Online Voting System Powered By Biometric Security Using Steganography

PROJECT REPORT

Submitted in partial fulfillment of the requirements for the award of

Degree of ***Bachelor of Technology***

In ***Electronics and Communication Engineering***

of ***University of Kerala***

Submitted by

AMAL V. (10400004)

ANAND O. (10400005)

PRASANTH G. P. (10400025)

EMMANUEL BRISTOW KUBALIO (10400012)

DEPARTMENT OF ELECTRONICS & COMMUNICATION

COLLEGE OF ENGINEERING

THIRUVANANTHAPURAM

2014

DEPARTMENT OF ELECTRONICS & COMMUNICATION

COLLEGE OF ENGINEERING

THIRUVANANTHAPURAM

CERTIFICATE

This is to certify that this report entitled **"Online Voting System Powered By Biometric Security Using Steganography"** submitted herewith is a bonafide record of the project design done by **AMAL V., ANAND O., PRASANTH G.P., EMMANUEL BRISTOW KUBALIO** of Electronics and Communication department in partial fulfillment of requirements for the award of **Bachelor of Technology** in **Electronics & Communication Engineering** of **University Of Kerala**, during the academic year 2013.

Guided by

Ms. Viji R.

Asst Professor

Dept. of ECE

Coordinated by

Mr. Prajith C. A.

Associate Professor

Dept. of ECE

Dr. Vrinda V. Nair

Professor & Head of Department

Dept. of ECE

ACKNOWLEDGEMENT

With prayers to GOD for his grace and blessings, for without his unforeseen guidance, this project would have remained only in dreams.

We express our sincere gratitude to our principal, Dr. Sheela S and our Head of the Department Dr. Vrinda V Nair for providing us the ambience for carrying out the work of our project and Mr. Jayakrishnaraj , our project co-ordinator, without whose help and motivation we could not have conceived this project.

We are profoundly indebted to guide Ms.Viji.R and all other lecturers in Electronics and Communication Department for their incessant guidance, constructive criticism and encouragement throughout the tenure of the study.

Last, but not the least, we extend our deepest gratitude to our parents and friends without whose support this project wouldn't have become a reality.

AMAL V.

ANAND O.

PRASANTH G. P.

EMMANUEL BRISTOW KUBALIO

ABSTRACT

Steganography aims to hide secret messages into a cover media, such as text, audio, image and video, without drawing suspicious. In many applications, the most important requirement for steganography is the security, which means that the stegos should be visually and statistically similar to their corresponding covers. Usually, there are several factors to influence the security performances significantly, such as the hiding scheme, the length of the secret message to be embedded, and the selection of the cover media. On the other side, steganalysis firstly tries to differentiate stego media from covers, then further estimate the length of the secret message, and finally extract and decipher the secret message as the ultimate goal. Integrity of the election process will determine the integrity of democracy itself. So the election system must be secure and robust against a variety of fraudulent behaviors, should be transparent and comprehensible that voters and candidates can accept the results of an election. But in history, there are examples of elections being manipulated in order to influence their outcome. Using Cryptography and Steganography at the same time, we try to provide Biometric as well as Password security to voter accounts. The scheme uses images as cover objects for Steganography and as keys for Cryptography. The key image is a Biometric measure, such as a fingerprint image. Proper use of Cryptography greatly reduces the risks in these systems as the hackers have to find both secret key and the template. The basic idea is to merge the secret key with the cover image on the basis of key image.

TABLE OF CONTENTS

1.INTRODUCTION

A Voting system or Electoral system is a method by which voters make a choice between options, often in an election. A voting system contains rules for valid voting and how the votes are counted and aggregated to yield the final result. Elections allow the populace to choose their representatives and express their preferences for how they will be governed. Naturally, the integrity of the election process is fundamental to the integrity of democracy itself. The election system must be sufficiently robust to withstand a variety of fraudulent behaviors and must be sufficiently transparent and comprehensible that voters and candidates can accept the results of an election. Unsurprisingly, history is littered with examples of elections being manipulated in order to influence their outcome.

The design of a "good" voting system, whether electronic or using traditional paper ballots or mechanical devices, must satisfy a number of sometimes competing criteria. The anonymity of a voter's ballot must be preserved, both to guarantee the voter's safety when voting against a malevolent candidate, and to guarantee that voters have no evidence that proves which candidates received their votes. The existence of such evidence would allow votes to be purchased by a candidate. The voting system must also be tamper-resistant to thwart a wide range of attacks, including ballot stuffing by voters and incorrect tallying by insiders. It should provide the voter a secure environment for casting the vote without any fear or under any pressure.

Another factor, as shown by the so-called "butterfly ballots" in the Florida 2000 presidential election, is the importance of human factors. A voting system must be comprehensible to and usable by the entire voting population, regardless of age, infirmity, or disability. Providing accessibility to such a diverse population is an important engineering problem and one where, if other security is done

well, electronic voting could be a great improvement over current paper systems.

1.1 OBJECTIVE

The project aims to achieve the following:

- To develop a secure and time saving platform, wherein the voters could easily cast their vote.
- Security to be provided using the concepts of cryptography and steganography.

1.2 EXISTING SYSTEM

In the existing system the person has to go to the voting centers for casting his/her vote. There is a high risk of fraud and booth capture at some places. The candidates also pressurizes the voters of an specific area to vote only for them .Apart from that, there could be errors while counting the votes or the person counting could be biased to a candidate which may affect the genuine outcome .

Electronic voting (also known as e-voting) is a term encompassing several different types of voting, embracing both electronic means of casting a vote and electronic means of counting votes.

Electronic voting technology can include punched cards, optical scan voting systems and specialized voting kiosks (including self-contained direct-recording electronic voting systems, or DRE). It can also involve transmission of ballots and votes via telephones, private computer networks, or the Internet.

In general, two main types of e-Voting can be identified: Firstly e-voting which is physically supervised by representatives of governmental or independent electoral authorities (e.g. electronic voting machines located at polling stations); Secondly ,remote e-Voting where voting is performed within the voter's sole influence, and is not physically supervised by representatives of governmental

authorities (e.g. voting from one's personal computer, mobile phone, television via the internet (i-voting).

Electronic voting technology can speed the counting of ballots and can provide improved accessibility for disabled voters. However, there has been contention, especially in the United States, that electronic voting, especially DRE voting, could facilitate electoral fraud.

Electronic voting systems for electorates have been in use since the 1960s when punched card systems debuted. Their first widespread use was in the USA where 7 counties switched to this method for the 1964 presidential election. The newer optical scan voting systems allow a computer to count a voter's mark on a ballot. DRE voting machines which collect and tabulate votes in a single machine are used by all voters in all elections in Brazil and India, and also on a large scale in Venezuela and the United States. They have been used on a large scale in the Netherlands but have been decommissioned after public concerns.

Paper-based electronic voting system

Sometimes called a "document ballot voting system", paper-based voting systems originated as a system where votes are cast and counted by hand, using paper ballots. With the advent of electronic tabulation came systems where paper cards or sheets could be marked by hand, but counted electronically. These systems included punched card voting, marksense and later digital pen voting systems.

Most recently, these systems can include an Electronic Ballot Marker (EBM), which allows voters to make their selections using an electronic input device, usually a touch screen system similar to a DRE. Systems including a ballot marking device can incorporate different forms of assistive technology.

DIRECT RECORDING ELECTRONIC (DRE) VOTING SYSTEM

A direct-recording electronic (DRE) voting machine records votes by means of a ballot display provided with mechanical or electro-optical components that can be activated by the voter (typically buttons or a touchscreen) that processes data with computer software; and that records voting data and ballot images in memory components. After the election it produces a tabulation of the voting data stored in a removable memory component and as printed copy. The system may also provide a means for transmitting individual ballots or vote totals to a central location for consolidating and reporting results from precincts at the central location. These systems use a precinct count method that tabulates ballots at the polling place. They typically tabulate ballots as they are cast and print the results after the close of polling.

Electronic voting systems may offer advantages compared to other voting techniques. An electronic voting system can be involved in any one of a number of steps in the setup, distributing, voting, collecting, and counting of ballots, and thus may or may not introduce advantages into any of these steps. Potential disadvantages exist as well including the potential for flaws or weakness in any electronic component. It has been demonstrated that as voting systems become more complex and include software, different methods of election fraud become possible. Others also challenge the use of electronic voting from a theoretical point of view, arguing that humans are not equipped for verifying operations occurring within an electronic machine and that because people cannot verify these operations, the operations cannot be trusted. Furthermore, some computing experts have argued for the broader notion that people cannot trust any programming they did not author.

The online voting that was tried out in Gujarat had not been so efficient one. A lot of indisciplinary situations have been reported in the system. The user authentication and password protection has not been secured. The algorithms used are still obsolete and are unable to provide a secured environment for the

voting process. There has been many flaws with the system as also has been seen in Florida and California some years ago.

1.2.1 LITERATURE SURVEY

The internet voting system has been tried out at places like Florida and California but the results were not so good. There were many flaws with the software and also the parts used were outdated. As the world watched the electoral drama unfold in Florida at the end of 2000, people started wondering, "Wouldn't all our problems be solved if they just used Internet Voting?" People all over the world soon started taking a hard look at their voting equipment and procedures, and trying to figure out how to improve them. There is a strong inclination towards moving to Remote Internet Voting – at least among the politicians – in order to enhance voter convenience, increase voter confidence and voter turnout. However, there are serious technological and social aspects that make Remote Internet Voting infeasible in the visible future (as seen by 2000). Therefore, many technologists have suggested that remote poll-site electronic voting, where the voter can vote at any poll-site (not only his home county poll-site), seems to be the best step forward as it provides better voter convenience, but at the same time, does not compromise security.

The Caltech/MIT Voting Technology Project came into being in order to develop a new voting technology in order to prevent a recurrence of the problems that threatened the 2000 U. S. Presidential Elections. The report assesses the magnitude of the problems, their root causes and how technology can reduce them. They address a wide range of "What is" issues including voting procedures, voting equipment, voter registration, polling places, absentee and early voting, ballot security, cost and public finance of elections, etc. They then propose a novel "What could be" framework for voting technology (that moves away from monolithic voting structures), and propose that a process for

innovation be setup. The framework is "A Modular Voting Architecture ("Frogs")" in which vote generation is performed separately from vote casting, and the "Frog" forms a permanent audit trail, the importance of which cannot be over-stressed. Here, the vote generation machine can be proprietary whereas the vote casting machine must be open-source and thoroughly verified and certified for correctness and security. Finally, the report provides a set of short-term and long-term recommendations on the various issues related to voting.

In 'e-voting' ,Rivest addresses some issues like the " secure platform problem" and the (im)possibility of giving a receipt to the voter . He also provides some personal opinions on a host of issues including the striking dissimilarity between e-commerce and e-voting, the dangers of adversaries performing automated, wide-scale attacks while voting from home, the need for extreme simplicity of voting equipment, the importance of audit-trails, support for disabled voters, security problems of absentee ballots, etc.

The NSF Internet Voting Report addresses the feasibility of different forms of Internet voting from both the technical and social science perspectives, and defines a research agenda to pursue if Internet voting is to be viable in the future. It groups Internet voting systems into three general categories as follows:

- Poll-site Internet voting: It offers the promise of greater convenience and efficiency in that voters could cast their ballots from any poll site, and the tallying process would be both fast and certain. More importantly, since election officials would control both the voting platform and the physical environment, managing the security risks of such systems is feasible.
- Kiosk voting: Voting machines would be located away from traditional polling places, in such convenient locations as malls, libraries, or schools. The voting platforms would still be under the control of election officials, and the physical environment could be modified as needed and monitored (e.g., by

election officials, volunteers, or even cameras) to address security and privacy concerns, and prevent coercion or other forms of intervention.

- Remote Internet voting: It seeks to maximize the convenience and access of the voters by enabling them to cast ballots from virtually any location that is Internet accessible. While this concept is attractive and offers significant benefits, it also poses substantial security risks and other concerns relative to civic culture. Current and near-term technologies are put into use to address these risks.

The report presents some findings on the feasibility of each of these categories and provides research recommendations for the long-term future. It then identifies criteria for election systems. Finally, it addresses the technological issues (including voting system vulnerabilities, reliability, testing, certification and standards, specifications of source code, platform compatibility, secrecy and non-coercibility, etc.) and social science issues (such as voter participation, voter access, the election process, voter information, deliberative and representative democracy, community and character of elections, distribution of roles, legal concerns, voter registration, etc.)

The California Internet Voting Report suggests a strategy of evolutionary rather than revolutionary change towards achieving the goal of providing voters with the opportunity to cast their ballots at any time from any place via the Internet. The report defines four distinct Internet voting models – Internet voting at voter's polling place, Internet voting at any polling place, Remote Internet voting from County computers or kiosks, Remote Internet voting from any Internet connection – and the corresponding technical and design requirements that must be met when implementing any of the stages. It addresses the advantages, implementation and security issues of each of the four stages. They believe that additional technical innovations are necessary before remote

Internet voting can be widely implemented as a useful tool to improve participation in the elections process and that current technology however would allow for the implementation of new voting systems that would allow voters to cast a ballot over the Internet from a computer at any one of a number of county-controlled polling places in a county. Finally, the report presents the findings and recommendations of the task force on policy issues.

"Security Criteria for Electronic Voting" considers some basic criteria for confidentiality, integrity, availability, reliability, and assurance for computer systems involved in electronic voting. After an assessment of the realizability of those criteria, it concludes that, operationally, many of the criteria has to be satisfied through various techniques.

The reports states the problems that is being faced by the remote e – voting system and also clarifies the issues could be solved through use of new techniques and algorithms. This is what we have tried to achieve in the project.

2.ENHANCED SECURITY FOR ONLINE VOTING SYSTEM

We first give an overview of our approach, and discuss each of the major components of our system in this section. Our approach is composed of two phases: initial phase for processing the stego image, that is the image over which a message could be send, and also the allocation of secure key using the algorithm. The second is the casting of the vote by the voter himself.

In this paper we have presented a method for integrating cryptography and steganography. The strength of our system resides in the new concept of key image and the use of pseudo random algorithms for enhancing the security. We are also able to change the cover coefficients randomly. This strategy does not give any chance to steganalytic tools of searching for a predictable set of modifications. Also, considering the complexity of elections, we have provided sufficient proof of authenticity of an individual in form of both biometric measures and secret key.

The algorithm uses image based steganographic and cryptographic system are proposed. The Steganography part is needed as we want to involve biometric identity to provide added security. Mostly, Steganography uses images as cover media because after digitalization images contain the quantization noise which provides space to embed data.

In a voting system, whether electronic or using traditional paper ballots, the system should meet the following criteria: Anonymity, Tamper-resistant, Human factors. If other security is done well, electronic voting could be a great improvement over paper systems. Flaws in any of these aspects of a voting system, however, can lead to indecisive or incorrect election results. To handle the situation the concept of stegano image is used where the message is encoded onto an finger print image. There are some pre-requisites to support such a

system. Firstly, each and every individual in the country should be provided with a Personal Identification Number, such as SSN (Social Security numbers) in some countries. This is needed for maintenance of voter accounts in the database. Secondly, we need Thumb Impressions (fingerprint images) of all the individuals. Thirdly, during the account creation every individual will be provided with a system generated Secret key which he/she should not disclose to anybody. This will be needed to cast the vote. Assuming all voter's information in a country is securely collected, biometric reader available for voting, the system is online during the election period only, the methodology is as follows.

2.1 PROPOSED METHODOLOGY

As the system handles a large amount of database, the main constraints of system are storage media and its capacity. A good capacity hard disk will enable efficient running of system. Along with Pentium Microprocessors and other hardware requirements. The software on which the system is designed to work is under windows operating system, which is satisfied under technical requirement of the system.

The system haves two basic parts the admin and the voter. The admin will have the work of registering the information of all the voters as well as the candidates . It will also have the job of taking the finger print of the voters. At the time of the elections the server will be put ready and will be managed by the admin.

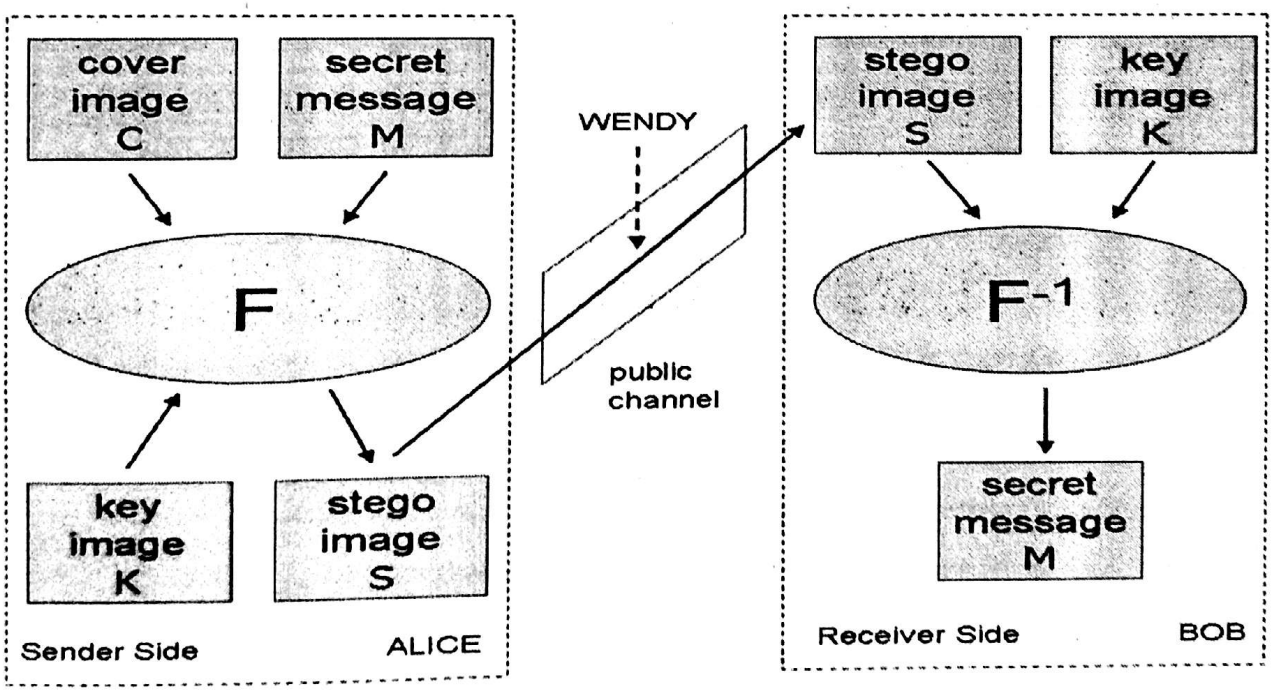
To cast a vote, a voter logs in to the system by entering the personal identification number and secret key. Along with this voter has to give the thumb impression on the fingerprint sensor. The system will generate the cover image and embed the secret key into it according to the predefined procedure to generate the stego image. Now this stego image will be sent securely to the

server for voter authentication. Fingerprint forgery may be restricted by using advanced fingerprint readers which employ Ultrasonic and Capacitance. At the server side, it will use the Optical Character Recognition technique to read the personal identification number represented on the image. After reading it, the server will find out the details of that individual from the database. These details will be his/her fingerprint image and secret key.

Using these details, the image can be decoded to find out the embedded message which should be the secret key of that individual. The mechanism has been shown in the Fig 2.1

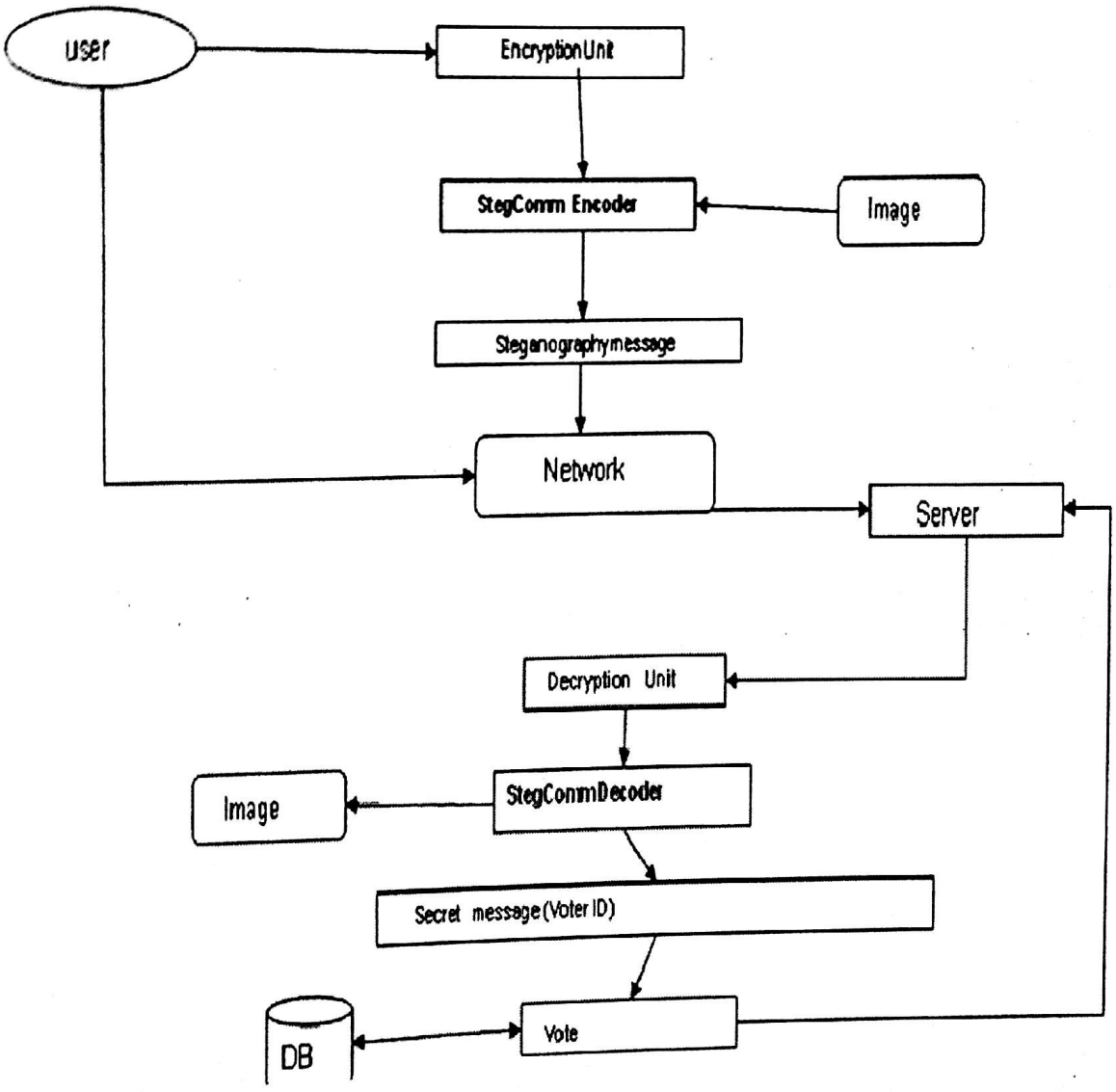
Once authentication is complete, the voter will be allowed to vote. In this next page, all the details regarding the voting boundaries of that individual will be shown. Here voter can select the desired candidate and finalize the vote.

After casting the vote, the account will be closed and in the database the voted bit will be set to one for that voter. Due to the setting of the voter count he/she won't be allowed to cast the vote twice.



2.1.1 SYSTEM ARCHITECTURE:

System architecture diagram is a simple diagrammatic form of the functions happening in the voting process , and it is used to understand the entire process as shown in the Fig 2.2



3.IMPLEMENTATION

The online voting system has two basic parts, the admin and the voter . Besides that, there are four special modules that makes the system functional while handling the security features.

1. Cover image creation
2. Secret key expansion
3. Generation of the secret message
4. Description of the embedding Algorithms

3.1 COVER IMAGE CREATION

The cover image creation will be under the voter usage time. The voter when logging in for the casting of his\her vote will have to give the passkey and the finger image. Every voter should have a personal identification number. This number will be provided at the time of registration and has to be kept confidential after that. The loosing of the identification number may result in non casting of the vote by the candidate. This number will be automatically written over a base image in predefined font style & size. Here only the idea of steganography will be implemented. It is the art and science of writing hidden messages in such a way that no one, apart from the sender and intended recipient, suspects the existence of the message, a form of security through obscurity.

Let us use 256*256 pixels bitmap cover image. The base image should be clear so that the text written over it is machine readable. So it has to be noted that the image used for the base should be a clear picture, avoid the darker images as not to give any option of no reading by the machine. This image will be finally modified into a stego image and sent over insecure channel.

The base image is a default image for the system, same for all. Cover image is a simple inscription of personal identification number over the base image. So,

the cover image for every voter will be same except the digits written over it as shown in fig 3.1.

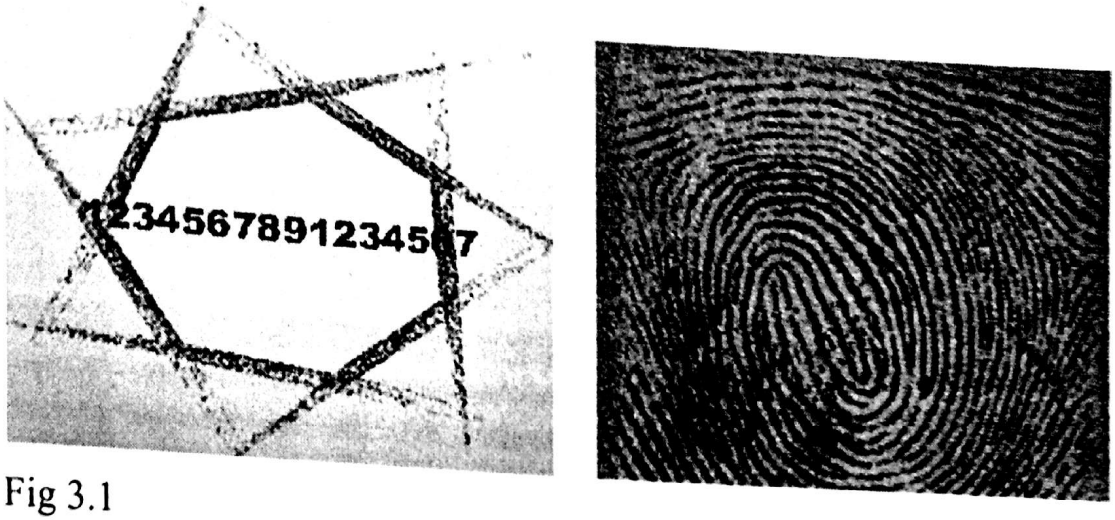


Fig 3.1

As it has been shown in the figure the base image is carrying the identification number which will be modified into the stego image, hard to be detected by any person. The choice of embedding algorithm in the most cases is driven by the results of the steganographic channel robustness analysis. One of the areas that improve steganographic robustness is usage of a key scheme for embedding messages. Various key steganographic schemes have various levels of protection. Key scheme term means a procedure of how to use key steganographic system based on the extent of its use. However, when the steganographic robustness is increased a bandwidth of the whole embedding system is decreased. Therefore the task of a scheme selection for achieving the optimal values of the steganographic system is not trivial. Embedding messages in steganographic system can be carried out without use of a key or with use of a key. To improve steganographic robustness key can be used as verification option. It can make an impact on the distribution of bits of a message within a container, as well as an impact on the procedure of forming a sequence of embedded bits of a message. The larger the cover message is (in data content terms—number of bits) relative to the hidden message, the easier it is to hide the latter. For this reason, digital pictures (which contain large amounts of data) are used to hide messages on the Internet and on other communication media. It

is not clear how commonly this is actually done. For example: a 24-bit bitmap will have 8 bits representing each of the three colour values (red, green, and blue) at each pixel. If we consider just the blue there will be 28 different values of blue. The difference between 11111111 and 11111110 in the value for blue intensity is likely to be undetectable by the human eye. Therefore, the least significant bit can be used (more or less undetectably) for something else other than colour information. If we do it with the green and the red as well we can get one letter of ASCII text for every three pixels. Stated somewhat more formally, the objective for making steganographic encoding difficult to detect is to ensure that the changes to the carrier (the original signal) due to the injection of the payload (the signal to covertly embed) are visually (and ideally, statistically) negligible; that is to say, the changes are indistinguishable from the noise floor of the carrier. Any medium can be a carrier, but media with a large amount of redundant or compressible information are better suited.

3.2 SECRET KEY EXPANSION USING HASHING

The secret key plays very important role in the whole process. It should not be compromised in any case. There is a limitation with the secret key here, as the system is designed for general public which is quite negligent in these issues. It should be short enough to be remembered by everybody, a single key is used for both encryption and decryption. The sender uses the key (or some set of rules) to encrypt the plaintext and sends the cipher text to the receiver. The receiver applies the same key to decrypt the message and recover the plaintext. Because a single key is used for both functions, secret key cryptography is also called symmetric encryption.

The eavesdroppers will never be able to deduce that some data is hidden in the image. But if somehow they know that it is a stego image, they can easily extract the PIN from it. So we apply SHA 256 hashing algorithm. The Secure

Hash Algorithm is one of a number of cryptographic hash functions. Now these 256 bits will become a part of the actual secret message. When the secret message is embedded in the cover image, its statistical properties will not remain same.

The stego image will remain more complex to be analyzed because more features of the keyimage are utilized in this case. So, even if eavesdroppers know that this is a stego image, it would be more difficult for them to predict the embedded data.

The secret key could be something o a 4 digit number .This 4-digit PIN can easily be represented using 2 bytes. But 2 byte data looks very much vulnerable in terms of length. As we have to finally embed it into the image, which is quite big. The cover image is a 24-bit image where every pixel is represented using three bytes. So, we have $3 * 2^{16}$ byte data in total. Now hiding only 2 bytes in this much space will not fully exploit the resources in terms of cryptography. This is because the algorithm we are using provides both cryptography and steganography at the same time. Steganography says its good as the statistical properties of the cover image will remain intact due to underperform modification. The eavesdroppers will never be able to deduce that some data is hidden in the image. But if somehow they know that it is a stego image, they can easily extract the PIN.

From the cryptography point of view, the key image will remain under utilized as well. As the fingerprint image is also of the same dimension, we will be exploiting very less features of the key image. So, to increase the complexity of analysis, the 2 byte secret key is expanded to 32 byte key by applying SHA 256 hashing algorithm. Now these 256 bits will become a part of the actual secret message. When the secret message is embedded in the cover image, its statistical properties will not remain same. The stego image will remain more complex to be analyzed because more features of the key image are utilized in

this case. So, even if eavesdroppers know that this is a stego image, it would be more difficult for them to predict the embedded data.

In cryptography, SHA-2 is a set of cryptographic hash functions (SHA-224, SHA-256, SHA-384, SHA-512) designed by the National Security Agency (NSA) and published in 2001 by the NIST as a U.S. Federal Information Processing Standard. SHA stands for Secure Hash Algorithm. SHA-2 includes a significant number of changes from its predecessor, SHA-1.

SHA-2 consists of a set of four hash functions with digests that are 224, 256, 384 or 512 bits.

SHA-256 is one of the successor hash functions to SHA-1 (collectively referred to as SHA-2), and is one of the strongest hash functions available. While SHA-1 has not been compromised in real-world conditions, SHA-256 is not much more complex to code, and has not yet been compromised in any way. The 256-bit key makes it a good partner-function for AES. It is defined in the NIST (National Institute of Standards and Technology) standard 'FIPS 180-2'. NIST also provide a number of test vectors to verify correctness of implementation.

3.3 GENERATION OF THE SECRET MESSAGE

In this phase of the methodology, we will get a 288 bit secret message from a 16 bit secret key. The secret key is concatenated with the time-stamp value. The timestamp is a 32 bit value which represents the current date. Now we will apply SHA 256 algorithm to get a 256 bit hash code for that key. Now the same time-stamp is concatenated with this hash code to get the secret message. So, our secret message will be of 288 bit length.

As the actual secret key is never embedded in the stego image, there will be no chance of predicting secret key from it.

The generation of the secret message takes place according the steps shown in Fig 3.2.

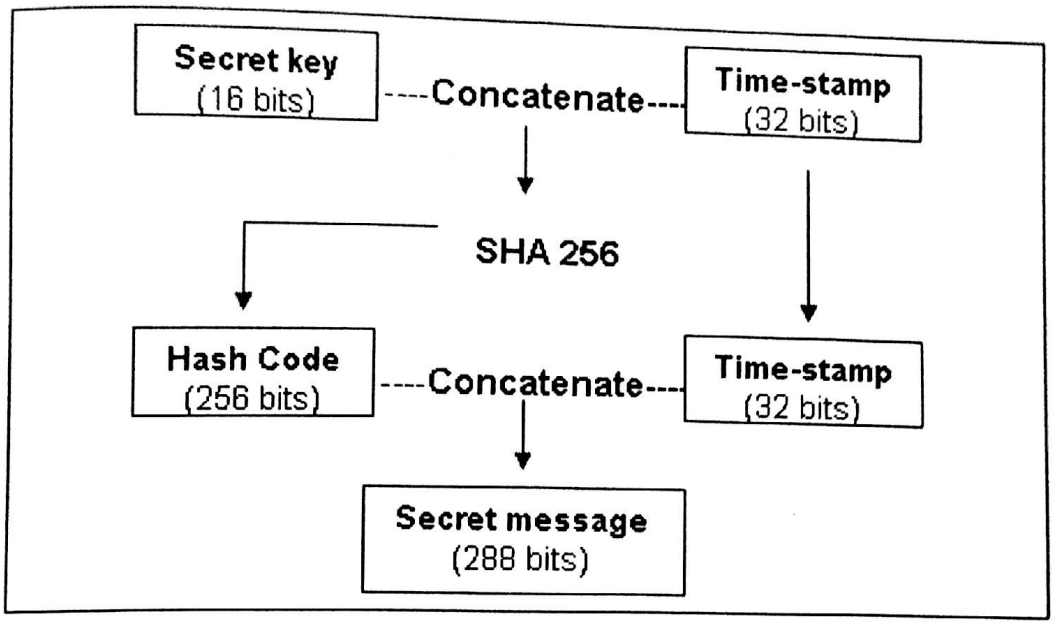


Fig 3.2

The algorithm that comes into play at this juncture are the pseudo random algorithms. Pseudorandom functions (PRFs) figure as central tools in the design of algorithms, especially those for key cryptography. At one level, PRFs can be used to model block ciphers, and they thereby enable the security analysis of algorithms based on block ciphers. A PRF is a deterministic function which is efficient and takes two inputs x, k. Now, we actually only consider x to be a variable and let k be a hidden random seed and function index, f (x, k) = fk (x).

x ---> f ---> fk(x)

|

K

A true random function is essentially a look-up table with random entries. The function x → fk (x) should look like a random function and if it does it's considered a good PRF.

The methodology of using pseudorandom function in the embedding algorithm will be used in the generation of the secret key.

This type of secret message generation using the pseudorandom function speeds up the algorithm, aiding in faster and secured retrieval of the secret message at the server side.

Significance of Pseudorandom Generators

The pseudo random generators are efficient amplifiers/expanders of randomness. Using very little randomness (a randomly chosen seed) they produce very long sequences which look random to any efficient observer. Pseudorandom generators allow to produce high quality random sequences at low costs making them very useful in cryptography. They produce unpredictable sequences i.e. no efficient algorithm can guess its next bit given a prefix of the sequence.

Cryptanalytic Attacks on Random functions

Examples of random parameters in cryptography could be Session key, numbers to be hashed with passwords, parameters in digital signatures and nonces.

Classes of Attacks on PRGs:

- Direct Cryptanalytic Attack:

When the attacker can directly distinguish between PRG numbers and random number, it is termed as direct attack . It is applicable to most PRG's. It may however not be applicable when the attacker is not able to directly see the output of the PRG. Eg A PRG used to generate triple-DES keys. Here the output of the PRG is never directly seen by an attacker.

- Input Based Attack:

It happens when the attacker is able to use knowledge and control of PRG inputs to cryptanalyze the PRG. Its types include: Known Input which is when if the inputs to the PRG, that are designed to be difficult for a user to guess, turn out to be easily deducible. Chosen input is practical against smartcards, applications that feed incoming messages (username/password etc) to the PRG as entropy samples. Replayed input is similar to chosen input, except it requires less sophistication on the part of the attacker.

- State Compromise Extension Attacks

Occurs when the attacker can guess some information due to an earlier breach of security. Attempts to extend the advantages of a temporary security breach These breaches can be:

  - Inadvertent leak
  - Previous cryptographic success

This attack is successful when the attacker learns the internal state of the system at state S. Able to recover unknown PRG outputs from before S was compromised. OR .Recover outputs from after a PRG has collected a sequence of inputs that an attacker cannot otherwise guess. These attacks usually succeed when the system is started in guessable state (due to lack of entropy).

Overcoming the attacks

- Hash functions usage in key generation
- Level of cryptography
- Deterministic algorithm
- Using a stronger counting mechanism

3.4 DESCRIPTION OF EMBEDDING ALGORITHM

The algorithms used are for two purposes, encryption and secret key generation. The image will be encrypted with the key and sent to the server. At the server the data will be read after decryption. The pseudo random mechanism generates the unique key.

3.4.1 LSB ALGORITHM

The algorithm is used for embedding the data into the bit map image . The algorithm works in 3 stages namely

- Encryption phase
- Transmission phase

• Decryption phase

The encryption phase is in process when the secret message is being encrypted along with the image. The transmission phase shows the transfer of the message from client to server. The last, Decryption phase is just the reverse of encryption where the message is read. The idea behind the LSB algorithm is to insert the bits of the hidden message into the least significant bits of the pixels.

Simplified Example with a 24 bit pixel:

1 pixel:

(00100111 11101001 11001000)

Insert 101:

(00100111 11101000 11001001)

Red Green Blue

Simplified Example with an 8 bit pixel:

1 pixel:

(00	01	10	11)
White	Red	Green	Blue

Insert 0011:

(00	00	11	11)
White	White	Blue	Blue

3.5 HARDWARE AND SOFTWARES ENVIRONMENT

HARDWARE REQUIREMENTS

The hardware requirements may serve as the basis for a contract for the implementation of the system and should therefore be a complete and consistent specification of the whole system. They are used by software engineers as the starting point for the system design. It should what the system do and not how it should be implemented.

PROCESSOR : PENTIUM IV 2.6 GHz, Intel Core 2 Duo.

RAM : 512 MB DD RAM

MONITOR : 15" COLOR

HARD DISK : 40 GB

INTERNET : WiFi Router

KEYBOARD : STANDARD 102 KEYS

BIOMETRIC DEVICE : FINGER PRINT READER

SOFTWARE REQUIREMENTS

The software requirements document is the specification of the system. It should include both a definition and a specification of requirements. It is a set of what the system should do rather than how it should do it. The software requirements provide a basis for creating the software requirements specification. It is useful in estimating cost, planning team activities, performing tasks and tracking the teams and tracking the team's progress throughout the development activity.

OPERATING SYSTEM : Windows XP or later versions

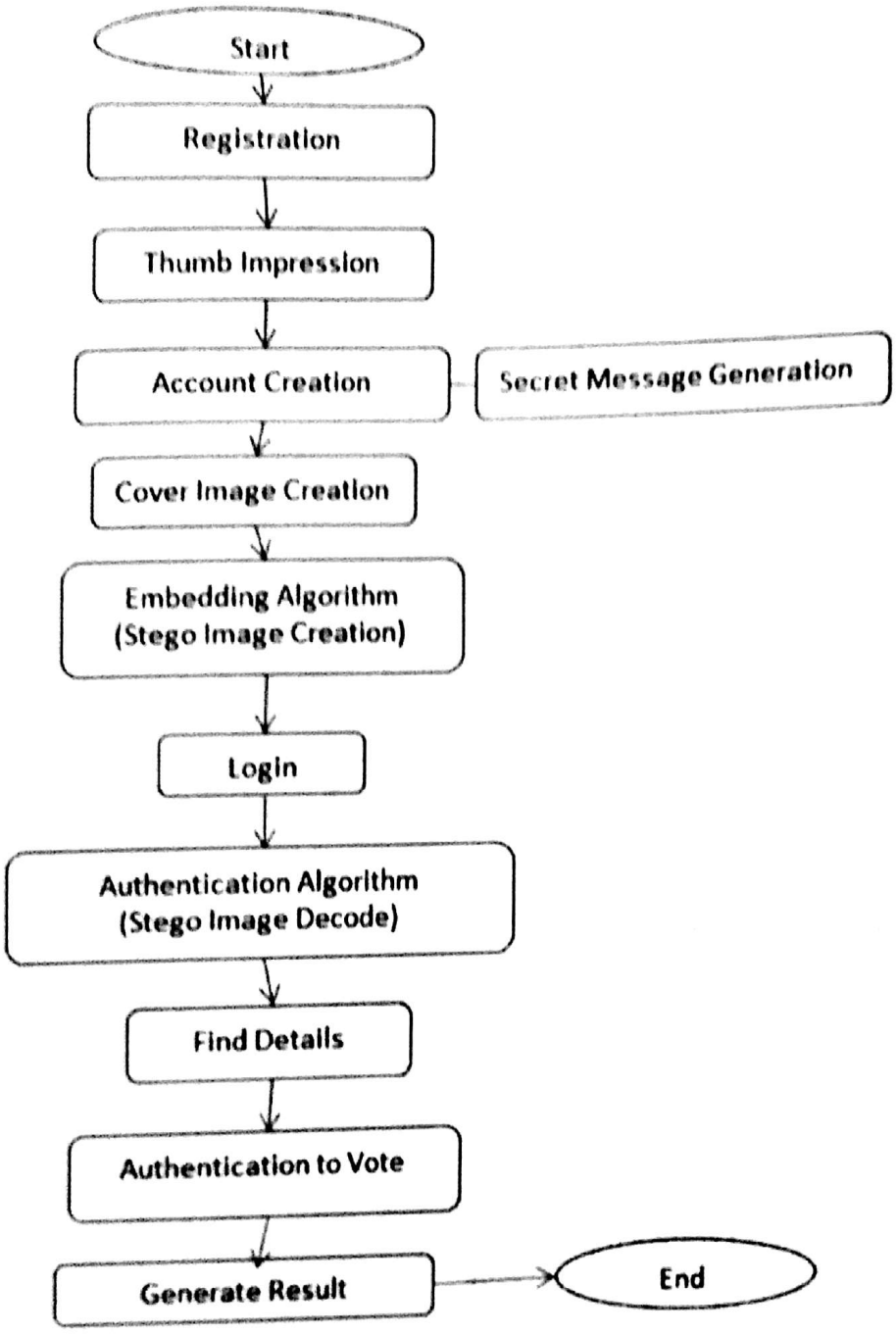
SOFTWARES : MATLAB, Microsoft Visual C++

3.6 VOTING PROCEDURE

1) Every individual in the country is first register for voting. So, our first step is the registration.

2) At the time of registration each person gives thumb impression for the security purpose by using Digital Persona Hardware.

3) After doing this system provide to person a personal identification number (PIN) and secrete key is generated as shown in figure.

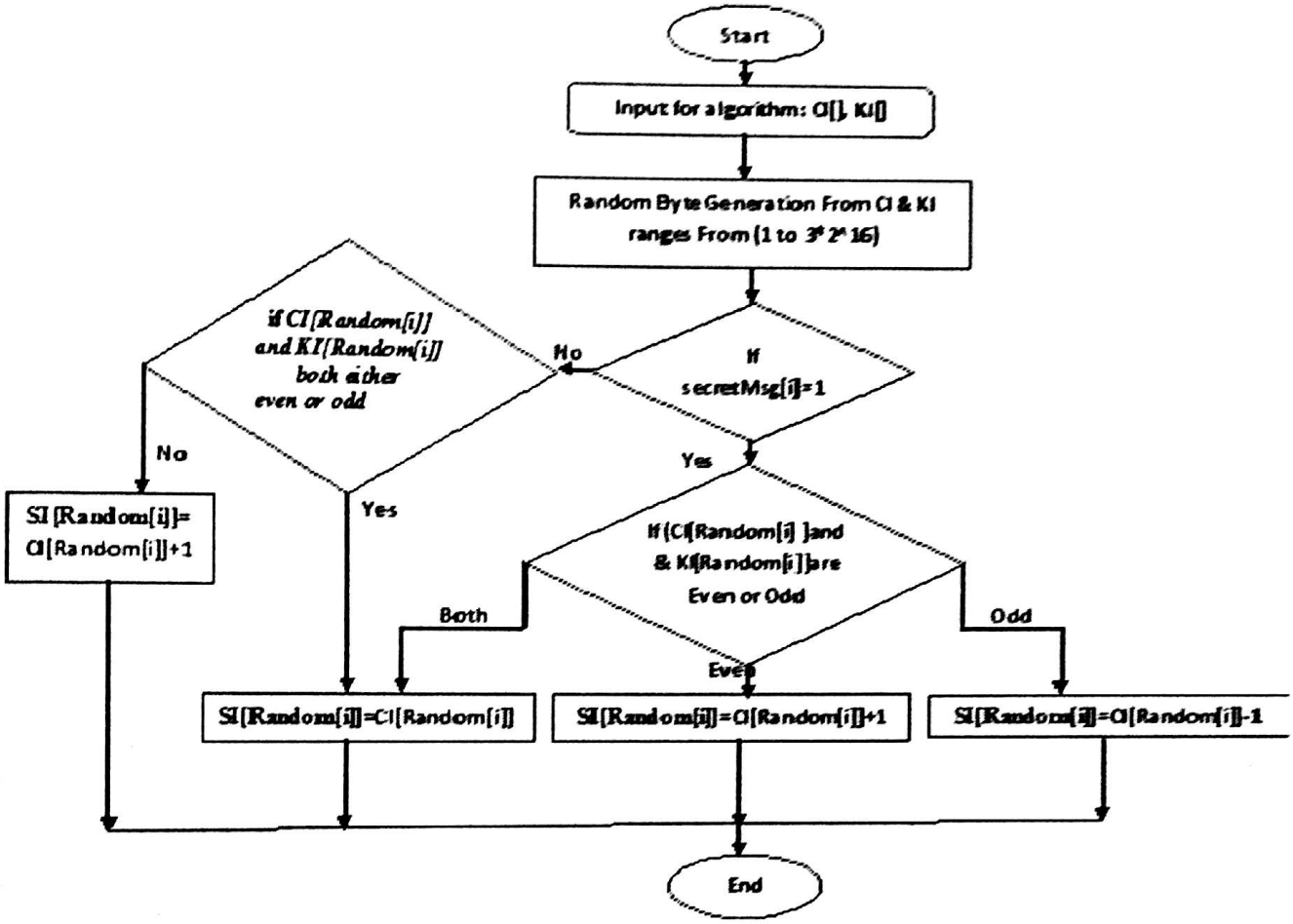


4) By using Personal identification number and secrete key with thumb gives cover image.

5) After that stego image is generated by using cover image

6) The registered user login to the system for voting by entering PIN and secrete key with thumb impression.

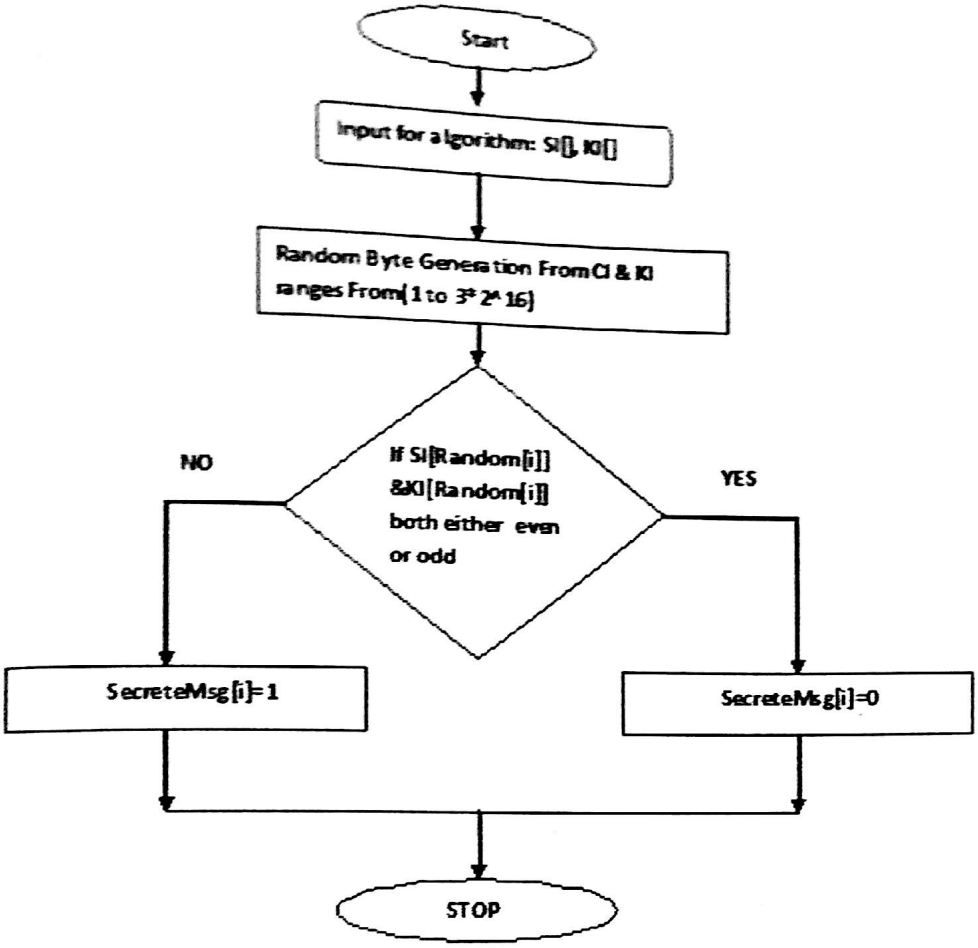
7) After login at the server side stego image will be decoded by using authentication algorithm as shown in fig.



8) Decoding the stego image details of the voter are find from the database at the server side.

8) Decoding the stego image details of the voter are find from the database at the server side.

10) Finally result is generated.


Start
Input for algorithm: SI[], KI[]
Random Byte Generation From CI & KI ranges From(1 to 3* 2^ 16)
If SI[Random[i]] &KI[Random[i]] both either even or odd
NO
YES
SecreteMsg[i]=1
SecreteMsg[i]=0
STOP

4.CONCLUSION

The focus of this paper is on use of techniques like cryptography and Steganography for fingerprint recognition also by using these techniques the highly secure voting system will be established. The security level of our system is greatly improved by the new idea of random cover image generation for each user.

The strength of our system resides in the new concept of key image. We are also able to change the cover coefficients randomly. This strategy does not give any chance to steganalytic tools of searching for a predictable set of modifications. Also, considering the complexity of elections, we have provided sufficient proof of authenticity and integrity of an individual in form of both biometric measures and secret key.

By using the biometric and password security the authentication process of system is highly improved. Thus citizens can be sure that they alone can choose their leaders, thus exercising their right in the democracy.

5.REFERENCES

1. *"Secure Online Voting System Proposed By Biometrics And Steganography "* . Nikita Malwade, Chetan Patil, Suruchi Chavan, Prof. Raut S. Y BE Computer, P.R.E.C., Loni. International Journal of Emerging Technology and Advanced Engineering (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 3, Issue 5, May 2013).
2. *"Online Voting System Powered By Biometric Security Using Steganography"*. Shivendra Katiyar, Kullai Reddy Meka, Ferdous A. Barbhuiya, Sukumar Nandi Department of Computer Science and Engineering Indian Institute of Technology Guwahati, India – 781039 2011 Second International Conference on Emerging Applications of Information Technology.
3. *"A Novel Data Hiding Technique based Bio-Secure Online Voting System"*. International Conference on Computing and Control Engineering (ICCCE 2012), 12 & 13 April, 2012 — ‖ Prabha Susy Mammen and S. Ramamoorthy .
4. *"E-Voting through Biometrics and Cryptography-Steganography Technique with conjunction of GSM Modem"*. Emerging Trends in Computer Science and Information Technology -2012(ETCSIT2012) Proceedings published in International Journal of Computer Applications (IJCA) by Shobha Lokhande.

6.APPENDIX

6.1 CODE IMLEMENTATION

CLIENT SIDE

FUNCTION CLIENTOPEN

```
function varargout = clientopen(varargin)
% CLIENTOPEN M-file for clientopen.fig
%      CLIENTOPEN, by itself, creates a new CLIENTOPEN or raises the
existing
%      singleton*.
%
%      H = CLIENTOPEN returns the handle to a new CLIENTOPEN or the handle
to
%      the existing singleton*.
%
%      CLIENTOPEN('CALLBACK',hObject,eventData,handles,...) calls the local
%          function named CALLBACK in CLIENTOPEN.M with the given input
arguments.
%
%          CLIENTOPEN('Property','Value',...) creates a new CLIENTOPEN or
raises the
%          existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before clientopen_OpeningFcn gets called.  An
%          unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to clientopen_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help clientopen

% Last Modified by GUIDE v2.5 18-Mar-2014 23:25:25

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @clientopen_OpeningFcn, ...
                   'gui_OutputFcn',  @clientopen_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```

```
% --- Executes just before clientopen is made visible.
function clientopen_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to clientopen (see VARARGIN)

% Choose default command line output for clientopen
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);



% UIWAIT makes clientopen wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = clientopen_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

FUNCTION HASH

```
function Hash = DataHash(Data, Opt)
% DATAHASH - Checksum for Matlab array of any type
% This function creates a hash value for an input of any type. The type and
% dimensions of the input are considered as default, such that UINT8([0,0])
and
% UINT16(0) have different hash values. Nested STRUCTs and CELLs are parsed
% recursively.
%
% Hash = DataHash(Data, Opt)
% INPUT:
%   Data: Array of these built-in types:
%           (U)INT8/16/32/64, SINGLE, DOUBLE, (real or complex)
%           CHAR, LOGICAL, CELL (nested), STRUCT (scalar or array, nested),
%           function_handle.
%   Opt:  Struct to specify the hashing algorithm and the output format.
%         Opt and all its fields are optional.
%         Opt.Method: String, known methods for Java 1.6 (Matlab 2009a):
%              'SHA-1', 'SHA-256', 'SHA-384', 'SHA-512', 'MD2', 'MD5'.
%           Known methods for Java 1.3 (Matlab 6.5):
%             'MD5', 'SHA-1'.
%           Default: 'MD5'.
%         Opt.Format: String specifying the output format:
%           'hex', 'HEX':      Lower/uppercase hexadecimal string.
%           'double', 'uint8': Numerical vector.
%           'base64':          Base64 encoded string, only printable
%                              ASCII characters, 33% shorter than 'hex'.
%           Default: 'hex'.
```

```
%         Opt.Input: Type of the input as string, not case-sensitive:
%             'array': The contents, type and size of the input [Data] are
%                      considered  for the creation of the hash. Nested
CELLs
%                      and STRUCT arrays are parsed recursively. Empty
arrays of
%                      different type reply different hashs.
%                'file':  [Data] is treated as file name and the hash is
calculated
%                      for the files contents.
%              'bin':   [Data] is a numerical, LOGICAL or CHAR array. Only
the
%                       binary contents of the array is considered, such
that
%                      e.g. empty arrays of different type reply the same
hash.
%             Default: 'array'.
%
% OUTPUT:
%   Hash: String, DOUBLE or UINT8 vector. The length depends on the hashing
%         method.
%
% EXAMPLES:
% % Default: MD5, hex:
%   DataHash([])                    % 7de5637fd217d0e44e0082f4d79b3e73
% % MD5, Base64:
%   Opt.Format = 'base64';
%   Opt.Method = 'MD5';
%   DataHash(int32(1:10), Opt)   % bKdecqzUpOrL4oxzk+cfyg
% % SHA-1, Base64:
%   S.a = uint8([]);
%   S.b = {{1:10}, struct('q', uint64(415))};
%   Opt.Method = 'SHA-1';
%   DataHash(S, Opt)             % ZMe4eUAp0G9TDrvSW0/Qc0gQ9/A
% % SHA-1 of binary values:
%   Opt.Method = 'SHA-1';
%   Opt.Input  = 'bin';
%   DataHash(1:8, Opt)           % 826cf9d3a5d74bbe415e97d4cecf03f445f69225
%
% NOTE:
%   Function handles and user-defined objects cannot be converted uniquely:
%     - The subfunction ConvertFuncHandle uses the built-in function
FUNCTIONS,
%     but the replied struct can depend on the Matlab version.
%   - It is tried to convert objects to UINT8 streams in the subfunction
%     ConvertObject. A conversion by STRUCT() might be more appropriate.
%   Adjust these subfunctions on demand.
%
%   MATLAB CHARs have 16 bits! In consequence the string 'hello' is treated
as
%   UINT16('hello') for the binary input method.
%
%     DataHash uses James Tursa's smart and fast TYPECASTX, if it is
installed:
%     http://www.mathworks.com/matlabcentral/fileexchange/17476
%   As fallback the built-in TYPECAST is used automatically, but for large
%   inputs this can be more than 100 times slower.
%   For Matlab 6.5 installing typecastx is obligatory to run DataHash.
%
% Tested: Matlab 6.5, 7.7, 7.8, 7.13, WinXP/32, Win7/64
%      Author:      Jan       Simon,      Heidelberg,      (C)      2011-2012
```

```
matlab.THISYEAR(a)nMINUSsimon.de
%
% See also: TYPECAST, CAST.
% FEX:
% Michael Kleder, "Compute Hash", no structs and cells:
%   http://www.mathworks.com/matlabcentral/fileexchange/8944
% Tim, "Serialize/Deserialize", converts structs and cells to a byte
stream:
%   http://www.mathworks.com/matlabcentral/fileexchange/29457
% Jan Simon, "CalcMD5", MD5 only, faster C-mex, no structs and cells:
%   http://www.mathworks.com/matlabcentral/fileexchange/25921

% $JRev: R-k V:011 Sum:kZG25iszfKbg Date:28-May-2012 12:48:06 $
% $License: BSD (use/copy/change/redistribute on own risk, mention the
author) $
% $File: Tools\GLFile\DataHash.m $
% History:
% 001: 01-May-2011 21:52, First version.
% 007: 10-Jun-2011 10:38, [Opt.Input], binary data, complex values
considered.
% 011: 26-May-2012 15:57, Fails for binary input and empty data.

% Main function:
% ============================================================
% Java is needed:
if ~usejava('jvm')
   error(['JSimon:', mfilename, ':NoJava'], ...
      '*** %s: Java is required.', mfilename);
end

% typecastx creates a shared data copy instead of the deep copy as Matlab's
% TYPECAST - for a [1000x1000] DOUBLE array this is 100 times faster!
persistent usetypecastx
if isempty(usetypecastx)
   usetypecastx = ~isempty(which('typecastx'));  % Run the slow WHICH once
only
end

% Default options: ------------------------------------------------------
-----
Method    = 'SHA-256';
OutFormat = 'hex';
isFile    = false;
isBin     = false;

% Check number and type of inputs: --------------------------------------
-----
nArg = nargin;
if nArg == 2
   if isa(Opt, 'struct') == 0   % Bad type of 2nd input:
      error(['JSimon:', mfilename, ':BadInput2'], ...
         '*** %s: 2nd input [Opt] must be a struct.', mfilename);
   end

   % Specify hash algorithm:
   if isfield(Opt, 'Method')
      Method = upper(Opt.Method);
   end

   % Specify output format:
```

```
   if isfield(Opt, 'Format')
      OutFormat = Opt.Format;
   end

   % Check if the Input type is specified - default: 'array':
   if isfield(Opt, 'Input')
      if strcmpi(Opt.Input, 'File')
         isFile = true;
         if ischar(Data) == 0
            error(['JSimon:', mfilename, ':CannotOpen'], ...
               '*** %s: 1st input is not a file name', mfilename);
         end

         if exist(Data, 'file') ~= 2
            error(['JSimon:', mfilename, ':FileNotFound'], ...
               '*** %s: File not found: %s.', mfilename, Data);
         end

      elseif strncmpi(Opt.Input, 'bin', 3)  % Accept 'binary'
         isBin = true;
         if (isnumeric(Data) || ischar(Data) || islogical(Data)) == 0
            error(['JSimon:', mfilename, ':BadDataType'], ...
               '*** %s: 1st input is not numeric, CHAR or LOGICAL.',
mfilename);
         end
      end
   end

elseif nArg ~= 1  % Bad number of arguments:
   error(['JSimon:', mfilename, ':BadNInput'], ...
      '*** %s: 1 or 2 inputs required.', mfilename);
end

% Create the engine: ------------------------------------------------
-----
try
   Engine = java.security.MessageDigest.getInstance(Method);
catch
   error(['JSimon:', mfilename, ':BadInput2'], ...
      '*** %s: Invalid algorithm: [%s].', mfilename, Method);
end

% Create the hash value: --------------------------------------------
-----
if isFile
   % Read the file and calculate the hash:
   FID = fopen(Data, 'r');
   if FID < 0
      error(['JSimon:', mfilename, ':CannotOpen'], ...
         '*** %s: Cannot open file: %s.', mfilename, Data);
   end
   Data = fread(FID, Inf, '*uint8');
   fclose(FID);

   Engine.update(Data);
   if usetypecastx
      Hash = typecastx(Engine.digest, 'uint8');
   else
      Hash = typecast(Engine.digest, 'uint8');
   end
```

```
elseif isBin                % Contents of an elementary array:
   if isempty(Data)           % Nothing to do, Engine.update fails for empty
input!
      Hash = typecastx(Engine.digest, 'uint8');
   elseif usetypecastx      % Faster typecastx:
      if isreal(Data)
         Engine.update(typecastx(Data(:), 'uint8'));
      else
         Engine.update(typecastx(real(Data(:)), 'uint8'));
         Engine.update(typecastx(imag(Data(:)), 'uint8'));
      end
      Hash = typecastx(Engine.digest, 'uint8');

   else                        % Matlab's TYPECAST is less elegant:
      if isnumeric(Data)
         if isreal(Data)
            Engine.update(typecast(Data(:), 'uint8'));
         else
            Engine.update(typecast(real(Data(:)), 'uint8'));
            Engine.update(typecast(imag(Data(:)), 'uint8'));
         end
      elseif islogical(Data)                % TYPECAST cannot handle LOGICAL
         Engine.update(typecast(uint8(Data(:)), 'uint8'));
      elseif ischar(Data)                   % TYPECAST cannot handle CHAR
         Engine.update(typecast(uint16(Data(:)), 'uint8'));
         Engine.update(typecast(Data(:), 'uint8'));
      end
      Hash = typecast(Engine.digest, 'uint8');
   end

elseif usetypecastx  % Faster typecastx:
   Engine = CoreHash_(Data, Engine);
   Hash   = typecastx(Engine.digest, 'uint8');

else                    % Slower built-in TYPECAST:
   Engine = CoreHash(Data, Engine);
   Hash   = typecast(Engine.digest, 'uint8');
end

% Convert hash specific output format: ------------------------------------
-----
switch OutFormat
   case 'hex'
      Hash = sprintf('%.2x', double(Hash));
   case 'HEX'
      Hash = sprintf('%.2X', double(Hash));
   case 'double'
      Hash = double(reshape(Hash, 1, []));
   case 'uint8'
      Hash = reshape(Hash, 1, []);
   case 'base64'
      Hash = fBase64_enc(double(Hash));
   otherwise
      error(['JSimon:', mfilename, ':BadOutFormat'], ...
         '*** %s: [Opt.Format] must be: HEX, hex, uint8, double, base64.',
...
         mfilename);
end
```

```
% return;

%
**************************************************************************
***
function Engine = CoreHash_(Data, Engine)
% This mothod uses the faster typecastx version.

% Consider the type and dimensions of the array to distinguish arrays with
the
% same data, but different shape: [0 x 0] and [0 x 1], [1,2] and [1;2],
% DOUBLE(0) and SINGLE([0,0]):
Engine.update([uint8(class(Data)), typecastx(size(Data), 'uint8')]);

if isstruct(Data)                          % Hash for all array elements and
fields:
   F      = sort(fieldnames(Data));  % Ignore order of fields
   Engine = CoreHash_(F, Engine);    % Catch the fieldnames

   for iS = 1:numel(Data)             % Loop over elements of struct array
      for iField = 1:length(F)        % Loop over fields
         Engine = CoreHash_(Data(iS).(F{iField}), Engine);
      end
   end

elseif iscell(Data)                   % Get hash for all cell elements:
   for iS = 1:numel(Data)
      Engine = CoreHash_(Data{iS}, Engine);
   end

elseif isnumeric(Data) || islogical(Data) || ischar(Data)
   if isempty(Data) == 0
      if isreal(Data)                 % TRUE for LOGICAL and CHAR also:
         Engine.update(typecastx(Data(:), 'uint8'));
      else                            % typecastx accepts complex input:
         Engine.update(typecastx(real(Data(:)), 'uint8'));
         Engine.update(typecastx(imag(Data(:)), 'uint8'));
      end
   end

elseif isa(Data, 'function_handle')
   Engine = CoreHash(ConvertFuncHandle(Data), Engine);

else  % Most likely this is a user-defined object:
   try
      Engine = CoreHash(ConvertObject(Data), Engine);
   catch
      warning(['JSimon:', mfilename, ':BadDataType'], ...
         ['Type of variable not considered: ', class(Data)]);
   end
end

% return;

%
**************************************************************************
***
function Engine = CoreHash(Data, Engine)
% This methods uses the slower TYPECAST of Matlab
```

```
% See CoreHash_ for comments.

Engine.update([uint8(class(Data)), typecast(size(Data), 'uint8')]);

if isstruct(Data)
fields:                                    % Hash for all array elements and
   F      = sort(fieldnames(Data));  % Ignore order of fields
   Engine = CoreHash(F, Engine);     % Catch the fieldnames
   for iS = 1:numel(Data)            % Loop over elements of struct array
      for iField = 1:length(F)       % Loop over fields
         Engine = CoreHash(Data(iS).(F{iField}), Engine);
      end
   end
elseif iscell(Data)                        % Get hash for all cell elements:
   for iS = 1:numel(Data)
      Engine = CoreHash(Data{iS}, Engine);
   end
elseif isempty(Data)
elseif isnumeric(Data)
   if isreal(Data)
      Engine.update(typecast(Data(:), 'uint8'));
   else
      Engine.update(typecast(real(Data(:)), 'uint8'));
      Engine.update(typecast(imag(Data(:)), 'uint8'));
   end
elseif islogical(Data)                     % TYPECAST cannot handle LOGICAL
   Engine.update(typecast(uint8(Data(:)), 'uint8'));
elseif ischar(Data)                        % TYPECAST cannot handle CHAR
   Engine.update(typecast(uint16(Data(:)), 'uint8'));
elseif isa(Data, 'function_handle')
   Engine = CoreHash(ConvertFuncHandle(Data), Engine);
else  % Most likely a user-defined object:
   try
      Engine = CoreHash(ConvertObject(Data), Engine);
   catch
      warning(['JSimon:', mfilename, ':BadDataType'], ...
         ['Type of variable not considered: ', class(Data)]);
   end
end

% return;

%
**********************************************************************
***
function FuncKey = ConvertFuncHandle(FuncH)
%   The subfunction ConvertFuncHandle converts function_handles to a struct
%    using the Matlab function FUNCTIONS. The output of this function
changes
%    with the Matlab version, such that DataHash(@sin) replies different
hashes
%   under Matlab 6.5 and 2009a.
%   An alternative is using the function name and name of the file for
%    function_handles, but this is not unique for nested or anonymous
functions.
%   If the MATLABROOT is removed from the file's path, at least the hash of
%   Matlab's toolbox functions is (usually!) not influenced by the version.
%    Finally I'm in doubt if there is a unique method to hash function
handles.
%   Please adjust the subfunction ConvertFuncHandles to your needs.
```

```
% The Matlab version influences the conversion by FUNCTIONS:
% 1. The format of the struct replied FUNCTIONS is not fixed,
% 2. The full paths of toolbox function e.g. for @mean differ.
FuncKey = functions(FuncH);

% ALTERNATIVE: Use name and path. The <matlabroot> part of the toolbox
functions
% is replaced such that the hash for @mean does not depend on the Matlab
% version.
% Drawbacks: Anonymous functions, nested functions...
% funcStruct = functions(FuncH);
% funcfile   = strrep(funcStruct.file, matlabroot, '<MATLAB>');
% FuncKey    = uint8([funcStruct.function, ' ', funcfile]);

% Finally I'm afraid there is no unique method to get a hash for a function
% handle. Please adjust this conversion to your needs.

% return;

%
**************************************************************************
***
function DataBin = ConvertObject(DataObj)
% Convert a user-defined object to a binary stream. There cannot be a
unique
% solution, so this part is left for the user...

% Perhaps a direct conversion is implemented:
DataBin = uint8(DataObj);

% Or perhaps this is better:
% DataBin = struct(DataObj);

% return;

%
**************************************************************************
***
function Out = fBase64_enc(In)
% Encode numeric vector of UINT8 values to base64 string.

Pool = [65:90, 97:122, 48:57, 43, 47];  % [0:9, a:z, A:Z, +, /]
v8   = [128; 64; 32; 16; 8; 4; 2; 1];
v6   = [32, 16, 8, 4, 2, 1];

In  = reshape(In, 1, []);
X   = rem(floor(In(ones(8, 1), :) ./ v8(:, ones(length(In), 1))), 2);
Y   = reshape([X(:); zeros(6 - rem(numel(X), 6), 1)], 6, []);
Out = char(Pool(1 + v6 * Y));

% return;
```

FUNCTION INVALID LOGIN

```
function varargout = invalidlogin(varargin)
% INVALIDLOGIN M-file for invalidlogin.fig
%       INVALIDLOGIN, by itself, creates a new INVALIDLOGIN or raises the
existing
```

```
%      singleton*.
%
%       H = INVALIDLOGIN returns the handle to a new INVALIDLOGIN or the
handle to
%      the existing singleton*.
%
%         INVALIDLOGIN('CALLBACK',hObject,eventData,handles,...) calls the
local
%         function named CALLBACK in INVALIDLOGIN.M with the given input
arguments.
%
%       INVALIDLOGIN('Property','Value',...) creates a new INVALIDLOGIN or
raises the
%       existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before invalidlogin_OpeningFcn gets called.  An
%          unrecognized property name or invalid value makes property
application
%          stop.   All inputs are passed to invalidlogin_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help invalidlogin

% Last Modified by GUIDE v2.5 01-Mar-2014 17:03:09

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @invalidlogin_OpeningFcn, ...
                   'gui_OutputFcn',  @invalidlogin_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before invalidlogin is made visible.
function invalidlogin_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to invalidlogin (see VARARGIN)

% Choose default command line output for invalidlogin
handles.output = hObject;
```

```
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes invalidlogin wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = invalidlogin_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close invalidlogin;
```

FUNCTION INVALID REG

```
function varargout = invalidreg(varargin)
% INVALIDREG M-file for invalidreg.fig
%        INVALIDREG, by itself, creates a new INVALIDREG or raises the
existing
%      singleton*.
%
%      H = INVALIDREG returns the handle to a new INVALIDREG or the handle
to
%      the existing singleton*.
%
%      INVALIDREG('CALLBACK',hObject,eventData,handles,...) calls the local
%         function named CALLBACK in INVALIDREG.M with the given input
arguments.
%
%         INVALIDREG('Property','Value',...) creates a new INVALIDREG or
raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before invalidreg_OpeningFcn gets called.  An
%         unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to invalidreg_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help invalidreg

% Last Modified by GUIDE v2.5 06-Mar-2014 22:21:09
```

```
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @invalidreg_OpeningFcn, ...
                   'gui_OutputFcn',  @invalidreg_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before invalidreg is made visible.
function invalidreg_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to invalidreg (see VARARGIN)

% Choose default command line output for invalidreg
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes invalidreg wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = invalidreg_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close invalidreg;
```

FUNCTION ISEVEN

```
function ise = iseven(x)
%ISEVEN True for even numbers.
```

```
%    ISEVEN(X) is 1 for the elements of X that are even and 0 for odd
elements.
%
%   Class support for input X:
%      float: double, single
%      integer: [u]int8, [u]int16, [u]int32, [u]int64
%
%   An error is raised if X is a float and is:
%   1) too large or small to determine if it is odd or even (this includes
-Inf
%      and +Inf), or
%   2) contains a fraction (e.g. isodd(1.5) raises an error), or
%   3) NaN (not-a-number).
%
%   See also ISODD.

% By Ulf Carlberg 29-MAY-2011.

ise = ~isodd(x);
```

FUCTION ISODD

```
function iso = isodd(x)
%ISODD True for odd numbers.
%    ISODD(X) is 1 for the elements of X that are odd and 0 for even
elements.
%
%   Class support for input X:
%      float: double, single
%      integer: [u]int8, [u]int16, [u]int32, [u]int64
%
%   An error is raised if X is a float and is:
%   1) too large or small to determine if it is odd or even (this includes
-Inf
%      and +Inf), or
%   2) contains a fraction (e.g. isodd(1.5) raises an error), or
%   3) NaN (not-a-number).
%
%   See also ISEVEN.

% By Ulf Carlberg 29-MAY-2011.

if isa(x, 'float') && isreal(x)
    ax = abs(double(x));
    if isa(x, 'single') && any(ax(:)>=2^24)
        % Also x=Inf or x=-Inf is will be taken care of here.
        error('isodd:InputOutOfRange', ...
            'The maximum value of abs(X) allowed is 2^24-1 for singles.');
    end
    if any(ax(:)>=2^53)
        % Also x=Inf or x=-Inf is will be taken care of here.
        error('isodd:InputOutOfRange', ...
            'The maximum value of abs(X) allowed is 2^53-1 for doubles.');
    end
    if any(isnan(x(:)))
        error('isodd:InputNaN', 'No entries of X are allowed to be NaN.');
    end
    if any(floor(ax(:))~=ax(:))
        error('isodd:InputNotInt', 'All entries of X must be integers.');
    end
    iso = logical(bitget(ax, 1));
```

```
elseif isa(x, 'integer')
    if isa(x, 'uint64')
        % MOD can not handle 64 bit numbers on some Matlab versions, but
BITGET
        % works on unsigned 64-bit integers.
        iso = logical(bitget(x, 1));
    elseif isa(x, 'int64')
        % MOD can not handle 64 bit numbers on some Matlab versions, but
ABS
        % works, and BITGET works on unsigned 64-bit integers.
        ax = uint64(abs(x));
        ax(x==-2^63) = 0;  % "the weird number" is even
        iso = logical(bitget(ax, 1));
    else
        iso = logical(mod(x, 2));
    end
else
    error('isodd:UnsupportedClass', 'Unsupported class.');
end
```

FUCTION IS ODD TEST

```
% Some tests for the ISODD function.

% Test vector and correct answer.
test_vector = 0:9;
aa = logical([0 1 0 1 0 1 0 1 0 1]);

% First simplest test.
try
    a1 = isodd(test_vector);
    if all(a1==aa)
        disp('Test 1 passed.');
    else
        disp('*** Test 1 failed. ***');
    end
catch me
    disp('*** Test 1 failed, an unexpected error was raised! ***');
end

% Other data types, positive and negative numbers.
try
    a1 = isodd(int8(test_vector-4));
    a2 = isodd(int16(test_vector-4));
    a3 = isodd(int32(test_vector-4));
    a4 = isodd(int64(test_vector-4));
    a5 = isodd(single(test_vector-4));
    a6 = isodd(double(test_vector-4));
    a7 = isodd(uint8(test_vector));
    a8 = isodd(uint16(test_vector));
    a9 = isodd(uint32(test_vector));
    a10 = isodd(uint64(test_vector));
    if all(a1==aa) && all(a2==aa) && all(a3==aa) && all(a4==aa) && ...
            all(a5==aa) && all(a6==aa) && all(a7==aa) && all(a8==aa) && ...
            all(a9==aa) && all(a10==aa)
        disp('Test 2 passed.');
    else
        disp('*** Test 2 failed. ***');
    end
catch me
    disp('*** Test 2 failed, an unexpected error was raised! ***');
```

```
end

% The "weird numbers".
try
    a1 = isodd(int8(-2^7));
    a2 = isodd(int16(-2^15));
    a3 = isodd(int32(-2^31));
    a4 = isodd(int64(-2^63));
    if ~a1 && ~a2 && ~a3 && ~a4
        disp('Test 3 passed.');
    else
        disp('*** Test 3 failed. ***');
    end
catch me
    disp('*** Test 3 failed, an unexpected error was raised! ***');
end

% Too large numbers.
try
    isodd(1e20);
    disp('*** Test 4 failed, no error was raised! ***');
catch me
    disp('Test 4 passed.');
end
try
    isodd(single(1e10));
    disp('*** Test 5 failed, no error was raised! ***');
catch me
    disp('Test 5 passed.');
end

% Special numbers where we want an error raised.
try
    isodd(1.1);
    disp('*** Test 6 failed, no error was raised! ***');
catch me
    disp('Test 6 passed.');
end
try
    isodd(Inf);
    disp('*** Test 7 failed, no error was raised! ***');
catch me
    disp('Test 7 passed.');
end
try
    isodd(-Inf);
    disp('*** Test 8 failed, no error was raised! ***');
catch me
    disp('Test 8 passed.');
end
try
    isodd(NaN);
    disp('*** Test 9 failed, no error was raised! ***');
catch me
    disp('Test 9 passed.');
end
try
    isodd(complex(sqrt(2), sqrt(2)));
    disp('*** Test 10 failed, no error was raised! ***');
catch me
    disp('Test 10 passed.');
```

```
end
try
    isodd(single([1e15 0 ; 0 0]));
    disp('*** Test 11 failed, no error was raised! ***');
catch me
    disp('Test 11 passed.');
end
```

FUNCTION ISKEYCHECK

```
function [ secretmsg ] = keycheck(key,timestamp )
%UNTITLED4 Summary of this function goes here
%   Detailed explanation goes here
key=timestamp*10^(4)+key;
shakey=DataHash(key);
shakey=hex2dec(shakey);
shakey=de2bi(shakey,256);
timestamp=de2bi(timestamp,32);
secretmsg=horzcat(shakey,timestamp);

end
```

FUCTION MAKEVOTE

```
function varargout = makevote(varargin)
% MAKEVOTE M-file for makevote.fig
%      MAKEVOTE, by itself, creates a new MAKEVOTE or raises the existing
%      singleton*.
%
%      H = MAKEVOTE returns the handle to a new MAKEVOTE or the handle to
%      the existing singleton*.
%
%      MAKEVOTE('CALLBACK',hObject,eventData,handles,...) calls the local
%          function named CALLBACK in MAKEVOTE.M with the given input
arguments.
%
%       MAKEVOTE('Property','Value',...) creates a new MAKEVOTE or raises
the
%       existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before makevote_OpeningFcn gets called.  An
%          unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to makevote_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help makevote

% Last Modified by GUIDE v2.5 10-Mar-2014 23:42:10

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @makevote_OpeningFcn, ...
                   'gui_OutputFcn',  @makevote_OutputFcn, ...
```

```
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before makevote is made visible.
function makevote_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to makevote (see VARARGIN)

% Choose default command line output for makevote
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes makevote wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = makevote_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in cand1.
function cand1_Callback(hObject, eventdata, handles)
% hObject    handle to cand1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of cand1



% --- Executes on button press in cand2.
function cand2_Callback(hObject, eventdata, handles)
% hObject    handle to cand2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hint: get(hObject,'Value') returns toggle state of cand2


% --- Executes on button press in submit.
function submit_Callback(hObject, eventdata, handles)
% hObject    handle to submit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
switch  get(get(handles.candidates,'SelectedObject'),'Tag')
    case 'cand1'
        vote='cand1';
    case 'cand2'
        vote='cand2';
    otherwise
        vote=[];
end
if (isempty(vote))
    close makevote;clear all;makevote;choosecandidate;return;
end
cand1=importdata('c:\cand1.mat');
cand2=importdata('c:\cand2.mat');
if(vote=='cand1')
    cand1=cand1+1;
elseif (vote=='cand2')
    cand2=cand2+1;
end
save('c:\cand1.mat','cand1');
save('c:\cand2.mat','cand2');
close makevote;clear all;voterlogin;



% --- Executes when selected object is changed in candidates.
function candidates_SelectionChangeFcn(hObject, eventdata, handles)
% hObject    handle to the selected object in candidates
% eventdata  structure with the following fields (see UIBUTTONGROUP)
%   EventName: string 'SelectionChanged' (read only)
%   OldValue: handle of the previously selected object or empty if none was
selected
%   NewValue: handle of the currently selected object
% handles    structure with handles and user data (see GUIDATA)
switch get(eventdata.NewValue,'Tag')
    case 'cand1'
        vote='cand1';
    case 'cand2'
        vote='cand2';
    otherwise
        vote=[];
end
```

FUCTION OPENINGFUNCTION

```
function [  ] = openingfunction(  )

    %these values to be those of your first computer:
ipA = '192.168.173.1'; portA = 9090;
%these values to be those of your second computer:
```

```
ipB = '192.168.173.223'; portB = 9091;
%% Create UDP Object
udpB = udp(ipA,portA,'LocalPort',portB);
%% Connect to UDP Object
clientopen
rec=receivestring(udpB);
if(strcmp(rec(1:numel('start poll')),'start poll'))
    clear all; close all hidden
    cand1=0;
cand2=0;save('c:\cand1.mat','cand1');
save('c:\cand2.mat','cand2');
    voterlogin;
    return;
elseif(strcmp(rec(1:numel('start registration')),'start registration'))
  clear all;close all hidden
   voterregister;
     return;

end
end
```

FUNCTION RECIEVE STRING

```
function [ string ] = receivestring(udpB)
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here
fopen(udpB);
while(1)
string=fscanf(udpB);
pause(1);
if(~isempty(string))
    break;
end
end
fclose(udpB);
end
```

FUNCTION REG FINISH

```
function varargout = regfinish(varargin)
% REGFINISH M-file for regfinish.fig
%      REGFINISH, by itself, creates a new REGFINISH or raises the existing
%      singleton*.
%
%      H = REGFINISH returns the handle to a new REGFINISH or the handle to
%      the existing singleton*.
%
%      REGFINISH('CALLBACK',hObject,eventData,handles,...) calls the local
%          function named CALLBACK in REGFINISH.M with the given input
arguments.
%
%       REGFINISH('Property','Value',...) creates a new REGFINISH or raises
the
%       existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before regfinish_OpeningFcn gets called.  An
%          unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to regfinish_OpeningFcn via varargin.
%
```

```
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help regfinish

% Last Modified by GUIDE v2.5 09-Mar-2014 21:14:27

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @regfinish_OpeningFcn, ...
                   'gui_OutputFcn',  @regfinish_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before regfinish is made visible.
function regfinish_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to regfinish (see VARARGIN)

% Choose default command line output for regfinish
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
prevhandle=get(0,'UserData');
voterid=getappdata(prevhandle.submit,'mydata');
key=get(prevhandle.submit,'UserData');
vname=get(prevhandle.votername,'UserData');
if(strcmp(voterid(3),'M'))
    sex1='MALE';
else
    sex1='FEMALE';
end
switch(voterid(1:2))
    case 'EC'
        dep='ELECTRONICS AND COMMUNICATION';
    case 'ME'
        dep='MECHANICAL';
    case 'CE'
        dep='CIVIL';
    case 'EE'
        dep='ELECTRICAL AND ELECTRONICS';
```

```
    otherwise
        dep='COMPUTER SCIENCE';
end
set(handles.name,'String',vname);
set(handles.id,'String',voterid);
set(handles.skey,'String',key);
set(handles.sex,'String',sex1)
set(handles.department,'String',dep)
% UIWAIT makes regfinish wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = regfinish_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes during object creation, after setting all properties.
function id_CreateFcn(hObject, eventdata, handles)
% hObject    handle to id (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes on button press in finish.
function finish_Callback(hObject, eventdata, handles)
% hObject    handle to finish (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
clear all;
close regfinish;
voterregister;
```

FUNCTION SECRETMSG GEN

```
function [ secretmsg ] = secretmsggen( key,timestamp )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here

timestamp(1)=timestamp(1)-2000;
timestamp(1)=10^(8)*timestamp(1)+10^(6)*timestamp(2)+10^(4)*timestamp(3)+10
^(2)*timestamp(4)+timestamp(5);
timestamp(2:6)=[];
key=timestamp*10^(4)+key;
shakey=DataHash(key);
shakey=hex2dec(shakey);
shakey=de2bi(shakey,256);
timestamp=de2bi(timestamp,32);
secretmsg=horzcat(shakey,timestamp);
end
```

FUNCTION TRANSFERTHIS

```
function [  ] = tranferthis( voterid,stegoimage,fingertransmit )
```

```
ipA = '192.168.173.223'; portA = 8090; % Modify these values to be those of
your first computer.
ipB = '192.168.173.1'; portB = 8091; % Modify these values to be those of
your second computer.
%% Create UDP Object
udpA = udp(ipB,portB,'LocalPort',portA);
%% Connect to UDP Object
udpA.Timeout=3000;
transmitstring(udpA,voterid);


ipA = '192.168.173.223'; portA = 9090; % Modify these values to be those of
your first computer.
ipB = '192.168.173.1'; portB = 9091; % Modify these values to be those of
your second computer.
%% Create UDP Object
udpA = udp(ipB,portB,'LocalPort',portA);
%% Connect to UDP Object
udpA.timeout=3000;
transmitdata(udpA,stegoimage);

ipA = '192.168.173.223'; portA = 7090; % Modify these values to be those of
your first computer.
ipB = '192.168.173.1'; portB = 7091; % Modify these values to be those of
your second computer.
%% Create UDP Object
udpA = udp(ipB,portB,'LocalPort',portA);
%% Connect to UDP Object
udpA.timeout=3000;
transmitdata(udpA,fingertransmit);
end
```

FUNCTION TRANSMIT DATA

```
function [  ] =transmitdata( udpA,data)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
data=reshape(data,1,3*256*256);
s=whos('data');
set(udpA,'OutputBufferSize',s.bytes);
fopen(udpA);
fwrite(udpA,data(:),'uint8');
fclose(udpA);
end
```

FUNCTION TRANSMITSTRING

```
function [ ] =transmitstring( udpA,string )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
fopen(udpA);
fprintf(udpA,string);
fclose(udpA);

end
```

FUNCTION VALID ADMIN

```
function varargout = validadmin(varargin)
% VALIDADMIN M-file for validadmin.fig
%      VALIDADMIN, by itself, creates a new VALIDADMIN or raises the
existing
%      singleton*.
%
%      H = VALIDADMIN returns the handle to a new VALIDADMIN or the handle
to
%      the existing singleton*.
%
%      VALIDADMIN('CALLBACK',hObject,eventData,handles,...) calls the local
%         function named CALLBACK in VALIDADMIN.M with the given input
arguments.
%
%         VALIDADMIN('Property','Value',...) creates a new VALIDADMIN or
raises the
%          existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before validadmin_OpeningFcn gets called.  An
%          unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to validadmin_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help validadmin

% Last Modified by GUIDE v2.5 02-Mar-2014 22:20:48

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @validadmin_OpeningFcn, ...
                   'gui_OutputFcn',  @validadmin_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before validadmin is made visible.
function validadmin_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to validadmin (see VARARGIN)
```

```
% Choose default command line output for validadmin
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);


% UIWAIT makes validadmin wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = validadmin_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close validadmin
```

FUNCTION VALIDKEY

```
function varargout = validkey(varargin)
% VALIDKEY M-file for validkey.fig
%      VALIDKEY, by itself, creates a new VALIDKEY or raises the existing
%      singleton*.
%
%      H = VALIDKEY returns the handle to a new VALIDKEY or the handle to
%      the existing singleton*.
%
%      VALIDKEY('CALLBACK',hObject,eventData,handles,...) calls the local
%          function named CALLBACK in VALIDKEY.M with the given input
arguments.
%
%       VALIDKEY('Property','Value',...) creates a new VALIDKEY or raises
the
%       existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before validkey_OpeningFcn gets called.  An
%          unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to validkey_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help validkey

% Last Modified by GUIDE v2.5 02-Mar-2014 14:49:51
```

```
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @validkey_OpeningFcn, ...
                   'gui_OutputFcn',  @validkey_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before validkey is made visible.
function validkey_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to validkey (see VARARGIN)

% Choose default command line output for validkey
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);


% UIWAIT makes validkey wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = validkey_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close validkey;
```

FUNCTION VOTERLOGIN

```
function varargout = voterlogin(varargin)
% VOTERLOGIN M-file for voterlogin.fig
%        VOTERLOGIN, by itself, creates a new VOTERLOGIN or raises the
existing
%      singleton*.
%
%      H = VOTERLOGIN returns the handle to a new VOTERLOGIN or the handle
to
%      the existing singleton*.
%
%      VOTERLOGIN('CALLBACK',hObject,eventData,handles,...) calls the local
%         function named CALLBACK in VOTERLOGIN.M with the given input
arguments.
%
%         VOTERLOGIN('Property','Value',...) creates a new VOTERLOGIN or
raises the
%       existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before voterlogin_OpeningFcn gets called.  An
%          unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to voterlogin_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help voterlogin

% Last Modified by GUIDE v2.5 05-Mar-2014 22:16:44

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @voterlogin_OpeningFcn, ...
                   'gui_OutputFcn',  @voterlogin_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before voterlogin is made visible.
function voterlogin_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to voterlogin (see VARARGIN)
```

```
% Choose default command line output for voterlogin
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes voterlogin wait for user response (see UIRESUME)


% --- Outputs from this function are returned to the command line.
function varargout = voterlogin_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



function voterid_Callback(hObject, eventdata, handles)
% hObject    handle to voterid (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of voterid as text
%        str2double(get(hObject,'String')) returns contents of voterid as a
double

voter= get(hObject,'String');
set(hObject,'UserData',voter);

% --- Executes during object creation, after setting all properties.
function voterid_CreateFcn(hObject, eventdata, handles)
% hObject    handle to voterid (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if          ispc          &&          isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function secretkey_Callback(hObject, eventdata, handles)
% hObject    handle to secretkey (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of secretkey as text
%        str2double(get(hObject,'String')) returns contents of secretkey as
a double
```

```
key=str2double(get(hObject,'String'));
set(hObject,'UserData',key);


% --- Executes during object creation, after setting all properties.
function secretkey_CreateFcn(hObject, eventdata, handles)
% hObject    handle to secretkey (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if          ispc          &&          isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in choosefingerprint.
function choosefingerprint_Callback(hObject, eventdata, handles)
% hObject    handle to choosefingerprint (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[filename,pathname]=uigetfile('*.bmp','Choose the fingerprint image');
set(hObject,'UserData',pathname);
setappdata(hObject,'mydata',filename);
try
    imread(fullfile(pathname,filename));
catch exception;
   if (exception.identifier)
       close voterlogin;
clear all;
voterlogin();
 invalidlogin
 return
   else
    throw(exception);
   end
end
keyimage=imread(fullfile(pathname,filename));
handles.img=keyimage;
axes(handles.axes2);
imshow(keyimage);
guidata(hObject,handles);


% --- Executes on button press in login.
function login_Callback(hObject, eventdata, handles)
% hObject    handle to login (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

voter=get(handles. voterid,'UserData');
if (isempty(voter))
 close voterlogin;
clear all;
voterlogin;invalidlogin
 return
end
key=get(handles.secretkey,'UserData');
```

```
if isnan(key)
  close voterlogin;clear all;voterlogin;validkey
  return
 end
if (key>9999)
 close voterlogin;clear all;voterlogin;validkey
 return
end
if (key<1000)
  close voterlogin;clear all;voterlogin;validkey
  return
end
    if ((key-floor(key))>0)
 close voterlogin;clear all;voterlogin;validkey
 return
    end
     if (isempty(key))
 close voterlogin;clear all;voterlogin;validkey
 return
    end
pathname=get(handles.choosefingerprint,'UserData');
filename=getappdata(handles.choosefingerprint,'mydata');
try
    imread(fullfile(pathname,filename));
catch exception;
   if (exception.identifier)
       close voterlogin;
clear all;
voterlogin();
 invalidlogin
 return
   else
    throw(exception);
   end
end
keyimage=imread(fullfile(pathname,filename));
keyimage=imresize(keyimage,[256,256]);
keyimage=reshape(keyimage,1,3*256*256);
try
    imread(fullfile('c:\stegoimage',voter));
catch exception;
   if (exception.identifier)
  close voterlogin;clear all;voterlogin;invalidlogin
 return
   else
    throw(exception);
   end
end
stegoimage=imread(fullfile('c:\stegoimage',voter));
stegoimage=reshape(stegoimage,1,3*256*256);
rand('seed',key);
randomnumber=randi(3*256*256,[1,288]);
for i=1:288

if((isodd(keyimage(randomnumber(i)))&&isodd(stegoimage(randomnumber(i))))||
(iseven(keyimage(randomnumber(i)))&&iseven(stegoimage(randomnumber(i))))==1
)
        newsecret(i)=0;
    else
        newsecret(i)=1;
    end
```

```
end
for i=257:288;
    timestamp(i-256)=newsecret(i);
end;
timestamp=bi2de(timestamp);
secretmsg=keycheck(key,timestamp);
if(secretmsg==newsecret)
    makevote;close voterlogin;delete(fullfile('c:\stegoimage',voter));
else
    close voterlogin;clear all;voterlogin;invalidlogin();
end

% --- Executes on button press in endpoll.
function endpoll_Callback(hObject, eventdata, handles)
% hObject    handle to endpoll (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
ipA = '192.168.173.223'; portA = 6090; % Modify these values to be those of
your first computer.
ipB = '192.168.173.1'; portB = 6091; % Modify these values to be those of
your second computer.
%% Create UDP Object
udpA = udp(ipB,portB,'LocalPort',portA);
udpA.Timeout=3000;
%% Connect to UDP Object
fopen(udpA);
fprintf(udpA,'end poll');
cand1=importdata('c:\cand1.mat');
cand2=importdata('c:\cand2.mat');cand1=num2str(cand1);cand2=num2str(cand2);

udpA.Timeout=3000;cand=[cand1 cand2];fprintf(udpA,cand);fclose(udpA);
close voterlogin;openingfunction();
```

FUNCTION VOTERREG

```
function varargout = voterregister(varargin)
% VOTERREGISTER M-file for voterregister.fig
%      VOTERREGISTER, by itself, creates a new VOTERREGISTER or raises the
existing
%      singleton*.
%
%      H = VOTERREGISTER returns the handle to a new VOTERREGISTER or the
handle to
%      the existing singleton*.
%
%      VOTERREGISTER('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in VOTERREGISTER.M with the given input
arguments.
%
%      VOTERREGISTER('Property','Value',...) creates a new VOTERREGISTER or
raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before voterregister_OpeningFcn gets called.  An
%          unrecognized property name or invalid value makes property
application
%          stop.  All inputs are passed to voterregister_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
```

```
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help voterregister

% Last Modified by GUIDE v2.5 29-Mar-2014 21:53:01

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @voterregister_OpeningFcn, ...
                   'gui_OutputFcn',  @voterregister_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before voterregister is made visible.
function voterregister_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to voterregister (see VARARGIN)

% Choose default command line output for voterregister
handles.output = hObject;
guidata(hObject, handles);

set(0,'UserData',handles);


% Update handles structure
% UIWAIT makes voterregister wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = voterregister_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

```
% --- Executes on selection change in sex.
function sex_Callback(hObject, eventdata, handles)
% hObject    handle to sex (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns sex contents as
cell array
%        contents{get(hObject,'Value')} returns selected item from sex
sex1=get(hObject,'Value');
switch(sex1)
    case 1
    voterid='M';
    case 2
    voterid='F';
end
setappdata(hObject,'mydata',voterid);

% --- Executes during object creation, after setting all properties.
function sex_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sex (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if          ispc          &&          isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes on selection change in dept.
function dept_Callback(hObject, eventdata, handles)
% hObject    handle to dept (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns dept contents as
cell array
%        contents{get(hObject,'Value')} returns selected item from dept

department=get(hObject,'Value');
switch(department)
    case 1
        voterid='EC';
    case 2
        voterid='ME';
    case 3
        voterid='CE';
    case 4
        voterid='EE';
    otherwise
        voterid='CS';
end
setappdata(hObject,'mydata',voterid);
% --- Executes during object creation, after setting all properties.
```

```
function dept_CreateFcn(hObject, eventdata, handles)
% hObject    handle to dept (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function votername_Callback(hObject, eventdata, handles)
% hObject    handle to votername (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of votername as text
%        str2double(get(hObject,'String')) returns contents of votername as
a double
voter=get(hObject,'String');
set(hObject,'UserData',voter);


% --- Executes during object creation, after setting all properties.
function votername_CreateFcn(hObject, eventdata, handles)
% hObject    handle to votername (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function admitno_Callback(hObject, eventdata, handles)
% hObject    handle to admitno (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of admitno as text
%        str2double(get(hObject,'String')) returns contents of admitno as a
double
admno=str2double(get(hObject,'String'));
set(hObject,'UserData',admno);

% --- Executes during object creation, after setting all properties.
function admitno_CreateFcn(hObject, eventdata, handles)
% hObject    handle to admitno (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if          ispc          &&          isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in submit.
function submit_Callback(hObject, eventdata, handles)
% hObject    handle to submit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
sex1=getappdata(handles.sex,'mydata');
if(isempty(sex1))
    sex1='M';
end
department=getappdata(handles.dept,'mydata');
if(isempty(department))
    department='EC';
end
voter=get(handles.votername,'UserData');
if(isempty(voter))
    invalidreg;
    return
end
admno=get(handles.admitno,'UserData');
if(isempty(admno))
    invalidreg
    return
end
if(isnan(admno))
    invalidreg
    return
end
pathname=get(handles.fingerprint,'UserData');
filename=getappdata(handles.fingerprint,'mydata');
try
    imread(fullfile(pathname,filename));
catch exception;
   if (exception.identifier)
 invalidlogin
 return
   else
    throw(exception);
   end
end
keyimage=imread(fullfile(pathname,filename));
key=ceil(1000+8999.*rand(1,1));
if(key==8154)
    key=8156;
elseif(key==8155)
    key=8157;
elseif(key==8160)
    key=8159;
elseif(key==1648)
    key=1234;
end
coverimage=imread('coverimage.bmp');
keyimage=imresize(keyimage,[256,256]);fingertransmit=keyimage;
coverimage=imresize(coverimage, [256, 256]);
```

```
keyimage=reshape(keyimage,1,3*256*256);
coverimage=reshape(coverimage,1,3*256*256);
stegoimage=coverimage;
rand('seed',key);
randomnumber=randi(3*256*256,[1,288]);
secretmsg=secretmsggen(key,clock);
for i=1:288
if(secretmsg(i)==1)

if(isodd(coverimage(randomnumber(i)))&&isodd(keyimage(randomnumber(i)))==1)
        stegoimage(randomnumber(i))=stegoimage(randomnumber(i))-1;

elseif(iseven(coverimage(randomnumber(i)))&&iseven(keyimage(randomnumber(i)
))==1)
        stegoimage(randomnumber(i))=stegoimage(randomnumber(i))+1;
    else
        stegoimage(randomnumber(i))=coverimage(randomnumber(i));
    end
elseif(secretmsg(i)==0)

if((isodd(coverimage(randomnumber(i)))&&isodd(keyimage(randomnumber(i))))||
(iseven(coverimage(randomnumber(i)))&&iseven(keyimage(randomnumber(i))))==1
)
        stegoimage(randomnumber(i))=coverimage(randomnumber(i));
    else
        stegoimage(randomnumber(i))=stegoimage(randomnumber(i))+1;
    end
end
end
stegoimage=reshape(stegoimage,256,256,3);
voterid=[department,sex1];
admno=int2str(admno);
voterid=[voterid,admno];
setappdata(hObject,'mydata',voterid);set(hObject,'UserData',key);

imwrite(stegoimage,fullfile('c:\stegoimage',voterid),'bmp');imwrite(fingert
ransmit,fullfile('c:\fingerprint',voterid),'bmp');
regfinish;close voterregister;
```

```
% --- Executes on button press in fingerprint.
function fingerprint_Callback(hObject, eventdata, handles)
% hObject    handle to fingerprint (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[filename,pathname]=uigetfile('*.bmp','Choose the fingerprint image');
set(hObject,'UserData',pathname);
setappdata(hObject,'mydata',filename);
try
    imread(fullfile(pathname,filename));
catch exception;
   if (exception.identifier)
clear all;
 invalidreg;
 return
   else
```

```
    throw(exception);
   end
end
keyimage=imread(fullfile(pathname,filename));
axes(handles.axes1);
imshow(keyimage);
guidata(hObject,handles);



% --- Executes on button press in endreg.
function endreg_Callback(hObject, eventdata, handles)
% hObject    handle to endreg (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

ipA = '192.168.173.223'; portA = 6090; % Modify these values to be those of
your first computer.
ipB = '192.168.173.1'; portB = 6091; % Modify these values to be those of
your second computer.
%% Create UDP Object
udpA = udp(ipB,portB,'LocalPort',portA);
udpA.Timeout=3000;
%% Connect to UDP Object
transmitstring(udpA,'end registration');
close voterregister;openingfunction();
```

SERVER SIDE

FUNCTION ADMIN AUTHENTICATION

```
function varargout = adminauthent2(varargin)
% ADMINAUTHENT2 M-file for adminauthent2.fig
%      ADMINAUTHENT2, by itself, creates a new ADMINAUTHENT2 or raises the
existing
%      singleton*.
%
%      H = ADMINAUTHENT2 returns the handle to a new ADMINAUTHENT2 or the
handle to
%      the existing singleton*.
%
%      ADMINAUTHENT2('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in ADMINAUTHENT2.M with the given input
arguments.
%
%      ADMINAUTHENT2('Property','Value',...) creates a new ADMINAUTHENT2 or
raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before adminauthent2_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to adminauthent2_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
```

```
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help adminauthent2

% Last Modified by GUIDE v2.5 02-Mar-2014 12:58:07

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @adminauthent2_OpeningFcn, ...
                   'gui_OutputFcn',  @adminauthent2_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});

end

% End initialization code - DO NOT EDIT


% --- Executes just before adminauthent2 is made visible.
function adminauthent2_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to adminauthent2 (see VARARGIN)

% Choose default command line output for adminauthent2
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
clear all

% UIWAIT makes adminauthent2 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = adminauthent2_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
clear all
```

```
function adminid_Callback(hObject, eventdata, handles)
% hObject    handle to adminid (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of adminid as text
%        str2double(get(hObject,'String')) returns contents of adminid as a
double
admin= get(hObject,'String');
set(hObject,'UserData',admin);


% --- Executes during object creation, after setting all properties.
function adminid_CreateFcn(hObject, eventdata, handles)
% hObject    handle to adminid (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if          ispc          &&          isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function skey_Callback(hObject, eventdata, handles)
% hObject    handle to skey (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of skey as text
%        str2double(get(hObject,'String')) returns contents of skey as a
double
key=str2double(get(hObject,'String'));
set(hObject,'UserData',key);


% --- Executes during object creation, after setting all properties.
function skey_CreateFcn(hObject, eventdata, handles)
% hObject    handle to skey (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if          ispc          &&          isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in choosefingerprint.
function choosefingerprint_Callback(hObject, eventdata, handles)
% hObject    handle to choosefingerprint (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[filename,pathname]=uigetfile('*.bmp','Choose the fingerprint image');
set(hObject,'UserData',pathname);
setappdata(hObject,'mydata',filename);
try
    imread(fullfile(pathname,filename));
catch exception;
   if (exception.identifier)
close adminauthent2;
clear all;
adminauthent2();
 invalidlogin
 return
   else
    throw(exception);
   end
end
keyimage=imread(fullfile(pathname,filename));
handles.axes2=keyimage;

imshow(keyimage);
guidata(hObject,handles);




% --- Executes on button press in login.
function login_Callback(hObject, eventdata, handles)
% hObject    handle to login (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
admin=get(handles. adminid,'UserData');
if (isempty(admin))
 close adminauthent2;clear all;adminauthent2;validadmin;
 return
end
key=get(handles.skey,'UserData');
if isnan(key)
  close adminauthent2;clear all;adminauthent2;validkey;
  return
 end
if (key>9999)
 close adminauthent2;clear all;adminauthent2;validkey;
 return
end
if (key<1000)
 close adminauthent2;clear all;adminauthent2; validkey;
  return
end
    if ((key-floor(key))>0)
 close adminauthent2;clear all;adminauthent2;validkey;
 return
    end
     if (isempty(key))
 close adminauthent2;clear all;adminauthent2;validkey;
 return
    end
pathname=get(handles.choosefingerprint,'UserData');
filename=getappdata(handles.choosefingerprint,'mydata');
```

```
try
    imread(fullfile(pathname,filename));
catch exception;
   if (exception.identifier)
close adminauthent2;
clear all;
adminauthent2();
 invalidlogin
 return
   else
    throw(exception);
   end
end
keyimage=imread(fullfile(pathname,filename));
keyimage=imresize(keyimage,[256,256]);
keyimage=reshape(keyimage,1,3*256*256);
try
    imread(admin);
catch exception;
   if (exception.identifier)
 close adminauthent2;clear all;adminauthent2;validadmin;
 return
   else
    throw(exception);
   end
end
stegoimage=imread(admin);
stegoimage=reshape(stegoimage,1,3*256*256);
rand('seed',key);
randomnumber=randi(3*256*256,[1,288]);
for i=1:288

if((isodd(keyimage(randomnumber(i)))&&isodd(stegoimage(randomnumber(i))))||
(iseven(keyimage(randomnumber(i)))&&iseven(stegoimage(randomnumber(i))))==1
)
        newsecret(i)=0;
    else
        newsecret(i)=1;
    end
end
for i=257:288;
    timestamp(i-256)=newsecret(i);
end;
timestamp=bi2de(timestamp);
secretmsg=keycheck(key,timestamp);
if(secretmsg==newsecret)
    adminchoice();close adminauthent2;
else
   close adminauthent2;
clear all;
adminauthent2();
 invalidlogin();
end




% --- Executes during object deletion, before destroying properties.
function axes2_DeleteFcn(hObject, eventdata, handles)
% hObject    handle to axes2 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

FUNCTION ADMIN CHOICE

```
function varargout = adminchoice(varargin)
% ADMINCHOICE M-file for adminchoice.fig
%      ADMINCHOICE, by itself, creates a new ADMINCHOICE or raises the
existing
%      singleton*.
%
%      H = ADMINCHOICE returns the handle to a new ADMINCHOICE or the
handle to
%      the existing singleton*.
%
%      ADMINCHOICE('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in ADMINCHOICE.M with the given input
arguments.
%
%      ADMINCHOICE('Property','Value',...) creates a new ADMINCHOICE or
raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before adminchoice_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to adminchoice_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help adminchoice

% Last Modified by GUIDE v2.5 27-Feb-2014 22:02:34

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @adminchoice_OpeningFcn, ...
                   'gui_OutputFcn',  @adminchoice_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```

```
% --- Executes just before adminchoice is made visible.
function adminchoice_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to adminchoice (see VARARGIN)

% Choose default command line output for adminchoice
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes adminchoice wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = adminchoice_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
mkdir('c:\stegoimage');mkdir('c:\fingerprint');
ipA = '192.168.173.1'; portA = 9090; % Modify these values to be those of
your first computer.
ipB = '192.168.173.223'; portB = 9091; % Modify these values to be those of
your second computer.
%% Create UDP Object
udpA = udp(ipB,portB,'LocalPort',portA);
%% Connect to UDP Object
transmitstring(udpA,'start registration')
close adminchoice
regfunction;


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
cand1=0;
cand2=0;save('c:\cand1.mat','cand1');
save('c:\cand2.mat','cand2');
ipA = '192.168.173.1'; portA = 9090; % Modify these values to be those of
your first computer.
ipB = '192.168.173.223'; portB = 9091; % Modify these values to be those of
```

```
your second computer.
%% Create UDP Object
udpA = udp(ipB,portB,'LocalPort',portA);
%% Connect to UDP Object
transmitstring(udpA,'start poll')
close adminchoice;
pollfunction;


% --------------------------------------------------------------------
function Untitled_1_Callback(hObject, eventdata, handles)
% hObject    handle to Untitled_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

FUNCTION DATAHASH

```
function Hash = DataHash(Data, Opt)
% DATAHASH - Checksum for Matlab array of any type
% This function creates a hash value for an input of any type. The type and
% dimensions of the input are considered as default, such that UINT8([0,0])
and
% UINT16(0) have different hash values. Nested STRUCTs and CELLs are parsed
% recursively.
%
% Hash = DataHash(Data, Opt)
% INPUT:
%   Data: Array of these built-in types:
%           (U)INT8/16/32/64, SINGLE, DOUBLE, (real or complex)
%           CHAR, LOGICAL, CELL (nested), STRUCT (scalar or array, nested),
%           function_handle.
%   Opt:  Struct to specify the hashing algorithm and the output format.
%         Opt and all its fields are optional.
%         Opt.Method: String, known methods for Java 1.6 (Matlab 2009a):
%              'SHA-1', 'SHA-256', 'SHA-384', 'SHA-512', 'MD2', 'MD5'.
%           Known methods for Java 1.3 (Matlab 6.5):
%             'MD5', 'SHA-1'.
%           Default: 'MD5'.
%         Opt.Format: String specifying the output format:
%           'hex', 'HEX':      Lower/uppercase hexadecimal string.
%           'double', 'uint8': Numerical vector.
%           'base64':          Base64 encoded string, only printable
%                              ASCII characters, 33% shorter than 'hex'.
%           Default: 'hex'.
%         Opt.Input: Type of the input as string, not case-sensitive:
%            'array': The contents, type and size of the input [Data] are
                      considered  for the creation of the hash. Nested
CELLs
%                      and STRUCT arrays are parsed recursively. Empty
arrays of
%                    different type reply different hashs.
%              'file':  [Data] is treated as file name and the hash is
calculated
%                    for the files contents.
%              'bin':   [Data] is a numerical, LOGICAL or CHAR array. Only
the
%                      binary contents of the array is considered, such
```

```
that
%                       e.g. empty arrays of different type reply the same
hash.
%             Default: 'array'.
%
% OUTPUT:
%   Hash: String, DOUBLE or UINT8 vector. The length depends on the hashing
%         method.
%
% EXAMPLES:
% % Default: MD5, hex:
%   DataHash([])
% % MD5, Base64:                          % 7de5637fd217d0e44e0082f4d79b3e73
%   Opt.Format = 'base64';
%   Opt.Method = 'MD5';
%   DataHash(int32(1:10), Opt)  % bKdecqzUpOrL4oxzk+cfyg
% % SHA-1, Base64:
%   S.a = uint8([]);
%   S.b = {{1:10}, struct('q', uint64(415))};
%   Opt.Method = 'SHA-1';
%   DataHash(S, Opt)                      % ZMe4eUAp0G9TDrvSW0/Qc0gQ9/A
% % SHA-1 of binary values:
%   Opt.Method = 'SHA-1';
%   Opt.Input  = 'bin';
%   DataHash(1:8, Opt)                    % 826cf9d3a5d74bbe415e97d4cecf03f445f69225
%
% NOTE:
%   Function handles and user-defined objects cannot be converted uniquely:
%     - The subfunction ConvertFuncHandle uses the built-in function
FUNCTIONS,
%     but the replied struct can depend on the Matlab version.
%   - It is tried to convert objects to UINT8 streams in the subfunction
%     ConvertObject. A conversion by STRUCT() might be more appropriate.
%   Adjust these subfunctions on demand.
%
%   MATLAB CHARs have 16 bits! In consequence the string 'hello' is treated
as
%   UINT16('hello') for the binary input method.
%
%     DataHash uses James Tursa's smart and fast TYPECASTX, if it is
installed:
%     http://www.mathworks.com/matlabcentral/fileexchange/17476
%   As fallback the built-in TYPECAST is used automatically, but for large
%   inputs this can be more than 100 times slower.
%   For Matlab 6.5 installing typecastx is obligatory to run DataHash.
%
% Tested: Matlab 6.5, 7.7, 7.8, 7.13, WinXP/32, Win7/64
%     Author:      Jan      Simon,      Heidelberg,      (C)      2011-2012
matlab.THISYEAR(a)nMINUSsimon.de
%
% See also: TYPECAST, CAST.
% FEX:
% Michael Kleder, "Compute Hash", no structs and cells:
%   http://www.mathworks.com/matlabcentral/fileexchange/8944
%  Tim, "Serialize/Deserialize", converts structs and cells to a byte
stream:
%   http://www.mathworks.com/matlabcentral/fileexchange/29457
% Jan Simon, "CalcMD5", MD5 only, faster C-mex, no structs and cells:
%   http://www.mathworks.com/matlabcentral/fileexchange/25921

% $JRev: R-k V:011 Sum:kZG25iszfKbg Date:28-May-2012 12:48:06 $
```

```
% $License: BSD (use/copy/change/redistribute on own risk, mention the
author) $
% $File: Tools\GLFile\DataHash.m $
% History:
% 001: 01-May-2011 21:52, First version.
%  007:  10-Jun-2011  10:38,  [Opt.Input],  binary  data,  complex  values
considered.
% 011: 26-May-2012 15:57, Fails for binary input and empty data.

%                                Main
% ------------------------------------------------------------------------
% Java is needed:                                                  function:
if ~usejava('jvm')
   error(['JSimon:', mfilename, ':NoJava'], ...
      '*** %s: Java is required.', mfilename);
end

% typecastx creates a shared data copy instead of the deep copy as Matlab's
% TYPECAST - for a [1000x1000] DOUBLE array this is 100 times faster!
persistent usetypecastx
if isempty(usetypecastx)
   usetypecastx = ~isempty(which('typecastx'));  % Run the slow WHICH once
only
end

% Default options: ------------------------------------------------------
-----
Method    = 'SHA-256';
OutFormat = 'hex';
isFile    = false;
isBin     = false;

% Check number and type of inputs: --------------------------------------
-----
nArg = nargin;
if nArg == 2
   if isa(Opt, 'struct') == 0   % Bad type of 2nd input:
      error(['JSimon:', mfilename, ':BadInput2'], ...
         '*** %s: 2nd input [Opt] must be a struct.', mfilename);
   end

   % Specify hash algorithm:
   if isfield(Opt, 'Method')
      Method = upper(Opt.Method);
   end

   % Specify output format:
   if isfield(Opt, 'Format')
      OutFormat = Opt.Format;
   end

   % Check if the Input type is specified - default: 'array':
   if isfield(Opt, 'Input')
      if strcmpi(Opt.Input, 'File')
         isFile = true;
         if ischar(Data) == 0
            error(['JSimon:', mfilename, ':CannotOpen'], ...
               '*** %s: 1st input is not a file name', mfilename);
         end
```

```
      if exist(Data, 'file') ~= 2
         error(['JSimon:', mfilename, ':FileNotFound'], ...
            '*** %s: File not found: %s.', mfilename, Data);
      end

   elseif strncmpi(Opt.Input, 'bin', 3)  % Accept 'binary'
      isBin = true;
      if (isnumeric(Data) || ischar(Data) || islogical(Data)) == 0
         error(['JSimon:', mfilename, ':BadDataType'], ...
            '*** %s: 1st input is not numeric, CHAR or LOGICAL.',
mfilename);
      end
   end
 end

elseif nArg ~= 1  % Bad number of arguments:
 error(['JSimon:', mfilename, ':BadNInput'], ...
   '*** %s: 1 or 2 inputs required.', mfilename);

end

% Create the engine: ------------------------------------------------
-----
try
 Engine = java.security.MessageDigest.getInstance(Method);
catch
 error(['JSimon:', mfilename, ':BadInput2'], ...
   '*** %s: Invalid algorithm: [%s].', mfilename, Method);
end

% Create the hash value: --------------------------------------------
-----
if isFile
 % Read the file and calculate the hash:
 FID = fopen(Data, 'r');
 if FID < 0
   error(['JSimon:', mfilename, ':CannotOpen'], ...
     '*** %s: Cannot open file: %s.', mfilename, Data);
 end
 Data = fread(FID, Inf, '*uint8');
 fclose(FID);

 Engine.update(Data);
 if usetypecastx
   Hash = typecastx(Engine.digest, 'uint8');
 else
   Hash = typecast(Engine.digest, 'uint8');
 end

elseif isBin
 if isempty(Data)          % Contents of an elementary array:
nput!                      % Nothing to do, Engine.update fails for empty

   Hash = typecastx(Engine.digest, 'uint8');
 elseif usetypecastx   % Faster typecastx:
   if isreal(Data)
     Engine.update(typecastx(Data(:), 'uint8'));
   else
     Engine.update(typecastx(real(Data(:)), 'uint8'));
     Engine.update(typecastx(imag(Data(:)), 'uint8'));
   end
   Hash = typecastx(Engine.digest, 'uint8');
```

```
  else                    % Matlab's TYPECAST is less elegant:
     if isnumeric(Data)
        if isreal(Data)
           Engine.update(typecast(Data(:), 'uint8'));
        else
           Engine.update(typecast(real(Data(:)), 'uint8'));
           Engine.update(typecast(imag(Data(:)), 'uint8'));
        end
     elseif islogical(Data)                % TYPECAST cannot handle LOGICAL
        Engine.update(typecast(uint8(Data(:)), 'uint8'));
     elseif ischar(Data)                   % TYPECAST cannot handle CHAR
        Engine.update(typecast(uint16(Data(:)), 'uint8'));
        Engine.update(typecast(Data(:), 'uint8'));
     end
     Hash = typecast(Engine.digest, 'uint8');
  end

elseif usetypecastx  % Faster typecastx:
  Engine = CoreHash_(Data, Engine);
  Hash   = typecastx(Engine.digest, 'uint8');

else                    % Slower built-in TYPECAST:
  Engine = CoreHash(Data, Engine);
  Hash   = typecast(Engine.digest, 'uint8');
end

% Convert hash specific output format: ------------------------------------
-----
switch OutFormat
  case 'hex'
     Hash = sprintf('%.2x', double(Hash));
  case 'HEX'
     Hash = sprintf('%.2X', double(Hash));
  case 'double'
     Hash = double(reshape(Hash, 1, []));
  case 'uint8'
     Hash = reshape(Hash, 1, []);
  case 'base64'
     Hash = fBase64_enc(double(Hash));
  otherwise
     error(['JSimon:', mfilename, ':BadOutFormat'], ...
        '*** %s: [Opt.Format] must be: HEX, hex, uint8, double, base64.',
...
        mfilename);
end

% return;

%
**********************************************************************
***
function Engine = CoreHash_(Data, Engine)
% This mothod uses the faster typecastx version.

% Consider the type and dimensions of the array to distinguish arrays with
the
% same data, but different shape: [0 x 0] and [0 x 1], [1,2] and [1;2],
% DOUBLE(0) and SINGLE([0,0]):
Engine.update([uint8(class(Data)), typecastx(size(Data), 'uint8')]);
```

```
if isstruct(Data)
fields:                                    % Hash for all array elements and
   F      = sort(fieldnames(Data));        % Ignore order of fields
   Engine = CoreHash_(F, Engine);          % Catch the fieldnames

   for iS = 1:numel(Data)                  % Loop over elements of struct array
      for iField = 1:length(F)             % Loop over fields
         Engine = CoreHash_(Data(iS).(F{iField}), Engine);
      end
   end

elseif iscell(Data)                        % Get hash for all cell elements:
   for iS = 1:numel(Data)
      Engine = CoreHash_(Data{iS}, Engine);
   end

elseif isnumeric(Data) || islogical(Data) || ischar(Data)
   if isempty(Data) == 0
      if isreal(Data)                      % TRUE for LOGICAL and CHAR also:
         Engine.update(typecastx(Data(:), 'uint8'));
      else                                 % typecastx accepts complex input:
         Engine.update(typecastx(real(Data(:)), 'uint8'));
         Engine.update(typecastx(imag(Data(:)), 'uint8'));
      end
   end

elseif isa(Data, 'function_handle')
   Engine = CoreHash(ConvertFuncHandle(Data), Engine);

else  % Most likely this is a user-defined object:
   try
      Engine = CoreHash(ConvertObject(Data), Engine);
   catch
      warning(['JSimon:', mfilename, ':BadDataType'], ...
         ['Type of variable not considered: ', class(Data)]);
   end
end

% return;

%
**************************************************************************
***
function Engine = CoreHash(Data, Engine)
% This methods uses the slower TYPECAST of Matlab
% See CoreHash_ for comments.

Engine.update([uint8(class(Data)), typecast(size(Data), 'uint8')]);

if isstruct(Data)                          % Hash for all array elements and
fields:
   F      = sort(fieldnames(Data));        % Ignore order of fields
   Engine = CoreHash(F, Engine);           % Catch the fieldnames
   for iS = 1:numel(Data)                  % Loop over elements of struct array
      for iField = 1:length(F)             % Loop over fields
         Engine = CoreHash(Data(iS).(F{iField}), Engine);
      end
   end
```

```
elseif iscell(Data)
   for iS = 1:numel(Data)                    % Get hash for all cell elements:
      Engine = CoreHash(Data{iS}, Engine);
   end
elseif isempty(Data)
elseif isnumeric(Data)
   if isreal(Data)
      Engine.update(typecast(Data(:), 'uint8'));
   else
      Engine.update(typecast(real(Data(:)), 'uint8'));
      Engine.update(typecast(imag(Data(:)), 'uint8'));
   end
elseif islogical(Data)                       % TYPECAST cannot handle LOGICAL
   Engine.update(typecast(uint8(Data(:)), 'uint8'));
elseif ischar(Data)                          % TYPECAST cannot handle CHAR
   Engine.update(typecast(uint16(Data(:)), 'uint8'));
elseif isa(Data, 'function_handle')
   Engine = CoreHash(ConvertFuncHandle(Data), Engine);
else  % Most likely a user-defined object:
   try
      Engine = CoreHash(ConvertObject(Data), Engine);
   catch
      warning(['JSimon:', mfilename, ':BadDataType'], ...
         ['Type of variable not considered: ', class(Data)]);
   end
end

% return;

%
**********************************************************************
***
function FuncKey = ConvertFuncHandle(FuncH)
%   The subfunction ConvertFuncHandle converts function_handles to a struct
%    using the Matlab function FUNCTIONS. The output of this function
changes
%    with the Matlab version, such that DataHash(@sin) replies different
hashes
%   under Matlab 6.5 and 2009a.
%   An alternative is using the function name and name of the file for
%    function_handles, but this is not unique for nested or anonymous
functions.
%   If the MATLABROOT is removed from the file's path, at least the hash of
%   Matlab's toolbox functions is (usually!) not influenced by the version.
%    Finally I'm in doubt if there is a unique method to hash function
handles.
%   Please adjust the subfunction ConvertFuncHandles to your needs.

% The Matlab version influences the conversion by FUNCTIONS:
% 1. The format of the struct replied FUNCTIONS is not fixed,
% 2. The full paths of toolbox function e.g. for @mean differ.
FuncKey = functions(FuncH);

% ALTERNATIVE: Use name and path. The <matlabroot> part of the toolbox
functions
% is replaced such that the hash for @mean does not depend on the Matlab
% version.
% Drawbacks: Anonymous functions, nested functions...
% funcStruct = functions(FuncH);
% funcfile   = strrep(funcStruct.file, matlabroot, '<MATLAB>');
```

```
% FuncKey     = uint8([funcStruct.function, ' ', funcfile]);

% Finally I'm afraid there is no unique method to get a hash for a function
% handle. Please adjust this conversion to your needs.

% return;

%
**************************************************************************
***
function DataBin = ConvertObject(DataObj)
% Convert a user-defined object to a binary stream. There cannot be a
unique
% solution, so this part is left for the user...

% Perhaps a direct conversion is implemented:
DataBin = uint8(DataObj);

% Or perhaps this is better:
% DataBin = struct(DataObj);

% return;

%
**************************************************************************
***
function Out = fBase64_enc(In)
% Encode numeric vector of UINT8 values to base64 string.

Pool = [65:90, 97:122, 48:57, 43, 47];  % [0:9, a:z, A:Z, +, /]
v8   = [128; 64; 32; 16; 8; 4; 2; 1];
v6   = [32, 16, 8, 4, 2, 1];

In  = reshape(In, 1, []);
X   = rem(floor(In(ones(8, 1), :) ./ v8(:, ones(length(In), 1))), 2);
Y   = reshape([X(:); zeros(6 - rem(numel(X), 6), 1)], 6, []);
Out = char(Pool(1 + v6 * Y));

% return;
```

FUNCTION ENDPOLL

```
function varargout = endpoll(varargin)
% ENDPOLL M-file for endpoll.fig
%      ENDPOLL, by itself, creates a new ENDPOLL or raises the existing
%      singleton*.
%
%      H = ENDPOLL returns the handle to a new ENDPOLL or the handle to
%      the existing singleton*.
%
%      ENDPOLL('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in ENDPOLL.M with the given input arguments.
%
%      ENDPOLL('Property','Value',...) creates a new ENDPOLL or raises the
%       existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before endpoll_OpeningFcn gets called.  An
%           unrecognized property name or invalid value makes property
```

```
application
%      stop.  All inputs are passed to endpoll_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help endpoll

% Last Modified by GUIDE v2.5 18-Mar-2014 23:20:03

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @endpoll_OpeningFcn, ...
                   'gui_OutputFcn',  @endpoll_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before endpoll is made visible.
function endpoll_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to endpoll (see VARARGIN)

% Choose default command line output for endpoll
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes endpoll wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = endpoll_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

```
% --- Executes on button press in endpoll.
function endpoll_Callback(hObject, eventdata, handles)
% hObject    handle to endpoll (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

FUNCTION ENDREG

```
function varargout = endregistration(varargin)
% ENDREGISTRATION M-file for endregistration.fig
%      ENDREGISTRATION, by itself, creates a new ENDREGISTRATION or raises
the existing
%      singleton*.
%
%      H = ENDREGISTRATION returns the handle to a new ENDREGISTRATION or
the handle to
%      the existing singleton*.
%
%      ENDREGISTRATION('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in ENDREGISTRATION.M with the given input
arguments.
%
%      ENDREGISTRATION('Property','Value',...) creates a new
ENDREGISTRATION or raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before endregistration_OpeningFcn gets called.
An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to endregistration_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help endregistration

% Last Modified by GUIDE v2.5 18-Mar-2014 13:41:18

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @endregistration_OpeningFcn, ...
                   'gui_OutputFcn',  @endregistration_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
```

```
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before endregistration is made visible.
function endregistration_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to endregistration (see VARARGIN)

% Choose default command line output for endregistration
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes endregistration wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = endregistration_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in endreg.
function endreg_Callback(hObject, eventdata, handles)
% hObject    handle to endreg (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

FUNCTIONFINGERPRINTMATCH

```
function [ match ] = Untitled( pic1,pic2 )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here

%so that we obtain white and black points and edges of the objects present
%in the picture.
pic1=imresize(pic1,[256,256]);pic2=imresize(pic2,[256,256]);
pic1=rgb2gray(pic1);pic2=rgb2gray(pic2);

edge_det_pic1  =  edge(pic1,'prewitt');%applying  edge  detection  on  first
picture
```

```
%so that we obtain white and black points and edges of the objects present
%in the picture.

edge_det_pic2 = edge(pic2,'prewitt');%%applying edge detection on second
picture

%initialization of different variables used
matched_data = 0;
white_points = 0;
black_points = 0;
x=0;
y=0;
l=0;
m=0;

%for loop used for detecting black and white points in the picture.
for a = 1:1:256
    for b = 1:1:256
        if(edge_det_pic1(a,b)==1)
            white_points = white_points+1;
        else
            black_points = black_points+1;
        end
    end
end

%for loop comparing the white (edge points) in the two pictures
for i = 1:1:256
    for j = 1:1:256
        if(edge_det_pic1(i,j)==1)&&(edge_det_pic2(i,j)==1)
            matched_data = matched_data+1;
            else
                ;
        end
    end
end



%calculating percentage matching.
total_data = white_points;
total_matched_percentage = (matched_data/total_data)*100;

%outputting the result of the system.
if(total_matched_percentage >= 90)              %can add flexability at this
point by reducing the amount of matching.

    match=1;
else

    match=0;
end



end
```

FUNCTION INVALIDLOGIN

```
function varargout = invalidlogin(varargin)
% INVALIDLOGIN M-file for invalidlogin.fig
%      INVALIDLOGIN, by itself, creates a new INVALIDLOGIN or raises the
existing
%      singleton*.
%
%      H = INVALIDLOGIN returns the handle to a new INVALIDLOGIN or the
handle to
%      the existing singleton*.
%
%      INVALIDLOGIN('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in INVALIDLOGIN.M with the given input
arguments.
%
%      INVALIDLOGIN('Property','Value',...) creates a new INVALIDLOGIN or
raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before invalidlogin_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to invalidlogin_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help invalidlogin

% Last Modified by GUIDE v2.5 01-Mar-2014 17:03:09

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @invalidlogin_OpeningFcn, ...
                   'gui_OutputFcn',  @invalidlogin_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before invalidlogin is made visible.
```

```
function invalidlogin_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to invalidlogin (see VARARGIN)

% Choose default command line output for invalidlogin
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes invalidlogin wait for user response (see UIRESUME)


% --- Outputs from this function are returned to the command line.
function varargout = invalidlogin_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close invalidlogin;
```

FUNCTION INVALID REG

```
function varargout = invalidreg(varargin)
% INVALIDREG M-file for invalidreg.fig
%      INVALIDREG, by itself, creates a new INVALIDREG or raises the
existing
%      singleton*.
%
%      H = INVALIDREG returns the handle to a new INVALIDREG or the handle
to
%      the existing singleton*.
%
%      INVALIDREG('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in INVALIDREG.M with the given input
arguments.
%
%      INVALIDREG('Property','Value',...) creates a new INVALIDREG or
raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before invalidreg_OpeningFcn gets called.  An
```

```
%          unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to invalidreg_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help invalidreg

% Last Modified by GUIDE v2.5 06-Mar-2014 22:21:09

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @invalidreg_OpeningFcn, ...
                   'gui_OutputFcn',  @invalidreg_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before invalidreg is made visible.
function invalidreg_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to invalidreg (see VARARGIN)

% Choose default command line output for invalidreg
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes invalidreg wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = invalidreg_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
```

```
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close invalidreg;
```

FUNCTION ISEVEN

```
function ise = iseven(x)
%ISEVEN True for even numbers.
%    ISEVEN(X) is 1 for the elements of X that are even and 0 for odd
elements.
%
%   Class support for input X:
%      float: double, single
%      integer: [u]int8, [u]int16, [u]int32, [u]int64
%
%   An error is raised if X is a float and is:
%   1) too large or small to determine if it is odd or even (this includes
-Inf
%      and +Inf), or
%   2) contains a fraction (e.g. isodd(1.5) raises an error), or
%   3) NaN (not-a-number).
%
%   See also ISODD.

% By Ulf Carlberg 29-MAY-2011.

ise = ~isodd(x);
```

FUNCTION ISODD

```
function iso = isodd(x)
%ISODD True for odd numbers.
%    ISODD(X) is 1 for the elements of X that are odd and 0 for even
elements.
%
%   Class support for input X:
%      float: double, single
%      integer: [u]int8, [u]int16, [u]int32, [u]int64
%
%   An error is raised if X is a float and is:
%   1) too large or small to determine if it is odd or even (this includes
-Inf
%      and +Inf), or
%   2) contains a fraction (e.g. isodd(1.5) raises an error), or
%   3) NaN (not-a-number).
%
%   See also ISEVEN.

% By Ulf Carlberg 29-MAY-2011.

if isa(x, 'float') && isreal(x)
```

```
    ax = abs(double(x));
    if isa(x, 'single') && any(ax(:)>=2^24)
        % Also x=Inf or x=-Inf is will be taken care of here.
        error('isodd:InputOutOfRange', ...
            'The maximum value of abs(X) allowed is 2^24-1 for singles.');
    end
    if any(ax(:)>=2^53)
        % Also x=Inf or x=-Inf is will be taken care of here.
        error('isodd:InputOutOfRange', ...
            'The maximum value of abs(X) allowed is 2^53-1 for doubles.');
    end
    if any(isnan(x(:)))
        error('isodd:InputNaN', 'No entries of X are allowed to be NaN.');
    end
    if any(floor(ax(:))~=ax(:))
        error('isodd:InputNotInt', 'All entries of X must be integers.');
    end
    iso = logical(bitget(ax, 1));
elseif isa(x, 'integer')
    if isa(x, 'uint64')
        % MOD can not handle 64 bit numbers on some Matlab versions, but
BITGET
        % works on unsigned 64-bit integers.
        iso = logical(bitget(x, 1));
    elseif isa(x, 'int64')
        % MOD can not handle 64 bit numbers on some Matlab versions, but
ABS
        % works, and BITGET works on unsigned 64-bit integers.
        ax = uint64(abs(x));
        ax(x==-2^63) = 0;  % "the weird number" is even
        iso = logical(bitget(ax, 1));
    else
        iso = logical(mod(x, 2));
    end
else
    error('isodd:UnsupportedClass', 'Unsupported class.');
end
```

FUNCTION IS ODD TEST

```
% Some tests for the ISODD function.

% Test vector and correct answer.
test_vector = 0:9;
aa = logical([0 1 0 1 0 1 0 1 0 1]);

% First simplest test.
try
    a1 = isodd(test_vector);
    if all(a1==aa)
        disp('Test 1 passed.');
    else
        disp('*** Test 1 failed. ***');
    end
catch me
    disp('*** Test 1 failed, an unexpected error was raised! ***');
end

% Other data types, positive and negative numbers.
try
```

```
    a1 = isodd(int8(test_vector-4));
    a2 = isodd(int16(test_vector-4));
    a3 = isodd(int32(test_vector-4));
    a4 = isodd(int64(test_vector-4));
    a5 = isodd(single(test_vector-4));
    a6 = isodd(double(test_vector-4));
    a7 = isodd(uint8(test_vector));
    a8 = isodd(uint16(test_vector));
    a9 = isodd(uint32(test_vector));
    a10 = isodd(uint64(test_vector));
    if all(a1==aa) && all(a2==aa) && all(a3==aa) && all(a4==aa) && ...
            all(a5==aa) && all(a6==aa) && all(a7==aa) && all(a8==aa) && ...
            all(a9==aa) && all(a10==aa)
        disp('Test 2 passed.');
    else
        disp('*** Test 2 failed. ***');
    end
catch me
    disp('*** Test 2 failed, an unexpected error was raised! ***');
end

% The "weird numbers".
try
    a1 = isodd(int8(-2^7));
    a2 = isodd(int16(-2^15));
    a3 = isodd(int32(-2^31));
    a4 = isodd(int64(-2^63));
    if ~a1 && ~a2 && ~a3 && ~a4
        disp('Test 3 passed.');
    else
        disp('*** Test 3 failed. ***');
    end
catch me
    disp('*** Test 3 failed, an unexpected error was raised! ***');
end

% Too large numbers.
try
    isodd(1e20);
    disp('*** Test 4 failed, no error was raised! ***');
catch me
    disp('Test 4 passed.');
end
try
    isodd(single(1e10));
    disp('*** Test 5 failed, no error was raised! ***');
catch me
    disp('Test 5 passed.');
end

% Special numbers where we want an error raised.
try
    isodd(1.1);
    disp('*** Test 6 failed, no error was raised! ***');
catch me
    disp('Test 6 passed.');
end
try
    isodd(Inf);
    disp('*** Test 7 failed, no error was raised! ***');
catch me
```

```
    disp('Test 7 passed.');
end
try
    isodd(-Inf);
    disp('*** Test 8 failed, no error was raised! ***');
catch me
    disp('Test 8 passed.');
end
try
    isodd(NaN);
    disp('*** Test 9 failed, no error was raised! ***');
catch me
    disp('Test 9 passed.');
end
try
    isodd(complex(sqrt(2), sqrt(2)));
    disp('*** Test 10 failed, no error was raised! ***');
catch me
    disp('Test 10 passed.');
end
try
    isodd(single([1e15 0 ; 0 0]));
    disp('*** Test 11 failed, no error was raised! ***');
catch me
    disp('Test 11 passed.');
end
```

FUNCTION KEYCHECK

```
function varargout = validkey(varargin)
% VALIDKEY M-file for validkey.fig
%      VALIDKEY, by itself, creates a new VALIDKEY or raises the existing
%      singleton*.
%
%      H = VALIDKEY returns the handle to a new VALIDKEY or the handle to
%      the existing singleton*.
%
%      VALIDKEY('CALLBACK',hObject,eventData,handles,...) calls the local
%          function named CALLBACK in VALIDKEY.M with the given input
arguments.
%
%       VALIDKEY('Property','Value',...) creates a new VALIDKEY or raises
the
%       existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before validkey_OpeningFcn gets called.  An
%          unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to validkey_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help validkey

% Last Modified by GUIDE v2.5 02-Mar-2014 14:49:51

% Begin initialization code - DO NOT EDIT
```

```
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @validkey_OpeningFcn, ...
                   'gui_OutputFcn',  @validkey_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before validkey is made visible.
function validkey_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to validkey (see VARARGIN)

% Choose default command line output for validkey
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);


% UIWAIT makes validkey wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = validkey_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close validkey;
```

FUNCTION POLLFUNCTION

```
function [ ] =pollfunction( )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
endpoll;

%these values to be those of your first computer:
ipA = '192.168.173.223'; portA = 6090;
%these values to be those of your second computer:
ipB = '192.168.173.1'; portB = 6091;
%% Create UDP Object
udpend = udp(ipA,portA,'LocalPort',portB);
udpend.Timeout=3000;
fopen(udpend);
while(1)
string=fscanf(udpend);
    pause(1);
    if(~isempty(string))
    if(strcmp(string(1:8),'end poll'))
      cand=fscanf(udpend);cand1=str2num(cand);
cand1=cand(1);cand1=str2num(cand1);cand2=cand(2);cand2=str2num(cand2);save(
'c:\cand1.mat','cand1');
save('c:\cand2.mat','cand2');
        fclose(udpend);
  close all hidden;
        result
        return;
    end
    end
end
end
```

FUNCTION RECIEVEDATA

```
function [data] = receivedata( udpB )
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here
udpB.InputBufferSize=1;
set(udpB,'Timeout',9000);
fopen(udpB);
data=fread(udpB,3*256*256,'uint8');
fclose(udpB);
data=reshape(data,256,256,3);
end
```

FUNCTION RECIEVESTRING

```
function [ string ] = receivestring(udpB)
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here
fopen(udpB);
while(1)
string=fscanf(udpB);
pause(1);
if(~isempty(string))
    break;
end
end
```

```
end

FUNCTION REGFUNSTION function [ ] = regfunction(  )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
endregistration;
while(1)
%these values to be those of your first computer:
ipA = '192.168.173.223'; portA = 6090;
%these values to be those of your second computer:
ipB = '192.168.173.1'; portB = 6091;
%% Create UDP Object
udpend = udp(ipA,portA,'LocalPort',portB);
udpend.Timeout=3000;
fopen(udpend);

ipA = '192.168.173.223'; portA = 8090;
%these values to be those of your second computer:
ipB = '192.168.173.1'; portB = 8091;
%% Create UDP Object
udpad = udp(ipA,portA,'LocalPort',portB);
udpad.Timeout=9000;
fopen(udpad);

%these values to be those of your first computer:
ipA = '192.168.173.223'; portA = 9090;
%these values to be those of your second computer:
ipB = '192.168.173.1'; portB = 9091;
%% Create UDP Object
udpst = udp(ipA,portA,'LocalPort',portB);
udpst.Timeout=9000;

%these values to be those of your first computer:
ipA = '192.168.173.223'; portA = 7090;
%these values to be those of your second computer:
ipB = '192.168.173.1'; portB = 7091;
%% Create UDP Object
udpf = udp(ipA,portA,'LocalPort',portB);
udpf.Timeout=90000;
 string=fscanf(udpend);
    pause(1);
    if(~isempty(string))
    if(strcmp(string(1:16),'end registration'))
       fclose(udpend);fclose(udpad);
        close all hidden;
        adminchoice;
        return;
    end
    end

%admin=fscanf(udpad);
%pause(1);

%if(~isempty(admin))
%    fclose(udpad);fclose(udpend);
%    admin=admin(1:9);
```

```
stegoimage=receivedata(udpst);imshow(stegoimage);
imwrite(stegoimage,fullfile('c:\stegoimage',admin),'bmp');

fingerprint=receivedata(udpf);
imwrite(fingerprint,fullfile('c:\fingerprint',admin),'bmp');
break;


end
end
```

FUNCTION RESULT

```
function varargout = result(varargin)
% RESULT M-file for result.fig
%      RESULT, by itself, creates a new RESULT or raises the existing
%      singleton*.
%
%      H = RESULT returns the handle to a new RESULT or the handle to
%      the existing singleton*.
%
%      RESULT('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in RESULT.M with the given input arguments.
%
%      RESULT('Property','Value',...) creates a new RESULT or raises the
%       existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before result_OpeningFcn gets called.  An
%           unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to result_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help result

% Last Modified by GUIDE v2.5 05-Mar-2014 22:15:58

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @result_OpeningFcn, ...
                   'gui_OutputFcn',  @result_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
```

```
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before result is made visible.
function result_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to result (see VARARGIN)

% Choose default command line output for result
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
cand1=importdata('c:\cand1.mat');cand1=int2str(cand1);
cand2=importdata('c:\cand2.mat');cand2=int2str(cand2);
set(handles.cand1vote,'String',cand1)
set(handles.cand2vote,'String',cand2)
% UIWAIT makes result wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = result_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

FUNCTION SECRETMSG

```
function [ secretmsg ] = secretmsggen( key,timestamp )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here

timestamp(1)=timestamp(1)-2000;
timestamp(1)=10^(8)*timestamp(1)+10^(6)*timestamp(2)+10^(4)*timestamp(3)+10
^(2)*timestamp(4)+timestamp(5);
timestamp(2:6)=[];
key=timestamp*10^(4)+key;
shakey=DataHash(key);
shakey=hex2dec(shakey);
shakey=de2bi(shakey,256);
timestamp=de2bi(timestamp,32);
secretmsg=horzcat(shakey,timestamp);
end
```

FUNCTION TRANSMIT DATA

```
function [  ] =transmitdata( udpA,data)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
data=reshape(data,1,3*256*256);
data=num2str(data);
s=whos('data');
set(udpA,'OutputBufferSize',s.bytes);
fopen(udpA);
fprintf(udpA,data);
fclose(udpA);
end
```

FUNCTION TRANSMITSTRING

```
function [ ] =transmitstring( udpA,string )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
fopen(udpA);
fprintf(udpA,string);
fclose(udpA);

end
```

FUNCTION VALID ADMIN

```
function varargout = validadmin(varargin)
% VALIDADMIN M-file for validadmin.fig
%          VALIDADMIN, by itself, creates a new VALIDADMIN or raises the
existing
%       singleton*.
%
%       H = VALIDADMIN returns the handle to a new VALIDADMIN or the handle
to
%       the existing singleton*.
%
%       VALIDADMIN('CALLBACK',hObject,eventData,handles,...) calls the local
%          function named CALLBACK in VALIDADMIN.M with the given input
arguments.
%
%          VALIDADMIN('Property','Value',...) creates a new VALIDADMIN or
raises the
%       existing singleton*.  Starting from the left, property value pairs
are
%       applied to the GUI before validadmin_OpeningFcn gets called.  An
%          unrecognized property name or invalid value makes property
application
%       stop.  All inputs are passed to validadmin_OpeningFcn via varargin.
%
%       *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%       instance to run (singleton)".
```

```
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help validadmin

% Last Modified by GUIDE v2.5 02-Mar-2014 22:20:48

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @validadmin_OpeningFcn, ...
                   'gui_OutputFcn',  @validadmin_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before validadmin is made visible.
function validadmin_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to validadmin (see VARARGIN)

% Choose default command line output for validadmin
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);


% UIWAIT makes validadmin wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = validadmin_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
```

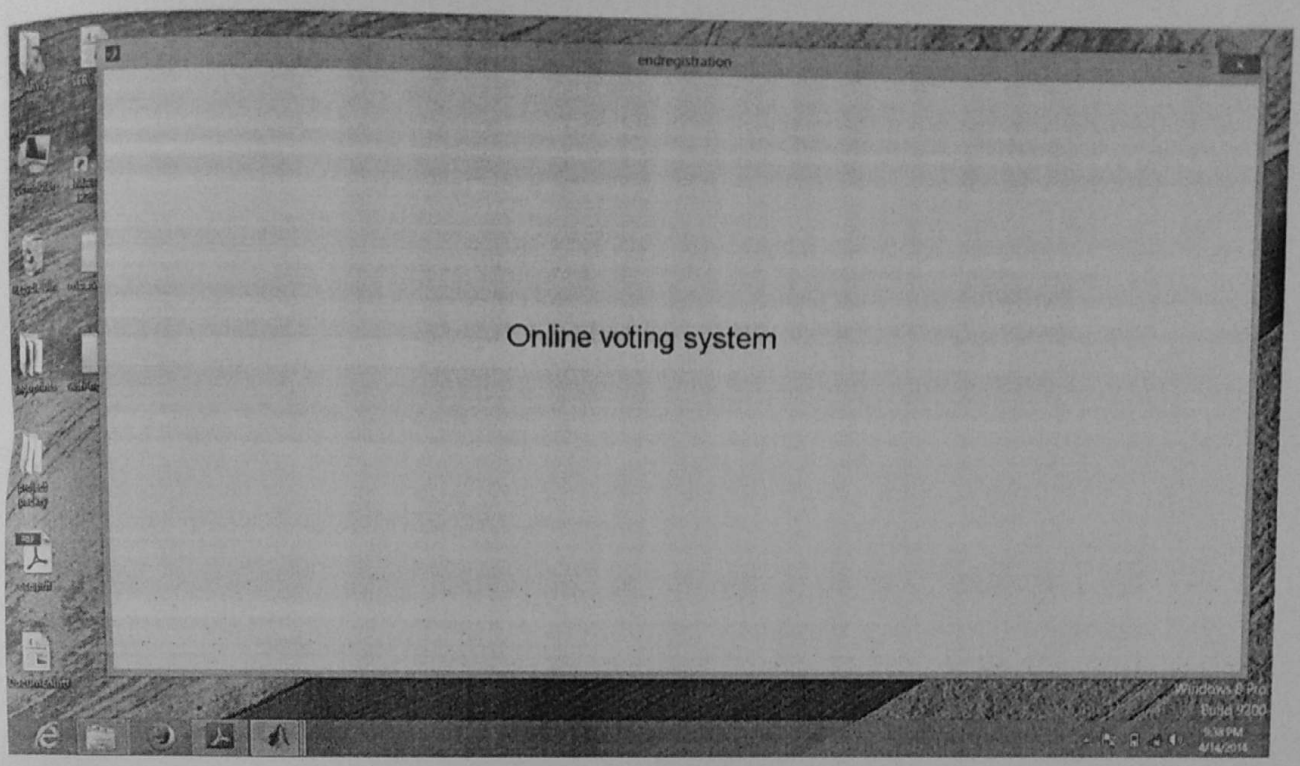
```
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close validadmin
```

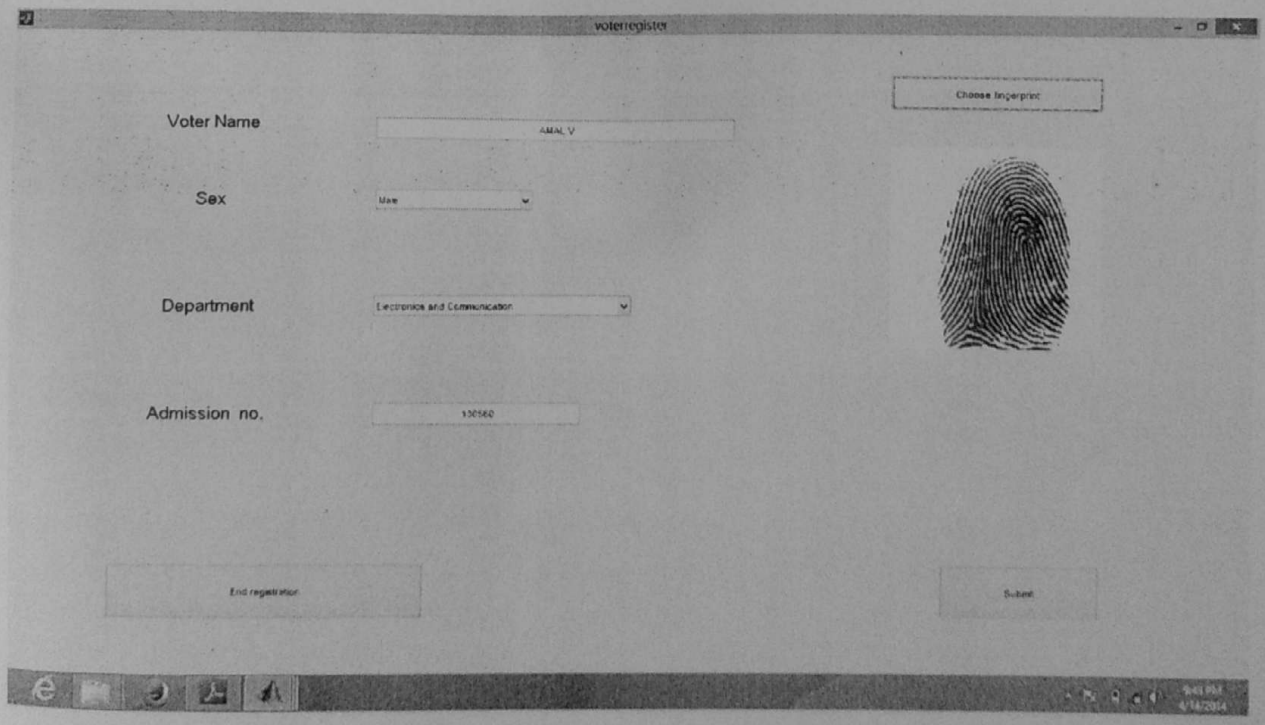
FUNCTION VALID KEY

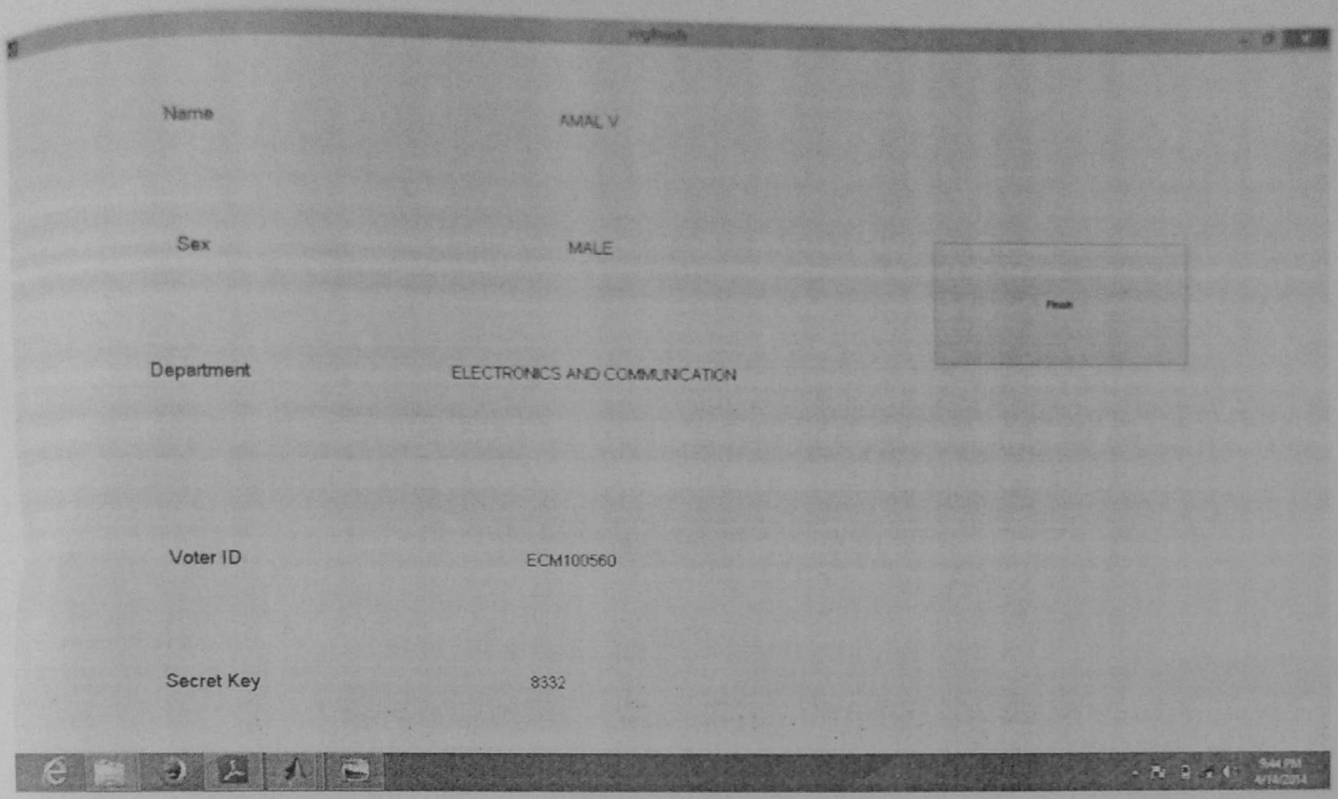
```
function varargout = validkey(varargin)
% VALIDKEY M-file for validkey.fig
%      VALIDKEY, by itself, creates a new VALIDKEY or raises the existing
%      singleton*.
%
%      H = VALIDKEY returns the handle to a new VALIDKEY or the handle to
%      the existing singleton*.
%
%      VALIDKEY('CALLBACK',hObject,eventData,handles,...) calls the local
%          function named CALLBACK in VALIDKEY.M with the given input
arguments.
%
%       VALIDKEY('Property','Value',...) creates a new VALIDKEY or raises
the
%       existing singleton*.  Starting from the left, property value pairs
are
%       applied to the GUI before validkey_OpeningFcn gets called.  An
%          unrecognized property name or invalid value makes property
application
%       stop.  All inputs are passed to validkey_OpeningFcn via varargin.
%
%       *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%       instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help validkey

% Last Modified by GUIDE v2.5 02-Mar-2014 14:49:51

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @validkey_OpeningFcn, ...
                   'gui_OutputFcn',  @validkey_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before validkey is made visible.
function validkey_OpeningFcn(hObject, eventdata, handles, varargin)
```

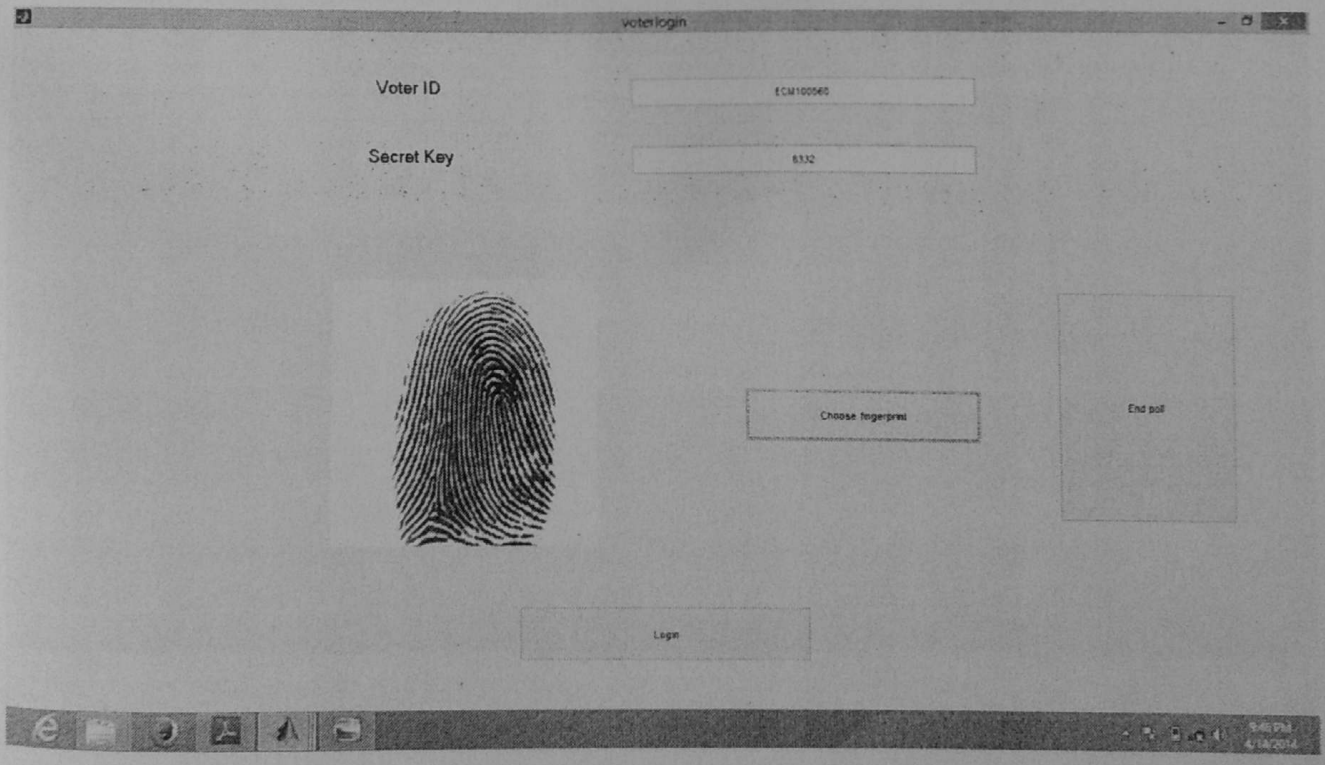
```
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to validkey (see VARARGIN)

% Choose default command line output for validkey
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);


% UIWAIT makes validkey wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = validkey_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close validkey;
```

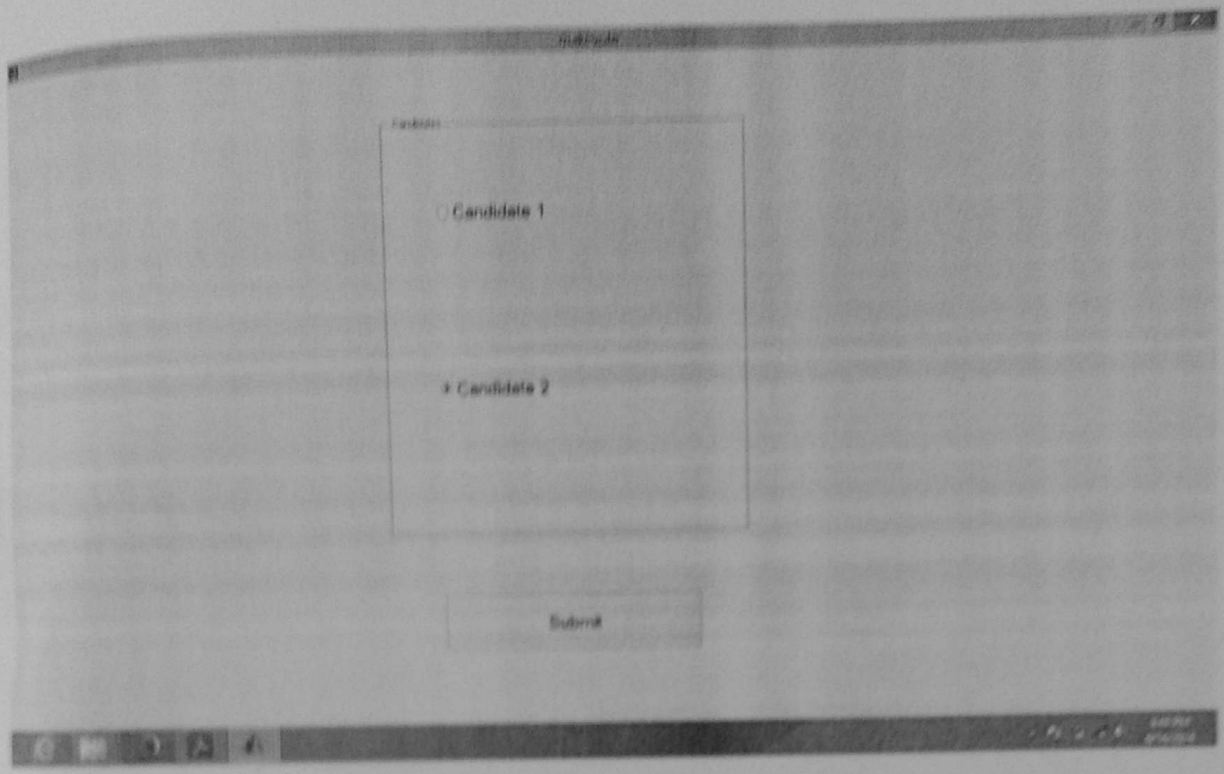
6.2 SNAPSHOTS

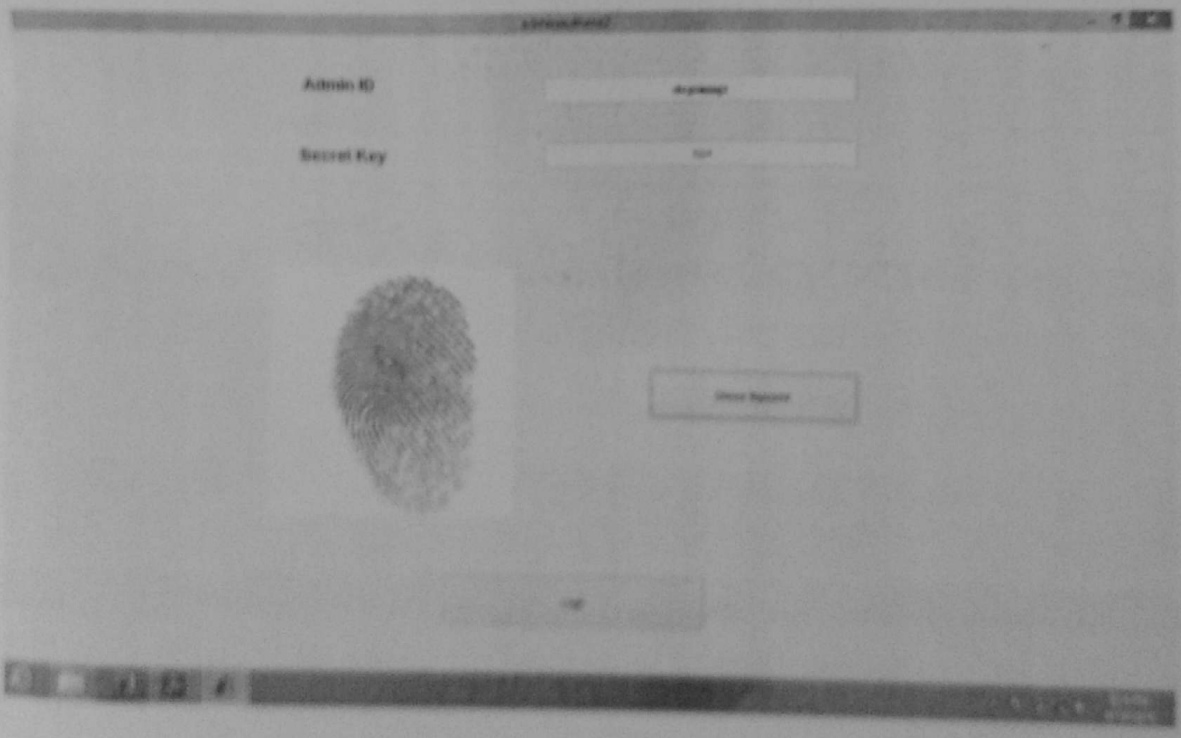


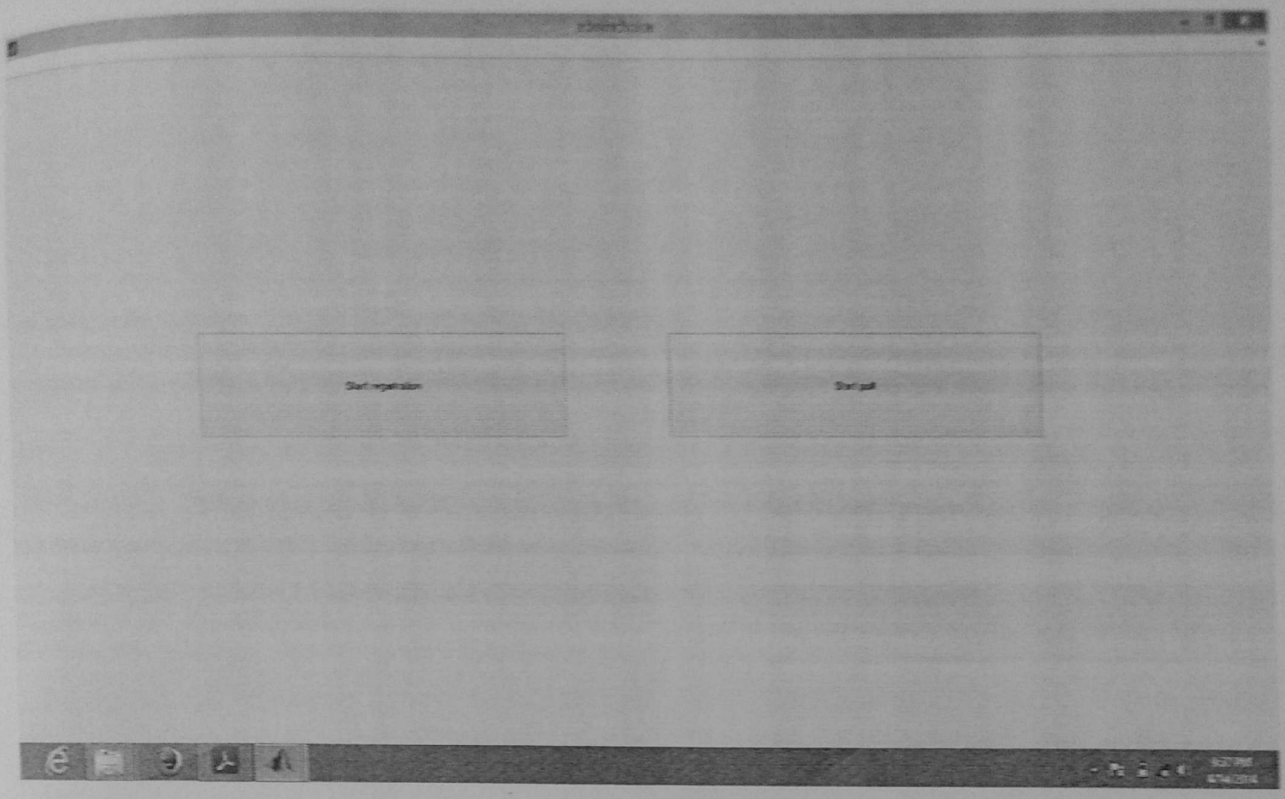


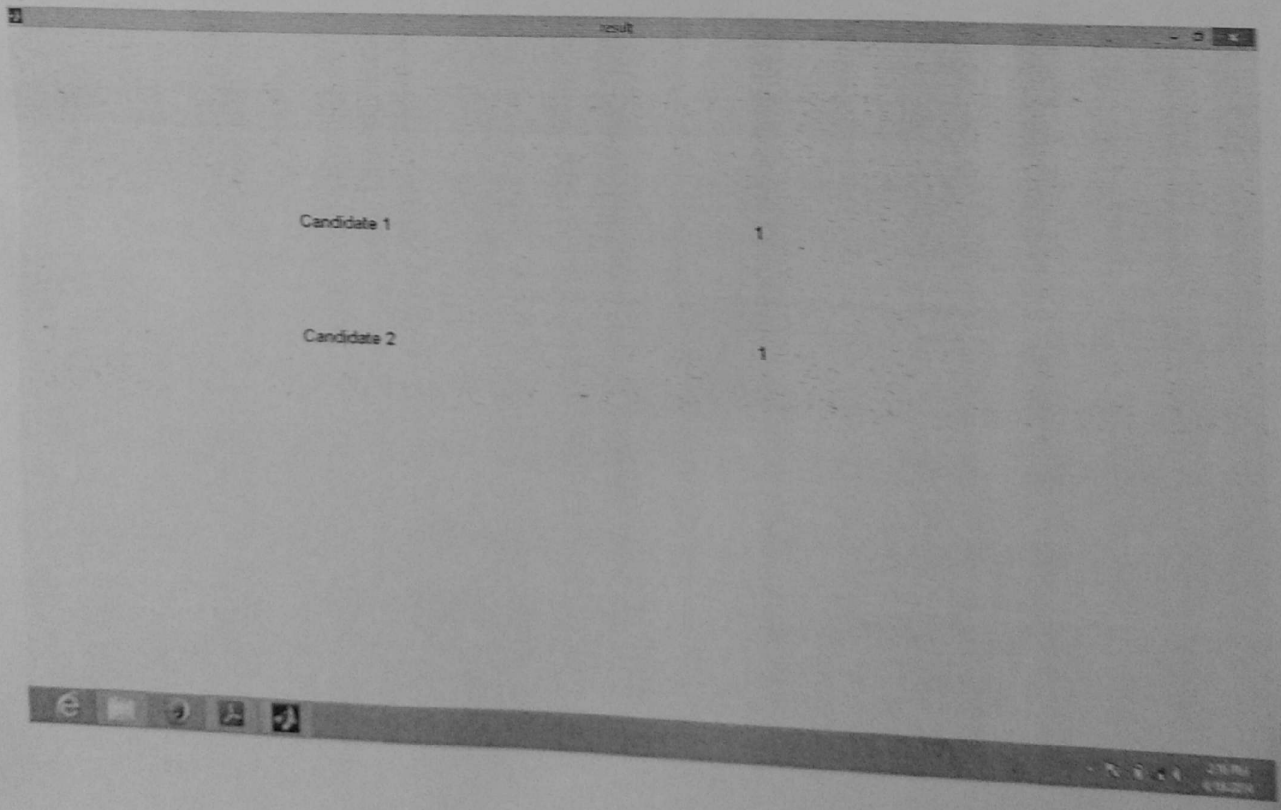

Name
AMAL V
Sex
MALE
Department
ELECTRONICS AND COMMUNICATION
Voter ID
ECM100560
Secret Key
8332


voterlogin
Voter ID
ECM100560
Secret Key
8332
Choose fingerprint
End poll
Login


Candidate 1
Candidate 2
Submit


Admin ID
Secret Key




result
Candidate 1
1
Candidate 2
1