Een technische analyse van RIESKOA

Sven Haster

Juni 2007

1 Inhoudsopgave

2 Inleiding

In de verkiezingen voor het waterschap De Dommel en het hoogheemraadschap Rijnland in 2004 werd voor het eerst gebruik gemaakt van internetstemmen, je stem uitbrengen via het internet. Van de 2,2 miljoen mogelijke stemmers maakten meer dan 120.000 gebruik van deze mogelijkheid om hun stem uit te brengen.
Het hierbij gebruikte systeem trok de aandacht van het ministerie van Binnenlandse Zaken en Koninkrijksrelaties, die onder de noemer Kiezen op Afstand (KOA) al enige tijd aan het experimenteren waren met het fenomeen internetstemmen. Rijnland Internet Election System, kortweg RIES, was een simpel, dus doorzichtig, maar krachtig systeem gebaseerd op de afstudeerscriptie van een student van de TU Delft.
Onder de noemer RIESKOA werd een verbeterde versie van het systeem tijdens de Tweede Kamerverkiezingen van november 2006 ingezet. De zgn. ex-pats (Nederlanders die in het buitenland wonen of ten tijde van de verkiezingen in het buitenland verblijven) konden kiezen uit een aantal manieren via welke ze hun stem uitbrachten en internetstemmen met behulp van RIESKOA was een van de mogelijkheden.

In deze scriptie ga ik dieper in op de security-technische aspecten van RIESKOA. Hoe zit het nu eigenlijk in elkaar? Wat zijn de gebruikte protocollen? Hoe vatbaar is het systeem voor aanvallers? Ook doe ik verslag van een kleine internetservice om het controleren van de uitgebrachte stemmen te vergemakkelijken.

RIES

2.1 Introductie

Rijnland Internet Election Systeem, kortweg RIES, is een door TTPI ontwikkeld systeem om online (via internet) stemmen mogelijk te maken. RIES is ontwikkeld op basis van een scriptie, geschreven door Herman Robers [4], over een elektronisch stemsysteem met smartcards dat zijn veiligheid baseert op sterke protocollen. Zoals het systeem uit die scriptie is RIES gebaseerd op sterke protocollen en versleutelingen.
RIES werd ontwikkeld voor de verkiezingen van het hoogheemraadschap Rijnland en waterschap De Dommel in 2004, en werd aangepast voor gebruik tijdens de Tweede Kamerverkiezingen van 2006. De versies staan bekend als respectievelijk RIES2004 en RIESKOA.
In dit document wordt met name RIESKOA onder de loep genomen, hoewel verwijzingen naar RIES2004 niet ontbreken.
Alle informatie m.b.t. tot RIESKOA is afgeleid uit [2], tenzij anders vermeld.

2.2 Fases

RIES, zoals gebruikt bij beide verkiezingen, bestaat uit de volgende fases:

A. Voorbereidingsfase

a. Hierin wordt als eerste het totale aantal virtuele kiezers bepaald. Er zijn meer virtuele kiezers (‘virtuele identiteiten’) dan daadwerkelijk geregistreerde kiezers, zodat kiezers een nieuwe virtuele identiteit kunnen krijgen als ze de codes van hun oude identiteit zijn kwijtgeraakt o.i.d.

b. Vervolgens wordt voor iedere virtuele identiteit een unieke *DES-key* K*p* gegenereerd. Tevens wordt er een unieke verkiezings id ElID en een uitgebreide verkiezings id BalBxID, en voor iedere kandidaat een unieke kandidaats id Cm gegenereerd.
Cm is een codering van de verkiezing gevolgd door de partij en kandidaat van de betreffende kandidaat; Cm = BalBxID//PolGrp//CSeqNmbr [2 (p. 113)]

c. De MDC{DESmac_{Kp}(BalBxID)}, ook bekend als RnPID, wordt berekend voor elke virtuele identiteit. De combinatie van MDC{MAC_{Kp}{Cm}}, RnCm genaamd, voor alle Cm’s is de collectie van mogelijke stemmen die uitgebracht kunnen worden door de virtuele identiteit geïdentificeerd door RnPID.
Deze collectie wordt gegenereerd voor elke virtuele identiteit, en dit samen vormt het zgn. pre-election referentiebestand.

d. Elke geregistreerde kiezer krijgt een virtuele identiteit toegewezen, en hem of haar wordt op papier een stemcode toegezonden waaruit zijn/haar unieke *DES-key* K*p* kan worden berekend.

e. In het pre-election referentiebestand, wat feitelijk een gezipte map is, met in die map zestien gezipte bestanden, RT_0.zip t/m RT_F.zip, met in elke RT_x.zip voor elke RnPID beginnend met x een bestand, met de naam gelijk aan de RnPID waarin drie status[bits] zijn opgenomen en de stemcollectie. De drie statusbits geven aan of de betreffende RnPID is uitgegeven, bedoeld als evt. vervangende RnPID, of is herroepen. De stemcollectie bestaat uit voor elke RnCm behorende bij deze RnPID een regel uit twee delen, met links van het =-teken RnCm en rechts Cm.

f. Dit referentiebestand wordt online gezet en een *checksum* van dit bestand, om te controleren dat het bestand ongewijzigd is, wordt gepubliceerd in de Staatscourant en een aantal andere kranten.

g. Gedurende enige tijd is er de mogelijkheid voor de kiezers om nieuwe/vervangende virtuele identiteiten aan te vragen. Als er kiezers een andere RnPID aanvragen, wordt hun oude RnPID op 'herroepen' gezet en een RnPID die als status 'vervangend' had wordt op 'uitgegeven' gezet.

h. De door de helpdesk uitgevoerde mutaties t.b.v. kiezers die een vervangende RnPID aanvroegen worden ingevoerd in het referentiebestand. Nadat de termijn waarin kiezers een vervangende RnPID kunnen aanvragen is verstreken wordt de nieuwste versie van het referentiebestand online gezet en de *checksum* hiervan wordt ook gepubliceerd. Dit bestand staat ook wel bekend als de *post-electiontable.*

i. Hieronder wordt dit laatste referentiebestand getoond, net als één van de RT_x-bestanden en (een gedeelte van) een bestand uit zo'n RT_x-bestand:

```
$ unzip -l reftablesmut.zip
Archive:  reftablesmut.zip
  Length     Date   Time    Name
 --------    ----   ----    ----
 19924438  11-17-06 22:12   50011201/RT_0.zip
 20660031  11-17-06 22:12   50011201/RT_1.zip
 19680625  11-17-06 22:12   50011201/RT_2.zip
 20034165  11-17-06 22:12   50011201/RT_3.zip
 20183630  11-17-06 22:12   50011201/RT_4.zip
 20914626  11-17-06 22:12   50011201/RT_5.zip
 20548506  11-17-06 22:12   50011201/RT_6.zip
 20938451  11-17-06 22:12   50011201/RT_7.zip
 18774773  11-17-06 22:13   50011201/RT_8.zip
 19869972  11-17-06 22:13   50011201/RT_9.zip
 19836343  11-17-06 22:13   50011201/RT_A.zip
 19345074  11-17-06 22:13   50011201/RT_B.zip
 19022743  11-17-06 22:13   50011201/RT_C.zip
 20779952  11-17-06 22:13   50011201/RT_D.zip
 19495250  11-17-06 22:13   50011201/RT_E.zip
 20604108  11-17-06 22:13   50011201/RT_F.zip
        0  11-03-06 12:32   50011201/
 --------                   -------
320612687                   17 files
```

```
$ unzip -lM RT_0.zip
Archive:  RT_0.zip
  Length     Date   Time    Name
 --------    ----   ----    ----
    27629  01-01-80 00:00   0AACE8B9E0B4E0CF180D0FE020605199
    27629  01-01-80 00:00   05132BC34CA6F8A5BCDDBBE0747EBB53
    27629  01-01-80 00:00   099C8147AC20ABA67C61181CAA38C733
    27629  01-01-80 00:00   0BE7964CD45D4186D4787DA3265FDE02
    27629  01-01-80 00:00   04272241639788B38B0EE11B3BA07CDD8
    27629  01-01-80 00:00   0972C67E0066BE6212836F4CD7898428
    27629  01-01-80 00:00   05CCC162D7F573A6EDB21BD2C686A0AC
    27629  01-01-80 00:00   0DD807F1BEE697E26149EF5D7CBFCD64


vervangend=0
verstrekt=1
vervallen=0
9C9894D707CE7F906EBDA91CD27A9882=500112010101
8B88BBC438247DBD190D1B6E3847B2BE=500112010102
BA585ABE4E6BF9219C845BB061EAB2A0=500112010103
674A297915B563DD2D647A454955144C=500112010104
```

Hieruit blijkt duidelijk dat de RT_x.zip-bestanden vóór de verkiezingen zijn gemaakt. De datum staat op 17 november 2006, en de verkiezing liep (op internet) van 18 november 2006 tot 22 november 2006.

B. Stemmen

a. Met behulp van een aantal internetpagina's wordt de kiezer door het kiesproces geholpen. De verbinding is beveiligd via SSL en de verstuurde informatie is beveiligd door de *hashes* van RIESKOA.

b. Na het maken van een keuze voor een kandidaat om op te stemmen wordt $DESmac_{Kp}$ {Cm}, VnCx[1], berekend. Tezamen met de VnPID (=$DESmac_{Kp}$(BalBxID)) wordt dit naar de stemserver gestuurd tijdens een SSL-sessie.

c. De stemserver slaat de ontvangen combinatie, ook wel bekend als de 'technische stem', op en berekent een ontvangstbevestiging die naar de kiezer wordt gestuurd.

d. De kiezer kan na het versturen van de technische stem eenmalig ervoor kiezen om deze te bewaren. Dit kan echter niet na te zijn uitgelogd en later opnieuw in te loggen. Te allen tijde kan de kiezer zijn ontvangstbevestiging opvragen.

C. Stemcontrole en tellen

a. Om de stem te verwerken wordt van elke ontvangen stem de MDC-hash berekend. Deze wordt vergeleken met de hashes in het referentiebestand. Komt hij voor, dan wordt aan de hand van de referentietabel berekend op welke kandidaat is gestemd. Zo niet, dan is de stem ongeldig.

b. De stemmen worden ook nog gecontroleerd op geldigheid. (Bijv. als er twee stemmen zijn ontvangen van één kiezer op verschillende kandidaten worden alle stemmen van die kiezer ongeldig verklaard).

c. Alle geldige stemmen worden per kandidaat geteld en hiervan wordt een rapport opgesteld.

d. Het rapport, de ontvangen stemmen (de verwerkte én de onverwerkte versie) en het uiteindelijke referentiebestand worden online gezet.

---

[1] VnCx ipv het verwachte VnCm. Dit is waarschijnlijk omdat de 'x' in Cx een 'geheim' kandidaatsnummer, als in de keuze voor die kandidaat, aanduidt in tegenstelling tot Cm, wat gewoon het publieke kandidaatsnummer is.

2.3 Protocollen en versleutelingen

In deze sectie wordt er wat dieper ingegaan op de door RIESKOA gebruikte versleutelingen en protocollen.
Ook hier kunnen we de grove indeling in fases gebruiken.

A. Voorbereidingsfase

a. Er wordt uitgegaan van maximaal 180.000 internetstemmers [2 (p. 19)]. Voor het aantal vervangende stembescheiden wordt er uitgegaan van 1% van de stembusomvang [2 (p. 30)]. Uitgaande van een stembusomvang van 180.000 zal dit dus 1.800 zijn geweest.

b. Voor iedere stemmer wordt er een unieke sleutel gegenereerd. Deze sleutel wordt gerepresenteerd met behulp van 18 karakters in het AN34 formaat (*lowercase*). Deze 18 karakters bestaan uit:
   - 2 karakters voor de deelnamegroepering (DG)
   - 8 karakters voor de Stemcode-1, VPID1
   - 8 karakters voor de Stemcode-2, VPID2

   Deze drie sleuteldelen zijn terug te vinden in het WV-STUF-C10. [2 (p. 39-40)]

c. Ten behoeve van de cryptografische algoritmen worden er nog 3 geheime DES-sleutels gegenereerd, t.w. de Masterkey, de GenVotersKey en de ReceiptKey [2 (p. 41)]. Deze worden gegenereerd met behulp van zgn. *saltparts* waarbij voor iedere sleutel nieuwe *saltparts* worden gegenereerd [2 (p.41)].

d. Per sleutel wordt één *saltpart* gegenereerd door TTPI, de rest wordt gegenereerd door BZK. *Passphrases* zijn de basis voor de *saltparts*, deze *passphrases* zijn vrij te kiezen maar dienen 16 tot 64 karakters lang te zijn. Na een karakterfilter (houdt minimaal 8 bytes over) wordt van elke *passphrase* door middel van PKCS#5 met IBM's MDC-2 en een onbekende *salt* en *iteration count* een 16-byte *saltpart* gegenereerd.
   Bij PKCS#5 wordt een sleutel, of in dit geval saltpart, afgeleid van een *passphrase* door een *pseudorandom* functie, zoals een hash functie, toe te passen op de passphrase gecombineerd met de *salt*. Deze berekening wordt herhaaldelijk toegepast op het resultaat hiervan, typisch 1000 maal. (Dit herhalingsaantal wordt de *iteration count* genoemd.) [9]

e. De hele procedure wordt dubbel uitgevoerd om cryptografische fouten te voorkomen. De gegenereerde *saltparts* worden op CD gebrand en van de gebruikte machines verwijderd. De CD wordt door BZK bewaard in een kluis en op verzoek van TTPI ter beschikking gesteld, welke verzoeken door BZK worden bijgehouden in een log. [2 (p. 41)]

f. De Masterkey (MK) wordt op onbekende wijze berekend uit vier *saltparts* en vervolgens opgeslagen in een bestand genaamd KM_salt_store.txt [2 (p.41)].

De GenVotersKey en ReceiptKey worden ook berekend aan de hand van de *saltparts*. [2 (p. 42)]
Verdere informatie over de berekening van bovenstaande sleutels was niet te verkrijgen uit de beschikbare documenten.

g. Volgens [2 (p.141-144)] is Kp:=DESe(Kgenvoterkey, {VnID//ElID//ParGp}). Hierbij is:
    - Kgenvoterkey is een 16 bytes encryptie sleutel voor 3DES.
    - VnID is een publieke *voter id*, de binaire representatie van een 10 cijferig decimaal getal, *most significant side padding* met 0-bytes tot 5 bytes lengte.
    - ElID is de *Election ID*, gespecificeerd door TTPI in het S00-bestand [2 (p. 28)] en opgenomen in de K10, K11 en C10 bestanden [2 (p. 28, 30, 31)]., 2 bytes.
    - ParGp (*partcipation group*) is de bovengenoemde DG, 1 byte.
    - DESe is DES-encryptie, 3DES ECB encryptie met een resultaat van 8 bytes.

    [2 (p. 45)] vermeldt een ogenschijnlijk andere berekeningswijze voor Kp, maar op het gebruik van andere namen voor dezelfde dingen na lijkt dit dezelfde berekening te zijn.

B. Stemmen

De gegevenswisseling tussen client en stemserver bij een normale stemming ziet er als volgt uit (hierbij zijn 'triviale' informatieschermen niet vermeld) [2 (p. 86-87)]: *Zodra er pagina's worden opgevraagd van de stemserver wordt er een SSL-verbinding opgezet.*

In deze tabel staat tekst in de derde persoon actief voor acties ondernomen door een persoon. Tekst in de derde persoon passief staat voor acties ondernomen door software, bijv. de javascript-code in de webpagina's.

Cliënt		Stemserver
	ElID, RnTC ←	
Voert VPID1 en VPID2 in. Hieruit wordt de ReSPID berekend.		
	ElID, ReSPID, RnTC →	
		ReSPID wordt gecontroleerd.
	Lijst PolGrp (lijst politieke groeperingen) ←	
Kiest PolGrp.		
	PolGrp →	
	Lijst kandidaten ←	
Kiest kandidaat. De technische stem wordt berekend.		
	Technische Stem, ReSPID, RnTC →	
		ReSPID wordt opnieuw gecontroleerd, de technische stem wordt opgeslagen en een ontvangstbevestiging wordt berekend.
	Ontvangstbevestiging (cliëntdeel) ←	

a. ReSPID = MDC[DESmac$_{Kp}${ f(ElID//ExtParGp)}] [2 (p. 142)].
RnTC is bedoeld voor een extra identificatiefunctionaliteit die binnen

RIESKOA niet gebruikt wordt, hiervoor wordt een standaardwaarde ingevuld [2 (p. 133)].

b. ReSPID wordt gecontroleerd in een 'status'-database, waarin met behulp van de ReSPID kan worden gecontroleerd of de bijbehorende identiteit wel mág stemmen (uitgegeven is, niet ongeldig is verklaard, etc.).

c. De technische stem bestaat uit twee delen: VnPID en VnCx (ieder 8 byte lang).
VnPID = $DESmac_{K_p}$ {f(BalBxID)}
VnCx = $DESmac_{K_p}${f(Cx)}
Cx = Kandidaatsnummer
f() is een functie die zijn argument uitbreidt d.m.v. *least significant side padding* met 0-bytes (tot een lengte van 8 bytes)
[2 (p. 142)]
Meer informatie over DES en DESmac is te vinden in secties 3.2 en 3.3

d. De ontvangstbevestiging is hoogstwaarschijnlijk gelijk aan VotRecConCnt.
VotRecConCnt = laatste (*low order*) 4 bytes van VotRecCon
VotRecCon = $DESmac_{Kbbs\_b}$ { (VnPID//VnCx)}
Kbbs_b is een 16 bytes sleutel voor DESmac van BBS_b
BBS_b is BallotBoxServer b, oftewel een stemserver (er zijn tijdens RIESKOA twee stemservers actief) [2 p.(16, 143)].
Het blijkt dat cliënten voor verschillende kandidaten dezelfde ontvangstbevestiging krijgen. Blijkbaar wordt de ontvangstbevestiging dus niet beïnvloed door de kandidaatskeuze, maar door de (virtuele) identiteit van de kiezer. Dit wordt bevestigd door [8, (p. 27)]: "*The acknowledgement cannot be used as a proof of vote as it does not reveal the contents*…".

C. Stemcontrole en tellen

a. De ontvangen stemmen worden omgezet naar *uppercase*, de stemmen worden in het juiste *format* gezet [zie 3 (p. 113)] en de RnVPID en RnCx worden berekend [2 (p.99)].
RnVPID = MDC{VnPID} [2 (p. 113)]
VnPID = $DESmac_{K_p}$ {f(BalBxID) } [2 (p. 142)]
BalBxID = 4 bytes, (ElID//BalDis//CatID)
BalDis = 1 byte indicatie van het kiesdistrict
CatID = 1 byte Categorie ID
Overigens staat in het FO nog een andere berekening van de VnPID [2 (p. 113)], maar uit de implementatie blijkt dat bovengenoemde wordt gebruikt.

b. De stemmen worden gesorteerd op RnVPID, dan op RnCx. Het zo verkregen *bruto received votes* bestand wordt gepubliceerd en verder verwerkt [2 (p. 99)].

c. Alle stemmen met dezelfde RnVPID worden samen genomen (geclusterd) en het eerste cluster wordt geselecteerd [2 p. (100)],

d. Van het geselecteerde cluster wordt bekeken (in deze volgorde) [2 (p. 101-102)]:

- Is de RnVPID geldig (komt deze voor in een referentiebestand)? Zo nee, markeer alle stemmen in dit cluster als ongeldig (Invalid_RnVPID) en ga naar punt e.
- Is de RnVPID een testidentiteit? Zo ja, markeer alle stemmen in dit cluster als ongeldig (Invalid_Test) en ga naar punt e.
- Is de RnVPID van een vervallen identiteit? Zo ja, markeer alle stemmen in dit cluster als ongeldig (Invalid_Vervallen) en ga naar punt e.
- Is de RnVPID verstrekt (zijn de stembescheiden hiervan uitgegeven)? Zo nee, markeer alle stemmen in dit cluster als ongeldig (Invalid_NietVerstrekt) en ga naar punt e.
- Nu wordt voor elke stem in dit cluster gecontroleerd of de RnCx (de kandidaatskeuze) geldig is. Zo ja, dan is er een keuze gemaakt. Zo nee, dan wordt deze stem gemarkeerd als ongeldig (Invalid_RnCx). Als alle stemmen in dit cluster ongeldig zijn wordt het cluster als ongeldig geregistreerd.
- Als de keuze is gemaakt: Zijn er nog meer stemmen binnen dit cluster? Zo ja controleer voor elke stem de RnCx. Is deze ongeldig dan wordt de stem als ongeldig gemarkeerd (Invalid_RnCx). Is deze geldig en op dezelfde kandidaat dan wordt deze stem aangemerkt als ongeldig (double vote), maar de Cx (kandidaatscode) wordt wel ingevuld.
  Is deze geldig maar op een andere kandidaat, dan wordt deze stem gemarkeerd als ongeldig (contrary votes), maar de Cx (kandidaatscode) wordt wel ingevuld, en de huidige keuze wordt ook gemarkeerd als ongeldig (contrary votes).
- Is de keuze uiteindelijk alleen nog over, dan wordt de Cx ingevuld en als de keuze nog steeds geldig is wordt deze aangemerkt als ‘Geteld’.

e. Als er nog een onbehandeld cluster over is wordt dit geselecteerd en punt d wordt uitgevoerd [2 (p. 100)].

f. Het ‘performed_votes’-bestand (het bestand met alle gecontroleerde en ingevulde stemmen), het *received_votes*-bestand, het *post-election*-referentiebestand en de uitslag van de verkiezingen worden via internet gepubliceerd.

2.4 Samenvatting

RIES is een systeem wat is gebaseerd op het idee van veiligheid door helderheid en controleerbaarheid, zonder aan kracht in te boeten. Dit komt tot uiting in het simpele ontwerp en de, over het algemeen, vrij beschikbare informatie, wat alle stappen gemakkelijk te volgen maakt, en het geregeld publiceren van referentiebestanden en *checksums* hiervan, wat de telling zowel als het verwerken van individuele stemmen te controleren maakt.

3 Security-aspecten

Hoe veilig is RIESKOA? Wat moet geheim blijven, en wat mag openbaar worden? Welke aanvallen zijn er mogelijk, op welke manier kunnen deze worden uitgevoerd en wat zijn de mogelijke beveiligingen hiertegen?

3.1 CIA

In de *computersecurity* is 'CIA' een veelgebruikt acroniem voor de belangrijkste veiligheidsaspecten. Hier worden vaak ook letters aan toegevoegd voor extra aspecten, zoals het acroniem CIA-ANA. Dit staat in zijn algemeenheid voor (geadapteerd van [6 (p. 6-7)]):

- **Confidentiality**
  Dit staat voor de verzekering dat data niet kan worden gelezen door ongeautoriseerde entiteiten (personen, toepassingen, apparaten).
  Confidentiality wordt meestal verzekerd met behulp van cryptografie.
- **Integrity**
  Letterlijk integriteit. De zekerheid dat data niet is veranderd. Hiervoor wordt meestal gebruik gemaakt van digitale handtekeningen en *hash*-algoritmen.
- **Availability**
  De beschikbaarheid van data en diensten is een subjectiever aspect. Deze moeten snel en betrouwbaar beschikbaar zijn. Meestal wordt dit geformuleerd als: snel en betrouwbaar genoeg dat het hoofdsysteem zijn taken kan uitvoeren zoals bedoeld. Specifiek, het vermijden van *Denial of Service*: een toestand waarin geautoriseerde toegang tot data en diensten onmogelijk is of tijdskritieke operaties vertraagd worden.
- **Accountability**
  Dit ietwat minder strikt geformuleerde aspect gaat over de verzekering dat elke activiteit met betrekking tot de data binnen redelijke tijd en moeite getraceerd kan worden tot een geautoriseerde entiteit.
- **Non-repudiation**
  Letterlijk betekent dit zoveel als 'non-verwerping', waarmee 'verwerpen' gelezen moet worden als 'het ontkennen van een transactie', dus de zekerheid dat geen van beide partijen (zender en ontvanger) het bestaan van een transactie kan ontkennen. De verzender heeft een bewijs van ontvangst, en de ontvanger heeft een bewijs van de identiteit van de verzender.
- **Authentication**
  Authenticatie, ofwel de mogelijkheid om de identiteit van betrokken entiteiten te verifiëren.

Deze zes aspecten zijn ook vertegenwoordigd in RIESKOA. Hieronder wordt de 'vertaling' gegeven van deze aspecten naar doelen en beveiligingen die betrekking hebben op de specifieke situatie van RIESKOA.

- **Confidentiality**
  Binnen de context van internetstemmen, staat dit voor het stemgeheim. Niemand mag erachter kunnen komen wat er door een specifieke persoon gestemd is. Binnen RIES is dit gecompromitteerd: na het uitbrengen van een stem ontvangt de kiezer een zgn. technische stem. Met behulp hiervan en het 'performed_votes'-bestand, dat na de

verwerking van de stemmen wordt gepubliceerd, kan iedereen die over een technische stem beschikt controleren of deze is geaccepteerd en voor welke kandidaat er was gestemd.
Of de technische stem genoeg is beveiligd dat alleen de kiezer zelf deze te zien kan krijgen is te lezen in sectie 4.3.

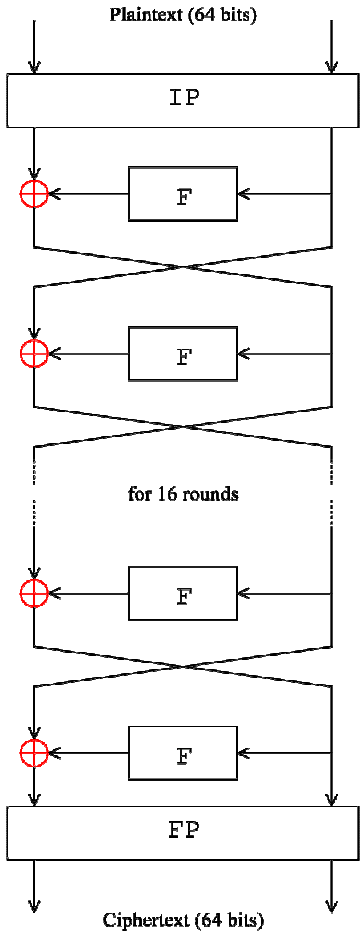
- **Integrity**
  In de meest letterlijke zin, gaat het er hier om dat niemand (ongemerkt) een stem kan wijzigen terwijl hij onderweg is van de kiezer naar de stemserver. Of dit het geval is hangt af van de gebruikte protocollen en versleutelingsalgoritmen. Zie secties 2.3, 4.2, 4.4 voor meer informatie en een onderzoek hiernaar.
  Als de betekenis wat breder wordt genomen, gaat het er ook over dat niemand de stem kan wijzigen nadat de kiezer deze heeft uitgebracht. Een voor de hand liggende manier om de stem te wijzigen is een tweede stem uit te brengen op een andere kandidaat, aangezien dit ervoor zorgt dat beide stemmen ongeldig worden verklaard; effectief is hiermee de stem van de kiezer geannuleerd. Dit is binnen RIES een risico; zie sectie 4.5 voor een onderzoek naar de gevoeligheid voor zo een aanval.
- **Availability**
  Is de stemserver beschikbaar voor de volledige tijd dat de kiezers kunnen stemmen? Zie sectie 4.5 voor een onderzoek naar de haalbaarheid van een (D)DoS-aanval.
- **Accountability**
  Wordt er gecontroleerd of een stem wel is uitgebracht door een geldige ‘virtuele’ identiteit? In het kort, ja. Een stem die is uitgebracht door een ongeldige ‘virtuele’ identiteit wordt bij de telling genegeerd (maar wel vermeld in het ‘performed_votes’-bestand). Zie ook 2.3.C
- **Non-repudiation**
  Naast de al vermelde technische stem, ontvangt de kiezer ook een ontvangstbevestiging na het uitbrengen van zijn stem. Deze ontvangstbevestiging bevestigd alleen het bestaan van een transactie, maar is niet beïnvloed door de inhoud hiervan (iedere transactie van dezelfde kiezer krijgt dezelfde ontvangstbevestiging, zie ook 2.3.B.d).
  De stemserver slaat de technische stem op. Deze is uniek voor de persoonlijke stemcode van de kiezer. Het geheim houden van deze stemcode is de verantwoordelijkheid van de kiezer.
  Zie sectie 4.4 voor een onderzoek naar hoe goed de stemcode (de geheime sleutel van de kiezer) beveiligd is in het stemproces zelf.
- **Authentication**
  De stemserver authenticeert de kiezer door de uitgebrachte stem, er van uitgaande dat de voornoemde kiezer de enige is die deze stem bezit.
  De kiezer authenticeert de stemserver op basis van het webadres (URL) dat getoond wordt in het adresvak van zijn webbrowser, alsmede door het certificaat behorende bij de opgezette SSL-verbinding.

3.2 DES

DES is hét versleutelingsalgoritme dat RIESKOA gebruikt. Hier wordt in het kort uitgelegd wat DES inhoudt.

3.2.1 Introductie

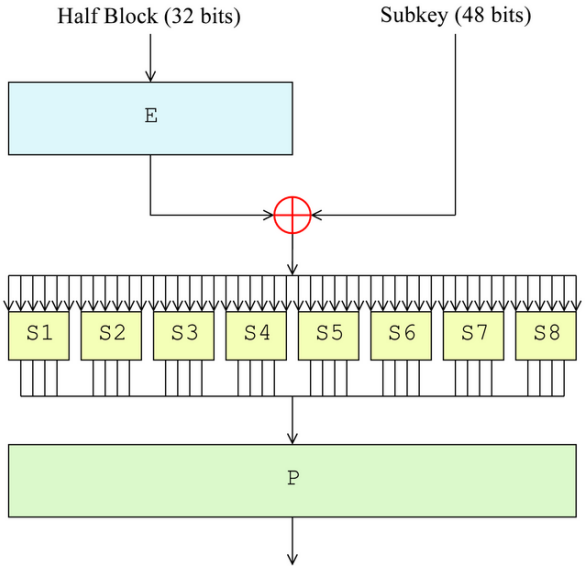
De Data Encryption Standard (DES), werd in 1974 ontwikkeld door IBM in reactie op een publieke oproep van de Amerikaanse NBS (nu NIST)[2]. Het werd in 1976 aangenomen als federale standaard en geautoriseerd voor gebruik op alle ongeclassificeerde overheidsgegevens.
DES is een zogenaamde *block cipher*, die werkt op blokken data van elk 64 bits, en deze versleutelt met behulp van een *symmetrische* sleutel; d.w.z. dat dezelfde sleutel voor versleuteling en voor ontsleuteling wordt gebruikt
De sleutel is 56 bit lang. Meestal wordt elke 7 bits uitgebreid met een *parity*-bit op de minst significante positie, resulterend in een uiteindelijke sleutellengte van 64 bits, maar deze *parity*-bits worden tijdens de encryptie genegeerd. De encryptie maakt gebruik van simpele aritmetische en logische bitoperaties die gemakkelijk te implementeren waren in hardware in de jaren 70 (en ook nu nog).



Het DES-netwerk [A]

3.2.2 Beschrijving

Een 64-bits blok plaintext wordt gepermuteerd (de initiële permutatie of IP), en daarna opgebroken in een linker- en rechterhelft van 32 bits. Hierna komen 16 ronden van dezelfde operaties, ook wel functie F genoemd, waarin de data wordt gecombineerd met de sleutel (en zichzelf). Hierna worden de twee helften weer samengevoegd en opnieuw gepermuteerd, ditmaal op de omgekeerde manier (dit heet de *final permutation* of FP).



De functie F [A]

Gedurende elke ronde wordt de sleutel ge*shift*, en worden hiervan via een permutatie 48 bits geselecteerd. Dit vormt samen met de rechterhelft van de data de invoer van functie F. De rechterhelft van de data wordt via een permutatie opgeblazen (*expanded*) tot 48 bits. Hierna worden deze twee 48-bits blokken geXORd, en de resulterende 48 bits worden, in groepjes van 6,

[2] National Bureau of Standards, nu National Institute of Standards and Technology. Dit overheidsorgaan van de Verenigde Staten draagt onder andere zorg voor het ontwikkelen en verbeteren van technologische standaarden.

door 8 zgn. S-boxen (*substitution boxes*) gestuurd.
Deze S-boxen doen feitelijk niet veel meer dan hun (6-bits) invoer opzoeken in een grote tabel en de bijbehorende vier bits uitvoeren. Dus uit 8 S-boxen komt 32-bits. Deze 32-bits worden vervolgens weer gepermuteerd, en vormen dan de uitvoer van f.
Deze uitvoer wordt geXORd met de linkerhelft van de data, en het resultaat hiervan vormt de nieuwe rechterhelft, terwijl de oude rechterhelft de nieuwe linkerhelft vormt.

3.2.3 De veiligheid van DES

DES was oorspronkelijk ontwikkeld om in hardware te worden geïmplementeerd, en dit is ook de snelste manier. Een hardware implementatie van een brute-force aanval op DES is dan ook de snelst bekende aanval.
In 1993 kostte een machine die DES (*brute-force*) kon kraken in gemiddeld 3.5 uur 1 miljoen USD, en de kosten hiervoor worden elke 5 jaar 10 keer zo klein. [3 (p.300)] Bovendien was de verhouding tussen prijs en snelheid van zo'n machine lineair [3 (p.153)].
Dus nu zou zo'n machine minder dan 10.000 USD kosten, en gezien Moore's Wet[3] zelfs slechts 2.500 USD.

[3] Naar een opmerking van Gordon E. Moore in een artikel in Electronics Magazine, 19 April 1965: De kosten van Integrated Circuits halveren elke twee jaar. Effectief wordt dezelfde computer dus elke twee jaar de helft goedkoper.

3.3 DES of 3-DES

Voor de veiligheid van de versleutelingen is het van belang te weten welke versleuteling er wordt gebruikt.

3.3.1 Het FO

Het Functioneel Ontwerp vertelt ons dat er voor de MAC bij RIESKOA gebruik wordt gemaakt van 3-DES [2 (p. 141)]. Inspectie van de gebruikte DES-functie uit des.js leert ons dat deze in staat is tot zowel *single* als *triple* DES. De gebruikte modus wordt bepaald aan de hand van de sleutellengte: als de functie een sleutel krijgt van (minimaal) 24 karakters, wordt er 3-DES gebruikt, anders 'gewoon' single DES.

Verder weten we ook dat ReSPID = MDC[DESmac_{Kp}{(ElID//ExtParGp)}], Kp is de persoonlijke DES-*key* van elke stemmer, DESmac = "last 8-byte block of ANSI-X9.9 mac calculation with 3DES" en 3DES = "Triple DES […] 2x56'key encryption with DES" [2 (p.141-142)].

3.3.2 De javascript bestanden

Als we kijken naar a010.js, zien we daar de volgende regels staan (hier wordt de ResPID berekend):

```
abKp=hexStringToData(convert34ANtoHex(sVPID1.substring(0,7))+
        convert34ANtoHex(sVPID2.substring(0,7)));

// -- calculate ReSPID --
abMac=generateMAC( abKp, pad(sElID+sExtParGp, 16), abICV);
abMDC=mdc2(abMac);
sReSPID=dataToHexString(abMDC);
```

`sVPID1` en `sVPID2` bevatten de Stemcode-1 resp. Stemcode-2, die ieder bestaan uit 7 karakters 34AN [2 (p. 142)].

```
function convert34ANtoHex( x34AN )
{
    var xHex;
    xHex=decimalToAnother(
                        toDecimal(x34AN)
                         ,16);
    if( xHex.length%2>0)
        xHex="0"+xHex;
    while(xHex.length<8)
    {
        xHex="0"+xHex;
    }
    return xHex;
}
```

Deze functiedefinitie laat zien dat `convert34ANtoHex` gebruikt maakt van andere functies voor de actuele conversie van een 34AN-string naar een hex-string. De functie zorgt er wel

voor dat de opgeleverde hex-string (`xHex`) een even aantal van minimaal 8 karakters bevat, door indien nodig de hex-string links (*most significant side*) te *padden* met "0"-karakters (niet NULL-karakters).

De functie `toDecimal` ziet er als volgt uit:

```
function toDecimal(N)
{
    var iLength=N.length;
    var iPos=0;
    var xResult = 0;
    var xPos;

    for( ;iLength>0;iLength--)
    {
        xPos = lookupPos( N.charAt(iPos));
        if( (iLength-1)>0)
        {
            xResult=xResult+(xPos*(Math.pow(34,iLength-1)));
        }
        else
            xResult=xResult+xPos;

        iPos=iPos+1;

    }
    return xResult;
}
```

Gegeven dat `N` een 34AN-string is van 7 karakters, wordt deze geconverteerd naar een natuurlijk getal volgens het volgende principe:

Gegeven een 34AN-string *a* van *n* karakters. We indexeren de karakters van links naar rechts, waarbij het meest significante karakter index 0 krijgt en het minst significante karakter index *n*-1. Met *a*(i) bedoelen we het 34AN karakter van *a* met index i, dus *a*(0) is het meest linkse (meest significante) karakter en *a*(*n*-1) het meest rechtse. Hierbij staat[*a*(i)] voor het natuurlijke getal gerepresenteerd door het karakter van *a* op index i: ["0"] = 0, ["1"] = 1, …, ["a"] = 10, …, ["k"] = 20, ["m"] = 21, ["n"] = 22, ["p"] = 23, …, ["z"] = 33.

Nu is het natuurlijk getal gerepresenteerd door *a* gelijk aan

$$[a(0)]*34^{(n-1)} + [a(1)]*34^{(n-2)} + \ldots + [a(n-1)]*34^{0}$$

Op deze manier wordt uit N een getal berekend met een waarde tussen 0 en 52,523,350,143 ($= 34^7 - 1 = 33*34^6 + 33*34^5 + \ldots + 33$).
Dit getal wordt vervolgens als invoer gegeven aan de functie `decimalToAnother`, samen met het getal 16 als radix:

```
function decimalToAnother(xN,xRadix)
{
    var xS="";
    var xA=xN;

    while(xA>=xRadix)
    {
```

```
        var xB=xA%xRadix;
        xA=Math.floor(xA/xRadix);
        xS+=xTokens[xB];
    }
    xS+=xTokens[xA];
    return transpose(xS);
}
```

Het resultaat hiervan is dat `decimalToAnother` als resultaat de hex-string teruggeeft die het hierboven berekende getal representeert. Bovenstaande min/max waardes vertalen naar resp de strings “0” en “C3AA2B87F”. Met andere woorden, decimalToAnother geeft een hex-string terug waarvan het aantal karakters tussen de 1 en de 9 ligt (en waarvan de decimale waarde tussen 0 en 52,523,350,143 ligt). `convert34ANtoHex` geeft dus, gegeven een 34AN-string van 7 karakters als invoer, uiteindelijk een hex-string terug waarvan het aantal karakters 8 of 10 is en waarvan de decimale representatie tussen 0 en 52,523,350,143 (inclusief) ligt.

3.3.3 De berekening van de sleutel

Kijken we dan terug naar de volgende regel uit a010.js,

```
abKp=hexStringToData(convert34ANtoHex(sVPID1.substring(0,7))+
        convert34ANtoHex(sVPID2.substring(0,7)));
```

dan zien we dat Kp, de sleutel die gebruikt wordt bij de DESmac berekening van de ResPID, het resultaat is van `hexStringToData`, die als invoer de concatenatie krijgt van twee van zulke hex-strings. Met andere woorden, de invoer voor `hexStringToData` is een hex-string van 16, 18 of 20 karakters (de grenzen van de decimale waarde van deze hex-string zijn voor de volgende berekeningen niet van belang, maar kunnen vrij simpel berekend worden).

```
function hexStringToData (xHexString)
{
    var xDataString = "";
    for (var i= (xHexString.substr(0, 2)=="0x")?2:0;
i<xHexString.length; i+=2)
    {
        xDataString += String.fromCharCode (parseInt
(xHexString.substr (i, 2), 16));
    }
    return xDataString;
}
```

Nadere bestudering van deze functie leert ons dat het achterliggende algoritme redelijk simpel is:

- Verdeel de hex-string in groepjes van twee hex-karakters
- Bereken bij elk groepje de bijbehorende decimale waarde (een getal van 0 t/m 255)
- Om de datastring te berekenen wordt deze waarde opgevat als ascii-code, en wordt het karakter behorende bij deze ascii-code in de datastring gezet.

De resulterende datastring bevat exact half zoveel karakters als de hex-string, dus gegeven een invoer van 16, 18 of 20 karakters levert `hexStringToData` een datastring op van 8, 9, of 10 ascii-karakters.

`abKp`, de *key* voor de DESmac-functie, is dus uiteindelijk een ascii-string van 8, 9 of 10 karakters. Deze wordt gebruikt in de volgende regel in a010.js:

```
abMac=generateMAC( abKp, pad(sElID+sExtParGp, 16), abICV);
```

Kijken we naar `generateMAC`,

```
function generateMAC(xKey, xDataIn, xICV)
{
    var xMAC;
    var xData;
    var xCiphertext;
    //xData = pad( xDataIn, 16);
    xData=xDataIn; //no padding
    xCiphertext = des (xKey, xData, 1, 1, xICV); // encrypt
cbc
    xMAC = xCiphertext.substring(xCiphertext.length-8,
xCiphertext.length);
    return xMAC;
}
```

dan zien we dat `abKp` als `xKey` direct wordt doorgegeven aan de des-functie. Dus deze functie ontvangt als *key* een ascii-string van 8 tot 10 karakters. Maar eerder hadden we al gezien dat deze functie slechts 3-DES toepast als hij een sleutel krijgt van minimaal 24 karakters. Er wordt dus ‘Single DES’ toegepast bij de berekening van de MAC, in plaats van ‘Triple DES’ zoals het FO ons doet geloven. Aangezien bij de berekening van de technische stem ook DESmac wordt toegepast, met dezelfde Kp [2 (p. 142)], nemen we aan dat de berekening hiervan op analoge wijze geschiedt, en dat de technische stem dus ook niet met Triple-DES wordt versleuteld, maar met Single-DES.

4 Aanvallen

Nu er hierboven al verschillende veiligheidsaspecten zijn onderzocht is het natuurlijk van belang in hoeverre de veiligheid van RIESKOA gewaarborgd is. Hiertoe heb ik de haalbaarheid van drie mogelijke aanvallen op RIESKOA onderzocht, de drie aanvallen die mij gezien de gebruikte protocollen en versleutelingen het meest waarschijnlijk leken.

4.1 Mogelijke Aanvallen

Van de volgende aanvallen wordt de mogelijkheid en effectiviteit m.b.t. RIESKOA onderzocht:

- Achterhalen van de stem van een kiezer
  - Bijvoorbeeld door het onderscheppen van de technische stem.
- Achterhalen van de (geheime) sleutel van een kiezer
  - Door een *known-plaintext-attack* uit te voeren op onderschepte gegevens.
  - Als dit lukt dan kan de aanvaller stemmen fabriceren alsof deze door de kiezer gedaan zijn.
- Een (D)DoS-aanval
  - De stemservice overbelasten met stemmen, waardoor de servers of de uitslag gecompromitteerd raken.
  - Als dit lukt dan is de *availability* van de stemservice in het geding.

4.2 Het onderscheppen van informatie verzonden naar de stemserver

De computer die wordt gebruikt door de kiezer communiceert uitgebreid met de stemserver. Omdat de verbinding tussen de kiezer en de stemserver wordt versleuteld via SSL, het zogenoemde *https*-protocol, is het normaal gesproken onmogelijk om deze communicatie te onderscheppen.
Deze situatie verandert zodra de kiezer gebruikt maakt van een onveilige computer. Met onveilig wordt hier bedoeld, een computer waarop programmatuur is geïnstalleerd waarvan de kiezer niet op de hoogte is. In [5] wordt gedetailleerd uit de doeken gedaan hoe, gegeven gebruikerstoegang tot een computer, hierop programmatuur geïnstalleerd kan worden die ervoor zorgt dat deze computer verbinding maakt met een andere server dan de bedoelde. Dat dit gebeurt is te detecteren, maar de gemiddelde gebruiker zal hiervan niet op de hoogte zijn, noch de moeite nemen dit te onderzoeken.
In dit hoofdstuk wordt er, waar van toepassing, van uitgegaan dat een kwaadwillende partij op een soortgelijke manier een *man-in-the-middle* aanval uitvoert, door de computer die gebruikt wordt door de kiezer in werkelijkheid verbinding te laten maken met zijn eigen server, en zelf een verbinding te leggen met de stemserver. Alle informatie die de kiezer verstuurt, wordt door deze partij opgevangen. Die kan dit vervolgens bekijken, opslaan, bewerken, etc., en doorsturen naar de stemserver. Als deze de informatie die hij ontvangt van de stemserver ook weer terugstuurt naar de kiezer heeft geen van beide partijen iets door, maar de kwaadwillende partij beschikt wel over alle gecommuniceerde gegevens.

4.3 Het achterhalen van de stem van een kiezer

In een van de minst destructieve toepassingen van een *man-in-the-middle* aanval bewaart de kwaadwillende partij (hier verder Eve genoemd) slechts de gegevens. Bij het versturen van de stem wordt simpelweg de technische stem verzonden. Voor Eve is het relatief eenvoudig om aan de hand hiervan uit te rekenen wat er gestemd is; ook al beschikt zij niet over de persoonlijke stemcode, iedereen kan de MDC-hash uitrekenen van een technische stem. De MDC-hash van het linkerdeel verwijst naar de virtuele identiteit van de kiezer die de technische stem heeft uitgebracht, en de MDC-hash van het rechterdeel verwijst naar de kandidaatskeuze. Deze hashes zijn opgenomen in het pre-election-bestand dat wordt gepubliceerd. Hierin wordt de koppeling gelegd tussen de hashes en de kandidaten. M.a.w., als men van een bepaalde kiezer de technische stem (en dus de hashes) kan achterhalen, kan men hiervan ook bepalen op welke kandidaat deze kiezer heeft gestemd.

4.4 Achterhalen van de (geheime) sleutel van een kiezer

Maar met de technische stem kan Eve nog veel verder gaan. De technische stem, die van de computer van de kiezer naar de stemserver wordt verzonden, is namelijk niets anders dan de versleuteling van de stem van de kiezer. Ze heeft dan dus zowel de *plaintext* (onversleutelde tekst) van een stem in handen (de kandidaatskeuze, achterhaald m.b.v. de bovenstaande methode) als de *ciphertext* (versleutelde tekst) van diezelfde stem (de technische stem). Theoretisch is het hiermee mogelijk de gebruikte sleutel te achterhalen. Als dit Eve zou lukken, kan ze zelf technische stemmen genereren alsof deze van de kiezer afkomstig zijn. Deze techniek staat bekend als een *known-plaintext-attack*. Typisch worden hierbij met elke mogelijke sleutel versleutelingen van de onversleutelde tekst gegenereerd totdat dit de juiste versleuteling oplevert.
Aangezien RIESKOA openbaar is, is de gebruikte methode van versleuteling te achterhalen. Uit het Functioneel Ontwerp blijkt dat de technische stem bestaat uit een linker- en rechterdeel, en dat beide delen feitelijk een MAC[4] zijn van de bijbehorende *plaintext*. Helaas zijn de bestanden waarin de technische stem zelf wordt berekend niet tot onze beschikking. Wel kunnen we kijken naar de berekening van de ReSPID, de virtuele identiteit van de kiezer zoals die gebruikt wordt op de stemserver tijdens het stemmen [2 p.(142)], die volgens het FO op dezelfde manier wordt berekend als de technische stem, met als enige verschil de gebruikte *plaintext* [2 (p. 141)]. Alle hieronder genoemde .js-bestanden zijn bestanden met javascript-code die daadwerkelijk gebruikt werden voor RIESKOA.

[4] Message Authentication Code, een code die cryptografisch berekend is uit de *plaintext* en met behulp waarvan de integriteit van de plaintext kan worden geverifieerd.

4.4.1 Known-plaintext-attack

Doordat er 'slechts' DES wordt gebruikt (ipv 3-DES) is de kans dat de versleuteling gekraakt kan worden toegenomen. Hieronder wordt onderzocht óf en hoe de versleuteling kan worden gekraakt.

Opnieuw uit a010.js:

```
abMac=generateMAC( abKp, pad(sElID+sExtParGp, 16), abICV);
abMDC=mdc2(abMac);
sReSPID=dataToHexString(abMDC);
```

En uit utils.js:

```
function pad( xData, xLength)
{
    var xPad = 0;
    var xBuffer =xData;
    var xPadChar=0x00;
    if ((xData.length % xLength) > 0)
        xPad = xLength - (xData.length % xLength);

    // check to see if we must pad
    if (xPad != 0)
    {
        var i = xData.length;
        // pad with 0x00
        for (; i < (xData.length + xPad); i++)
        {
            xBuffer += String.fromCharCode(xPadChar);
        }
    }
    return xBuffer;
}
```

Dit toont dat ElID//ExtParGp vóór de versleuteling wordt uitgebreid (*padded*) tot (een veelvoud van) 16 bytes middels het rechts toevoegen van 0-bytes (hex: 0x00).

We wisten al dat ReSPID = MDC[DESmac$_{Kp}$ {(ElID//ExtParGp)}]. [2 (p. 141)] leert ons dat:
ElID = 2 bytes
ExtParGp = 2 pos 34AN

Met andere woorden, ElID//ExtParGp = 4 bytes, waarvan de laatste twee ieder slechts 34 verschillende waardes aannemen. (Om precies te zijn, waarden uit het bereik {48 … 57, 97 … 107, 109, 110, 112 … 122}).
Dat geeft ons voor ElID//ExtParGp, 256*256*34*34 = 75 759 616 verschillende mogelijkheden. Maar hier zijn we niet zo heel erg in geïnteresseerd, want deze waardes zijn openbaar. Het gaat erom, kunnen we, gegeven een ElID//ExtParGp, de sleutel achterhalen (*known-plaintext attack*).
Om dit te onderzoeken, maken we een testomgeving (geschreven in Java) waarbij geprobeerd wordt uit een bekende ElID//ExtParGp en de bijbehorende ReSPID, de gebruikte Kp te destilleren via een brute-force aanval.

4.4.2 Het kraken van DES

De vraag: Is het, met een algemeen beschikbare computer, mogelijk om, gegeven een 16-byte *plaintext* (bestaande uit 4 bytes informatie en 12 bytes padding) en een 8-byte MAC (de laatste 8 bytes van de *ciphertext* behorende bij de *plaintext*), binnen enkele dagen de gebruikte sleutel te achterhalen?
Om dit uit te zoeken heb ik een testprogramma geschreven in Java, zie de bijlage, waarin *brute force* alle mogelijke sleutels worden gegenereerd en geprobeerd.

Ik heb een *plaintext* en een sleutel (0x0000000000000069) gekozen. Daarmee heb ik de *ciphertext* bij die *plaintext* berekend. Deze *plaintext* en *ciphertext* heb ik aan mijn programma gevoerd, welke brute-force alle sleutels afgaat en de juiste sleutel probeert te vinden. Elke matchende sleutel wordt afgedrukt.
Omdat DES-sleutels eigenlijk 56 bits lang zijn, maar mijn versie net als de meeste implementaties 64 bits (8 bytes) gebruikt en van elke byte het minst significante bitje gebruikt als parity bit zijn er meerdere sleutels die voldoen. De enige verschillen tussen deze sleutels zijn de parity bits.

Op 04-05-2007 om 18:37:21 was dit programma begonnen, op 10-05-2007 om 18:22:19 had het sleutel 0x101010169 gevonden.
0x101010169 is 4.311.810.409 decimaal, dus in (bijna) zes dagen heeft het programma 4.311.810.410 sleutels geprobeerd. Dat lijkt op het eerste gezicht veel, maar aangezien er 256^8 oftewel 18.446.744.073.709.551.616 sleutels zijn valt dit wel tegen.
18.446.744.073.709.551.616 / 4.311.810.410 = 4.278.189.975,8. Maal zes dagen is dit 25.669.139.854,8 dagen.

Dit betekent dat het op de testcomputer (ongeveer 4 jaar oud, AMD 1600 Mhz) meer dan 25 miljard dagen zou kosten om met het geschreven programma alle sleutels te proberen.
Laten we een factor 2 introduceren omdat het een java-implementatie betreft, en deze real-time geïnterpreteerd worden wat extra tijd kost. Nog een factor 2 omdat de implementatie waarschijnlijk veel efficiënter kan. Zelfs met inachtneming van de vele heuristieken die er ongetwijfeld nog toe te passen zijn op dit programma lijkt het onmogelijk om zonder gespecialiseerde hardware DES binnen een paar dagen te kraken.

4.5 Een (D)DoS-aanval

(D)DoS staat voor (Distributed) Denial of Service, een type aanval die meer gericht is op het uitschakelen van een doelwit dan op het stelen van informatie. Hierdoor kan het doelwit geen service meer verlenen, *Denial of Service*. De gedistribueerde versie is er eentje waarbij de aanvaller van vele andere computers gebruik maakt, zonder medeweten of toestemming van de eigenaars, om deze aanval op nog veel grotere schaal en nog veel anoniemer uit te voeren [7 p.(778)].
In deze wordt specifiek gekeken of het mogelijk is de stemservers te overbelasten met valse gegevens. Hiervoor zijn een aantal gegevens van belang:

1. De stemservers zijn beschermd met een *firewall* die selecteert op vorm; de aanval moet dus de vorm aannemen van een legitiem stemproces hierdoor te komen [2 (p. 131)].
2. Een legitiem stemproces bestaat feitelijk uit het aanvragen van bestanden van de webserver met een URL waarin de gegevens van eventuele eerdere stappen uit het stemproces zitten verwerkt.
3. Door het namaken van deze URL aanvragen in de juiste vorm kunnen de meest intensieve stappen van het stemproces worden aangeroepen, waar met 'intensief' wordt bedoeld die stappen waarin het meeste werk wordt verzet door de stemserver voor de aanvraag als ongeldig wordt bestempeld.
4. Misschien is het zelfs mogelijk om aanvragen zo te fabriceren dat ze in eerste instantie niet als ongeldig worden bestempeld, en dus in het *received_votes*-bestand terechtkomen, wat de server op andere manieren kan belasten. Bovendien is de mogelijkheid dan aanwezig dat er per ongeluk een 'geldige' stem is gefabriceerd die de stemuitslag compromitteert.

4.5.1 De webpagina's

Om te weten hoe de aanvragen precies eruit zien wordt er gekeken naar de HTML- en JavaScript-bestanden. Ten eerste kijken we naar stemcode.html, het bestand waar de gebruiker zijn stemcodes invult. Hierin zien we onder andere de volgende regel:

```
<form name="xform" action="server" method="POST"
autocomplete="off">
```

Dit wordt gevolgd door de specificatie van het formulier. Hieruit volgt dat de informatie wordt verzonden als een POST-request, maar de actionhandler is 'server'. Hoewel dit een specifiek *keyword* lijkt, wordt dit hoogstwaarschijnlijk opgevat als een relatieve URL, wat betekent dat deze wordt omgeschreven naar een absolute URL, en vervolgens door de server wordt geïnterpreteerd.
Gegeven een URL van 'www.internetstembureau.nl/stemcode.html' zou de actionhandler 'www.internetstembureau.nl/server' worden. Voor een http-server is het vrij gemakkelijk om dit om te schrijven naar een interne locatie waar zich de daadwerkelijke actionhandler bevind.

Ook in deze pagina zien we de regel:

```
<div id="button4"><A class="button_active"
href="javascript:doNext();" tabindex=6>Verder ></A></div>
```

Dit beschrijft wat er moet gebeuren nadat er op de ‘Verder’ knop is gedrukt, nl. dan wordt de JavaScript-functie doNext() aangeroepen. Om te weten wat er dan gebeurt kijken we in de JavaScript-bestanden die bij dit HTML-bestand horen, en zien in a010.js:

```
/**
 * Function : doNext
 * Description : This function will process the form and load
the next page a012
 */
function doNext()
{
    if( processform() )
    {
        document.xform.actionreq.value="next";
        document.xform.submit();
   }
}
```

Hieruit blijkt dat de functie doNext() niet veel meer doet dan een andere functie aanroepen, processform(), die blijkbaar als doel heeft om het formulier te verwerken. Als dit goed gaat wordt het formulier door de browser omgezet in een zgn. HTTP “post” transactie waarbij het formulier gecodeerd in de *body* van de transactie wordt ingevoegd.
Dit soort gedrag is allemaal te repliceren door een eigen formulier te maken en op de juiste waardes in te stellen. Wat nu belangrijk is, is wat er in processform() gebeurt. Dit blijkt een vrij grote functie te zijn, waarin voornamelijk wordt gecontroleerd of de invoer wel van de juiste vorm is. Ook wordt er de ReSPID berekend, en deze wordt teruggegeven aan de server via het formulier:

```
//get the session data
    sSessionData=document.xform.sessiondata.value;
    //parse the sessiondata
    oSessionData=parseSessionData(sSessionData);
    setSessionParam(oSessionData, 'respid', sReSPID);
    sSessionData=toSessionData(oSessionData);

    // -- reset the form, reset all data --

    document.xform.reset();
    document.xform.abelpi.value=sAbelPI;
    document.xform.sessiondata.value=sSessionData;
    return true;
```

Wat hier gebeurt is dat de ‘oude’ *session data*, die als *string* was opgeslagen in een zgn. verborgen veld van het formulier, wordt opgeslagen, geparseerd volgens `parseSessionData(sSessionData)`, afkomstig uit SessionData.js, en hieraan wordt het *key-value-pair* ‘respid’ met de berekende ReSPID toegevoegd. Vervolgens wordt de *session data* weer omgezet naar een *string*. Dan wordt het formulier gewist, de abelpi-parameter wordt gezet (op een standaardwaarde aangezien deze functionaliteit niet in RIESKOA aanwezig is) en de *session data* wordt weer in hetzelfde verborgen veld gezet, waarna de functie *true* teruggeeft om aan te geven dat alles goed ging.

De betekenis van dit alles is dat het vrij gemakkelijk is om eigen stemmen in te sturen. Men maakt gewoon een webpagina met een formulier met daarop twee verborgen velden, abelpi en sessiondata genaamd, en vult hierop wat waardes in.

Een voorbeeld, uitgaande van 'www.internetstembureau.nl' als basis URL van de stemserver:

```
<html>
<head>
</head>
<body>
<form name=”xform”
action=”http://www.internetstembureau.nl/server” method=”POST”
autocomplete=”off”>
<input type="hidden" name="sessiondata" value="0"/>
<input type="hidden" name="abelpi" value="0"/>
</form>
</html>
```

Aangezien de ReSPID door de server wordt gebruikt om te controleren of de gebruiker mag stemmen, zou de server de HTTP "post" transactie serieus moeten nemen om erachter te komen dat de sessiondata niet van de juiste vorm is, op die manier werk verzettend voor iets onbelangrijks. Door de sessiondata, of zelfs het hele formulier, zo goed mogelijk te vervalsen kan de server zo lang mogelijk voor de gek worden gehouden en dus zo lang mogelijk zo veel mogelijk werk verzetten.
Een blik op stemcontrole.html en a010.js leert ons de beginwaarde voor sessiondata en dat er vóór het insturen nog een extra veld op waarde wordt gezet. Als we dit toevoegen komen we op:

```
<html>
<head>
</head>
<body>
<form name=”xform”
action=”http://www.internetstembureau.nl/server” method=”POST”
autocomplete=”off”>
<input type="hidden" name="sessiondata" value="
aWdub3Jlc3RhdHVzPWZhbHNlJm1vZGU9bm9ybWFs"/>
<input type="hidden" name="abelpi” value="0"/>
<input type=”hidden” name=”actionreq” value=”next”/>
</form>
</html>
```

Voor de zekerheid kunnen we alle velden die in het officiële formulier bestaan kopiëren en ze de waardes geven die ze vlak voor het insturen zouden moeten hebben (voor zover bekend). Deze stap is hier omwille van de leesbaarheid niet weergegeven, maar dit zou de auteur weinig moeite kosten.
Hiermee zijn alle oppervlakkige testen omzeild, en de enige manier waarop de server nu kan deduceren dat het formulier niet van een echte stemmer afkwam is door de sessiondata te parseren en de respid te bemachtigen (die niet gecodeerd is in de sessiondata). Willen we ook deze stap bemoeilijken, dan moet de respid dus vervalst worden.

Om de respid te vervalsen moeten we kijken naar hoe deze eruitziet. Hiervoor kijken we wederom naar a010.js en zien daar de volgende drie regels waar de respid wordt berekent:

```
abMac=generateMAC( abKp, pad(sElID+sExtParGp, 16), abICV);
abMDC=mdc2(abMac);
sReSPID=dataToHexString(abMDC);
```

Hoewel we wel beschikken over de ElID en ExtParGp (die zitten gecodeerd in de standaardwaarde voor de sessiondata) moeten we in afwezigheid van een sleutel, die berekend wordt uit de stemcode, afzien van het berekenen van een respid. Echter, de bovenstaande regels (in het stukje "DES of 3-DES" al behandeld) leren ons dat een ReSPID benaderd kan worden door `mdc2()` en `dataToHexString()` toe te passen op een 8-bytes *string*. (In essentie vervangen we in de regels hierboven 'abMac' door een zelfverzonnen waarde van de juiste vorm, en voeren de rest van de berekening gewoon uit). Dit resulteert in een waarde sReSPID van de juiste vorm die we via

```
sSessionData=document.xform.sessiondata.value;
oSessionData=parseSessionData(sSessionData);
setSessionParam(oSessionData, 'respid', sReSPID);
sSessionData=toSessionData(oSessionData);
document.xform.reset();
document.xform.abelpi.value=sAbelPI;
document.xform.sessiondata.value=sSessionData;
```

kunnen toevoegen aan de sessiondata en aan het formulier. Zie bijlage B voor een uitwerking van de resulterende webpagina.

Om de stemserver nog meer extra werk te kunnen bezorgen is het handig om een geldige ReSPID, d.w.z. een ReSPID die behoort bij een uitgegeven set stembescheiden, te gebruiken. Doordat deze geheim en persoonlijk zijn kan dit problemen opwerpen.
Bij gebrek hieraan kan ook elke keer een verschillende `abMac` worden gebruikt als basis voor de berekening van de valse ReSPID, om de kans dat er een geldige waarde wordt ingezonden te vergroten.

Deze vorm van het insturen van valse ReSPID's, effectief het inloggen met een valse stemcode, is al een manier om de stemserver extra werk te bezorgen. Willen we ook nog valse stemmen toevoegen moet er wat meer werk gebeuren.
[2 (p. 63)] leert ons dat in de webpagina's a020 en a025 de lijst resp. de kandidaat wordt gekozen. A030 is de pagina waarin de keuze wordt getoond en wordt gevraagd om deze te bevestigen. Uit de bijbehorende HTML- en JavaScript bestanden blijkt dat hierin de keuze wordt gecodeerd en toegevoegd aan de sessiondata. Het echte versleutelen (berekenen van de technische stem) gebeurt niet in deze pagina's, wat doet vermoeden dat dit gebeurt in a040. Dit vermoeden wordt ondersteund door [2 (p.63)].
Op dit moment beschikken we helaas niet over deze pagina en dus over de informatie hoe dit precies gebeurt en hoe de technische stem wordt toegevoegd aan de sessiondata. Ten tijde van de verkiezingen is deze pagina als het goed is wel openbaar. De informatie hieruit zou dan kunnen worden gebruikt om technische stemmen te vervalsen en zo te proberen valse stemmen in te sturen op een manier analoog aan de bovenstaande methode.
Dit zou twee doelen dienen: als eerste de server te overbelasten, en als tweede doel de hoop dat een paar valse stemmen toevallig als legitieme stemmen worden herkend en zo de verkiezingsuitslag te beïnvloeden.

4.5.2 De aanval

Het doel van dit document is natuurlijk niet om een handleiding te geven hoe men de verkiezing kan dwarsbomen, het doel is om te onderzoeken hoe bestendig de stemserver(s) zijn. We hebben laten zien dat de informatie zoals een legitieme kiezer die zou insturen vrij

gemakkelijk vervalst kan worden. Ervan uitgaande dat deze ‘valse stemmen’ er uiteindelijk uitgepikt worden door de stemserver, leveren deze wel een extra belasting op maar hebben geen gevolgen voor de uitslag.
Helaas is er geen informatie beschikbaar over de robuustheid van de gebruikte stemservers. Er van uit gaan dat deze bestand zijn tegen de gegenereerde transacties vanaf één computer is niet zo’n grote veronderstelling, de servers moeten tenslotte meer dan honderdduizend kiezers aankunnen. Weliswaar over een paar dagen, maar de piekbelasting daarvan kan ook hoog oplopen.

De mogelijkheid die eventuele aanvallers dan nog hebben is het uitvoeren van een gedistribueerde aanval. Hierbij worden er meerdere computers, soms wel duizenden tegelijk, gebruikt om de aanval uit te voeren. Dit vergroot de belasting van de server aanzienlijk. Echter, om controle over deze duizenden computers te verkrijgen moeten de aanvallers virussen en ander gespuis gebruiken om deze computers over te nemen, of in ieder geval op eigen houtje transacties met het internet stembureau uit te laten voeren.

4.5.3 De verdediging

Heel veel mogelijkheden zijn er niet om je te beschermen tegen een (D)DoS-aanval. Het hele punt van servers is dat ze beschikbaar moeten zijn voor de juiste verzoeken, en (D)DoS-aanvallen kunnen zich dus ‘vermommen’ als een op het eerste gezicht legitiem verzoek. Dat een nader onderzoek al snel uitwijst dat het verzoek malefide is, is niet erg; de schade is al gedaan. De server heeft tijd en rekenkracht besteed aan het verzoek, en dat was het doel. Hoe meer tijd en rekenkracht hoe liever; als men dan maar genoeg verzoeken stuurt is elke server ‘plat’ te krijgen.
Toch kunnen er wel enkele maatregelen worden getroffen. Het aantal geaccepteerde verzoeken binnen een bepaalde tijd van eenzelfde ip-adres kan beperkt worden, om te voorkomen dat een enkele computer in korte tijd zeer veel verzoeken kan sturen.
Binnen RIESKOA specifiek kunnen zgn. PHP-sessions gebruikt worden om een cliënt te dwingen het hele proces te doorlopen ipv halverwege in te springen (zoals in het voorbeeld). Een slimme aanvaller weet dit proces wel te automatiseren, maar het is een klein extra laagje van moeite die door de aanvaller moet worden gedaan, zonder extra moeite te vragen van legitieme cliënten.
Maar uiteindelijk is de beste manier om je tegen een (D)DoS-aanval te beschermen ook de moeilijkste: malefide transacties eerder herkennen en dumpen. Hoe dit te doen is op zich zelf een scriptie waard.

5 Stemcontrole

5.1 Inleiding

Als onderdeel van deze bachelorscriptie heb ik het zgn. stemcontrole systeem gemaakt. Met dit systeem konden internetstemmers voor de Tweede Kamerverkiezingen m.b.v. hun technische stem controleren of hun stem goed was verwerkt.

5.2 Achtergrond

Zowel door het telsysteem van TTPI als door het telsysteem van de SOS-groep wordt een referentiebestand ('performed_votes'bestand) opgeleverd waarin alle ontvangen stemmen vermeld staan, of ze geaccepteerd zijn en zo ja voor welke kandidaat ze zijn meegeteld, zo nee waarom niet.
Het referentiebestand is in *plain-text*, en elke regel bevat de informatie van exact één ontvangen stem. Voor een exacte beschrijving van het *format*, zie [2 p.112-113].
Elke keer dat een internetstemmer via internet een stem uitbracht, kregen ze een technische stem te zien. Deze bestond uit twee delen, namelijk de VOTER_ID en een versleuteling van de CANDIDATE_ID.
In het 'performed_votes'bestand staat van elke ontvangen stem zowel deze technische stem, als de MDC-hash van de twee delen vermeld. Dit samen geeft genoeg informatie om een stem te kunnen controleren.

5.3 Ontwerp

Het basisontwerp van deze service was niet moeilijk. Sla het 'performed_votes'bestand ergens op, maak een webpagina waar een gebruiker zijn technische stem invoert, en na een "*push of the button*" gebeurt er iets magisch waardoor de juiste stem (of stemmen) wordt opgezocht in het bestand en de informatie wordt getoond.
In overleg met mijn scriptiebegeleider werd er al snel besloten tot de volgende opzet:

1. Een gedeelte van de informatie van het 'performed_votes'bestand werd in een MySQL-database opgeslagen.
2. Er komt een webpagina, geschreven in php, waarin de gebruiker zijn technische stem invoert. Deze webpagina zal *client-side* de technische stem hashen, en de gehashte stem naar de server sturen om hem te laten vergelijken met de stemmen in de database.
3. Voor elke gevonden matchende stem in de database zal de informatie van deze stem worden getoond.
4. De communicatie tussen gebruiker en webserver zou via SSL versleuteld worden, om zo de kans nog extra te verkleinen dat kwaadwillenden achter de stem van de gebruiker kunnen komen.

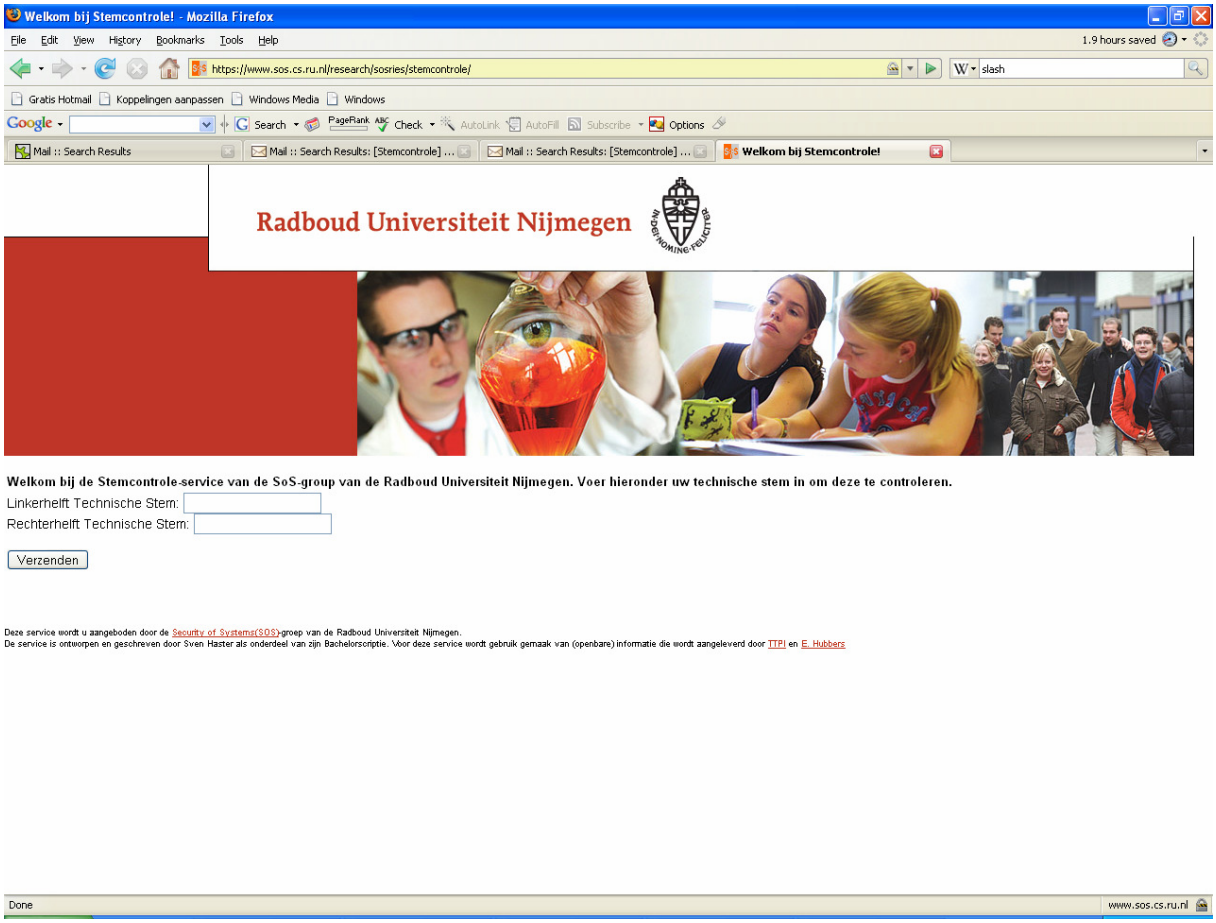
5.4 Implementatie

Na het regelen van ruimte op een webserver en MySQL-database, schreef ik de eerste php-bestanden waarmee de gebruiker zijn technische stem kon invoeren.
Het *hashen* bleek nog enigszins lastig. Om zeker te weten dat mijn versleuteling volledig compatibel was met die van TTPI, besloten we de JavaScript bestanden van TTPI te gebruiken, omdat hierin de functies staan met behulp waarvan de officiële website de MDC-hash berekent.
Doordat de documentatie van deze functies niets voorstelde duurde het even voordat ik precies had uitgevogeld hoe ze te gebruiken, maar na behoorlijk wat *trial-and-error* kwam ik eruit.
Nu had ik een systeem waarin een stem kon worden ingevoerd, versleuteld, opgezocht in de database en vervolgens getoond. Helaas was de database nog akelig leeg, afgezien van een aantal teststemmen die ik erin had gezet.
De volgende stap was dus om de informatie uit het 'performed_votes'bestand in de database te krijgen. Het ging hierbij om specifieke informatie:

- de MDC-hash van zowel het linker- als het rechterdeel van de technische stem
- of de stem was meegeteld of niet
- de kandidaatscode van de bijbehorende kandidaat
- de reden waarom de stem was afgekeurd

Aangezien het bestand meer informatie bevatte, en om het algehele proces te vereenvoudigen, besloot ik hiervoor in C++ een programmaatje te schrijven dat, gegeven een 'performed_votes'bestand, dit parseert en vervolgens de gewenste informatie per stem in een MySQL-query op te nemen. Al deze queries werden vervolgens weggeschreven naar een ander bestand. Nu hoefde men alleen maar dit bestand als *source* aan de MySQL server te geven, en als deze server op de juiste database stond ingesteld worden de queries juist uitgevoerd en de informatie in de database opgenomen.
Omdat zowel TTPI als de SOS-groep met een 'performed_votes'bestand zou komen, kan men met een van de argumenten aangeven welk van de twee bestanden het betreft, en dit zorgt ervoor dat de queries betrekking hebben op een andere tabel. Er is dus uiteindelijk één tabel met stemmen zoals ze verwerkt (gekeurd) zijn door TTPI en één zoals ze verwerkt zijn door SOS.

Nu waren er twee databases met stemmen en een php-pagina waarop de stemmen vergeleken konden worden. Echter, om ook daadwerkelijk de gegevens van de kandidaten te kunnen tonen (in plaats van alleen hun identificatiecode) moest er nog een tabel bij komen waarin de informatie van de kandidaten werd opgeslagen.
Hiertoe werd het reeds gecreëerde programma uitgebreid zodat dit nu ook de STUF-K40 en STUF-K50 bestanden kon parseren en de gewenste informatie per kandidaat in een MYSQL-query wegschreef analoog aan hoe de 'performed_votes'bestanden werden geparseerd.
Later werd nog ontdekt dat de interface die gebruikt werd voor communicatie met de database moeite kon hebben met grote bestanden en dus werd het programma zo gewijzigd dat het meerdere bestanden van een vaste maximumlengte creëerde. Dit kon wel leiden tot meer werk voor de persoon die het systeem instelde, maar zorgde ervoor dat te grote bestanden geen struikelblok meer was.
De service werd nog zo uitgebreid dat er, gegeven een technische stem, niet alleen de gevonden ontvangen stemmen met die exacte technische stem werden getoond, maar ook alle andere stemmen van diezelfde identiteit.



Ten slotte werd er nog in een tabel bijgehouden door welke identiteiten en hoe vaak er een stem werd opgevraagd. Dit was puur en alleen voor de eigen interesse, om na te gaan hoe vaak er gebruik werd gemaakt van de controleservice.
In de eerste week na de verkiezingen hadden ongeveer 80 verschillende identiteiten een controle uitgevoerd. Op 16 mei 2007 was dit opgelopen tot 91 verschillende identiteiten en 105 controles.

6 Conclusie

RIESKOA is een simpel systeem, simpel in de zin van doorzichtig en helder. Hierdoor is het relatief gemakkelijk om te begrijpen hoe het werkt en wat er allemaal gebeurt als je je stem uitbrengt. Ook verkleint dit de kans op gaten in de beveiliging; zowel omdat bij een simpel systeem minder snel fouten worden gemaakt in ontwerp of uitvoering, alsmede omdat het voor derden gemakkelijker is om het systeem te controleren en eventuele fouten te melden. De simpelheid doet in dit geval echter niets af aan de kracht van het systeem. Door het goed gebruik van een paar simpele, maar krachtige protocollen en versleutelingen is RIESKOA een moeilijk penetreerbaar systeem. In dit document zijn drie waarschijnlijke aanvallen onderzocht en de enige hiervan met enige kans van slagen is een (D)DoS-aanval. Er zijn nog wat maatregelen die gebruikt kunnen worden om deze aanval moeilijker uitvoerbaar te maken. Voor aanvallen gericht op het achterhalen van de stem of het met goed gevolg uitbrengen van vervalste stemmen is bij juist gebruik van het systeem eigenlijk geen kans van slagen.
Een goed voorbeeld van de controleerbaarheid van RIESKOA is het stemcontrolesysteem en het telsysteem van de SOS-groep van de RU Nijmegen. Helaas heeft slechts 0.5% van de internetstemmers gebruik gemaakt van deze service om hun stem te controleren. Ervan uitgaande dat het aantal mensen dat hun stem handmatig heeft gecontroleerd net zo laag is, is de kans om door de legitieme stemmer ontdekt te worden bij het aanbrengen van wijzigingen in de uitslag laag.
RIES, en specifiek RIESKOA, is een geweldig systeem met vele mogelijkheden. Maar RIES bevat niet alleen maar positieve punten. De specifieke uitvoering van het systeem laat nog wel eens te wensen over. Vooral het Functioneel Ontwerp liet heel wat steken vallen, zoals ook uit deze scriptie blijkt. Beweringen uit het FO die niet overeenkomen met de daadwerkelijke implementatie, ontbrekende informatie, vage teksten, inconsistente naamgeving en zelfs regelrechte tegenspraak tussen twee verschillende delen van het FO; ze komen allemaal voor. Maar toch zou ik zelf RIES zonder meer gebruiken om te stemmen. De hier genoemde kritiek is nauwelijks van invloed op de veiligheid en betrouwbaarheid van het systeem zelf, en de huidige stand van zaken daarop is voor mij meer dan goed genoeg.

7 Referenties

[1] Engelbert Hubbers, Bart Jacobs, and Wolter Pieters. *RIES - Internet Voting in Action*. In: R. Bilof (ed.), *Proceedings of the 29th Annual International Computer Software and Applications Conference, COMPSAC'05*, p.417-424. IEEE Computer Society, 2005. (http://www.cs.ru.nl/B.Jacobs/PAPERS/compsac2005.pdf)

[2] Piet Maclaine Pont, Arnou. Hannink, Jacques Hoeienbos, Marco Rijkschroeff, Jacques. Schuurman, *RIES-KOA Functioneel Ontwerp*, 2006.

[3] Bruce Schneier, *Applied Cryptography*, Second Edition, 1996, John Wiley & Sons

[4] Herman Robers, *Electronic elections employing DES smartcards*, 1998. (http://www.surfnet.nl/info/attachment.db?188548)

[5] François Kooman, *Websurfen met onbetrouwbare computers*, 2006. (http://www.student.ru.nl/fkooman/%5Bscriptie%5D%20Websurfen%20met%20onbetrouwbare%20computers.pdf)

[6] Dieter Gollman, *Computer Security*, 1999, John Wiley & Sons

[7] Andrew S. Tanenbaum, *Computer Networks*, Fourth Edition, 2003, Prentice Hall PTR

[8] Cynthia. Maasbommel, *A Formal Analysis of the RIES Internet Voting Protocol*, 2007

[10] Burton Kaliski, *PKCS#5: Password-Based Cryptography Specification Version 2.0*, September 2000. (http://tools.ietf.org/html/rfc2898)

[A] http://en.wikipedia.org/wiki/Data_Encryption_Standard

http://en.wikipedia.org/wiki/Message_authentication_code
http://en.wikipedia.org/wiki/MDC-2
http://www.kiezenuithetbuitenland.nl/
nl.espacenet.com (voor het octrooi aangevraagd door Pont en Rijnland)

8 Bijlage A: Gebruikte namen en afkortingen

8.1 Algemeen

AN34	AN34 is een codering analoog aan hexadecimaal, met dien verstande dat het gebruik maakt van alle cijfers en letters van het alfabet behalve l (lima) en o (oscar). Er wordt geen onderscheid gemaakt tussen *lower*- en *uppercase*.
Hashen	Een hash-functie is een functie die zo geconstrueerd is, dat f(x) (de hashfunctie f toegepast op data x) relatief gemakkelijk te berekenen, maar om uit f(x) de originele data, x, af te leiden is zo goed als onmogelijk.
Salt, Saltpart	Een salt is een getal wat ter plekke gekozen wordt en aan de invoer van een hash-functie of versleuteling wordt vastgeplakt om zo een semi-random element toe te voegen aan deze cryptografische berekening. Als een bepaald wachtwoord een salt kan krijgen uit de getallen 1 tot 1000, moet een aanvaller op jacht gaan naar 999 verschillende versleutelingen om alle mogelijke versleutelingen van dit wachtwoord te vinden, nog afgezien van de sleutel.
Checksum	Letterlijk: controleer-som. Een functie die een bepaalde waarde berekent van zijn invoer. Deze waarde is veel kleiner, en dus makkelijker opgeslagen en verstuurd, dan de invoer, maar toch redelijkerwijs uniek voor deze invoer. Vergelijkbaar met een hash-functie, alleen bij een hash-functie ligt de nadruk op onomkeerbaarheid en bij een checksum op de uniekheid en snelheid van berekening.
MDC	Message Digest Code, een hash of checksum van een 'bericht' (in tegenstelling tot een MAC zonder gebruik van een sleutel).
MAC	Message Authentication Code, een soort MDC, alleen wordt er een cryptografische versleuteling gebruikt. Alleen mensen met de juiste sleutel kunnen de controle-code aanmaken en ontcijferen, wat het mogelijk maakt de afzender te identificeren als behorende tot deze groep.

8.2 RIES-specifiek

In deze tabel en het gehele document staat '//' voor de concatenatie van twee waardes. Dus ElID//PolGrp staat voor de bytes van ElID direct gevolgd door de bytes van PolGrp.

Cm	De kandidaatscode behorend bij een kandidaat. Bestaat uit BalBxID//PolGrp//CSeqNmbr.
BalBxID	Een uitgebreide code uniek voor elke verkiezing. Bestaat uit ElID//BalDis//CatID
ElID	De 2 bytes codering van een verkiezing.
PolGrp	De tweecijferige (één byte) codering van de politieke groepering, een getal van 01 tot 98 (99 is blanco).
SeqNumber	Het volgnummer van een kandidaat op de lijst van zijn/haar politieke groepering. Bestaat uit twee cijfers (één byte).

BalDis	Ballot District, een codering voor het specifieke district waarop deze ronde van toepassing is. Hiermee kan een verkiezing voor verschillende gebieden gescheiden worden gehouden, bijvoorbeeld als de verkiezing op verschillende tijdstippen moet worden gehouden.
CatID	Een codering van de categorie id. Van belang als kiezers uit hoofde van verschillende functies konden stemmen (bijv. als eigenaar van een bedrijf en als inwoner).
Kp	DES-sleutel, uniek voor iedere virtuele identiteit.
VnPID	De pseudo-id zoals gebruikt in de stem. Bestaat uit $DESmac_{Kp}$(BalBxID).
ParGp of DG	Deelnamegroepering van de kiezer. Voor RIESKOA niet van belang (maar wel gebruikt). Defaultwaarde '0b'.
RnTC	Een code die beschikbaar was voor AbelPI, een extra authenticatiemechanisme, dat binnen RIESKOA niet beschikbaar was. De waarde voor RnTC werd op een standaardwaarde ingevuld.
ReSPID	Een identiteitscode voor een kiezer ten behoeve van de status-service gebruikt door de stemserver. Wordt tijdens het stemproces berekend en opgestuurd. Bestaat uit MDC[$DESmac_{Kp}$\{ f(ElID//ExtParGp)\}]
RnCm	De versleutelde kandidaatskeuze van een kiezer zoals gepubliceerd in de referentie bestanden. Bestaat uit MDC[VnCx].
VnCx	De versleutelde geheime kandidaatskeuze van een kiezer. Onderdeel van de technische stem. Bestaat uit $DESmac_{Kp}$\{Cx\}.
RnVPID	De pseudo-id zoals gepubliceerd in de referentiebestanden. Bestaat uit MDC[VnPID].

9 Bijlage B: Een webpagina om valse ReSPID's in te sturen

Met de onderstaande code kan een de stemserver voor de gek worden gehouden zodat deze 'denkt' dat een kiezer zijn of haar stemcode heeft ingevuld in de echte webpagina (stemcode.html) van RIESKOA. Dit wordt pas ontdekt op het moment dat de stemserver de ReSPID controleert in de tabel van geldige ReSPID's (wat een bepaalde hoeveelheid werk vergt). In essentie leidt dit tot dezelfde hoeveelheid werk voor de stemserver als dat iemand een typfout zou maken in zijn of haar stemcode, waardoor de stemcode op het eerste gezicht nog wel geldig zou lijken maar pas als fout wordt aangemerkt op het moment dat de stemserver de ontvangen ReSPID zou controleren. (En mocht de ReSPID toevallig wel geldig lijken wordt zelfs de volgende pagina teruggestuurd door de server!)
Alle hieronder genoemde javascript-bestanden worden door de echte pagina's ook gebruikt en zijn tijdens de verkiezing openbaar.

```
<html>
<head>
   <script type="text/javascript"
src="js/general.js"></script>
   <script type="text/javascript" src="js/a010.js"></script>
   <script type="text/javascript" src="js/utils.js"></script>
   <script type="text/javascript" src="js/des.js"></script>
   <script type="text/javascript" src="js/mdc.js"></script>
   <script type="text/javascript"
src="js/browser.js"></script>
   <script type="text/javascript"
src="js/SessionData.js"></script>
</head>
<body>

<script type="text/javascript">
function doPreSubmitProcessing()
{
   abMac = "12345678";
   abMDC = mdc2(abMac);
   sReSPID = dataToHexString(abMDC);

   sSessionData=document.xform.sessiondata.value;
   oSessionData=parseSessionData(sSessionData);
   setSessionParam(oSessionData, 'respid', sReSPID);
   sSessionData=toSessionData(oSessionData);
   document.xform.reset();
   document.xform.abelpi.value=sAbelPI;
   document.xform.sessiondata.value=sSessionData;
   document.xform.actionreq.value="next";
   return true;
}
</script>
```

```
<form name="xform"
action="http://www.internetstembureau.nl/server" method="POST"
onsubmit="return doPreSubmitProcessing()">
   <input type="hidden" name="pageid" value="A010"/>
   <input type="hidden" name="elid" value="5001"/>
   <input type="hidden" name="actionreq" value="next"/>
   <input type="hidden" name="language" value="NL"/>
   <input type="hidden" name="sessiondata"
value="aWdub3Jlc3RhdHVzPWZhbHNlJm1vZGU9bm9ybWFs"/>
   <input type="hidden" name="respid" value="0"/>
   <input type="hidden" name="abelpi" value="0"/>
   <input type="hidden" name="abeltype" value="NONE"/>
   <input type="hidden" name="extpargp" value="0b"/>
   <input type="hidden" name="ABEL" value="000000"/>
   <input type="hidden" name="actionreq" value=""/>
   <input type="submit" name="submitknop" value="Verzend!"/>
</form>
</body>
</html>
```

10 Bijlage C: De broncode van het programma om DES te kraken.

10.1 DESMacEncrypter.java

```
import java.io.IOException;
import java.io.PrintStream;
import java.security.AlgorithmParameters;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.text.FieldPosition;
import java.text.DateFormat.Field;
import java.util.Properties;
import java.util.Date;
import javax.crypto.*;
import javax.crypto.spec.SecretKeySpec;
import javax.crypto.spec.IvParameterSpec;

/**
 * A public class to generate MAC's using the DesMAC-method
also used in RIESKOA.
 */
public class DESMacEncrypter
{
     private Cipher DESEncrypter;
     private SecretKey sleutel;

     /**
      * A string specifiying the mode to be used in the
internal DES-encryption.
      */
     public final static String MODE = "DES/CBC/NoPadding";

     /**
      * The Initialisation Vector for the encryption.
      */
     public final static byte[] IV = {0,0,0,0,0,0,0,0};

     /**
      * The outputformat for the timestamps. Used to format
the information of 'Date' into a String.
      */
     public final static SimpleDateFormat sdf = new
SimpleDateFormat("yyyyMMdd-HHmmss");


     public DESMacEncrypter()
     {
          this("");
     }

     /**
```

```
      * Initializes a DESMacEncrypter for encryption using the
given key.
      * @param key The bytes from this string will be used as
the key for DES. key.getBytes() should result in 8 bytes (this
is character-encoding dependent! windows-1252 will result in 8
bytes for 8 characters!)
      */
     public DESMacEncrypter (String key)
     {
          try
          {
               DESEncrypter = Cipher.getInstance(MODE);
               sleutel = new SecretKeySpec(key.getBytes(),
"DES");
               IvParameterSpec ips = new IvParameterSpec(IV);
               DESEncrypter.init(Cipher.ENCRYPT_MODE, sleutel,
ips);
          }
          catch (Exception e)
          {
               DESEncrypter = null;
               e.printStackTrace();
          }


     }


     /**
      * Generates a DesMAC as per RIESKOA from the
inputmessage;
      * @param input The message to be DesMAC-ed.
      * @param return An 8-byte MAC.
      */
     public byte[] getMac(byte[] input)
     {
          byte[] out = new byte[8];

          try
          {
               byte[] bla = DESEncrypter.doFinal(input);
               for (int i=0; i < 8; i++)
               {
                    out[i] = bla[bla.length-8+i];
               }
          }
          catch (Exception e)
          {
               e.printStackTrace();
          }
          return out;

     }


     /**
```

```
      * Encodes the input to a hexidecimal string, using
exactly two
      * hex-characters for each byte
      * @param input the byte-array to be converted
      * @return A string containing the encoding of input in
order of it's bytes.
      */
     public static String toHex(byte[] input)
     {
          String ret = "";
          for (int k=0; k < input.length; k++)
          {
               String bla =
Integer.toHexString(Byte.valueOf(input[k]).intValue());
               if (bla.length() > 1)
                   //to ensure we only have the hex-coding of
out[k]
                   //and no other bytes
                    bla = bla.substring(bla.length()-2,
bla.length());
               else
                    bla = "0"+bla;
               ret+= bla;
          }
          return ret;
     }


     /**
      * Encrypts a message, then uses that message and it's
encryption as input
      * to the cracker.
      */
     public static void main(String[] args)
     {
          Properties props = System.getProperties();

          String message = "bladiebladieblad";         //the
plaintext
          String key = "00000105";                     //the
key

        KeyString ks = new KeyString(keys[4], 10);

        //initialise the encrypter
        DESMacEncrypter dme = new
DESMacEncrypter(ks.getKeyString());

        PrintStream ps = null;     //in case the file can't be
opened.

        Date nu = new Date();      //represents the current
date/time

        //create the filename for the log-file
        StringBuffer sb = new StringBuffer("DESMacCracker.");
```

```
        sdf.format(nu, sb, new FieldPosition(
DateFormat.Field.YEAR ) );
        sb.append(".output.txt");

        //open the logfile
        try
        {
          ps = new PrintStream(sb.toString());
        }
        catch (IOException ioe)
        {
          ioe.printStackTrace();
          System.exit(1);
        }

        byte[] cipher = dme.getMac(messages[0].getBytes());
     //the ciphertext

        //initialise the cracker
        DESMacCracker dmc = new
DESMacCracker(messages[0].getBytes(), cipher);

        //start
        dmc.crack(ps);
     }
}
```

10.2 DESMacCracker.java

```
import java.io.PrintStream;
import java.text.FieldPosition;
import java.text.DateFormat;
import java.text.DateFormat.Field;
import java.util.Date;

/**
 * A class to simplify cracking a DESMac using a known
plaintext-attack
 */
public class DESMacCracker
{
     private byte[] plaintext;
     private byte[] ciphertext;
     private KeyString key;


     private DESMacCracker(){};

     /**
      * @param m The plaintext in bytes
      * @param c The ciphertext in bytes
      */
     public DESMacCracker(byte[] m, byte[] c)
     {
          plaintext = m.clone();                    //shallow
copy, but because they're
          ciphertext = c.clone();                   //primitives
it's safe
     }

     /**
      * @param m The plaintext as a String
      * @param c The ciphertext as a String
      */
     public DESMacCracker(String m, String c)
     {
          this(m.getBytes(), c.getBytes());
     }

     /**
      * @param m The plaintext in bytes
      * @param c The ciphertext as a String
      */
     public DESMacCracker(byte[] m, String c)
     {
          this(m, c.getBytes());
     }

     /**
      * @param m The plaintext as a String
      * @param c The ciphertext in bytes
      */
     public DESMacCracker(String m, byte[] c)
```

```
	{
		this(m.getBytes(), c);
	}

	/**
	 * Tries to succesfully perform a known-plaintext attack
on
	 * a DESMac-encoding.
	 * If succesful the key can be retrieved using getKey().
	 * @returns True as soon as a key has been found, false
if no
	 * key can be found
	 */
	public boolean crack ()
	{
		boolean cont = true;
		byte[] b = {0,0,0,0,0,0,0,0};
		key = new KeyString( b );
		for (; cont; cont=key.plusplus())
		{
			DESMacEncrypter dme = new
DESMacEncrypter(key.getKeyString());
			byte[] out = dme.getMac(plaintext.clone());
			if ( equals(ciphertext, out) )
			{
				return true;
			}
		}
		return false;
	}

	/**
	 * Tries every possible key to see if it matches with the
	 * plaintext/ciphertext combination
	 * Prints some text (including the plaintext and
ciphertext in Hex)
	 * and every succesfull key to the printstream ps.
	 */
	public void crack (PrintStream ps)
	{
		ps.println("Trying to crack the key...");
		ps.println("Plaintext:
"+DESMacEncrypter.toHex(plaintext));
		ps.println("Ciphertext:
"+DESMacEncrypter.toHex(ciphertext));

		boolean cont = true;
		byte[] b = {0,0,0,0,0,0,0,0};
		key = new KeyString( b );

		//to log the key
		ps.println(DESMacEncrypter.toHex(
			new KeyString("00000105", 10).getKeyBytes()));

		for (; cont; cont=key.plusplus())
		{
```

```
                DESMacEncrypter dme = new
DESMacEncrypter(key.getKeyString());
                byte[] out = dme.getMac(plaintext.clone());
                if ( equals(ciphertext, out) )
                {
                     Date nu = new Date();
                     StringBuffer sb = new StringBuffer("");
                     DESMacEncrypter.sdf.format(nu, sb,
                        new FieldPosition(
DateFormat.Field.YEAR ));
                     ps.println("["+sb.toString()+"] Succes!:
#"+

DESMacEncrypter.toHex(key.getKeyBytes()));
                }
           }
           ps.flush();
     }

     /**
      * Only to be used after crack() returns succesfully!
      * @returns The internally saved key.
      */
     public byte[] getKey()
     {
          return key.getKeyBytes().clone();
     }

     /**
      * Convenience function. Two byte-arrays are equal if and
only if
      * a) they are of the same length
      * b) they are byte-wise equal
      */
     public static boolean equals (byte[] a, byte[] b)
     {
          if ( a.length != b.length )
          {
               return false;
          }
          for (int i=0; i < a.length; i++)
          {
               if ( a[i] != b[i] )
               {
                    return false;
               }
          }
          return true;
     }
}
```

10.3 KeyString.java

```
/**
 * A class to store a (DES-)key and to do some basic
arithmetic on it.
 */
public class KeyString
{
     private short[] key;

     private KeyString(){};

     /**
      * Basic constructor.
      * @param k The byte array to become the key.
      */
     public KeyString(byte[] k)
     {
          key = new short[k.length];
          for (int i=0; i < k.length; i++)
          {
               key[i] = (short) k[i];
          }
     }

     /**
      * Convenience constructor. Really just calls
this(k.getBytes()).
      */
     public KeyString(String k)
     {
          this(k.getBytes());
     }

     /**
      * Parses a string to an integer based on the given radix
according to
      * Integer.parseInt(String, int).
      * Then splits the integer into bytes for internal
representation.
      * @param k A String representing an integer in the given
radix
      * @param radix The radix to be used on k
      */
     public KeyString(String k, int radix)
     {
          int i = Integer.parseInt(k, radix);
          short[] bla = {0,0,0,0,0,0,0,0};
          bla[4] = (short) (i&0xFF000000);
          bla[5] = (short) (i&0xFF0000);
          bla[6] = (short) (i&0xFF00);
          bla[7] = (short) (i&0xFF);
          key = bla.clone();
     }

     /**
```

```
	 * Returns the bytes representing the key.
	 */
	public byte[] getKeyBytes()
	{
		byte[] b = new byte[key.length];
		for (int i=0; i < key.length; i++)
		{
			b[i] = (byte) (key[i]&255);
		}
		return b;
	}

	/**
	 * Returns the bytes representing the key as a String
build from
	 * those bytes (using the String-constructor).
	 * Dependent on default character-encoding!
	 */
	public String getKeyString()
	{
		return new String(getKeyBytes());
	}

	/**
	 * Adds a certain number to the bytes representing the
key internally.
	 * This for key-arithmetic (looping through all possible
keys, etc.).
	 * See also KeyString.plusplus().
	 * @returns False when p < 0 or overflow, true otherwise
	 */
	public boolean plus(int p)
	{
		if (p < 0)
		{
			return false;
		}
		for (int i=key.length-1; i >= 0 && p > 0; i--)
		{
			int n = (key[i]+p);
			p = n/256;
			key[i] = (short) (n%256);
		}
		if ( p > 0 )
		{
			return false;
		}
		else
		{
			return true;
		}
	}

	/**
	 * Really just a convenience function. Calls
KeyString.plus(1)
```

```
      * Intended Usage: along the lines of i++ in a for-loop,
to loop through
      * all available keys (0 ... MAX)
      * @returns False when overflow, true otherwise.
      */
     public boolean plusplus()
     {
          return plus(1);
     }

     /**
      * Two KeyString objects are equal when
      * a) Both are KeyString objects
      * b) The internal byte-representation of the keys are of
the same length
      * c) The internal byte-representation of the keys are
byte-wise equal
      */
     public boolean equals(Object o)
     {
          if (o instanceof KeyString)
          {
               KeyString other = (KeyString) o;
               byte[] otherBytes = other.getKeyBytes();
               if ( otherBytes.length != key.length )
               {
                    return false;
               }
               for (int i=0; i < key.length; i++)
               {
                    if ( key[i] != otherBytes[i] )
                    {
                         return false;
                    }
               }
               return true;
          }
          return false;
     }

}
```

11 Bijlage D: Broncode van de StemControle

Hieronder wordt de broncode getoond van de php- en cpp-bestanden die gebruikt zijn voor het Stemcontrole-systeem.

11.1 index.php

Dit bestand genereert de hoofdpagina van Stemcontrole waarop de technische stem wordt ingevoerd.

```
<?php

require './utilslocal.php';

$html_begin =  "<table style=\"font-size:large\"><tr><td>\n
          <p class=\"rubriekTitle\">Welkom bij de
Stemcontrole-service van de SoS-group van de Radboud
Universiteit
          Nijmegen. Voer hieronder uw technische stem in om
deze te controleren.</p>\n
          </td></tr>\n";

$html_form =   "<tr><td><form name=\"hashform\"
action=\"https://".$pathToControlePHP. "\" method=\"post\"
onsubmit=\"return blahash();\">\n
           \t<label>Linkerhelft Technische Stem: <input
type=\"text\" name=\"LinkerTechnischeStem\" id=\"links\"
maxlength=\"16\"/></label><br/>\n
           \t<label>Rechterhelft Technische Stem: <input
type=\"text\" name=\"RechterTechnischeStem\" id=\"rechts\"
maxlength=\"16\"/></label><br/>\n
           \t<input type=\"hidden\" id=\"mdclinks\"
name=\"LinkerTechnischeStemMDC\"/><input type=\"hidden\"
id=\"mdcrechts\" name=\"RechterTechnischeStemMDC\"/><br/>\n
           \t<input type=\"submit\" name=\"MDC_SUBMIT\"
value=\"Verzenden\"/><br/>\n
           <form/></td></tr></table>\n";

$html_hashfunction = "<script type=\"text/javascript\">\n
                function blahash()\n{\n\tvar links =
hexToString(document.getElementById('links').value);\n\tvar
rechts =
hexToString(document.getElementById('rechts').value);\n\n\t
                if (links == \"\" || rechts == \"\")
{window.alert('Niet alle velden zijn ingevuld!');return
false;}\n\t
                var rechtsMDC =
stringToHex(mdc2(rechts));\n\tvar linksMDC =
stringToHex(mdc2(links));\n\t
                document.getElementById('rechts').value =
\"\";\n\tdocument.getElementById('links').value = \"\";\n\t
                document.getElementById('mdcrechts').value =
rechtsMDC;\n\tdocument.getElementById('mdclinks').value =
linksMDC;\n\t
                return true;\n}\n</script></br>\n";
```

```
echo $html_head;
echo $html_body_start;
echo $html_begin;
echo $html_hashfunction;
echo $html_form;
echo $html_end;

?>
```

11.2 controle.php

Dit bestand vraagt de gegevens van de ingevoerde stem (en alle andere stemmen van dezelfde kiezer) op en genereert een webpagina om deze gegevens te tonen.

```
<?php
require './utilslocal.php';

echo $html_head;
echo $html_body_start;

if (isset($_POST['MDC_SUBMIT']))
{
     echo "    <br /><table><tr><td><p
class=\"rubriekTitle\">Hieronder zit u de resultaten van de
stemcontrole.<br
          />\nKlik <a href=\"#sos\">:hier:</a> voor de
resultaten van de SOS-groep als die buiten de pagina
vallen.</p></td></tr>\n
          <tr><td><table border=\"2\">\n";


     $mdclinks = $_POST['LinkerTechnischeStemMDC'];
     $mdcrechts = $_POST['RechterTechnischeStemMDC'];

     $link = mysql_connect($host, $user, $pass) or die("1:"
.mysql_error());
     mysql_select_db($db, $link) or die ("2:".mysql_error());

     $mdclinks = mysql_real_escape_string($mdclinks, $link);
     $mdcrechts = mysql_real_escape_string($mdcrechts, $link);



     $queryViews = "UPDATE `Views` SET `Number`=`Number`+1
WHERE `Identiteit`='$mdclinks'";
     $queryTTPI = "SELECT * FROM OntvangenStemmenTTPI WHERE
LinkerDeel='$mdclinks' AND RechterDeel='$mdcrechts'";
     $querySOS = "SELECT * FROM OntvangenStemmenSOS WHERE
LinkerDeel='$mdclinks' AND RechterDeel='$mdcrechts'";

     $resultViews = mysql_query($queryViews, $link) or
die("4:". mysql_errno(). " ". mysql_error());
     if ( mysql_affected_rows($link) <= 0 )
     {
          $queryViews = "INSERT INTO `Views` SET
`Identiteit`='$mdclinks', `Number`= 1";
          $resultViews = mysql_query($queryViews, $link) or
die("4:". mysql_errno(). " ". mysql_error());
     }
     $resultTTPI = mysql_query($queryTTPI, $link) or die("4:".
mysql_errno(). " ". mysql_error());
     $resultSOS = mysql_query($querySOS, $link) or die("4:".
mysql_errno(). " ". mysql_error());
```

```
     $numberTTPI = mysql_num_rows($resultTTPI);
     $numberSOS = mysql_num_rows($resultSOS);

     echo "<tr><td colspan=\"3\"
style=\"color:#be311a\"><b>$numberTTPI ontvangen stemmen met
deze exacte technische stem verwerkt door het officiele
telbureau
     (TTPI).</b></td></tr>";
     $i = 1;
     while ($row = mysql_fetch_assoc($resultTTPI))
     {
          echo "<tr><td>$i. </td>";
          $kandidaat = "";
          if ($row['Kandidaatscode'] > 0)
          {
               $code = $row['Kandidaatscode'];
               $kandidaatquery = "SELECT * FROM Kandidaten
WHERE `Code` = '$code'";
               $resultkandidaat =
mysql_query($kandidaatquery);
               if (mysql_num_rows($resultkandidaat) > 0)
               {
                    $kandidaatrow =
mysql_fetch_assoc($resultkandidaat);
                    $kandidaatnaam =
$kandidaatrow['Kandidaat'];
                    $kandidaatlijst = $kandidaatrow['Lijst'];
                    $kandidaat =
"\n<tr><td> </td><td>Kandidaat:
$kandidaatnaam</td><td>Lijst: $kandidaatlijst</td></tr>";
               }
          }
          if ($row['Accepted'] == 'yes')
          {
               echo "<td>De stem is
geaccepteerd.</td><td> </td></tr>$kandidaat\n";
          }
          else
          {
               echo "<td>De stem is niet geaccepteerd.</td>";
               if ($row['DubbeleStem'] == 'yes')
               {
                    echo "<td>Er is meerdere malen op dezelfde
kandidaat gestemd, en er is maar een stem hiervan
meegeteld.</td></tr>$kandidaat\n";
               }
               else if ($row['OnbekendeVoter'] == 'yes')
               {
                    echo "<td>Deze technische stem behoorde
bij geen bekende (Virtuele)
Identiteit.</td></tr>$kandidaat\n";
               }
               else if ($row['OnbekendeKandidaat'] == 'yes')
               {
                    echo "<td>Deze technische stem was voor
geen bekende kandidaat.</td></tr>$kandidaat\n";
```

```
                }
                else if ($row['Ongeldig'] == 'yes')
                {
                     echo "<td>Er zijn meerdere stemmen op
verschillende kandidaten gedaan. Alle stemmen zijn
afgekeurd.</td></tr>$kandidaat\n";
                }
                else if ($row['LowerPrio'] == 'yes')
                {
                     echo "<td>Deze stem heeft een lagere
prioriteit als een andere stem van dezelfde stemmer. U bent
heel knap bezig.</td></tr>$kandidaat\n";
                }
                else if ($row['KlachtOID'] == 'yes')
                {
                     echo "<td>Deze technische stem behoorde
bij een virtuele identiteit dat is
vervallen.</td></tr>$kandidaat\n";
                }
                else if ($row['Test'] == 'yes')
                {
                     echo "<td>Deze technische stem behoorde
bij een stembiljet dat werd gebruikt om te
testen.</td></tr>$kandidaat \n";
                }
                else if ($row['NietVerstrekt'] == 'yes')
                {
                     echo "<td>Deze technische stem behoorde
bij een stembiljet dat nooit is verstrekt.</td></tr>$kandidaat
\n";
                }
                else
                {
                     echo "<td>De precieze reden van afkeuring
is onbekend.</td></tr>$kandidaat \n";
                }

           }
           $i++;
     }

     $queryTTPI = "SELECT * FROM OntvangenStemmenTTPI WHERE
LinkerDeel='$mdclinks' AND RechterDeel!='$mdcrechts'";
     $resultTTPI = mysql_query($queryTTPI, $link) or die("4:".
mysql_errno(). " ". mysql_error());
     $numberTTPI = mysql_num_rows($resultTTPI);

     echo "<tr><td colspan=\"3\" style=\"color:#be311a\">Er
zijn $numberTTPI andere ontvangen stemmen van deze virtuele
identiteit verwerkt door het officiele telbureau
(TTPI).</td></tr>";
     $i = 1;
     while ($row = mysql_fetch_assoc($resultTTPI))
     {
          echo "<tr><td>$i. </td>";
          $kandidaat = "";
```

```
          if ($row['Kandidaatscode'] > 0)
          {
               $code = $row['Kandidaatscode'];
               $kandidaatquery = "SELECT * FROM Kandidaten
WHERE `Code` = '$code'";
               $resultkandidaat =
mysql_query($kandidaatquery);
               if (mysql_num_rows($resultkandidaat) > 0)
               {
                    $kandidaatrow =
mysql_fetch_assoc($resultkandidaat);
                    $kandidaatnaam =
$kandidaatrow['Kandidaat'];
                    $kandidaatlijst = $kandidaatrow['Lijst'];
                    $kandidaat =
"\n<tr><td> </td><td>Kandidaat:
$kandidaatnaam</td><td>Lijst: $kandidaatlijst</td></tr>";
               }
          }
          if ($row['Accepted'] == 'yes')
          {
               echo "<td>De stem is
geaccepteerd.</td><td> </td></tr>$kandidaat\n";
          }
          else
          {
               echo "<td>De stem is niet geaccepteerd.</td>";
               if ($row['DubbeleStem'] == 'yes')
               {
                    echo "<td>Er is meerdere malen op dezelfde
kandidaat gestemd, en er is maar een stem hiervan
meegeteld.</td></tr>$kandidaat\n";
               }
               else if ($row['OnbekendeVoter'] == 'yes')
               {
                    echo "<td>Deze technische stem behoorde
bij geen bekende (Virtuele)
Identiteit.</td></tr>$kandidaat\n";
               }
               else if ($row['OnbekendeKandidaat'] == 'yes')
               {
                    echo "<td>Deze technische stem was voor
geen bekende kandidaat.</td></tr>$kandidaat\n";
               }
               else if ($row['Ongeldig'] == 'yes')
               {
                    echo "<td>Er zijn meerdere stemmen op
verschillende kandidaten gedaan. Alle stemmen zijn
afgekeurd.</td></tr>$kandidaat\n";
               }
               else if ($row['LowerPrio'] == 'yes')
               {
                    echo "<td>Deze stem heeft een lagere
prioriteit als een andere stem van dezelfde stemmer. U bent
heel knap bezig.</td></tr>$kandidaat\n";
               }
```

```
                else if ($row['KlachtOID'] == 'yes')
                {
                     echo "<td>Deze technische stem behoorde
bij een virtuele identiteit dat is
vervallen.</td></tr>$kandidaat\n";
                }
                else if ($row['Test'] == 'yes')
                {
                     echo "<td>Deze technische stem behoorde
bij een stembiljet dat werd gebruikt om te
testen.</td></tr>$kandidaat \n";
                }
                else if ($row['NietVerstrekt'] == 'yes')
                {
                     echo "<td>Deze technische stem behoorde
bij een stembiljet dat nooit is verstrekt.</td></tr>$kandidaat
\n";
                }
                else
                {
                     echo "<td>De precieze reden van afkeuring
is onbekend.</td></tr>$kandidaat \n";
                }

           }
           $i++;
     }

     echo "<tr><td colspan=\"3\"> </td></tr>\n";

     echo "<tr><td colspan=\"3\" style=\"color:#be311a\"><a
name=\"sos\" style=\"text-decoration:none\"><b>$numberSOS
ontvangen stemmen
     met deze exacte technische stem verwerkt door het
telsysteem van de
     SOS-groep van de Radboud Universiteit
Nijmegen.</b></a></td></tr>";
     $i = 1;
     while ($row = mysql_fetch_assoc($resultSOS))
     {
           echo "<tr><td>$i. </td>";
           $kandidaat = "";
           if ($row['Kandidaatscode'] > 0)
           {
                $code = $row['Kandidaatscode'];
                $kandidaatquery = "SELECT * FROM Kandidaten
WHERE `Code` = '$code'";
                $resultkandidaat =
mysql_query($kandidaatquery);
                if (mysql_num_rows($resultkandidaat) > 0)
                {
                     $kandidaatrow =
mysql_fetch_assoc($resultkandidaat);
                     $kandidaatnaam =
$kandidaatrow['Kandidaat'];
                     $kandidaatlijst = $kandidaatrow['Lijst'];
```

```
                    $kandidaat =
"\n<tr><td> </td><td>Kandidaat:
$kandidaatnaam</td><td>Lijst: $kandidaatlijst</td></tr>";
               }
          }
          if ($row['Accepted'] == 'yes')
          {
               echo "<td>De stem is
geaccepteerd.</td><td> </td></tr>$kandidaat\n";
          }
          else
          {
               echo "<td>De stem is niet geaccepteerd.</td>";
               if ($row['DubbeleStem'] == 'yes')
               {
                    echo "<td>Er is meerdere malen op dezelfde
kandidaat gestemd, en er is maar een stem hiervan
meegeteld.</td></tr>$kandidaat\n";
               }
               else if ($row['OnbekendeVoter'] == 'yes')
               {
                    echo "<td>Deze technische stem behoorde
bij geen bekende (Virtuele)
Identiteit.</td></tr>$kandidaat\n";
               }
               else if ($row['OnbekendeKandidaat'] == 'yes')
               {
                    echo "<td>Deze technische stem was voor
geen bekende kandidaat.</td></tr>$kandidaat\n";
               }
               else if ($row['Ongeldig'] == 'yes')
               {
                    echo "<td>Er zijn meerdere stemmen op
verschillende kandidaten gedaan. Alle stemmen zijn
afgekeurd.</td></tr>$kandidaat\n";
               }
               else if ($row['LowerPrio'] == 'yes')
               {
                    echo "<td>Deze stem heeft een lagere
prioriteit als een andere stem van dezelfde stemmer. U bent
heel knap bezig.</td></tr>$kandidaat\n";
               }
               else if ($row['KlachtOID'] == 'yes')
               {
                    echo "<td>Deze technische stem behoorde
bij een virtuele identiteit dat is
vervallen.</td></tr>$kandidaat\n";
               }
               else if ($row['Test'] == 'yes')
               {
                    echo "<td>Deze technische stem behoorde
bij een stembiljet dat werd gebruikt om te
testen.</td></tr>$kandidaat \n";
               }
               else if ($row['NietVerstrekt'] == 'yes')
               {
```

```
                    echo "<td>Deze technische stem behoorde
bij een stembiljet dat nooit is verstrekt.</td></tr>$kandidaat
\n";
               }
               else
               {
                    echo "<td>De precieze reden van afkeuring
is onbekend.</td></tr>$kandidaat \n";
               }

          }
          $i++;
     }

     $querySOS = "SELECT * FROM OntvangenStemmenSOS WHERE
LinkerDeel='$mdclinks' AND RechterDeel!='$mdcrechts'";
     $resultSOS = mysql_query($querySOS, $link) or die("4:".
mysql_errno(). " ". mysql_error());
     $numberSOS = mysql_num_rows($resultSOS);

     echo "<tr><td colspan=\"3\" style=\"color:#be311a\">Er
zijn $numberSOS andere ontvangen stemmen van deze virtuele
identiteit verwerkt door het telsysteem van de
     SOS-groep van de Radboud Universiteit
Nijmegen.</td></tr>";
     $i = 1;
     while ($row = mysql_fetch_assoc($resultSOS))
     {
          echo "<tr><td>$i. </td>";
          $kandidaat = "";
          if ($row['Kandidaatscode'] > 0)
          {
               $code = $row['Kandidaatscode'];
               $kandidaatquery = "SELECT * FROM Kandidaten
WHERE `Code` = '$code'";
               $resultkandidaat =
mysql_query($kandidaatquery);
               if (mysql_num_rows($resultkandidaat) > 0)
               {
                    $kandidaatrow =
mysql_fetch_assoc($resultkandidaat);
                    $kandidaatnaam =
$kandidaatrow['Kandidaat'];
                    $kandidaatlijst = $kandidaatrow['Lijst'];
                    $kandidaat =
"\n<tr><td> </td><td>Kandidaat:
$kandidaatnaam</td><td>Lijst: $kandidaatlijst</td></tr>";
               }
          }
          if ($row['Accepted'] == 'yes')
          {
               echo "<td>De stem is
geaccepteerd.</td><td> </td></tr>$kandidaat\n";
          }
          else
          {
```

```
                echo "<td>De stem is niet geaccepteerd.</td>";
                if ($row['DubbeleStem'] == 'yes')
                {
                     echo "<td>Er is meerdere malen op dezelfde
kandidaat gestemd, en er is maar een stem hiervan
meegeteld.</td></tr>$kandidaat\n";
                }
                else if ($row['OnbekendeVoter'] == 'yes')
                {
                     echo "<td>Deze technische stem behoorde
bij geen bekende (Virtuele)
Identiteit.</td></tr>$kandidaat\n";
                }
                else if ($row['OnbekendeKandidaat'] == 'yes')
                {
                     echo "<td>Deze technische stem was voor
geen bekende kandidaat.</td></tr>$kandidaat\n";
                }
                else if ($row['Ongeldig'] == 'yes')
                {
                     echo "<td>Er zijn meerdere stemmen op
verschillende kandidaten gedaan. Alle stemmen zijn
afgekeurd.</td></tr>$kandidaat\n";
                }
                else if ($row['LowerPrio'] == 'yes')
                {
                     echo "<td>Deze stem heeft een lagere
prioriteit als een andere stem van dezelfde stemmer. U bent
heel knap bezig.</td></tr>$kandidaat\n";
                }
                else if ($row['KlachtOID'] == 'yes')
                {
                     echo "<td>Deze technische stem behoorde
bij een virtuele identiteit dat is
vervallen.</td></tr>$kandidaat\n";
                }
                else if ($row['Test'] == 'yes')
                {
                     echo "<td>Deze technische stem behoorde
bij een stembiljet dat werd gebruikt om te
testen.</td></tr>$kandidaat \n";
                }
                else if ($row['NietVerstrekt'] == 'yes')
                {
                     echo "<td>Deze technische stem behoorde
bij een stembiljet dat nooit is verstrekt.</td></tr>$kandidaat
\n";
                }
                else
                {
                     echo "<td>De precieze reden van afkeuring
is onbekend.</td></tr>$kandidaat \n";
                }

           }
           $i++;
```

```
	}

	echo "</table>\n</td></tr></table>";

	echo "\n\n<br /><br /><a href=\"#top\">Ga terug naar
boven</a><br />\n";
	echo "<a href=\"./index.php\">Ga terug naar het
beginscherm</a><br />\n";

	mysql_close($link);
}
else
{
	echo "\n\n<br /><br /><a href=\"./index.php\">Ga naar het
beginscherm om uw technische stem in te voeren</a><br />\n";
}

echo $html_end;
?>
```

11.3 utilslocal.php

Dit bestand bevat een aantal variabelen die door meerdere bestanden van Stemcontrole worden gebruikt.

```
<?php

$pathToControlePHP =
'www.sos.cs.ru.nl/research/sosries/stemcontrole/controle.php';

$host = ########;
$user = ########;
$pass = ########;
$db   = ########;


$html_head = "<html><head><title>Welkom bij
Stemcontrole!</title>\n
          <link rel=\"stylesheet\" type=\"text/css\"
href=\"run.css\" />\n
          <link rel=\"stylesheet\" type=\"text/css\"
href=\"default_normal.css\" />\n
          <script type=\"text/javascript\" src=\"./mdc.js\"
defer></script>\n
          <script type=\"text/javascript\" src=\"./utils.js\"
defer></script>\n
          <script type=\"text/javascript\" src=\"./des.js\"
defer></script></head>\n";

$html_body_start = "<body><table border=\"0\" frame=\"void\"
cellspacing=\"0\" cellpadding=\"0\">
          <tr><td width=\"216\"> </td><td><a
href=\"http://www.ru.nl\"><img width=\"544\" height=\"76\"
alt=\"Logo\" border=\"0\" hspace=\"0\" vspace=\"0\"
          src=\"logo_ru_groot.gif\" /></a></td></tr>
          <tr><td colspan=\"2\"><img
src=\"homepage_radboud1.jpg\"></td></tr></table>";

$html_end = "  <br /><br /><br />\n
          <p style=\"font-size:xx-small\">Deze service wordt u
aangeboden door de <a href=\"http://www.sos.cs.ru.nl\"
target=\"_new\">Security of Systems(SOS)</a>-groep van de
Radboud Universiteit Nijmegen.<br />
          De service is ontworpen en geschreven door Sven
Haster als onderdeel van zijn Bachelorscriptie. Voor deze
service wordt gebruik gemaak van (openbare)
          informatie die wordt aangeleverd door <a
href=\"http://www.ttpi.nl\" target=\"_new\">TTPI</a> en <a
href=\"http://www.cs.ru.nl/staff/Engelbert.Hubbers\"
          target=\"_new\">E. Hubbers</a></body></html>\n";

?>
```

11.4 createSQLfromCSV.cpp

Dit programma werd gebruikt om de bestanden informatie over de kandidaten en informatie van de getelde stemmen zoals verstrekt door TTPI en SOS (Engelbert Hubbers) om te zetten in SQL-queries die deze informatie in de voor Stemcontrole gebruikte MySQL-database konden invoegen.

```
#include <fstream>
#include <string>
#include <iostream>
#include <cctype>
#include <cstdlib>

using namespace std;

const int MAX_LINE_LENGTH = 512;        //outputLine would end
up a little over 300 characters
const long MAX_LINES = (2*1024*1024)/MAX_LINE_LENGTH;
     //to create no file larger than 2 MB
const int MAX_NUMBER_FIELDS = 17;
const int MAX_NUMBER_LISTS = 100;
const int MAX_LISTNAME_LENGTH = 100;
const char CREATE_DB[] = "CREATE DATABASE IF NOT EXISTS
`stemcontrole` DEFAULT CHARACTER SET latin1 COLLATE
latin1_general_ci;\nUSE `stemcontrole`;\n";
const int CREATE_DB_LEN = strlen(CREATE_DB);
const char* outFileName;
int numberFiles = 0;


/*
** Splits 'line' by replacing any ',' with a '\0', effectively
creating substrings, the starts of which are saved in 'fields'
*/
void getFields(char* line, int* fields)
{
     int i=0, j=0;
     fields[i++] = 0;
        while ( line[j] != '\0' )
        {
             if ( line[j] == ',' )
             {
                  line[j] = '\0';
                  fields[i++] = j+1;
          }
          j++;
     }
}

/*
** Read a line from the inputfile, parse it for te required
information, construct a MySQL INSERT query, and write that
query to the output file.
** Wash, rinse, repeat.
*/
```

```
bool createAndWriteSQL (ifstream& in, ofstream& out, const
char* table)
{
     int lines = 0;
     long totalLines=0;
     char inLine [MAX_LINE_LENGTH];
     int fields [MAX_NUMBER_FIELDS];
     char outLine [MAX_LINE_LENGTH];
     int outLineLength;

     out.write(CREATE_DB, CREATE_DB_LEN);

     //to drop an existing table
     outLine[0] = '\0';
     strcat(outLine, "DROP TABLE IF EXISTS
`OntvangenStemmen");
     strcat(outLine, table);
     strcat(outLine, "`;\n");
     out.write(outLine, strlen(outLine));

     //to create a clean table
     outLine[0] = '\0';
     strcat(outLine, "CREATE TABLE `OntvangenStemmen");
     strcat(outLine, table);
     strcat(outLine, "` (\n`Index` int(11) NOT NULL
auto_increment,\n");
     strcat(outLine, " `LinkerDeel` varchar(32) collate
latin1_general_ci NOT NULL default '' COMMENT
'MDC(MAC(ELECTION_ID))',\n");
     strcat(outLine, "`RechterDeel` varchar(32) collate
latin1_general_ci NOT NULL default '' COMMENT
'MDC(MAC(CANDIDATE_ID))',\n");
     strcat(outLine, "`Accepted` enum('yes','no') collate
latin1_general_ci NOT NULL default 'yes',\n");
     strcat(outLine, "`Kandidaatscode` int(8) NOT NULL default
'0',\n");
     out.write(outLine, strlen(outLine));
     outLine[0] = '\0';
     strcat(outLine, "`DubbeleStem` enum('yes','no') collate
latin1_general_ci NOT NULL default 'yes',\n");
     strcat(outLine, "`OnbekendeVoter` enum('yes','no')
collate latin1_general_ci NOT NULL default 'yes',\n");
     strcat(outLine, "`OnbekendeKandidaat` enum('yes','no')
collate latin1_general_ci NOT NULL default 'yes',\n");
     strcat(outLine, "`Ongeldig` enum('yes','no') collate
latin1_general_ci NOT NULL default 'yes',\n");
     strcat(outLine, "`LowerPrio` enum('yes','no') collate
latin1_general_ci NOT NULL default 'yes',\n");
     strcat(outLine, "`KlachtOID` enum('yes','no') collate
latin1_general_ci NOT NULL default 'yes',");
     strcat(outLine, "`Test` enum('yes','no') collate
latin1_general_ci NOT NULL default 'yes',\n");
     strcat(outLine, "`NietVerstrekt` enum('yes','no') collate
latin1_general_ci NOT NULL default 'yes',\n");
     strcat(outLine, "PRIMARY KEY  (`Index`)\n");
```

```
     strcat(outLine, ") ENGINE=MyISAM DEFAULT CHARSET=latin1
COLLATE=latin1_general_ci;\n");
     out.write(outLine, strlen(outLine));


     //to intitialize the query
     outLine[0] = '\0';
     strcat(outLine, "INSERT INTO `OntvangenStemmen");
     strcat(outLine, table);
     strcat(outLine, "` ( `LinkerDeel` , `RechterDeel` ,
`Accepted` , `Kandidaatscode` , `DubbeleStem` ,
`OnbekendeVoter` , `OnbekendeKandidaat` ,");
     strcat(outLine, " `Ongeldig`, `LowerPrio` , `KlachtOID` ,
`Test` , `NietVerstrekt` ) VALUES ('");
     outLineLength = strlen(outLine);

     while (!in.eof())
     {
          in.getline(inLine, MAX_LINE_LENGTH);
          if ( in.eof() ) break;
          if ( in.fail() )
          {
               cout << "Failed to read string from
inputFile\n";
               return false;
          }
          getFields(inLine, fields);


          outLine[outLineLength] = '\0';
          strcat(outLine, (inLine+fields[4]));
     //MDC(linkerdeel technischestem), which is MDC(VOTER_ID)
          strcat(outLine, "', '");
          strcat(outLine, (inLine+fields[5]));
     //MDC(rechterdeel technischestem), which is
MDC(MAC(CANDIDATE_ID))
          strcat(outLine, "', '");

          strcat(outLine, (inLine+fields[6]));
     //accepted ('yes' or 'no' )
          strcat(outLine, "', '");

          strcat(outLine, (inLine+fields[8]));
     //candidate code
          strcat(outLine, "', '");

          for ( int i=9; i < MAX_NUMBER_FIELDS; i++ )
          {
               if ( strcmp( (inLine+fields[i]), "1" ) == 0 )
     //error-codes
               {
                    strcat(outLine, "yes', '");
               }
               else
               {
                    strcat(outLine, "no', '");
```

```
                }
            }

            //to close the query
            int len = strlen(outLine);
            outLine[len-3] = ')';
            outLine[len-2] = ';';
            outLine[len-1] = '\n';

            out.write(outLine, len);
            lines++;

            if (out.fail())
            {
                cout << "Failed to write line " << lines << ":
\n\t" << outLine <<endl;
                return false;
            }
            if (lines >= MAX_LINES)
            {
                totalLines += lines;
                lines = 0;
                out.close();

                numberFiles++;
                char numberString[4];
                numberString[0] = '0' + ((numberFiles/100)%10);
                numberString[1] = '0' + ((numberFiles/10)%10);
                numberString[2] = '0' + (numberFiles%10);
                numberString[3] = '\0';

                char outputfilename [520];
                strcpy(outputfilename, outFileName);
                strcat(outputfilename, numberString);
                strcat(outputfilename, ".sql");

                out.open(outputfilename);
                if ( out.fail() )
                {
                    cout << "Outputbestand: " <<
outputfilename << " kon niet geopend worden!\n";
                    exit(1);
                }
            }



    }
    totalLines += lines;
    cout << totalLines << " lines written." << endl;
    return true;
}

/*
** Read a line from the lists-file, parse it for the listname
and listcode, save the listname at position listcode.
```

```
** Wash, rinse, repeat.
** Read a line from the candidates-file, parse it for the
candidates-name and code, get the appropriate listname, create
sql-insert-query, write that to output file.
** Wash, rinse, repeat.
*/
bool createAndWriteSQL (ifstream& lists, ifstream& candidates,
ofstream& out)
{
     int lines = 0;
     long totalLines = 0;
     char listNames[MAX_NUMBER_LISTS][MAX_LISTNAME_LENGTH+1];
     char inLine[MAX_LINE_LENGTH];
     char outLine[MAX_LINE_LENGTH];


     while ( !lists.eof() )
     {
          lists.getline(inLine, MAX_LINE_LENGTH);
          if ( lists.eof() ) break;

          if ( lists.fail() )
          {
               cout << "Failed to read from the listname-
file.\n";
               exit(3);
          }

          int PolGrp = (inLine[13] - '0');        //read the
Political Group ('Party', 'Lijst') Number
          PolGrp *= 10;
          PolGrp += (inLine[14] - '0');

          inLine[117] = '\0';

          lines++;
          strcpy(listNames[PolGrp], (inLine+17));      //copy
the name of the political group into the list of listnames, at
the position PolGrp

          if ( PolGrp == 99 ) break;              //PolGrp =
99 is the last one (BLANCO)
     }

     cout << lines << " Listnames read." << endl;
     lines = 0;

     out.write(CREATE_DB, CREATE_DB_LEN);

     //to drop an existing table
     outLine[0] = '\0';
     strcat(outLine, "DROP TABLE IF EXISTS `Kandidaten`;\n");
     out.write(outLine, strlen(outLine));

     //to create a clean table
     outLine[0] = '\0';
```

```
     strcat(outLine, "CREATE TABLE `Kandidaten` (\n");
     strcat(outLine, "`Code` int(11) NOT NULL default
'0',\n");
     strcat(outLine, " `Lijst` varchar(100) collate
latin1_general_ci NOT NULL default '',\n");
     strcat(outLine, "`Kandidaat` varchar(100) collate
latin1_general_ci NOT NULL default '',\n");
     strcat(outLine, "PRIMARY KEY  (`Code`)\n");
     strcat(outLine, ") ENGINE=MyISAM DEFAULT CHARSET=latin1
COLLATE=latin1_general_ci;\n");
     out.write(outLine, strlen(outLine));

     strcpy(outLine, "INSERT INTO `Kandidaten` ( `Code` ,
`Lijst` , `Kandidaat` ) VALUES (\"");
     int outLen = strlen(outLine);


     while ( !candidates.eof() )
     {
          candidates.getline(inLine, MAX_LINE_LENGTH);
          if ( candidates.eof() ) break;

          if ( candidates.fail() )
          {
               cout << "Failed to read from the candidates-
file.\n";
               exit(3);
          }

          int PolGrp = (inLine[13] - '0');        //read the
Political Group ('Party', 'Lijst') Number
          PolGrp *= 10;
          PolGrp += (inLine[14] - '0');

          for (int i=0; i < 4; i++) outLine[outLen+i] =
inLine[13+i];       //Copy Candidate code (which is PolGrp (2
digits) and Candidate number (2 digits)

          outLine[outLen+4] = '\0';
          strcat(outLine, "\", \"");

          strcat(outLine, listNames[PolGrp]);
     //insert the name of the PolGrp the candidate belongs to
into the query
          strcat(outLine, "\", \"");

          inLine[117] = '\0';
          strcat(outLine, (inLine+17));
     //insert the name of the candidate into the query
          strcat(outLine, "\");");

          int len = strlen(outLine);

          out.write(outLine, len);

          lines++;
```

```
          if (out.fail())
          {
               cout << "Failed to write line " << lines << ":
\n\t" << outLine <<endl;
               return false;
          }
          if (lines >= MAX_LINES)
          {
               totalLines += lines;
               lines = 0;
               out.close();

               numberFiles++;
               char numberString[4];
               numberString[0] = '0' + ((numberFiles/100)%10);
               numberString[1] = '0' + ((numberFiles/10)%10);
               numberString[2] = '0' + (numberFiles%10);
               numberString[3] = '\0';

               char outputfilename [520];
               strcpy(outputfilename, outFileName);
               strcat(outputfilename, numberString);
               strcat(outputfilename, ".sql");

               out.open(outputfilename);
               if ( out.fail() )
               {
                    cout << "Outputbestand: " <<
outputfilename << " kon niet geopend worden!\n";
                    exit(1);
               }
          }
     }

     totalLines += lines;
     cout << totalLines << " lines written." << endl;
     return true;
}




/*
** Get inputfilename, outputfilename and tablename-appendix
from commandline-parameters, open appropriate files, and start
the sql-creating
*/
int main (int argc, char* argv[])
{
     if ( argc < 4 )
     {
          cout << "Geef de bestandsnamen mee als parameters 1
en 2, en {TTPI,SOS} of de derde bestandsnaam als parameter
3!\n";
          exit(0);
```

```
     }
     if ( strcmp("TTPI", argv[3]) == 0 || strcmp("SOS",
argv[3]) == 0 )
     {
          cout << "OntvangenStemmen" << argv[3] << ": ..." <<
endl;
          ifstream inputFile;
          ofstream outputFile;

          inputFile.open(argv[1]);
          if ( inputFile.fail() )
          {
               cout << "Inputbestand: " << argv[1] << " kon
niet geopend worden!\n";
               exit(1);
          }
          outFileName = argv[2];

          numberFiles++;
          char numberString[4];
          numberString[0] = '0' + ((numberFiles/100)%10);
          numberString[1] = '0' + ((numberFiles/10)%10);
          numberString[2] = '0' + (numberFiles%10);
          numberString[3] = '\0';

          char outputfilename [520];
          strcpy(outputfilename, outFileName);
          strcat(outputfilename, numberString);
          strcat(outputfilename, ".sql");

          outputFile.open(outputfilename);
          if ( outputFile.fail() )
          {
               cout << "Outputbestand: " << outputfilename <<
" kon niet geopend worden!\n";
               exit(1);
          }

          if (!createAndWriteSQL(inputFile, outputFile,
argv[3]))
          {
               cout << "Fatal error: Program closing
down...\n";
               exit(2);
          }
          inputFile.close();
          outputFile.close();
     }
     else
     {
          cout << "Candidates: ..." << endl;
          ifstream listsFile;
          ifstream candidatesFile;
          ofstream outputFile;
          listsFile.open(argv[1]);
          if ( listsFile.fail() )
```

```
          {
               cout << "Inputbestand: " << argv[1] << " kon
niet geopend worden!\n";
               exit(1);
          }
          candidatesFile.open(argv[3]);
          if ( candidatesFile.fail() )
          {
               cout << "Inputbestand: " << argv[3] << " kon
niet geopend worden!\n";
               exit(1);
          }
          outFileName = argv[2];

          numberFiles++;
          char numberString[4];
          numberString[0] = '0' + ((numberFiles/100)%10);
          numberString[1] = '0' + ((numberFiles/10)%10);
          numberString[2] = '0' + (numberFiles%10);
          numberString[3] = '\0';

          char outputfilename [520];
          strcpy(outputfilename, outFileName);
          strcat(outputfilename, numberString);
          strcat(outputfilename, ".sql");

          outputFile.open(outputfilename);
          if ( outputFile.fail() )
          {
               cout << "Outputbestand: " << outputfilename <<
" kon niet geopend worden!\n";
               exit(1);
          }

          if (!createAndWriteSQL(listsFile, candidatesFile,
outputFile))
          {
               cout << "Fatal error: Program closing
down...\n";
               exit(2);
          }
          listsFile.close();
          candidatesFile.close();
          outputFile.close();
     }

     return 0;
}
```