SECURITY IN INTERNET VOTING SYSTEMS USING CRYPTOGRAPHIC TECHNIQUES

Project Report submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology

In

Electronics and Communication Engineering

Under the supervision of

Prof. T.S. Lamba

By

Prateek Dhaka – 081002
Shashwat Jain – 081060
Sarvesh Kumar Yadav – 081109

To

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY

Jaypee University of Information and Technology

Waknaghat, Solan – 173234, Himachal Pradesh

CERTIFICATE

This is to certify that project report entitled "SECURITY IN INTERNET VOTING SYSTEMS USING CRYPTOGRAPHIC TECHNIQUES", submitted by Prateek Dhaka (081002), Shashwat Jain (081060) and Sarvesh Kumar Yadav (081109) in partial fulfillment for the award of degree of Bachelor of Technology in Electronics and Communication Engineering to Jaypee University of Information Technology, Waknaghat, Solan has been carried out under my supervision.

This work has not been submitted partially or fully to any other University or Institute for the award of this or any other degree or diploma.

[illegible]

Date: 29-5-2012

[Prof. T.S. Lamba]

Dean (Academics & Research)

ACKNOWLEDGEMENT

This project is an outcome of our serious effort to implement "Security In Internet Voting Systems Using Cryptographic Techniques" as part of our Bachelor's Degree Program.

Enhancing security in an internet voting system, involving the concepts of network security and cryptography, under guidance our esteemed mentor Prof. T.S. Lamba not only cleared all our ambiguities but also generated a high level of interest in the subject. We are highly grateful to him.

The prospect of working in a group with a high level of accountability fostered a spirit of teamwork and created a feeling of oneness which thus motivated us to perform to the best our ability and create a report of the highest quality.

To do only the best quality work, with utmost sincerity and precision has been our constant endeavor.

This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

Prateek

Date: 29-5-2012 [Prateek Dhaka]

[illegible]

[Shashwat Jain]

S. K. Yadav

[Sarvesh Kumar Yadav]

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

PIN	Personal Identification Number
RSA	Rivest Shamir Adleman
SSL	Secure Socket Layer
FTP	File Transfer Protocol
AES	Advanced Encrypted Standard
DES	Data Encrypted Standard
PB	Private Key of User B
UB	Public Key of User B
SHA	Secure Hash Algorithm
TCP	Transmission Control Protocol
EBCDIC	Extended Binary Coded Decimal Interchange Code
ASCII	American Standard Code for Information Interchange
EOF	End Of File
HTTP	Hyper Text Transfer Protocol
DB	Data Base

ABSTRACT

Internet voting is currently one of the most intensely debated subjects in information and communication technology, our project is to develop a secure internet based voting system that would deliver a free and fair election.

By making use of various cryptographic protocols available to us, we have tried to make internet voting systems as secure as possible. Main focus in the project is securing the voting mechanism between the user and client.

SHA-1 hash algorithm is used for authentication of each voter at the time of login process. Password entered by the voter during the login process is hashed using SHA-1 and is then sent over the internet to verify it with the hashed value of the password stored for that particular voter at the server database. This process verifies whether the voter is actually the person who he is trying to masquerade as.

Next is the voting process in which the voter casts his vote. After this, the vote is encrypted using the RSA algorithm and is sent over the internet to the server database where it is stored in the encrypted form till the counting process. Also, after the voter has casted his vote, a flag assigned to the voter is given the value '1' to make sure that he is not able to vote again.

When the voting process is completed, counting process begins. In this, since all the votes are stored in encrypted form, decryption of the votes is required. For the decryption, in order to avoid the possibility of any election official trying to alter the outcome of the counting process by decrypting the votes himself and then changing the votes in order to favor a particular candidate, threshold cryptography is employed which makes sure that the decryption process could not take place without the presence of a specified number of election officials.

CHAPTER 1

INTRODUCTION

1.1 Internet Voting

Among the many issues in the ongoing discussion about the Internet is its use in the voting process. Because voting determines who runs the government and entails two absolute requirements—the secret ballot and security from fraud—the stakes are higher than for many other transactions routinely conducted via the Internet. Public confidence about Internet security is increasing, but many feel that voting online requires a degree of security from fraud beyond the current standard for everyday Internet use.
Proponents of Internet voting suggest it could increase turnout, particularly among younger voters who are familiar with Internet technology.

1.2 Types Of Internet Voting

Two types of Internet voting are possible, and both were used in voting trials in 2000 in local elections in USA. One method, the more basic from a technical standpoint, is Internet voting at a traditional polling site, with computer voting machines connected to the Internet and where election officials authenticate voters before ballots are cast. The other method, more technically advanced, is to cast ballots over the Internet from remote locations using electronic authentication and computer security technologies.
Both methods were used; voters could cast their ballots from remote locations or at any polling place. Some observers believe that remote Internet voting should not be attempted until voters become comfortable with polling site Internet voting and until procedures are well established to ensure accurate voter authentication, ballot secrecy, and security.

1.3 Technologies Behind Internet Voting

Internet voting systems use several technologies to ensure authentication, secrecy, and security. These include encryption and electronic signatures (methods that use such techniques as passwords, personal identification numbers (PINs), smart cards, biometrics, and digital signatures) to verify the identity of the voter and provide data integrity (i.e., assurance that the data is not altered during. Other computer security technologies, such as firewalls, antivirus programs, and intrusion detection systems, are also used to prevent unauthorized hacker access to computer systems used in the election process.

1.4 The Current Debate: Issues and Challenges

While the computer security technologies mentioned above are well established in theory, they have not yet been used on a wide scale. Some government agencies, large companies, and financial institutions use encryption, electronic signatures, and other computer security techniques in conducting business transactions with established suppliers and customers. Some analysts predict that computer security technologies will proliferate at an accelerated rate in the next few years. Internet voting systems could be phased in over time, from the use of Internet-connected computers at state and !ocal government-controlled polling sites, to remote Internet voting from users' home PCs. The new voting systems must also be user-friendly enough that many voters will prefer to use the Internet method over the traditional method of voting.

1.5 Security Issues

Protecting the voting process from electronic attacks is a fundamental challenge both for vendors who design online voting systems and for election administrators who run elections. As with current voting systems, any vulnerability that could allow for voting more than once, changing a voted ballot or the election tally, or otherwise compromising the integrity of the process, raises the potential for fraud. In addition, Internet voting systems could be vulnerable to "denial-of-service" attacks in which the system is flooded with e-mail messages, causing it to shut down.

1.6 Design Criteria

Authentication: Only authorized voters should be able to vote.
Uniqueness: No voter should be able to vote more than once.
Integrity: Votes should not be able to be modified without detection.
Secrecy: No one should be able to determine how any individual voted.
Convenience: Voters should be able to cast votes with minimal equipment and skills.
Cost-effectiveness: Systems should be affordable and efficient.

1.7 Cryptographic Techniques In Internet Voting

a. RSA : Encryption & Decryption
b. Hash Function : Authentication
c. Threshold Cryptography : Trust
d. SSL : Secure Transmission (using FTP or HTTP)

CHAPTER 2

ELECTRONIC VOTING MACHINE

2.1 Introduction

At present, elections in India are held by employing the use of Electronic Voting Machines(EVM).This makes polling much fast and is more reliable than ballot papers. The EVMs save considerable time, money and manpower. It also helps in maintaining the secrecy of individual voting.

2.2 Overview Of EVM

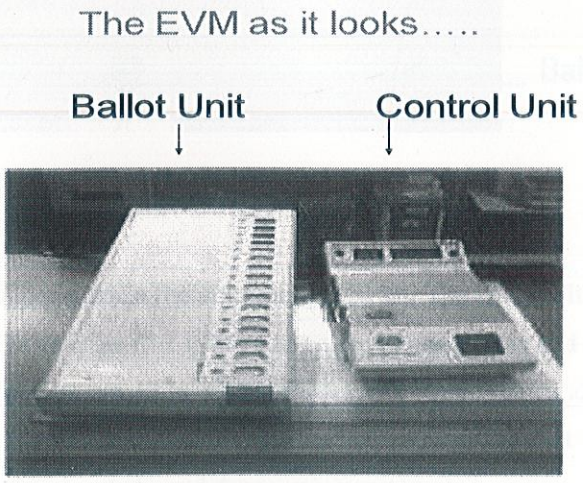


Fig. 2.1: Overview [7]

It has mainly 2 units: Ballot and Control Units. Itoperates on a special battery and is tamper-proof and is easily portable. Information recorded is retained in memory even when the battery is removed. Manufactured by Electronics Corporation of India & Bharat Electronics Ltd. and approved by Election Commission, India. Each EVM can cater to a maximum of 64 candidates with 4 Ballot Units cascaded.The EVM can be used for conducting "TWO" simultaneous polls.

2.3 Ballot Units Details

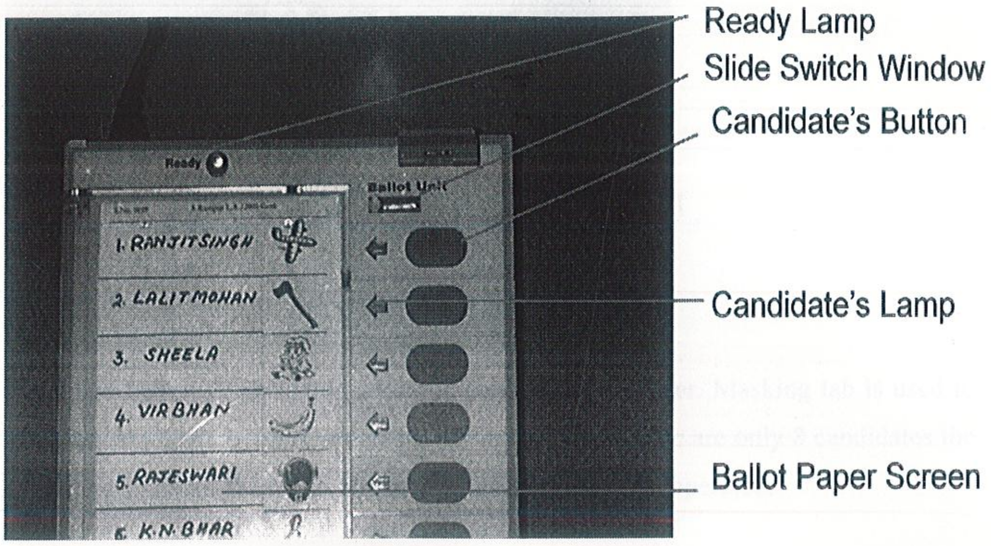


Fig. 2.2: Ballot Unit Details [7]

The ready lamp glows when the unit is switched on. The slide switch is used to set the no. of the unit, i.e. it is set to 1 if there are only 16 candidates and 1 for the first and 2 for the second if there are 17 to 32 candidates and so on.The candidate's lamp glows indicating to the voter that his/her vote is cast in favor of that candidate.After the ballot paper is placed and aligned the screen is put in place and sealed.

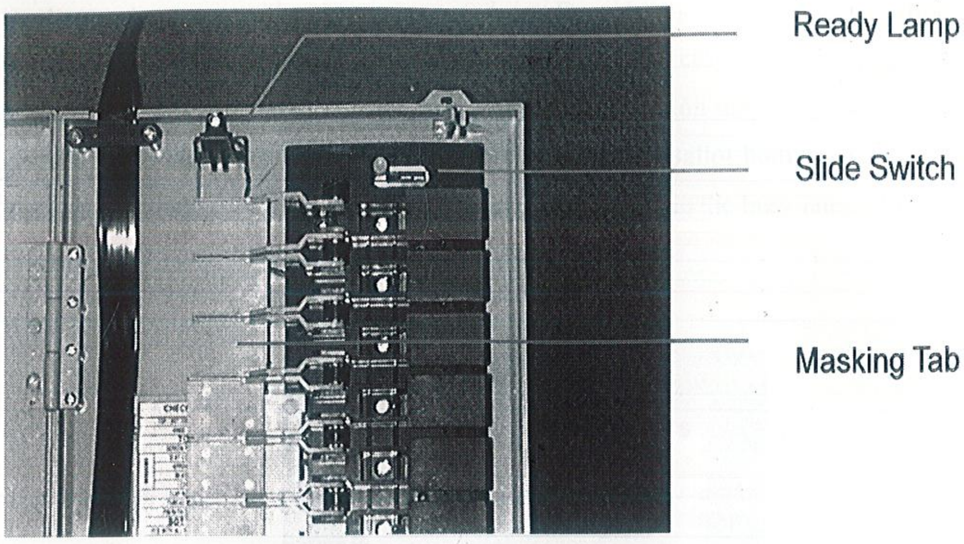


Fig. 2.3: Ballot Unit-Internal Parts [7]

Candidate button is the button which is pressed by the voter. Masking tab is used to mask the candidate buttons which are not in use, i.e. if there are only 8 candidates the remaining switches from 9 to 16 are masked and cannot be operated.

2.4 Control Unit

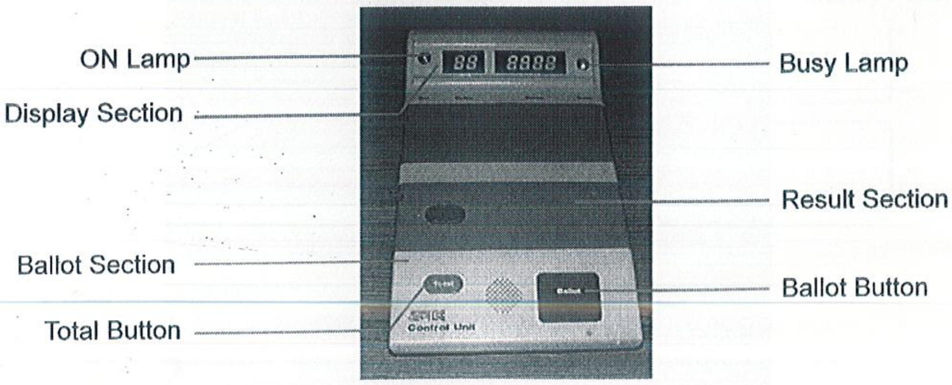


Fig. 2.4: Control Unit [7]

On lamp glows when the unit is powered on. Busy Lamp glows when a ballot is released and a voter is in the process of voting. After the casting the lamp goes off with a beep thus indicating that the vote is cast. Total button may be pressed at any given time to know the total no of votes polled till then. Ballot button – pressing of this button releases a vote in the ballot unit and also results in the busy lamp glowing.

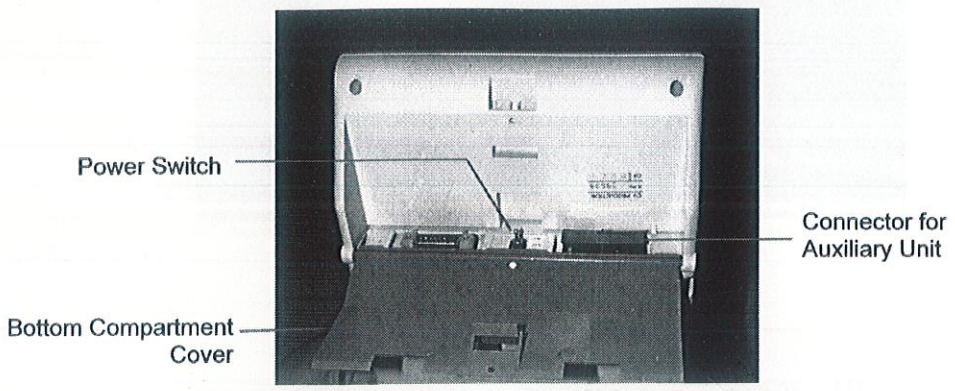


Fig. 2.5: View Of Bottom Compartment [7]

Power Switch powers on / off the EVM. Connecter is for connecting the Ballot unit with interconnecting cable. Auxiliary unit connector is used to connect second ballot unit in case of two simultaneous polls.

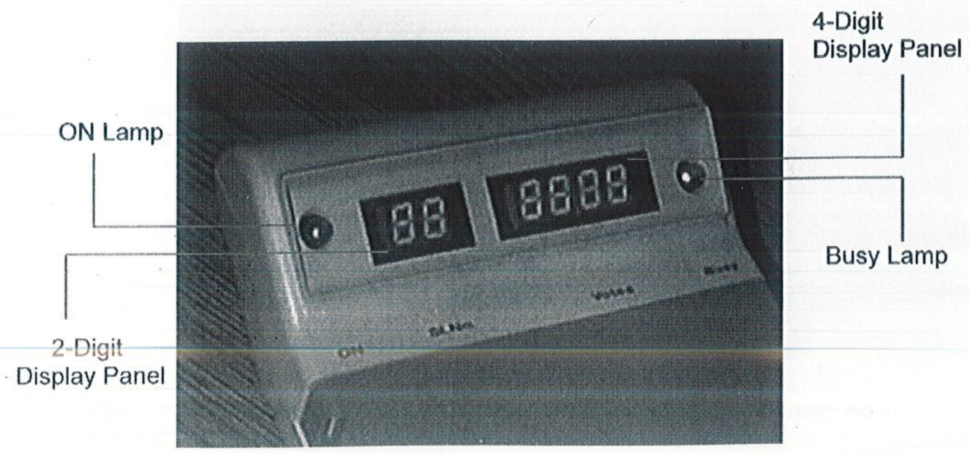


Fig. 2.6: Control Unit Display Section [7]

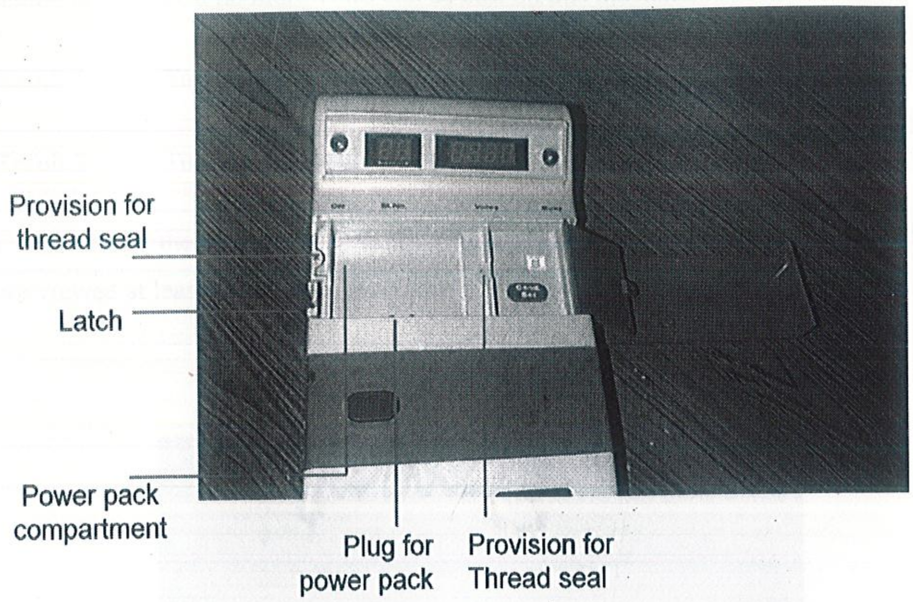


Fig. 2.7: Control Unit-Candidate Set Section [7]

Candidate set button: For setting the no. of candidates in the poll. The Ballot unit and the control unit are connected and powered on. This button is pressed and the candidate button on the ballot unit corresponding to the last candidate is pressed.

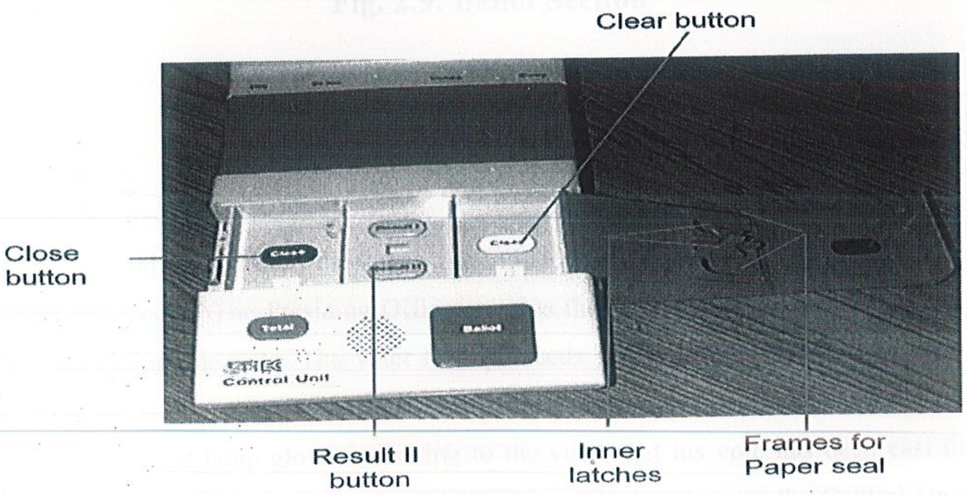


Fig. 2.8: Result Section [7]

Close button is used to close the poll at the end of the appointed period. Once this button is depressed no more votes can be cast on this machine.

Result 1:	To view the results of poll 1
Result 2:	To view the results of poll 2

Clear: Clears the data recorded in the voting machine – operable only after the results are viewed at least once

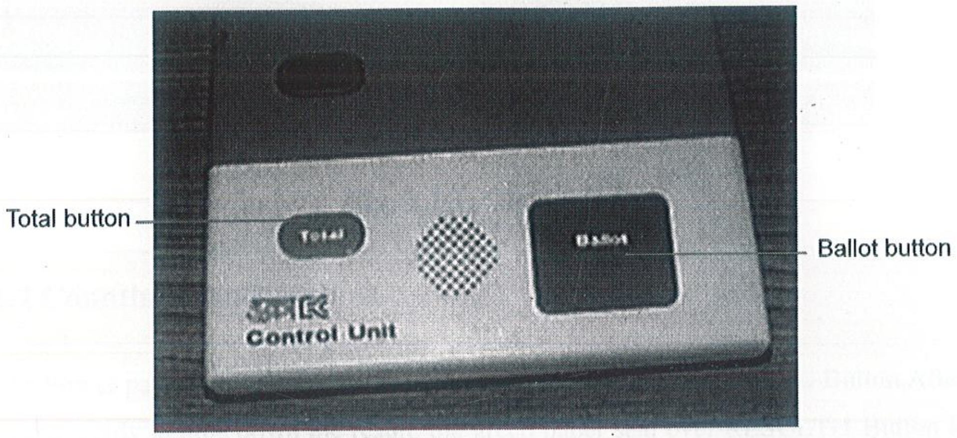


Fig. 2.9: Ballot Section [7]

2.5 Polling

The voter is identified from the voters list and records his presence by a signature or thumb impression. The Presiding Officer presses the "Ballot" button on the Control Unit permitting one vote. The voter then proceeds to the polling cubicle and after perusing the ballot paper on the Ballot Unit, presses the key against the candidate of his choice. A red lamp glows indicating to the voter that his vote has been cast in favour of that candidate. The casting of the vote results in a beep in the Control Unit indicating to the Presiding Officer that a vote has been cast. He then proceeds to release another vote by pressing the "Ballot" button and the process continues.

2.6 Closing

The cap on the "Close Button" is removed and the button pressed. The cap is then replaced. The unit is then switched "Off" and the interconnecting cable disconnected.

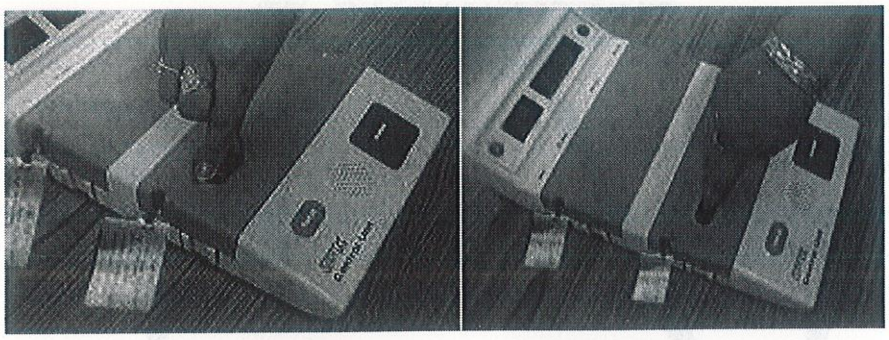


Fig. 2.10: Closing [7]

2.7 Counting And Result

The Power pack / Battery is checked for health by pressing the TOTAL Button.After getting ready to note down the result, the green paper seal over RESULT-1 Button is pierced and RESULT-1 Button is pressed to display the results.

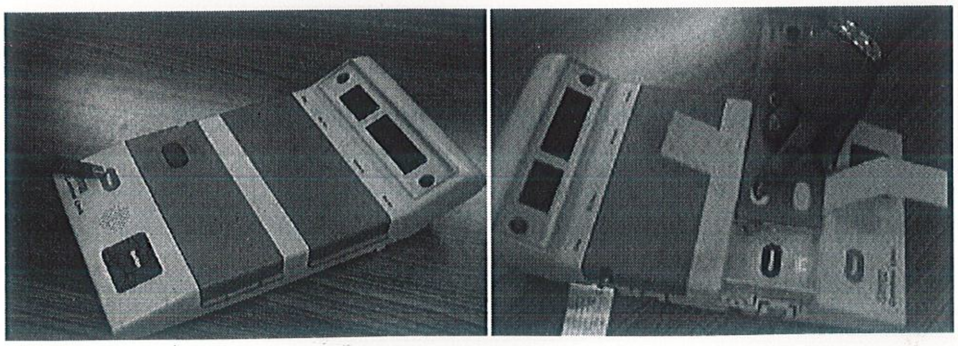


Fig. 2.11: Counting And Result [7]

The results are then noted.

2.8 Sequence Of Operations Of Buttons

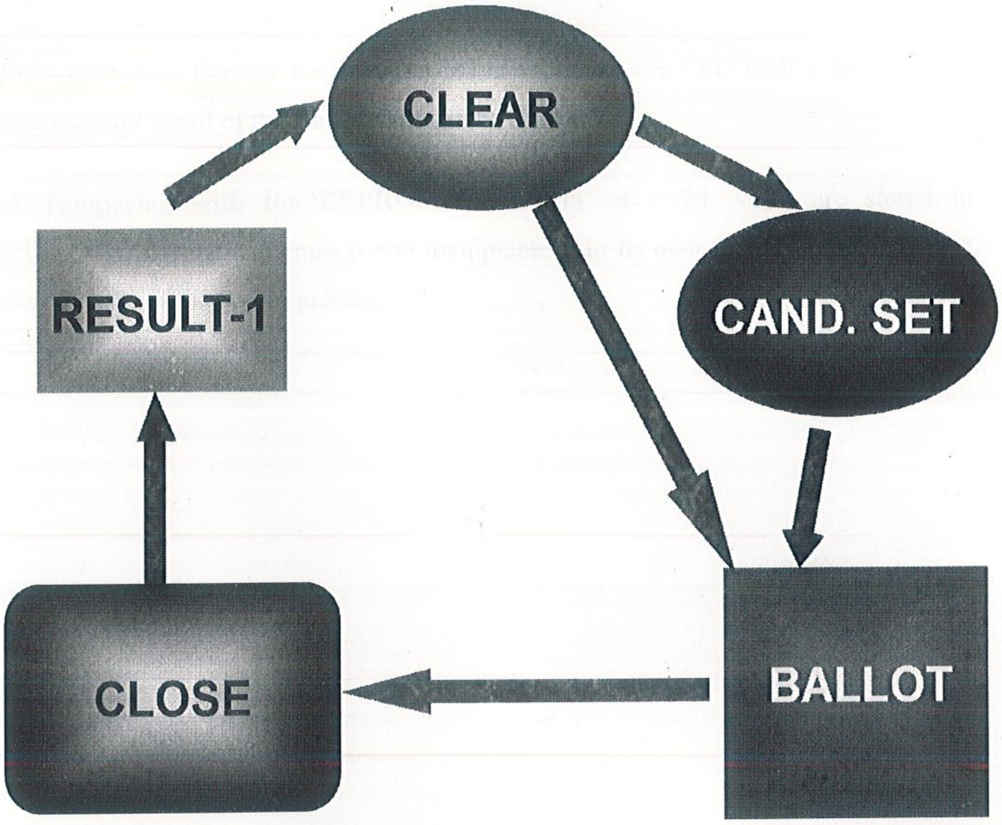


Fig. 2.12: Sequence Of Operation Buttons [8]

2.9 Attacks On EVM

a) Substituting Look-Alike CPUs: Since code burned in the CPU cannot be verified, anyone can change the CPU with coded CPU working in a dishonest way.

b) Substituting Circuit Boards: Instead of replacing the CPU with a fake one the entire circuit board of the control unit can be changed.

c) Tampering with the EEPROMs: Since in an EVM, votes are stored in EEPROMs, anyone can erase it and manipulate it in its own way, thereby changing the outcome of the voting process.

CHAPTER 3

PUBLIC KEY CRYPTOGRAPHY

3.1 Introduction

The data transferred from one system to another over public network can be protected by the method of encryption. On encryption the data is encrypted by any encryption algorithm using the 'key'. Only the user having the access to the same 'key' can decrypt the encrypted data. This method is known as private key or symmetric key cryptography. There are several standard symmetric key algorithms defined. Examples are AES, 3DES etc. These standard symmetric algorithms defined are proven to be highly secured and time tested. But the problem with these algorithms is the key exchange. The communicating parties require a shared secret, 'key', to be exchanged between them to have a secured communication. The security of the symmetric key algorithm depends on the secrecy of the key. Keys are typically of a large number of bits in length, depending on the algorithm used. Since there may be number of intermediate points between the communicating parties through which the data passes, these keys cannot exchanged online in a secured manner. In a large network, where there are hundreds of system connected, offline key exchange seems too difficult and even unrealistic. This is where public key cryptography comes to help. Using public key algorithm a shared secret can be established online between communicating parties without the need for exchanging any secret data.

In public key cryptography each user or the device taking part in the communication have a pair of keys, a public key and a private key, and a set of operations associated with the keys to do the cryptographic operations. Only the particular user/device knows the private key whereas the public key is distributed to all users/devices taking part in the communication. Since the knowledge of public key does not compromise the security of the algorithms, it can be easily exchanged online.

A shared secret can be established between two communicating parties online by exchanging only public keys. Any third party, who has access only to the exchanged public information, will not be able to calculate the shared secret unless it has access to the private key of any of the communicating parties.

3.2 Key Agreement

Key agreement is a method in which the device communicating in the network establishes a shared secret between them without exchanging any secret data. In this method the devices that need to establish shared secret between them exchange their public keys. Both the devices on receiving the other device's public key performs key generation operation using its private key to obtain the shared secret.

Let P be the private key of a device and U(P, C) be the public key. Since public key is generated using private key, the representation U(P, C) shows that the public key contain the components of private key P and some constants C where C is known by all the device taking part in the communication.

Consider two devices A and B. Let PA and UA(PA, C) be the private key and public key of device A, and PB and UB(PB, C) be the private key and public key of device B respectively. Both device exchanges their public keys. Device A, having got the public key of B, uses its private key to calculate shared secret;

KA=Generate_Key (PA, UB (PB, C))

Device B, having got the public key of A, uses its private key to calculate the shared secret;

KB=Generate_Key (PB, UA (PA, C))

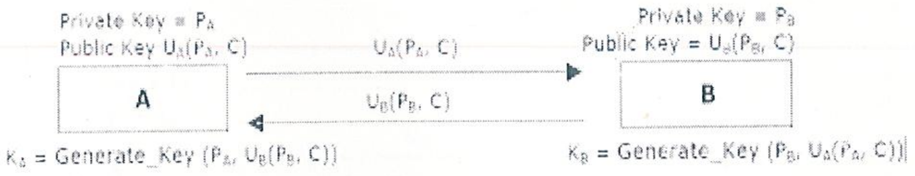

Private Key = P_A
Public Key $U_A(P_A, C)$
$U_A(P_A, C)$
Private Key = P_B
Public Key = $U_B(P_B, C)$
A
$U_B(P_B, C)$
B
K_A = Generate_Key (P_A, $U_B(P_B, C)$)
K_B = Generate_Key (P_B, $U_A(P_A, C)$)

Fig. 3.1 : Key Agreement [6]

The key generation algorithm 'Generate_Key' will be such that the generated keys at the device A and B will be the same, that is shared secret KA=KB=K(PA, PB, C). Since it is practically impossible to obtain private key from the public key any middleman, having access only to the public keys UA (PA, C) and UB (PB, C), will never be able to obtain the shared secret K.

3.3 Encryption

Encryption is a process in which the sender encrypts/scrambles the message in such a way that only the recipient will be able to decrypt/ descramble the message.
Consider a device B whose private key and public key are PB and UB respectively. Since UB is public key all devices will be able to get it. For any device that needs to send the message 'Msg' in a secured way to device B, it will encrypt the data using

B's public key to obtain the cipher text 'Ctx'. The encrypted message, cipher text, can only be decrypted using B's private key. On receiving the message the B decrypts it using its private key PB. Since only B knows its private key PB none other including A can decrypt the message.

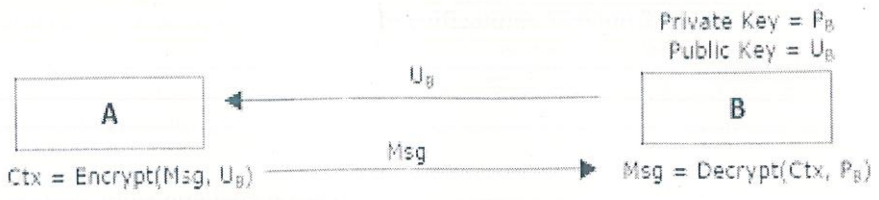


Fig. 3.2: Encryption [6]

3.4 Digital Signature

Using Digital signature a message can be signed by a device using its private key to ensure authenticity of the message. Any device that has got the access to the public key of the signed device can verify the signature. Thus the device receiving the message can ensure that the message is indeed signed by the intended device and is not modified during the transit. If any the data or signature is modified, the signature verification fails.

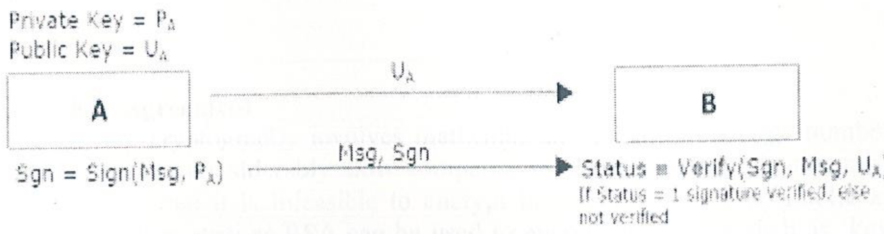

Private Key = P_A
Public Key = U_A
U_A
A
B
Msg, Sgn
Sgn = Sign(Msg, P_A)
Status = Verify(Sgn, Msg, U_A)
If Status = 1 signature verified, else not verified

Fig. 3.3: Digital Signature [6]

For e.g. if a device A need to ensure the authenticity of its message, the device A signs its message using its private key PA. The device A will then send the message 'Msg' and signature 'Sgn' to device B. The device B, on receiving the message, can verify the message using A's public key UA and thereby ensuring that the message is indeed sent by A and is also not tampered during the transit. Since only the device A knows its private PA key, it is impossible for any other device to forge the signature.

3.5 RSA

RSA is a public key algorithm that is used for Encryption, Signature and Key Agreement. RSA typically uses keys of size 1024 to 2048. The RSA standard is specified RFC 3447, RSA Cryptography Specifications Version 2.1.

3.5.1 RSA Encryption

Parameter generation

R1. Select two prime numbers p and q.

R2. Find n=p*q, Where n is the modulus that is made public. The length of n is considered as the RSA key length.

R3. Choose a random number e as a public key in the range 0<e< (p-1) (q-1) such that gcd(e,(p-1)(q-1))=1.

R4. Find private key d such that ed=1(mod (p-1)(q-1)).

Encryption

Consider the device A that needs to send a message to B securely.

R5. Let e be B′s public key. Since e is public, A has access to e.

R6. To encrypt the message M, represent the message as an integer in the range 0<M<n.

R7. Cipher text C = M^e mod n, where n is the modulus.

Decryption

R8. Let C be the cipher text received from A.

R9. Calculate Message M = C^d mod n, where d is B′s private key and n is the modulus.

3.5.2 RSA Key Agreement

Since public key cryptography involves mathematical operation on large numbers, these algorithms are considerably slow compared to the symmetric key algorithm. They are so slow that it is infeasible to encrypt large amount of data. Public key encryption algorithm such as RSA can be used to encrypt small data such as 'keys' used in private key algorithm. RSA is thus used as key agreement algorithm.

Key agreement algorithm

For establishing shared secret between two device A and B

R10. Generate a random number, key, at device A.

R11. Encrypt key by RSA encryption algorithm using B's public key and pass the cipher text to B

R12. At B decrypt the cipher text using B's private key to obtain the key.

Key Generation	
Select p, q	p, q both prime, $p \neq q$
Calculate $n = p \times q$	
Calculate $\phi(n) = (p-1) \times (q-1)$	
Select integer e	$\gcd(\phi(n), e) = 1;\ 1 < e < \phi(n)$
Calculate d	
Public key	$KU = \{e, n\}$
Private key	$KR = \{d, n\}$

Encryption	
Plaintext:	$M < n$
Ciphertext:	$C = M^e \pmod{n}$

Decryption	
Ciphertext:	C
Plaintext:	$M = C^d \pmod{n}$

Fig. 3.4 : RSA Algorithm [6]

3.6 Rabin-Miller Primality Test

The **Miller–Rabin primality test** or **Rabin–Miller primality test** is a primality test which determines whether a given number is prime.

First, a proposition about square roots of unity in the finite field $\mathbb{Z}/p\mathbb{Z}$, where p is prime and $p > 2$. Certainly 1 and −1 always yield 1 when squared mod p; call these trivial square roots of 1. There are no *nontrivial* square roots of 1 mod p (a special case of the result that, in a field, a polynomial has no more zeroes than its degree). To show this, suppose that x is a square root of 1 mod p. Then:

$$x^2 \equiv 1 \pmod{p}$$
$$(x-1)(x+1) \equiv 0 \pmod{p}.$$

In other words, p divides the product $(x-1)(x+1)$. It thus divides one of the factors and it follows that x is either congruent to 1 or -1 mod p.

Now, let n be an odd prime. Then $n-1$ is even and we can write it as $2^s \cdot d$, where s and d are positive integers (d is odd). For each $a \in (\mathbb{Z}/n\mathbb{Z})^*$, either

$$a^d \equiv 1 \pmod{n}$$

or

$$a^{2^r d} \equiv -1 \pmod{n} \text{for some } 0 \leq r \leq s-1.$$
$$a^{n-1} \equiv 1 \pmod{n}.$$

By the proposition above, if we keep taking square roots of a^{n-1}, we will get either 1 or -1. If we get -1 then the second equality holds and we are done. If we never get -1, then when we have taken out every power of 2, we are left with the first equality.

If we can find an a such that

$$a^d \not\equiv 1 \pmod{n}$$

and

$$a^{2^r d} \not\equiv -1 \pmod{n} \text{for all } 0 \leq r \leq s-1$$

then n is not prime. We call a a witness for the compositeness of n (sometimes misleadingly called a *strong witness*, although it is a certain proof of this fact). Otherwise a is called a *strong liar*, and n is a strong probable prime to base a. The term "strong liar" refers to the case where n is composite but nevertheless the equations hold as they would for a prime.

Example

Suppose we wish to determine if $n = 221$ is prime. We write $n-1 = 220$ as $2^2 \cdot 55$, so that we have $s = 2$ and $d = 55$. We randomly select a number a such that $a < n$, say $a = 174$. We proceed to compute:

- $a^{2^0 \cdot d} \bmod n = 174^{55} \bmod 221 = 47 \neq 1, n-1$
- $a^{2^1 \cdot d} \bmod n = 174^{110} \bmod 221 = 220 = n-1.$

Since $220 \equiv -1 \bmod n$. We try another random a, this time choosing a=137:

- $a^{2^0 \cdot d} \bmod n = 137^{55} \bmod 221 = 188 \neq 1, n-1$
- $a^{2^1 \cdot d} \bmod n = 137^{110} \bmod 221 = 205 \neq n-1.$

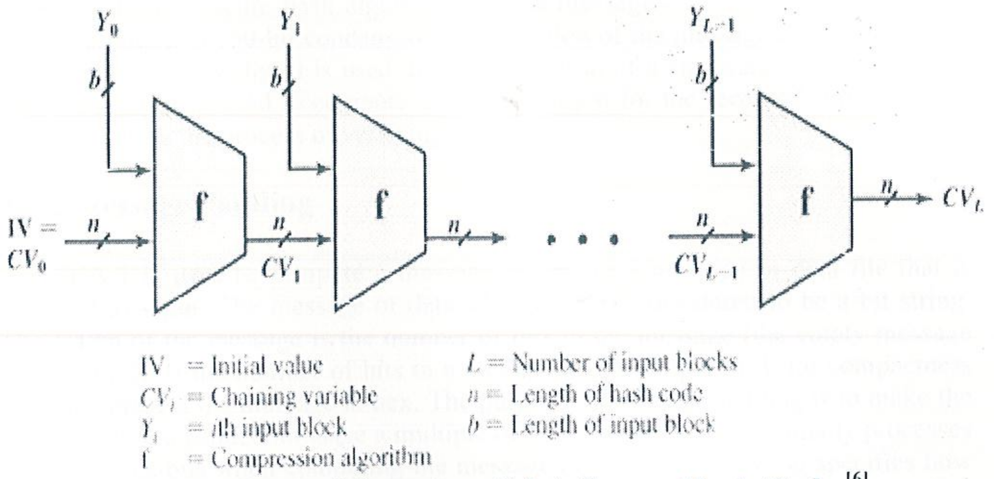
Hence 137 is a witness for the compositeness of 221. So, 221 is not prime.

CHAPTER 4

HASH FUNCTIONS

4.1 Definition

A **hash function** is an algorithm that maps large data sets to smaller data sets. The values returned by a hash function are called **hash values**. A hash function accepts a variable-size message M as input and produces a fixed-size output, referred to as a hash code H(M).



Y_0
Y_1
Y_{L-1}
b
n
f
IV =
CV_0
CV_1
CV_{L-1}
CV_L
IV = Initial value
CV_i = Chaining variable
Y_i = ith input block
f = Compression algorithm
L = Number of input blocks
n = Length of hash code
b = Length of input block

Fig. 4.1 : General Structure Of A Secure Hash Code [6]

The hash algorithm involves repeated use of a compression function, f, that takes two inputs (ann -bit input from the previous step, called the chaining variable, and a b-bit block) and produces an n-bit output. At the start of hashing, the chaining variable has an initial value that is specified as part of the algorithm. The final value of the chaining variable is the hash value. Often, b > n; hence the term compression.

The hash function can be summarized as follows:
CVo = IV = initial n-bit value
CVi = f(CVi1, Yi1) 1 i L
H(M) = CVL
where the input to the hash function is a message M consisting of the blocks Yo, Y1,..., YL1.

4.2 Properties Of Hash Functions

The ideal cryptographic hash function has four main or significant properties:

- it is easy to compute the hash value for any given message.
- it is infeasible to generate a message that has a given hash.
- it is infeasible to modify a message without changing the hash.
- it is infeasible to find two different messages with the same hash.

4.3 SHA-1

4.3.1 Introduction

SHA stands for "secure hash algorithm". For a message of length < 2^64 bits, the SHA-1 produces a 160-bit condensed representation of the message called a message digest. The message digest is used during generation of a signature for the message. The SHA-1 is also used to compute a message digest for the received version of the message during the process of verifying the signature.

4.3.2 Message Padding

The SHA-1 is used to compute a message digest for a message or data file that is provided as input. The message or data file should be considered to be a bit string. The length of the message is the number of bits in the message (the empty message has length 0). If the number of bits in a message is a multiple of 8, for compactness we can represent the message in hex. The purpose of message padding is to make the total length of a padded message a multiple of 512. The SHA-1 sequentially processes blocks of 512 bits when computing the message digest. The following specifies how this padding shall be performed. As a summary, a "1" followed by m "0"s followed by a 64-bit integer are appended to the end of the message to produce a padded message of length 512 * n. The 64-bit integer is i. the length of the original message. The padded message is then processed by the SHA-1 as n 512-bit blocks.

Suppose a message has length $l < 2^{64}$. Before it is input to the SHA-1, the message is padded on the right as follows:

a. "1" is appended.

Example: if the original message is "01010000", this is padded to "010100001".

b. "0"s are appended. The number of "0"s will depend on the original length of the message. The last 64 bits of the last 512-bit block are reserved for the length l of the original message.

Example: Suppose the original message is the bit string

01100001 01100010 01100011 01100100 01100101.

After step (a) this gives
01100001 01100010 01100011 01100100 01100101 1.
Since l = 40, the number of bits in the above is 41 and 407 "0"s are appended, making the total now 448. This gives (in hex)
61626364 65800000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000.

c. Obtain the 2-word representation of l, the number of bits in the original message. If $l < 2^{32}$ then the first word is all zeroes. Append these two words to the padded message.

Example: Suppose the original message is as in (b). Then l = 40 (note that l is computed before any padding). The two-word representation of 40 is hex 00000000 00000028. Hence the final padded message is hex
61626364 65800000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000028.

The padded message will contain 16 * n words for some n > 0. The padded message is regarded as a sequence of n blocks M_1 , M_2, ... , M_n, where each M_i contains 16 words and M_1 contains the first characters (or bits) of the message.

4.3.3 Computing The Message Digest

The message digest is computed using the final padded message. The computation uses two buffers, each consisting of five 32-bit words, and a sequence of eighty 32-bit words. The words of the first 5-word buffer are labeled A,B,C,D,E. The words of the second 5-word buffer are labeled H_0, H_1, H_2, H_3, H_4. The words of the 80-word sequence are labeled W_0, W_1,..., W_{79}. A single word buffer TEMP is also employed.

To generate the message digest, the 16-word blocks M_1, M_2,..., M_n defined in Section 4 are processed in order. The processing of each M_i involves 80 steps.

Before processing any blocks, the $\{H_i\}$ are initialized as follows: in hex,

H_0 = 67452301

H_1 = EFCDAB89

H_2 = 98BADCFE

H_3 = 10325476

H_4 = C3D2E1F0.

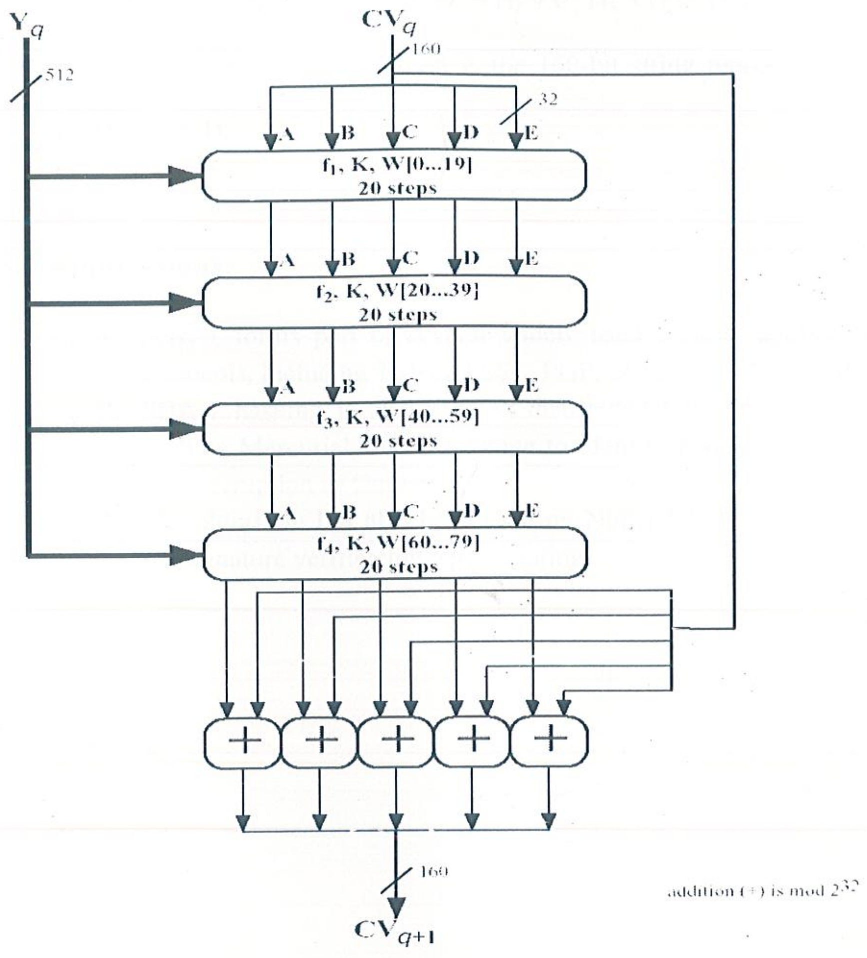


Fig. 4.2: SHA-1 Processing Of A Single 512 Bit Block (SHA-1 Compression Function) [6]

Now $M_1, M_2, \ldots, M_n$ are processed. To process M_i, we proceed as follows:

a. Divide M_i into 16 words $W_0, W_1, \ldots, W_{15}$, where W_0 is the left-most word.

b. For $t = 16$ to 79 let $W_t = S^1(W_{t-3} \text{ XOR } W_{t-8} \text{ XOR } W_{t-14} \text{ XOR } W_{t-16})$.

c. Let $A = H_0$, $B = H_1$, $C = H_2$, $D = H_3$, $E = H_4$.

d. For $t = 0$ to 79 do
$TEMP = S^5(A) + f_t(B,C,D) + E + W_t + K_t$;

$E = D; D = C; C = S^{30}(B); B = A; A = TEMP;$

e. Let $H_0 = H_0 + A$, $H_1 = H_1 + B$, $H_2 = H_2 + C$, $H_3 = H_3 + D$, $H_4 = H_4 + E$.

After processing M_n, the message digest is the 160-bit string represented by the 5 words

$H_0\ H_1\ H_2\ H_3\ H_4$.

4.3.4 Applications

1. SHA-1 forms part of several widely used security applications and protocols, including TLS and SSL, PGP, SSH, S/MIME, and IPsec.
2. SHA-1 hashing is also used in distributed revision control systems such as Mercurial, and Monotone to identify revisions, and to detect data corruption or tampering.
3. The algorithm has also been used on Nintendo's Wii gaming console for signature verification when booting.

CHAPTER 5

AUTHENTICATION

5.1 One Way Ciphers

A one-way cipher is an irreversible function f from plaintext to ciphertext. It is computationally infeasible to systematically determine a plaintext message M from the ciphertext C = f(M).

One-way ciphers are used in applications that do not require deciphering the data. One such class of applications involves determining whether there is a correspondence between a given message M and a ciphertext C stored in the system.This correspondence is determined by computing f(M), and comparing the result with C. For this to be effective, f should be one-to-one, or at least not too degenerate. Otherwise, a false message M' may pass the test f(M') = C.

A one-way cipher can be implemented using a computationally secure block encryption algorithm E by letting

f(M) = EM(Mo),

where Mo is any given, fixed message. The message M serves as the key to E. As long as E is secure, it is computationally infeasible to determine the enciphering key M with a known plaintext attack by examining pairs (M0, EM(Mo)).

5.2 Password Storage

To avoid storage of passwords, some operating systems (e.g. UNIX, LINUX) store a hash of the password rather than storing the password itself. During authentication, the system verifies that the hash of the password entered matches the hash stored in the password database. If the intruder somehow obtains a password hash, he or she can use any password that generates the same hash. However, the intruder still needs to produce such a password, which, depending on the hashing algorithm strength, may be a very difficult cryptographic problem.

Often, the hashed password is retrieved from the actual password together with a **password salt**. Then, the hash together with the salt is stored.

5.3 Login Protocol

Suppose a user logs into the system and supplies password P. If P is transmitted from the user's terminal to the system in the clear, it could be compromised on the way (e.g., by wiretapping). It may not even make it to the system: a program masquerading as the login procedure might trick the user into typing ID and P.

As a solution to this problem, Feistel, Notz, and Smith describe a login procedure that does not expose the user's password, and allows the user and system to mutually authenticate each other. They assume each user A has a private key on some digital storage medium (e.g., a magnetic-stripe card) which can be inserted into A's terminal, and that a copy of the key is stored on file at the system. To log into the system, A transmits ID in the clear (for simplicity, we assume ID = A). The system responds with a "challenge-reply" test that allows A to determine whether the communication is "live" (and not a replay of an earlier login), and allows the system to establish A's authenticity.

5.3.1 Login Protocol Using Passwords

1. A transmits ID = A to S.

2. S sends to A:

x = EA(T), where T is the current date and time, and EA is the enciphering transformation derived from A's private key.

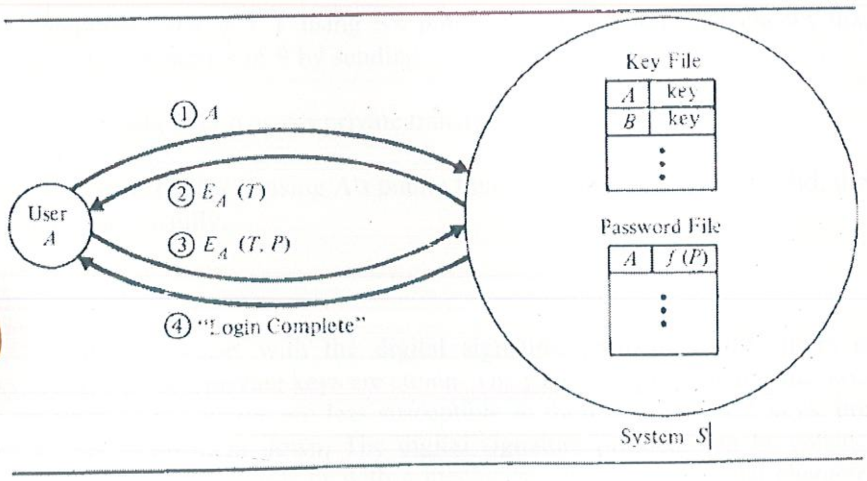


Fig. 5.1: Login Protocol Using Passwords [11]

3. A deciphers X to get T, and checks that T is current. If it is, A replies to S by sending

Y = EA(T , P) where P is A's password.

4. S deciphers Y to get T and P. It checks T against the time transmitted to A in Step 2,and checks f(P) against the password file. If both check, the login completes successfully.

5.3.2 Login Protocol Using Digital Signature

The above protocol is easily modified for a public-key system. In Step 2, the system S uses its private transformation D s (for sender authenticity) to create X = Ds(T), which A can validate using S's public transformation E s. In Step 3, A uses the system's public enciphering transformation (for secrecy) to create Y = Es(T, P). Only S can decipher Y to obtain A's password and complete the login. Note that A's private transformation (i.e., digital signature) can be used for authentication instead of A's password; in this case, the protocol becomes:

1. A transmits ID = A to S.

2. S sends to A:

X= Ds(T) , where T is the current date and time, and D s is S's private transformation.

3. A computes Es(X) = T using S's public transformation, and checks that T is current. If it is, A replies to S by sending

Y = DA(T), where D A is A's private transformation.

4. The system validates Y using A's public transformation E A. If it is valid, the login completes successfully.

One possible weakness with the digital signature protocol is that users can be impersonated if their private keys are stolen. The password protocol has the advantage that memorized passwords are less susceptible to theft than physical keys, provided that memorized passwords are less susceptible to theft than physical keys, provided users do not write them down. The digital signature protocol can be enhanced by combining it with passwords or with a mechanism that uses personal characteristics (e.g., a handprint) for identification. If passwords are used, then A sends to S DA(T), Es(T, P) in Step 3 of the protocol.

CHAPTER 6

THRESHOLD CRYPTOGRAPHY

6.1 Introduction

In cryptography, a cryptosystem is called a 'threshold cryptosystem', if in order to decrypt an encrypted message a number of parties exceeding a threshold is required to cooperate in the decryption protocol. The message is encrypted using a public key and the corresponding private key is shared among the participating parties. Let n be the number of parties. Such a system is called *(t,n)*-threshold, if at least t of these parties can efficiently decrypt the ciphertext, while less than t have no useful information. Similarly it is possible to define *(t,n)*-threshold signature scheme, where at least t parties are required for creating a signature.

Threshold versions of encryption schemes can be built for many public encryption schemes. The natural goal of such schemes is to be as secure as the original scheme. Such threshold versions have been defined for:

- RSA
- Pallier cryptosystem
- Damgård–Jurjk cryptosystem
- El-Gamal

6.2 Definition

-n parties share the ability of performing a cryptographic operation (e.g. creating a digital signature)

-any t of those n parties can perform the operation jointly

-any t-1 (or less) parties cannot perform the operation

6.3 Description

For t out of n parties to construct a secret :

f(x) = a(0) + a(1)*x + . . . + a(t-1)*x^(t-1)

a(0) : secret

a(1)...a(t-1): random numbers

share s(i) : (x(i),y(i)) with y(i) = f(x(i)) ; x(i) ≠0

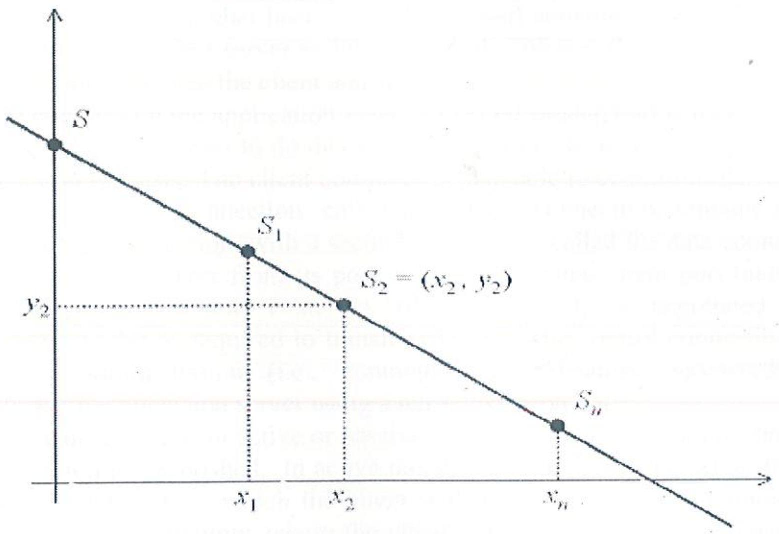


Fig. 6.1: Secret Sharing (Threshold) [2]

Any 2 points determine the line, hence the secret f(0)

With just 1 point, the secret could be any point on the y –axis

CHAPTER 7

SECURE TRANSMISSION

7.1 FTP

7.1.1 Introduction

File Transfer Protocol (FTP) is a standard network protocol used to transfer files from one host to another host over a TCP-based network, such as the Internet. FTP is built on a client-server architecture and utilizes separate control and data connections between the client and server.

FTP operates on the application layer of the OSI model, and is used to transfer files using TCP/IP. In order to do this an FTP server needs to be running and waiting for incoming requests. The client computer is then able to communicate with the server on port 21. This connection, called the control connection, remains open for the duration of the session, with a second connection, called the data connection, either opened by the server from its port 20 to a negotiated client port (active mode) or opened by the client from an arbitrary port to a negotiated server port (passive mode) as required to transfer file data. The control connection is used for session administration (i.e., commands, identification, passwords) exchanged between the client and server using a telnet-like protocol.

FTP can be run in active or passive mode, which determine how the data connection is established. In active mode, the client sends the server the IP address and port number on which the client will listen, and the server initiates the TCP connection. In situations where the client is behind a firewall and unable to accept incoming TCP connections, passive mode may be used. In this mode the client sends a PASV command to the server and receives an IP address and port number in return. The client uses these to open the data connection to the server.

FTP Control Connection

- Must be functioning for Data transfer to occur.
- Control connection utilized the TELNET protocol.
- Special FTP commands and responses — the FTP Protocol
- Text (ASCI) Command line oriented

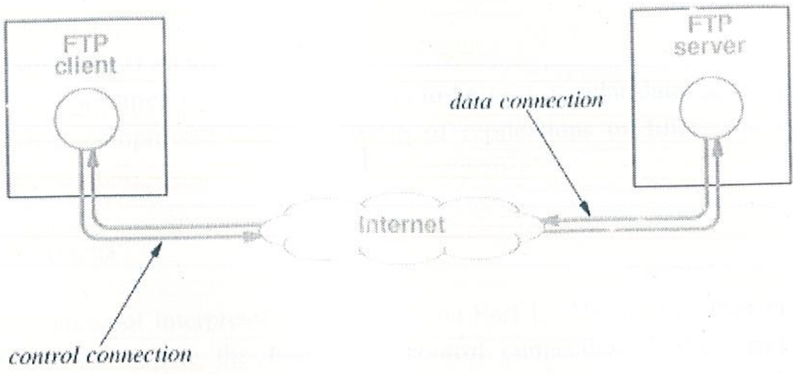


Fig. 7.1: Structure Of FTP [8]

While transferring data over the network, four data representations can be used:

- ASCII mode: used for text. Data is converted, if needed, from the sending host's character representation to "8-bit ASCII" before transmission, and (again, if necessary) to the receiving host's character representation. As a consequence, this mode is inappropriate for files that contain data other than plain text.
- Image mode (commonly called Binary mode): the sending machine sends each file byte for byte, and the recipient stores the byte stream as it receives it. (Image mode support has been recommended for all implementations of FTP).
- EBCDIC mode: use for plain text between hosts using the EBCDIC character set. This mode is otherwise like ASCII mode.
- Local mode: Allows two computers with identical setups to send data in a proprietary format without the need to convert it to ASCII

7.1.2 Transmission Modes

There are three modes: one which formats the data and allows for restart procedures; one which also compresses the data for efficient transfer; and one which passes the data with little or no processing. In this last case the mode interacts with the structure attribute to determine the type of processing. In the compressed mode, the representation type determines the filler byte. All data transfers must be completed with an end-of-file (EOF) which may be explicitly stated or implied by the closing of the data connection.

- STREAM MODE:
  The data is transmitted as a stream of bytes. There is no restriction on the representation type used; record structures are allowed.

- BLOCK MODE

  The file is transmitted as a series of data blocks preceded by one or more header bytes.
- COMPRESSED MODE

  There are three kinds of information to be sent: regular data, sent in a byte string; compressed data, consisting of replications or filler; and control information, sent in a two-byte escape sequence.

7.1.3 Connections

The server protocol interpreter shall "listen" on Port L. The user or user protocol interpreter shall initiate the full-duplex control connection. Server- and user-processes should follow the conventions of the Telnet protocol as specified in the ARPA-Internet Protocol Handbook. Servers are under no obligation to provide for editing of command lines and may require that it be done in the user host. The control connection shall be closed by the server at the user's request after all transfers and replies are completed.

When data is to be transferred between two servers, A and B the user-PI, C, sets up control connections with both server-PI's. One of the servers, say A, is then sent a PASV command telling him to "listen" on his data port rather than initiate a connection when he receives a transfer service command. When the user-PI receives an acknowledgment to the PASV command, which includes the identity of the host and port being listened on, the user-PI then sends A's port, a, to B in a PORT command; a reply is returned. The user-PI may then send the corresponding service commands to A and B. Server B initiates the connection and the transfer proceeds

The data connection is to be closed following a data transfer where closing the connection is not required to indicate the end-of-file, the server must do so immediately. Waiting until after
a new transfer command is not permitted because the user-process will have already tested the data connection to see if it needs to do a "listen".

7.2 SSL

7.2.1 Introduction

The Secure Sockets Layer (SSL) is a commonly-used protocol for managing the security of a message transmission on the Internet. SSL uses a program layer located between the Internet's Hypertext Transfer Protocol (HTTP) and Transport Control Protocol (TCP) layers. SSL is included as part of both the Microsoft and Netscape browsers and most Web server products. The "sockets" part of the term refers to the sockets method of passing data back and forth between a client and a server program in a network or between program layers in the same computer. SSL uses the public-and-private key encryption system from RSA, which also includes the use of a digital certificate.

The SSL protocol allows client/server applications to communicate across a network in a way designed to prevent eavesdropping and tampering. Since most protocols can be used either with or without SSL it is necessary to indicate to the server whether the client is making a SSL connection or not. There are two main ways of achieving this, one option is to use a different port number for SSL connections. The other is to use to the regular port number and have the client request that the server switch the connection to SSL using a protocol specific mechanism.

7.2.2 Handshake

During this handshake, the client and server agree on various parameters used to establish the connection's security.

- The handshake begins when a client connects to a SSL-enabled server requesting a secure connection and presents a list of supported Cipher Suites(ciphers and hash functions).
- From this list, the server picks the strongest cipher and hash function that it also supports and notifies the client of the decision.
- The server sends back its identification in the form of a digital certificate. The certificate usually contains the server name, the trusted certificate authority(CA) and the server's public_encryption key.
- The client may contact the server that issued the certificate (the trusted CA as above) and confirm the validity of the certificate before proceeding.
- In order to generate the session keys used for the secure connection, the client encrypts a random number with the server's public key and sends the result to the server. Only the server should be able to decrypt it, with its private key.

- From the random number, both parties generate key material for encryption and decryption.

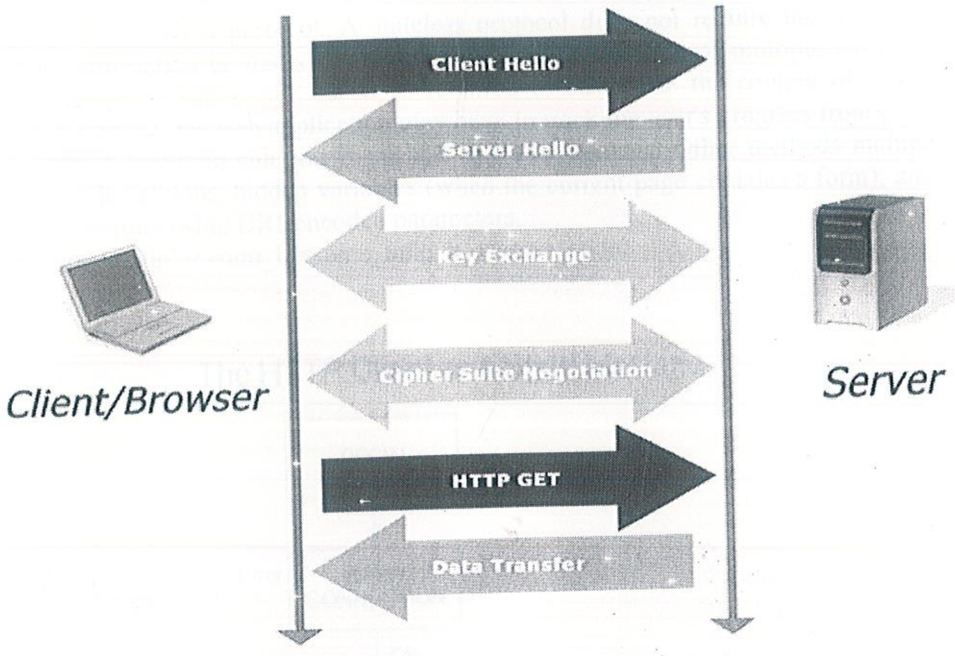


Fig. 7.2: SSL Handshake [8]

7.3 HTTP

The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web. Hypertext is a multi-linear set of objects, building a network by using logical links (hyperlinks) between the nodes (e.g. text or words). HTTP is the protocol to exchange or transfer hypertext.

7.3.1 HTTP Session

An HTTP session is a sequence of network request-response transactions. An HTTP client initiates a request by establishing aTransmission Control Protocol (TCP) connection to a particular port on a server (typically port 80: see List of TCP and UDP port numbers). An HTTP server listening on that port waits for a client's request message. Upon receiving the request, the server sends back a status line, such as "HTTP/1.1 200 OK", and a message of its own. The body of this message is typically the requested resource, although an error message or other information may also be returned.

7.3.2 HTTP Session State

HTTP is a stateless protocol. A stateless protocol does not require the server to retain information or status about each user for the duration of multiple requests. For example, when a web server is required to customize the content of a web page for a user, the web application may have to track the user's progress from page to page. A common solution is the use of HTTP cookies. Other methods include server side sessions, hidden variables (when the current page contains a form), and URL-rewriting using URI-encoded parameters.
e.g., /index.php?session_id=some_unique_session_code.

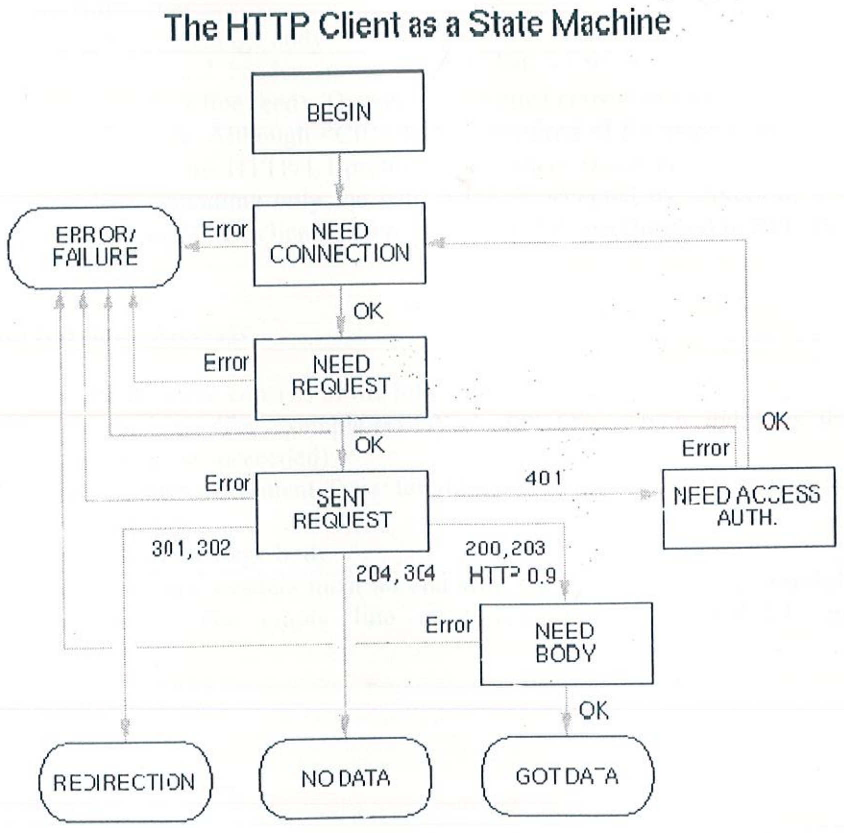


Fig. 7.3: HTTP Client As A State Machine [8]

7.3.3 Secure HTTP

There are three methods of establishing a secure HTTP connection: HTTP Secure, Secure Hypertext Transfer Protocol and the HTTP/1.1 Upgrade header. Browser support for the latter two is, however, nearly non-existent, so HTTP Secure is the dominant method of establishing a secure HTTP connection.

7.3.4 Request Message

The request message consists of the following:

- A request line, for example GET /images/logo.png HTTP/1.1, which requests a resource called /images/logo.png from the server.
- Headers, such as Accept-Language: en
- An empty line.
- An optional message body.

The request line and headers must all end with <CR><LF> (that is, a carriage return followed by a line feed). The empty line must consist of only <CR><LF> and no other whitespace. Although <CR><LF> is required <LF> alone is also accepted by most servers. In the HTTP/1.1 protocol, all headers except Host are optional.
A request line containing only the path name is accepted by servers to maintain compatibility with HTTP clients before the HTTP/1.0 specification in RFC 1945.

7.3.5 Response Message

The response message consists of the following:

- A Status-Line (for example HTTP/1.1 200 OK, which indicates that the client's request succeeded)
- Headers, such as Content-Type: text/html
- An empty line
- An optional message body

The Status-Line and headers must all end with CR+LF (a carriage return followed by a line feed). The empty line must consist of only CR+LF and no other whitespace.

7.3.6 Example Session

Below is a sample conversation between an HTTP client and an HTTP server running on www.example.com, port 80.

Client request

```
GET /index.html HTTP/1.1 CR LF
Host: www.example.com CR LF
```

```
CR LF
```

A client request (consisting in this case of the request line and only one header) is followed by a blank line, so that the request ends with a double newline, each in the form of a carriage return followed by a line feed. The "Host" header distinguishes between various DNS names sharing a single IP address, allowing name-based virtual hosting. While optional in HTTP/1.0, it is mandatory in HTTP/1.1.

Server response

```
HTTP/1.1 200 OK
 Date: Mon, 23 May 2005 22:38:34 GMT
 Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
 Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
 Etag: "3f80f-1b6-3e1cb03b"
 Accept-Ranges: none
 Content-Length: 438
 Connection: close
 Content-Type: text/html; charset=UTF-8
```

The ETag (entity tag) header is used to determine if a cached version of the requested resource is identical to the current version of the resource on the server. Content-Type specifies the Internet media type of the data conveyed by the HTTP message, while content-length indicates its length in bytes. The HTTP/1.1 webserver publishes its ability to respond to requests for certain byte ranges of the document by setting the header Accept-Ranges: bytes. This is useful, if the client needs to have only certain portions of a resource sent by the server, which is called byte serving. When Connection: close is sent in a header, it means that the web server will close the TCP connection immediately after the transfer of this response.

Most of the header lines are optional. When Content-Length is missing the length is determined in other ways. Chunked transfer encoding uses a chunk size of 0 to mark the end of the content. Identity encoding without Content-Length reads content until the socket is closed.

A Content-Encoding like gzip can be used to compress the transmitted data.

7.4 HTTPS

Hypertext Transfer Protocol Secure (HTTPS) is a combination of the Hypertext Transfer Protocol(HTTP) with the SSL/TLS protocol. It provides encrypted communication to prevent eavesdropping and secure identification of a network web server to know which web server you are really talking to. Historically, HTTPS connections were primarily used for payment transactions on the World Wide Web, e-mail and for sensitive transactions in corporate information systems.

During connecting to a website HTTPS makes it possible to know whether you are talking to the right server and protects from passive and active network attacks such as Man-in-the-middle attacks. During a session it can protect against eavesdropping and tampering with the contents of the site or with the information you send to the site. As an example HTTPS can protect from an adversary replacing downloadable content on a site with malware.

HTTPS is especially important over unencrypted Wi-fi as it is completely insecure by design and attacks on unencrypted Wi-fi networks are relatively common.

7.4.1 Overview

HTTPS is a URI scheme which has identical syntax to the standard HTTP scheme, aside from its scheme token. However, HTTPS signals the browser to use an added encryption layer of SSL/TLS to protect the traffic. SSL is especially suited for HTTP since it can provide some protection even if only one side of the communication is authenticated. This is the case with HTTP transactions over the Internet, where typically only the server is authenticated (by the client examining the server's certificate).

The main idea of HTTPS is to create a secure channel over an insecure network. This ensures reasonable protection from eavesdroppers and man-in-the-middle attacks, provided that adequate cipher suites are used and that the server certificate is verified and trusted.

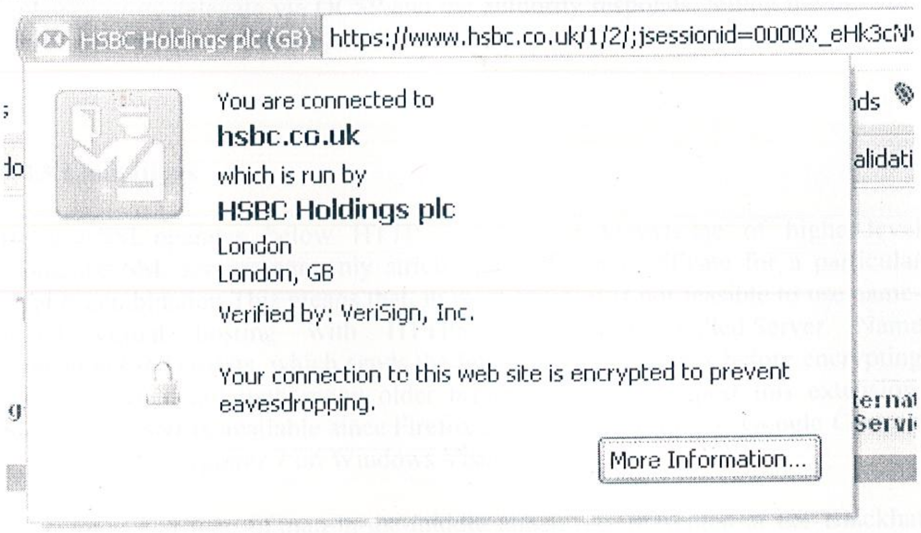
Web browsers know how to trust HTTPS websites based on certificate authorities that come pre-installed in their software. Certificate authorities (e.g. VeriSign/Microsoft/etc.) are in this way being trusted by web browser creators to provide valid certificates. Logically, it follows that a user should trust an HTTPS connection to a website if and only if all of the following are true:

1. The user trusts that the browser software correctly implements HTTPS with correctly pre-installed certificate authorities.
2. The user trusts the certificate authority to vouch only for legitimate websites.
3. The website provides a valid certificate, which means it was signed by a trusted authority.
4. The certificate correctly identifies the website (e.g., when the browser visits "https://example.com", the received certificate is properly for "Example Inc." and not some other entity).

5. Either the intervening hops on the Internet are trustworthy, or the user trusts that the protocol's encryption layer (TLS/SSL) is sufficiently secure against eavesdroppers.

7.4.2 Browser Integration

Most browsers display a warning if they receive an invalid certificate. Older browsers, when connecting to a site with an invalid certificate, would present the user with a dialog box asking if they wanted to continue. Newer browsers display a warning across the entire window. Newer browsers also prominently display the site's security information in the address bar. Extended validation certificates turn the address bar green in newer browsers. Most browsers also display a warning to the user when visiting a site that contains a mixture of encrypted and unencrypted content.



https://www.hsbc.co.uk/1/2/;jsessionid=0000X_eHk3cN

You are connected to
hsbc.co.uk
which is run by
HSBC Holdings plc
London
London, GB
Verified by: VeriSign, Inc.

Your connection to this web site is encrypted to prevent eavesdropping.

Fig. 7.4: Browser Integration [8]

The Electronic Frontier Foundation, opining that "In an ideal world, every web request could be defaulted to HTTPS", has provided an add-on called "HTTPS Everywhere" for Mozilla Firefox that enables HTTPS by default for hundreds of frequently used websites. A beta version of this plugin is also available for Google Chrome and Chromium.

7.4.3 Difference From HTTP

HTTPS URLs begin with "https://" and use port 443 by default, where HTTP URLs begin with "http://" and use port 80 by default.

HTTP is unsecure and is subject to man-in-the-middle and eavesdropping attacks, which can let attackers gain access to website accounts and sensitive information. HTTPS is designed to withstand such attacks and is considered secure against such attacks (with the exception of older deprecated versions of SSL).

7.4.4 In Case Of Compromised Private Key

A certificate may be revoked before it expires, for example because the secrecy of the private key has been compromised. Newer versions of popular browsers such as Google Chrome, firefox, Opera and Internet Explorer on Windows Vista implement the Online Certificate Status Protocol (OCSP) to verify that this is not the case. The browser sends the certificate's serial number to the certificate authority or its delegate via OCSP and the authority responds, telling the browser whether or not the certificate is still valid.

7.4.5 Limitations

Because SSL operates below HTTP and has no knowledge of higher-level protocols, SSL servers can only strictly present one certificate for a particular IP/port combination.This means that, in most cases, it is not feasible to use name-based virtual hosting with HTTPS. A soiution called Server Name Indication (SNI) exists, which sends the hostname to the server before encrypting the connection, although many older browsers do not support this extension. Support for SNI is available since Firefox 2, Opera 8, Safari 2.1, Google Chrome 6, and Internet Explorer 7 on Windows Vista.

A sophisticated type of man-in-the-middle attack was presented at the Blackhat Conference 2009. This type of attack defeats the security provided by HTTPS by changing the https: link into an http: link, taking advantage of the fact that few Internet users actually type "https" into their browser interface: they get to a secure site by clicking on a link, and thus are fooled into thinking that they are using HTTPS when in fact they are using HTTP. The attacker then communicates in clear with the client.

In May, 2010, a research paper by researchers from Microsoft Research and Indiana University discovered that detailed sensitive user data can be inferred from side channels such as packet sizes. More specifically, the researchers found that an eavesdropper can infer the illnesses/medications/surgeries of the user, her family income and investment secrets, despite HTTPS protection in several high-profile, top-of-the-line web applications in healthcare, taxation, investment and web search.

CHAPTER 8

PROPOSED SCHEME

8.1 Authentication

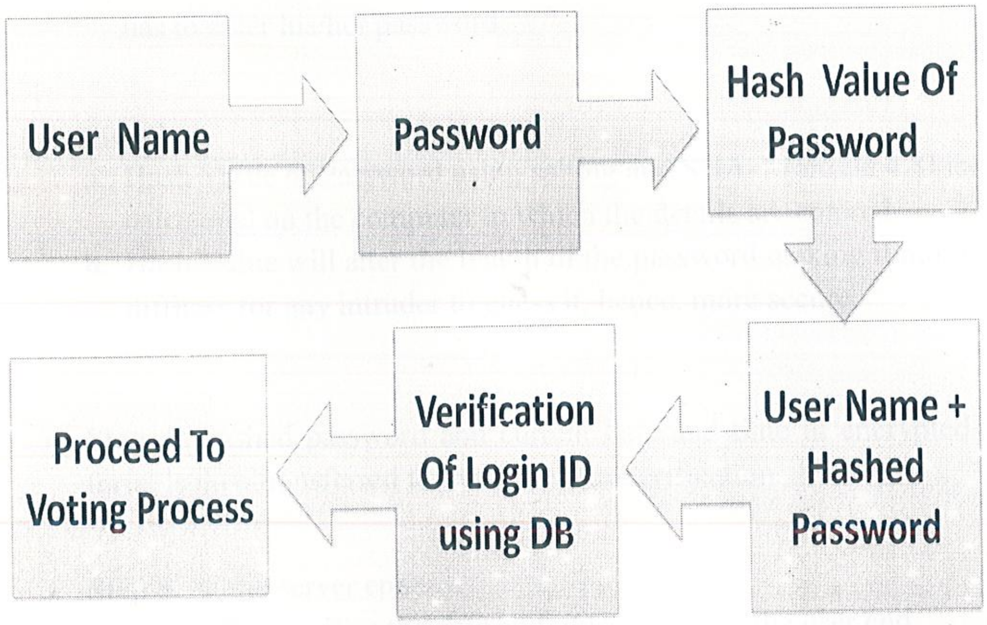

User Name
Password
Hash Value Of Password
User Name + Hashed Password
Verification Of Login ID using DB
Proceed To Voting Process

Fig. 8.1: Authentication

Steps Involved

- **Step 1.**
  - **a.** The user will make a login through the website. First he / she has to enter the User Name.
  - **b.** After that, the user will have to wait till he/she gets a response from the website.

- **Step 2.**
  - **a.** Challenge-response protocol [article 5.3.1] is used to ensure live communication.
  - **b.** For this purpose, server will send the current date and time encrypted with the user's private key.
  - **c.** Then the user will decrypt the encrypted message sent by the server using his/her public key to make sure that the date and time sent by the server is current.
  - **d.** When live communication is verified by the user, then the user has to enter his/her password.

- **Step 3.**
  - **a.** Hash Value of Password using salting and SHA-1 [article 4.3] is calculated on the computer in which the details are entered.
  - **b.** Hash Value will alter the length of the password making it more difficult for any intruder to guess it, hence, more secure.

- **Step 4.** Hashed password and current date and time in encrypted form is then transferred to the server for verification.

- **Step 5.** At the server end,
  - **a.** Server first verifies the date and time sent from the user end.
  - **b.** Using DB, the details of the user are checked.
  - **c.** If found invalid, the process terminates else there is a succcssful login and the user is forwarded to the voting menu.

8.2 Voting

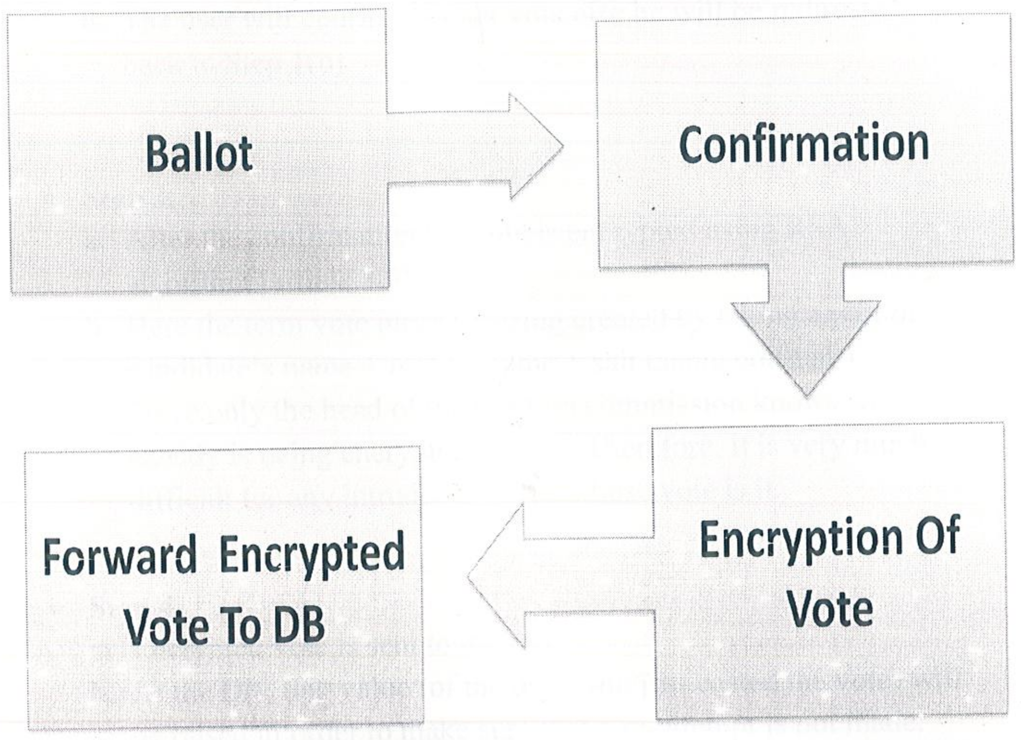


Fig. 8.2: Voting

Steps Involved

- **Step 1.**
  - a. After the user has been verified from the DB side, a webpage will appear before him/her.
  - b. Names along with the Symbols of different candidates appearing for the Election will be displayed on that webpage.
  - c. User will have to click against the option of the intended candidate.

- **Step 2.**
  - **a.** Then a confirmation window will appear before the user which will ask for confirmation from the user.
  - **b.** The user will confirm his/her vote else he will be redirected back to Step 1(b).

- **Step 3.**
  - **a.** After the confirmation the vote is encrypted using RSA algorithm [article 3.5] .
  - **b.** Here the term vote means a string created by taking a part of candidate's name + party's name + salt (some constant).
  - **c.** Since only the head of the election commission knows what exactly is being encrypted as vote. Therefore, it is very much difficult for any intruder to guess whose vote is it.

- **Step 4.**
  - **a.** Encrypted vote is sent to the DB.
  - **b.** At the DB, flag value (of the user who just casted the vote) will be raised in order to make sure that a re-attempt is not made.

8.3 Counting

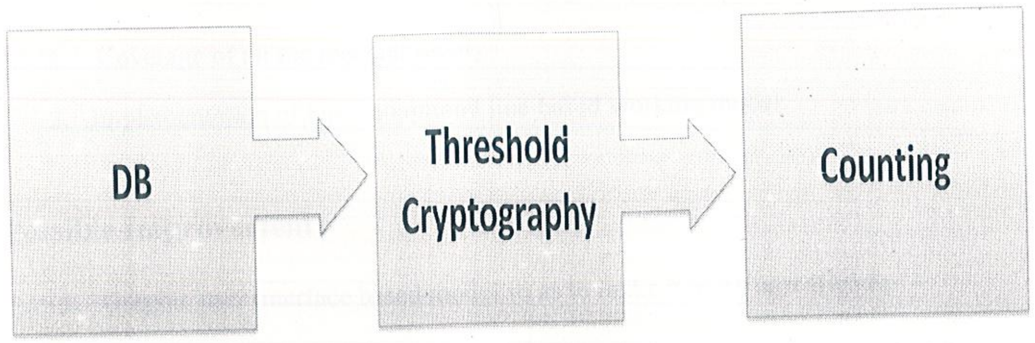


Fig. 8.3: Counting

Steps Involved

- **Step 1.**
  - **a.** After the Voting process is over, it is time for the counting of votes. Threshold Cryptography [chapter 6] is adopted here.
  - **b.** A minimum number of election officials must be present to start the counting process.
  - **c.** Then the data is decrypted and the votes are counted.

- **Step 2.** When the Counting Process is over, the results will be available on the website, showing the number of votes secured by each and every candidate.

WORK DONE AND POSSIBLE IMPROVEMENTS

Work Done

1. Coverage of all the relevant theory
2. Implementation of basic command line based working model

Possible Improvements

1. Graphic user interface based model so as to make it more user friendly
2. The concept of servers and browsers is required to be implemented. After that key exchange methods should be integrated.
3. Inclusion of biometry (such as fingerprint etc.) will make the system more accurate.

APPENDIX

[PROGRAM CODE (IN JAVA)]

1. SignIn.java : For User Login

```
import java.io.*;
import java.util.*;
import java.math.*;
import javax.crypto.*;
import java.security.*;
import PasswordField.*;
import Login.*;
import RSA.*;
class SignIn
{
        public static void main(String args[])throws IOException
        {
                char password[] = null;
                System.out.println("User Name: ");
                BufferedReader br=new BufferedReader(new
                InputStreamReader(System.in));
                String name=br.readLine();
                password = PasswordField.getPassword(System.in, "Password: ");
                String pass=String.valueOf(password);
                Boolean a=Login.login(name,pass);
                if(a==true)
                {
                        String n,p,c,str;
                        int ch=0;
                        Boolean temp=true;
                        while(temp==true)
                        {
System.out.println("\n\nS.NO.\tNAME\t\t\tPARTY\n\n");
                                FileReader fr=new FileReader("List.txt");
                                BufferedReader br1=new BufferedReader(fr);
                                int i=1;
                                while((str=br1.readLine())!=null)
                                {
                                        StringTokenizer st=new
                                        StringTokenizer(str,";");
                                        n=st.nextToken();
                                        p=st.nextToken();
                                        c=st.nextToken();
                System.out.println(i+"\t"+n+"\t\t"+p);
                i++;
        }
        fr.close();
        System.out.println("\n\nEnter The S.No. Of The
        Candidate You Want To Vote: ") ;
        ch=Integer.parseInt(br.readLine());
        while(ch<1 || ch>=i)
        {
                System.out.println("\nWrong Choice. Enter
                Again: ");
                ch=Integer.parseInt(br.readLine());
        }
        System.out.println("\nEnter Again To Confirm: ");
        int cnf=Integer.parseInt(br.readLine());
        if(cnf==ch)
        {
                temp=false;
                System.out.println("\n\t\tYour Vote Has Been
                Successfully Casted.\n\t\t\t\tTHANK YOU");
                FileReader fr1=new FileReader("DB.txt");
                BufferedReader br2=new BufferedReader(fr1);
                FileWriter fw=new FileWriter("Temp.txt");
                PrintWriter pw=new PrintWriter(fw);
                String nm,pwd;
                int fl=1;
                while((str=br2.readLine())!=null)
                {
                        StringTokenizer st1=new
                        StringTokenizer(str,";");
                        nm=st1.nextToken();
                        if(nm.equals(name))
                        {
                                pwd=st1.nextToken();
                                pw.println(nm+";"+pwd+";"+fl);
                        }
                        else
                        pw.println(str);
                }
                fr1.close();
                fw.close();
                File ob1=new File("DB.txt");
                File ob2=new File("Temp.txt");
                ob1.delete();
                ob2.renameTo(ob1);
        }
        else
        System.out.println("\n\n\n\t\t\tInputs Do Not Match.
        Enter Again.");
}
```

```
                    FileReader fr2=new FileReader("List.txt");
                    BufferedReader br3=new BufferedReader(fr2);
                    str="";
                    for(int i=0;i<ch;i++)
                    str=br3.readLine();
                    fr2.close();
                    StringTokenizer st=new StringTokenizer(str,";");
                    n=st.nextToken();
                    p=st.nextToken();
                    c=st.nextToken();
                    BigInteger[] enc_vote=RSA.encrypt(c);
                    FileWriter fw1=new FileWriter("Votes.txt",true);
                    PrintWriter pw1=new PrintWriter(fw1);
                    int j;
                    for(j=0;j<enc_vote.length-1;j++)
                    pw1.print(enc_vote[j]+";");
                    pw1.println(enc_vote[j]);
                    fw1.close();
                }
        }
}
```

2. Login.java : Imported By SignIn.java.

```
package Login;
import java.io.*;
import java.security.*;
import java.util.*;
public class Login
{
        public static final String Hash_const= "hash-my-pass";
        public  static void signup(String username, String password) throws
        IOException
        {
                String temp = Hash_const + password;
                String hashedPassword = generateHash(temp);
                int flag=0;
                FileWriter fw=new FileWriter("DB.txt",true);
                PrintWriter pw=new PrintWriter(fw);
                pw.println(username+";"+hashedPassword+";"+flag);
                fw.close();
        }
        public static Boolean login(String username, String password) throws
        IOException
        {
                Boolean auth = false;
                String temp = Hash_const + password;
                String hashedPassword = generateHash(temp);
                FileReader fr=new FileReader("DB.txt");
```

```
            BufferedReader br=new BufferedReader(fr);
            String str,u,p;
            int t=0,f;
            while((str=br.readLine())!=null)
            {
                  StringTokenizer st=new StringTokenizer(str,";");
                  u=st.nextToken();
                  p=st.nextToken();
                  f=Integer.parseInt(st.nextToken());
                  if(u.equals(username) && p.equals(hashedPassword))
                  {
                        t=1;
                        if(f==1)
                        System.out.println("\nYou Have Already Casted The
                        Vote." );
                        else
                        {
                              System.out.println("\nProceed To Voting
                              Process.");
                              auth=true;
                        }
                        break;
                  }
            }
            fr.close();
            if(t==0)
            System.out.println("\nInvalid Username Or Password.");
            return auth;
      }
      public static String generateHash(String input)
      {
            StringBuilder hash = new StringBuilder();
            try
            {
                  MessageDigest md = MessageDigest.getInstance("SHA-1");
                  byte[] hashedBytes = md.digest(input.getBytes());
                  char[] digit = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9','a', 'b', 'c', 'd',
                  'e', 'f' };
                  for (int i = 0; i < hashedBytes.length; ++i)
                  {
                        byte b = hashedBytes[i];
                        hash.append(digit[(b & 0xf0) >> 4]);
                        hash.append(digit[b & 0x0f]);
                  }
            }
            catch (NoSuchAlgorithmException e)
            {

            }
            return hash.toString();
```

```
        }
}
```

3. PasswordField.java : For Password Input. Imported By SignIn.java .

```
package PasswordField;
import MaskingThread.*;
import java.io.*;
import java.util.*;
public class PasswordField
{
        public static final char[] getPassword(InputStream in, String prompt) throws
        IOException
        {
        MaskingThread maskingthread = new MaskingThread(prompt);
        Thread thread = new Thread(maskingthread);
        thread.start();
        char[] lineBuffer=new char[128];;
        char[] buf=new char[128];
        int i;
        int room = buf.length;
        int offset = 0;
        int c;
        loop:   while (true)
        {
                switch (c = in.read())
                {
                                case -1:
                                case '\n':
                                        break loop;
                        case '\r':
                                        int c2 = in.read();
                                if ((c2 != '\n') && (c2 != -1))
                        {
                                if (!(in instanceof PushbackInputStream))
                                {
                                          in = new PushbackInputStream(in);
                                }
                                ((PushbackInputStream)in).unread(c2);
                        }
                        else
                        {
                                 break loop;
                        }
                        default:
                        if (--room < 0)
                        {
                                buf = new char[offset + 128];
```

```
                    room = buf.length - offset - 1;
                    System.arraycopy(lineBuffer, 0, buf, 0, offset);
                    Arrays.fill(lineBuffer, ' ');
                    lineBuffer = buf;
                }
                buf[offset++] = (char) c;
                break;
            }
        }
        maskingthread.stopMasking();
        if (offset == 0)
        {
            return null;
        }
        char[] ret = new char[offset];
        System.arraycopy(buf, 0, ret, 0, offset);
        Arrays.fill(buf, ' ');
        return ret;
        }
}
```

4. MaskingThread.java : Imported By PasswordField.java

```
package MaskingThread;
import java.io.*;
public class MaskingThread extends Thread
{
        private volatile boolean stop;
        private char echochar = '*';
        public MaskingThread(String prompt)
        {
            System.out.print(prompt);
        }
        public void run()
        {
            int priority = Thread.currentThread().getPriority();
            Thread.currentThread().setPriority(Thread.MAX_PRIORITY);
            try
            {
                stop = true;
                while(stop)
                {
                    System.out.print("\010" + echochar);
                    try
                    {
                        Thread.currentThread().sleep(1);
                    }
                    catch (InterruptedException iex)
                    {
```

```
                                        Thread.currentThread().interrupt();
                                return;
                        }
                }
            }
            finaliy
            {
                    Thread.currentThread().setPriority(priority);
            }
        }
        public void stopMasking()
        {
            this.stop = false;
        }
}
```

5. RSA.java : To perform RSA Encryption & Decryption.

```
package RSA;
import java.io.*;
import java.math.*;
import java.util.*;
public class RSA
{
        public static BigInteger[] encrypt( String message)throws IOException
        {
                int i ;
                BigInteger[] pk_num=publicKey();
                byte[] temp = new byte[1] ;
                byte[] digits = message.getBytes() ;
                BigInteger[] bigdigit = new BigInteger[digits.length] ;
                for( i = 0 ; i < bigdigit.length ; i++ )
                {
                        temp[0] = digits[i] ;
                        bigdigit[i] = new BigInteger( temp ) ;
                }
                BigInteger[] encrypted_msg = new BigInteger[bigdigit.length] ;
                for( i = 0 ; i < bigdigit.length ; i++ )
                {
                        encrypted_msg[i] = bigdigit[i].modPow(pk_num[0],
                        pk_num[1] ) ;
                }
                return(encrypted_msg) ;
        }
        static BigInteger[] publicKey()throws IOException
        {
                BigInteger[] temp=new BigInteger[2];
        FileReader fr=new FileReader("Public_Key.txt");
        BufferedReader br=new BufferedReader(fr);
        String str="",pk,num;
        str=br.readLine();
        StringTokenizer st=new StringTokenizer(str,";");
        pk=st.nextToken();
        num=st.nextToken();
        temp[0]=new BigInteger(pk);
        temp[1]=new BigInteger(num);
        fr.close();
        return temp;
    }
    static BigInteger[] privateKey()throws IOException
    {
        BigInteger[] temp=new BigInteger[2];
        FileReader fr=new FileReader("Private_Key.txt");
        BufferedReader br=new BufferedReader(fr);
        String str="",pk,num;
        str=br.readLine();
        StringTokenizer st=new StringTokenizer(str,";");
        pk=st.nextToken();
        num=st.nextToken();
        temp[0]=new BigInteger(pk);
        temp[1]=new BigInteger(num);
        fr.close();
        return temp;
    }

    public static String decrypt(BigInteger[] encrypted_msg)throws IOException
    {
        int i ;
        BigInteger[] pk_num=privateKey();
        BigInteger[] decrypted_msg= new BigInteger[encrypted_msg.length] ;
        for( i = 0 ; i < decrypted_msg.length ; i++ )
        decrypted_msg[i] =
        encrypted_msg[i].modPow(pk_num[0],pk_num[1]);
        char[] dec_msg = new char[decrypted_msg.length] ;
        for( i = 0 ; i < dec_msg.length ; i++ )
        dec_msg[i] = (char) (decrypted_msg[i].intValue());
        return( new String(dec_msg)) ;
    }
}
```

6. CRProtocol.java : To ensure live communication (server end)

```
import java.io.*;
import java.util.*;
import java.math.*;
import java.util.Date;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import EncTime.*;
class CRProtocol
{
        public static void main(String[] args)throws IOException
        {
                DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd
                HH:mm:ss");
                Date date = new Date();
                String s=dateFormat.format(date);
                BigInteger[] enc_vote=EncTime.encrypt(s);
                FileWriter fw1=new FileWriter("CR.txt",true);
                PrintWriter pw1=new PrintWriter(fw1);
                int j;
                for(j=0;j<enc_vote.length-1;j++)
                pw1.print(enc_vote[j]+";");
                pw1.println(enc_vote[j]);
                fw1.close();
        }
}
```

7. Authentication.java : To ensure live communication (user end)

```
import java.io.*;
import java.util.*;
import java.math.*;
import java.util.Date;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import EncTime.*;
class Authentication
{
        public static void main(String[] args)throws IOException
        {
                DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd
                HH:mm:ss");
```

```
				Date date = new Date();
				String s=dateFormat.format(date);
				FileReader fr=new FileReader("CR.txt");
				BufferedReader br=new BufferedReader(fr);
				String str;
				while((str=br.readLine())!=null)
				{
					int temp=0;
					StringTokenizer st=new StringTokenizer(str,";");
					String t=st.nextToken();
					while(t!=null)
					{
						temp++;
						try
						{
							t=st.nextToken();
						}
						catch(NoSuchElementException nsel)
						{
							break;
						}
					}
					BigInteger[] enc_msg=new BigInteger[temp];
					StringTokenizer stw=new StringTokenizer(str,";");
					for(int j=0;j<temp;j++)
					{
						enc_msg[j]=new BigInteger(stw.nextToken());
					}
					String dec_msg=EncTime.decrypt(enc_msg);
					//System.out.println(dec_msg);
					//for(ncc=0;ncc<nc;ncc++)
					//{
					//	if(dec_msg.equals(s))
						//{
						//	System.out.println("Live Communication");
						//}
					//}
				}
				fr.close();
			}
		}
```

8. EncTime.java : For encryption & decryption of challenge to ensure live communication. Imported by Authentication.java & CRProtocol.java .

```
package EncTime;
import java.io.*;
import java.math.*;
import java.util.*;
public class EncTime
{

        public static BigInteger[] encrypt( String message)throws IOException
        {
                int i ;
                BigInteger[] pk_num=privateKey();
                byte[] temp = new byte[1] ;
                byte[] digits = message.getBytes() ;
                BigInteger[] bigdigit = new BigInteger[digits.length] ;
                for( i = 0 ; i < bigdigit.length ; i++ )
                {
                        temp[0] = digits[i] ;
                        bigdigit[i] = new BigInteger( temp ) ;
                }
                BigInteger[] encrypted_msg = new BigInteger[bigdigit.length] ;
                for( i = 0 ; i < bigdigit.length ; i++ )
                {
                        encrypted_msg[i] = bigdigit[i].modPow(pk_num[0],
                        pk_num[1] ) ;
                }
                return(encrypted_msg) ;
        }
        static BigInteger[] publicKey()throws IOException
        {
                BigInteger[] temp=new BigInteger[2];
                FileReader fr=new FileReader("Public_Key_User.txt");
                BufferedReader br=new BufferedReader(fr);
                String str="",pk,num;
                str=br.readLine();
                StringTokenizer st=new StringTokenizer(str,";");
                pk=st.nextToken();
                num=st.nextToken();
                temp[0]=new BigInteger(pk);
                temp[1]=new BigInteger(num);
                fr.close();
                return temp;
        }
        static BigInteger[] privateKey()throws IOException
        {
                BigInteger[] temp=new BigInteger[2];
                FileReader fr=new FileReader("Private_Key_User.txt");
                BufferedReader br=new BufferedReader(fr);
                String str="",pk,num;
                str=br.readLine();
                StringTokenizer st=new StringTokenizer(str,";");
                pk=st.nextToken();
                num=st.nextToken();
                temp[0]=new BigInteger(pk);
                temp[1]=new BigInteger(num);
                fr.close();
                return temp;
        }

        public static String decrypt(BigInteger[] encrypted_msg)throws IOException
        {
                int i ;
                BigInteger[] pk_num=publicKey();
                BigInteger[] decrypted_msg= new BigInteger[encrypted_msg.length] ;
                for( i = 0 ; i < decrypted_msg.length ; i++ )
                decrypted_msg[i] =
                encrypted_msg[i].modPow(pk_num[0],pk_num[1]);
                char[] dec_msg = new char[decrypted_msg.length] ;
                for( i = 0 ; i < dec_msg.length ; i++ )
                dec_msg[i] = (char) (decrypted_msg[i].intValue());
                return( new String(dec_msg)) ;
        }
}
```

9. Counting.java : For counting using threshold cryptography.

```
import java.io.*;
import java.util.*;
import java.math.*;
import javax.crypto.*;
import java.security.*;
import PasswordField.*;
import Counting.*;
class Count
{
        public static void main(String args[])throws IOException
        {
                char key[] = null;
                BufferedReader br=new BufferedReader(new
                InputStreamReader(System.in));
                int n=0,c=0;
                System.out.println("\n\tOFFICIALS' KEY INPUT");
                for(int i=1;i<=10;i++)
```

```
{
        System.out.print("OFFICIAL "+i+" present? Enter 1 for yes, 0
        otherwise: ");
        int p=Integer.parseInt(br.readLine());
        if(p==1)
        {
                key = PasswordField.getPassword(System.in, "Enter
                Key: ");
                String pass=String.valueOf(key);
                final String Hash_const= "hash-my-pass";
                String temp = Hash_const + pass;
                String hashedKey = generateHash(temp);
                FileReader fr=new FileReader("Keys.txt");
                BufferedReader br1=new BufferedReader(fr);
                String str="";
                int t=0;
                for(int j=0;j<i;j++)
                str=br1.readLine();
                if(hashedKey.equals(str))
                System.out.println("VALID KEY");
                else
                {
                        System.out.println("INVALID KEY");
                        n++;
                        if(n>2)
                        {
                                System.out.println("Error: At Least 8
                                Valid Keys Required. Cannot Start
                                Counting Process.");
                                c=1;
                                break;
                        }
                }
                fr.close();
        }
        else
        {
                n++;
                if(n>2)
                {
                                System.out.println("Error: At Least 8
                                Valid Keys Required. Cannot Start
                                Counting Process.");
                                c=1;
                                break;
                }
        }
}
if(c!=1)
{
				System.out.println("Counting Started...........................");
				Counting.count();
			}
		}
		public static String generateHash(String input)
		{
			StringBuilder hash = new StringBuilder();
			try
			{
				MessageDigest md = MessageDigest.getInstance("SHA-1");
					byte[] hashedBytes = md.digest(input.getBytes());
				char[] digit = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9','a', 'b', 'c', 'd',
'e', 'f' };
				for (int i = 0; i < hashedBytes.length; ++i)
				{
					byte b = hashedBytes[i];
					hash.append(digit[(b & 0xf0) >> 4]);
					hash.append(digit[b & 0x0f]);
				}
			}
			catch (NoSuchAlgorithmException e)
			{

			}
			return hash.toString();
		}
}
```

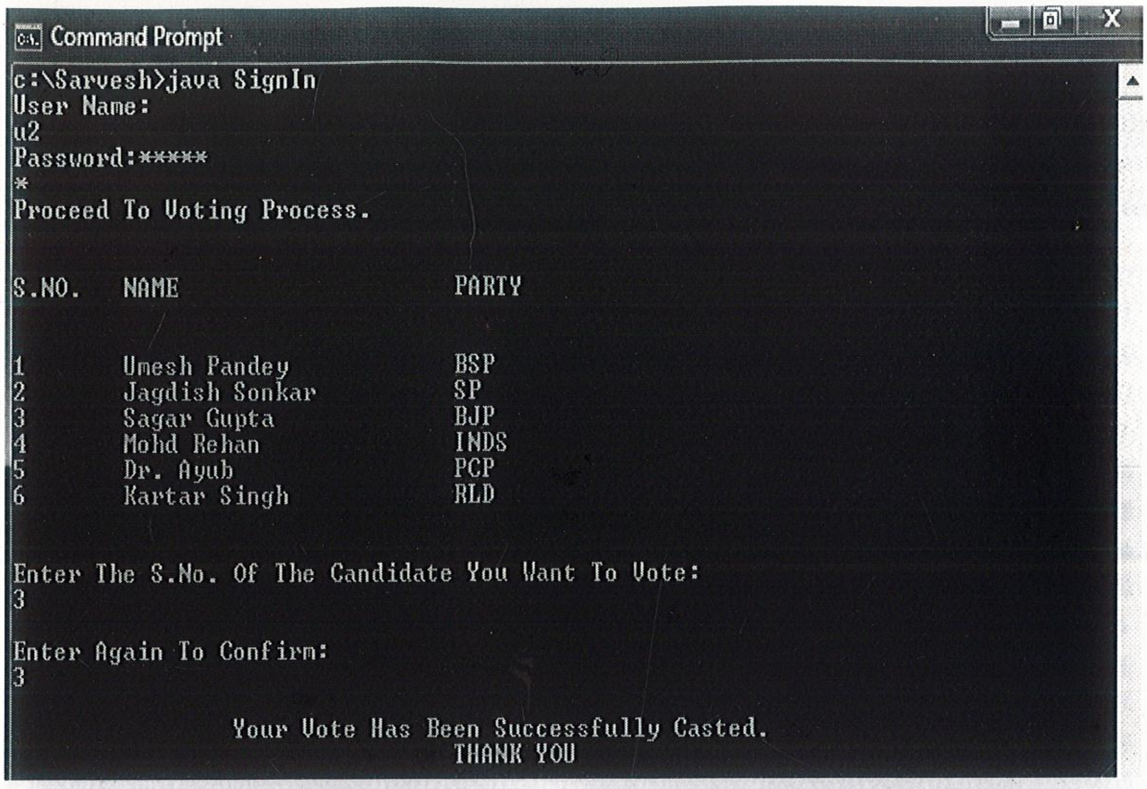
10. Counting.java : Imported by Count.java .

```
package Counting;
import java.math.*;
import RSA.*;
import java.io.*;
import java.util.*;
public class Counting
{
	public static void count()throws IOException
	{
		int nc=0;
		FileReader frr=new FileReader("List.txt");
		BufferedReader brr=new BufferedReader(frr);
		String strr;
		while((strr=brr.readLine())!=null)
```

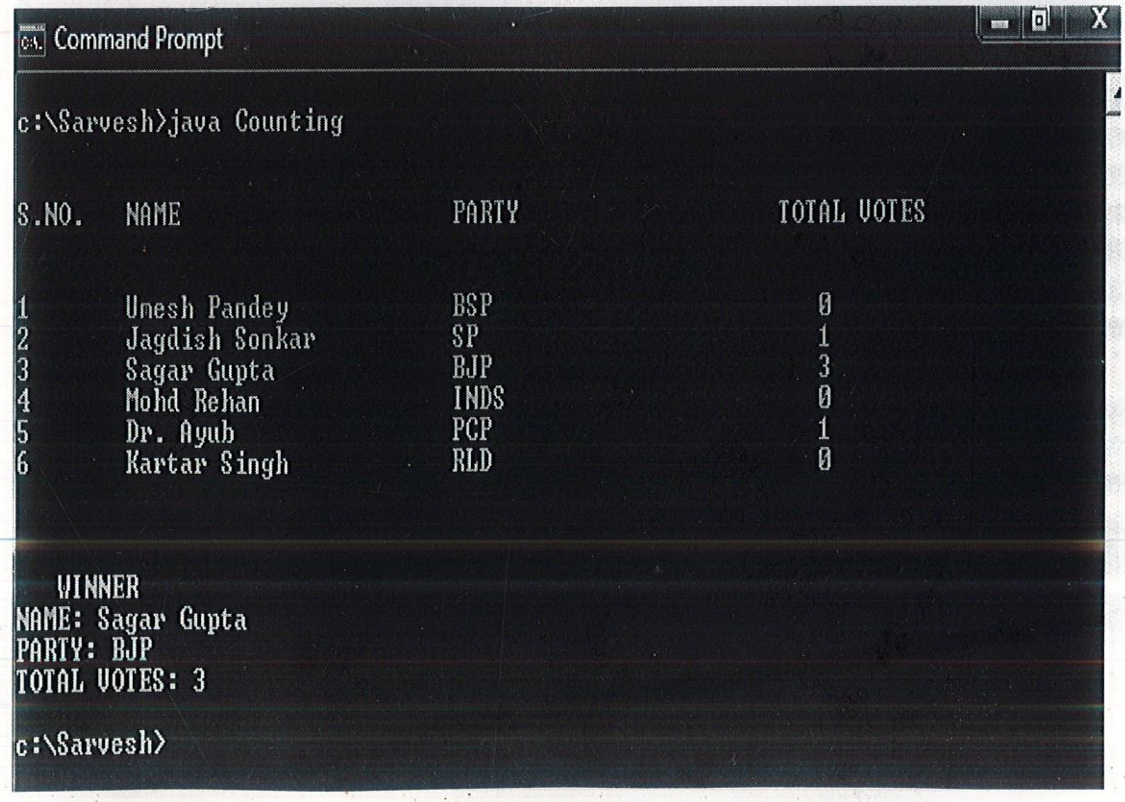
```
nc++;
frr.close();
String[] codes=new String[nc];
int[] votes=new int[nc];
FileReader frr1=new FileReader("List.txt");
BufferedReader brr1=new BufferedReader(frr1);
int ncc=0;
while((strr=brr1.readLine())!=null)
{
        StringTokenizer stt=new StringTokenizer(strr,";");
        String n=stt.nextToken();
        String p=stt.nextToken();
        codes[ncc]=stt.nextToken();
        ncc++;
}
frr1.close();
for(ncc=0;ncc<nc;ncc++)
votes[ncc]=0;
FileReader fr=new FileReader("Votes.txt");
BufferedReader br=new BufferedReader(fr);
String str;
while((str=br.readLine())!=null)
{
        int temp=0;
        StringTokenizer st=new StringTokenizer(str,";");
        String s=st.nextToken();
        while(s!=null)
        {
                temp++;
                try
                {
                        s=st.nextToken();
                }
                catch(NoSuchElementException nsel)
                {
                        break;
                }
        }
        BigInteger[] enc_msg=new BigInteger[temp];
        StringTokenizer stw=new StringTokenizer(str,";");
        for(int j=0;j<temp;j++)
        {
                enc_msg[j]=new BigInteger(stw.nextToken());
        }
        String dec_msg=RSA.decrypt(enc_msg);
        for(ncc=0;ncc<nc;ncc++)
        {
                if(dec_msg.equals(codes[ncc]))
                {
                        votes[ncc]++;
```

```
                                        break;
                                }
                        }
                }
                fr.close();
                int highest=0;
                for(ncc=1;ncc<nc;ncc++)
                {
                        if(votes[ncc]>votes[highest])
                        highest=ncc;
                }
                FileReader frr2=new FileReader("List.txt");
                BufferedReader brr2=new BufferedReader(frr2);
                ncc=0;
                String winn="",winp="";
                System.out.println("\n\nS.NO.\tNAME\t\t\tPARTY\t\t\tTOTAL
                VOTES\n\n");
                while((strr=brr2.readLine())!=null)
                {
                        StringTokenizer stt=new StringTokenizer(strr,";");
                        String n=stt.nextToken();
                        String p=stt.nextToken();
                        System.out.println((ncc+1)+"\t"+n+"\t\t"+p+"\t\t\t
                        "+votes[ncc]);
                        if(ncc==highest)
                        {
                                winn=n;
                                winp=p;
                        }
                        ncc++;
                }
                frr2.close();
        System.out.println("\n\n\n   WINNER \nNAME: "+winn+"\nPARTY:
        "+winp+"\nTOTAL VOTES: "+votes[highest]);
        }
}
```

11. Screenshots Of Sample Output

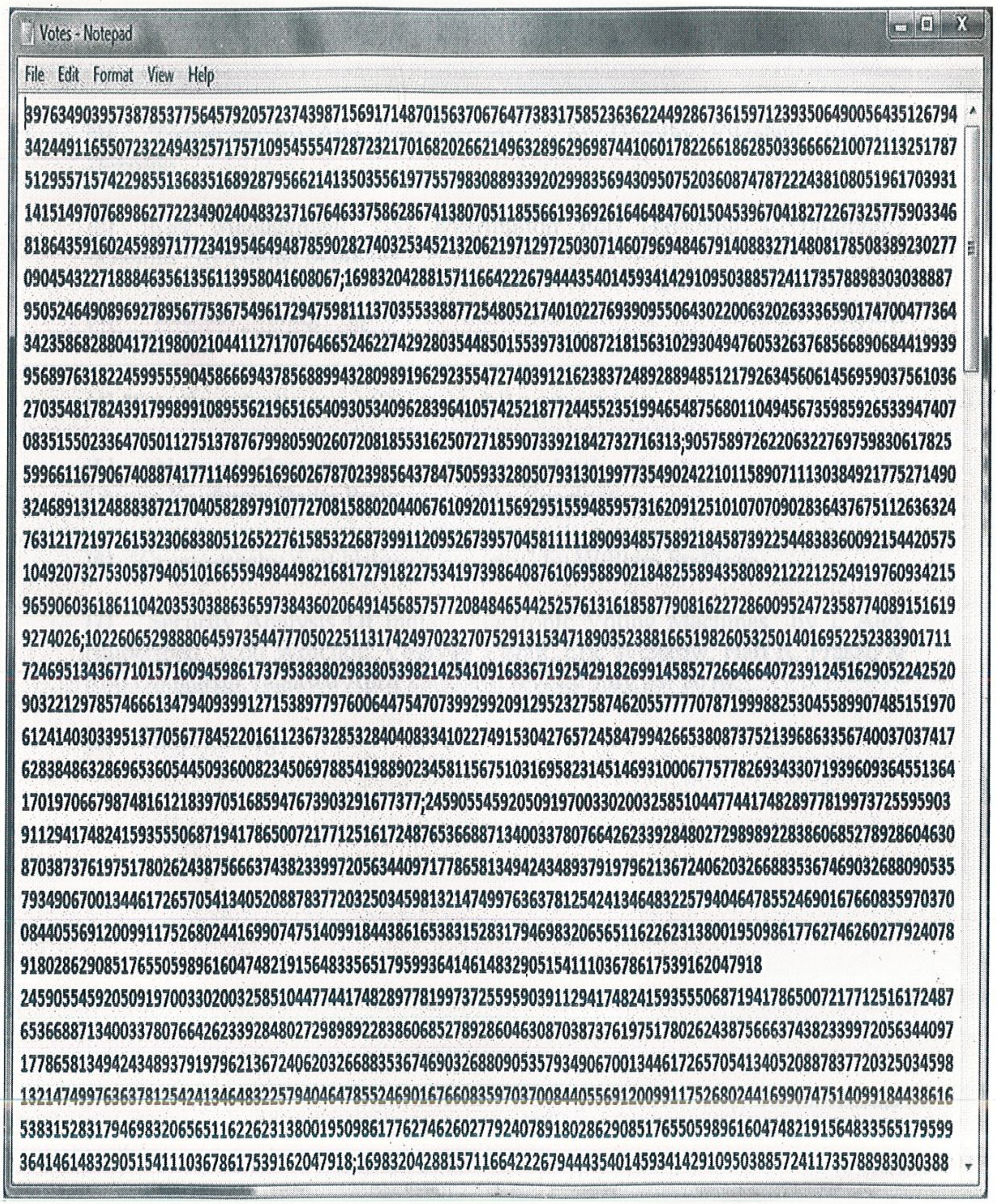


```
c:\Sarvesh>java SignIn
User Name:
u2
Password:*****
*
Proceed To Voting Process.


S.NO.   NAME                    PARTY


1       Umesh Pandey            BSP
2       Jagdish Sonkar          SP
3       Sagar Gupta             BJP
4       Mohd Rehan              INDS
5       Dr. Ayub                PCP
6       Kartar Singh            RLD


Enter The S.No. Of The Candidate You Want To Vote:
3

Enter Again To Confirm:
3

                Your Vote Has Been Successfully Casted.
                                THANK YOU
```



```
c:\Sarvesh>java Counting


S.NO.   NAME                    PARTY                   TOTAL VOTES


1       Umesh Pandey            BSP                       0
2       Jagdish Sonkar          SP                        1
3       Sagar Gupta             BJP                       3
4       Mohd Rehan              INDS                      0
5       Dr. Ayub                PCP                       1
6       Kartar Singh            RLD                       0



  WINNER
NAME: Sagar Gupta
PARTY: BJP
TOTAL VOTES: 3

c:\Sarvesh>
```

12. Screenshot Of Encrypted Votes



```
3976349039573878537756457920572374398715691714870156370676477383175852363622449286736159712393506490056435126794
3424491165507232249432571757109545554728723217016820266214963289629698744106017822661862850336666210072113251787
5129557157422985513683516892879566214135035561977557983088933920299835694309507520360874787222438108051961703931
1415149707689862772234902404832371676463375862867413807051185566193692616464847601504539670418272267325775903346
8186435916024598971772341954649487859028274032534521320621971297250307146079694846791408832714808178508389230277
09045432271888463561356113958041608067;1698320428815711664222679444354014593414291095038857241173578898303038887
9505246490896927895677536754961729475981113703553388772548052174010227693909550643022006320263336590174700477364
3423586828804172198002104411271707646652462274292803544850155397310087218156310293049476053263768566890684419939
9568976318224599555904586669437856889943280989196292355472740391216238372489288948512179263456061456959037561036
2703548178243917998991089556219651654093053409628396410574252187724455235199465487568011049456735985926533947407
083515502336470501127513787679980590260720818553162507271859073392184273271631 3;90575897262206322769759830617825
5996611679067408874177114699616960267870239856437847505933280507931301997735490242210115890711130384921775271490
3246891312488838721704058289791077270815880204406761092011569295155948595731620912510107070902836437675112636324
7631217219726153230683805126522761585322687399112095267395704581111189093485758921845873922544838360092154420575
1049207327530587940510166559498449821681727918227534197398640876106958890218482558943580892122212524919760934215
9659060361861104203530388636597384360206491456857577208484654425257613161858779081622728600952472358774089151619
9274026;1022606529888064597354477705022511317424970232707529131534718903523881665198260532501401695225238390171 1
7246951343677101571609459861737953838029838053982142541091683671925429182699145852726646640723912451629052242520
9032212978574666134794093991271538977976006447547073992992091295232758746205577770787199988253045589907485151970
6124140303395137705677845220161123673285328404083341022749153042765724584799426653808737521396863356740037037417
6283848632869653605445093600823450697885419889023458115675103169582314514693100067757782693433071939609364551364
1701970667987481612183970516859476739032916773 77;2459055459205091970033020032585104477441748289778199737255 95903
9112941748241593555068719417865007217712516172487653668871340033780766426233928480272989892283860685278928604630
8703873761975178026243875666374382339972056344097177865813494243489379197962136724062032668835367469032688090535
7934906700134461726570541340520887837720325034598132147499763637812542413464832257940464785524690167660835970370
0844055691200991175268024416990747514099184438616538315283179469832065651162262313800195098617762746260277924078
9180286290851765505989616047482191564833565179599364146148329051541110367861753916204 7918
2459055459205091970033020032585104477441748289778199737255959039112941748241593555068719417865007217712516172487
6536688713400337807664262339284802729898922838606852789286046308703873761975178026243875666374382339972056344097
1778658134942434893791979621367240620326688353674690326880905357934906700134461726570541340520887837720325034598
1321474997636378125424134648322579404647855246901676608359703700844055691200991175268024416990747514099184438616
5383152831794698320656511622623138001950986177627462602779240789180286290851765505989616047482191564833565179599
3641461483290515411103678617539162047918;1698320428815711664222679444354014593414291095038857241173578898303038 8
```

REFERENCES

[1] "Cryptography And Data Security" by Dorothy Elizabeth Robling Denning

[2] "Key Management And Distribution For Threshold Cryptography Schemes" by Fabian Schilcher

[3] "Internet Voting" by Kevin Coleman, Analyst in American National Government

[4] "File Transfer Protocol (FTP)" by J.Postel and J.Reynolds

[5] "Network Security: Private Communication In A Public World" by Charlie Kaufman, Radia Perlman & Mike Speciner

[6] "Cryptography And Network Security" by William Stallings

[7] "Security Analysis Of India's Electronic Voting Machines" by J. Alex Halderman, Scott Wolchok, Vasavya Yagati, Eric Wustrow, Hari K. Prasad & Rop Gonggrijp, released April 29, 2010, revised July 29, 2010.

[8] Wikipedia.org