listen();
    }

    int cac;
    RSA aRSA;
    Entity LA;
    Entity EA;
    Juror juror;
    ServerSocket listener;
}

```

12. LEA.java

```

import java.awt.*;
import java.io.*;
import java.net.*;

class LEA_InfoDialog extends Dialog
{
    public LEA_InfoDialog(LEA parent, Info u, String fD1, String fD2, String fD3)
    {
        super(parent, "Input Info", true);
        Panel p1 = new Panel();
        p1.setLayout(new GridLayout(3, 2));
        p1.add(new Label(fD1));
        p1.add(FD1 = new TextField(u.fd1, 8));
        p1.add(new Label(fD2));
        p1.add(FD2 = new TextField(u.fd2, 8));
        p1.add(new Label(fD3));
        p1.add(FD3 = new TextField(u.fd3, 8));
        add("Center", p1);

        Panel p2 = new Panel();
        p2.add(new Button("Ok"));
    }
}

```

```

        p2.add(new Button("Cancel"));
        add("South", p2);
        resize(350, 150);
    }

    public boolean action(Event evt, Object arg)
    {
        if(arg.equals("Ok"))
        {
            dispose();
            ((ResultProcessor)getParent()).processResult(this,
                new Info(FD1.getText(), FD2.getText(),FD3.getText()));
        }
        else if (arg.equals("Cancel"))
            dispose();
        else return super.action(evt, arg);
        return true;
    }

    public boolean handleEvent(Event evt)
    {
        if (evt.id == Event.WINDOW_DESTROY)
            dispose();
        else return super.handleEvent(evt);
        return true;
    }

    private TextField FD1;
    private TextField FD2;
    private TextField FD3;
}

```

```

public class LEA extends Frame implements ResultProcessor
{
    public LEA()
    {
        EA = new Entity();
        user = new User();
        fname = "LEAkey.dat";
        w = 5;
        setTitle("User Key Recovery");
        setLayout(new FlowLayout());
        add(new Button("User Information"));
        add(new Button("EA Connection Information"));
        add(new Button("Request Jury's Shares"));
        add(new Button("Recover User's Key"));
        Label st = new Label ("Status Bar");
        add(st);
    }
}

```



```

        status = new TextField("Start: enter User Information",35);
        status.setEditable(false);
        add(status);
    }

```

```

public void processResult(Dialog source, Object result)
{
    if(source instanceof LEA_InfoDialog)
    {
        Info info = (Info)result;
        if("EA".equals(info.fd1))
        {
            EA.host = info.fd2;
            EA.port = Integer.parseInt(info.fd3);
        }
        else
        {
            user.name = info.fd1;
            user.address = info.fd2;
            user.tel = Integer.parseInt(info.fd3);
        }
    }
}

```

```

public boolean handleEvent(Event evt)
{
    if(evt.id == Event.WINDOW_DESTROY) System.exit(0);
    return super.handleEvent(evt);
}

```

```

public boolean action(Event evt, Object arg)
{
try{
    if (arg.equals("User Information"))
    {
        Info in1 = new Info("name", "uow","8888");
        LEA_InfoDialog pd1 = new LEA_InfoDialog(this, in1,"name",
                                                "address","tel no");
        pd1.show();
        status.setText("next step: enter EA information");
    }
    else if(arg.equals("EA Connection Information"))
    {
        Info in2 = new Info("EA", "vivaldi","3838");
        LEA_InfoDialog pd2 = new LEA_InfoDialog(this, in2,"EA",
                                                "vivaldi","3838");
        pd2.show();
        status.setText("Trying to connect to EA...");
        EA.sock = new Socket(EA.host, EA.port);
    }
}
}

```

```

EA.dis = new DataInputStream(EA.sock.getInputStream());
EA.ps = new PrintStream(EA.sock.getOutputStream());
status.setText("Connected to EA, request jury now!!!");
EA.ps.println("LEA REQ USER EA");
//
//
EA.ps.println(user.name);
user.ID = Integer.parseInt(EA.dis.readLine());
r = Long.parseLong(EA.dis.readLine());
System.out.println(r);
}
else if(arg.equals("Request Jury's Shares"))
{
    System.out.println("Request Jury's Shares");
    status.setText("Requesting jury...");
    user.jury = new Juror[w];
    shares = new long[w];
    EA.ps.println("LEA REQ JURY EA");
    System.out.println(EA.dis.readLine());
    for(int i = 0; i < w; i++)
    {
        System.out.println("info of juror "+i);
        Juror juror = new Juror();
        user.jury[i] = juror;
        juror.name = EA.dis.readLine();
        System.out.println(juror.name);
        juror.host = EA.dis.readLine();
        System.out.println(juror.host);
        juror.port = Integer.parseInt(EA.dis.readLine());
        System.out.println(String.valueOf(juror.port));
        juror.sock = new Socket(juror.host, juror.port);
        juror.dis = new
            DataInputStream(juror.sock.getInputStream());
        juror.ps = new PrintStream(juror.sock.getOutputStream());
        juror.ps.println("LEA RECOVER J");
        juror.ps.println(String.valueOf(user.ID));
        System.out.println(juror.dis.readLine());
        shares[i] = Long.parseLong(juror.dis.readLine());
        System.out.println(String.valueOf(shares[i]));
        juror.sock.close();
    }
    EA.sock.close();
    status.setText("Recover User's Key Now!!!");
}
else if (arg.equals("Recover User's Key"))
{
    int[] pos = {1,3,5}; //can be set by inputdialog box
    long[] shadow = new long[3];
    RSA aRSA = new RSA();
    for(int i=0; i<3; i++)
    {
        shadow[i] = shares[pos[i]-1];
    }

    sk = aRSA.Recover_key(pos, shadow ,r);

```

```

        System.out.println("suspect's sk: " + String.valueOf(sk));
        status.setText("Start: enter User Information");
    }
    else return super.action(evt, arg);
} catch(IOException ioe) { }
    repaint();
    return true;
}

public static void main(String[] args)
{
    Frame f = new LEA();
    f.resize(320, 200);
    f.show();
}

    TextField status;
    Entity EA;
    User user;
    String fname;
    int w;
    long r, sk;
    long[] shares;
}

```

OUTPUT OF AN EXAMPLE SESSION

The following is the output of all entities when the user is honest.

1. Output from user client

```
User Registration Information      null vivaldi 3838
Trying to connect to EA...        Connected to EA, request jury now!!!
Request Jury                      EA SEND JKEYS U
pk of juror 0  499    164009      pk of juror 1  475    256769
pk of juror 2  823    538409      pk of juror 3  727    385487
pk of juror 4  865    338243
Challenge or Accept jury now!!!    Challenge Jury      EA SEND JINFO U
connect to juror 0  peter3 vivaldi 6003  J SEND SECRET KEY U
401  409  honesty value is: 1
connect to juror 1  peter2 vivaldi 6002  J SEND SECRET KEY U
593  433  honesty value is: 2
connect to juror 2  peter8 vivaldi 6008  J SEND SECRET KEY U
607  887  honesty value is: 3
connect to juror 3  peter5 vivaldi 6005  J SEND SECRET KEY U
397  971  honesty value is: 4
connect to juror 4  peter10 vivaldi 6010  J SEND SECRET KEY U
827  409  honesty value is: 5
EA is honest!!!
Request Jury  EA SEND JKEYS U
pk of juror 0  911    210871      pk of juror 1  167    280151
pk of juror 2  121    186503      pk of juror 3  443    437579
pk of juror 4  787    507427
Challenge or Accept jury now!!!    Challenge Jury      EA SEND JINFO U
connect to juror 0  peter2 vivaldi 6002  J SEND SECRET KEY U
433  487  honesty value is: 1
connect to juror 1  peter6 vivaldi 6006  J SEND SECRET KEY U
647  433  honesty value is: 2
connect to juror 2  peter8 vivaldi 6008  J SEND SECRET KEY U
443  421  honesty value is: 3
connect to juror 3  peter1 vivaldi 6001  J SEND SECRET KEY U
467  937  honesty value is: 4
connect to juror 4  peter4 vivaldi 6004  J SEND SECRET KEY U
557  911  honesty value is: 5
EA is honest!!!
Request Jury  EA SEND JKEYS U
pk of juror 0  221    235981      pk of juror 1  907    403127
pk of juror 2  71    235387      pk of juror 3  439    166217
pk of juror 4  655    71677
Challenge or Accept jury now!!!    Challenge Jury      EA SEND JINFO U
connect to juror 0  peter6 vivaldi 6006  J SEND SECRET KEY U
367  643  honesty value is: 1
connect to juror 1  peter7 vivaldi 6007  J SEND SECRET KEY U
673  599  honesty value is: 2
connect to juror 2  peter4 vivaldi 6004  J SEND SECRET KEY U
```

587 401 honesty value is: 3
connect to juror 3 peter5 vivaldi 6005 J SEND SECRET KEY U
463 359 honesty value is: 4
connect to juror 4 peter9 vivaldi 6009 J SEND SECRET KEY U
313 229 honesty value is: 5
EA is honest!!!

Request Jury EA SEND JKEYS U

pk of juror 0	727	398687	pk of juror 1	503	797191
pk of juror 2	247	151447	pk of juror 3	509	330007
pk of juror 4	691	130381			

Challenge or Accept jury now!!! next step: Generate Keys!!!

Generate Keys and write to file

p: 139 q: 71 n: 9869 m: 9660	public key: 169	private key: 1429	$e * d \bmod m = 1$
p: 43 q: 157 n: 6751 m: 6552	public key: 881	private key: 1889	$e * d \bmod m = 1$
p: 109 q: 157 n: 17113 m: 16848	public key: 191	private key: 13055	$e * d \bmod m = 1$
p: 131 q: 173 n: 22663 m: 22360	public key: 943	private key: 21127	$e * d \bmod m = 1$
p: 199 q: 97 n: 19303 m: 19008	public key: 181	private key: 6301	$e * d \bmod m = 1$
p: 167 q: 157 n: 26219 m: 25896	public key: 997	private key: 22909	$e * d \bmod m = 1$
p: 131 q: 173 n: 22663 m: 22360	public key: 61	private key: 19061	$e * d \bmod m = 1$
p: 167 q: 193 n: 32231 m: 31872	public key: 349	private key: 9589	$e * d \bmod m = 1$
p: 139 q: 173 n: 24047 m: 23736	public key: 401	private key: 16337	$e * d \bmod m = 1$
p: 109 q: 71 n: 7739 m: 7560	public key: 179	private key: 4139	$e * d \bmod m = 1$

Read file and Generate Shares

r = 642873293 coeff 0 = 1429 coeff 1 = 172 coeff 2 = 319
share 0 = 1920 share 1 = 3049 share 2 = 4816 share 3 = 7221 share 4 = 10264
r = 844907627 coeff 0 = 1889 coeff 1 = 847 coeff 2 = 73
share 0 = 2809 share 1 = 3875 share 2 = 5087 share 3 = 6445 share 4 = 7949
r = 664502803 coeff 0 = 13055 coeff 1 = 219 coeff 2 = 151
share 0 = 13425 share 1 = 14097 share 2 = 15071 share 3 = 16347 share 4 = 17925
r = 379350749 coeff 0 = 21127 coeff 1 = 417 coeff 2 = 300
share 0 = 21844 share 1 = 23161 share 2 = 25078 share 3 = 27595 share 4 = 30712
r = 428969557 coeff 0 = 6301 coeff 1 = 154 coeff 2 = 437
share 0 = 6892 share 1 = 8357 share 2 = 10696 share 3 = 13909 share 4 = 17996
r = 631253789 coeff 0 = 22909 coeff 1 = 521 coeff 2 = 594
share 0 = 24024 share 1 = 26327 share 2 = 29818 share 3 = 34497 share 4 = 40364
r = 973767029 coeff 0 = 19061 coeff 1 = 10 coeff 2 = 714
share 0 = 19785 share 1 = 21937 share 2 = 25517 share 3 = 30525 share 4 = 36961
r = 636357217 coeff 0 = 9589 coeff 1 = 30 coeff 2 = 956
share 0 = 10575 share 1 = 13473 share 2 = 18283 share 3 = 25005 share 4 = 33639
r = 808230767 coeff 0 = 16337 coeff 1 = 526 coeff 2 = 233
share 0 = 17096 share 1 = 18321 share 2 = 20012 share 3 = 22169 share 4 = 24792
r = 500158847 coeff 0 = 4139 coeff 1 = 458 coeff 2 = 927
share 0 = 5524 share 1 = 8763 share 2 = 13856 share 3 = 20803 share 4 = 29604

Send encrypted shares to Jury send cac public keys to EA first

pk, n, r for i = 0	169 9869 642873293	pk, n, r for i = 1	881 6751 844907627
pk, n, r for i = 2	191 17113 664502803	pk, n, r for i = 3	943 22663 379350749
pk, n, r for i = 4	181 19303 428969557	pk, n, r for i = 5	997 26219 631253789
pk, n, r for i = 6	61 22663 973767029	pk, n, r for i = 7	349 32231 636357217
pk, n, r for i = 8	401 24047 808230767	pk, n, r for i = 9	179 7739 500158847

send encrypted shares to jurors via EA

shares to juror 0

392823 359371 133254 314297 21310 382979 239508 275098 20710 345187

shares to juror 1
252777 496631 569178 788925 360335 396526 472822 251095 222202 71865
shares to juror 2
6565 26846 52447 93512 139060 51523 30440 94981 132817 120791
shares to juror 3
231785 188635 27217 80887 31882 15455 4532 299004 253339 216669
shares to juror 4
67411 119540 117940 17958 37591 16252 54236 57702 101254 94160
EA accepts key registration chosen key no: 3
pk: 943 sk: 21127 n: 22663 key escrow done!!!

/*
Content of Ukey.dat before cut-and-choose protocol
*/

169	1429	9869	881	1889	6751
191	13055	17113	943	21127	22663
181	6301	19303	997	22909	26219
61	19061	22663	349	9589	32231
401	16337	24047	179	4139	7739

/*
Content of Ukey.dat after cut-and-choose protocol
*/
943 21127 22663

/*
output from EA
*/
User request record user info name
uow 8888 0
U REQ EA
summons the jury
EA REQ J LA
EA SEND JKEYS U
LA SEND J EA
read jury's connection information
Juror 0 peter3 vivaldi 6003 peter3 vivaldi 6003
EA SUMMON J 0 JUROR SEND KEY EA 499 164009
Juror 1 peter2 vivaldi 6002 peter2 vivaldi 6002
EA SUMMON J 0 JUROR SEND KEY EA 475 256769
Juror 2 peter8 vivaldi 6008 peter8 vivaldi 6008
EA SUMMON J 0 JUROR SEND KEY EA 823 538409
Juror 3 peter5 vivaldi 6005 peter5 vivaldi 6005
EA SUMMON J 0 JUROR SEND KEY EA 727 385487
Juror 4 peter10 vivaldi 6010 peter10 vivaldi 6010
EA SUMMON J
0 JUROR SEND KEY EA 865 338243
U CHALLENGE J EA challenge of jury

EA SEND JINFO U
EA CHALLENGE J peter3 vivaldi 6003
EA CHALLENGE J peter2 vivaldi 6002
EA CHALLENGE J peter8 vivaldi 6008
EA CHALLENGE J peter5 vivaldi 6005
EA CHALLENGE J peter10 vivaldi 6010
U REQ EA summons the jury
EA REQ J LA
EA SEND JKEYS U
LA SEND J EA
read jury's connection information
Juror 0 peter2 vivaldi 6002 peter2 vivaldi 6002
EA SUMMON J 0 JUROR SEND KEY EA 911 210871
Juror 1 peter6 vivaldi 6006 peter6 vivaldi 6006
EA SUMMON J 0 JUROR SEND KEY EA 167 280151
Juror 2 peter8 vivaldi 6008 peter8 vivaldi 6008
EA SUMMON J 0 JUROR SEND KEY EA 121 186503
Juror 3 peter1 vivaldi 6001 peter1 vivaldi 6001
EA SUMMON J 0 JUROR SEND KEY EA 443 437579
Juror 4 peter4 vivaldi 6004 peter4 vivaldi 6004
EA SUMMON J 0 JUROR SEND KEY EA 787 507427
U CHALLENGE J EA challenge of jury
EA SEND JINFO U
EA CHALLENGE J peter2 vivaldi 6002
EA CHALLENGE J peter6 vivaldi 6006
EA CHALLENGE J peter8 vivaldi 6008
EA CHALLENGE J peter1 vivaldi 6001
EA CHALLENGE J peter4 vivaldi 6004
U REQ EA
summons the jury
EA REQ J LA
EA SEND JKEYS U
LA SEND J EA
read jury's connection information
Juror 0 peter6 vivaldi 6006 peter6 vivaldi 6006
EA SUMMON J 0 JUROR SEND KEY EA 221 235981
Juror 1 peter7 vivaldi 6007 peter7 vivaldi 6007
EA SUMMON J 0
JUROR SEND KEY EA 907 403127
Juror 2 peter4 vivaldi 6004 peter4 vivaldi 6004
EA SUMMON J 0
JUROR SEND KEY EA 71 235387
Juror 3 peter5 vivaldi 6005 peter5 vivaldi 6005
EA SUMMON J 0
JUROR SEND KEY EA 439 166217
Juror 4 peter9 vivaldi 6009 peter9 vivaldi 6009
EA SUMMON J 0
JUROR SEND KEY EA 655 71677
U CHALLENGE J EA challenge of jury
EA SEND JINFO U
EA CHALLENGE J peter6 vivaldi 6006

EA CHALLENGE J peter7 vivaldi 6007
 EA CHALLENGE J peter4 vivaldi 6004
 EA CHALLENGE J peter5 vivaldi 6005
 EA CHALLENGE J peter9 vivaldi 6009
 U REQ EA
 summons the jury
 EA REQ J LA
 EA SEND JKEYS U
 LA SEND J EA
 read jury's connection information
 Juror 0 peter5 vivaldi 6005 peter5 vivaldi 6005
 EA SUMMON J 0
 JUROR SEND KEY EA 727 398687
 Juror 1 peter6 vivaldi 6006 peter6 vivaldi 6006
 EA SUMMON J 0
 JUROR SEND KEY EA 503 797191
 Juror 2 peter2 vivaldi 6002 peter2 vivaldi 6002
 EA SUMMON J 0
 JUROR SEND KEY EA 247 151447
 Juror 3 peter10 vivaldi
 6010 peter10 vivaldi 6010
 EA SUMMON J 0
 JUROR SEND KEY EA 509 330007
 Juror 4 peter7 vivaldi 6007 peter7 vivaldi 6007
 EA SUMMON J 0
 JUROR SEND KEY EA 691 130381
 U SEND SHARES EA call Accept_J
 U SEND SHARES EA
 read cac pk, n and r
 user key: 0 169 9869 642873293
 user key: 1 881 6751 844907627
 user key: 2 191 17113 664502803
 user key: 3 943 22663 379350749
 user key: 4 181 19303 428969557
 user key: 5 997 26219 631253789
 user key: 6 61 22663 973767029
 user key: 7 349 32231 636357217
 user key: 8 401 24047 808230767
 user key: 9 179 7739 500158847
 read all shares from user and send all shares to jurors
 juror: 0
 encrypted share: 0 392823 encrypted share: 1 359371
 encrypted share: 2 133254 encrypted share: 3 314297
 encrypted share: 4 21310 encrypted share: 5 382979
 encrypted share: 6 239508 encrypted share: 7 275098
 encrypted share: 8 20710 encrypted share: 9 345187
 juror: 1
 encrypted share: 0 252777 encrypted share: 1 496631
 encrypted share: 2 569178 encrypted share: 3 788925
 encrypted share: 4 360335 encrypted share: 5 396526
 encrypted share: 6 472822 encrypted share: 7 251095
 encrypted share: 8 222202 encrypted share: 9 71865

juror: 2

encrypted share: 0 6565 encrypted share: 1 26846
encrypted share: 2 52447 encrypted share: 3 93512
encrypted share: 4 139060 encrypted share: 5 51523
encrypted share: 6 30440 encrypted share: 7 94981
encrypted share: 8 132817 encrypted share: 9 120791

juror: 3

encrypted share: 0 231785 encrypted share: 1 188635
encrypted share: 2 27217 encrypted share: 3 80887
encrypted share: 4 31882 encrypted share: 5 15455
encrypted share: 6 4532 encrypted share: 7 299004
encrypted share: 8 253339 encrypted share: 9 216669

juror: 4

encrypted share: 0 67411 encrypted share: 1 119540
encrypted share: 2 117940 encrypted share: 3 17958
encrypted share: 4 37591 encrypted share: 5 16252
encrypted share: 6 54236 encrypted share: 7 57702
encrypted share: 8 101254 encrypted share: 9 94160

chosen no: 3

jury send unchosen keys to EA

juror: 0

decrypted share: 0 1920 decrypted share: 1 2809
decrypted share: 2 13425
decrypted share: 3
decrypted share: 4 6892 decrypted share: 5 24024
decrypted share: 6 19785 decrypted share: 7 10575
decrypted share: 8 17096 decrypted share: 9 5524

juror: 1

decrypted share: 0 3049 decrypted share: 1 3875
decrypted share: 2 14097
decrypted share: 3
decrypted share: 4 8357 decrypted share: 5 26327
decrypted share: 6 21937 decrypted share: 7 13473
decrypted share: 8 18321 decrypted share: 9 8763

juror: 2

decrypted share: 0 4816 decrypted share: 1 5087
decrypted share: 2 15071
decrypted share: 3
decrypted share: 4 10696 decrypted share: 5 29818
decrypted share: 6 25517 decrypted share: 7 18283
decrypted share: 8 20012 decrypted share: 9 13856

juror: 3

decrypted share: 0 7221 decrypted share: 1 6445
decrypted share: 2 16347
decrypted share: 3
decrypted share: 4 13909 decrypted share: 5 34497
decrypted share: 6 30525 decrypted share: 7 25005
decrypted share: 8 22169 decrypted share: 9 20803

juror: 4 d

encrypted share: 0 10264 decrypted share: 1 7949
decrypted share: 2 17925
decrypted share: 3

decrypted share: 4 17996 decrypted share: 5 40364
decrypted share: 6 36961 decrypted share: 7 33639
decrypted share: 8 24792 decrypted share: 9 29604
verify shares are correct
recovered sk: 1429 honesty value: 1 recovered sk: 1889 honesty value: 2
recovered sk: 13055 honesty value: 3 recovered sk: 6301 honesty value: 4
recovered sk: 22909 honesty value: 5 recovered sk: 19061 honesty value: 6
recovered sk: 9589 honesty value: 7 recovered sk: 16337 honesty value: 8
recovered sk: 4139 honesty value: 9
ACCEPT KEY
LEA requests suspect's key
send r :379350749
LEA REQ JURY EA
EA SEND JURY INFO LEA
send suspect's jury
peter5 vivaldi 6005 peter6 vivaldi 6006 peter2 vivaldi 6002
peter10 vivaldi 6010 peter7 vivaldi 6007

/*
output from LA
*/
read juror file
peter1 vivaldi 6001 peter2 vivaldi 6002 peter3 vivaldi 6003
peter4 vivaldi 6004 peter5 vivaldi 6005 peter6 vivaldi 6006
peter7 vivaldi 6007 peter8 vivaldi 6008 peter9 vivaldi 6009
peter10 vivaldi 6010
EA REQ J LA
send jury to EA
peter3 vivaldi 6003 peter2 vivaldi 6002 peter8 vivaldi 6008
peter5 vivaldi 6005 peter10 vivaldi 6010
EA REQ J LA
send jury to EA
peter2 vivaldi 6002 peter6 vivaldi 6006 peter8 vivaldi 6008
peter1 vivaldi 6001 peter4 vivaldi 6004
EA REQ J LA
send jury to EA
peter6 vivaldi 6006 peter7 vivaldi 6007 peter4 vivaldi 6004
peter5 vivaldi 6005 peter9
vivaldi 6009
EA REQ J LA
send jury to EA
peter5 vivaldi 6005 peter6 vivaldi 6006 peter2 vivaldi 6002
peter10 vivaldi 6010 peter7 vivaldi 6007

/*

output from juror 1

*/

juror port: 6001

EA SUMMON J

p: 467 q: 937 n: 437579 m: 436176 public key: 443 private key: 97475

$e * d \bmod m = 1$

JUROR SEND KEY EA

443 437579

U CHALLENGE J user challenges this juror

J SEND SECRET KEY U 467 937

/*

output from juror 2

*/

juror port: 6002

EA SUMMON J

p: 593 q: 433 n: 256769 m: 255744 public key: 475 private key: 38227

$e * d \bmod m = 1$

JUROR SEND KEY EA 475 256769

U CHALLENGE J user challenges this juror

J SEND SECRET KEY U 593 433

EA SUMMON J p: 433 q: 487 n: 210871 m: 209952

public key: 911 private key: 31343 $e * d \bmod m = 1$

JUROR SEND KEY EA 911 210871

U CHALLENGE J user challenges this juror

J SEND SECRET KEY U 433 487

EA SUMMON J

p: 269 q: 563 n: 151447 m: 150616

public key: 247 private key: 131103 $e * d \bmod m = 1$

JUROR SEND KEY EA 247 151447

user accepts this juror

EA SEND SHARES J

read in cac shares from the user via EA

share 0 encrypted: 6565 decrypted: 4816

share 1 encrypted: 26846 decrypted: 5087

share 2 encrypted: 52447 decrypted: 15071

share 3 encrypted: 93512 decrypted: 25078

share 4 encrypted: 139060 decrypted: 10696

share 5 encrypted: 51523 decrypted: 29818

share 6 encrypted: 30440 decrypted: 25517

share 7 encrypted: 94981 decrypted: 18283

share 8 encrypted: 132817 decrypted: 20012

share 9 encrypted: 120791 decrypted: 13856

EA CHOOSE KEY J

chosen no: 3

LEA RECOVER J

key recovery request from LEA

J SEND SHARE LEA

25078

```
/*
output from juror 3
*/
juror port: 6003
EA SUMMON J
p: 401 q: 409 n: 164009 m: 163200 public key: 499 private key: 151099
 $e * d \bmod m = 1$ 
JUROR SEND KEY EA
499 164009
U CHALLENGE J user challenges this juror
J SEND SECRET KEY U 401 409
```

```
/*
output from juror 4
*/
juror port: 6004
EA SUMMON J
p: 557 q: 911 n: 507427 m: 505960
public key: 787 private key: 462243  $e * d \bmod m = 1$ 
JUROR SEND KEY EA 787 507427
U CHALLENGE J user challenges this juror
J SEND SECRET KEY U 557 911
EA SUMMON J
p: 587 q: 401 n: 235387 m: 234400
public key: 71 private key: 72631  $e * d \bmod m = 1$ 
JUROR SEND KEY EA 71 235387
U CHALLENGE J user challenges this juror
J SEND SECRET KEY U 587 401
```

```
/*
output from juror 5
*/
juror port: 6005
EA SUMMON J
p: 397 q: 971 n: 385487 m: 384120
public key: 727 private key: 289543  $e * d \bmod m = 1$ 
JUROR SEND KEY EA 727 385487
U CHALLENGE J user challenges this juror
J SEND SECRET KEY U 397 971
EA SUMMON J
p: 463 q: 359 n: 166217 m: 165396
public key: 439 private key: 105115  $e * d \bmod m = 1$ 
JUROR SEND KEY EA 439 166217
U CHALLENGE J user challenges this juror
J SEND SECRET KEY U 463 359
EA SUMMON J
p: 947 q: 421 n: 398687 m: 397320
public key: 727 private key: 13663  $e * d \bmod m = 1$ 
JUROR SEND KEY EA 727 398687
user accepts this juror
```

EA SEND SHARES J

read in cac shares from the user via EA

share 0 encrypted: 392823 decrypted: 1920
share 1 encrypted: 359371 decrypted: 2809
share 2 encrypted: 133254 decrypted: 13425
share 3 encrypted: 314297 decrypted: 21844
share 4 encrypted: 21310 decrypted: 6892
share 5 encrypted: 382979 decrypted: 24024
share 6 encrypted: 239508 decrypted: 19785
share 7 encrypted: 275098 decrypted: 10575
share 8 encrypted: 20710 decrypted: 17096
share 9 encrypted: 345187 decrypted: 5524

EA CHOOSE KEY J

chosen no: 3

LEA RECOVER J

key recovery request from LEA

J SEND SHARE LEA

21844

/*

output from juror 6

*/

juror port: 6006

EA SUMMON J

p: 647 q: 433 n: 280151 m: 279072

public key: 167 private key: 148727 $e * d \bmod m = 1$

JUROR SEND KEY EA 167 280151

U CHALLENGE J user challenges this juror

J SEND SECRET KEY U 647 433

EA SUMMON J

p: 367 q: 643 n: 235981 m: 234972

public key: 221 private key: 9569 $e * d \bmod m = 1$

JUROR SEND KEY EA 221 235981

U CHALLENGE J user challenges this juror

J SEND SECRET KEY U

367 643

EA SUMMON J

p: 821 q: 971 n: 797191 m: 795400

public key: 503 private key: 273567 $e * d \bmod m = 1$

JUROR SEND KEY EA 503 797191

user accepts this juror

EA SEND SHARES J

read in cac shares from the user via EA

share 0 encrypted: 252777 decrypted: 3049
share 1 encrypted: 496631 decrypted: 3875
share 2 encrypted: 569178 decrypted: 14097
share 3 encrypted: 788925 decrypted: 23161
share 4 encrypted: 360335 decrypted: 8357
share 5 encrypted: 396526 decrypted: 26327
share 6 encrypted: 472822 decrypted: 21937
share 7 encrypted: 251095 decrypted: 13473

share 8 encrypted: 222202 decrypted: 18321
share 9 encrypted: 71865 decrypted: 8763
EA CHOOSE KEY J
chosen no: 3
LEA RECOVER J
key recovery request from LEA
J SEND SHARE LEA
23161

/*

output from juror 7

*/

juror port: 6007

EA SUMMON J

p: 673 q: 599 n: 403127 m: 401856

public key: 907 private key: 387235 $e * d \bmod m = 1$

JUROR SEND KEY EA 907 403127

U CHALLENGE J user challenges this juror

J SEND SECRET KEY U 673 599

EA SUMMON J

p: 541 q: 241 n: 130381 m: 129600

public key: 691 private key: 81211 $e * d \bmod m = 1$

JUROR SEND KEY EA 691 130381

user accepts this juror

EA SEND SHARES J

read in cac shares from the user via EA

share 0 encrypted: 67411 decrypted: 10264

share 1 encrypted: 119540 decrypted: 7949

share 2 encrypted: 117940 decrypted: 17925

share 3 encrypted: 17958 decrypted: 30712

share 4 encrypted: 37591 decrypted: 17996

share 5 encrypted: 16252 decrypted: 40364

share 6 encrypted: 54236 decrypted: 36961

share 7 encrypted: 57702 decrypted: 33639

share 8 encrypted: 101254 decrypted: 24792

share 9 encrypted: 94160 decrypted: 29604

EA CHOOSE KEY J

chosen no: 3

LEA RECOVER J

key recovery request from LEA

J SEND SHARE LEA

30712

/*

output from juror 8

*/

juror port: 6008

EA SUMMON J

p: 607 q: 887 n: 538409 m: 536916

public key: 823 private key: 11743 $e * d \bmod m = 1$

JUROR SEND KEY EA 823 538409

U CHALLENGE J user challenges this juror

J SEND SECRET KEY U 607 887
EA SUMMON J
p: 443 q: 421 n: 186503 m: 185640
public key: 121 private key: 164161 $e * d \bmod m = 1$
JUROR SEND KEY EA 121 186503
U CHALLENGE J user challenges this juror
J SEND SECRET KEY U 443 421

/*
output from juror 9
*/
juror port: 6009
EA SUMMON J
p: 313 q: 229 n: 71677 m: 71136
public key: 655 private key: 37903 $e * d \bmod m = 1$
JUROR SEND KEY EA 655 71677
U CHALLENGE J user challenges this juror
J SEND SECRET KEY U 313 229

/*
output from juror 10
*/
juror port: 6010
EA SUMMON J
p: 827 q: 409 n: 338243 m: 337008
public key: 865 private key: 16753 $e * d \bmod m = 1$
JUROR SEND KEY EA 865 338243
U CHALLENGE J user challenges this juror
J SEND SECRET KEY U 827 409
EA SUMMON J
p: 331 q: 997 n: 330007 m: 328680
public key: 509 private key: 12269 $e * d \bmod m = 1$
JUROR SEND KEY EA 509 330007
user accepts this juror
EA SEND SHARES J
read in cac shares from the user via EA
share 0 encrypted: 231785 decrypted: 7221
share 1 encrypted: 188635 decrypted: 6445
share 2 encrypted: 27217 decrypted: 16347
share 3 encrypted: 80887 decrypted: 27595
share 4 encrypted: 31882 decrypted: 13909
share 5 encrypted: 15455 decrypted: 34497
share 6 encrypted: 4532 decrypted: 30525
share 7 encrypted: 299004 decrypted: 25005
share 8 encrypted: 253339 decrypted: 22169
share 9 encrypted: 216669 decrypted: 20803
EA CHOOSE KEY J
chosen no: 3

/*

output from LEA

*/

LEA RECOVER J

key recovery request from LEA

J SEND SHARE LEA 27595 379350749

Request Jury's Shares

EA SEND JURY INFO LEA

info of juror 0 peter5 vivaldi 6005 J SEND SHARE LEA 21844

info of juror 1 peter6 vivaldi 6006 J SEND SHARE LEA 23161

info of juror 2 peter2 vivaldi 6002 J SEND SHARE LEA 25078

info of juror 3 peter10 vivaldi 6010 J SEND SHARE LEA 27595

info of juror 4 peter7 vivaldi 6007 J SEND SHARE LEA 30712

suspect's sk: 21127

The following is the output of all entities when the user tries to cheat.

/*

Output from the user

*/

User Registration Information null vivaldi 3838

Trying to connect to EA... Connected to EA, request jury now!!! Request Jury

EA SEND JKEYS U

pk of juror 0 23 322061 pk of juror 1 151 710453 pk of juror 2 457 279827

pk of juror 3 947 177311 pk of juror 4 461 634877

Challenge or Accept jury now!!! next step: Generate Keys!!!

Generate Keys and write to file

p: 193 q: 181 n: 34933 m: 34560

public key: 719 private key: 33839 $e * d \bmod m = 1$

p: 181 q: 179 n: 32399 m: 32040

public key: 683 private key: 2627 $e * d \bmod m = 1$

p: 179 q: 113 n: 20227 m: 19936

public key: 949 private key: 5693 $e * d \bmod m = 1$

p: 199 q: 191 n: 38009 m: 37620

public key: 643 private key: 20887 $e * d \bmod m = 1$

p: 131 q: 137 n: 17947 m: 17680

public key: 939 private key: 659 $e * d \bmod m = 1$

p: 199 q: 131 n: 26069 m: 25740

public key: 797 private key: 6653 $e * d \bmod m = 1$

p: 179 q: 47 n: 8413 m: 8188

public key: 751 private key: 7403 $e * d \bmod m = 1$

p: 157 q: 179 n: 28103 m: 27768

public key: 197 private key: 3101 $e * d \bmod m = 1$

p: 103 q: 191 n: 19673 m: 19380

public key: 431 private key: 5171 $e * d \bmod m = 1$

p: 137 q: 157 n: 21509 m: 21216

public key: 155 private key: 6707 $e * d \bmod m = 1$

Read file and Generate Shares

r = 724948453 coeff 0 = 33839 coeff 1 = 48 coeff 2 = 312

share 0 = 34199 share 1 = 35183 share 2 = 36791 share 3 = 39023 share 4 = 41879

r = 341324441 coeff 0 = 2627 coeff 1 = 1 coeff 2 = 726

share 0 = 3354 share 1 = 5533 share 2 = 9164 share 3 = 14247 share 4 = 20782

r = 817614947 coeff 0 = 5893 coeff 1 = 703 coeff 2 = 349

share 0 = 6945 share 1 = 8695 share 2 = 11143 share 3 = 14289 share 4 = 18133

r = 649524781 coeff 0 = 20887 coeff 1 = 428 coeff 2 = 410

share 0 = 21725 share 1 = 23383 share 2 = 25861 share 3 = 29159 share 4 = 33277

r = 944068087 coeff 0 = 659 coeff 1 = 376 coeff 2 = 158

share 0 = 1193 share 1 = 2043 share 2 = 3209 share 3 = 4691 share 4 = 6489

r = 839164757 coeff 0 = 6653 coeff 1 = 923 coeff 2 = 121

share 0 = 7697 share 1 = 8983 share 2 = 10511 share 3 = 12281 share 4 = 14293

r = 998202529 coeff 0 = 7403 coeff 1 = 486 coeff 2 = 319

share 0 = 8208 share 1 = 9651 share 2 = 11732 share 3 = 14451 share 4 = 17808

r = 767927183 coeff 0 = 3101 coeff 1 = 442 coeff 2 = 474

share 0 = 4017 share 1 = 5881 share 2 = 8693 share 3 = 12453 share 4 = 17161

```

r = 581973499  coeff 0 = 5171  coeff 1 = 489  coeff 2 = 380
share 0 = 6040  share 1 = 7669  share 2 = 10058  share 3 = 13207  share 4 = 17116
r = 824250523  coeff 0 = 6707  coeff 1 = 483  coeff 2 = 569
share 0 = 7759  share 1 = 9949  share 2 = 13277  share 3 = 17743  share 4 = 23347
Send encrypted shares to Jury
send cac public keys to EA first
pk, n, r for i = 0  719  34933  724948453
pk, n, r for i = 1  683  32399  341324441
pk, n, r for i = 2  949  20227  817614947
pk, n, r for i = 3  643  38009  649524781
pk, n, r for i = 4  939  17947  944068087
pk, n, r for i = 5  797  26069  839164757
pk, n, r for i = 6  751  8413  998202529
pk, n, r for i = 7  197  28103  767927183
pk, n, r for i = 8  431  19673  581973499
pk, n, r for i = 9  155  21509  824250523
send encrypted shares to jurors via EA
shares to juror 0
275095 238011 40948 84600 278641
16555 23254 179358 38512 202310
shares to juror 1
366300 547319 174025 63640 306329
164921 51346 349809 617423 579472
shares to juror 2
36147 147225 269222 277293 260989
164241 238431 60048 203999 231086
shares to juror 3
55233 117913 116431 171653 158394
36519 3728 93513 128136 66940
shares to juror 4
133036 81613 416270 273670 606899
539352 599243 540041 308054 464585
You are caught cheating!!!

```

```

/*

```

```

key pairs in Ukey.dat before tampering

```

```

*/

```

```

719    33839    34933
683    2627     32399
949    5693     20227
643    20887    38009
939    659      17947
797    6653     26069
751    7403     8413
197    3101     28103
431    5171     19673
155    6707     21509

```

```
/*  
key pairs in Ukey.dat after tampering  
*/
```

719	33839	34933
683	2627	32399
949	5893	20227 /*tampered secret key*/
643	20887	38009
939	659	17947
797	6653	26069
751	7403	8413
197	3101	28103
431	5171	19673
155	6707	21509

```
/*  
output from EA  
*/
```

```
User request record user info name uow 8888 0  
U REQ EA summons the jury  
EA REQ J LA  
EA SEND JKEYS U  
LA SEND J EA  
read jury's connection information  
Juror 0 peter1 vivaldi 6001 peter1 vivaldi 6001  
EA SUMMON J 0 JUROR SEND KEY EA 23 322061  
Juror 1 peter2 vivaldi 6002 peter2 vivaldi 6002  
EA SUMMON J 0 JUROR SEND KEY EA 151 710453  
Juror 2 peter5 vivaldi 6005 peter5 vivaldi 6005  
EA SUMMON J 0 JUROR SEND KEY EA 457 279827  
Juror 3 peter8 vivaldi 6008 peter8 vivaldi 6008  
EA SUMMON J 0 JUROR SEND KEY EA 947 177311  
Juror 4 peter6 vivaldi 6006 peter6 vivaldi 6006  
EA SUMMON J 0 JUROR SEND KEY EA 461 634877  
U SEND SHARES EA call Accept_J  
U SEND SHARES EA  
read cac pk, n and r  
user key: 0 719 34933 724948453  
user key: 1 683 32399 341324441  
user key: 2 949 20227 817614947  
user key: 3 643 38009 649524781  
user key: 4 939 17947 944068087  
user key: 5 797 26069 839164757  
user key: 6 751 8413 998202529  
user key: 7 197 28103 767927183  
user key: 8 431 19673 581973499  
user key: 9 155 21509 824250523  
read all shares from user and send all shares to jurors  
juror: 0  
encrypted share: 0 275095 encrypted share: 1 238011  
encrypted share: 2 40948 encrypted share: 3 84600
```

encrypted share: 4 278641 encrypted share: 5 16555
encrypted share: 6 23254 encrypted share: 7 179358
encrypted share: 8 38512 encrypted share: 9 202310

juror: 1

encrypted share: 0 366300 encrypted share: 1 547319
encrypted share: 2 174025 encrypted share: 3 63640
encrypted share: 4 306329 encrypted share: 5 164921
encrypted share: 6 51346 encrypted share: 7 349809
encrypted share: 8 617423 encrypted share: 9 579472

juror: 2

encrypted share: 0 36147 encrypted share: 1 147225
encrypted share: 2 269222 encrypted share: 3 277293
encrypted share: 4 260989 encrypted share: 5 164241
encrypted share: 6 238431 encrypted share: 7 60048
encrypted share: 8 203999 encrypted share: 9 231086

juror: 3

encrypted share: 0 55233 encrypted share: 1 117913
encrypted share: 2 116431 encrypted share: 3 171653
encrypted share: 4 158394 encrypted share: 5 36519
encrypted share: 6 3728 encrypted share: 7 93513
encrypted share: 8 128136 encrypted share: 9 66940

juror: 4

encrypted share: 0 133036 encrypted share: 1 81613
encrypted share: 2 416270 encrypted share: 3 273670
encrypted share: 4 606899 encrypted share: 5 539352
encrypted share: 6 599243 encrypted share: 7 540041
encrypted share: 8 308054 encrypted share: 9 464585

chosen no: 3

jury send unchosen keys to EA

juror: 0

decrypted share: 0 34199 decrypted share: 1 3354 decrypted share: 2 6945
decrypted share: 3
decrypted share: 4 1193 decrypted share: 5 7697 decrypted share: 6 8208
decrypted share: 7 4017 decrypted share: 8 6040 decrypted share: 9 7759

juror: 1

decrypted share: 0 35183 decrypted share: 1 5533 decrypted share: 2 8695
decrypted share: 3
decrypted share: 4 2043 decrypted share: 5 8983 decrypted share: 6 9651
decrypted share: 7 5881 decrypted share: 8 7669 decrypted share: 9 9949

juror: 2

decrypted share: 0 36791 decrypted share: 1 9164 decrypted share: 2 11143
decrypted share: 3
decrypted share: 4 3209 decrypted share: 5 10511 decrypted share: 6 11732
decrypted share: 7 8693 decrypted share: 8 10058 decrypted share: 9 13277

juror: 3

decrypted share: 0 39023 decrypted share: 1 14247 decrypted share: 2 14289
decrypted share: 3
decrypted share: 4 4691 decrypted share: 5 12281 decrypted share: 6 14451
decrypted share: 7 12453 decrypted share: 8 13207 decrypted share: 9 17743

juror: 4

decrypted share: 0 41879 decrypted share: 1 20782 decrypted share: 2 18133
decrypted share: 3

decrypted share: 4 6489 decrypted share: 5 14293 decrypted share: 6 17808
decrypted share: 7 17161 decrypted share: 8 17116 decrypted share: 9 23347
verify shares are correct
recovered sk: 33839 honesty value: 1
recovered sk: 2627 honesty value: 2
recovered sk: 5893 honesty value: 2
recovered sk: 659 honesty value: 3
recovered sk: 6653 honesty value: 4
recovered sk: 7403 honesty value: 5
recovered sk: 3101 honesty value: 6
recovered sk: 5171 honesty value: 7
recovered sk: 6707 honesty value: 8
not ACCEPT KEY

/*
output from LA
*/

EA REQ J LA
send jury to EA
peter1 vivaldi 6001
peter2 vivaldi 6002
peter5 vivaldi 6005
peter8 vivaldi 6008
peter6 vivaldi 6006

juror port: 6001
EA SUMMON J
p: 443 q: 727 n: 322061 m: 320892
public key: 23 private key: 83711 $e * d \bmod m = 1$
JUROR SEND KEY EA 23 322061
user accepts this juror
EA SEND SHARES J
read in cac shares from the user via EA
share 0 encrypted: 275095 decrypted: 34199
share 1 encrypted: 238011 decrypted: 3354
share 2 encrypted: 40948 decrypted: 6945
share 3 encrypted: 84600 decrypted: 21725
share 4 encrypted: 278641 decrypted: 1193
share 5 encrypted: 16555 decrypted: 7697
share 6 encrypted: 23254 decrypted: 8208
share 7 encrypted: 179358 decrypted: 4017
share 8 encrypted: 38512 decrypted: 6040
share 9 encrypted: 202310 decrypted: 7759
EA CHOOSE KEY J
chosen no: 3

juror port: 6002
EA SUMMON J
p: 857 q: 829 n: 710453 m: 708768
public key: 151 private key: 572647 $e * d \bmod m = 1$

JUROR SEND KEY EA 151 710453
user accepts this juror
EA SEND SHARES J
read in cac shares from the user via EA
share 0 encrypted: 366300 decrypted: 35183
share 1 encrypted: 547319 decrypted: 5533
share 2 encrypted: 174025 decrypted: 8695
share 3 encrypted: 63640 decrypted: 23383
share 4 encrypted: 306329 decrypted: 2043
share 5 encrypted: 164921 decrypted: 8983
share 6 encrypted: 51346 decrypted: 9651
share 7 encrypted: 349809 decrypted: 5881
share 8 encrypted: 617423 decrypted: 7669
share 9 encrypted: 579472 decrypted: 9949
EA CHOOSE KEY J
chosen no: 3

juror port: 6003

juror port: 6004

juror port: 6005
EA SUMMON J
p: 461 q: 607 n: 279827 m: 278760
public key: 457 private key: 195193 $e * d \bmod m = 1$
JUROR SEND KEY EA 457 279827
user accepts this juror
EA SEND SHARES J
read in cac shares from the user via EA
share 0 encrypted: 36147 decrypted: 36791
share 1 encrypted: 147225 decrypted: 9164
share 2 encrypted: 269222 decrypted: 11143
share 3 encrypted: 277293 decrypted: 25861
share 4 encrypted: 260989 decrypted: 3209
share 5 encrypted: 164241 decrypted: 10511
share 6 encrypted: 238431 decrypted: 11732
share 7 encrypted: 60048 decrypted: 8693
share 8 encrypted: 203999 decrypted: 10058
share 9 encrypted: 231086 decrypted: 13277
EA CHOOSE KEY J
chosen no: 3

juror port: 6006
EA SUMMON J
p: 883 q: 719 n: 634877 m: 633276
public key: 461 private key: 417605 $e * d \bmod m = 1$
JUROR SEND KEY EA 461 634877
user accepts this juror
EA SEND SHARES J
read in cac shares from the user via EA
share 0 encrypted: 133036 decrypted: 41879
share 1 encrypted: 81613 decrypted: 20782

share 2 encrypted: 416270 decrypted: 18133
share 3 encrypted: 273670 decrypted: 33277
share 4 encrypted: 606899 decrypted: 6489
share 5 encrypted: 539352 decrypted: 14293
share 6 encrypted: 599243 decrypted: 17808
share 7 encrypted: 540041 decrypted: 17161
share 8 encrypted: 308054 decrypted: 17116
share 9 encrypted: 464585 decrypted: 23347
EA CHOOSE KEY J
chosen no: 3

juror port: 6007

juror port: 6008

EA SUMMON J

p: 281 q: 631 n: 177311 m: 176400

public key: 947 private key: 118283 $e * d \bmod m = 1$

JUROR SEND KEY EA 947 177311

user accepts this juror

EA SEND SHARES J

read in cac shares from the user via EA

share 0 encrypted: 55233 decrypted: 39023
share 1 encrypted: 117913 decrypted: 14247
share 2 encrypted: 116431 decrypted: 14289
share 3 encrypted: 171653 decrypted: 29159
share 4 encrypted: 158394 decrypted: 4691
share 5 encrypted: 36519 decrypted: 12281
share 6 encrypted: 3728 decrypted: 14451
share 7 encrypted: 93513 decrypted: 12453
share 8 encrypted: 128136 decrypted: 13207
share 9 encrypted: 66940 decrypted: 17743
EA CHOOSE KEY J

chosen no: 3

juror port: 6009

juror port: 6010

Bibliography

- [1] M. Bellare and S. Goldwasser. “Encapsulated key escrow”. <http://www-cse.ucsd.edu/users/mihir/papers/crypto-papers.html>, April 1996.
- [2] M. Bellare and S. Goldwasser. “Verifiable partial key escrow”. *Proc. 4th ACM Conference on Computer and Communications Security*, April 1997.
- [3] M. Bellare and R. L. Rivest. “Translucent Cryptography”. <http://www-cse.ucsd.edu/users/mihir/papers/crypto-papers.html>, Feb 1996.
- [4] William Blackstone. *Commentaries*, vol iii. page 379.
- [5] M. Blaze. “Protocol failure in the escrowed encryption standard”. *Proceedings of the 2nd ACM Conference on Computer and Communication*, pages 59–67, 1994.
- [6] M. Blaze. “Oblivious Key Escrow”. *Workshop on Information Hiding, published in Springer-Verlag, LNCS Vol 1174*, pages 335–343, 1996.
- [7] William J. Caelli. “Commercial Key Escrow: An Australian Perspective”. *Proceedings of Cryptography Policy and Algorithms Conference*, pages 41–64, 3-5 July 1995.
- [8] Magna Carta. 1215.
- [9] W.R. Cornish. “The Jury”. *Allen Lane The Penguin Press London*, 1968.
- [10] Dorothy E. Denning. “Taxonomy for key escrow encryption systems”. *Communication of the ACM*, pages 35–40, 3/1996.
- [11] Dorothy E. Denning. “Key escrowing today”. *IEEE communications magazine*, pages 58–67, 9/1994.

- [12] Dorothy E. Denning. "The Future of Cryptography". <http://guru.cosc.georgetown.edu/~denning/crypto/Future.html>, January 6, 1996.
- [13] Dorothy E. Denning. "My position on encryption policy". <http://guru.cosc.georgetown.edu/~denning/crypto/position.html>, June 6, 1997.
- [14] Dorothy E. Denning et al. "Codes, Keys and Conflicts: Issues in U.S. Crypto Policy". http://Info.acm.org/REPORTS/ACM_CRYPTOSTUDY/WEB/contents.html, June 1994.
- [15] Lord Devlin. Trial by jury. *Steven and Son Limited*, page 164, 1966.
- [16] Lord Devlin. The judge. *OUP. Oxford*, page 127, 1981.
- [17] Lord Devlin. *The Times*, 20 November 1978.
- [18] Sir Patrick Devlin. "Trial by Jury". *Steven and Son Limited*, 1966.
- [19] Henk van Tilborg Eric Verheul. "Binding the ElGamal Encrytion Scheme". <http://cwis.kub.nl/~frw/people/koops/bindtech.htm>, 1996.
- [20] Henk van Tilborg Eric Verheul, Bert-Jaap. "Binding ElGamal: A Fraud-Detectable Alternative". *Advances in Cryptology - EUROCRYPT 97, LNCS 1233*, 1997.
- [21] Yongfei Han Feng Bao, Robert H Deng and Albert Jeng. "Design and analyses of two basic protocols for use in TTP-based key escrow manuscript". *Second Australasian Conference on Information Security and Privacy*, pages p. 261 – 270, 1997.
- [22] Helen Fenwick. Civil liberties. *Cavendish Publishing Ltd*, pages 270–271, 1994.
- [23] National Institute for Standards and Technology. "Escrowed Encryption Standard". *Federal Information Processing Standards Publication 185*, U.S.Dept. of Commerce, 1994.
- [24] Yair Frankel and Moti Yung. "Escrow Encryption Systems Visited: Attacks, Analysis and Designs". *Advances in Cryptology, Proceedings of CRYPTO 95, LNCS*, pages 113–138, 1995.

- [25] Michael Froomkin. "The Metaphor is the Key: Cryptography, The Clipper Chip, and the Constitution". *U. Penn. Law Review* vol 143, pages Section 2 of chapter III, p. 709. 1995.
- [26] Michael Froomkin. "It Came From Planet Clipper: The Battle Over Cryptographic Key Escrow". *Law of Cyberspace issue of the University of Chicago Legal Forum*. pages Section C, Introduction, p.15, Nov. 12, 1996.
- [27] J. Kilian and T. Leighton. "Fair cryptosystems revisited". *Advances in Cryptology, Proceedings of CRYPTO 95, LNCS*, pages 113–138, 1995.
- [28] S. Macali. "Fair public key cryptosystems". *Advances in Cryptology, Proceedings of CRYPTO 92, LNCS*. pages 113–138, 1992.
- [29] Tim May. "Crypto Anarchy and Virtual Communities". *Internet Security*, pages 4–12, Aril 1992.
- [30] Ross Anderson Micheal Roe. "The GCHQ Protocol and its Problems". [http ://www.cl.cam.ac.uk/~rja14/](http://www.cl.cam.ac.uk/~rja14/) to appear in *Eurocrypt97*, 8th June 1996.
- [31] M. Walker N. Jefferies, C. Mitchell. "A Proposed Architecture for Trusted Third Party Services". *Proceedings of Cryptography Policy and Algorithms Conference*, pages 67–81, 3-5 July 1995.
- [32] National Institute of Science and Technology. "Enabling Privacy, Commerce, Security and Public Safety in the Global Information Infrastructure". <http://www.eff.org/pub/Privacy/Key-escrow/Clipper-III/HTML/960520-nist-clipper3-paper.draft.html>, May 20, 1996.
- [33] Office of the Press Secretary. "Statement by the press secretary". [http ://csrc.nist.gov/key_recovery/clipper/txt](http://csrc.nist.gov/key_recovery/clipper/txt), April 16, 1993.
- [34] George Orwell. "1984". *Penguin Books*, pages 6–7, 1954.
- [35] M. O. Rabin. "Digital Signatures". *Foundations of Secure Communication*, pages 155–168. 1978.
- [36] M. O. Rabin. "Digitized Signatures and PK Functions as Intractable as Factorization". *MIT Technical Report TR-212*, 1979.

- [37] President Franklin Delano Roosevelt. *Fireside Chat*, April 14, 1938.
- [38] Adi Shamir. "How to share a secret". *Communication of the ACM*, pages 612–613, 11/1979.
- [39] G. J. Simmons. "The prisoners' problem and the subliminal channel". *Advances in Cryptology. Proceedings of CRYPTO 83, LNCS*, pages 51–67, 1984.
- [40] G. J. Simmons. "How to (really) share a secret". *Advances in Cryptology, Proceedings of CRYPTO 88. LNCS*, pages 390–448, 1988.
- [41] Raymond Wacks. Privacy, volume i. *Dartmouth Publishing Company Ltd*, pages xii–xiii. 1993.