Managing Secured Documents Over Long Periods

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

Bachelor of Technology
in
Computer Science & Engineering
by

Gorav Jindal (2004CS10163)
Vikrant Kumar (2004CS10223)

Under the guidance of

Prof. B.N. Jain
Dr. Vinay Joseph Ribeiro

Department of Computer Science & Engineering
Indian Institute of Technology, Delhi

May 2008

Certificate

This is to certify that the thesis entitled **Managing Secured Documents Over Long Periods** submitted by **Gorav Jindal** and **Vikrant Kumar** to the Indian Institute of Technology, Delhi for the award of the degree of Bachelor of Technology in Computer Science and Engineering, is a bonafide record of work carried out by them under our joint supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

May 12, 2008.
IIT Delhi.

Vinay Joseph Ribeiro
Assistant Professor
Dept. of Comp. Science and Engg.
IIT Delhi.

B.N. Jain
Professor
Dept. of Comp. Science and Engg.
IIT Delhi.

Acknowledgements

We express our deepest gratitude towards Prof. B.N. Jain and Dr. Vinay Ribeiro, our project supervisors, who have been a tremendous source of inspiration and support throughout the project. We are also very thankful to Mr. Saurabh Kaura and Ms. Binda from NISG who have guided us with their knowledge in management and law.

Abstract

Certificates and other legal documents play a very important role in our lives. However, currently there exists no such framework within which we can construct systems that can handle digitally secured documents over a period of say a couple of decades. This project is an attempt at analyzing the situation and then providing a suitable solution to the aforementioned problem. In this paper we present the high level initial design with significant details of implementation of the system which can be used to address the problem. Our design and implementation is focused around creating an online degree certificate maintenance system for IIT Delhi. We will more specifically focus on on-fly generation of degree document and its verification on client side.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Objective and Need

The documents pervade everybody's life, consuming a substantial amount of effort that is spent to manage them. A natural solution to this problem would be a system that manages documents automatically. This paper analyzes the requirements and describes a system designed for retaining records and ensuring their legibility, interpretability, availability, and provable authenticity over long periods of time. In general, information preservation is accomplished not by any one single technique, but by avoiding all of the many possible events that might cause loss. The focus of the system is on preservation in the 10 to 100 year time span-a long enough period such that many difficult problems are known and can be addressed, but not unimaginable in terms of the longevity of computer systems and technology. The general approach focuses on eliminating single points of failure - single elements whose failure would cause information loss - combined with active detection and repair in the event of failure. Any solution however should conform to the specifications that come with a manual system like authenticating the source of the document, non repudiation etc. It is here that advanced cryptographic techniques developed by scientific community over the last 30 years come handy. The solution that we present in this paper is based on many of such techniques.

To conceptualize a system that would act as a dual to all the existing document management systems would be a humongous task and too difficult to start with, hence the focus of the work done so far has been to develop a scalable system that can be used to handle degree certificates, issued to the students by a university, such as IIT Delhi. The following principles have been used as guideline for preparing the design.

- The **integrity** of degree certificate document is of supreme importance
- The system should be able to **authenticate** the proper right of the office holders

- The **confidentiality** of the document is also important but less than the other two

In what follows we systematically introduce the subject first and then describe the design and implementaion details of the system that can be used to manage degree documents. This is followed by an analysis of the issues that could arise over time and possible ways to handle them.

1.2 Basic Approach

The approach we took to solve the above mentioned problem was a very usual one. First we started with comprehensive study of usual stuff that is pre-requisite to build any security related application. We started with study of famous public key cryptography algorithmic schemes like RSA and ECC and also studied DSA and ECDSA briefly. We studied the concept of digital signatures as they are essential for building any integrity preserving application. For digital signatures, we went through some of the popular hash algorithms like MD5, SHA-family which are pre-dominantly used in digital signatures. We went through some popular papers that tried to address the issue of long term security of documents by above mentioned techniques. Then we tried to judge the performance of above competing technologies on the basis of their efficiency. Finally we decided to go with RSA for encryption and SHA-1 for hashing as they are the most dominant techniques used for digital signatures in commercial and academic applications, so there will not be any compatibility issues. Then we started to work on the basic design of the system that can be used to address the problem of long term security and specifically a degree management system for IIT Delhi. First we suggested that we can keep every degree in single file and sign it. There were a lot of issues with this design as will be explained in coming chapters. Then we proposed the idea of a single master document for managing degrees of a single year. This idea was much better than the first one, as we will see in coming chapters. After nearly finalizing this 2nd design, we went for implementation of the prototype for IITD degree management system. We have tried to explain all the relevant details in coming sections.

1.3 Brief Outline

If we analyze the main problems which can occur in implementation of such a system, they can be

- The **Expiration** of private keys of signing authorities
- **Document Format** becomes obsolete after certain time.

The obvious solution to these problems can be re-signing and keeping Document in text. But there are lot of issues in the process of re-signing and we will see that re-signing alone is not the sure shot solution of our problem, there are many issues in the process of re-signing itself. In what follows we will explain these solutions in detail along with background study needed to understand these solutions fully.

Chapter 2

Background

The document in paper format has many interesting and self securing properties since any attempt to compromise the integrity of the document will either be self evident or easily verifiable, digital documents however do not enjoy the same luxury. Even the millionth copy of a digital document would be exactly similar to the initial one. This is where beautiful mathematical techniques developed under the aegis of cryptography come to our rescue. Since any document security project must involve one of these techniques we find it useful to give a brief overview of the tool and techniques involved in cryptography which we present in the following subsection. The information presented here is not in the least sense comprehensive and it is highly recommended that one should go through reference 1 to fully appreciate the subject matter. However for the purpose of understanding this document the information given below should suffice.

2.1 Public Key Cryptography

Public key cryptography is a form of cryptography in which encryption and decryption are performed using the different keysone a public key and one a private key. Unlike symmetric key encryption, it is free from the problem of key distribution and key sharing. Each user of public key cryptosystem has two keys(public and private key). Public key is known to all the other users of public key cryptosystem(can be verified by contacting a trusted third party called Certificate Authority) while private key is known only to corresponding user. As it will be explained in coming sections, calculation of private key from known public key is a very difficult problem and strength of any public key cryptosystem directly depends entirely upon level of difficulty of this problem.

2.2 RSA

RSA is the most popular algorithm used in public key cryptography. It was the first algorithm known to be suitable for signing as well as encryption. It's difficulty depends upon factoring a large number into 2 primes. Following are the steps for RSA.

1. Choose p, q, two prime numbers. (private)
2. Calculate $n = pq$. (public)
3. Choose an integer $e(1 < e < \phi(n))$ such that $(e, \phi(n)) = 1$. (public)
4. Calculate $d \equiv e^{-1}(mod\ \phi(n))$

Here pair e, n is the public key of user known to all and d, n is the private key known only to the user. Now if a message has to be encrypted, first it is represented as a number less than n. Let it be M. Then $C \equiv M^e(mod\ n)$ is the number corresponding to cipher. For decryption we perfrom $M' \equiv C^d(mod\ n)$. As it can be seen, $M' \equiv (M^e)^d(mod\ n)$, and $ed \equiv 1(mod\ \phi(n))$ meaning that $ed = 1 + k\phi(n)$ for some k. So $M' \equiv M^{1+k\phi(n)}(mod\ n)$. As by euler's theorm $M^{\phi(n)} \equiv 1(mod\ n)$. We get $M' \equiv M(mod\ n)$. So this way we can recover the original message. Note that, a message here can also be encrypted with private key and decrypted using public key (Used in digital signtaures).

2.2.1 Breaking RSA

Suppose an adversary A wants to get the private key of a user U, what should he/she do?. Everyone knows the public key of U i.e. pair e, n. Private key of U is pair d, n, now to know that private key of U, A has to somehow calculate d from knowledge of e and n. As we defined above $d \equiv e^{-1}(mod\ \phi(n))$. Inverse calculation mod $\phi(n)$ is not difficult (we can use extended euclidean algorithm), problem lies in calcualting $\phi(n)$ itself from n. For calcualting $\phi(n)$, A has to know factors p, q of n. So calculating private key is as difficult as factoring a number. As of know there are not any efficeint algorithms for factoring. Thats why a RSA private key is safe until a fast factoring algorithm is found or there are enough computing resources to factor a number by brute force.

2.3 Elliptic Curve Cryptography

In elliptic curve cryptography, we operate on points of a elliptic curve. Here we will explain elliptic curves only on field Z_p, where p is a prine number. Every elliptic curve

cryptosystem has a curve of the follwing form

$$y^2 mod\ p = (x^3 + ax + b) mod\ p \tag{2.1}$$

$E_p(a, b)$ is the set of all points (x, y) where x , y belong to Z_p. We operate on all points of $E_p(a, b)$. Follwing equations define, how to perform addition on two points $P = (x_p, y_p)$ and $Q = (x_q, y_q)$ to a point $R = (x_r, y_r)$. And also, how to calculate $R = 2P$ from P.

$$\Delta = \left(\frac{y_q - y_p}{x_q - x_p}\right) (mod\ p) \tag{2.2}$$

$$x_r = \Delta^2 - x_p - x_q (mod\ p) \tag{2.3}$$

$$y_r = -y_p + \Delta(x_p - x_r) (mod\ p) \tag{2.4}$$

And for calculating $2P$ from P

$$x_r = \left(\frac{3x_p^2 + a}{2y_p}\right)^2 - 2x_p (mod\ p) \tag{2.5}$$

$$y_r = \left(\frac{3x_p^2 + a}{2y_p}\right)^2 (x_p - x_r) - y_p (mod\ p) \tag{2.6}$$

Now for any elliptic curve cryptosystem, there are some parameters called domain parameters represented as (q, FR, a, b, G, n, h) where G is some point of feeliptic curve and n is its order. FR is the feild representation. Every user has a private key $n_{pr} < n$ amd corresponding public key is $n_{pr}G$. Message is represented as a point on elliptic curve(P_m) and then follwing encryption and decryption steps are performed if A wants to send P_m in encrypted form. P_B is public key of B and n_B is private key of B.

$$C_m = \{kG, P_m + kP_B\} \tag{2.7}$$

B receives a pair of points P_1, P_2. To decrypt, he/she does following

$$P_{m'} = P_2 - n_B P_1 = P_m + kn_B G - n_B kG = P_m \tag{2.8}$$

2.4 Hash Functions

A hash function is mathematical function or method to map data into a small domain that is used as fingerprint of data.These fingerprints are called digest or hash values. This fingerprints are coincise represention of larger message(data).Hash function are widely used to check message integrity and during digital signature. MD-5,SHA-1,SHA-224,SHA-

256 and SHA-512 are some most commonly used hash function in applications and various standards.

A cryptographic hash function should work as a random function while still being deterministic and computationally feasable. As hash function maps messages to a smaller domain, then it is possible that two message have same digest.this situation is called collison. A cryptographic hash function is said to be insecure

- If given a digest,one can easily determine the message that have same digest.
- If hash function have high collison rate, if probability that two message have same digest is high.

Cryptographic hash properties

- Given h it should be hard to find any m such that $h = hash(m)$
- It should be hard to find any 2 input $m1$ and $m2$ such that $hash(m1) = hash(m2)$
- Given an input $m1$, it should be hard to find another input, $m2$ (not equal to $m1$) such that $hash(m1) = hash(m2)$

Cryptographic hash function are very sensitive to any change in message. Any small change in message changes the message digest completely.

2.5 Digital Signatures

Digital signatures are used to provide authentication and integrity of a message. Following are the steps for creating and verifying digital signatures.

1. Hash the message using any standrad hash function.
2. Encrypt the hash using signing person's private key, send the message with encrypted hash to receiver.
3. At verifier end, compute hash of the message, decrypt the encrypted hash using sender's public key and match it with computed hash.

Here we explain how to ECC for digital signatures by a algorithm called ECDSA. Following are the steps for ECDSA.

1. Caculate hash of message m, let it be e.
2. Calculate $(x, y) = kG$, where k is a random integer with $0 < k < n - 1$.

3. Calculate $r \equiv x(mod\ n)$, if $r = 0$, then go to step 2.

4. Calcualte $s \equiv k^{-1}(e + rd_A)(mod\ n)$. if $s = 0$, then go to step 2.

5. Signature is (r, s).

At the verifier's end, following steps are performed

1. Caculate hash of message m, let it be e.

2. Calculate $w = s^{-1}(mos\ n)$.

3. Calcualte $u_1 = ew(mod\ n)$.

4. Calculate $u_2 = rw(mod\ n)$.

5. Calculate $(x', y') = u_1G + u_2Q_A$.

6. Verify $r = x'(mod\ n)$.

2.6 Performance Analysis of various Technologies

We now present the results of the test runs for RSA, ECC and SHA respectively. We used standard libraries for the algorithms and the code was run on Intel-DualCore(2.0 GHz) machines with 2Gb RAM

2.6.1 RSA

The RSA algorithm was found to have almost linear variation in time with file size during both encryption and decryption. The results of the experiments are given below. The library used was GMP. Interestingly decryption time was found to be greater than the decryption time

2.6.2 ECC

The ECC algorithm was also found to have almost linear variation in time with file size during both encryption and decryption. The results of the experiments are given below. The library used was openSSL. Encryption time was found to be greater than the decryption time as in case of RSA.

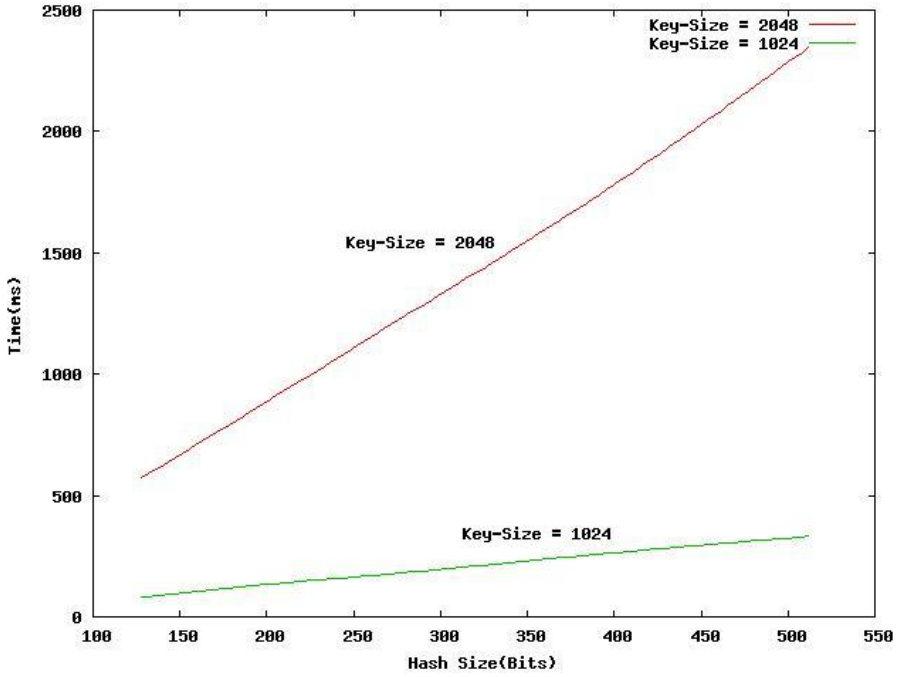

Figure 2.1: RSA Encryption performance

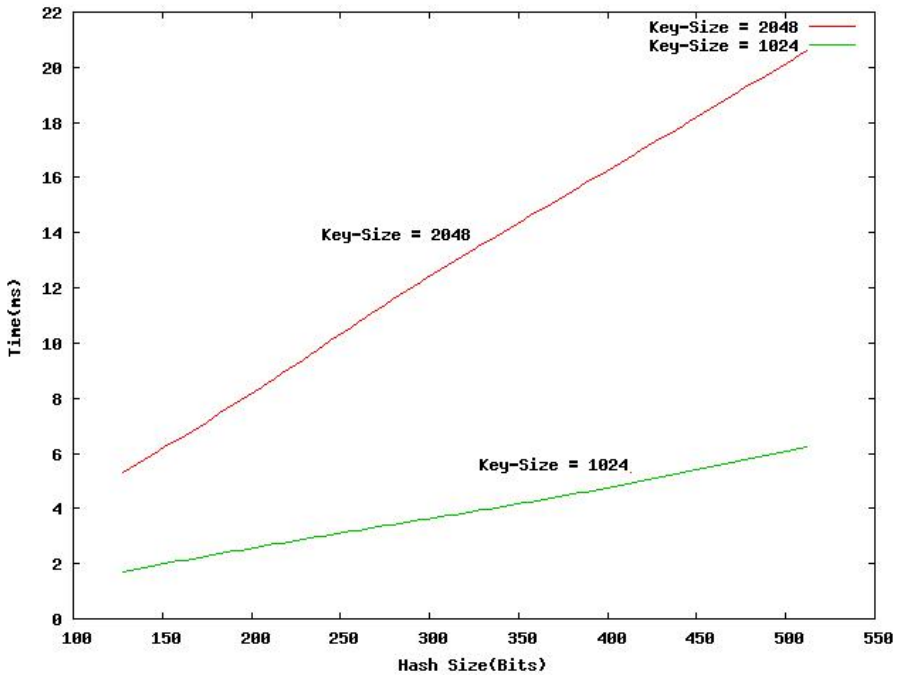

Figure 2.2: RSA Decryption performance

2.6.3 Hash Functions(SHA)

Hash functions were coded and analyzed the performance of SHA-2 functions viz. SHA224, SHA256, SHA384, SHA512. The performance of SHA224 was found to be nearly same as that of SHA256, similar was the case with SHA384 and SHA512.

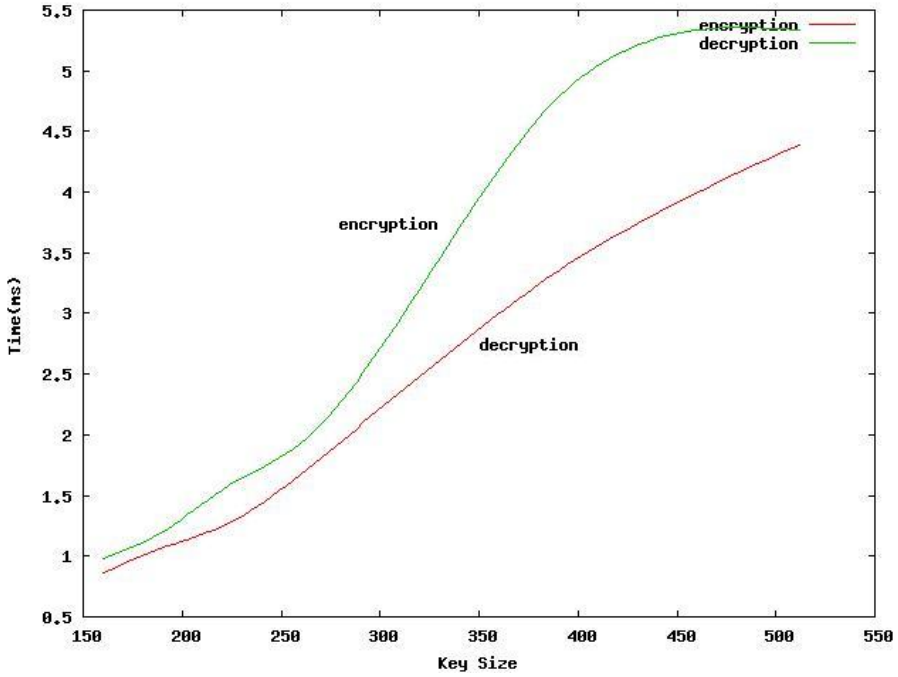

Figure 2.3: ECC performance

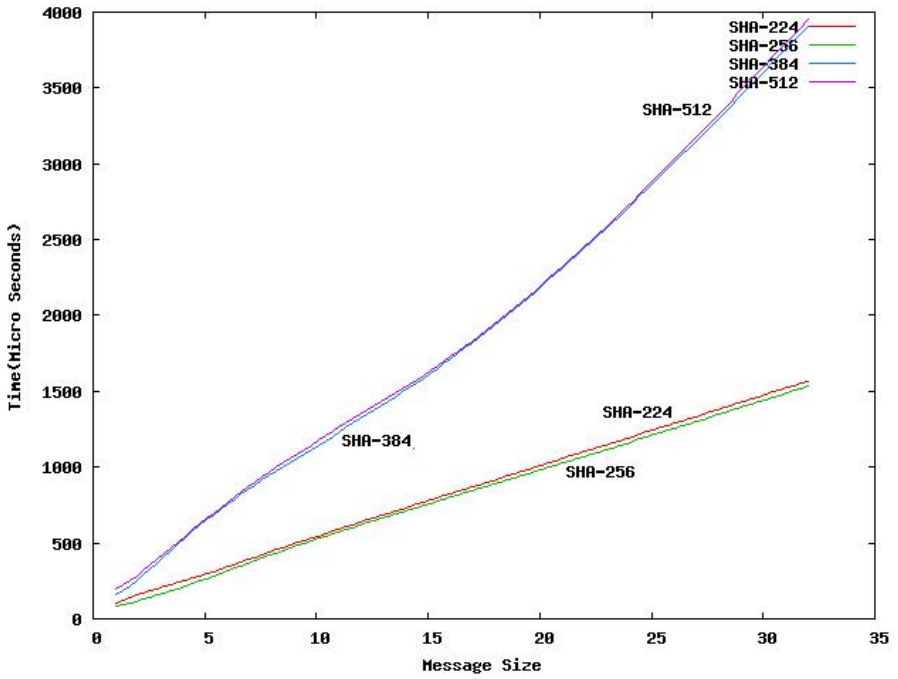

Figure 2.4: ECC performance

2.7 Technologies and their comparison

Past has shown that the computing technology has been developing at an exponential pace over the last 30 years since first microchips appeared on the horizon. Since then the

technology has not looked back and the computers have been growing smaller and faster. If we assume that the past trend continues to grow at almost the past pace we should be in a position to predict when the next change of key pairs should take place. Precisely what we intend to do in this section. Before we begin it would be beneficial to state the assumptions on which all our extrapolations are based.

It should be clear that the size of a key must be tied to the value of the data being protected by the key and also tied to the expected lifetime of that data. It makes no sense for an adversary to spend (say) $10 million breaking a key if the recovering key will only be worth (net) $10 thousand. But as the certificates which we target to replace by our system will naturally be worth more than that we would assume that there would be enough incentive for one to accrue enough computing power and break the code. But since there are technical difficulties we would assume as a standard that a $10 million machine would suffice to represent the standard code breaking computation device that could possibly be used by our adversary and base our calculations on that.

All extrapolations that follow shall be based upon extrapolating speed and memory enhancements of existing hardware and improvements in existing algorithms. While breakthroughs in both algorithm and hardware technology may occur, such events are inherently unpredictable and we do not consider them.

2.7.1 RSA

RSA cryptanalysis is based on factorizing numbers. Number Field Sieve(NFS) algorithm is the recent one and it is highly in use for prime factorization. On the basis of the heuristic complexity formula, time required to factor a prime number(N) using NFS is :

$$L(N) = e^{((c+o(1))((logN)^{\frac{1}{3}}(loglogN)^{\frac{2}{3}}))} \tag{2.9}$$

Once we have a benchmark for a particular N, Then it can predict the difficulty of factoring a larger number M relative to the difficulty of factoring N by computing L(M)/L(N) to get a time estimate. As a basis of comparison we could use data from the break of RSA-512. This effort required a total of 8400 MIPS-Years, represented by about 300 PCs averaging 400 MHz and with at least 64 Mbytes of RAM, running for 2 months, and 10 days and 2.3 Gbytes of memory on a Cray C90 to solve the matrix. Using following data we can predict how much harder it is to factor a number of 576, 640, 704, 768, 1024 or 2048 bits:
$L(2^{576})/L(2^{512}) \sim 10.9$
$L(2^{640})/L(2^{512}) \sim 101$
$L(2^{704})/L(2^{512}) \sim 835$
$L(2^{768})/L(2^{512}) \sim 6$ x 10^3

$L(2^{1024})/L(2^{512}) \sim 7$ x 10^6
$L(2^{2048})/L(2^{512}) \sim 9$ x 10^15
Thus,
1. 576 bits will take 10.9 times as long as RSA-512.
2. 768 bits will take 6100 times as long as RSA-512.
3. 1024 bits will take 7 million times as long as RSA-512.

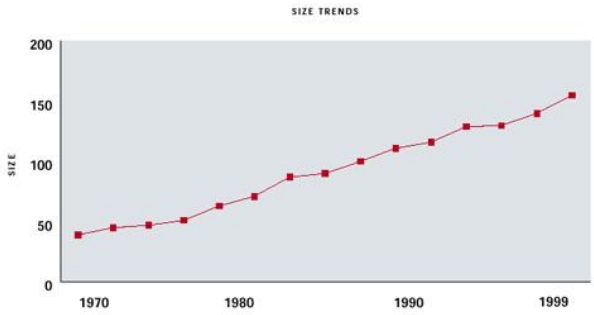

Figure 2.5: RSA Factoring Records Since 1970

If the computing power doubles every year it can be inferred from the above graph that the year Y in which D bit key would be broken is given by

$$Y = 13.24D^{\frac{1}{3}} + 1928.6 \tag{2.10}$$

The correlation coefficient is about 0.955. According to this formula, a general 768-bit number (D=231) will be factored by the year 2010, and a general 1024-bit number (D=309) by the year 2018.

2.7.2 ECC

The best known attack against an Elliptic Curve Discrete Log system is based upon a collision attack and the birthday paradox. One expects that after computing approximately sqrt(order of the curve) points, that one can find two points that have equivalent algebraic relationship. From this collision, the key can be found. Thus, the best known attack is purely exponential in the size of the key. The time complexity is:

$$T(k) = \sqrt{\frac{\pi}{2}} * 2^{\frac{k}{2}} \tag{2.11}$$

where k is the bitsize of the order of the basepoint.
In 1996, a special purpose hardware design was proposed that for $10 million could break a 155-bit Elliptic Curve key over a 120-bit sub-field in 32 days. The time to do a k-bit

Elliptic Curve is then $32 * \sqrt{2^{k-120}}$ days with one of these machines. Today same cost machine that is 50 times faster can break the given key(155 bit) in about 12 hours rather than 32 days.

2.7.3 RSA and ECC equivalent key size

Having done individual analysis of ECC and RSA we now present the relative strengths of same bit keys of the two technologies in the following table assuming that $10 million is available for computer hardware at all points of time.

ECC	RSA
112	430
160	760
192	1020
256	1620

Table 2.1: Comparitive key-lengths(no fo bits) of RSA and ECC

2.8 Manaul System for Degree management at IIT Delhi

An institute like IIT Delhi issues a lot of degrees every year. Managing them in such a manner that no fraudulent/incorrect degree is generated, requires a whole lot of effort. In manual system, IITD tries to ensure these things by distribution of trust and rigorousness of the method followed to generate the degrees. In the coming text, we will explain the manual system of management of degrees at IITD, so that we can build the digital system as close as well as at least that much secure.

2.8.1 Getting the Passing list

IIT Delhi maintains a database of courses undertaken by all the students and the grades they have obtained in all these courses. Each year at the time of degree generation, UG section receives a list of students which have completed degree requirements that year and are eligible for getting degree that year. This list is passed to the chairman of grades section for verification.

2.8.2 Verification and Final List

Chairman verifies the above generated list manually by grade and course database of the students. After verification, this list is uploaded on standalone computer (used only for

publishing the degrees). Then again to add another layer of trust, this list is verified by some senior officials of IIT Delhi, and by common consent this list is approved.

2.8.3 Generating the degrees

On the standalone computer on which the list which uploaded in step 2, now the process of typing every degree manually is started. Each degree is typed manually by reference of list that was generated in 2nd step. These set of degrees then are verified by UGS staff manually to remove any minor discrepancy that may have arrived during typing of degrees.

2.8.4 Signing

After generation of **authentic** list and set of degrees in 3rd step, set of degrees along with the list is sent to Registrar for signing. Registrar signs every degree after verifying with the list. Then the same set of degrees and list is passed to Director. Director also signs every degree after verifying it with list and also Registrar Signature. Same signing process is repeated for Chairman, Board of Governance also.

2.8.5 Publishing

At this stage, all the degrees have been prepared, signed and are ready to be released. Now every degree is distributed to corresponding students. To maintain the confidentiality, every degree is either handed out to the corresponding student or a person who has a letter signed by student to take his degree.

As we can see that in the manual system, IITD is ensuring that no fraudulent/faulty data is generated by exhaustive verification of the generated degree data several times. Also, they are ensuring that even if a fraudulent degree is generated then they can catch the responsible person for generating that degree.

2.8.6 Verification

The last but probably the most important part of IITD degree management is the verification. Verification of degrees is really important because an employer/university cannot trust every degree certificate that just looks like a degree certificate issued by IITD. So what they do is that after receiving the degree from student, they send it to IITD for

verification. The verification is again done manually, checking all the details of degree against the grades data of the student.

As numbers of degrees issued by IITD are increasing every year, the numbers of requests they are going to receive for degree verification are also going to increase at the same rate. It is also a worthy fact that you cannot give a job of verifying the degrees to an non-authentic person, the person should be an authentic person of IITD. In near future, it will require more time and more man-power if IITD has to serve all the verification requests. That is the main reason that the electronic system is necessary and should be considered for total replacement of manual system rather than just a supporting system.

Chapter 3

Initial Design and Problems

Cryptography is a field where finding a new method for encryption and decryption is very difficult and may take many years of many researchers.The complexity lies in the mathematical functions for cryptology having strong requirements. So in this paper, we will present the design which we have thought of, to work with existing technologies but handling most of the drawbacks of them.

Now lets delve into the design of the system for our specific application of issuing and managing degree certificates in IIT. The application is required to issue degree certificates for IIT students and the concerned authorities will digitally sign the document. The issued certificates will be used by clients like employers from various organisations to verify the GPA of IIT students. They will typically use Public Key Infrastructure (PKI) to verify the digitally signed document provided by IIT for that particular student.

The original design proposed for the problem had many drawbacks and it took several iterations to modify the design further. Here, we will go step by step by first discussing, in brief, the original design and its drawbacks, then the modifications leading to the current design finally. This approach will helpfully develop a good thinking process and will elaborate the problems and issues specifically.

3.1 Phase I

Initially we proposed a design to keep all degree documents in some format like jpeg or pdf and use re-encryption for multiple signatures on technology up-gradation or change. Here are the key features.

All documents will be signed by the current concerned authority e.g. director, registrar etc. and maintained on our internal machines which would be then accessed by web-server accessed by our clients e.g. potential employers. The design can be described into three parts:

1. Managing documents securely to avoid single point of failure in case of data theft or loss.

2. Signing and verification of the documents.

3. Re-encryption after some time to avoid breaking of keys because of increasing computation power and better algorithmic approach with time.

Let's discuss each of the points briefly as to find out what are the main considerations for each point:

3.1.1 Managing documents

All the documents on the server and machine have to be maintained securely, for that we take following steps:

1. All documents will be stored on internal duplicate machines so that an outsider can't hack to feed duplicate documents.

2. For uploading a document (in unsigned form also), we will use biometric access to the machines.

3. All access to signed documents (read-only) will be through a web server which will use login information from the required authorities.

3.1.2 Signing and verification

The degree issuing person will get his private-public key pair from a certifying authority. The signed degree will be posted on the IIT side with a unique ID given to the document so that the student can give the corresponding URL to his employer or anyone who wants to verify the document. The verifier will use PKI to verify the authentication of the signing person. If there are multiple people signing the document, verification can be done serially.

3.1.3 Re-encryption

All keys become insecure after a period of time because of the increase in computation power and improvements in algorithms to break down the public key to get corresponding private key, so re-encryption is required:

1. Each signed document will have a time-stamp that it is signed at this time so that during re-encryption, you can't create fake document stating that it was also created earlier.

2. After validation date expires, the authority will sign the signed document without decryption i.e. it will add extra layer with higher bits which will enclose the earlier smaller key. It will say that these documents are valid further. So the verification is complete only with both the signatures and with the time-stamp of the issuing date.

3. For re-encryption, we will use API plug-in which can be used to completely change the algorithm for encryption, so even if RSA fails tomorrow, we can use another algorithm and plug-in the corresponding API and re-encrypt.

This design was able to handle many of the issues like handling the issue of re-encryption but there were some serious drawbacks like:

- Digital Time-stamping has to be outsourced to some external certifying agency which may or may not exist later. So we want only standard permanent things to be used which will exist forever.

- Cost of re-encryption operation would be large after a while when total number of documents issued by system will be large because every document has to be over-encrypted at each re-encryption and with each over-encryption, size of each document will increase. So scalability concerns were there.

- Issue of hand-off was not solved because when one director changes, the new director can have his private key in future (when the ex-director's public key would be broken) and then can possible fake documents in name of previous director.

- Existence of document format of documents like jpeg or pdf may not be relied upon i.e. these formats may not exist after a period.

- Deletion or invalidation of documents was not taken care of.

So with all these issues, it is not feasible to make system secure which will manage documents of utter importance. This led to certain changes in design and each iteration has something important to say.

3.2 Phase II

Instead of handling multiple documents for re-encryption, a single master document is proposed which will keep information of all documents in it and it will be simple text document with ASCII characters maintaining all the information which can be later used by our server program to create proper degree document on the fly and show it to the client. Now lets discuss the design in detail as this will lay the foundation for our final design.

3.2.1 Master Document

The master document keeps the records for all the documents issued by the organisation in a tabular form with the corresponding signatures of issuing authorities. Each record will keep the basic information of the student with the signature of his contemporary authority like:

- Student Name
- Entry Number
- GPA records
- Department and Batch
- Unique ID

After signing the individual records, the authority will sign the whole batch of degree documents issued at that time. Because degrees are issued in batches only once a year, the whole batch signing by the same issuing authority provides extra layer of security. Now, there will be no case of single degree addition (probably fake) to the master document. The master document will be signed, other than individual record signing, by the current director of the organisation approving its validity.

3.2.2 Client Requests

Now whenever a client requests for a student's degree certificate, the server program will access the data from the master document on internal machines and generate a representable degree document with all the required information and digital signature of the concerned authority who issued that record (this key may or may not be broken today) and then the server program will use IIT's private key to sign the document which will be in accordance with the current technology and will be renewed by CA whenever the need is there. Now IIT stamped certificate (i.e. signed using IIT's private key) will be passed

on to the client using some secured protocol which will ensure secured transmission of document to client over insecure network.

3.2.3 Degrees Addition

Whenever a new batch of degrees are added to the master document, they are first individually signed record by record by the concerned authorities and then the whole batch of records is hashed and signed by the authorities again. This batch is now appended into the master document and again signed by the current director of the institution approving the new batch of degrees released. Now we have one automatic constraint here that only current authority can issue new documents and hence the added documents' authority should match with the current authority.

3.2.4 Re-encryption

Once we know that the present key size or the technology has become vulnerable, we need to re-encrypt the whole master document with the new technology.

1. One way to find this re-encryption time is to first calculate the expected time of current technology failure using empirical predictions, as we'll discuss in a later section, and re-encrypt the document in the middle of that time period. So, for example,if we're encrypting in 2020 and predicted time of the failure of this technology is 2040, i.e. 20 years span, then we will re-encrypt the whole document in 2030.

2. Create the document hash again with the current technology, i.e. the hash size can be different from the earlier one and then sign the document using the new keys. Over-encrypt the document with the new hash and new signature.

3. The document information as such won't change during re-encryption, just the hash and the signature of the document are added onto the document. So, it should be checked while re-encryption that the information in the document is preserved.

4. As we are re-encrypting before the technology is broken, so nobody can create fake records at the time of re-encryption.

3.2.5 Hand-Off

Another change in master document occurs when the authorities change, i.e. director of the institution changes. In that case, only the main signature on the master document

will change. The full document is hashed again and digitally signed with the key of new director but here also the information in the document is preserved. Just the concerned authority has changed, so encrypting the already existing document with new director's private key will suffice.

3.2.6 Document Invalidation/Deletion

Now there may be case where we want to invalidate some record or delete some record. Here, we don't want to play with the master documents because it will change all the hashes created and also creates a possibility of interference by the current director into the documents issued by the previous directors. So, we will keep a separate document similar to master document one, which will keep the invalidated or deleted records. Hence, there will be two documents maintained, one the master document containing information for all records issued by the organisation and the other containing the information for the invalidated records. So at the time of a query, we'll first look into the invalidated records document if the asked record has been invalidated and if yes, then we can simply provide the information to client else we'll find the information from the master document and create document to pass on to the client as usual.

3.2.7 Mail and Log Events

Also we will register any changes to master document as events and mail these events to certain multiple authorities so that any mishandling with the document can be traced. We can keep a hash of the document in multiple places and any change in the document will lead to change in the hash and then this hash can be used so as to ensure the integrity of the document.

We can note in this design that re-encryption and hand-off is much easier, convenient and robust than the previous design. Re-encryption requires just a hash creation and then encryption to get the digital signature. Just two operations of constant time make it really simple and fast. Similarly, hand-off is also very simple. Also since we keep the data in text format only, it is never going to die. This is the simplest form of data to keep for long periods with just ASCII characters. This feature of this design makes it far convenient and robust which can be scaled to any number of documents.

Chapter 4

Final Design

We have tried to make digital system for degree management as close as possible to the currents manual system. But digital system does not enjoy the same privileges as manual system. In manual system if you ensure that no fraudulent degree was not generated initially then you pretty much done. And manual system also enjoys the validity of manual signatures ffor very long periods but digital system signature become invalid after sometime. In what follows, we explain the final design of our system in detail. The whole design has below mentioned as major components.

4.1 List upload

In the same manner as in the manual system, we upload the list of passing students and their details to the document server. This list is again uploaded by Chairman, Grades section as done in manual system. It is responsibility of the Chairman that this list is correct in all the manners.

4.2 Generating Master Document

After getting the student details in the 1^{st} step, next step is to generate the corresponding Master Document in XML. All the details of student may not be ASCII text format, some of the data may be regional language name of student, image of student etc. As we promised that we would keep all the data in ASCII text format, we use UTF-8 and Base-64 encoding to store such data along with their encoding information. At the time of access of all this information, we have to decode all this information to their original format to generate a formatted degree certificate. Our final master document looks like a table as shown in the figure.

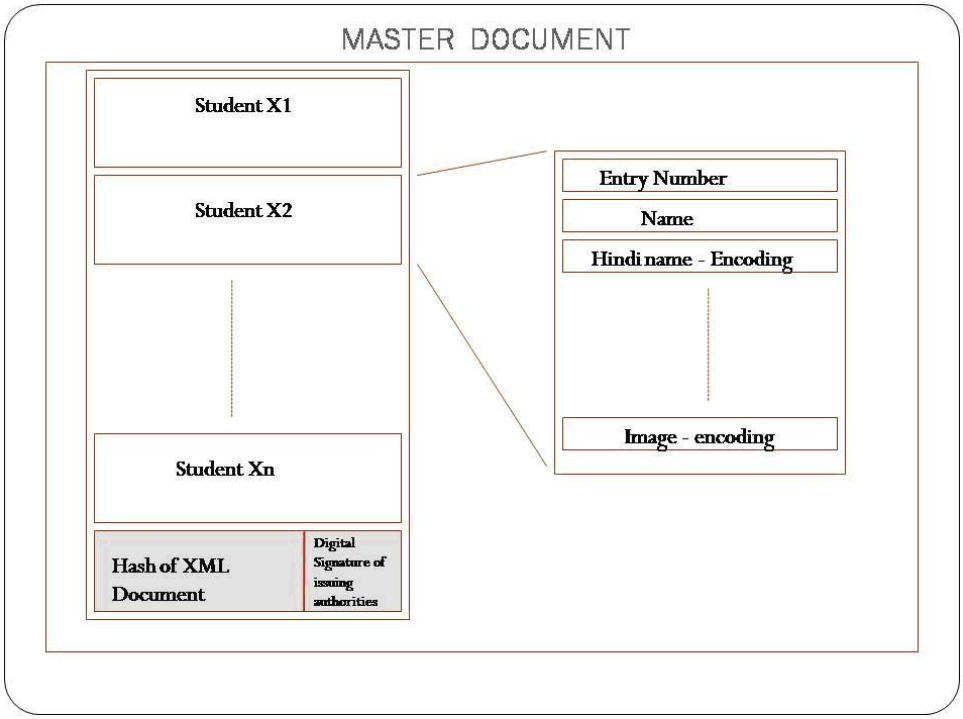

Figure 4.1: Master Document

4.3 Signing

After generating the master document, now its the turn of all signing authorities that they sign in same hierarchical order as done in manual system. As a resemblance to the manual system, each signing authority has to verify the master document and sign of the previous signing authority. So we have made our system constrained in such a way that it will only allow signing in the same order as in the manual system. When any authority signs the document, we publish the information and now the higher authority in the hierarchy is allowed to sign the master document after verifying the whole data and sign of previous authority. We also publish the public key of every signing authority after each signing.

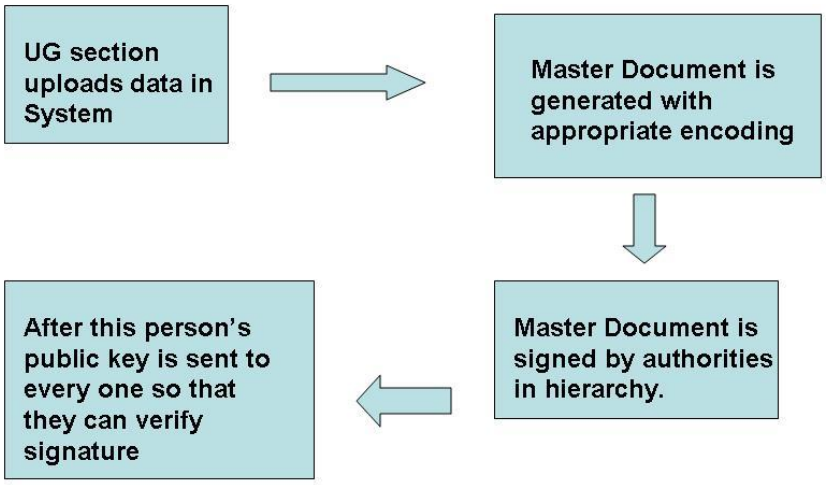

Figure 4.2: Generation and Signing of Master Document

4.4 ID generation

After every signing authority has signed the document, now we generate the student IDs which are used to maintain the confidentiality of the degree documents. We provide every student with some ID which will be unique for every student and will point to the information of corresponding student. This ID should be non-guessable and a random looking string. For generating this ID, we follow a very simple process. Foe each student, we append a randomly generated 64-Bit string to entry number of the student to get a complex unique string for every student. Then we hash this string using MD-5 hash function. This hash is the ID by which student can access his/her degree.

4.5 On-fly generation

In this component, we generate the degree document on the fly on the basis of ID provided by student. For every ID, we have the corresponding entry number and year in our database which was populated after publishing the Master Document. We get entry number and year from database, and then we get the Master Document for that specific year and verify it with public keys of authorities and system generated public keys. If verification of the Master Document turns out to be fine, we make a hash of the master document on the basis of entry number and we get the information of student from that hash by using the entry number we got earlier. Now this information is used to generate a formatted degree document on the fly. We sign it with the IITD private keys and send this degree along with signature and public key certificate of IITD to client for verification purposes.

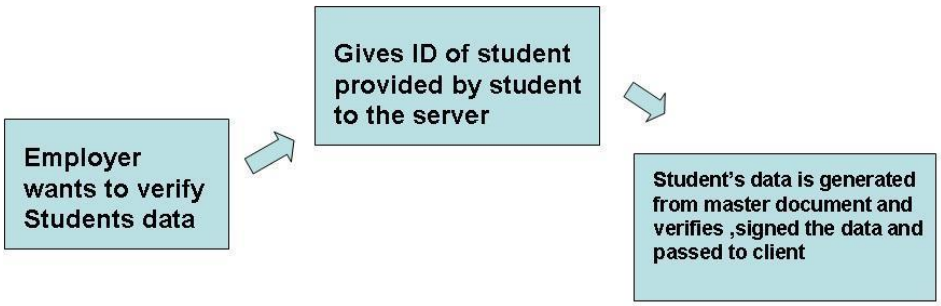

Figure 4.3: On-fly generation and Client Side Verification

4.6 Verification

Client is provided with degree document, signature data and public key certificate of IITD. First he verifies the public key certificate of IITD using PKI scheme and also the MS windows trusted root store. Then he verifies the integrity of the degree document which he is accessing by using already verified IITD public key certificate and the signature data we provided him at the time of On-fly generation.

4.7 Resigning

In the manual system, we are pretty much done after the signing part because manual signatures are valid forever and their validity never gets expired. But we are not that fortunate in case of digital signatures; digital signatures do not enjoy the luxury of being valid forever. They become invalid after some years of their generation. So we have to re-sign the document to ensure the integrity of document. For our application and many other applications of this kind, the record needs to be signed for years, decades and may be centuries. So for the purpose, we will resign the document with a fresh upgraded key before its original digital signature is vulnerable.

4.7.1 Lock

At the time of re-signing the documents, which would be every year whenever we are changing our database, the document would be signed by a "RESIGNING LOCK" public key certificate which will be used to sign the previous year documents with its private key and thereafter, this private key will be destroyed. This is analogous to lock something and then throwing away the key so that no one can ever change this data, only it can be

read and verified. We also timestamp this system generated public/private key pair so that no problem of forgery of this key arises.

Chapter 5

Implementation

We have implemented the prototype for IIT degree management system. Our main concern was to generate degree document on-fly and make client able to verify it without much hassle. We have used Perl/CGI for extracting the student information from the master document, generating a nicely formatted degree certificate and signing it. On the client side we have used Java applet technology to verify the degree with its corresponding signature. There are 4 main components of the whole prototype that we have implemented; now we will look at all these components in detail.

5.1 Master Document Management

The generation of the master document involves uploading a schema which defines the fields in each entry of the master document as well as their corresponding encoding. Both of things are needed for successfully decoding the document at later point of time. A variable number of entries can be added to the schema at this stage, which is taking care of the fact that any arbitrary document can have any number of fields and any type of encoding for them. Next stage involves uploading a data file which contains the student particulars in CSV format, one row per student. Once the file is successfully uploaded, the data in the file is displayed in tabular form and user is provided with an option to generate the master document in XML format. Once the document has been successfully generated, the other concerned authorities can login into the system for viewing the master document and signing it. They can sign only when the authority which is lower in hierarchy has signed. The singing algorithm that we have used is RSA with SHA1, standard implementation from the CPAN PERL package repository has been used. Once all the authorities have signed the document the document is published. The mechanism of publication is simple. As mentioned in the chapter on design, for each entry in the master document a non guessable id is generated and stored in the database. After all

the signings, we do the locking. Locking is done by generating a private key inside the system and then using them to sign all the existing documents from starting till the last year, and destroy after that. When everything is completed the public key can be time stamped to resolve to testify its existence at the time of signing.

5.2 Private keys

We got 4 pairs of experimental keys from (n)code solutions which is the Indian govt. certified Certificate Authority. We were provided these keys in the native private key format called .pfx format. For our purpose, we used openssl utilities to convert these into .pem format and also setting pass phrase on these private keys. Then we used standard Perl libraries to use these .pem files for signing.

5.3 On-fly generation

The students are given separate ids to them at time of convocation, in the form of a link of the form
https://xxx.xxx.xxx/getdegree.cgi?Student_ID=xxxxxxxxxxxxxxxxxxxxx
Here this ID is the same ID that we generated at the time of publishing of Master Document. When an employer/university supplies the server with this link, we perform below mentioned steps to generate a nicely formatted and sign it.

(i). We take this ID by POST method of CGI and we find out Entry Number and Year corresponding to this ID by querying the database that we populated with such IDs at the time of publishing of Master Document.
(ii). Every year will be having its separate master document. We will verify the validity of this document using the stored public certificates of all the signing authorities and destroyed keys.
(iii). After verification (If it turns out to be fine), we will load whole of the XML tree of that year into out program using Perl modules and generate an Entry Number based hash by the parsing the XML Master Document. Then we get all the information corresponding to the entry number obtained in 1st step. Then we will create a jpeg/html document by the information obtained. We will sign it with private key IITD assigned to the current root. We send out the generated degree document, signature data and IITD public key certificate to client.

Figure 5.1: A sample degree document generated on-fly

5.4 Verification on client side

Degree verification on client side may be the most useful component of our system. It gives client an extra assurance that degree document that he is accessing is authentic. For verification purpose, we have used the PKI scheme. In this part of design where we implement the client side verification modules for verifying the degree document generated on-fly and signed with private key of IITD.

On the client side, client will get following 3 files:

(i). Public Key certificate of IITD: This is the Public key certificate corresponding to private key that is used to sign the degree document on-fly.
(ii). Student degree file: This file contains nicely formatted student degree data.
(iii). Signature data file: This file contains the signature data that is obtained during on-fly signing of Student degree file by private key of IITD.

Client side verification has following two stages:

5.4.1 IITD Public Key Verification

A public key (or identity) certificate is a binding of a public key to an identity, which is digitally signed by the private key of another entity, often called a Certification Authority (CA). Users of public key applications and systems must be confident that a signer's public key is genuine, i.e., that the associated private key is owned by the subject. Public key certificates are used to establish this trust.
If someone wants to verify the public key of subject, he must have the public key of CA that signed the subject's public key certificate. If the user does not have a trusted copy of the public key of the CA that signed the subject public key certificate, then another public key certificate vouching for the signing CA is required. This logic can be applied recursively, until a chain of certificates (or a certification path) is discovered from a trust anchor or a root CA to the target subject (commonly referred to as the end-entity). The root CA is usually specified by a certificate issued to a CA that the user directly trusts. To verify the IITD public certificate at clients end, we supplies the complete certificate chain (certificates list containing certificates subject public certificate, Intermediary authoritys public certificates and root CA public certificate) and using the Window Trusted Root store, we verifies the supplied certificate chain. If certificate chain got verified then IITD public certificate that client is using is a genuine IITD certificate otherwise not.

For chain verification functionality, we have used java standard verification function provided for PKI scheme.

For our example, initially public certifciate of root CA corresponding to certificate chain of the IITD certificate is not installed in **MS windows trusted root store**, so It refuses to verify the chain. Thats why its is showing that **Not able to verify following Certificate Chain**.

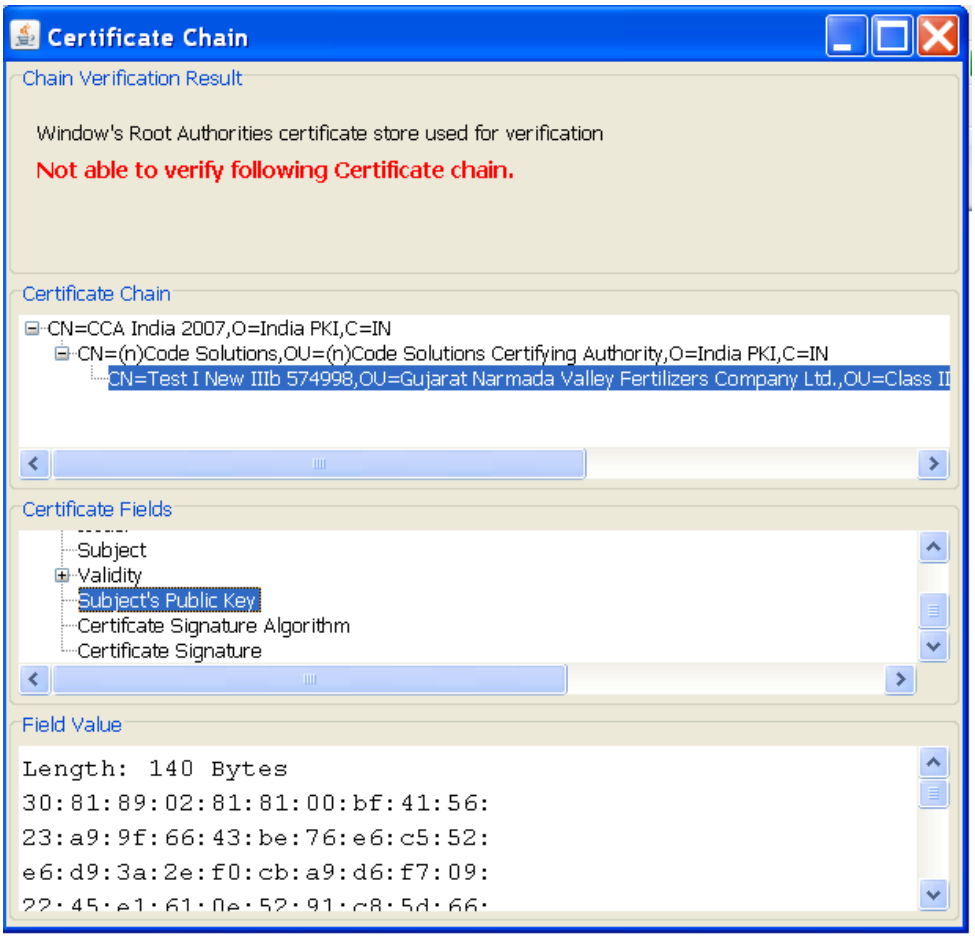

Figure 5.2: Certifciate Chain not verified by MS windows trusted root store

Now we installed the public certificate of root CA of corresponding certificate chain in **MS windows trusted root store** and now certificate chain is verified. Thats why it shows **Certificate Chain is OK**.

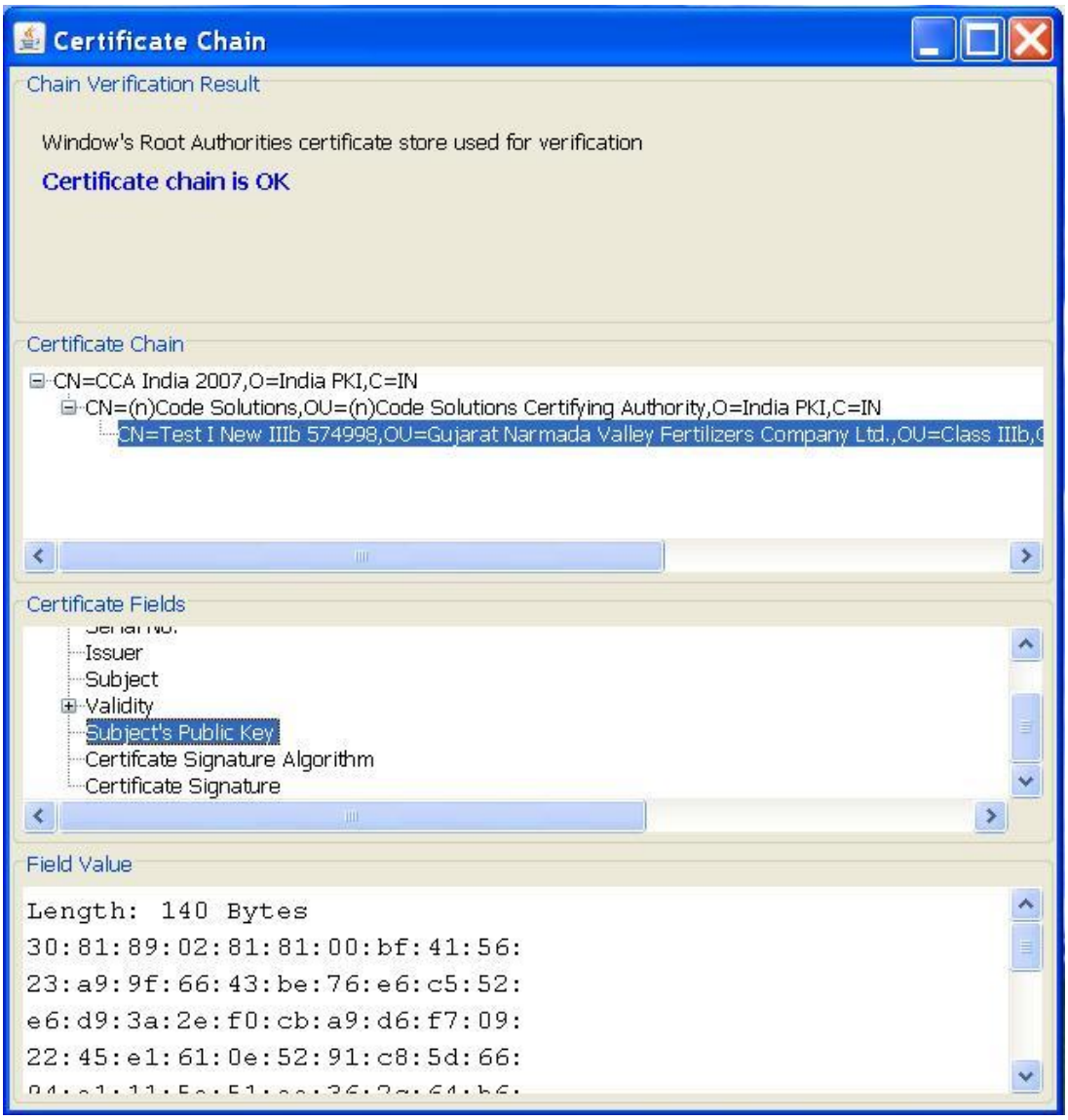

Figure 5.3: Certificate Chain is verified by MS windows trusted root store

5.4.2 Student Degree Verification

On hitting the verify button, Student Degree data file that client will be seeing in browser is verified using the signature data and automatically downloaded IITD public key cer-

tificate. If client wants to use self downloaded IITD public certificate instead of using automatically downloaded IITD certificate then he can browse to that certificate and it will be used for verification.We have provided client with an option of verifying the degree, by default it verifies the degree certificate which it automatically downloads from IITD server. User can also input self downloaded degree documemnt and can verify it.

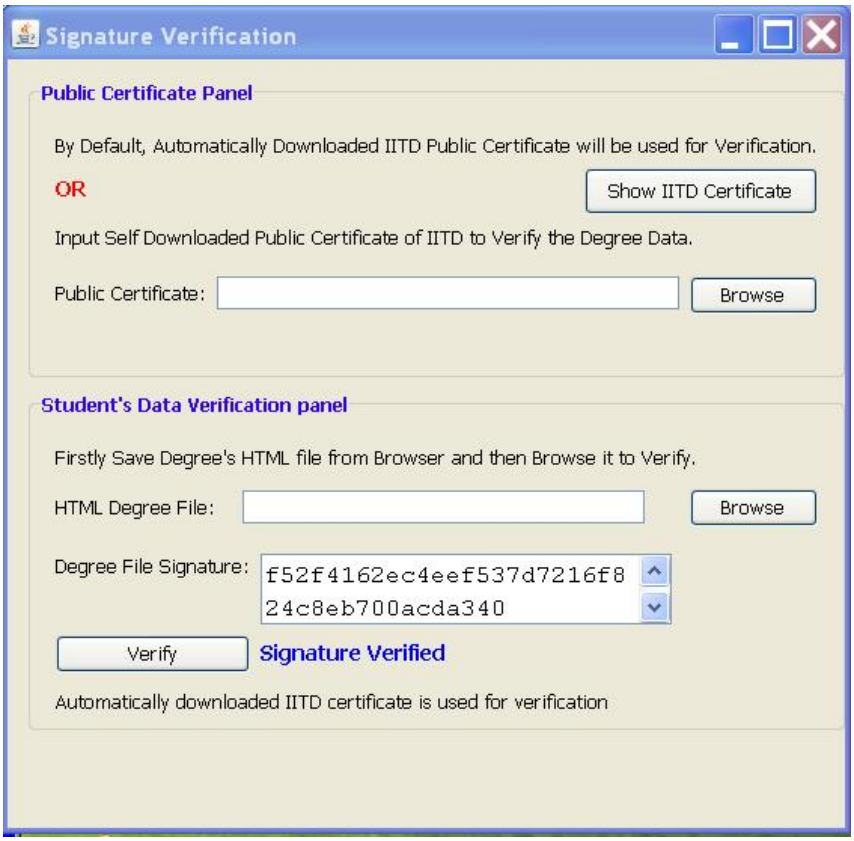

Figure 5.4: Degree Document Verification

As we can see in the following figure that degree is not verified if we try to change the signature value.

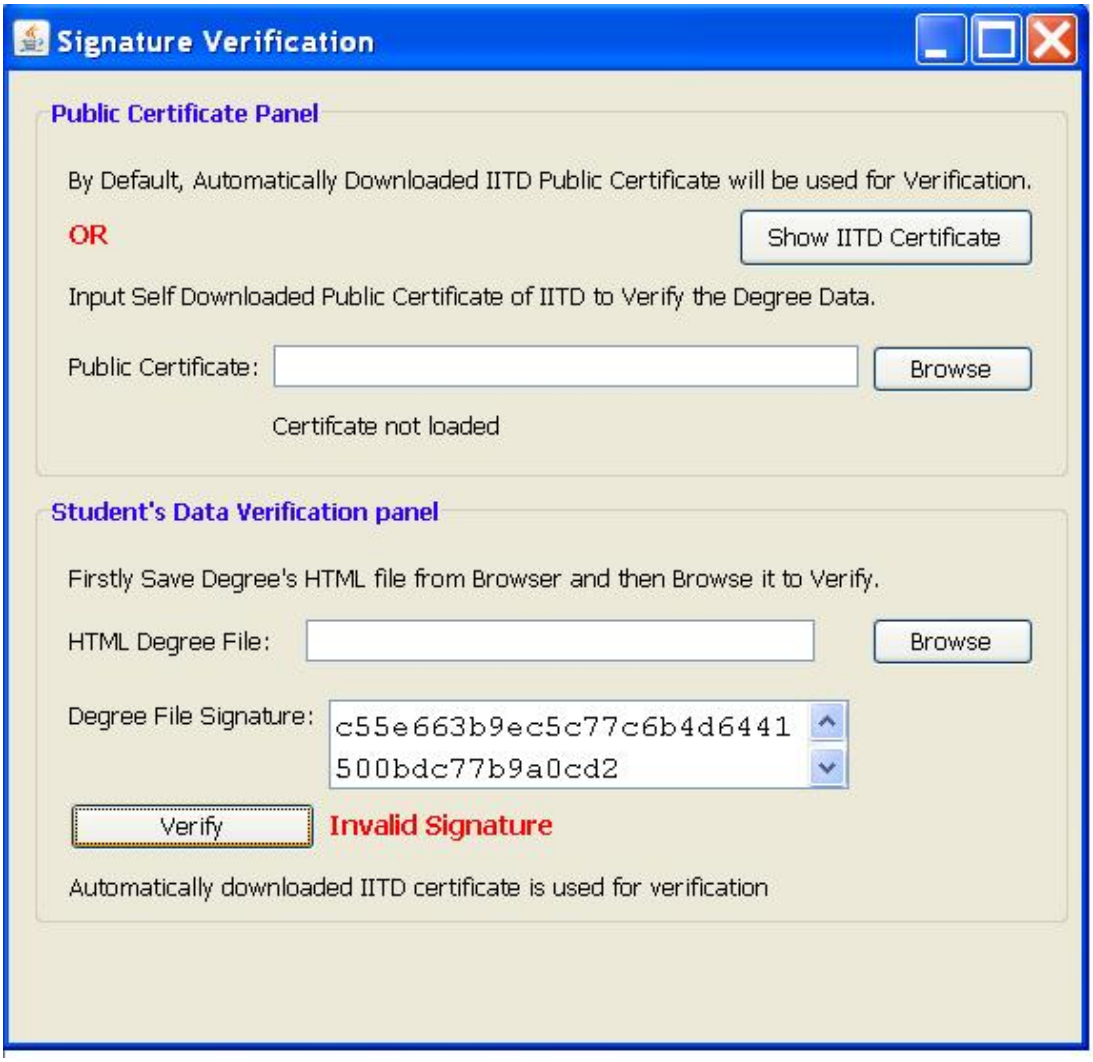

Figure 5.5: Degree Document Verification

Chapter 6

Issues and Assumptions

During the whole process of study of the technologies used for managing secure documents over long periods and also during the design of system, we came across some issues which will be inherent in any such system which tries to manage these documents in somewhat similar ways as we do. Some of these issues may be specific to IITD degree management system while some are general issues which will be inherent in any such system. Now we are going to explore these issues in detail.

6.1 Document Format

We do not have the privilege of storing these documents in any fancy format like PDF, jpeg, MS word etc. The reason is that we cannot be sure about the life of these formats. It is highly probable that these formats will become obsolete after some years. For example, Microsoft has already replaced .doc format with .docx. So we will have to store all the information in plain text format and at the time of access, we can generate any fancy format document which is being globally used at that point of time. This issue is not IITD degree management specific, it is a very general issue that will be found in any system that manages documents over long periods.

6.2 Long enough stay/ Consecutive Authority Collusion

This problem arises when some signing authority occupies that position for long enough time. Lets say current authority is C who has occupied this position for long enough period. Lets say previous authority was P. As in this scenario, C has stayed long enough to break the private key of P. Now C has access to all the private keys which have been

used so far because he knows all his private keys and he has private key of P and also the private keys of all the authorities before P. So now C can create any document from scratch on behalf of P and all the authorities before P.

In the 2^{nd} scenario, it is possible that C colludes with previous authority P to create any fraudulent document on behalf of authorities who occupied that position before P.It is a very improbable event though, as at any time we have 3 signing authorities and probability of all 3 staying for long time and colluding with 3 previous signing authorities looks very small. So we can assume that it will not happen.

6.3 Corrupted Root

There are some problems in managing the security of server which is managing the Master Document and also of server which is serving degree requests on the fly. The problem which is of most concern is that we don't have the luxury of blindly trusting the root of these servers. Root of the master document server can do anything with the master document as he can create a master document from the scratch with his own set of keys and that way front end server will keep serving the fraudulent degree documents without knowing the fact that master document itself has been compromised. Root of the front end server can also serve any fraudulent degree document without even contacting the master document server and even without extracting any information from master document. So either we have to tust these roots or we have to use special hardware boxes on which even root does not have full control.

6.4 Confidentiality

The documents which we are managing, our main concern is integrity of the document and we are not much concerned about the confidentiality of the document. But in some cases like degree certificates, confidentiality is a significant issue and we cannot ignore it. So we keep provide each student an ID which will map to his degree data. Only student will know his/her ID, and he/she can pass it to the employer/university. This ID has to be permanent for enough time and an option to change it on student's request can be provided.

6.5 Client Side Signing

Any signing anuthority will not want to upload their private keys to the signing server because it may led to stealing of their private key. So rather than signing takng place on

master document server, signing authority would like to sign master document on client side. Signing on client side will remove the risk of private key being compromised. So signing authority should verify the master document (verify signature of authority which is lower in hierachy) and then download the document from master document server and then upload it again on master document server after signing it on their side.

Chapter 7

Conclusion

Secure Document Management is a widely used protocol. Although it has obvious solution for short term documents, the solution for long term documents is not that obvious and it requires a lot of effort to fully conceptualize and realize such a system. We have tried to conceptualize and realize a prototype of such a system. It reduces a whole lot of effort which is needed to verify the degree certificates of IITD. In this system, we have not tried to use any new idea in cryptographic sense because it is too difficult to come with a new encryption scheme. We are using the usual asymmetric key encryption techniques to build a secure system with the help of techniques like locking, re-signing and time stamping. Most of the time, the stress is only laid on the integrity part of the document and people tend to forget the importance of storage format, we have also taken that into account. Although it is sure that if someone builds a quantum computer or discovers a faster algorithm to factor the numbers, then the whole infrastructure of PKI and RSA will become a failure along with our system. In the end, we can conclude that if computing infrastructure continues to increase at the same rate as it is today, our system will keep working securely for years to come.

Appendix A

Code Documentation for on-fly generation

In this section, we are mainly dealing with the main degree generating script **gerdegree.cgi**. This file does all the degree generation and signing operations with the help of 2 Perl packages. Brief details of all these 3 files are provided below:

A.1 Signature.pm

This Perl package contains the Perl functions for signing, verification, reading private key from .pem file and decrypting the .pem file. Input and output parameters of these are self understood and can be understood by looking at definition of these functions.

A.2 VerifyMasterDoc.pm

This file contains 1 important function which can be used to verify the signatures of the Master Document and it verifies the master document on the basis of access level you want to verify.

A.3 getdegree.cgi

This script is responsible for generating and showing signed degree to the user. It extracts the student ID from the link user has provided. Then it contacts the database for corresponding entry number and year. The master document corresponding to that year is verified using the function in **VerifyMasterDoc.pm**. Then a hash is generated by XML document, Entry Number of the student being the key. Then corresponding to the information obtained by this hash, we generate the HTML degree. And then HTML

degree is signed using IIT private key (using the function in **Signature.pm**) and then degree, signature and IIT public certificate are send to client for verification.

Appendix B

Code Documentation for Client Side Verification

JAVA provides Applet Technology to implement functionality on client side of web services. Applet technology is widely used and is a great success. So we also have used the same to provide client side verification of degree documents.

1. sp_sign_verifcation_applet.java
2. convert.java
3. certificate_read.java
4. certificate_validate.java
5. fileVerificationFrame.java
6. certificateChainFrame.java
7. sign_verify.java
8. windowVPanel.java
9. WindowUtilities.java
10. compile.bat

Above code files will be discussed in detail below:

B.1 sp_sign_verifcation_applet.java

This file is the main and starting point of whole code. This class defines a applet that shows "Verify Signature" button on the web page. it also initializes two Frames titled "Certificate Chain" and "Signature Verification" using classes *certificateChainFrame* and *fileVerificationFrame* respectively. Initially both Frames *Certificate Chain* and *Signature Verification* are invisible but when we press *Verify Signature* button on webpage *Signature Verification* frame becomes visible. Frame titled *Certificate Chain* becomes visible when *Show IITD Certificate*(available on frame, *Signature Verification* frame) button is pressed. In this file, actions of various buttons lying on *Signature Verification* and *Certificate Chain* frames are also defined. For more technical detail, see the comments in file.

B.2 convert.java

This class contains methods to convert one data type to another datatype. Following are the main conversion methods defined in this class:

1. **bytesToHex:** It is used to convert byte array to hex string.
2. **hexToBytes:** It is used to convert hex string to byte array.
3. **BytesToBase64:** It is used to convert byte array to Base64 encoded string.
4. **Base64ToBytes:** It is used to convert Base64 encoded string to byte array.

B.3 certificate_read.java

This file contains class to read public certificates from server and forms a certificate chain (JAVA CertPath Object) containing subject public certificate, intermediary CA public certificate and root CA public certificate. Order of arguments passed to constructor method *certificate_read* is very important. First argument must be name of subjectŠs Public Certificate file name (i.e. IITD public certificate file name), second argument must be intermediary CA public certificate file name (i.e. (n)Code public certificate file name) and third argument must be Root CA public certificate file name (i.e. CCA public certificate). Using above passed certificate, if forms the certificate chain (CertPath object) which is stored as a **certificate_read** class variable named **cp** which will be used later for verification.

B.4 certificate_validate.java

This file defines class to verify a provided certificate chain (CertPath Object) using Root CA public certificates installed in MS windows Trusted Root Store. JAVA's SUNMSCAPI provider is used to access MS windows Trusted Root Store. SUNMSCAPI provides tutorial is available on following link:
http://java.sun.com/javase/6/docs/technotes/guides/security/SunProviders.html#SunMSCAPI
Reader also must go through following tutorial to understand java crypto API.
http://java.sun.com/javase/6/docs/technotes/guides/security/certpath/CertPathProgGuide.html
Certificate chain (CertPath Object) generated using *certificate_read* class and stored in its variable named *cp* is passed to *certificate_validate* class method *certificate_valid* and result of verification is stored as CertPathValidatorResult Object *cpvResult*(variable name).

B.5 fileVerificationFrame.java

This file contains code that initializes a Frame which is used for Degree file verification. It is the frame with title "Signature Verification". Code was generated using Netbeans Form generator. Function of various components on this form is described below:

B.5.1 Show IITD Certificate Button

This button makes frame titled *Certificate Chain* visible. It also calls *certificate_valid* method of class *certificate_validate* to verify the certificate chain.

B.5.2 Public Certificate Browse Button

If user wants to use self downloaded IITD public certificate instead of using automatically downloaded IITD public certificate for degree file verification. Then using this button he can browse for his self downloaded IITD certificate and then browsed IITD certificate will be used for verification.

B.5.3 HTML Degree File Browse Button

If user saved the Degree file and corresponding signature value and at later time he wants to verify that degree file. In that case, user can browse to save degree document using this button.

B.5.4 Degree file signature

It shows the degree file signature value in hex string format. This signature value is used to verify the degree file.

B.5.5 Verify Button

This button verifies the signature on Degree file. Automatically downloaded IITD public certificate will be used if user did not supply self downloaded certificate.

B.6 certificateChainFrame.java

This file's code initializes a Frame with title *Certificate Chain* that shows the public certificates in certificate chain (read from server) and shows various fields and their corresponding values of a selected public certificate. Also shows whether or not certificate chain got verified using MS windows Trusted Root store.

B.6.1 Chain Verification Result Panel

This panel shows the result of verification of certificate chain.

B.6.2 Certificate Chain

This panel shows the tree whose root node is correspond to Root CA of certificate chain(stored as *certificate_read* object variable*cp*) , node at second level is correspond to intermediary CA and leaf node is correspond to subject certificate.

B.6.3 Certificate Fields

It shows tree correspond to the various fields of certificate selected in *Certificate Chain* panel.

B.6.4 Field Value

It shows the value of field selected in *certificate Fields* panel.

B.7 sign_verify.java

This class provides methods to sign and verify signatures. There are two versions of signing and verifying, one where data is passed as String and another data is passed as

File.

B.7.1 signFile

This method signs data passed as file using provided private key.

B.7.2 verifyFileSignature

This method verifies signature using provided public key and signature data as string where data is passed as file.

B.7.3 VerifyFileStreamSignature

Verifies signature using signature data and public key certificate and data file is chosen depending on condition.

B.7.4 signString

This function signs the data passed as string with provided private key.

B.7.5 verifyStringSignature

Verifies signature on data passed in String format.

B.8 windowVPanel.java

It defines the structure of top panel (Chain Verification Result) of frame *Certificate Chain*. Code for this was generated using Netbeans Frame generator.

B.9 WindowUtilities.java

This class provides few utilities that simplify using windows in Swing. It has following methods available:

B.9.1 setNativeLookAndFeel()

Tell system to use native look and feel.

B.9.2 setJavaLookAndFeel()

Tell System to use JAVA look and feel. It is default one.

B.9.3 setMotifLookAndFeel()

Tell System to use Motif look and feel.

Look refers to the appearance of GUI widgets (more formally, JComponents) and *feel* refers to the way the widgets behave

B.10 Compile.bat

Applet runs with limited privileges on client side machine. With these privileges they can not read and write and local machine. But we need to read from local machine (browsing self downloaded IITD certificate, using MS window Trusted Root store). So we need to provide extra permissions to our Applets. So for this we make the jar of all class files and then sign jar file using self generated Private Key. On the client side, a Dialog box appears asking whether client trusts the signer of jar. If clients trusts then applet gets extra privileges on local machine and it is able to read and write on local machine. This is a script file to create jar and then signs it. You need to have public private key pair to sign jar. If you did not have them then generate pair using JAVA *keytool* utility.

We hope that We have provided enough information about each file (class) that is part of this client side verification project. Source code in these files is also widely documented and you should go through these comments for more technical details. Reader should also go through *JAVA PKI guide* for basis knowledge of JAVA Crypto API and how they work.

References

[1] Information Security, Mark Stamp *John Wiley and Sons*, Edition 2006.

[2] Cryptography and Network Security Principles and Practices by William Stallings, *Prentice Hall, Fourth Edition 2005.*

[3] Openssl toolkit FAQ *http://www.madboa.com/geek/openssl/.*

[4] JavaTM PKI Programmer's Guide *http://java.sun.com/javase/6/docs/technotes /guides/security/certpath/CertPathProgGuide.html*

[5] Crypt::OpenSSL::RSA - RSA encoding and decoding, using the openSSL libraries *http://cpan.uwinnipeg.ca/htdocs/Crypt-OpenSSL-RSA/Crypt/OpenSSL/RSA.html.*

[6] RSA Labs *http://www.rsa.com/.*

[7] Public key infrastructure *http://en.wikipedia.org/wiki/Public_key_infrastructure.*

[8] Elliptic Curve Cryptography *http://www.certicom.com/.*

[9] SHA hash functions *http://en.wikipedia.org/wiki/SHA-1.*

[10] Crypt::OpenSSL::X509 *http://search.cpan.org/∼daniel/Crypt-OpenSSL-X509-0.6/X509.pm.*

[11] XML::Simple - Easy API to maintain XML *http://search.cpan.org/ grantm/XML-Simple-2.18/lib/XML/Simple.pm.*