# How not to build an evoting system

Quentin Kaiser

contact@quentinkaiser.be

### Abstract

*Back in 1994, Belgium was one of the first european country to push for the deployment of electronic voting systems. Thought at the time as a sign of Belgium stepping foot in the 21st century, the system stayed in use up to the latest european elections that took place in May 2014. As years passed, bugs got discovered, issues were raised, and public concern grew up to the point where the government was obliged by law to publish the source code of those systems in 2001[8]. We jumped on the opportunity to audit the code in June 2014, looking at the internals and seeing for ourselves what was really going on. By auditing the source code provided by the Ministry of Home Affairs, we found multiple vulnerabilities in the system that could easily be exploited by an attacker to tamper with the election process.*

## I. Introduction

It's been more than 20 years since the first electronic voting system boot up in the small city of Flemalle, Belgium. During all those years, the only external audits came from Affront[1] and the PourEVA association[9, 7, 10]. This can be explained by the tremendous amount of code that composes this system, along with the fact that specialiazed hardware on which this code runs is not available to the public.

When one of the most important part of democratic elections is transparency, how does one achieve this without access to the source code ? Even more, how does one achieve this when missing the necessary technical skills ? In those cases, the citizen is left to trust blindly the third parties that his government task to audit the code that runs the elections.

Compared to previous audits that focused only on static code analysis, we decided to take a new approach that consider the link between all systems and see how small errors in those different systems could lead to serious vulnerabilities, when linked together.

Our analysis start at the polling stations by looking at the software running on voting booth and ballot box machines. We then follow the votes by going through the PGM programs, finally concluding with the Election Management System.

**Summary of Contributions** First of all, our analysis helped to cast a light on the debate about electronic voting. By helping the investigation of Medor journalists[5], we were able to restart the public debate about electronic vote in Belgium which led to the decision of the Walloon Parliament to remove their support of electronic voting systems[4].

Second, by discovering and responsibly disclosing vulnerabilities found in the web applications developed by Stesud, we helped secure their own system and brought a reflexion about information security within the ranks of that company.

Finally, the process by which we discovered those vulnerabilities helped us to craft a methodology that can be re-used to audits electronic voting systems. A formal document will be issued with details regarding this methodology.

**Related Work** We are not the first in this field. Affront did an initial review of the code in 2004[1] that was already pointing out issues regarding in-memory storage of votes and vote anonymity. The PourEVA association also took a shot at it by documenting the code DNA[9]. Following the 25th of May bug - aka 2505bug - Philippe Teuwen released a technical description of the code that led to the bug, along with a Linux port of the code so people could test it for themselves[10].

**Preliminaries** For the sake of our readers and to help clarify the situation, we will use the following nomenclature to designate software that we analyzed:

- Digivote: the electronic voting systems running on the polling stations
- MAV: voting booth software (v.9.16)
- Urn: ballot box software (v.9.15)
- PGM2: windows executable used by polling station presidents (v.275)
- PGM3: windows executable used by polling station presidents (v.1.66.b)
- CODI: the electronic voting systems running on government servers

We'll also use the term EMS - Election Management System - to designate the network infrastructure and web applications used by the Ministry of Home Affairs to prepare and execute the elections.

Curious readers can download all source codes on the belgian governement website at http://www.elections.fgov.be/index.php?id=3285&L=0

## II. Polling Station

Each voter receive a magnetic card[1] that has been initialized by an assessor with the help of the machine running the Urn program. Once the voter received her card, she can go to the voting booth where the machine running the Mav program allows her to cast her vote on the magnetic card. She then get back to the machine running the Urn program that acts as a ballot box by reading the card and storing the vote locally.

At the end of the election day, polling station presidents fetch all the floppy disks storing the votes and get the votes tally of their station with the help of the PGM2. The tally is then sent securely to the Ministry of Home Affairs with the PGM3 software. The results are received and processed by one of the CODI web applications running on the EMS.

### Magnetic Cards

As noted earlier, the Digivote system is based on magnetic cards so it seemed natural to us to first tackle this aspect of the system. We first introduce the magnetic card and the way data are stored on it then we go on demonstrating the different types of attacks along with the errors that make those attacks feasible.

**Card Layout** The magnetic card layout, as described in figure 1, is composed of 5 different sections:

- Token [5 bytes] The token is a string of 5 characters intended to uniquely identify the polling station where the card was initalized.
- P [1 byte] A binary value indicating whether a vote has already been casted on the card.
- HMAC [4 byte] The HMAC of the vote generated with ISO-9797-1.
- Test [1 byte] A single character indicating the voter type ('N' for belgian, 'E' for european and 'S' for foreigners).

[1] http://en.wikipedia.org/wiki/Magnetic_stripe_card

- Vote [2 + x bytes] Two characters giving the political party's list number for which voting was conducted and a variable number of characters corresponding to the hexadecimal representation of the table containing preference votes for the selected list (e.g. [1,0,1,1,0,0,1,1] become A2).

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td></tr>
<tr><td colspan="5">token</td><td>P</td><td colspan="4">HMAC</td><td>test</td><td colspan="2">list</td><td colspan="7">vote</td></tr>
</table>

**Figure 1:** *Magnetic card layout*

**Reading the card content** The data being stored unencrypted on the magnetic card, it is very easy to read the vote stored on it. There is two different ways on how to consider this, you can consider it as a normal condition regarding what the paper vote provide (by considering your vote written on paper as plaintext) or you consider that you need to respect the cryptographically secure voting conditions[3, 2, 6]. In the latter, being able to read the vote would be considered as an issue in itself.

**Modifying the card content** By inspecting the data stored on the magnetic card, changing specific values can lead to users cheating the system by executing the following modifications on the card:

- arbitrary modification of the token value, the HMAC or the vote will invalidate a card
- modifying the test byte with a value other than N, E or S will invalidate a card
- changing the test byte to a valid value (N, E or S) would elevate or drop an elector privileges (e.g. changing from S to N would allow a european citizen residing outside Belgium to vote for belgian federal elections).

## I. Voting Booth

**Authentication and Authorization** Voting booth machines are protected against unauthorized access with a set of security precautions. First, floppy disks and presidents passwords are generated in a central and secure place. Presidents receive by mail the password as a scratch ticket a few days before the election date. Floppy disks are in a sealed envelope which awaits at the polling station. The president wait until the office is constituted (assessors and witnesses) before opening the envelope. There he starts and initializes the voting booth and ballot box machines. Floppy disks (in duplicate) are not write-protected and goes from machine to machine.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr>
<tr><td colspan="8">value</td><td colspan="2">extension</td></tr>
</table>

$$extension = 99 - (value \bmod 97)$$

**Figure 2:** *Password checksum*

[2]

[2] If you feel some kind of déjà-vu, this is because this checksum formula is nearly identical to the one used for belgian national identification numbers.

The president's password has 10 digits. It is not compared to a predefined value, but the program checks the validity of it by calculating a checksum as detailed in figure 2. This password validation system is a priori not problematic, because the value of the password is then used to derive a secret key. If the right key is not obtained, the program will not run.

**Integrity Protection** The Digivote system verifies the data integrity of files stored on both initialization floppy disks and vote storage floppy disks. The data integrity check is based on a comparison between a HMAC value stored on the floppy disk and an HMAC value of files stored on the floppy disk. Only files that don't starts by "BE" and that are different than "TABLE.URN", "FE_DSK.BRK", "FE_DSK", "FE_DSK.CRP", "CTRLVOTE" and "LOG.ERR" are taken into account during the HMAC computation. The HMAC is computed on a string that is composed of HMAC of multiple 1024 bytes chunks read from each file, appended to each other. The key used to compute the HMAC is stored on the floppy disk.

The code snippet in figure 3 shows how this integrity protection was flawed until they deployed a fix for the 2014 elections.

```
#ifdef EL2014
    for (i = 0; i < macResultLen;i++)
      if(macResult[i] != wrkspc[i+16])
        return(0);
    return(1);
#else
    for (i = 0; i < macResultLen;i++)
      if(macResult[i] != wrkspc[i+16])
        return(0);
      else
        return(1);
    return(0);
#endif
```

**Figure 3:** *Integrity protection code*

## II. Ballot Box

Ballot box machines are running the Urn program and possess two distinct purposes: magnetic card initialization, and magnetic card reading.

**Authentication and Authorization** Ballot box software share the same authentication and authorization code than voting booth machines described in I.

**Integrity Protection** Ballot box software share the same integrity protection code than voting booth machines described in I.

**Fraud Detection** Although it is easy to read the vote content off a card or invalidate it by an arbitrary modification of its content, the ability to forge valid cards, however, seems compromised in view of the validation performed by the program running on ballot boxes.

This validation is based on three elements: the number of bytes contained on the card, a token value, and an HMAC. During our analysis, we discovered that the values used to generate the token and the secret key used to compute the HMAC can be retrieved, leading to successful exploitation of the system without being detected by the fraud detection.

**Rogue Magnetic Card** For an attacker to forge a magnetic card, it is necessary to know the number of bytes contained in a valid card, the value of the token but also how the MAC value is computed. Obtaining the number of bytes stored on the card is trivial and can even be done without reading a card. Indeed, knowing how vote preferences are stored on the card (see 1) we just need to get the maximum number of candidates for each election in progress to obtain the total number of bytes stored.

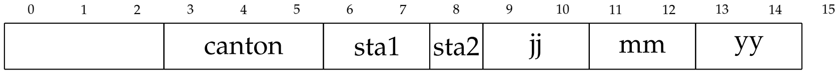

**Figure 4:** *Token computation - Initial token format*

As for the token value, it is written on the card so that the ballot box can verify that the card has been initialized in the same polling station. It is obtained in three steps:

1. An initial token is read from a configuration file stored on the floppy disk. This value is composed of the canton number, secondary and primary number of the polling station and the date of the vote in *ddmmyy* format
2. A hard-coded value, present on the voting machines and ballot boxes containing the value *0EC3ZN678LAB2DFRH1IJK9M5OPQGSTUVWXY4* is read. This value is used to generate the token.
3. The token is generated by fetching values from the hard coded value obtained on step 2 at indexes dependent on values coming from the initial value read on step 1.

```
void Calcul_Jeton(char *Jeton_Etendu, char *Jeton)
{
    int jj, mm,aa;
    int cant,buv1,buv2,tmp;
    char Cjj[3],Cmm[3],Caa[3];
    char Ccant[4], Cbuv1[4],Cbuv2[4];

    strncpy(Ccant,Jeton_Etendu + 3,3);
    strncpy(Cbuv1,Jeton_Etendu  + 6,2);
    strncpy(Cbuv2,Jeton_Etendu  + 8,1);
    strncpy(Cjj,Jeton_Etendu   + 9,2);
    strncpy(Cmm,Jeton_Etendu  + 11,2);
    strncpy(Caa,Jeton_Etendu  + 13,2);

    Ccant[3] = '\0';
    Cbuv1[2] = '\0';
    Cbuv2[1] = '\0';
    Cjj[2]   = '\0';
    Cmm[2]   = '\0';
    Caa[2]   = '\0';

    cant     = atoi(Ccant);
    buv1     = atoi(Cbuv1);
    buv2     = atoi(Cbuv2);
    jj       = atoi(Cjj);
    mm       = atoi(Cmm);
    aa       = atoi(Caa);


    Jeton[1] = _Code[buv1 % 35];
    Jeton[3] = _Code[buv2 % 35];
    Jeton[2] = _Code[(jj + cant) % 35];
    Jeton[0] = _Code[mm   % 35];
    Jeton[4] = _Code[aa   % 35];

    Jeton[5] = '\0';
}
```

**Figure 5:** *Token generation code*

Knowing that the hard coded value has not been changed between each elections since 2006, there are two possibilities to obtain this token: read a magnetic card for the current election and extract the token or infer the initial token by reading your electoral convocation that contains the primary and secondary number of the polling station, along with the election date.

The latest step in forging a rogue magnetic card is computing the HMAC of our vote. Digivote systems use IEC 9797-1[3] (algorithm 2, padding 2). The data on which it is applied when calculating the HMAC is composed of the test and vote bytes. Knowning the algorithm at work, the difficulty lies in obtaining the key used to compute the HMAC of the card.

As shown in Figure 6, the key is derived from a 10 digits value that is composed of 4 digits from the president password (*fullPassword*), 4 digits from a hardcoded value (*Minicodage*) and 2 digits that is obtained by computing the extension of those 8 digits to get a valid checksum (*extendPassword* function).

[3] https://en.wikipedia.org/wiki/ISO/IEC_9797-1

This means that the key is derived from a password of 10 characters from which we know 6 characters. Indeed, the *Minicodage* variable is known and contains 4 characters while we know the algorithm to obtain the last two digits of this password. Which leaves us with $10^4$ possible values to enumerate to obtain this password. The attentive reader will also see how they implemented some kind of security-through-obscurity by modifying the *Minicodage* value for the 2014 elections.

```
#ifdef EL2014
#define MINI_PWD "6987"
#define MINI_POS "2368"
#endif

#ifdef EL2014
char Minicodage[] = MINI_PWD;
#else
char Minicodage[] = "6870";
#endif
// ...
extern char gszMinipassword[12];
//...
#ifdef EL2014
CMinipassword[0] = fullPassword[MINI_POS[0]-49];  //it’s 50 - 49 (1)
CMinipassword[1] = fullPassword[MINI_POS[1]-49];  //it’s 51 - 49 (2)
CMinipassword[2] = fullPassword[MINI_POS[2]-49];  //it’s 54 - 49 (5)
CMinipassword[3] = fullPassword[MINI_POS[3]-49];  //it’s 56 - 49 (7)
#else
CMinipassword[0] = fullPassword[0];
CMinipassword[1] = fullPassword[1];
CMinipassword[2] = fullPassword[3];
CMinipassword[3] = fullPassword[7];
#endif
gszMinipassword[4] = 0x00;
strcat(gszMinipassword,Minicodage);
//...
extendPassword(fullPasswordMini,gszMinipassword);
//...
computeKeyFromPassword (decryptedMacKeyMini, fullPasswordMini);
```

**Figure 6:** *Key derivation code*

In possession of a magnetic card, we can read the vote and HMAC values. Knowing these values, we can use a brute force attack to obtain the secret key with the following process:

1. enumerate the $10^4$ possible combinations of password
2. derive the key for each obtained password
3. compute the vote’s HMAC with the derived key
4. if for a derived key, the HMAC matches the one on the card, we found the key

We are now in posession of all the necessary elements to forge a rogue magnetic card.

**Secure Storage** Once a voter casted his vote on a card at the voting booth, he can insert his card in the reader connected to the ballot box machine. This machine will read the magnetic card content, validate it, write the vote in a file and store the card in a sealed box. As long as the election is in progress, all votes are stored, encrypted, in a temporary file. When the president close the vote, the content of this temporary file is decrypted then encrypted with AES and copied into a new file. The temporary file is then deleted.

**Temporary file encryption** Votes stored on the floppy disk temporary file are encrypted with a XOR cipher that use the *pzPassword* variable as filter. Implementation details can be found on figure 7 and 8.

```
void Encrypt_Decrypt(char *pzInputData, char *pzPassword, unsigned int iSize)
{
    unsigned int i, iKeySize;
    iKeySize = strlen(pzPassword);

    for (i= 0; i < iSize; i++)
    {
        pzInputData[i] ^=  pzPassword[i % iKeySize];
    }
    pzInputData[iSize] = 0x00;
}
```

**Figure 7:** *Temporary file encryption*

The XOR cipher filter is made out of seven bytes from the vote's position in the file, four bytes from *Minicodage* and four bytes from the president's password. Knowing the position of the vote and the value of Minicodage, we have two possibilities at our disposal to get the remaining four bytes of this filter:

- perform a brute force attack by listing $10^4$ possibilities, assuming it is possible to determine whether the decrypted content is valid or not
- since the missing four bytes correspond to the four bytes obtained by brute force in the attack on the magnetic card, we just use this attack to get the four missing bytes

Knowing the filter used for XOR cipher we are able to add and edit the contents of registered votes in the temporary file.

**Decrypting file content** On polling station closing, the temporary file content is decrypted by re-applying the XOR cipher, then encrypted with AES and stored in a new file. At first glance, deciphering the file content seems impossible. To do that, an attacker must get access to the secret key used to encrypt the file.

This key is stored, along with the initialization vector, on a file named *floppy.be*. This file resides on the same floppy disk than the encrypted votes. The key is stored on the first 16 bytes of the file and the initialization vector is stored on bytes 16 to 31. To protect this key and initialization vector, the programs encrypt the *floppy.be* file content with AES, by using a secret key derived from the president's password and a null IV.

As we already know, it is possible to recover six characters out of ten of the president's password via an attack targeting the magnetic card. It would thus be possible to list the $10^4$ possible combinations to obtain the four missing characters, deriving each combination and apply the decryption process with each obtained key.

The format of the lines in the file being known, we consider to be possible to verify that the content matches the decrypted format (for example, ensuring that the type of voting of each line is N, S or E). If it is possible to perform this validation, we will have not only recovered all the

```
void Generate_Password(char *pzPassword, long Position, boolean bIndic)
{
        long E_Position;
        char szPos[8];

        //compute the position in the file
        if(bIndic)
        {
                E_Position = (long)_E_TABLE;
                E_Position +=(long)((long)((long)C_VOTE_MAX_BYTE + 5L) * (long)Position);
        }
        else
                E_Position = Position;

        sprintf(szPos,"%07ld",E_Position);
        if(szPos[0] == ’-’)
                szPos[0] = ’0’;

        pzPassword[0] = CMinipassword[0];
        pzPassword[1] = szPos[3];
        pzPassword[2] = CMinipassword[2];
        pzPassword[3] = szPos[4];
        pzPassword[4] = CMinipassword[7];
        pzPassword[5] = szPos[5];
        pzPassword[6] = CMinipassword[4];
        pzPassword[7] = szPos[6];
        pzPassword[8] = szPos[0];
        pzPassword[9] = CMinipassword[1];
        pzPassword[10] = CMinipassword[3];
        pzPassword[11] = szPos[2];
        pzPassword[12] = CMinipassword[6];
        pzPassword[13] = szPos[1];
        pzPassword[14] = CMinipassword[5];
    pzPassword[15] = 0x00;
}
```

**Figure 8:** *Temporary file encryption*

votes in the file but also the administrator password of the polling station president. The complete process is described in figure 9.

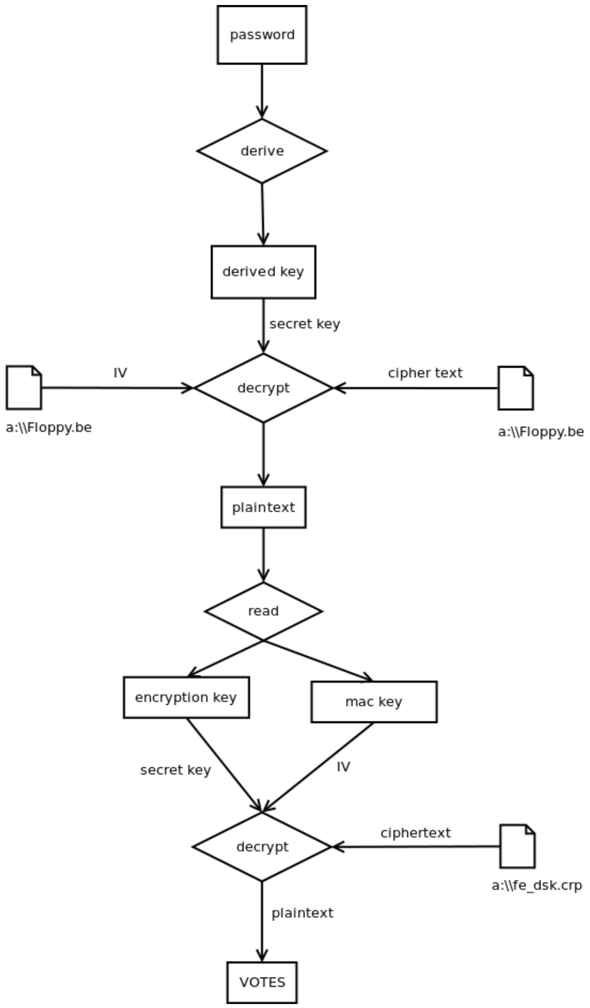

**Figure 9:** *Vote decryption process*

## III. PGM2 & PGM3

**Overview** Totalization software asks the president to encode his password, then read the floppy disk content and produces a result that is printed locally.

### Secure Transmission

## III. Election Management System

When we started our analysis, we have always been convinced that addressing the infrastructure used to send and harvest votes would have been a difficult exercise, entirely based on supposition. Fortunately, Stesud provided the complete technical documentation of this infrastructure (networking maps, description of services and systems). The file was in fact well buried in the public archive as our command log from Figure 10 confirms.

```
$ cd /tmp
$ wget http://www.elections.fgov.be/fileadmin/user\_upload/\
Elections2014/FR/Electeurs/en\_pratique/soft/codi.zip
$ unzip codi.zip
$ cd Codi
$ cd PGM2\ -\ 275/
$ unzip PgmRef.zip
$ cd ZCOCKPIT
$ unzip t15M
$ libreoffice doctechnique01150842.doc
```

**Figure 10:** *Technical documentation digging*

One can assume that the presence of this file is an unfortunate error from Stesud. However, no changes have been applied to this archive since september 2014 when we reported this information leak.

**Secure Transmission** Once the encrypted file written on the floppy disk, this disk is transmitted to the township office so that its content is decrypted and integrated to the results of the canton with the PGM2 software. Once all canton's votes have been encoded, the software generates a PDF file containing a summary of the results. This pdf file is signed by the president with his electronic identity card. This pdf file is then transmitted via PGM3 to the Ministry of Internal Affairs infrastructure. Note that the inner workings of PGM2 and PGM3 is pure speculation, based on the content of user guides provided by Stésud.

**Services** The Pgm programs were developed with Centura and are used on PCs main offices.

- Pgm1: constituency office and college, introduction and validation of lists
- Pgm2: introductions of results, calculation and generation of minutes by main offices
- Pgm3: introduction of results and generation of minutes by cantons electronic offices
- Pgm5: results comparison

The Web applications are developed in php (most with Zend), running on the Ministry of Internal Affairs servers.

- Web1: encoding of lists by political parties, voting offices by municipalities, cantons offices coordinated consultations.
- Web2: results recording by foreign embassies
- Web3: internal intranet for FPS Interior with election results
- Web4: cockpit to monitor operations (logging facility)
- Web5: public web server where results are published

The Loc programs have been developed by Centura and run on the Ministry of Internal Affairs servers.

- Loc1: receiving files from the main offices and transfer to Loc2

- Loc2: checking files received from Loc2, database loading, calculation and transfer of results to Loc3, consultation of data stored in the Loc2 database.

- Loc3: transfer of results to partners like the press.

**Network** As it can be seen on the network maps in figure 11, clients can access the servers both via the Internet and Publilink networks.

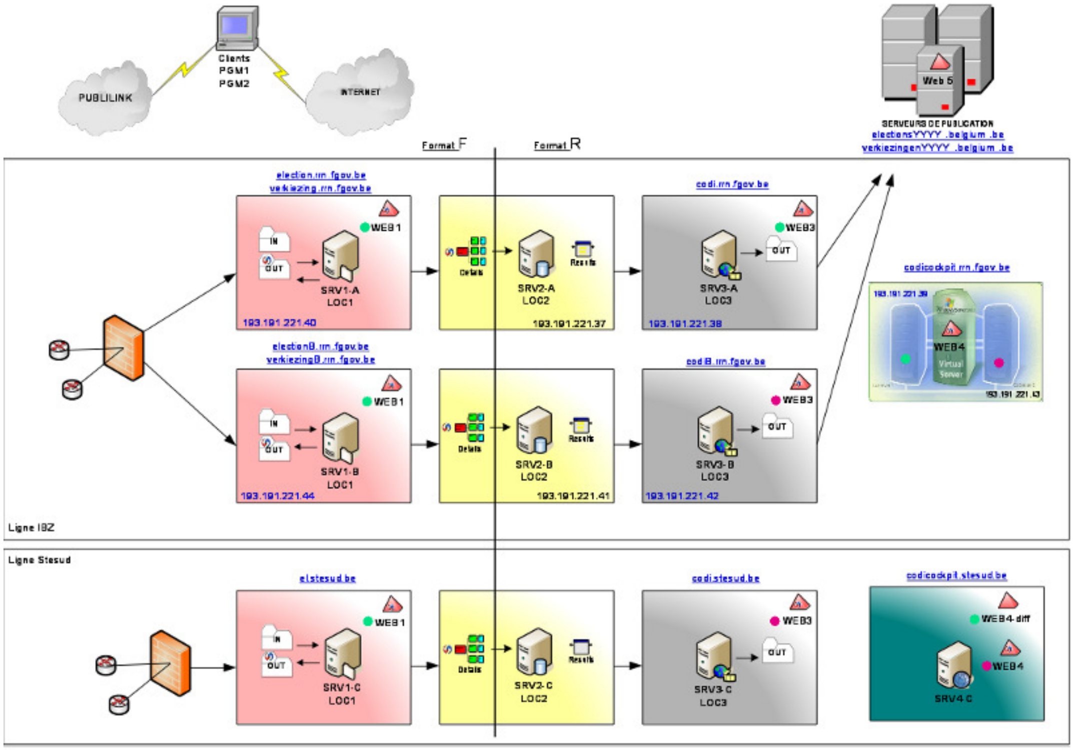

**Figure 11:** *Network map*

For information, "PubliLink is designed on a private and completely locked network managed by Belgacom for various governments services, Belfius Bank and other public service providers."

One can decently ask the following questions about this network:

- Why provide the opportunity for PGM1 and PGM2 clients to connect to the vote infrastructure via the Internet while Publilink seems to be a much safer and secure solution ?

- Why is the backup infrastructure hosted by a private provider?

- What guarantees regarding network security can Stésud and the Ministry of Internal Affairs can provide? Does the network has been audited?

**Web Application (In)Security** After going up the chain to the infrastructure of the Ministry of Internal Affairs, we looked at the applications that enable the entire electoral process to take place. These web applications receive, compute and redistribute voting results for all Belgian voters. Whether you voted on paper or electronically, your vote was recorded via these applications. Be aware that only Web2 sources are available on the website of the Ministry of Internal Affairs. This means that seven out of eight applications used in the receipt, calculation and dissemination of the votes are not provided to citizens even if they take an equally important part in the electoral process.

During our analysis, we found the following kind of vulnerabilities in Web2:

- Arbitrary file download
- Sensitive information disclosure (private keys)
- Cleartext storage of password

All those findings are documented in our online article[4] and were responsibly disclosed to Stesud. We chose not to explore those findings in this paper for legibility concerns.

## IV. Conclusion

Our analysis proved that the Digivote system do not provide new guarantees in regard to paper voting. It is possible to create rogue magnetic card to make ballot stuffing or vote buying and it is also possible to manipulate the votes when stored locally. Even if we covered a good part of the software running on polling stations, many gray areas remain, mainly related to web applications behind the Election Management System and the network on which they run.

Furthermore, we discovered multiple vulnerabilities affecting one of the many web applications running the EMS; which lead us to think that many more are yet to be discovered. We hope that the belgian Ministry of Home Affairs will take the necessary steps to ensure the security of their infrastructure before the next election take place.

An electronic voting system, when properly designed and implemented, is an effective tool to address the challenges of democratic elections. Unfortunately, the lack of control, means and expertise of the Ministry of Internal Affairs led it to renew this system beyond its limits. Renewal that ultimately led to a total loss of confidence from citizens towards these systems.

## V. Future Work

Must evoting stay in place, we think this kind of audit should automatically be performed by groups of citizens possessing the necessary skills. To help this process taking place, we are currently devising a methodology to audit electronic voting systems that try to capture all the threats that those systems faces from network tampering to hardware and web applications hacking. Once we publish this methodology, we will likely start to look at other electronic systems deployed in countries around the world. More localy, we started to look at the Smartmatic system which is also deployed in Belgium.

[4] http://qkaiser.github.io/analysis/2015/05/12/how-not-to-build-an-evoting-system/

## References

[1] Affront. Affront analysis of 2003/2004 versions of digivote. *Affront*, 2004.

[2] D. Wagner C. Karlof, N. Sastry. Cryptographic voting protocols: A systems perspective. *14th USENIX Security Symposium.*

[3] Internet Policy Institute. Voting systems design criteria. report of the national workshop on internet voting: Issues and research agenda. March 2000.

[4] LaLibre.be. Le parlement wallon se prononce en faveur de la fin du vote électronique en belgique, June 2015. .

[5] Medor Mag. Le jour où la belgique a bugué., May 2015. .

[6] Oladiran Tayo Arulogun Olayemi Mikail Olaniyi, Adeoye Oludotun and Elijah Olusayo Omidior. Design of secure electronic voting system using multifactor authentication and cryptographic hash functions. *International Journal of Computer and Information Technology*, November 2013.

[7] PourEVA. Comment frauder lors d'une élection communale sans trop de connaissances informatiques ?, November 2006. .

[8] PourEVA. Victoire de la transparence au conseil d'etat, May 2011. .

[9] PourEVA. Généalogie du code source des systèmes digivote et jites, June 2014. .

[10] PourEVA. On vous dit tout ce que l'on sait du bug2505, June 2014. .