Principper i moderne blok-kryptering

Med udgangspunkt i FIPS-197-standarden AES, baseret på Rijndael-algoritmen

Af Mathias Vestergaard

Figur 1 | "Security matters" | Marco Javier Guerrero

FORORD

Denne opgave er skrevet som det almene gymnasiums større skriftlige opgave, SSO, i faget **matematik**.
Emneområdet er kryptologi, og opgaveformuleringen lyder:

> *AES kryptosystem.*
> *Der ønskes en beskrivelse af Rijndael-algoritmen. Beskrivelsen skal fremstå på en sådan måde, at man uden den store matematiske indsigt kan forstå tankegangen i algoritmen og få en fornemmelse af dette kryptosystems virkemåde også set i et historisk perspektiv.*
>
> *Der ønskes endvidere en mere matematisk præsentation af nogle af de algebraiske begreber man støder på, når man beskæftiger sig med Rijndael-algoritmen.*

Opgaven er, ligesom titlen, delt op i to dele, hvor den første del netop kan læses uden den store matematiske indsigt. Den anden del kan også læses uden den store matematiske indsigt, men fordres dog af en matematisk forståelse. Teksten er formuleret som en vekselvirkning mellem prosa og matematiske formler, og de to dele af opgaven knyttes til hinanden.

De algebraiske begreber der beskrives i anden del af opgaven er: **algebraiske strukturer, semigrupper, grupper, ringe** og **legemer** – især **endelige legemer**, der også **kaldes Galois legemer** – samt begreberne **homomorfi** og **isomorfi**.
Herunder beskrives **addition** og **multiplikation af polynomier i Galois legemer**.
Rijndael-algoritmens forskellige del-funktioner beskrives i detaljer, herunder: **SubBytes, AddRoundKey, ShiftRows** og **MixColumns**. Funktionen KeyExpansion beskrives kun konceptuelt, da denne ikke indeholder noget specielt interessant matematik.
Kryptologisk set beskæftiger opgaven sig primært med **hemmeligholdelse af data**. Autenticitetsbekræftelse, nøgleudveksling og kryptanalyse nævnes blot i sammenhængen – evt. ledsaget af litterære henvisninger.

Hensigten med opgaven er, udover at være en del af min studentereksamen, samtidig at bidrage med en tekst, der på dansk **forklarer principper i den mest moderne form for kryptering,** der i dag anvendes i praksis, og som kan læses af alle, der bare har en smule matematisk indsigt.

Jeg referer i øvrigt til en del kilder på Internettet, samt "svært tilgængelige" fagtekster, som man ikke altid bare kan låne på biblioteket. Derfor findes der sammen med denne opgave en cd med dette kildemateriale i pdf-format. Mangler denne, kan du altid henvende dig via e-mail. Så vil jeg fremsende kilderne.

Opgaven gør rigt brug af noter, hvilket ikke er for at forstyrre eller ødelægge læsningen, da opgaven sagtens kan læses uden at kigge i noterne. Noterne bruges for det første til at dokumentere påstande i kilder, og sekundært til at uddybe stof og overvejelser som falder lidt udenfor opgavens egentlige problemstilling.
Hvis du ønsker at læse noterne, er notehenvisningerne i teksten desuden tydeligt markeret, så man let kan finde tilbage til hvor man kom fra i teksten.

Eksempler på formatering:

Bitkode:	`{10101010}`
Hexadecimal:	**AA**
Kildehenvisning:	[Singh]

Omfanget af selve opgaven, ekskl. noter, bilag, litteraturliste, forord og indholdsfortegnelse, er beregnet til 6000 ord, svarende til 15 sider, efter de angivne normer.

Mathias Vestergaard
mathias@mtproductions.dk
Århus Statsgymnasium | 21. december 2004

INDHOLDSFORTEGNELSE

HVORFOR KRYPTO-LOGI?

Kryptologi handler dybest set om kontrol med informationer. Vha. kryptologi kan man fx sikre, at det kun er den rette modtager af en meddelelse, der er i stand til at læse indholdet, dvs. en hemmeligholdelse for alle andre. Man kan også sikre, at det indhold, vedkommende læser, ikke er blevet ændret undervejs, og at modtageren kan være sikker på, at afsenderen er, hvem han giver sig ud for at være, dvs. en bekræftelse af indholdets og afsenderens autenticitet. Begge dele brugbart. Men ligesom der skal en gift til at udvikle modgift, har kryptologien også en modsatrettet side – kryptanalysen, som netop går ud på, at bryde denne kontrol, dvs. at finde metoder til at læse og evt. ændre det, som kun er beregnet til andre, eller til at sende meddelelser, der giver sig ud for at komme fra en anden.

Det kan let virke en smule overdrevet at tale om al denne hemmeligholdelse og sikkerhed. Når samtalen falder på kryptologi, møder jeg tit kommentaren: "jamen jeg har ikke noget der er så hemmeligt." Det må klart være et produkt af vores samfunds forherligelse af ærligheden, for der er tilsyneladende noget, folk glemmer: vi er på vej ind i en digital tidsalder. Når vi sender et alm. brev med posten i dag, sætter vi vores lid til Post Danmark, men på Internettet er der ikke noget "postvæsen", som man kan stole på. Alle mekanismer i det "digitale postvæsen" kan på den ene eller anden måde bestikkes, misbruges eller forfalskes. Hvis vi skal udnytte de muligheder, Internettet giver, er kryptologi en fundamental nødvendighed for succes. Når man betjener sin netbank, skal man være sikker på, at det faktisk ER den bank, man tror det er. Når man betaler med sit dankort, skal man være sikker på at oplysningerne ikke opsnappes af andre, og i øvrigt ligesom med netbanken, at de sendes til den rette modtager.

Kryptologi – definition:
Videnskaben om at udvikle og konstruere kryptosystemer, og videnskaben om at bryde disse kryptosystemer, kryptanalyse.

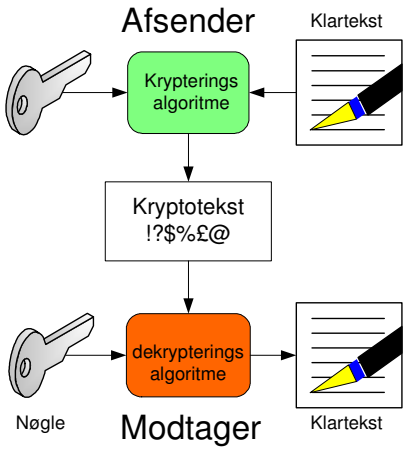
FØRSTE DEL

Kryptologiske begreber og forudsætninger

For den videre læsning af opgaven, forudsættes et grundlæggende kendskab til bits, bytes og hexadicimale koder. Start evt. med at læse bilag 1. Og lad os så i øvrigt få styr på nogle begreber:

Kryptosystemer

Lad os først se på begrebet 'kryptosystem'. Et kryptosystem er et generelt udtryk for en systematiseret proces, der ud fra en valgt nøgle, gør det muligt at omdanne information (klartekst), til noget volapyk (kryptotekst), der for udenforstående, uden kendskab til nøglen, er umuligt at forstå[1]. Denne proces kaldes for kryptering[2]. Den modsatte proces, hvor kryptoteksten laves tilbage til klartekst ud fra samme nøgle, kaldes dekryptering[3].



Figur 2 | Kryptosystem

Matematisk definition på et symmetrisk kryptosystem[4]:
$\boldsymbol{P}$ er en mængde af mulige klartekster
$\boldsymbol{C}$ er en mængde af mulige kryptotekster
$\boldsymbol{K}$ er en mængde af mulige nøgler
Til enhver nøgle $k \in \boldsymbol{K}$ skal vi have en krypteringsfunktion e(k,m), som er en afbildning af $\boldsymbol{P}$ ind i $\boldsymbol{C}$, og en dekrypteringsfunktion d(k,m) som er en afbildning af $\boldsymbol{C}$ ind i $\boldsymbol{P}$. For enhver klartekst $m \in \boldsymbol{P}$ skal der gælde at d(k,e(k,m)) = m.

Tænk på det som en slags kiste med hængelås. I kisten placeres en stak vigtige papirer(klarteksten) og låsen sættes på. Nu er det kun dem der har den rigtige nøgle, der kan låse den op.
Et andet formål med et kryptosystem kan være at sikre informationernes autenticitet, dvs. om de er blevet ændret undervejs, hvilket jeg dog ikke vil behandle i denne opgave. Bemærk dog at netop dette felt er mindst lige så afgørende for kryptologi i praksis[5].

---

[1] Umuligt er ikke nødvendigvis absolut, men blot et udtryk for at det er så svært og vil tage så lang tid, at det bliver absurd.

[2] I litteraturen bruges også ordet enkryptering, fra det engelske encryption, men jeg foretrækker ordet kryptering uden 'en'.

[3] Ikke at forveksle med kryptanalyse, hvor nøglen netop ikke er kendt.

[4] [Erlandsen] s. 18.

[5] Jf. indledningen og [Erlandsen] s. 2.

Typer af kryptosystemer

Der findes flere typer af kryptosystemer. Min allegori med kisten ovenfor, viser funktionen af et symmetrisk kryptosystem, hvor man bruger samme nøgle til at låse i og låse op, det afhænger blot af hvilken vej man drejer nøglen. Der findes også en anden type kryptosystem, kaldet asymmetriske kryptosystemer. Hvis de skal forklares som en hængelås, så vil det være en meget speciel hængelås, da den kræver to forskellige nøgler: en der KUN kan låse, og en der KUN kan låse op. Teorien om eksistensen af et sådant system, blev første gang formuleret af Whitfield Diffie og Martin Hellman i 1976, ca. samtidig med at de samme principper blev udviklet i klassificerede militær-laboratorier[1]. Et godt eksempel på et asymmetrisk kryptosystem, er det såkaldte RSA-kryptosystem[2]. RSA er baseret på, at det er beregningsmæssigt svært at faktorisere meget store tal, samt irreversibiliteten i modulo-regning, altså regning med rest[3]. Der er skrevet adskillige forklaringer og beskrivelser af RSA. Jeg vil anbefale [Landrock] s. 101ff, til videre læsning. Her findes også beviser for den talteori, der ligger bag. Andre asymmetriske kryptosystemer er Merckle-Hellmann og El Gamal, som begge er beskrevet i [Savard] kapitel 5.

Billede 1 | De tre hoveder bag RSA – 36 år efter

Fra venstre: Ronald Rivest, Adi Shamir og Leonard Adleman

Fælles for de asymmetriske kryptosystemer er, at de alle er langsomme at bruge i praksis, sammenlignet med den sikkerhed de yder i forhold til symmetriske kryptosystemer. Derfor bruges de ofte kun til at udveksle den nøgle, som efterfølgende bruges i et symmetrisk kryptosystem som AES. Jeg vil derfor ikke behandle asymmetriske kryptosystemer yderligere men blot påpege den konceptuelle nødvendighed af dem til udveksling af en fælles nøgle.

Klartekst og nøgler

Nu har vi stiftet bekendtskab med et kryptosystem, men det kan tit give anledning til forvirring, når man taler om klartekst og nøgler. Det, man bare skal huske på, er, at alt data i en computer blot er en lang række af tal mellem 0 og 255. På bitform fra `{00000000}` til `{11111111}` eller på hexadecimal, fra `00` til `FF`.

I denne sammenhæng er informationer og nøgler altså blot store tal, som vi kan regne på med vores matematik.

---

[1] For videre læsning om opdagelsen af denne type kryptosystem henvises til [Singh].

[2] RSA er initialerne på de tre matematikere: Rivest, Shamir og Adleman. Historien lyder at det skulle have hedet ARS, altså sorteret alfabetisk, men Adleman, der havde været med i grundforskningen der førte frem til opdagelsen, men som ikke havde været med i den egentlige opdagelse af selve systemet, var meget beskeden, og foreslog at A'et blev flyttet om bagerst. [Singh] s. 287f.

[3] Hvis man har et tal X, delt med en anden tal Z og man får en rest der hedder Q, er det ikke muligt, entydigt at bestemme hvad X var, selvom man kender både Q og Z.

Krav til et moderne kryptosystem

Der er mange ting, der skal tages i betragtning, når man ønsker at designe et kryptosystem. Jeg mener, følgende principper er fundamentale for et moderne, symmetrisk kryptosystem[1]:

- Sikkerheden i et kryptosystem skal udelukkende bero på hemmeligholdelse af nøglen. Dette kendes også som Kerckoffs princip.
- Det skal ikke være let at genskabe den hemmelige nøgle selvom man kender både klartekst og tilhørende kryptotekst.
- Nøglen skal være af en længde der gør det praktisk talt umuligt at prøve alle nøgler.
- Kryptoteksten skal være modstandsdygtig overfor kendte typer af angreb.
- Systemet skal være sikret mod valg af "uheldige nøgler", dvs. nøgler der medfører at styrken af kryptoteksten forringes.
- Kryptosystemet skal være let for afsender og modtager at bruge.

Kerckoffs princip:
"en angriber kender det anvendte kryptosystem."
[Erlandsen] s. 7

Et glimrende eksempel på en moderne krypteringsalgoritme som IKKE lever op til disse krav, er WEP-krypteringen der bl.a. bruges i trådløse netværk efter 802.11b-standarden, bedre kendt som WiFi[2].

Historisk

Kravene til et moderne kryptosystem er ikke opstået ud af ingenting, men bygger på tidligere tiders erfaringer, og derfor er det relevant, kort at beskæftige sig med kryptologiens lange historie.
Kryptosytemer har været brugt lige så længe, som skriftsproget har eksisteret, og har været forsøgt brudt lige så længe. [Singh] beskriver hele forløbet, fra egypternes hieroglyffer, og frem til opdagelsen af RSA, og slutter til sidst med at skue ind i fremtiden, ved, i flygtige vendinger, at nævne begreber som kvantekryptering. Samtidig beskriver han, hvordan man har forsøgt at bryde kryptosystemerne, og hvilke historiske konsekvenser det har fået for forskellige personer gennem hi-

[1] Jeg har forgæves ledt efter en liste over, hvad andre har formuleret som krav, men er endt med selv at forklare det ud fra hvad jeg husker vi drøftede i en klasse-diskussion i efteråret 2004.

[2] På bilag 4 har jeg ud fra [Fogie] kort beskrevet hvorfor netop denne standard er meget uheldigt konstrueret.

storien[1]. Matematikken i bogen er stort set fraværende, men det er absolut spændende læsning fra start til slut. Jeg vil her meget kort ridse historien op, frem til i dag.

Nogle af de første eksempler på mystiske koder, er egypternes hieroglyffer, som har optaget arkæologer i århundreder. Disse kryptosystemer, er ikke kryptosystemer efter moderne definitioner, da de ikke bygger på nogen hemmelig nøgle, men blot bygger på, at det kun er afsender og modtager, der kender de forskellige tegns betydning[2].

Når vi kommer lidt længere op i historien, støder vi på Cæsars ciffer, eller Cæsar-kode. Princippet er her, at man forskyder alfabetet en eller flere pladser. Antallet af pladser, der skal forskydes, er nøglen. Når man krypterer, forskydes alfabetet den ene vej, og når man dekrypterer, forskydes det samme antal pladser i modsat retning. Dette system lever på sin vis op til Kerckoffs princip, om at man ikke skal kunne bryde systemet, bare fordi man ved hvordan, det fungerer: selvom man ved, at man skal forskyde alfabetet, kan man ikke umiddelbart genskabe klarteksten, da man ikke ved hvor langt, man skal forskyde det. Cæsar-koden falder til gengæld på at der højst kan være 28 nøgler[3], hvoraf den ene er 0, dvs. ingen forskydning, og derfor må forkastes. Der er altså blot 27 forskellige nøgler, som skal prøves af. Og faktisk kan det gøres endnu mere elegant: ved at studere frekvensfordelingen af de forskellige bogstaver, vil man få en profil, der alt efter hvilket sprog klarteksten er skrevet på, vil have nogle karakteristika, der kan sammenlignes med frekvensfordelingen for en generel klartekst skrevet på det givne sprog. På dansk er bogstavet 'e', det hyppigst forekomne i en tilfældig tekst af en vis længde. Ved at finde den højeste top i frekvensfordelingen i kryptoteksten, og se hvor langt den er forskudt fra e'et (bogstav nr. 5), har man allerede et godt bud på nøglen. Dette angrebsprincip kaldes også frekvensanalyse, fordi man analyserer på frekvensen af de forskellige bogstavers forekomster[4].

Efter Cæsar-koden følger en hel række modifikationer af denne, men det er alt sammen småting, der ikke ændrer væsentligt på Cæsarkodens svaghed. Det er først omkring midten af 1500-tallet da Blaise de Vigenère skaber et såkaldt polyalfabetisk kryptosystem, at der virkelig sker

---

[1] Hans udlægning af engelske og amerikanske kodebrydere under Anden Verdenskrig bliver dog en smule sentimental og heroisk, så man ikke kan undgå at fælde en lille tåre undervejs.

[2] En anden type kodesystemer går under betegnelsen stenografi. Her skjuler man blot meddelelsen på en eller anden måde. Fx kan man barbere en slave skaldet, tatovere beskeden i hovedbunden på ham, og lade hans hår vokse ud igen. Nu vil ingen opdage at han har en hemmelig besked på sig. Det kan også være mere fredeligt, fx i en pin-kode-husker til et dankort.

[3] med et dansk alfabet(uden w).

[4] Historien om brydning af Cæserkode er beskrevet i [Singh] s. 28ff, og metoden er beskrevet nøjere i [Landrock] s. 26.

en markant ændring. Fra at nøglen er et tal mellem 1 og 27, bliver nøglen nu et kodeord med langt større variationsmulighed. Desuden kan man ikke umiddelbart foretage frekvensanalyse på kryptoteksten[1]. Selvsikkert kalder han det for "Le Chiffre Indéchiffrable" – den ubrydelige kode[2], hvilket den også forblev indtil 1863, hvor Friedrich W. Kasiski fandt en metode, der kunne bruges i praksis. Det afgørende i metoden, er at man kan gætte sig frem til nøglelængden, og ud fra denne få opdelt kryptoteksten i et antal dele, som hver især er lette ofre for frekvensanalyse[3]. I 1920 beskrev William Friedman en mere matematisk/statistisk metode til brydning af Vigenères system, som bygger på det statistiske begreb "measure of roughness", – et mål for hvor jævn en fordeling er – til at bestemme nøglelængden[4]. Vigenères kryptosystem nåede aldrig at komme i brug, da det var for besværligt at bruge i praksis. Når store mængder information skulle krypteres og dekrypteres, brugte man – også efter Vigenères opfindelse var blevet kendt – fortsat Cæser-kode, til trods for at man vidste at den ikke var sikker. Grunden til at man anvendte Cæsar-kode, var kravet om, at et kryptosystem skal være let at bruge.

Netop besværligheden i brugen af disse manuelle systemer, der fælles kan beskrives som "papir og blyant"-systemer, fordi de kan konstrueres udelukkende med pen og papir, leder os videre i historien og frem til de mekaniske krypteringsmaskiner.

Den mest kendte krypteringsmaskine er den tyske ENIGMA, som blev udviklet af en tysk opfinder ved navn Arthur Scherbius til erhvervsfolk, der ønskede at sikre deres kommunikation[5]. ENIGMA kan betragtes som en mekaniseret hjælp til at gøre "papir og blyant"-systemerne lettere at bruge i praksis, da maskinen ikke laver nogen sjuskefejl. Sikkerheden i ENIGMA lå som en blanding af en todelt hemmelig nøgle, samt det faktum at maskinens opbygning var hemmelig. Dens sikkerhed levede altså ikke helt op Kerkoffs princip, da en angriber i dette tilfælde ikke kendte alle detaljer i det anvendte kryptosystem.

---

[1] Historien om Vigenère, og hvordan systemet virker, er beskrevet i [Singh] s. 59ff og kan også læses kort i [Landrock] s. 35ff.

[2] iflg. [Singh]. Iflg. [Landrock] kaldte han det: "Le carré Indéchiffrable" – det ubrydelige kvadrat.

[3] Koden var dog allerede i 1854 blevet brudt af englænderen Charles Babbage der bare aldrig fik udgivet sin opdagelse. Den spændende historie der knytter sig hertil kan læses i [Singh] s. 77ff.

[4] Jeg har desværre ikke plads til at beskrive denne metode her, men vil kraftigt anbefale et lille studie af [Landrock] s. 43ff.

[5] [Singh] s. 141.

ENIGMA blev ikke nogen succes før ca. 20 år efter, hvor den blev inddraget i det tyske militær. Her tjente den under hele Anden Verdenskrig, hvor det dog lykkedes englænderne at bryde systemet[1].
Da computeren gjorde sit indtog, ændredes hele opfattelsen af kryptosystemer, da alle grænser nu pludselig syntes brudt. Især princippet om, at systemet skal være let at bruge, mistede stort set sin betydning, da alle processer kan automatiseres i en computer. Ligeledes revolutioneredes kryptanalysen, da det nu var muligt at lave forskellige programmer og algoritmer, der kan afprøve forskellige nøgler[2]. Herved stiger behovet for en krypteringsalgoritme som er hurtig at bruge, og som samtidig er svær at bryde; selv vha. store computere.

Kryptosystemer i dag

De kryptosystemer jeg vil beskæftige mig med i denne opgave, er såkaldte blok-krypteringssystemer, hvor man ikke blot oversætter et bogstav til et andet, men samler bogstaverne i større blokke. Jeg vil tage udgangspunkt i AES – Advanced Encryption Standard, som er afløseren for den tidligere DES, Data Encryption Standard, udviklet tilbage i 1977. AES antyder blot, at der er tale om en standard. Da denne skulle vedtages, udstedte man en offentlig konkurrence og havde en mængde forskellige algoritmer i betragtning[3]. I oktober 2000 vedtog man Rijndael algoritmen, og i november 2001 lå den officielle specifikation [FIPS] klar til offentligheden[4]. Rijndael er udviklet af Joan Deamon og Vincen Rijmen, som begge er fra Belgien. Den bygger videre på mange af principperne i krypteringsalgoritmen Square. Hvorfor netop Rijndael blev valgt, har jeg ingenlunde plads til at beskrive her, men i [Twofish] kan man læse en praktisk test af de forskellige finalisters performance i forskellige sammenhænge, og på [CSRC] kan man finde adskillige dokumenter, noter, analyser og vurderinger af de forskellige algoritmer. Her vil jeg nøjes med at citere David Aucsmith, som er ansvarlig for Intels sikkerhedsstruktur:

> "Jeg har aldrig oplevet en mere retfærdig, velovervejet og betimelig proces,... Nu kan vi i branchen begynde at inkorporere AES-

[1] Denne historie er nok den mest interessante i [Singh] kapitel 3 og 4, fordi den giver et godt billede af hvordan den omvendte proces, kryptanalyse, foregår i praksis.

[2] Jeg vil endda vove den påstand at mange af de første skridt i retningen af at opdage computeren, netop sker i forbindelse med kryptanalysen i det 20. århundrede.

[3] Disse er alle beskrevet i detaljer på [CSRS], og i [Savard] kapitel 4.

[4] Rijndael udtales næsten som en dansker naturligt ville gøre. Fonetisk vil jeg skrive det 'raindaal'. Hvis du vil høre hvordan det lyder er der på [fan] en udtale der kan downloades.

> algoritmen i vore produkter i sikker forvisning om, at vi anvender en åben, robust og gennemprøvet løsning,.."

[JP], og i samme artikel kan man videre læse:

> ”Peter Landrock er også tilfreds med Rijndael. Han mener, det er en tiltrækkende algoritme, der er elegant og nem at implementere og teste. Cryptomathic kunne tilbyde produkter baseret på AES fire dage efter, at vinderen var offentliggjort.
> - AES er ikke en revolution eller et nyt, banebrydende princip. Det er en ny generation med en masse praktiske fordele. Blandt andet tilbyder AES hurtig kryptering, også på små chip, hvor DES nærmest er umulig at have med at gøre. Det får også betydning for mobiltelefoner, lyder hans vurdering.
> Rijndael er udviklet af J. Daemen og V. Rijmen fra Leuven Universitet i Belgien. Det er bemærkelsesværdigt, at Nist dermed har valgt en ikke-amerikansk krypteringsmetode.”

En stor fordel ved Rijndael, som de to ophavsmænd selv pointerer i [Answer] er at der udelukkende anvendes simple operationer. Det gør algoritmen meget gennemskuelig, hvilket minimerer risikoen for, at man senere opdager nogle hidtil usete sikkerhedsproblemer.

Blok-kryptering med Rijndael

Ud fra [Masnyk], [Savard] og [FIPS] vil jeg i dette afsnit gennemgå Rijndael-algoritmen på et overordnet plan, hvor jeg introducerer den generelle opbygning og forklarer de forskellige dele med små allegorier, der selvfølgelig ikke er helt præcise rent matematisk, men til gengæld giver god forståelse, af hvad der sker, uden at stille krav om matematisk indsigt. I næste del af opgaven stiller jeg skarpt på matematikken, der ligger til grund for algoritmen, og forklarer så alle delene én gang til, men denne gang i matematisk optik.

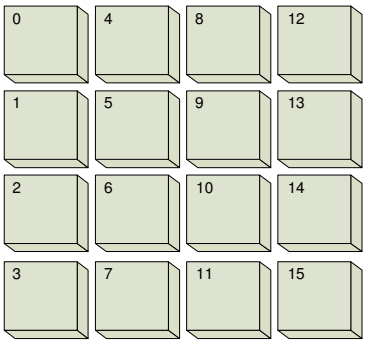
Inputs og outputs

Hvis vi tænker Rijndael-algoritmen som en stor maskine, så skal den jo have vores tekst ind i den ene ende: et input. Lad os tænke det således: Vi har vores information som vi ønsker at kryptere. Hvert bogstav i teksten svarer til en lille, kvadratisk træklods, der er nummereret med tal fra 0 til 255. Vi omsætter vores tekst til tal, fx efter ASCII-tabellen. Se bilag 1.[1]

[1] Det er nu ikke afgørende om det lige er den ene eller anden måde vi oversætter fra bogstaver til tal. Det afgørende er bare at vi bruger samme system når tallene er blevet dekrypteret, og skal omsættes tilbage til bogstaver.

Når vi har oversat vores tekst til træklodser med tal på, lægger vi først klodserne op i en lang række. Dette er vores klartekst, som vi nu vil kryptere med Rijndael ved at putte den ind i maskinen.

Det første maskinen gør, er at dele rækken op i mindre rækker, som hver består af 16 klodser. Så tager den de første 16 træklodser og lægger dem op i et kvadrat. Den starter med at placere den første klods i øverste venstre hjørne. Næste klods placeres lige under, og den fortsætter, så den får lagt brikkerne op som på figur 3. Dette er den første blok, der skal krypteres.



Figur 3 | opstilling af klodser i kvadrat

På præcis tilsvarende måde laves en række af klodser, som udgør nøglen, og som også fodres til maskinen. Denne må i dette tilfælde dog kun bestå af 16 klodser, altså ét kvadrat.

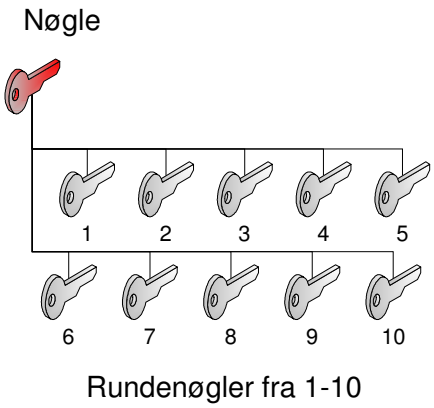
Kryptering af en blok med Rijndael

Maskinen har altså nu et 4x4-kvadrat med nummererede træklodser, der udgør første blok af vores klartekst og et tilsvarende kvadrat, der udgør nøglen. I maskinen findes desuden en kæmpe kasse med uendeligt mange løse klodser.

Maskinen gennemløber nu en lang række processer. Her følger en beskrivelse af, hvad hver proces gør, og bagefter beskrives hvordan processerne kombineres[1].

KeyExpansion

Ud fra nøglen, dvs. vores lille kvadrat med de 16 små klodser, skal der laves en række runde-nøgler. Dette har bl.a. til hensigt at forebygge, at man kan komme til at vælge et dårligt kodeord[2]. Hvordan KeyExpansion foregår, ligger udover denne opgaves fokus, men resultatet er, at maskinen efter KeyExpansion har 11 små kvadrater. Det første er den oprindelige nøgle. De andre kaldes rundenøgler, og er nummereret fra 1-10. Se figur 4.

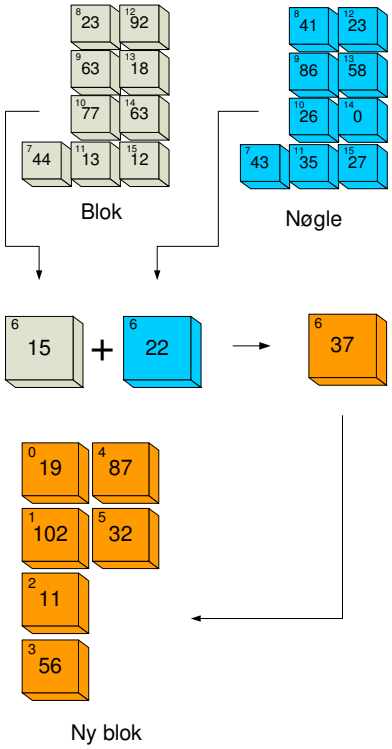


Figur 4 | KeyExpansion

[1] Jeg har her valgt at bruge de korrekte fagtermer, da jeg mener at en "fordanskning" af disse blot vil lede til endnu større forvirring.

[2] Med 'dårligt' mener jeg et kodeord, svarende til at lave Cæser-kode med et skift på 0, dvs. at intet bliver ændret.

AddRoundKey



Figur 5 | AddRoundKey
Her nået til klods nr. 6

Til denne proces skal vi bruge vores blok og en nøgle. Nøglen kan enten være vores alm. nøgle, eller en af de 10 rundenøgler, hvilket vi skal se nærmere på senere. Operationen svarer ca. til, at maskinen først tager tallet på den første klods i blokken, og lægger det sammen med tallet på den første klods i nøglen. Resultatet af dette finder maskinen i bunken af løse træklodser, og denne klods sættes i stedet for den første i blokken. På tilsvarende vis fortsættes med de andre klodser i blokken. Bemærk at hvis tallet på en af nøgleklodserne er 0, så ændres den tilsvarende klods i blokken ikke[1].

Til sidst har vi altså en blok, hvor klodserne er byttet ud med nogle andre.

SubBytes

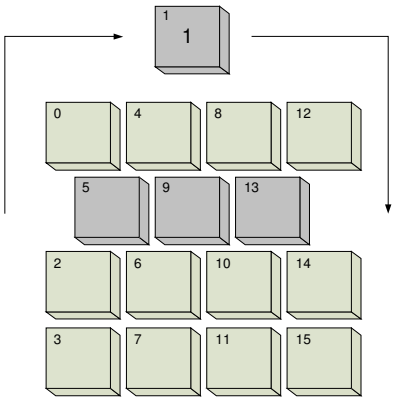
I denne proces bytter maskinen om på de forskellige klodser i vores blok efter et helt fast-defineret mønster. Dette mønster kan opskrives i en tabel, som man kalder S-box, men det venter vi med til senere. Det afgørende er her, at de forskellige klodser blandes godt og grundigt. Denne proces er bl.a. med til at sikre os mod det vi så i AddRoundKey, når en klods i nøglen er 0.

I SubBytes byttes ALLE klodserne i vores blok ud med andre[2].

[1] Da den virkelige metode er noget anderledes, er denne beskrivelse altså kun til for at give et billede af hvad der kan ske under særlige omstændigheder.

[2] Dette kan vi bevise matematisk, da den affine afbildning som bruges, ikke afbilder noget som sig selv.

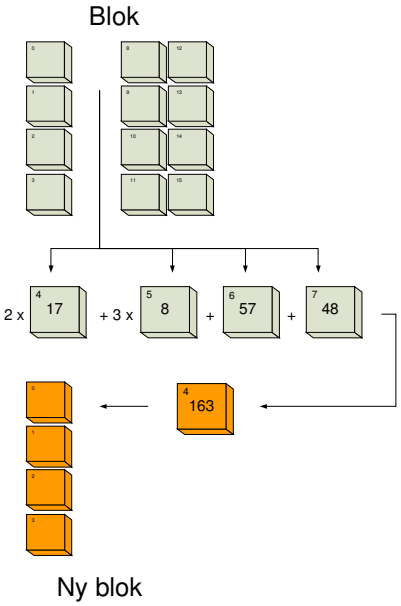
ShiftRows



Figur 6 | ShiftRows
Her i færd med at rotere anden række

Denne proces kan faktisk til fulde forklares med klodser. Her er der ikke tale om en allegorisk tilnærmelse. Processen foregår som følger:
Den øverste række i vores blok forbliver uændret.
Den anden række roteres én plads til venstre, dvs. at den klods der står længst til venstre i anden række, bliver roteret, så den kommer til at stå længst til højre, og de tre andre klodser skubbes derfor én plads mod venstre.
Tredje række roteres to pladser, og fjerde række roteres tre pladser.

MixColumns



Figur 7 | MixColumns

Hvor ShiftRows arbejder med de enkelte rækker i vores blok, arbejder MixColumns med de enkelte søjler i vores blok. Der sker altså ingen blanding mellem de fire søjler.
Det er denne del af algoritmen, som kræver langt den største matematiske forståelse, hvilket også gør den svær at forklare med klodser.
Det handler dog om, at man tager de fire klodser i en søjle. Den nye klods, der skal sidde i stedet for den første, beregnes ud fra tallene på de fire klodser i søjlen, efter en formel der minder om at sige: 2 gange tallet på den første klods + 3 gange tallet på den anden klods + 1 gange tallet på den tredje klods + 1 gange tallet på den fjerde klods.
Tallene 1, 2 og 3, som ganges med de enkelte klodser, skifter så, alt efter hvilken af de fire klodser i den nye søjle, man ønsker at beregne.

Round

En runde svarer til en sammensætning af processerne:

```
SubBytes
ShiftRows
MixColumns
AddRoundKey
```

Her angiver rundens nummer hvilken af de 10 rundenøgler, der skal lægges til i AddRoundKey.

Husk at det er den samme blok, man arbejder videre med hele tiden.

FinalRound

Den sidste runde har nr. 10, og er næsten identisk med de første 9 runder, bortset fra at MixColumns er udeladt. Den hedder altså:

```
SubBytes
ShiftRows
AddRoundKey
```

Hvor det er den 10. og sidste rundenøgle, der skal lægges til i AddRoundKey.

Kryptering

Når maskinen skal kryptere en lang række af klodser, ud fra en nøgle, kan den samlede proces beskrives i en slags pseudokode.

```
KeyExpansion
For hver blok i klarteksten
  AddRoundKey med nøglen
  Round 1-9
  FinalRound
Gentag
```

Til sidst samles de mange krypterede blokke sammen til igen at udgøre en lang række af klodser. Bemærk at tallet der nu står på disse klodser, ikke nødvendigvis svarer til et bogstav i alfabetet, da der jo kan stå 256 forskellige tal på klodserne, og der kun er 28 bogstaver. At kalde det kryptotekst er derfor misvisende, da der ret beset ikke er så meget tekst over det.

En meget flot visuel animation af krypteringen kan ses i [Ani]

ANDEN DEL

I denne anden del af min opgave stiller jeg skarpt på matematikken i algoritmen. Nu er vi ude over træklodser og maskiner. Jeg vil dog flere steder drage paralleller til min allegori, for på den måde at præcisere den, og samtidig give læseren en slags "aha"-oplevelse.

Matematikken i Rijndael

Matematikken der ligger til grund for Rijndael-algoritmen, kræver kendskab til en række mere avancerede begreber inden for algebraen.

Lad mig dog starte bagfra: Grunden til, at vi overhovedet skal have fat på denne matematik, er ønsket om at bevise, at systemet virker – at vi ALTID kan kryptere en given klartekst, med en given nøgle, og bagefter dekryptere den igen med samme nøgle, og få samme klartekst som vi startede med. Selvom det er fint at have et system, der bare virker, er det altså smart, hvis man kan bevise hvorfor det virker, ud fra matematiske sætninger.
Det vi skal have gjort, er at omsætte vores bits og bytes til polynomiumskoefficienter i et endeligt legeme, et såkaldt Galois legeme. Ud fra de regneregler, matematiske sætninger og beviser der gælder for endelige legemer, er det så muligt at vise hvordan disse komplekse polynomiums udregninger svarer til relativt simple bit-operationer i algoritmen, og herved bevise at systemet virker.

Komposition, algebraisk struktur og semigruppe

Vi skal altså først finde ud af hvad et endeligt legeme er, så lad os starte lidt længere tilbage.
Vi starter ved begrebet komposition. Ved at omskrive definitionen i [KriRin] s. 259, får vi at en komposition er: I en given mængde ***M***, vælges to elementer, a og b. Hvis a*b giver et element i ***M***, hvor * repræsenterer en af de fire elementære regningsarter, siges denne regningsart at være en komposition i ***M***. En komposition knyttet til en mængde kaldes også en algebraisk struktur.
Addition og multiplikation er begge kompositioner i ***N***, ***Z***, ***Q***, ***R*** osv..
Subtraktion er derimod ikke en komposition i ***N***, da der kan findes to tal i ***N***, der når de trækkes fra hinanden giver et negativt tal, som altså ikke tilhører ***N***. Bemærk i øvrigt at elementerne ikke nødvendigvis behøver være tal. Mængder af restklasser modulo n, hvor n er et naturligt tal,

kan også være elementer i $\boldsymbol{M}$. Hvis n fx er 5, har vi naturligvis 5 forskellige restklasser, som tilsammen udgør en mængde. Addition og multiplikation (modulo 5) er begge kompositioner i denne mængde[1].

Hvis den associative lov gælder for en komposition i en mængde $\boldsymbol{M}$, kaldes $\boldsymbol{M}$ for en semigruppe. Gælder den kommutative lov også, kaldes det blot en kommutativ semigruppe[2].

Neutralt og inverst element

Næste trin er at introducere det neutrale element, som helt banalt, er det element e der for en algebraisk struktur ($\boldsymbol{M}$,*) medfører at der for ethvert x gælder at x*e = e*x = x. e og x tilhører begge mængden $\boldsymbol{M}$.[3]

Tallet 0 er neutralt element i algebraiske strukturer med addition og subtraktion i diverse talmængder, og tallet 1 er neutralt element ved multiplikation[4].

På samme måde som det neutrale element, findes også et inverst element a således at x*a = a*x = e, hvor a, x og e alle tilhører mængden $\boldsymbol{M}$, og hvor e er det neutrale element[5]. Findes der et inverst element til x, siges x at være invertibelt. Denne egenskab er helt fundamental når vi skal sikre at vi både kan kryptere og dekryptere.

Grupper, Abelske grupper og ringe

Vi kan nu specificere en særlig art af semigrupper, som vi kalder grupper. En gruppe er blot en semigruppe der har et neutralt element, og hvor alle elementer er invertible, dvs. at alle elementer har et inverst element. Mængden af hele tal $\boldsymbol{Z}$, er en additiv gruppe, dvs. en komposition mht. addition. Mængden af positive, rationelle tal $\boldsymbol{Q}_+$ er en multiplikativ gruppe. Bemærk at der ikke findes nogen talmængder der både er en additiv- og multiplikativ gruppe.

Er gruppen endvidere kommutativ, kaldes den en Abelsk gruppe, efter den norske matematiker Niels Henrik Abel (1802-1829).

Hvis en additiv Abelsk gruppe også er associativ mht. multiplikation, kaldes gruppen en ring.[6] Eksempler på ringe, er de hele tal $\boldsymbol{Z}$, eller polynomier i x med heltalskoefficienter – $\boldsymbol{Z}$(x).

---

[1] For mere om restklasser og regning med disse, se [Erlandsen] s. 10ff.

[2] Kendskab til den associative og kommutative lov forudsættes, men en præcis, matematisk definition kan findes i [KriRin] s. 262ff.

[3] [KriRin] s. 265.

[4] Da jeg i første del beskrev AddRoundKey, hvor man lægger de to træklodsers værdier sammen, fortalte jeg at hvis nøgle-klodsen er 0, så ændres den tilsvarende klods i blokken ikke. Det skyldes selvfølgelig at 0 i denne regneoperation er det neutrale element. At der i virkeligheden ikke er tale om en alm. addition skal vi se på senere.

[5] Se endvidere [KriRin] s. 266.

[6] Se [MWring].

Legemer og Galois legemer

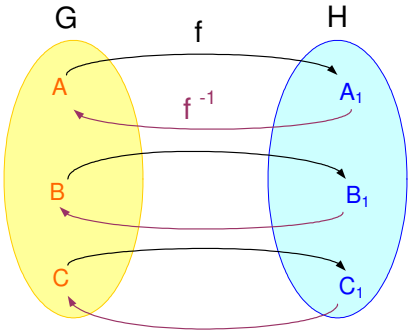
Billede 2 | Evariste Galois

Hvis en mængde af elementer tilfredsstiller alle de fire elementære regningsarter, er distributiv, kommutativ, indeholder det neutrale element for de fire regningsarter, og at alle elementer, har et inverst element ved addition, samt at alle, undtagen nul, har et inverst element ved multiplikation, kaldes den et legeme, eller et tallegeme[1]. For at kæde det sammen med alt det ovenstående, kan man sige at et legeme netop er en additiv- og multiplikativ gruppe, hvor man blot godtager at 0 ikke er invertibel ved multiplikation[2].

Eksempler på tallegemer er de komplekse tal $\boldsymbol{C}$, de reelle tal $\boldsymbol{R}$, og de rationale tal $\boldsymbol{Q}$. De hele tal $\boldsymbol{Z}$, er IKKE et legeme, da der ikke findes inverse elementer til alle elementer ved multiplikation. Bemærk i øvrigt at et legeme indeholder mindst to elementer, da det neutrale element for addition og multiplikation er forskellige.

Hvis et legeme indeholder et endeligt antal elementer, kaldes det et endeligt legeme, eller et Galois legeme, efter den franske matematiker Evariste Galois (1811-1832), og det er netop disse Galois legemer vi får brug for når vi skal forklare Rijndael.

Homomorfi og isomorfi



Figur 8 | Isomorfi

De sidste to begreber vi skal have styr på, knytter sig til afbildninger af algebraiske strukturer, altså mængder knyttet til en given komposition. For overskuelighedens skyld, forklarer jeg her begrebet med et eksempel[3].

Lad $\boldsymbol{G}$ og $\boldsymbol{H}$ være mængder af de naturlige tal, og lad f(x)=3x være en afbildning af $\boldsymbol{G}$ ind i $\boldsymbol{H}$. Afbildningen f siges nu at være en homomorf afbildning, da der til ethvert x tilhørende $\boldsymbol{G}$, findes en afbildning f(x) som tilhører $\boldsymbol{H}$. Gælder det endvidere at alle elementer i $\boldsymbol{H}$ kan skrives som en afbildning af f(x), siges afbildningen at være isomorf, og de to mængder siges at være isomorfe. Det afgørende er her, at der er en såkaldt 1:1 overensstemmelse mellem de to mængder, hvilket vi skal se er afgørende når vi skal sikre at algoritmen også kan vendes om, så vi bliver i stand til at dekryptere, uden at der opstår fejl.

[1] Læs mere om tallegemer i [Lytzen] s. 98f, og [MWfield].

[2] I kryptologi definerer man oftest blot at 0 er sin egen inverse, da der ikke er andre tal der har 0 som invers ved multiplikation.

[3] For en præcis definition, henvises til [KriRin] s. 268f.

Galois legemer og polynomiumsrepræsentation

Nu har vi fået introduceret en række matematiske begreber, og nu er tiden kommet, til at sætte dem sammen, og gøre det konkret i forhold til Rijndael.

Vi skal m.a.o. have udtrykt vores bits og bytes i form af noget matematik vi kender. Det afgørende her, er at indse at man kan betragte en byte, som et polynomium med koefficienter fra mængden af restklasser modulo 2 – $\mathbb{Z}_2$, altså $\{[0],[1]\}_2$.[1] Alle bytes kan derfor repræsenteres som koefficienterne i et polynomium fra ringen af polynomier $\mathbb{Z}_2(x)$.

Da en byte indeholder 8 bits, er det klart at dette polynomium skal have en grad der er mindre end 8. For at sikre dette konstruerer vi vores Galois legeme ved at tage vores polynomier fra før: $\mathbb{Z}_2(x)$, modulo et ireducibelt polynomium af 8.-grad, som vi kalder m(x).[2]

$m(x) = x^8 + x^4 + x^3 + x + 1.$

Vi ved fra vores kendskab til polynomiers division, at resten altid vil være af en lavere grad end divisoren. Vi kan derfor være sikre på at alle polynomierne i vores Galois legeme højst er af 7. grad.

Vi benævner vores Galois legeme $GF(2^8)$, og det bør være klart at dette indeholder netop 256 elementer, altså 256 forskellige polynomier, med koefficienterne $[0]_2$ eller $[1]_2$, svarende til at der er 256 forskellige bytes.[3]

En vigtig egenskab ved dette legeme, er at der både ved addition og multiplikation findes et entydigt bestemt inverst element. Ved addition er det inverse element normalt blot sig selv med modsat fortegn, men ved regning med restklasser modulo 2, ses det at subtraktion er samme operation som addition. Derfor er det inverse element i vores $GF(2^8)$, ved addition, elementet selv.

Ved multiplikation bliver det straks værre, og her kan jeg udelukkende forklare at man ud fra extended Euclidean Algoritm kan bestemme et entydigt, inverst element ved multiplikation i vores legeme.[4]

[1] For mere om regning med restklasser henvises igen til [Erlandsen] s. 11ff.

[2] Ved ireducibelt forstås et polynomium som ikke kan skrives som produktet af to polynomier af lavere grad, alt sammen indenfor $\mathbb{Z}_2(x)$. Se [Masnyk] s. 18, og [MWiredpol].

[3] Husk at 0 og 1 altså ikke skal forstås som tal, men som repræsentanter for restklasser modulo 2 – i [Masnyk] og [FIPS] betragtes det dog som tal, der blot reduceres modulo 2, men jeg mener det er mere logisk at regne med restklasser, da vi så med det samme kan indse at alt hvad bevises i [Erlandsen] s. 11ff, også gælder her.

[4] Se [Masnyk] s. 21f hvor der henvises til andre beviser, eller den korte forklaring i [FIPS] s. 15.

XOR og AddRoundKey

Det vi nu skal indse, er at vores avancerede regninger med polynomier, kan implementeres på bit-niveau, med langt simplere funktioner. Polynomiumsadditionen svarer fuldstændigt til bit-operationen XOR, som vi skriver $\oplus$. således bliver: $1 \oplus 1 = 0$, $0 \oplus 1 = 1$, $1 \oplus 0 = 1$, $0 \oplus 0 = 0$.
Det er denne operation der bruges i funktionen AddRoundKey. Her XOR'es hver byte i nøglen, eller runde-nøglen, blot til den tilsvarende byte i vores blok.

{01001101} $\oplus$
{10000100} =
{11001001}

Dette svarer til følgende:

$([1]x^6 + [1]x^3 + [1]x^2 + [1]) + ([1]x^7 + [1]x^2 + [1]) =$
$[1]x^7 + [1]x^6 + [1]x^3 + [2]x^2 + [1] =$
$[1]x^7 + [1]x^6 + [1]x^3 + [1]$

Da vi indser at restklassen [2] = [0].

Multiplikation og xtime

Som vi så, svarer XOR altså fuldstændigt til polynomiumsaddition i $GF(2^8)$. Der findes dog ikke umiddelbart nogen simpel bit-operation der svarer til multiplikationen[1], men selvfølgelig er der alligevel et trick.
Vi ser på multiplikationen af et polynomium i $GF(2^8)$, med x. Her ganges x blot med alle led.

$(a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0) \cdot x =$
$a_7x^8 + a_6x^7 + a_5x^6 + a_4x^5 + a_3x^4 + a_2x^3 + a_1x^2 + a_0x$

Da alle led stiger en grad, svarer det egentlig blot at man skubber koefficienterne en plads til venstre.

{00100110} $\cdot$ x = {01001100}

Eller

{10100110} $\cdot$ x = {01} $\cdot$ {01001100}

Som det ses, opstår der dog et problem, hvis a_7=1, da der så pludselig opstår en 9.-bit(et polynomiumsled med x^8), som sætter os ud af stand til at repræsentere vores polynomium som en byte. Derfor reducerer vi stadig vores polynomium modulo vores irreducible polynomium m(x) = **01 1B** = {01}{00011011}[2].

[1] [FIPS] s. 15

[2] Grunden til at den første bit står for sig, er netop for at gøre det tydeligt at der er tale om en byte der er længere end resten.

Funktionen xtime defineres altså som en funktion der blot "skubber" bits-ne én plads til venstre, og indeholder desuden en konditionel XOR, med betingelsen: er a_8=1, så XOR med `01 1B`.[1]

Vi har altså en funktion, xtime, der skubber bitsne én plads til venstre, og om nødvendigt, reducerer, så ingen bits "mistes". Det smarte er så at vi kan rotere lige så mange gange vi vil, ved blot at tage funktionen på sig selv.[2]

xtime(xtime({00000001})) =

{00000010} · {00000010} · {00000001} = {00000100} [3]

På den måde kan vi altså gange med et vilkårligt polynomium, hvilket vi skal se nærmere på i MixColumns. Multiplikation i endelige legemer har regneoperatoren: •

Polynomier med koefficienter i GF(2^8) og MixColumns

Vi har nu forstået at vi kan have polynomier med restklasser modulo 2 som koefficienter, men nu indfører vi endnu et begreb, hvor vi skal hæve abstraktionsniveauet endnu et trin.

Vi ser på polynomiet $a(x) = a_3x^3 + a_2x^2 + a_1x + a_0$

Hvor koefficienterne før var restklasser, er de nu i stedet polynomier i vores legeme, GF(2^8). Fx kunne a_3 = {01001110}, og a_2 = {00000001} osv., svarende til polynomier.

Vores polynomium a(x) repræsenterer nu ikke længere en 8-bit byte, men et 4-byte word[4].

Addition er, hvis vi husker XOR-operationen fra før, en elementær operation, som kan vises med polynomierne:

$a(x) = a_3x^3 + a_2x^2 + a_1x + a_0$ og

$b(x) = b_3x^3 + b_2x^2 + b_1x + b_0$

$c(x) = a(x) + b(x) =$

$(a_3 \oplus b_3)x^3 + (a_2 \oplus b_2)x^2 + (a_1 \oplus b_1)x + (a_0 \oplus b_0)$

Det er her klart at c(x) er et polynomium med en grad mindre end 4, og at koefficienterne vil være polynomiumsrepræsentationer med en grad mindre end 8. Altså er der ingen problemer her.

---

[1] Se evt. [ASP] l. 250-261 eller pseudekoden i [Masnyk] s. 34.

[2] Se endvidere [Masnyk] s. 33f, for en længere beskrivelse af multiplikation.

[3] Bemærk at {00000010} = `02`.

[4] Et word, er i denne forstand en datalogisk definition på hvor lang en data-sekvens der kan lagres i variablen. Andre eksempler på datatyper (QBasic) kunne være: boolean(sand/falsk, dvs. 1-bit), integer(16-bit heltal), double(64-bit decimaltal) eller string(oftest 256-byte word).

Ved multiplikation opstår der selvfølgelig nye problemer, da koefficienterne nu ikke længere skal reduceres modulo 2 eller regnes som restklasser modulo 2, i kraft af at der nu er tale om polynomier.

Ved multiplikation af $a(x) \bullet b(x)$ skriver vi først koefficienterne ud, og samler dem:

$$c(x) = c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0$$

Vi får koefficienterne udtrykt på følgende måde:

$$c_0 = a_0 \bullet b_0$$

$$c_1 = a_1 \bullet b_0 \oplus a_0 \bullet b_1$$

$$c_2 = a_2 \bullet b_0 \oplus a_1 \bullet b_1 \oplus a_0 \bullet b_2$$

$$c_3 = a_3 \bullet b_0 \oplus a_2 \bullet b_1 \oplus a_1 \bullet b_2 \oplus a_0 \bullet b_3$$

$$c_4 = a_3 \bullet b_1 \oplus a_2 \bullet b_2 \oplus a_1 \bullet b_3$$

$$c_5 = a_3 \bullet b_2 \oplus a_2 \bullet b_3$$

$$c_6 = a_3 \bullet b_3$$

Vi har pludselig alt for mange koefficienter, og vi skal derfor finde en måde at reducere vores $c(x)$. Dette gøres med polynomiet $x^4 + 1$.
Vi udfører polynomiers division, og får resten:

$$c_3x^3 + (c_2 - c_6)x^2 + (c_1 - c_5)x + (c_0 - c_4)$$

Vores reducerede polynomium $d(x)$ har altså koefficienterne:

$$d(x) = d_3x^3 + d_2x^2 + d_1x + d_0$$

$$d_0 = c_0 \oplus c_4$$

$$d_1 = c_1 \oplus c_5$$

$$d_2 = c_2 \oplus c_6$$

$$d_3 = c_3$$

Dette forudsætter dog at $a(x)$ er et fast polynomium, hvilket det i Rijndael netop er.
Bemærk i øvrigt at denne operation ikke nødvendigvis er invertibel, hvilket jo er en nødvendighed for at den kan bruges i algoritmen, men da Rijndael specificerer netop hvilket polynomium der skal multipliceres med, og da netop dette polynomium har en invers, er invertibiliteten sikret.
Alt dette bruges i MixColumns, men det er lettest at forstå ved at kigge på regneeksemplet på bilag 2.

Affine afbildninger og SubBytes

SubBytes, en affin transformation med forskriften:

$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}.$$

[FIPS] s. 15f

Bemærk at ingen bytes afbildes som sig selv.

Denne afbildning kan også opskrives skematisk:

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

[FIPS] s. 16.

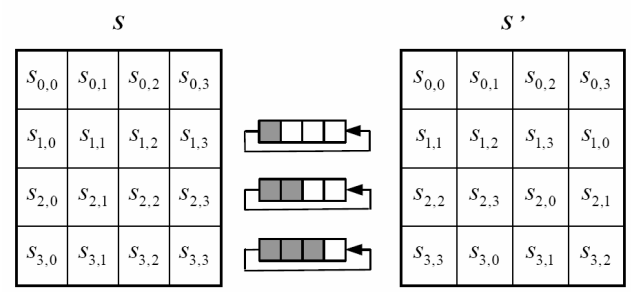
Hvis vi fx skal udføre SubBytes på **40**, så kigger vi på det som XY. I vores eksempel er X=**4** og Y=**0**. Vi kigger i skemaet, og ser at **40** afbildes som **09**.

I Twofish-algoritmen, som også var en AES-finalist, er dele af S-boxen nøgleafhængig, og i [Answer] s. 2 beskriver Rijmen og Daemen at dette gør det svært at vurdere kvaliteten af S-boxen.

ShiftRows

ShiftRows-operationen er matematisk set banal, og lader sig altså lettest forklare med klodser som i første del.

Denne figur fra [FIPS] s. 17. viser det lidt mere generelt:



Resten

Vi har nu set på en række af operationerne i Rijndael gennem matematisk optik. Vi har dog ikke været omkring KeyExpansion, men da der ikke er noget særlig matematik tilknyttet, blot en masse konditionelle statements, og simple regneoperationer, vil jeg ikke kigge mere på denne.

Dekrypteringsfunktionen, og alle de inverse regneoperationer gennemgås ligeledes heller ikke, da der ikke er plads til at bevise dem her. Jeg mener dog at have gjort opmærksom på ireversibiliteten i flere af delfunktionerne, og det burde være let at gennemskue at det ikke er væsentligt sværere at lave en tilsvarende dekrypteringsfunktion.

KONKLUSION

Grundet Rijndaels relativt simple opbygning, er det muligt at forklare algoritmens virkemåde i en ikke-matematisk form, og samtidig kan man betragte den strengt matematisk og med lidt mere plads til rådighed, bevise at den virker begge veje.

Hvis man skal perspektivere til virkeligheden, kan man sige, at behovet for sikker kryptering er konstant voksende, og at åbne standarder som AES er med til at sikre privatlivets fred, idet selv store offentlige institutioner vil have meget svært ved at "snage" i borgernes privatliv. En uheldig konsekvens af dette er selvfølgelig, at terrorister, som i dag er blevet den globale fjende, har samme muligheder for at hemmeligholde deres informationer for efterretningstjenesterne. God, kryptografisk hemmeligholdelse må dog ikke forveksles med it-sikkerhed generelt, og det er nok her, der alligevel vil være en bagdør, både i forhold til terrorister og privatlivets fred.

Men lige meget om Rijndael bidrager til verdensfreden eller ej, så er det et smukt stykke matematik, og de to ophavsmænd fortjener absolut stor ære for udviklingen af et så simpelt og samtidig så effektivt system.

LITTERATURLISTE

Primære

[KriRin]
Erik Kristensen og Ole Rindung
"Matematik 1"
G.E.C Gads Forlag | 2. reviderede udgave
s. 259-295.

[Landrock]
Peter Landrock og Knud Nissen
"Kryptologi – fra viden til videnskab"
ABACUS | 1. udage, 1. oplag
ISBN 87-89182-62-6

[Lytzen]
Jesper Lützen
"Cirklens kvadratur, Vinklens tredeling, Terningens fordobling"
Systime | s. 98-107

[Singh]
Simon Singh
"Kodebogen"
Gyldendal | 1. udgave, 1. oplag
ISBN 87-00-45556-3

Primære (på cd)

[Masnyk]
Olga Masnyk Hansen
IMADA Syddansk Universitet
Masnyk.pdf

[FIPS]
Federal Information Processing Standards
Publication 197
fips-197.pdf

[Erlandsen]
Mikkel Kamstrup Erlandsen
"Introduktion til Kryptologi"
erlandsen.pdf

Internet (på CD)

[Ani]
Enrique Zabala
"Rijndael Cipher" (Animation)
animation.exe

[Answer]
Joan Daemen og Vincent Rijmen
"Answer to "new observations on Rijndael""
Answer.pdf

[Fogie]
Seth Fogie
„Cracking WEP"
Fogie.pdf

[MWfield]
Eric W. Weisstein. "Field."
From MathWorld–A Wolfram Web Resource.
http://mathworld.wolfram.com/Field.html
MWfield.pdf

[MWring]
Eric W. Weisstein. "Ring."
From MathWorld–A Wolfram Web Resource.
http://mathworld.wolfram.com/Ring.html
MWring.pdf

[MWiredpol]
Eric W. Weisstein. "Irreducible Polynomial."
From MathWorld–A Wolfram Web Resource.
http://mathworld.wolfram.com/IrreduciblePolynomial.html
MWiredpol.pdf

[Savard]
John J. G. Savard
http://home.ecn.ab.ca/~jsavard/crypto/jscrypt.htm
Savard_kap4_Rijndael.pdf
Savard_kap4_MARS.pdf
Savard_kap4_Twofish.pdf
Savard_kap4_RC6.pdf
Savard_kap4_Serpent.pdf
Savard_kap5_DH.pdf
Savard_kap5_ElGamal.pdf

[Twofish]
Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall og Niels Ferguson
"Performance Comparison of the AES Submissions" twofish.pdf

Internet (øvrige)

Disse kilder er ikke nødvendige for forståelsen af opgaven, og jeg vurderer derfor at det er unødvendigt at bruge tid på at samle dem på cd.

[CSRS]
http://csrc.nist.gov/CryptoToolkit/aes/

[fan]
http://rijndael.info/

Andet (på CD)

[JP]
Artikel i Jyllandsposten – 11. okt. 2000
Jim Wolfreuters
"Amerikanerne vælger belgisk krypteringsmetode"
Artikler.pdf

[ASP]
VBScript-implementering af Rijndael
Rijndael asp.pdf

[JS]
Modificeret JavaScript-implementering af Rijndael
Rijdaeltest html.pdf
Rijndaeltest.html

Billeder

RSA-holdet
http://www.usc.edu/dept/molecular-science/RSA-2003.htm
Evariste Galois
http://www.win.tue.nl/~aeb/at/GaloisCorrespondence.html

B I L A G

1 – Bits, bytes, ASCII og HEX-koder

Moderne kryptering foregår i en computer, og derfor skal vi have omsat vores informationer til noget computeren kan forstå. I dette tilfælde tænker jeg at vores informationer er helt almindelig tekst, bestående af bogstaver og tegn. Det vi ønsker at hemmeligholde kunne fx være teksten: ”kryptosystemerne”. I en computer foregår alt i bits – enten eller – og vi skal derfor have et system til at lave teksten om til kombinationer af enten eller. Hvorfor alt i en computer er bits, er en større teknisk forklaring, som må udelades her. Vi behøver heldigvis ikke opfinde den dybe tallerken igen, for et sådant system findes allerede. Systemet kaldes ASCII, og er ganske enkelt en tabel, der oversætter mellem bogstaver og 8-bit sekvenser, som er det vi også kalder bytes. Den oprindelige ASCII-tabel havde kun 7-bits, og var derfor begrænset til 128 tegn. I al væsentlighed går tabellen ud på at omsætte hvert bogstav og tegn til et decimaltal mellem 0 og 256. A-Z ligger fra nr. 65-90 og a-z ligger fra 97-122. Disse decimaltal kan let omregnes til det binære talsystem.

Hvis vi skal oversætter ”kryptosystemerne” til binære ASCII-koder, får vi resultatet her til venstre.

”Kryptosystemerne”
{01101011}
{01110010}
{01111001}
{01110000}
{01110100}
{01101111}
{01110011}
{01111001}
{01110011}
{01110100}
{01100101}
{01101101}
{01100101}
{01110010}
{01101110}

Alt dette bliver let en smule uoverskueligt, thi det fylder væsentligt mere at skrive byte-koder. Det viser sig her at være smart at vi i stedet for at omregne vores decimaltals-koder til binære tal, bruger 16-talsystemet, eller hexadecimal, til at repræsentere vores koder. Omregning mellem talsystemer ligger udenfor denne opgaves fokus. I stedet for de 16 binære tal, får vi i stedet en blok som den til venstre.

”Kryptosystemerne”
6B 74 73 65
72 6F 74 72
79 73 65 6E
70 79 6D 65
Husk at blokken læses lodret, søjlevis.

Her er hver tocifret hexkode repræsentant for et bogstav. Herefter er bogstaverne sat op i matriks-form, som de bruges i Rijndael.

Bemærk at det her ikke længere er klart hvilke polynomier hver byte repræsenterer.

Det afgørende ved alt dette, er at vi kan omsætte vores klartekst til tal som vi kan regne på[1]. I praksis behøver vi slet ikke tænke over at teksten omdannes til tal, da dette naturligvis foregår helt automatisk.

[1] [Erlandsen] s. 6.

2 – Regneeksempel

Jeg vil her konkret gennemgå et par regninger i algoritmen, med forklaringer osv..

Vi starter med vores klartekst: ”kryptosystemerne”

Desuden vælger vi en nøgle. Her har jeg valgt den samme nøgle som er brugt i regneeksemplet i [FIPS] s. 33: **2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F 3C** – denne kan dog ikke umiddelbart oversættes til noget tekst, da der ikke er tegn som svarer til alle koderne.

Før krypteringen går i gang, skal vi have lavet vores runde-nøgler ud fra nøglen, vha. KeyExpansion. Denne proces har jeg ikke forklaret nærmere i opgaven, og ligesom jeg bruger nøglen fra [FIPS] bruger jeg også de rundenøgler som er udregnet i [FIPS] s. 27f og som kan ses i eksemplet på s. 33f.

Det første vi har, er altså vores klartekst, som i hex kan opskrives som vist til venstre.

”Kryptosystemerne”

6B 74 73 65
72 6F 74 72
79 73 65 6E
70 79 6D 65

Vi husker at første omgang udelukkende går ud på at lægge selve nøglen til, vha. XOR.

Nøglen

2B 28 AB 09
7E AE F7 CF
15 D2 15 4F
16 A6 88 3C

Første regnestykke er altså **6B** ⊕ **2B** =

{01101011} ⊕
{00101011} =
{01000000} = **40**

Næste regnestykke er **72** ⊕ **7E** =

{01110010} ⊕
{01111110} =
{01011010} = **0C**

Efter første runde

40 5C D8 6C
0C C1 83 BD
6C A1 70 21
66 DF E5 59

Næste regnestykke er **79 + 15** = **6C**

Og sådan fortsættes.

Vi har nu vores nye blok, efter første runde, som vist til ventre.

Efter SubBytes

09 4A 61 50

FE 78 EC 7A

50 32 51 FD

33 9E D9 CB

Vi regner videre, og er nu nået til den første rigtige runde. Første operation er SubBytes.

Vi bruger tabellen som vist i afsnittet om SubBytes, og slår op:

40 → 09

0C → FE

6C → 50

66 → 33

Og vi fortsætter, så vi får tabellen til venstre.

Næste trin er ShiftRows. Denne proces kræver ikke megen regnekraft. Første række i vores blok ændres ikke:

09 4A 61 50 → 09 4A 61 50

Efter ShiftRows

09 4A 61 50

78 EC 7A FE

51 FD 50 32

CB 33 9E D9

Næste række forskydes én plads:

FE 78 EC 7A → 78 EC 7A FE

Tredje række forskydes to pladser:

50 32 51 FD → 51 FD 50 32

Fjerde række forskydes tre pladser:

33 9E D9 CB → CB 33 9E D9

Vi får vores blok til venstre.

Så er vi nået til MixColumns. Her tager vi vores blok søjle for søjle. Den første søjle er således:

09 78 51 CB

Denne søjle skal ganges med vores faste polynomium a(x), med koefficienterne: **03 01 01 02**, mod $x^4 + 1$, svarende til **11**.

Først indser vi at det i praksis er langt nemmere at bruge xtime-funktionen, end at gange polynomier sammen og reducere, så vi skal have omsat vores multiplikationer til noget med xtime.

$s'_0 = 02 \bullet 09 \oplus 03 \bullet 78 \oplus 01 \bullet 51 \oplus 01 \bullet CB$

At gange med **01**, der er det neutrale element, ændrer naturligvis ikke noget. Som vi så under xtime, svarer det at gange med **02**, til et venstreskift, og evt. reduceret ved at XOR med vores m(x), **11B**. At gange med **03**, er banalt, når man indser at **03** = **01**+**02**, og da den distributive lov gælder, kan vi skrive en multiplikation med **03•CI** som **CI** + xtime(**CI**).

Vi får altså en mere regnevenlig form hvis vi skriver:

$s'_{0,c} = \text{xtime}(s_{0,c}) \oplus s_{1,c} \oplus \text{xtime}(s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$

Vi udfører mellemregninger:

02•D4 = xtime(**09**) =

xtime({00001001}) = {00010010} = **12**

03•78 = **02•78** ⊕ **01•78** =

xtime(**78**) ⊕ **78** =

xtime({10111111}) ⊕ {01111000} =

{11110000} ⊕ {01111000} =

{10001000} = **F0** ⊕ **78** = **88**

Og samler til sidst:

s_0 = **12** ⊕ **88** ⊕ **51** ⊕ **CB** = **00**

Efter MixColumns

00	**75**	**82**	**52**
C1	**A6**	**FB**	**38**
95	**12**	**02**	**BA**
BF	**A9**	**AE**	**95**

Således fortsættes med resten af søjlen, og de tre andre søjler, og vores nye blok fås som vist til venstre.

Nu skal vi lægge vores første RoundKey til, med XOR, men der er ingen grund til at gennemgå denne proces igen. For en komplet gennemregning, så se resultatet i bilag 3.

3 – Maskin-regnet eksempel med mellemresultater

```
VORES NØGLE ER: 2B7E151628AED2A6ABF7158809CF4F3C
VORES KLARTEKST ER: 6B727970746F73797374656D65726E65

R U N D E 1
BEFORE    = 400C6C665CC1A1DFD88370E56CBD2159
BYTESUB   = 09FE50334A78329E61EC51D9507AFDCB
SHIFTROW  = 097851CB4AECFD33617A509E50FE32D9
MIXCOLUMN = 00C195BF75A612A982FB02AE5238BA95
ADDRK     = A03B6BA8FDF23E18A1583B977854CC90
R U N D E 2
BEFORE    = A03B6BA8FDF23E18A1583B977854CC90
BYTESUB   = E0E27FC25489B2AD326AE288BC204B60
SHIFTROW  = E089E260546A4BC232207FADBCE2B288
MIXCOLUMN = D9B416909F9FF542D65E00486426A284
ADDRK     = 2B768362E5094C018F6B8032177F54FB
R U N D E 3
BEFORE    = 2B768362E5094C018F6B8032177F54FB
BYTESUB   = F138ECAAD901297C737FCD23F0D2200F
SHIFTROW  = F101CD0FD97F20AA73D2EC7CF0382923
MIXCOLUMN = 38B060DAA2ED03601B9FE653B9D8FF5C
ADDRK     = 053027A7E5FBFD5E05BC9817D4A27767
R U N D E 4
BEFORE    = 053027A7E5FBFD5E05BC9817D4A27767
BYTESUB   = 6B04CC5CD90F54586B6546F0483AF585
SHIFTROW  = 6B0F4685D965F55C6B3ACC58480454F0
MIXCOLUMN = 043A7CE5AF4BA9580C083AFB384CEF73
ADDRK     = EB7ED9A40719F227BA791FC0E3474273
R U N D E 5
BEFORE    = EB7ED9A40719F227BA791FC0E3474273
BYTESUB   = E9F33549C5D489CCF4B6C0BA11A02C8F
SHIFTROW  = E9D4C08FC5B62C49F4A035CC11F389BA
MIXCOLUMN = E18E2C31358FF05CF13C71111FD63E26
ADDRK     = 355FEAC9490C6DDB3BCEC9AD0E2F2B9A
R U N D E 6
BEFORE    = 355FEAC9490C6DDB3BCEC9AD0E2F2B9A
BYTESUB   = 96CF87DD3BFE3CB9E28BDD95AB15F1B8
SHIFTROW  = 96FEDDB83B8BF1DDE21587B9ABCF3C95
MIXCOLUMN = 4BB51AE9DCE33596DEE332C6AEFFB824
ADDRK     = 263DB993CDE80B6B051AB48764FF2BD9
```

```
R U N D E 7
BEFORE    = 263DB993CDE80B6B051AB48764FF2BD9
BYTESUB   = F72756DCBD9B2B7F6BA28D174316F135
SHIFTROW  = F79B8D35BDA2F1DC6B16567F43272B17
MIXCOLUMN = FB63327EB136992CC5C25003D3670BE7
ADDRK     = B537C570EE6950DF41641FB19DC1D7A8
R U N D E 8
BEFORE    = B537C570EE6950DF41641FB19DC1D7A8
BYTESUB   = D59AA65128F9539E8343C0C85E780EC2
SHIFTROW  = D5F9C0C228430E518378A69E5E9A53C8
MIXCOLUMN = A3A5EAC2CAED8497AD1C1567924C21A0
ADDRK     = 497799E37F603E459C37E007EDC1088F
R U N D E 9
BEFORE    = 497799E37F603E459C37E007EDC1088F
BYTESUB   = 3BF5EE11D2D0B26EDE9AE1C555783073
SHIFTROW  = 3BD0E173D29A3011DE78EE6E55F5B2C5
MIXCOLUMN = 8FCBA79A2BBC1BE5AF69D333D9AC8B29
ADDRK     = 23BCC1693246C7C487B8FA728EF08B47
R U N D E 10
BEFORE    = 23BCC1693246C7C487B8FA728EF08B47
BYTESUB   = 266578F9235AC61C176C2D40198C3DA0
SHIFTROW  = 265A2DA0236C3DF9178C781C1965C640
ADDRK     = F64ED408EA821870F6B374D4AF06CAE6
VI FÅR DET ENDELIGE RESULTAT:
F64ED408EA821870F6B374D4AF06CAE6
```

Resultatet er opnået med [JS], som indeholder en JavaScript-implementering af Rijndael, som jeg har modificeret med ret fra ophavsmanden, så den også giver mellemregningerne.
Forinden har jeg desuden regnet det samme eksempel som gennemgås i [FIPS] s. 33f, og fået samme resultater.

4 – Brydning af WEP

Ud fra [Fogie] vil jeg her kort sammenfatte hvorfor WEP-krypteringen i trådløse netværk ikke er særligt heldigt konstrueret.
De afgørende svagheder er:
Hver "pakke" der sendes over netværket er selvfølgelig krypteret, og hver pakke er krypteret med en nøgle der består af tre pseudo-random-karakterer sat sammen med et af brugeren defineret password, hvilket principielt er fint nok, da det medfører at hver pakke er krypteret med forskellige nøgler, hvilket vil sige at selvom en hacker fik dekrypteret en enkelt pakke, ville han ikke kunne dekryptere de andre. Problemet er at de tre pseudo-tilfældige-karakterer sendes som klartekst i starten af pakken. Det er med andre ord ganske ligetil at finde de første 3 karakterer som den givne pakke er krypteret med.
De algoritmer som ittererer gennem nogle nøgler, lidt på samme måde som KeyExpansion i Rijndael, er forudsigelige, fordi de altid starter med nogle bestemte konstanter.
XOR-operationen er sin egen inverse, hvilket vil sige at hvis man krypterer med XOR og en nøgle, og hackeren kender plaintext og kryptotekst, så kan han altså med det samme finde nøglen.
Den første byte i der er krypteret er altid AA.
Ud fra disse fakta, en smule snilde og ihærdighed, er det derfor muligt at konstruere et system der med sikkerhed kan bryde krypteringen.
Konklusionen er dog at der skal bruges ca. 7gb data, dvs. 7 milliarder bytes. I et hjem hvor det trådløse netværk udelukkende bruges til at dele en StofaNet-forbindelse, der er begrænset til 5gb data per måned, vil det altså tage ca. 5 uger at samle nok data til at bryde krypteringen.

5 – Programmerings-eksempler

For at forstå Rijndael-algoritmen, har jeg udover at studere [FIPS] og [Masnyk], desuden indhentet to forskellige implementeringer af algoritmen, skrevet i hhv. ASP/VBScript og JavaScript.
Ved at modificere JavaScript-koden, er det lykkedes mig at få systemet til at udskrive mellemresultaterne af krypteringen, så man kan følge med i hvad der sker. Desuden giver det på eksperimentel vis, god forståelse af algoritmen, hvis man selv prøver at ændre lidt på nøglen og klarteksten, og ser hvad der sker.
Selve programmeringskoderne kan læses i [ASP] og [JS], og desuden findes der et HTML-dokument på cd'en, med koden fra [JS], hvor man kan prøve den af, og se at det virker.

6 – Regning med polynomier i MathCad

Da jeg i forbindelse med den daglige klasseundervisning på Århus Statsgymnasium, har stiftet bekendtskab med matematik-programmet MathCad, faldt det mig naturligt at eksperimentere med hvordan polynomiumsregningen også kan foretages, svarende til [FIPS] s. 10f, hvor to polynomier ganges sammen og reduceres.

Vi ønsker at regne 57 * 83 = C1

57=01010111=i1
83=10000011=i2

$$i1 := \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} \qquad i2 := \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

Husk at bytes'sne skrives bagfra!

Vi opskriver vores polynomier:

$$p1(x) := \sum_{i=0}^{7} \left(i1_i \cdot x^i \right) \qquad p2(x) := \sum_{i=0}^{7} \left(i2_i \cdot x^i \right)$$

Polynomierne ganges sammen og gemmes som et nyt polynomie

$$p(x) := p1(x) \cdot p2(x) \text{ expand}, x \rightarrow 1 + 2 \cdot x + 2 \cdot x^7 + 2 \cdot x^2 + x^8 + x^3 + x^9 + x^4 + x^5 + x^{11} + x^6 + x^{13}$$

Vi uddrager koefficienterne, og reducerer modulo 2, hvilket der tilsyneladende ikke er nogen genvej til.

$$c := p(x)\ \text{coeffs}, x \rightarrow \begin{pmatrix} 1 \\ 2 \\ 2 \\ 1 \\ 1 \\ 1 \\ 1 \\ 2 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} \qquad cc := \begin{pmatrix} \text{mod}(c_0, 2) \\ \text{mod}(c_1, 2) \\ \text{mod}(c_2, 2) \\ \text{mod}(c_3, 2) \\ \text{mod}(c_4, 2) \\ \text{mod}(c_5, 2) \\ \text{mod}(c_6, 2) \\ \text{mod}(c_7, 2) \\ \text{mod}(c_8, 2) \\ \text{mod}(c_9, 2) \\ \text{mod}(c_{10}, 2) \\ \text{mod}(c_{11}, 2) \\ \text{mod}(c_{12}, 2) \\ \text{mod}(c_{13}, 2) \end{pmatrix}$$

Vi opstiller vores polynomie igen, med de reducerede koefficienter, og definerer vores reduktions-polynomie.

$$pp(x) := \sum_{i=0}^{13} \left(\text{mod}(cc_i, 2) \cdot x^i\right) \qquad m(x) := x^8 + x^4 + x^3 + x + 1$$

$$pp(x) \rightarrow 1 + x^3 + x^4 + x^5 + x^6 + x^8 + x^9 + x^{11} + x^{13}$$

Vi udfører polynomier division, og kan se resten:

$$z(x) := \frac{pp(x)}{m(x)}\ \text{convert}, \text{parfrac}, x \rightarrow x^5 + x^3 + \frac{1 - x^7 - x^6}{x^8 + x^4 + x^3 + x + 1}$$

Dette stemmer overens med FIPS, idet vi husker at -1 er kongruent med (1 mod 2).