Secure Electronic Voting System

Yichen Zhou, zhouyich@usc.edu, 2506069725

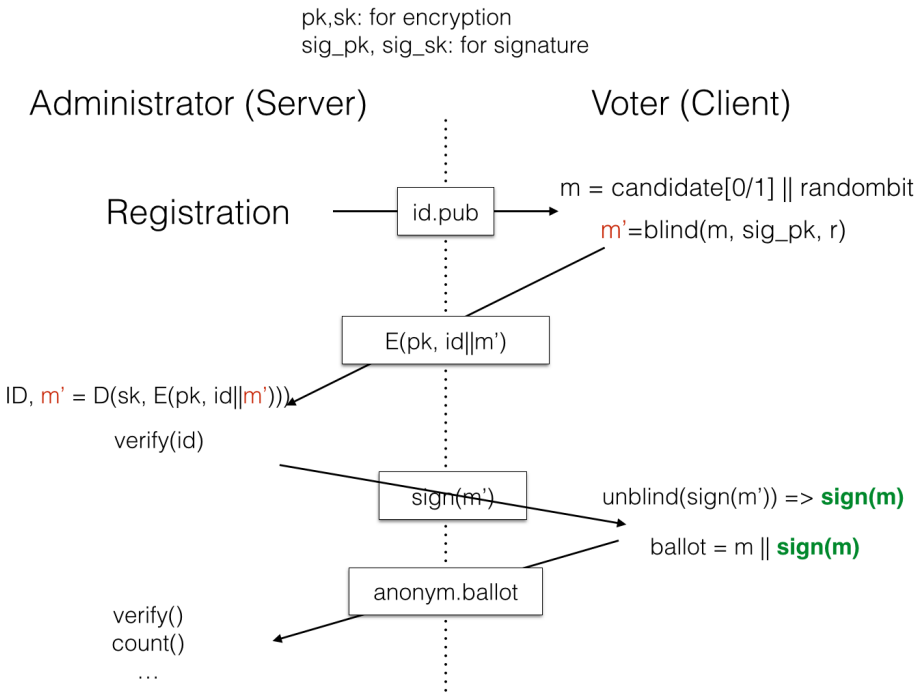
04/29/2020

This project is a simplified E-voting system, which implement cryptographic techniques to secure voters' privacy integrity, and confidentiality. This project is written in python and it reuses the code from PA 1 and PA 2.

Github: https://github.com/ZhouYC627/E-Voting.git

1. System Architecture

The system consist of two main entities: administrator(Server) and the voter(Client). The administrator is responsible in giving out registration, assigning identification, signing ballot (authentication), collecting ballot, tallying votes and verifying votes. While the client is used by voter to encrypt their votes and also help to keep them anonymous. Practically, the server and the client are supposed to running on different machine. The server should be running on an actual server node, while client should be running on voters' personal computer or mobile device. However, since our class focuses on cryptography rather than client-server architecture or sockets programming, this system is simplified and can only be running on the same node(But still needs two terminals). Instead of transferring socket through the network, the server and the client read and write file from the local disk. The protocol shows below. More details about how to executed the program and the format of the input/output will be covered later.



1.1 Election Stages

There are six stages in E-Voting. They are Voter Registration, Blinding and Unblinding ballots, Signing, Voting, Counting and Verifying. The following sections will describe the protocols and events in each of the six stages.

1.1.1 Voter Registration

Registration is the first step to disallow unregistered people to participate in the vote. During registration, people who have right to vote can register to the server and get a unique identification with a public key file. The administrator will first check if this person is eligible to vote. In real world scenario, the administrator is probably going to check the ID or passport of the voter, and see the picture in the ID, or even finger print. In this project, typing the string ‘y’ represent those process. Once a person is registered, two public key pairs is generated by RSA algorithm. One of the public key pair is to encrypt the communication between client and server, like what we did in PA 2. The private is stored in server’s database and the public is written to local file protected by password, i.e.“1000.pub”, which likely be saved in voter’s own disk in real case. So is the second RSA pair of public key. The only difference is the first key pair is used for encryption/decryption while the second pair of key is for blinding/signing. People who are not allow to vote will not get registered.

1.1.2 Blinding and Unblinding ballots

In this stage, the voter wants the administrator to sign the message m blindly. The voter already got a public key (n, e) after registration, and the administrator keeps a secret key d. The voter concat the number of candidate they want to vote(2 bytes) with a 14-byte random byte string. This 16-byte string is called m. The client blind message m to m’, then encrypt it with AES128 which the shared key is exchanged by the first public key pair for encryption/decryption. The encryption/ decryption function was implemented for PA 2 in the file “myCrypt.py”. Finally, the client save the E(m’) as well as voter’s Id to local file “xx.bm.cip”. Here, “bm” stands for blinded message and “cip” stand for ciphertext. The server will later decrypt the file and sign m’.

After server signed the m’, the client will use the random number r to unwrap it and finally get the signature of m.

1.1.3 Signing

In the signing stage, the server will decrypt the file “xx.bm.cip” and get the m’ and voter’s id. Before signing, a verification should be done first. To make sure that this ciphertext was sent by a registered voter, the server will compare the decrypted id in that file with the voter’s Id, which the decryption secret key belongs to. If so, the server will sign the message and saved it to local file.

1.1.4 Voting

After the server signing and unblinding, the client now has the message m and the signature of m. The client then combines these two and saves it to “anonym.ballot”. This file is the actual signed ballot and will be sent to the server. The server will then verify the signature with its private key(for signature) and then store the message m in its own database. To make the votes and the result immutable, I implement Merkle Tree(in PA1) to store the hash value of each vote. Before the vote is tallied, the server will check the inclusion of the message m in the merkle tree to see it there is a repeated votes. If so, that vote will be rejected and will not be counted twice.

1.1.5 Counting

To provide the integrity of the E-Voting, this system do not store the number of votes of each candidates, which is extremely vulnerable. Instead, it store every single message m and the its merkle tree. By recomputing a new merkle tree from messages and comparing the hash root of those two tree, we can easily proof that the data hasn’t been change. Therefore, every time when it comes to “counting”, the server will do the merkle tree comparing and output the number of votes at the same time. It can also print the entire merkle tree hash to allow every to check the integrity of the data.

1.1.6 Verifying

This system allows every voters to verify if their votes were received and tallied. The voter can use the hash value of their original message and see if that hash value exist in the merkle tree in the server. If the hash value exist, their votes must be already received and tallied.

2. Cryptographic Components

2.1 RSA

The RSA cryptosystem is one of the first public-key cryptosystems, based on the math of the modular exponentiations and the computational difficulty of the RSA problem and the closely related integer factorization problem (IFP). The RSA algorithm is named after the initial letters of its authors (Rivest–Shamir–Adleman) and is widely used in the early ages of computer cryptography.

The RSA algorithm provides: Key-pair generation: generate random private key and corresponding public key; Encryption: encrypt a secret message (integer in the range [0...key_length]) using the public key and decrypt it back using the secret key; Digital signatures: sign messages (using the private key) and verify message signature (using the public key); Key exchange: securely transport a secret key, used for encrypted communication later.

In this voting system, RSA is implemented for the key exchanging and the blinding signature.The python file genkeys.py is to generate RSA public and private keys and write them to file .pub and .prv. The key is int type.

•is_prime(n, k): Test if a number is prime

•generate_prime_number(len): Generate a prime of length len using Miller-Rabin algorithm.

•modular_inverse(a, b) : Modular inverse is computed with Extended Euclidian Algorithm.The prime numbers p and q are not public (although n = pq is). An attacker cannot therefore know φ(n), which is required to derive d from e. The strength of the algorithm rests on the difficulty of factoring n (i.e. of finding p and q, and thence φ(n) and thence d).

The python file myCrypt.py is to encrypt and decrypt data using AES-128 and RSA. RSA public key and private are generated by genkeys.py

•encrypt(n, e, mFile, cFile): taken public key (n, e), message file, and output the ciphertext to the cipher file. It first get n and e from the public key file and then use os.urandom() to generate a random key K for AES-128. Then, use that public key to encrypt the K'=E(pk, K) and write it to the .cip. All the data reading from message file are transferred to bytes and save it to .cip as well.

•decrypt(n, d, cFile, mFile): K' from .cip and decrypt it to K = D(sk, K'). And using that key K and AES cipher to decrypt the ciphertext and write to output file.

2.2 AES

The Advanced Encryption Standard (AES) is a symmetric block cipher chosen by the U.S. government to protect classified information. AES is implemented in software and hardware throughout the world to encrypt sensitive data. It is essential for government computer security, cybersecurity and electronic data protection.

There are 10 rounds for 128-bit keys, 12 rounds for 192-bit keys and 14 rounds for 256-bit keys. A round consists of several processing steps that include substitution, transposition and mixing of the input plaintext to transform it into the final output of ciphertext.

In this system, AES-128(ECB) mode is used to encrypt the communication between the server and the client. The shared secret key is exchanged by RSA. The implementation of AES encryption and decryption is in the file "myCrypt.py".

2.3 Blind Signature

Blind signature is a kind of digital signature in which the message is blinded before it is signed. Therefore, the signer will not learn the message content. Then the signed message will be unblinded. At this moment, it is similar to a normal digital signature, and it can be publicly checked against the original message. Blind signature can be implemented using a number of public-key encryption schemes. In this project, I choose the blind signature based on RSA encryption.

This algorithm is computed by raising the message m to the secret exponent d modulo the public modulus N. The blind version uses a random value r, such that r is relatively prime to N (i.e. gcd(r, N) = 1). r is raised to the public exponent e modulo

N, and the resulting value r^e mod N is used as a blinding factor. The author of the message computes the product of the message and blinding factor, i.e.:

m' = mr^e (mod N)

and sends the resulting value m' to the signing authority. Because r is a random value and the mapping *r -> r^e mod N* is a permutation it follows that *r^e mod N* is random too. This implies that m' does not leak any information about m. The signing authority then calculates the blinded signature s' as:

s' = (m')^d (mod N)

s' is sent back to the author of the message, who can then remove the blinding factor to reveal s, the valid RSA signature of m:

s = s'r^-1 (mod N)

This works because RSA keys satisfy the equation r^ed = r (mod N) and thus

*s = s'*r^-1 = (m')^d*r^-1 = m^d*r^ed*r^-1 = m^d*r*r^-1 = m^d (mod N)*

hence s is indeed the signature of m.

Blind signature is implemented in the third stage of the system architecture to make sure no one should know how a particular person voted. The key of Blind Signature is that the voter is using message m to vote, however the administrator can only see m' when signing the message. In signing stage, the administrator has to know the identity of the voter to verify(sign), but it can only see m', which contains no information of whom the voter votes. In voting stage, the administrator know the content of message, but it doesn't know whom that message belongs to. As the result, the server or the administrator or anyone cannot associate the vote with the voter.

2.4 Merkle Tree

A Merkle tree is a tree in which every leaf node is labelled with the cryptographic hash of a data block, and every non-leaf node is labelled with the cryptographic hash in the labels of its child nodes. In this project, I'm using SHA-256 as the hashing algorithm to implement the Merkle Tree. The implementation is in the python file "buildtree.py".

MerkleHashTree: class of Merkle tree. Use a python dict to store tree node and it's hash value.

rootHash(n): return the hash value of the root of D(n) mth(k1, k2): For leaf node mth(i,j): the leaf hash for data entry di with 0<=i<n is mth(i,i+1); for non-leaf node mth(k1,k2) = hash(mth(k1,k1+k) | mth(k1+k,k2)), where k = k1+lp2(k2-k1) , and lp2 is the largest power of 2 < (k2-k1)

audit paths: A Merkle audit path for a leaf in a Merkle Hash Tree is the shortest list of additional nodes in the Merkle Tree required to compute the Merkle Tree Hash for that tree.

verification algorithm: RFC 6962

3. Secure Requirement

Without these security requirements, numerous opportunities for a widespread fraud and corruption may exist. The following section will describe how the four security goals are achieved.

3.1 Privacy

As discussed in 2.3, Blind signature is implemented in the third stage of the system architecture to make sure tallying computation have no chance to reveal any information about the individual votes. In the meantime, everyone including the voters themselves can proof the accuracy of tallying. The key of Blind Signature is that the voter is using message m to vote, however the administrator can only see m' when signing the message. In signing stage, the administrator has to know the identity of the voter to sign, but it can only see m', which contains no information of whom the voter votes. In voting stage, the administrator know the content of message, but it doesn't know whom that message belongs to. Since no one can associate the vote with the voter, the privacy is compromised.

3.2 Eligibility

Only registered user has public key "xx.pub" and its private key is stored in server's database. Without the key, the client cannot send message to server, and the client cannot blind/unblind the message neither. In addition, the server will verify the ID of voter before signing the message m', which prevent Eve from using Alice's ID to get the signature and other forgeries.

The server verifies the ballot with its private key. If the server received a ballot without signature or with wrong signature, it will reject that ballot and that vote will not be count in as the total number.

3.3 Verifiability

The merkle hash tree of server is open to public. Everyone who is or is not registered voter can access and examine all the hash value. Thus, every can using SHA-256 to verify if any message stored in server's database has been changed. And every time when server doing the "tallying" or "counting", it will check the integrity of data and output "Good!" if passing the check.

Voters can check if their message are included in the databases of server and if the hash values of message exist in the merkle hash tree. If so, their votes are definitely received and tallied.

3.4 Immutability

Although there are only a few candidates in the election, every voters' message contains not only the vote but also a 14-bytes(120-bits) of random number generated by client. The length could increase as the number of voters grows. As a result, every voter's vote(message) is different. So the hash values of these votes are different as well. Once that message is signed, it is not able to change. Neither can Eve forge a new signed vote from any previous votes and submit then, because the original messages are all different.

If the server received a ballot which has the same hash value exist in the merkle hash tree, it will reject it and output “Repeated Vote!” Any vote will not be count twice.

4. Assumptions and Limitations

This system is based on the assumption that the key which stores at local disk of server and the client are both secure. All attackers who stand between administrator and voter can be prevented. Also, if Eve hacked into server’s device and modified the data, the administrator or anyone else can detected the changes(integrity check), as long as Eve doesn’t have the key.

Another assumption about privacy is that nobody has the ability to trace where the vote comes from, or voters can votes from a public computer(i.e. send the ballot from a public computer). If Eve knows who send that message, she can do a tally before the voter votes and do it again after the voter votes. Then Eve can figure out which candidate the voter votes based on the changes of the number of votes. To solve that issue, voters can use anonymous network like Onion Routing.

5. How to Run

Just simply run *pollServer.py* and *voterClient.py* on two terminals and follow the instruction printed. You have to input operation code to execute different kind of operation in server, like 0 for registration and 1 for signing.

1) register on server
2) blind vote on client
3) sign blinded message on server
4) unblinded message on client(ballot will be generated at this time)
5) submit ballot file on server
6) count/verify on server **at any time**

Make sure they are in the same directory because they are exchange values through reading/writing local files. Details are shown below.

6. Result and Screenshots

1. Launch server, input 0 to register, public key “1000.pub” will be generated.

```
➜ Project git:(master) ✗ ./pollServer.py
Please input Operation code:
 0: register;
 1: sign;
 2: submit ballot;
 3: count;
 4: verify vote;
 9: exit

0
Are you eligible? (y/n) :y
Registed!
Public Key:1000.pub
```

2. Switch to client, input the ID of voter and indicate which candidate you want to vote for. Then the encrypted blinded message will be generated.

```
➜ Project git:(master) ✗ python3.7 voterClient.py
Input Id to load public key: 1000
Vote for candidate: 0
20863825022233929852547396951356479897 6
Blind messsage: 1000.bm encrypted by public key
```

3. Back to server, type ‘1’ and then input ID of voter. The server will read 1000.bm, verify the ID, sign the m’ and then write to 1000.sign.

```
Please input Operation code:
 0: register;
 1: sign;
 2: submit ballot;
 3: count;
 4: verify vote;
 9: exit

1
Voter Id: 1000
Reading blind message from: 1000.bm
Registered voter
Signed to:1000.sign
```

4. Switch to client, input the name of the sign file, which is “1000.sign”. Ballot file will be automatically generated.

```
Signed filename:1000.sign
Unblind the signed ballot, output to file: anonym.ballot
You can submit it now.
```

5. On server, type ‘2’ to submit the ballot. It will valid the ballot and notify if the ballot is received.

```
Please input Operation code:
 0: register;
 1: sign;
 2: submit ballot;
 3: count;
 4: verify vote;
 9: exit

2
Ballot filename: anonym.ballot
20863825022233929852547396951356479897 6
valid ballot
Ballot received.
```

6. Feel free to type ‘3’ to count all votes and type ‘4’ to check if your vote has been tallied.

As shown in the screenshot, it will first print all the hash values in merkle tree for everyone to examine. In red box of this screen shot, {0: 2, 1: 1} stands for 2 votes for candidate #0 and 1 votes for candidate #1. Voters will be notified if their votes are tallied or not.

If the ballot is not signed by administrator, or has been modified, it outputs “invalid ballot”. However, if you submit the valid vote from same voter twice, it will first show that this is a valid ballot but then reject it because it is repeated.

References:

https://zhiqiang.org/cs/secure-voting-by-blind-signature.html
https://medium.com/coinmonks/implementing-an-e-voting-protocol-with-blind-signatures-on-ethereum-411e88af044a
https://www.sciencedirect.com/topics/computer-science/blind-signature
Subariah Ibrahim, et al. Secure E-Voting With Blind Signature.
Yi Liu and Qi Wang, An E-voting Protocol Based on Blockchain.