Fråga E (Beskriva) Läs avsnitt 1 och 2 i artikeln [1] och svarar på följande frågor. (1) Hur ser organisationen för ett val ut? (2) Vilka är aktörerna och vilka roller har de? (3) Vilka krav ställs för att uppnå ett säkert val? (4) Vilka eventuella försök till valfusk förebygger man?

I ett val med många kandidater delas de in i {1, 2, 3, …, p}. De aktörer som är involverade i ett val är väljare, en oberoende tidstämpel och olika myndigheter som räknar rösterna. Myndigheterna delas upp i tre nivåer, lokala, regionala och nationell. Strukturen bygger på att väljaren skickar sin röst till sin lokala myndighet. Den lokala myndigheten samlar ihop alla röster i den valkretsen, kontrollerar att alla röster är korrekt utförda och skickar resultatet till den regionala myndigheten. Där ser man till att de lokala myndigheterna utfört sin uppgift korrekt och sammanställer ett regionalt resultat och skickar det till den nationella myndigheten. Där kontrolleras alla regioners resultat och rösterna dekrypteras.

För att rösta lägger väljer anonymt sina röster på en slags anslagstavla, där man kan se hur många som röstat för att försäkra sig om att rösträkningen går rätt till.

Kraven som ställs för ett säkert val är att en röst inte ska kunna spåras till en viss person(privacy), utomstående ska kunna verifiera att räkningen går rätt till (public verifiability). Det är viktigt att systemet kan tåla att en illvillig rösträknare som försöker fuska, exempelvis om någon på lokal nivå lägger till felaktiga röster, då ska det kunna upptäckas på regional nivå(robustness). Väljarna ska vara anonyma. Det ska inte gå att få att få tag på ett kvitto på sin röst, detta motverkar rösttvång. Två typer av valfusk som förebyggs är mellanhandsattacker, att en myndighet försöker att förfalska ett valresultat, och en rusningsattack, att anhängare till en kandidat väntar med att lägga sin röst tills de lokala myndigheterna avslöjat sina resultat.

För att undvika mellanhandsattacker måste alla tre myndighetsnivåer vara med och skapa nycklarna.

Fråga D (Redogöra) Läs avsnitt 3 i artikeln [1]. (1) Redogör för nyckel-konstruktion, kryptering och dekryptering i Pailliers kryptosystem. (2) Beskriv vad som menas med homomorfisk kryptering och förklara vilken homomorfisk egenskap som Pailliers kryptosystem har. (3) Varför behövs ett tröskelsystem i röstningsprotokollet och på vilket sätt implementeras ett sådant här? (4) Vilka typer av bevis av kunskap diskuteras i artikeln?

Pailliers kryptosystem bygger sig på en form av RSA modul, där nyckelkonstruktionen tar inspiration från RSA men bygger vidare på den. För att konstruera en nyckel behövs det två primtal, $p, q \in \mathbb{Z}$, man sätter

$$p \cdot q = N$$

Där N är en produkt av de två valda primtal. Sedan tas en heltals-multipel av N modulo N^2 och sätter den som g. Med detta kan man konstruera den allmänna krypteringsnyckeln PK=(N,g), där

PK står för public key. För att konstruera en privat dekrypteringsnyckel definieras en ny variabel $\lambda(N)$, som lyder

$$\lambda(N) = \backslash operatornamelcm\big((p-1)(q-1)\big),$$

Där den privata nyckel PK(private key) definieras som PK=$\lambda(N)$. Med detta kan man nu påbörja själva krypteringen. För att kryptera sin klartext $M \in \mathbb{Z}_N$, behöver man först välja en slumpmässig $x \in \mathbb{Z}_N^*$, När x är definierad utförs krypteringen på följande vis

$$c = g^M \cdot x^N \ (mod\ N^2).$$

Sista steget är att kunna dekryptera c. För att kunna utföra dekrypteringen definieras en ny funktion

$$L(u) = \frac{U-1}{N},$$

Där L funktionen tar in värden u, som ges av

$$S_N = \{u < N^2 | \ u = 1 \ (mod\ N)\},$$

När u och L(u) är bestäm kan själva dekrypteringen ta form. För att dekryptera meddelandet följer

$$M = \frac{L\left(c^{\lambda(N)}\right)(mod\ N^2)}{L\left(g^{\lambda(N)}\right)(mod\ N^2)} (mod\ N).$$

Det som gör denna krypteringsmetod användbar i ett elektroniskt röstningssystem är de homomorfiska egenskaperna. Ett homomorfiskt kryptosystem betyder att man kan göra matematiska beräkningar på de krypterade meddelandet utan att förstöra klartexten, då kryptogrammet bär samma egenskap och struktur som klartexten. Varför detta är användbart i ett elektroniskt röstningssystem är just på grund av att vi inte behöver dekryptera klartexten för att utföra beräkningar på rösterna. Så man kan exempelvis beräkna antal röster och få fram ett resultat utan att behöva dekryptera varje röst för sig själv, detta är användbart då det vidhåller varje deltagares integritet. I palliers kryptosystem är den homomorfiska egenskapen

$$E(M_1 + M_2) = E(M_1) \cdot E(M_2).$$

För att undvika att myndigheterna skall kunna läsa rösterna men också för att lägga till ett extra lager skydd för användarnas integritet införs ett tröskelsystem i röstningsprotokollet. Tröskelsystemet fungerar på så vis att man delar ut flertal dekrypteringsnycklar till de olika myndigheterna, och för att kunna dekryptera rösterna så måste ett bestämt antal myndigheter gå tillsammans och använda sina nycklar. Detta skyddar myndigheterna från att kunna läsa av rösterna eller manipulera dom då en enstaka myndighet själv inte kan dekryptera meddelandet. Så för att dekryptera ett meddelande krävs det hemliga nycklar som har genererats av servarna i samband med krypteringsnyckeln. Efter det genereras även verifieringsnycklar. Sedan för att få fram klartexten ur kryptogrammet används dekrypteringsnyckeln och verifieringsnyckeln på klartexten. Till sist används en kombinationsalgoritm som sätter ihop de olika dekrypterade

meddelanden och det är även här man ser ifall det är tillräckligt många myndigheter har varit med för att få fram en korrekt klartext. Exempelvis om det finns n stycken myndigheter och sedan behövs det t stycken myndigheter föra att dekryptera ett meddelande, så kommer kombinationen ge fel ifall exempelvis bara 1 eller t-1 myndigheter försökt dekryptera kryptogrammet.

En viktigt den av det digitala röstningssystemet är att kunna verifiera alla röster som kommer in, men man vill fortfarande förvara själva rösten hemlig. Därför används olika typer av bevis av kunskap, även kallat "zero-knowledge proof". Det är ett sätt att kunna verifiera ett påstående utan att känna till själva påståendet. I detta system används det alltså för att kunna verifiera en röst utan att veta själva rösten. I artikeln diskuteras det 3 olika metoder för att uppnå detta. Den första är "Proof of knowledge of an encrypted message" eller bevis på kunskap om ett krypterat meddelande, vilket är en metod att kunna verifiera rösten utan att faktiskt ha tillgång till själva svaret. Utan med hjälp av kryptogrammet och genererade värden kan du verifierar rösten. Den andra metoden som diskuteras är "Bevis på att ett krypterat meddelande ligger i en given uppsättning meddelanden". Den tyder att det går att lägga till ett bevis som verifierar att rösten ligger i en publik lista S = $\{m_1, m_2, \dots, m_p\}$ med storleken p, utan att avslöja någon annan information om meddelandet. Den tredje och sista metoden är "Bevis på jämlikhet för klartexter" och betyder att vid kryptering av en röst är det möjligt att bifoga ett bevis på att två krypterade rösterna är lika.

Fråga C (Exemplifiera) Konstruera egna exempel på (1) kryptering och dekryptering med Pailliers kryptosystem, (2) illustration av den homomorfiska egenskapen hos Pailliers kryptosystem, (3) användning av tröskelsystem för att uppnå en sluten röstning och (4) olika bevis av kunskap.

Konstruktionen av krypteringen börjar med en nyckelgenerering. Då krävs det två primtal, säg p = 13 och q = 23. Sedan behövs ett valfritt g väljas som följer de krav vi gått igenom på D. Vi väljer g = 4886. Detta resulterar i att Den publika nyckeln blir (299,4886) och den privata blir λ(132).

```
In [3]: #Nyckelgenerering
        nyckelgenerering()
Out[3]: Ange p 13
        Ange q 23
        välj ett slumpmässigt heltal ur Z
        4886
        inversen existerar och är 102
        Det publika nyckeln blir (299,4886)
        Det privata nyckeln blir λ(132)
```

Sedan kommer krypteringen. I exemplet är rösten 0 ett nej och rösten 1 är ett ja. Vi påstår att vi röstar ja på något. Då blir alltså vår klartext m lika med 1, sedan väljer vi ett valfritt x, i detta exempel 59. Då blir kryptogrammet c lika med 53049.

```
In [4]: #Kryptering
        N = int(input("Ange N"))
        g = int(input("Ange g"))
        c = krypteringavm(N,g)
        print("kryptogramet c är lika med: {}".format(c))
Out[4]: Ange N 299
        Ange g 4886
        välj ett valfritt x i Zn
        59
        Ange din klartext 1
        kryptogramet c är lika med: 53049
```

För att sedan få fram klartexten skall man nu dektryptera kryptogrammet. Så om man anger c,lambda, N och g får vi tillbaka klartexten m.

```
Ange c 53049
Ange λ 132
Ange N 299
Ange g 4886
Klartexten är: 1
```

Det som gör palliers kryptosystem så bra är även de homomorfiska egenskaperna, $E(M_1 + M_2) = E(M_1) \cdot E(M_2)$. De kan lättast redovisas med ett enkelt exempel på två klartexter, m1 = 123 och m2 = 456. Deras kryptogram blir då c1 = 13250 och c2 = 33447 när vi använt primtalen p,q = 13,17, och de valda talet 4886. då blir vår privata nyckeln 48.

```
In [12]: #Homomorphic
m1 = int(input("Ange m1"))
m2 = int(input("Ange m2"))
c1 = int(input("Ange c1"))
c2 = int(input("Ange c2"))
λ = int(input("Ange λ"))
N = int(input("Ange N"))
g = int(input("Ange g"))
homomorphicprof(m1,m2,c1,c2,λ,N,g)
```

```
Out[12]: Ange m1 123
Ange m2 456
Ange c1 13250
Ange c2 33447
Ange λ 48
Ange N 221
Ange g 4886
E(m1)*E(m2) = 13250*33447 = 443172750 (mod 48841)
E(m1 + m2) = 123+456 = 579 (mod 221)
om man dekrypterar 443172750 får man: 137
Och E(m1)*E(m2) = E(m1 + m2) ==>  137 = 123 + 456 (mod 221)
```

Bilden ovan visar en tydlig visning av hur den homomorfiska projektionen fungerar. 123+456 (mod 221) = 137, och 137 får vi från att ta 13250*33447 och sedan krypterar det.

För att sedan visualisera bevis på kunskap där kommunikationen sker fram och tillbaka mellan P och V, så delade man in det i 4 steg. 1a steget är att bidra med ett u från P till V för att påbörja kontakten. Steg två är sedan för V att ge P ett valfritt värde e som skall utmana P´s påstående. 4de och sista steget är för V att utföra en form av kryptering för att se ifall han får samma kryptogram som u. Så steg 1 börjar med att välja ett x och s ur Z respektive Z_N^*

```
Out[5]: Vad är N 221
Vad är g 4886
välj ett x ur Zn 23
välj ett s ur Z*n 46
u valde att x är 23
u valde att s är 46
ditt u blir då 40067
```

Ur ett testperspektiv användes ganska små värden för att lättare kunna felsöka. I detta fall användes N 221 och g 4886 då vi redan vett att det är giltiga värden utifrån tidigare exempel. Sedan valde vi x som 23 och s som 46, vilket programmet verifierar är giltiga. Sedan väljs en utmaning e:

```
Out[1]: Välj en valfri utmaning e ∈ [0, A[
        12
        e = 12
```

Som blev 12 i detta test. Nu krävs steg 3, där med hjälp av e och den påstådda m skall skapa en verifieringsnyckel som skall fungera i krypteringen i steg 4.

```
Out[10]: Vad är N 221
         Vad är r 59
         Vad är g 4886
         Vad var ditt x? 23
         Vad var ditt s? 46
         Vilket e fick du tilldelat?
         12
         Vad tror du m är? 1
         v = 11
         w = 124
```

När P utför beräkningarna på V´s utmaning resulterar det i ett v värde på 11, och ett w värde på 124. Sedan skickar P dessa resultat till V, som sedan skall verifiera om det stämmer överens med det givna kryptogrammet u som gavs i början.

```
Out[2]: Vad är N 221
        Vad är g 4886
        Vad är c 35338
        Vilket e har du get ut?
        12
        Vad fick du för v? 11
        Vad fick du för w? 124
        Vad fick du för u i början?
        40067
        Du kan lita på att P känner till klartexten
```

I detta exempel ser vi att uträckningen matchar överens med u värdet och vi kan då anta att P faktiskt känner till klartexten m.

Vid programmeringen av tröskelversionen av Pailliers kryptosystem uppstod några problem. Eftersom de här beräkningarna kräver att man använder potenser ofta så måste man till slut arbeta med väldigt stora tal. På grund av detta så är det svårt att få skripten att fungera på en vanlig pc, ofta uppstår recursion depth error eller int overflow error. Särskilt i combining algorithm kan det bli problem då man ska beräkna stora tal med stora exponenter. För att få **μ behöver man ett stort *n* när man beräknar delta-värdet (delta = *n!*).**

Fråga B (Tillämpa) Läs avsnitt 4 i artikeln [1]. Beskriv med egna ord hur ett val och efterföljande rösträkning går till.

Ett val genomförs genom att de olika myndigheterna skapar publika nycklar, *PK* för nationell nivå, PK_i och $PK_{i,j}$ för regional respektive lokal nivå. ℓ noterar antalet röster. *M* väljs som ett tal större än ℓ .

För att rösta hämtar man nycklarna för sin valkrets. Man krypterar sedan sin röst en gång med respektive nyckel, detta resulterar i tre kryptogram. För att kunna försäkra sig om att ens röst har räknats med och är giltig skapas tre olika bevis. En giltig röst är ett heltal M^m där *m* tillhör mängden av alla kandidater. Man skapar även ett bevis som visar att kryptogrammen kommer från samma valsedel och krypteras från samma klartext.

Väljaren laddar upp sin valsedel på valkretsens anslagstavla. Då sätts en tidsstämpel på valsedeln

(lägg till exempel tabell)

Väljare	C_l	C_r	C_n
Väljare 1	$g_{i,j}^{v_{i,j,1}}$	$g_i^{v_{i,j,1}}$	$g^{v_{i,j,1}}$
Väljare 2	$g_{i,j}^{v_{i,j,2}}$	$g_i^{v_{i,j,2}}$	$g^{v_{i,j,2}}$
Väljare *k*	$g_{i,j}^{v_{i,j,k}}$	$g_{i,j}^{v_{i,j,k}}$	$g_{i,j}^{v_{i,j,k}}$
Väljare ℓ	$g_{i,j}^{v_{i,j,\ell}}$	$g_i^{v_{i,j,\ell}}$	$g^{v_{i,j,\ell}}$
Summa för $A_{i,j}$	$\sum_k v_{i,j,k}$	$\prod_k g_i^{v,j,k}$	$\prod_k g^{v,j,k}$

De lokala rösträknarna börjar rösträkningen att säkerställa alla valsedlar som kommit in. När det är gjort räknar man ut produkten av alla giltiga röster i valkretsen. Man dekrypterar produkten av kryptogrammen med den lokala publika nyckeln, $PK_{i,j}$. På regionens anslagstavla publiceras det lokala, dekrypterade resultatet, produkten av kryptogrammen som skapades med de regionala respektive nationella nycklarna. Räkningen fortsätter på regional nivå där dekrypterar man produkten av produkterna från de lokala räknarnas anslag tillhörande den regionala nyckeln, PK_i. Detta publiceras på den nya produkten tillsammans med produkten av produkterna för de nationella kryptogrammen.

Valkrets	$V_{j,k}$	$\prod_{j,k} C_r$	$\prod_{j,k} C_n$
Valkrets 1	$\sum_k v_{i,1,k}$	$\prod_k g_i^{v_{i,1,k}}$	$\prod_k g^{v_{1,j,k}}$
Valkrets 2	$\sum_k v_{i,2,k}$	$\prod_k g_i^{v_{i,2,k}}$	$\prod_k g^{v_{2,j,k}}$
Valkrets *j*	$\sum_k v_{i,j,k}$	$\prod_k g_i^{v_{i,j,k}}$	$\prod_k g^{v_{i,j,k}}$

Summa för A_i	$\sum_{j,k} v_{i,j,k}$	$\prod_{k} g_i^{v_{i,j,k}}$ Dekrypteras till $\sum_{j,k} v_{i,j,k}$	$\prod_{k} g^{v_{i,j,k}}$

De lokala och regionala resultaten kan bekräftas genom att man jämför dekrypteringarna med varandra. När de lokala resultaten är verifierade skickas produkten för de regionala resultaten vidare till nationell nivå.

regioner	$V_{i,j,k}$	$\prod_{i,j,k} C_n$
Region 1	$\sum_{j,k} v_{1,j,k}$	$\prod_{j,k} g^{v_{1,j,k}}$
Region 2	$\sum_{j,k} v_{2,j,k}$	$\prod_{j,k} g^{v_{2,j,k}}$
Region *i*	$\sum_{j,k} v_{1,j,k}$	$\prod_{j,k} g^{v_{i,j,k}}$
Summa för *A*	$\sum_{i,j,k} v_{i,j,k}$	$\prod_{i,j,k} g^{v_{i,j,k}}$ dekrypteras till $\sum_{i,j,k} v_{i,j,k}$

På nationell nivå dekrypteras produkterna för kryptogrammen tillhörande den nationella nyckeln, *PK*. Resultatet verifieras genom att jämföra med Summan tillhörande det regionala resultatet.

Genom att rekursivt beräkna trösklarna kan en väljare försäkra sig om att ens röst är räknad i sin valkrets och att valkretsen är räknad regionalt osv. Detta är en väldigt enkel uträkning som en helt vanlig dator klarar av att beräkna på en ganska kort tid.

Fråga A (Analysera) Läs avsnitt 5 i artikeln [1]. Visa med ett exempel hur man kan avslöja valfusk. Finns någon typ av valfusk som protokollet inte kan upptäcka?

Ett exempel på valfusk som protokollet skyddar mot är att en väljare röstar mer än en gång, eftersom varje väljare tilldelas en nyckel så kan man i resultaten se om en nyckel används två eller fler gånger. Detta är en blind signatur, det går inte att se vem som är avsändare, men det är fortfarande verifierbart av rösträknarna. Nyckeln som används för blindsignaturen är skapad med hjälp av alla nivåer i rösträkningen för att förhindra att en väljare får tillgång till mer än en nyckel. För att skapa en väljarpseudonym måste man gå igenom ett par steg. Den publika nyckeln är ett stort heltal e och $Z \in \mathbb{Z}_N^*$. Den hemliga nyckeln är paret (x_i,y_i), där $x_i \in \mathbb{Z}_N^*$ och $y\_i \in [0, e[$ sådana att $Z = \prod x_i^e \, a^{y_i} mod\ N$

Varje nivå väljer ett slumpmässigt valt tal u_i tillhörande $\mathbb{Z}_N^*$ samt v_i tillhörande [0, e[som skickas till väljaren i $w_i = u_i^e a^{v_i}$ mod *N*. Väljaren tar slumpmässiga tal, $\beta \in \mathbb{Z}_N^*$ och $\alpha, \gamma \in [0, e[$. dessa tal beräknas som $w = (\prod w_i) a^\alpha \beta^e Z^\gamma \ mod\ N$. Innan väljaren skickar in till myndigheterna beräknas en

utmaning $\varepsilon = H(w, V)$, $c = \varepsilon + \gamma \, mod \, e$. varje nivå räknar $r_i = v_i + cy_i \, mod \, e$, $s_i = (v_i + cy_i) \div e$ och $t_i = u_i x_i^e a^{s_i}$, r_i och t_i skickas tillbaka till väljaren.

Slutligen beräknas $r = \sum r_i + \alpha \, mod \, e$, $s = (\sum r_i + \alpha) \div e$, $s' = (c - \varepsilon) \div \, mod \, e$ och $t = \prod t_i \, \beta a^s Z^{-s'} \, mod \, N$. detta leder att man inte kan rösta mer än en gång utan att samarbeta med samtliga nivåer i rösträkningen.

Protokollet skyddar mot nästintill alla former av fusk som kan tillkomma i en digital miljö. Men något som protokollet inte kan skydda mot, som också är ett av de största hoten är att faktiskt verifiera röstaren som person. Då tanken med protokollet är att skapa en digital röstning, så tar man faktiskt bort den största verifieringen som används i dagsläget, och det är när man röstar på plats så verifierar man ifall röstaren är samma person i verklighet som han är på sitt ID. Så något detta protokoll inte tar i hänsyn till är ifall jag är samma person på ID som röstar. Protokollet kan alltså inte verifiera ifall jag har tagit någon annans verifiering och röstat åt den personen, och då vi använder Receipt-Freeness kan man häller inte få någon verifiering av att du har röstat. Samt Incoericibility, som egentligen skyddar från att du inte ska kunna kalla tillbaka en röst och sälja den, men nu också stoppar dig från att kunna kalla tillbaka en röst där någon röstat åt dig.