Digital Voter List System

Authors:

Emil Blædel Nygaard <ebln@itu.dk>,
Michael Oliver Urhøj Mortensen <miur@itu.dk>,
Rasmus Greve <ragr@itu.dk>

Abstract

The Digital Voter List System is an attempt at creating a (free) computer system to completely replace the current (and expensive) voting system used at the elections in Denmark. The system will be almost entirely automated, the exception being user input when scanning (or entering manually) the voting number of a person's ballot. The idea is to have three or more computers connected to a local network, and have these computers share encrypted information about each new registered vote. This information will be stored in a local database.

Running the program

To run the program, locate the folder named "Executable" on the CD and copy it to your desktop (The system edits a database file, and therefore cannot be run from the CD).
Open the file named "DVLTerminal.exe"

For testing purposes, this is the current data in the database:

```
CPR             Name,                       VotingNum
0912917853      "Anders Andersen"           7853785300
1003875433      "Bendt Bendtsen"            5433543300
1111764353      "Charlie Comwell"           4353435300
0102821142      "Dorte Ditlevsen"           1142114200
1212913847      "Espen Erlingsen"           3847384700
2412424242      "Chuck Norris"              4242424242
0101852197      "Good Guy Greg"             2197219700
0709799975      "Chuck Chester"             9975997500
1412661234      "Helle Thorning-Schmidt"    1234123400
1109747947      "Joseph R. Kiniry"          7947794700
3112939999      "Max Value"                 9999999900
0101011111      "Min Value"                 1111111100
```

Requirements

Mandatory Requirements

The system must

- be able to register that a person has received a ballot.
- never allow a person to receive a ballot if they have already gotten one.
- be able to run on multiple machines communicating over network.
- have a feature that ensures that the voter is using the right voter card.
- shut down gracefully when there are less than 3 computers present.
- be able to show the attendance percentage.
- be able to end the election and export a list of voters.
- encrypt network messages.
- be robust enough to withstand gibberish- and repeat-network-attacks.
- be easy to use for voulenteers at the voting place without training.
- be easy to set up for leaders at the voting place with minimal training.
- have an English Graphical User Interface.
- be Open-Source
- not use Security-by-Obscurity.

Secondary Requirements

The system may

- use an encrypted database.
- may be robust enough to withstand DoS attacks.

Fulfillment

We have fulfilled all our mandatory requirements, to a certain degree. In particular our systems allows the user to register that a person has received a ballot if the voter is using the right voting card. The system runs on multiple machines on a network, are communicating with encrypted messages and is able to withstand gibberish- and repeat-network-attacks. It shuts down gracefully if there are not enough peers connected, and overall it has an easy to use English user interface. All resources used in the system are either public domain or completely royalty free. Also we don't use security by obscurity and there should be no problem in making the system open-source.

The system does its best to ensure that a person can only receive a ballot once. We have discovered a case where a person could possibly be allowed to vote twice if our system is the only security measure. If a computer crashes right after the barcode has been scanned our protocol is to hand out the votercard. We choose to do this as there is no way to tell which other computers have received the right information. We have thought about using "Election Ink" which is a type of ink that is applied to a finger of the voter when they receive a ballot. This ink leaves a semi-permanent mark which cannot be washed off and will disappear over time, ensuring that a person can only vote once. This is rather invasive, which is why another solution might be better.

We don’t actively fulfill any secondary requirement, but might be able to withstand a DoS attack.

Overview

The system we have decided to implement is a part of a bigger system which could be used to run an election.
An election has multiple periods. At first, a system must generate a list of all eligible voters for a given election. These voters will be spread over different voting places and isolated lists of voters will be generated, one for each voting place.

Our system is the part running during the election day, making sure that all eligible voters can get one and only one voting ballot, and thereby only vote once.
Setup of the system requires that the list of voters designated for the given voting place is distributed amongst computer intended to be used at this voting place. This process happens along with the installation of our program and is assumed to have taken place when the program is launched.

The main feature of the system is that it maintains a database of eligible voters on each connected PC, and makes sure that they always are equal in the sense that if a person is registered as “Has Voted”, then she will be on all machines.
The main entry point for data takes two parameters. The voter ID which could be scanned in from a barcode and the check digits from the persons social security number as a confirmation.
When a voter registers a series of steps is taken:
Check that he has not already gotten a voting ballot, using the local database.
Ask all connected computers if the voter, has already voted. This step also distributes that the voter has voted.
If all computers answer that she hadn’t voted already, allow the user to hand out a ballot.

This would be considered normal operation and is the steps that is taken most of the time. To make sure that the databases are always in sync we consider two factors who can impact this.
A network packet is dropped and never reaches it’s destination.
A computer dies, either by OS crash, hardware failure or other circumstances rendering the computer unable to receive and send network packages.

A dropped packet will be discovered by the system and will be resent, and if a computer dies it will be noted and the computer will be taken out of operation.

To establish a connection between the computers , we go through a startup process where all network data is unencrypted and the computers are gathering IP addresses of each other.

When all computers are started and the program is running, computers can send discovery packets which all must respond to. When all computers have discovered all other computers, one of them initiate a key exchange process. This process broadcasts a key packet which should be received by everyone else. If one of the computers does not receive the key packet, the discovery process must be restarted.

After every machine has received keys, an operator must start the normal operation process of the program. This will make it use the encryption keys on all network packages, both incoming and outgoing.

Architecture decisions

We use an event simulation system to simulate multiple threads. We strove to use a Model-View-Control architecture which gives loose coupling and a structured division of classes.

Dictionary

Peer

A computer connected to the local network running the program.

EventSystem

The event simulation system running in all instances of the program, allowing multiple things to happen simultaneously independent of each other

Vote

When saying that a person has voted it means that they have received a voting ballot. A vote in the system contains information about the voter and provides methods for distributing the vote to the connected peers.

NetworkPacketHandler

There is multiple network packet handlers in the system. A network packet handler can receive a packet from the network and handle it properly. Furthermore it can time out after a given amount of time making it stop listening for network packets and performing a predefined action.

Database
A file of a specific type, containing all the information needed for the system, stored in tables that consists of rows and columns.

Message
A structured amount of data containing information about the sender and reciever of the message as well as the actual message.

Example
The user launches the program through the executable file. This file, along with the database file needed for the system, is provided and pre-configured by us. The first window that is displayed is a confirmation window, that will help the user ensure that all the computers that should be connected to the network, are connected and visible to the computer. If someone is missing on the list, the user needs to press “Connect”. When everyone are on the list, one person clicks the “Next” button, and a confirmation window will appear on every computer. If it did not, all the users needs to click “No, try again” and redo the previously described process. When everyone can see the window on their screen, everyone must click “Yes, it is shown on all PC’s” and the election can begin.

When the election can begin, the main window will be visible on the sreen. This window has two text fields: one for the voting number and one for the last four digits in the CPR number, all provided by the person wanting to vote. Both numbers must be entered. If they are not, or if there is an error, such as a non-digit character, a message box will inform the user of the error type, and reason.

Assuming that both numbers are entered correctly, the user will be greeted with a window showing that it is OK to hand out the ballot. If, however, there was an error, such as the voter having voted once already, a dialog box will inform the user about this. This happens if the voting number cannot be found, too, as well as if the CPR number provided does not match.

This work flow will continue as long as the system does not crash, or until the user presses the “End election” button. This button will launch a confirmation window, where the user is warned about the risk of exiting the application. Should the user however desire to do so, an automatically generated security code must be entered in the text field and click the confirmation button.

System validation
We have not been able to use unit tests on the core elements of our program, because these elements depend on network communication as well as the number of peers on the network and other tricky cases. All the testing has, in other words, been done manually by us.

Among scenarios we have been testing are:

- Making sure the election ends if there are less than 3 clients online.
- Making sure that our system handles package loss - this has been observed during testing in general, and has not been tested specifically.
- If your client is alone, you shall not be able to continue.
- If there is inconsistency in the databases, the clients with the missing updates cannot continue.
- Report that a 4th peer is dead, so the computers on the network wont wait for the reply from the 4th peer.
- Inform the 4th peer that said peer is dead, should the peer reappear on the network and try to submit a vote.
- Making sure that all peers receive the message that a person has voted.
- Making sure that all peers has got the same security key distributed by the person clicking “Next” in the Connection window.
- Making sure that the graphical user interface works in general, and that all components are displayed when expected.

Known issues

Unexplained database issues

There is, for whatever unexplained reason, an issue with the SQLite database and/or it’s API for communicating with the database, that causes a crash in rare occasions. We have no way of controling this crash, however. A fix we implemented is, besides being very ugly and risky, catching any exception thrown by the database methods and recalling the same methods, with the same parameters. This ensures that the databse most likely will work 2nd or 3rd time.
This however also means that if the database file cannot be found, the program will get stuck in an endless loop.

Split-network

We have discussed a case that just might happen in which the system could become inconsistent. If the network for any reason splits itself so that computer 1-3 can communicate and computer 4-6 can communicate, the program will continue executing without discovering what went wrong. Computer 1-3 will all conclude that 4-6 are dead and vice-verse. Then every voter assigned to this voting place could in theory be able to vote twice; first on 1/2/3 then on 4/5/6. We have chosen not to handle this error as the chance of it happening continuously is very very small. Though there is a way to ensure it never happens by only allowing two (MinimumAllowedNum - 1) computers to be registered dead at the same time. This will ensure that if the network splits, one or the other section of computers will conclude that they are dead and terminate.

Must follow setup process
If the process, as described in the graphical user interface throughout the program, isn't thoroughly followed, we cannot guarantee that the program will work as anticipated.

Bibliography

The MSDN page on AES (the AesManaged class)
http://msdn.microsoft.com/en-us/library/system.security.cryptography.aesmanaged.aspx

SQLite (The database):
http://www.sqlite.org/

ADO.NET 2.0 Provider for SQLite (The API for communicating with the database):
http://sourceforge.net/projects/sqlite-dotnet2/

Revision History and Subversion repository

Our revision history can be found at: http://www.emilnygaard.com/dvlt/timeline
The SVN repository can be accessed at: http://www.emilnygaard.com/dvltsystem
To check out repository:
Username: bdsaexaminator
Password: 4U7Exa

Some milestones from our revision history:
04:43 Changeset [14] by rasmus
Implemented EventSystemExecutable. Should be done

02:14 Changeset [37] by michael
All methods implemented. Doc. missing (Database)

00:06 Changeset [69] by rasmus
SecurityManager works

06:59 Changeset [103] by rasmus
PingRequestResponseListener implemented. VoteResponseListener implemented ...

00:37 Changeset [113] by rasmus
SecurityManager works again.

02:03 Changeset [229] by rasmus
Working program!

BON Specification - Informal

```
system_chart DVLSystem
indexing
	author: "Emil Blædel Nygaard (ebln@itu.dk), Michael Oliver Urhøj Mortensen
(miur@itu.dk), Rasmus Greve (ragr@itu.dk)";
	course: "BDSA";
	date: "2011/11/28";
explanation
	"The system running on all voting places"
cluster DVLTerminal description "The system running on the “scanning”-terminals"
cluster LIB description "Libraries provided in c sharp and dot net"
end

cluster_chart LIB
explanation "Libraries provided in c sharp and dot net"

class FORM description "The Form class from the libraries in C#"
end

cluster_chart DVLTerminal
explanation "The system running on all the 'scanning'-terminals"

class EVENTSYSTEMEXECUTABLE description "An executable item in the event system"
class EVENTSYSTEM description "A discrete event simulations system for doing tasks
simultaniously"
class NOTENOUGHPEERSEXCEPTION description "An exception to throw and catch if the
system ever goes below a specified number of computers in total"
class PEERSTATE description "Defines the different states that this computer can know
a peer to be in"
class PROGRAM description "The main program"
cluster GUI description "The graphical user interface"
cluster LOCAL description "Everything that is handled locally on the machine"
cluster NETWORKING description "All the classes that have to do with the network
traffic"
cluster UTILITIES description "General utility classes for use all over"
end

cluster_chart GUI
explanation "Graphical User Interface components"
class ACCEPTBALLOTHANDOUT description "A window to show that a ballot can be handed
out"
class CONFIRMATIONWINDOW description "A window to confirm that all the computers have
received the encryption keys"
class CONNECTIONWINDOW description "A window to check that all computers in the
network can see eachother"
class DECLINEBALLOTHANDOUT description "A window to show why a ballot should not be
handed out"
class ENDELECTIONWINDOW description "The window to confirm that you want to exit the
program and end the election"
class PLEASEWAIT description "Window shown while the system is gathering information
from the other computers"
class REGISTERVOTERWINDOW description "The main window where the voternumber and CPR
number is entered for validation"
end
```

```
cluster_chart LOCAL
explanation "Everything that is handled locally on the machine"
class LOCALVOTELISTENER description "Handles new votes on the local machine"
class VOTE description "Describes a vote from the local machine"
class VOTESTATE description "Enumeration describing the different states a Vote can
have"
class VOTERESULT description "Enumeration describing the result of the attempted Vote"
end

cluster_chart NETWORKING
explanation "All the classes that have to do with the network traffic"
class CONNECTOR description "Establishes connections to other peers also using the
connector"
class INBOX description "Receives and decrypts all messages"
class OUTBOX description "Encrypts and broadcasts messages on the network"
class INETWORKPACKETHANDLER description "Interface"
class MESSAGE description "A message to be sent or received over the network"
class MESSAGETYPE description "Defines the different kind of messages that can exist"
class NETWORKLISTENER description "Listens for incomming networkmessages"
class PACKETRESPONSE description "A type enum to indicate whether a handler is able to
handle a given package"
class PEERDEADLISTENER description "Handles incoming PeerDead messages"
class PINGLISTENER description "Listens for Ping messages and answer Pong immediately"
class PONGLISTENER description "Listens for Pong messages and sends answer to the
initiating peer"
class PINGREQUESTRESPONSE description "Defines the different responses that can be
received as answer to a ping request"
class PINGREQUESTRESPONSELISTENER description "Listens for responses to PingRequests"
class PINGREQUESTLISTENER description "Listens for PingRequests and creates
PongLinsteners"
class VOTELISTENER description "Listens for incoming votes, saves to database and
answers OK"
class VOTERESPONSELISTENER description "Listens for responses to a vote from network"
end

cluster_chart UTILITIES
class DATABASE description "API for communicating with the SQLite database."
class SECURITYMANAGER description "Get’s keys from a file and creates the AesManaged
object"
end

class_chart CONNECTOR
explanation "Establishes connections to other peers also using the connector"
inherit INETWORKPACKETHANDLER
command
   "Broadcast a request for other computers",
   "Send out the encryption keys",
end

class_chart INETWORKPACKETHANDLER
explanation "An interface that defines a method for handling networkpackets"
command
       "Handle this packet",
       "You have timed out"
```

```
end

class_chart VOTERESPONSELISTENER
explanation "Listens for VoteOK packets, and registers any incoming ok in the vote.
Terminates."
inherit INETWORKPACKETHANDLER
end

class_chart VOTELISTENER
explanation "Handles incoming votes by saving the info to database and replying
VoteOK"
inherit INETWORKPACKETHANDLER
end

class_chart PINGLISTENER
explanation "Handles ping packets by responding pong back to sender. Persists."
inherit INETWORKPACKETHANDLER
end

class_chart PONGLISTENER
explanation "Handles pong packets by responding PingOK to initiator. Terminates."
inherit INETWORKPACKETHANDLER
end

class_chart PINGREQUESTLISTENER
explanation "Handles pingRequest packets by creating a PONGLISTENER and seding a ping
to the wanted peer. Persists."
inherit INETWORKPACKETHANDLER
end

class_chart INBOX
explanation "Receives and decrypts all messages from the network"
query
       "Can I have the instance of you please",
   "Do you have any new messages?"
command
   "Start listening for new message",
       "Give me the first new message"
constraint
       "You must not ask for the first new message when there are no new messages"
end

class_chart MESSAGE
explanation "A message to be sent or received over the network"
query
       "Who sent this message",
   "Who is the recipient of this message",
   "What kind of message is this?",
   "What is the data array of this message?",
   "What is the string-representation of the message data?",
command
       "This peer sent this message",
   "This message is meant for this peer",
       "This is the data array for this message",
   "This is the string-representation of the message data?",
   "This message is invalid",
   "Create a new message from a string representation",
   "Convert the message to a bytearray packet"
constraint
       "A message must have exactly 1 recipient and exactly 1 sender",
       "A message must have a type"
end

class_chart OUTBOX
explanation "Encrypts and broadcasts messages on the network"
query
       "Give me the only instance of you"
command
       "Encrypt and send this message",
   "Send this message unencrypted"
constraint
       "There can be only one"
end

class_chart MESSAGETYPE
explanation "Defines the different kind of messages that can exist"
query
   "How to represent the type: 'Uknown'",
   "How to represent the type: 'HeartBeatRequest'",
   "How to represent the type: 'Heartbeat'",
   "How to represent the type: 'Vote'",
   "How to represent the type: 'VoteOk'",
   "How to represent the type: 'PingRequest'",
   "How to represent the type: 'Ping'",
   "How to represent the type: 'PingOk'",
   "How to represent the type: 'PingFailed'",
   "How to represent the type: 'Pong'",
   "How to represent the type: 'PeerDead'",
   "How to represent the type: 'EncryptionPacket'",
end

class_chart NETWORKLISTENER
explanation "Listens for incoming network Messages and tries to delegate them to all
registered handlers"
inherit EVENTSYSTEMEXECUTABLE
command
   "Register this handler",
   "Remove this handler"
end

class_chart PACKETRESPONSE
explanation "Defines the different kind of messages that can exist"
query
   "How to represent the response: 'Accept'",
   "How to represent the response: 'Decline'",
   "How to represent the response: 'AcceptAndFinish'"
end

class_chart PEERDEADLISTENER
explanation "Handles incoming PeerDead messages"
```

```
inherit INETWORKPACKETHANDLER
end

class_chart PINGREQUESTRESPONSE
explanation "Defines the different responses that can be received as answer to a ping
request"
query
   "How to represent the Response: 'Yes'",
   "How to represent the Response: 'No'",
   "How to represent the Response: 'NotYet'",
end

class_chart PINGREQUESTRESPONSELISTENER
explanation "Listens for PingRequestResponses"
inherit INETWORKPACKETHANDLER
end

class_chart LOCALVOTELISTENER
explanation "Processes votes received from the local machine"
inherit EVENTSYSTEMEXECUTABLE
query
       "Give me the list of peers",
       "Give me the current vote"
command
       "Give me a new LocalVoteListener from this list of peers",
       "This is now the current vote",
       "This peer is now dead"
end

class_chart VOTE
explanation "Contains information about the voter and the progress of distributing the
vote to other peers"
query
       "What is the result of this vote",
       "What is the state of this vote",
       "Give me a dictionary of peer expected to respond to this vote",
       "What is the voting number of the voter",
       "What is the provided cpr check number of the voter",
       "Give me a new vote for this voter and cpr check number, and expect response
from these peers",
       "Is the vote acknowledged by all expected peers"
command
       "This is now the result of the vote",
       "This is now the state of the vote",
       "This peer is now in this state",
       "This peer replied to the vote",
       "Broadcast the vote to the network"
constraint
       "The vote must never go back to the state ‘initialized’ after being in another
state",
       "The state of the vote must never leave ‘done’ once it has been marked ‘done’"
end

class_chart VOTESTATE
explanation "Describes the current state of a vote"
```

```
query
       "How to represent the state ‘initialized’",
       "How to represent the state ‘message sent’",
       "How to represent the state ‘done’"
end

class_chart VOTERESULT
explanation "Describes if a person was allowed to vote"
query
       "How to represent the result ‘yes - handout ballot’",
       "How to represent the result ‘yes - waiting for network’",
       "How to represent the result ‘no - already voted’",
       "How to represent the result ‘no - wrong cpr check number’",
       "How to represent the result ‘no - voting number not in database’",
       "How to represent the result ‘not yet evaluated’"
end


class_chart DATABASE
explanation "Gives acces to persistent storage of the voter information"
query
       "Get the only instance of the Database",
       "Is this a valid voter number",
       "Does this cpr check number match this voter number",
       "Has the person associated with this voternumber voted",
       "How many people have voted already",
       "How many people are supposed to vote here"
command
       "The person associated with this voternumber has (not) voted"
end


class_chart SECURITYMANAGER
explanation "Gives easy access to encryption of messages"
query
       "Get the only instance of the Database object",
       "Give me a byte array representation of the keys",
       "Give me an encrypted version of this unencrypted byte array",
       "Give me a decrypted version of this encrypted byte array"
command
       "Generate new encryption keys",
       "Set the keys from this byte array representation"
end



class_chart ACCEPTBALLOTHANDOUT
explanation "A window to show that a ballot can be handed out"
inherit FORM
end

class_chart CONFIRMATIONWINDOW
explanation "A window used in the start up phase to assure that all clients are
propperly connected. This window will appear on all clients, if and only if they have
recieved the security key."
```

```
inherit FORM
end

class_chart CONNECTIONWINDOW
explanation "This window will display an overview of all the computers connected to
and found in the network. Has functionality of scanning the network for computers and
go to the next state, displaying a CONFIRMATIONNWINDOW."
inherit FORM
command
       "Create a new CONNECTIONWINDOW with a given CONNECTOR"
end

class_chart DECLINEBALLOTHANDOUT
explanation "A window that displayes error messages related to the fact that a ballot
cannot be handed out. The message will change depending on the type of error."
inherit FORM
end

class_chart ENDELECTIONWINDOW
explanation "A confirmation window that is shown when a person clicks the End election
button in REGISTERVOTEWINDOW that will ensure that the person wanting to end the
election did not miss click. This window has a randomly generated number that the user
needs to retype in a textfield, before the application closes."
inherit FORM
end
class_chart PLEASEWAIT
explanation "A window that is displayed when a person sends a message to the network
that a person has voted. The window will automatically close, as soon as an OK has
been recieved from all clients."
inherit FORM
end

class_chart REGISTERVOTERWINDOW
explanation "The main window of the application. This window contains text fields for
both the voting number and the last four digits in "
inherit FORM
end

class_chart EVENTSYSTEM
explanation "A simulation of multiple threads, using an endless loop to loop through
different events."
command
       "Register a new EVENTSYSTEMEXECUTABLE that will be run in the event loop.",
       "Start the event loop.",
       "Stop the event loop of the entire system."
end


class_chart NOTENOUGHPEERSEXCEPTION
explanation "A message saying that there is not enough peer to continue operating"
query
       "How many peers is this computer connected to",
       "Give me a new not enough peers exception"
end
```

```
class_chart PEERSTATE
explanation "Describes the state of a peer"
query
       "How to represent the result ‘the peer is alive’",
       "How to represent the result ‘the peer might be dead’",
       "How to represent the result ‘the peer is dead’"
end

class_chart EVENTSYSTEMEXECUTABLE
explanation "The base class of executable parts of an event system."
command
       "Start running the executable until the timeout is reached."
end

class_chart FORM
explanation "The Form class from the libraries in C#"
end

class_chart PROGRAM
explanation "The main class of the entire system. This class initializes the program."
command
       "Set up a connection and start the process of registration of voters"
end
```

Demonstration

We have recorded a video of normal use of the program. This video is available on the supplied CD and also on Youtube: http://youtu.be/r3aTtGaBp1U