Een beschrijving en analyse van IDEA

Technische
Universiteit
Eindhoven

Instituut Wiskundige Dienstverlening Eindhoven

Rapport IWDE 98 - 04

Een beschrijving en analyse van IDEA

Prof.dr. A.E. Brouwer (primaire auteur)
Prof.dr. H.C.A. v. Tilborg
Dr. E.R. Verheul

(With an Engish summary at the end)

oktober 1998

Den Dolech 2
Postbus 513
5600 MB Eindhoven

Rapport IWDE 98 - 04

Een beschrijving en analyse van IDEA

Prof.dr. A.E. Brouwer (primaire auteur)
Prof.dr. H.C.A. v. Tilborg
Dr. E.R. Verheul

(With an Engish summary at the end)

oktober 1998

Management Samenvatting

Dit rapport geeft een beschrijving van IDEA (International Data Encryption Standard), alsmede een analyse van de cryptografische sterkte hiervan.

IDEA is een vercijferingsysteem dat in 1991 is voorgesteld door X. Lai en J. Massey als een veiliger alternatief voor DES (Digital Encryption Standard), de bekende Amerikaanse overheidsstandaard voor gevoelige, maar niet geclassificeerde informatie. Mede daarom zullen we in deze samenvatting de (beveiligings-)eigenschappen van IDEA bespreken met (Triple) DES als referentiepunt. Triple DES, ook wel EDE-DES genoemd bestaat uit het op speciale manier toepassen van drie DES versleutelingen met drie verschillende sleutels.

IDEA, DES en Triple DES hebben de volgende overeenkomsten:

- Alle zijn symmetrisch de vercijferingsystemen: de sleutel benodigd voor vercijfering en ontcijfering zijn dezelfde.
- Alle zijn *blok* vercijferingsystemen met een bloklengte van 64 bits (8 byte): een boodschap wordt in blokken van 64 bits opgedeeld en per blok versleuteld, dit betekent onder meer dat IDEA gebruikt kan worden in de "modes of operation" (ECB, CBC, CFB, OFB) waarin ook DES gebruikt kan worden. Met IDEA kunnen ook Message Authentication Codes (MAC's) worden gemaakt zoals ook het geval is bij DES.

IDEA, DES en Triple DES hebben de volgende verschillen:

- Het belangrijkste verschil tussen IDEA en DES is de gebruikte sleutellengte: IDEA gebruikt cryptografische sleutels van 128 bits lengte en DES van slechts 56 bits, Triple DES heeft een effectieve sleutellengte van ongeveer 112 bits (en niet van 3x56 = 168 bits zoals men zou verwachten).
- IDEA is in software ongeveer twee keer zo snel als DES, dat op zijn beurt weer drie keer zo snel is als Triple DES.
- Anders dan bij (Triple) DES, is het gebruik van IDEA niet kosteloos: de kosten voor commercieel gebruik lopen uiteen van $2 tot $15 per gebruiker. Niet-commercieel gebruik van IDEA is kosteloos, dit is een van de redenen dat IDEA gebruikt wordt in PGP (Pretty Good Privacy), een populair, gratis, pakket voor de versleuteling van *electronic mail* op het Internet.
- IDEA is speciaal ontwikkeld voor software implementaties op personal computers en maakt gebruik van bewerkingen die standaard aanwezig zijn op dit soort platforms, zoals vermenigvuldigen.

In het rapport worden de volgende aanvallen beschouwd op IDEA:

- "Exhaustive Key Search"
  Hierbij worden door een aanvaller, op basis van één of meer versleutelde boodschappen, eenvoudig weg alle mogelijke sleutels uitgeprobeerd: 2^{128} bij IDEA, 2^{112} bij Triple DES and 2^{56} bij DES. Zowel IDEA als Triple DES zijn veilig tegen deze aanval: met de huidige technologie is deze niet praktisch uitvoerbaar. Bij DES is deze aanval echter goed uitvoerbaar: recent heeft de Electronic Frontier Foundation voor $250.000 een "DES-cracker" laten bouwen die binnen 3 dagen een DES boodschap ontcijfert. Verder zijn de specificaties van de "DES-cracker" uitgebreid gepubliceerd in een boek, zodat er iedereen in principe een "DES-cracker" kan bouwen.
- "Birthday Attacks"
  Indien men IDEA (of (Triple) DES) gebruikt om grote hoeveelheden informatie te versleutelen met dezelfde sleutel, dan "lekt" men op een gegeven moment geheime informatie. Dit verschijnsel doet zich met grote waarschijnlijkheid voor, indien men $2^{32} . 64 = 2^{38}$ bits (=32 Gigabyte) met dezelfde sleutel versleutelt. Dit is onder de huidige communicatie- en opslagcapaciteiten geen ondenkbare hoeveelheid informatie. De hoeveelheid informatie die men met IDEA versleutelt met dezelfde sleutel, moet echter ruim onder deze grens blijven. Overigens, indien de bloklengte van IDEA 128 bits was geweest (i.p.v. van 64 bits) dan was deze grens $2^{64}. 128 = 2^{71}$ bits geweest en dat is, voor huidige begrippen, een onrealistische hoeveelheid informatie.
- Differentiële Cryptanalyse
  Hierbij is de aanname dat de aanvaller in staat is een IDEA versleutelbox met daarin een onbekende sleutel, naar believen boodschappen te laten versleutelen. De aanvaller probeert vervolgens deze sleutel te achterhalen door te kijken naar veranderingen in de versleutelde

boodschappen indien hij de oorspronkelijke boodschappen slechts in geringe mate van elkaar laat afwijken. Evenals als bij (Triple) DES is IDEA bestand tegen dit type aanvallen; dit was in feite een van de ontwerpspecificaties van IDEA, de publicaties hierover geven aan dat dit geslaagd is.

- "Timing Attacks"
  Hierbij is de aanname als bij het derde punt, maar let de aanvaller niet zozeer op de verschillen in de versleutelde teksten maar op de benodigde tijd voor versleuteling bij diverse boodschappen. IDEA (en (Triple) Des) zijn in theorie niet geheel veilig tegen dit soort aanvallen, maar praktisch uitvoerbaar lijken dit soort aanvallen niet.
- Zwakke sleutels
  Ook hier is de aanname als bij het derde punt. Sommige typen sleutels ("zwakke sleutels") geven bij IDEA een specifiek patroon versleutelde boodschappen bij bepaalde (gekozen) boodschappen. Hierdoor is het gebruik van zwakke sleutels eenvoudig vast te stellen, wat ongewenst is. Omdat de kans dat zo'n sleutel gebruikt is, zeer klein is (10^{-20}) kan het gevaar van zulke sleutels verwaarloosd worden. Omdat de zwakke sleutels herkenbaar zijn, kan men ook kiezen om bij de sleutelgeneratie, sleutels te testen op zwakheid vooraleer ze te gebruiken.

Resumerend, IDEA is bestand tegen alle bekende aanvallen maar is door zijn relatief korte bloklengte (64 bits) niet zo geschikt om grote hoeveelheden informatie (meerdere Gigabytes) te versleutelen onder dezelfde sleutel. Het is geen toeval dat een belangrijke specificatie-eis van de AES (Advanced Encryption Standard, de officiële Amerikaanse opvolger van DES) er uit bestaat dat de bloklengte 128 bits moet zijn. In die zin kan IDEA dus niet gezien worden als een opvolger van DES zoals bedoeld door Lai en Massey: hoogstens als tussenoplossing. Echter, ook Triple DES kan zo gezien worden. Hoewel IDEA zekere voordelen heeft boven Triple DES (hogere snelheid, iets meer veiligheid), hangt het van de applicatie af of dit de extra kosten rechtvaardigt.

IDEA

1 Oorsprong

IDEA is een blokversleutelingssysteem ontworpen door Xuejia Lai en James Massey van de ETH (Eidgenossische Technische Hochschule) in Zürich.

Het was bedoeld als veiliger alternatief voor DES (de Amerikaanse Digital Encryption Standard), en werd in 1990 geintroduceerd onder de naam PES (Proposed Encryption Standard, [29]). Toen de techniek van differentiële cryptanalyse [12] bekend werd, bleek dat een minuscule wijziging in het ontwerp het systeem veel beter bestand maakte tegen deze vorm van cryptanalyse. De verbeterde versie werd IPES (Improved Proposed Encryption Standard) genoemd [30]. In 1992 werd de naam gewijzigd in IDEA (International Data Encryption Algorithm, [28]).

2 Verkrijgbaarheid

Er rust patent op IDEA in Europa en de Verenigde Staten, en dit patent wordt beheerd door Ascom Systec AG, Dept CMVV, Gewerbepark, CH-5506 Mägenwil, Zwitserland (tel ++41 62 889 52 11, fax ++41 62 889 59 90). Niet-commercieel gebruik wordt aan een ieder kostenloos toegestaan.

Ascom is een groot Zwitsers telecommunicatiebedrijf. Op hun web-site [2] noemen ze een aantal van 12000 werknemers. Zie [3] voor de geleverde producten. Een hiervan is encryptie hardware en software; zie [4]. Onder [5] staat informatie gerelateerd aan IDEA.

De huidige situatie met betrekking tot octrooien wordt beschreven op [6]. IDEA is gepatenteerd in de USA (patent nummer 5'214'703, verstrijkt 25 mei 2010), West Europa (maar niet Belgie, Luxemburg, Portugal, Denemarken, Noorwegen, IJsland; patent nummer 0482154, verstrijkt 16 mei 2011) en Japan (patent aangevraagd).

De policy met betrekking tot licenties wordt beschreven op [7] en [8]. De kosten (variërend van $15 per gebruiker tot ruim $2 per gebruiker bij vele duizenden gebruikers) staan beschreven op [9].

Het gebruik van IDEA voor onderzoek, en het niet-commercieel privégebruik is gratis.

3 DES

De Amerikaanse Digital Encryption Standard is een blokversleutelingssysteem ingevoerd in 1977 [35] en goedgekeurd voor gebruik voor 'non-classified data'. Amerikaanse overheidsdiensten waren tot voor kort verplicht dit systeem te gebruiken, behalve bij toepassingen die een hoge graad van veiligheid vereisten.

Algemeen heerst een groot wantrouwen tegen dit systeem ([20, 23, 34]). De ontwerpcriteria zijn geheim. Het systeem is afgeleid van het door de IBM ontwikkelde systeem Lucifer dat een 128-bit sleutel gebruikte, maar bij DES is de sleutellengte tot 56 bits gereduceerd. Zoals Diffie & Hellman [20] opmerkten betekent dit dat voor enkele miljoenen dollars een machine is te bouwen die een DES sleutel kraakt binnen enkele uren. Zo'n machine is in detail ontworpen door Wiener [40].

Bovendien is de structuur van de S-dozen verdacht—vooral S-doos 5 bevat een vreemde lineariteit [37]—het is goed mogelijk dat DES een ingebouwd valluik heeft waardoor sleutels snel gekraakt kunnen worden ook zonder een dure special-purpose machine te bouwen.

Bedoeld of niet, zoals Matsui [31, 32] laat zien kan deze lineariteit werkelijk gebruikt worden om DES sleutels te vinden. Hij vond een DES sleutel die gebruikt was om een gigantische hoeveelheid klare tekst te versleutelen met 12 HP werkstations in 50 dagen.

Ook bij realistisch gebruik kan DES tegenwoordig gekraakt worden, gewoon met brute kracht. Op 26 februari 1998 werd de tweede RSA DES Challenge voltooid: een DES sleutel werd gevonden in 39 dagen, door 50000 machines op het internet in parallel te laten rekenen.

Dit betekent natuurlijk dat DES vandaag niet echt veilig is, en niet langer bruikbaar is voor serieuze toepassingen.[1]

[1]Dit werd geschreven in Maart 1998. Op 17 Juli 1998 kwam de mededeling (zie [18]) van EFF (the Electronic Frontier Foundation): *To prove the insecurity of DES, EFF built the first unclassified hardware for cracking messages encoded with it. On Wednesday of this week the EFF DES Cracker, which was built for less than $250,000, easily won RSA Laboratory's "DES Challenge II" contest and a $10,000 cash prize. It took the machine less than 3 days to complete the challenge, shattering the previous record of 39 days set by a massive network of tens of thousands of computers. The research results are fully documented in a book published this week by EFF and O'Reilly and Associates, entitled "Cracking DES: Secrets of Encryption Research, Wiretap Politics, and Chip Design."* De precieze constructie, met code en schema's, van de bijbehorende hardware en software, is gepubliceerd in [17].

Het vermoeden dat de NSA geen moeite zou hebben met het lezen van met DES versleutelde boodschappen bestond al lang, maar nu is het dus zo dat elke burger die er $250000 voor over heeft zulke boodschappen in een paar dagen kan kraken; bij een iets grotere investering zelfs binnen een half uur ([19]):

4 Exportbeperkingen

De Amerikaanse overheid (ITAR, International Traffic in Arms Regulations) beschouwt de export van cryptografische systemen als wapenexport, en voor export is een vergunning vereist. Zo'n vergunning wordt alleen verleend bij cryptografische systemen die een sleutellengte van ten hoogste 40 bits kennen (en, onder zekere voorwaarden, voor DES). Maar 2^{40} operaties is niet veel—dat is het aantal instructies dat een modale PC per dag uitvoert. Dit betekent dat cryptografische systemen en software afkomstig uit de Verenigde Staten volledig onbruikbaar zijn voor serieuze toepassingen in Europa.

(Recentelijk hebben we in de krant kunnen lezen dat Zweedse parlementariërs verontwaardigd waren toen ze er achter kwamen dat in de Microsoft pakketten die ze gebruikten de cryptografische opties gekortwiekt waren en dat met 40-bit sleutels versleutelde teksten on-line gedecodeerd kunnen worden. Deze pakketten vermelden vaak 'International Security' of 'Export Security'. Dit zijn synoniemen voor 'No Security'. Ames [1] betoogt dat de Amerikaanse restricties op de export van sterke cryptografie niet langer een militair doel hebben. Het op zeer grote schaal afluisteren van alle electronische communicatie zou vandaag vooral uit economische motieven gebeuren.)

De Amerikaanse industrie is hoogst ongelukkig met de huidige situatie. Om te beginnen is het buitengewoon onduidelijk wat de huidige situatie precies is. (Voor enige informatie, zie het Cryptorecht Overzicht van Bert-Jaap Koops [27].) Sinds januari van dit jaar is het Commerce Department verantwoordelijk voor het toezicht op de export van cryptografie. Een van de voorwaarden voor een exportvergunning is dat een vorm van 'key escrow' mogelijk is of weldra mogelijk zal zijn. Deze regels werkten niet, en na dit op 15 april publiekelijk te hebben vastgesteld, kondigde Commerce Secretary William Daley op 7 juli 1998 aan dat de exportbeperkingen voor encryptie software spoedig versoepeld zouden worden: systemen die door de Amerikaanse overheid goedgekeurd zijn zouden dan aan banken en dergelijke bedrijven geleverd mogen worden. (Zie Appendix G.)

5 IDEA

IDEA is afkomstig uit Zwitserland, en is niet onderhevig aan exportbeperkingen. IDEA gebruikt een 128-bit sleutel, en het met brute kracht kraken van een IDEA sleutel duurt 4722366482869645213696 keer zo lang als het kraken van een DES sleutel. Als de snelheid van de beschikbare computers elke 18 maanden verdubbelt, dan is IDEA nog veilig (voor een brute kracht aanval) tot honderd

Using current technology, a DES key can be recovered with a custom-designed $1 million machine in just 35 minutes. For attackers who lack the resources to design a chip and build such a machine, there are programmable forms of hardware such as FPGAs and CPLDs which can search the DES key space much faster than is possible using software on PCs and workstations. Attempts to thwart key search attacks by avoiding known plaintext and changing keys frequently are largely ineffective. The best course of action is to use a strong encryption algorithm with longer keys, such as triple-DES, 128-bit RC5, or CAST-128.

jaar nadat DES onbruikbaar werd. Net als voor DES, is ook voor IDEA snelle hardware verkrijgbaar (zie [10]). Hieronder in Appendix B wat metingen van de snelheid van software implementaties. Ruwweg mag men vandaag van software 1 MB/sec en van hardware 10 MB/sec verwachten, wat meer in ECB mode ('electronic codebook', zie sectie 10 hieronder). Software implementaties van IDEA zijn ruwweg twee maal zo snel als software implementaties van DES [36]. IDEA is vooral bekend als het systeem dat Zimmermann koos voor PGP (Pretty Good Privacy, [41]).

6 Bekende zwakten van IDEA

Van IDEA is slechts één zwak punt bekend: er zijn zwakke sleutels—sleutels waarvan bij een aanval met gekozen klare tekst direct vastgesteld kan worden dat een ervan gebruikt is, en die daarna ook eenvoudig zijn terug te vinden (Daemen et al. [15]). Dit is geen werkelijk probleem. Het eenvoudigst is, het te negeren: de kans dat bij een willekeurige keuze van een 128-bit sleutel een van deze zwakke sleutels gekozen wordt is astronomisch klein (10^{-20}). Maar natuurlijk kan een werkelijk paranoïde programma een sleutel verwerpen als het een van deze zwakke sleutels is. Een afdoende test is controleren of de tweede en derde byte van de sleutel niet beide NUL zijn. Voor details zie Appendix D.

Meer in het algemeen is IDEA iets zwakker als sommige van de gebruikte deelsleutels NUL zijn, en het is waarschijnlijk een goed idee om geen sleutels te gebruiken die een NUL byte bevatten.

Een algemene zwakte van dit type systemen is de 'timing attack' van Kocher [26]: meet met grote precisie hoe lang versleutelingen duren. Dit lekt enige informatie over de structuur van de sleutel als bij de implementatie er niet voor gezorgd is dat alle operaties een tijdsduur hebben die onafhankelijk is van de sleutel en de versleutelde boodschap. Dit soort aanval vergt enkele duizenden versleutelingen met dezelfde sleutel voordat hij enige informatie oplevert.

(Een klein beetje uitvoeriger: een versleuteling in IDEA bestaat uit 8 ronden, zie hieronder. In elk van de ronden wordt een 16-bit deelstuk van de sleutel gebruikt, en in het algoritme wordt o.a. een waarde vermenigvuldigd met deze deelsleutel. Omdat de vermenigvuldiging plaats vindt modulo $2^{16} + 1$ is 0 een speciaal geval waar het programma eerst op test. Kortom, als een deelsleutel 0 is, of als de waarde die met die deelsleutel vermenigvuldigd wordt 0 is, dan gaat de bijbehorende ronde iets sneller. Kelsey, Schneier & Wagner [25] rapporteren een verschil van 3 sec by 1000000 encrypties op een 33MHz 486SX. Dit is een meetbaar verschil (en als nauwkeurige tijdmeting beschikbaar is dit verschil al duidelijk na veel minder encrypties).)

Dit soort overwegingen leidt tot de aanbeveling te vermijden dat zeer grote hoeveelheden data (vele gigabytes) met dezelfde sleutel vercijferd worden.

Tenslotte is de blokgrootte van IDEA slechts 64 bits. Dat betekent dat als

2^{32} blokken versleuteld zijn, er een niet verwaarloosbare kans (40%) is dat twee ervan dezelfde code opleverden. ('Birthday paradox') Het hangt van het gebruikte protocol af of dit een bezwaar is, maar in het algemeen geldt hier weer: het is een slecht idee om vele gigabytes (2^{32} blokken is 32 GB) met dezelfde sleutel te vercijferen.

7 Quantumberekeningen

Een tamelijk nieuw idee is om de (volgens het huidige model van de quantummechanica) in de natuur aanwezige onzekerheid te gebruiken om een enkel deeltje een groot aantal berekeningen tegelijk te laten uitvoeren. Het is onduidelijk of de huidige quantummechanica in dit opzicht de natuur correct beschrijft, en zo ja of het ooit mogelijk zal zijn om dit principe om te zetten in een werkende computer. Maar er wordt zeer veel onderzoek naar dit soort computers gedaan, en er zijn al wat bescheiden succesjes. Mochten deze computers weldra geconstrueerd worden dan vervalt een groot deel van de huidige cryptografie. De berekeningsmogelijkheden van een quantumcomputer zijn slecht vergelijkbaar met die van een klassieke computer (cf. [39]), maar sommige problemen kan een quantumcomputer miljarden keren sneller oplossen dan een klassieke computer, en tot die problemen horen ook het factoriseren van grote getallen (en dus het kraken van RSA; Shor [38]) en het vinden van informatie in een grote database (en daaruit kan een snel algoritme gedestilleerd worden om systemen als DES en IDEA te kraken, [22]).

8 Beschrijving van IDEA

Zoals al gezegd is IDEA een blokcijfersysteem dat 64 bits brontekst met behulp van een 128-bit sleutel vercijfert tot 64 bits code. Meer in detail gaat dit als volgt:

Bekijk drie groepen van orde 2^{16}. De eerste groep is de elementair Abelse groep, met als groepsoperatie de bitsgewijze XOR (aangegeven met $\oplus$) tussen twee 16-bit woorden. De tweede groep is de cyclische groep van de gehele getallen modulo 2^{16}, met groepsoperatie aangegeven door $+$. De derde groep is de multiplicatieve groep van de niet-nul resten modulo $2^{16}+1$, met . als groepsoperatie. (Dit is inderdaad een groep, want $2^{16}+1$ is een priemgetal.) In dit laatste geval wordt 0 als notatie voor $2^{16} = 65536$ gebruikt. Elk van deze groepen werkt lineair, en behandelt de elementen op een uniforme manier, maar door deze drie groepsoperaties op geschikte manier te mengen ontstaat een zeer niet-lineaire operatie.

De versleuteling bestaat uit acht en een halve ronde. Elke volle ronde heeft als invoer en uitvoer vier blokken van 16 bits, en gebruikt zes sleutels van 16 bits elk. Een ronde bestaat uit een voorronde, die bij invoer (X_1, X_2, X_3, X_4) en

vier sleutels K_1, K_2, K_3, K_4 als uitvoer $(T_1, T_2, T_3, T_4) := (X_1.K_1, X_2 + K_2, X_3 + K_3, X_4.K_4)$ aflevert, en een vervolg, dat uit (T_1, T_2, T_3, T_4) en nog twee sleutels K_5, K_6 eerst $S_1 := K_5.(T_1 \oplus T_3)$ en $S_2 := K_6.(S_1 + (T_2 \oplus T_4))$ en $S_3 := S_1 + S_2$ uitrekent, en dan als uiteindelijk resultaat van de ronde $(Y_1, Y_2, Y_3, Y_4) = (T_1 \oplus S_2, T_2 \oplus S_3, T_3 \oplus S_2, T_4 \oplus S_3)$ geeft.

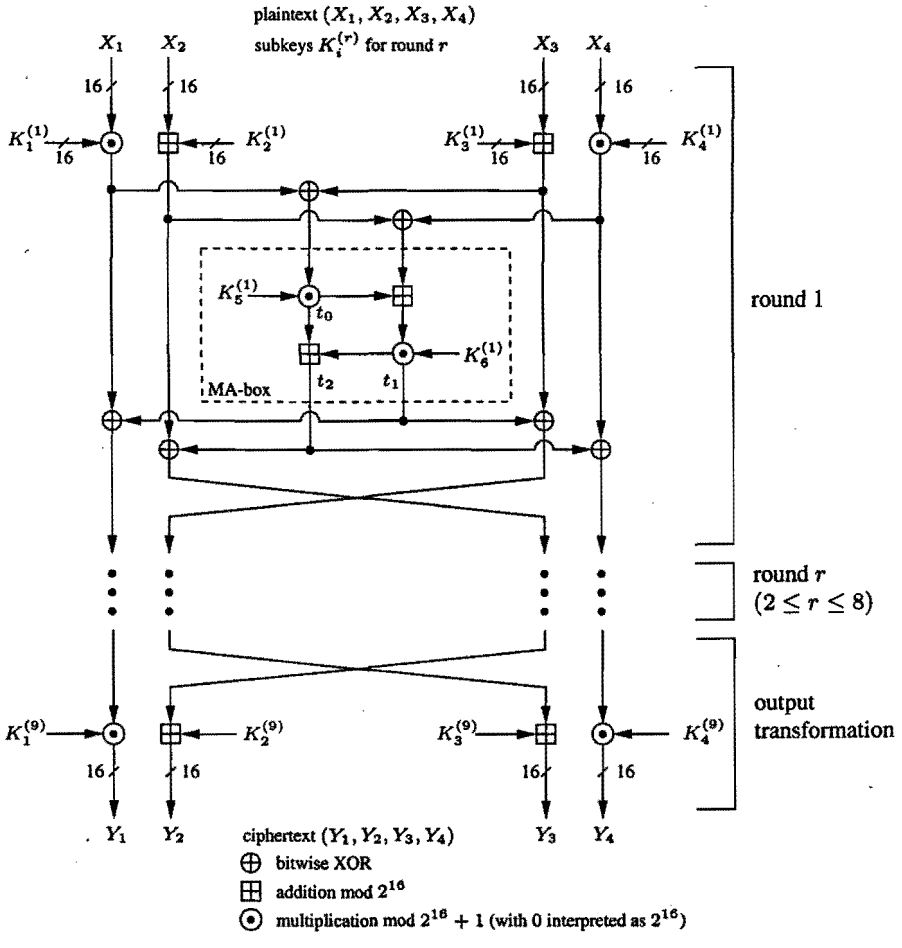


Figure 7.11: *IDEA computation path.*

De invoer van ronde 1 is de 64-bit brontekst, verdeeld in vier 16-bit blokken. De invoer van ronde i is de uitvoer van ronde $i-1$ waarin het tweede en derde 16-bit blok verwisseld zijn ($i = 2, ..., 8$). De laatste halve ronde is alleen een voorronde, met als invoer de uitvoer van ronde 8 (zonder verwisseling).

Programmatekst:

```
void cipher_idea(word16 in[4], word16 out[4], IDEAkey Z) {
        uint16 x1, x2, x3, x4, t1, t2;
        uint16 t16;
        word32 t32;
```

```
        int r = 8; /* 8 ronden */

        x1 = *in++;  x2 = *in++;
        x3 = *in++;  x4 = *in;
        do {
                /* voorronde */
                MUL(x1,*Z++);
                x2 += *Z++;
                x3 += *Z++;
                MUL(x4, *Z++);

                /* vervolg */
                t2 = x1^x3;
                MUL(t2, *Z++);
                t1 = t2 + (x2^x4);
                MUL(t1, *Z++);
                t2 = t1+t2;

                x1 ^= t1;
                x4 ^= t2;

                t2 ^= x2;
                x2 = x3^t1;
                x3 = t2;
        } while (--r);

        /* nu nog een voorronde, en maak de verwisseling ongedaan */
        MUL(x1, *Z++);
        *out++ = x1;
        *out++ = x3 + *Z++;
        *out++ = x2 + *Z++;
        MUL(x4, *Z);
        *out = x4;
}
```

In totaal worden dus $8 \times 6 + 4 = 52$ 16-bit sleutels gebruikt, dat is 832 bits. Deze sleutels worden gekozen als deelstukken van de 128-bit sleutel K: deze wordt eerst in 8 stukken van 16 bits opgedeeld, genummerd met de meest significante bits eerst, en dit levert de eerste 8 deelsleutels; daarna wordt K 25 bits cyclisch verschoven (naar links) en weer verdeeld in 8 stukken, dat levert de eerstvolgende 8 deelsleutels, enz.

Programmatekst:

```
void en_key_idea(word16 *userkey, word16 *Z) {
```

```
        int i,j;

        for (j=0; j<8; j++)
                Z[j] = *userkey++;

        for (i=0; j<KEYLEN; j++) {
                i++;
                Z[i+7] = Z[i & 7] << 9 | Z[i+1 & 7] >> 7;
                Z += i & 8;
                i &= 7;
        }
}
```

De laatste halve ronde dient om het algoritme symmetrisch te maken, zodat hetzelfde programma of dezelfde hardware zowel encryptie als decryptie kan doen. Merk op dat de tweede helft van elke ronde een involutie is: twee keer toepassen van deze operatie brengt de oorspronkelijke toestand terug.

Een eenvoudige berekening leidt uit de 52 deelsleutels die bij versleuteling gebruikt worden de 52 deelsleutels af die bij ontsleuteling gebruikt worden. Is deze sleutel eenmaal berekend dan gaat ontsleutelen via precies hetzelfde algoritme als versleuteling. Beide operaties zijn dan ook even snel.

9 Gebruik van IDEA

Zoals hierboven beschreven versleutelt IDEA blokken van 64 bits (8 bytes) met behulp van een 128-bit sleutel (weer tot blokken van 64 bits). Dit betekent dat bij een protocol waarbij kleinere hoeveelheden informatie versleuteld moeten worden (bijvoord een telnet sessie waarbij de toetsaanslagen byte voor byte verstuurd worden), er dummy informatie toegevoegd moet worden om aan het minimum van 8 bytes te komen. Dit is zelden een probleem.

(Iets uitvoeriger: Stel dat de dummy informatie altijd uit zeven spaties bestaat, en dat ECB ('electronic codebook') als protocol gebruikt wordt. Dan is de vercijfering een eenvoudige substitutie geworden, en het is kinderspel die te breken. Dus men moet hetzij CFB of zoiets gebruiken (dan is er helemaal geen vulsel nodig), hetzij, als toch ECB gebruikt wordt, het vulsel door een random generator laten produceren.)

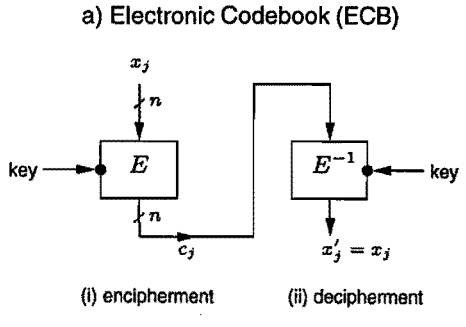
Het gebruik levert dus geen bijzondere problemen, maar natuurlijk is sleutelbeheer, protocol, enz. enz. van wezenlijk belang voor de veiligheid van het gehele systeem.

10 Gebruik van IDEA - protocollen

Hieronder in het kort beschrijvingen van de standaardprotocollen die door de Ascom IDEA chip ondersteund worden.

ECB

IDEA is een paar transformaties (E voor encryptie, D voor decryptie) die gegeven een 128-sleutel S en een 64-bit boodschap B een 64-bit versleuteling $C = E(S, B)$ oplevert, en gegeven dezelfde sleutel en de versleuteling C weer de oorspronkelijke boodschap $B = D(S, C)$ terugvindt. De eenvoudigste manier om zoiets te gebruiken bij een boodschap van willekeurige lengte is die op te hakken in stukken van 64 bits (8 bytes), en elk stuk afzonderlijk te versleutelen. Deze manier van gebruik wordt ECB (Electronic Codebook mode) genoemd. Meestal is dit een slecht plan: gelijke fragmenten in de brontekst worden nu op dezelfde manier versleuteld, en de afluisteraar verkrijgt eenvoudig enige informatie. Bij deze methode kan een tegenstander ook betrekkelijk eenvoudig een boodschap onderscheppen en vervangen door een lichtelijk gewijzigde boodschap (zonder dat hij eerst de code moet breken).



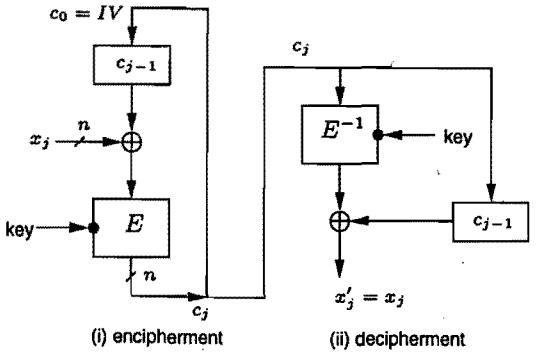
CBC

Een veel beter plan is om de versleuteling van elk blok te laten afhangen van alle voorafgaande tekst. Dit kost nauwelijks extra tijd: Het CBC (Cipher Block Chaining) protocol produceert in stap t ($t = 1, 2, ...$) de code $C_t = E(S, B_t \oplus C_{t-1})$, d.w.z., in elke stap wordt voor vercijfering eerst de code uit de vorige stap bij het huidige stuk boodschap opgeteld (optelling met XOR). Bij het ontcijferen krijg je dan $B_t = D(S, C_t) \oplus C_{t-1}$. Nu leveren gelijke fragmenten in de brontekst verschillende versleutelingen op. Blijft over de vraag hoe C_0 gekozen moet worden. Gewoon 0 kiezen (of een andere vaste waarde gebruiken) is slecht omdat dan twee boodschappen die hetzelfde beginnen, tot aan het punt waar ze voor het eerst verschillen op dezelfde manier vercijferd worden. De oplossing is om voor stap 1 nog een stap 0 in te lassen waar een random boodschap vercijferd

wordt ($C_{-1} = 0$, B_0 toevallig gekozen, iedere keer anders). De ontvanger kan B_0 gewoon weggooien. Nu heeft een afluisteraar geen houvast meer. (De IDEA chip van Ascom heeft geen mening over hoe C_0 gekozen moet worden, en vraagt van de gebruiker om expliciet een waarde aan te geven: de Initial Vector (IV).) Bij gebruik over een kanaal met ruis levert ECB bij een bitfout tijdens het versturen van C_t één fout blok B_t op; CBC levert hier twee foute blokken B_t en B_{t+1}.

Terwijl bij ECB gelijke codefragmenten duiden op gelijke tekstfragmenten, geldt hier dat gelijke codefragmenten informatie geven over het verschil van de bijbehorende tekstfragmenten (cf. [11]): Als $C_r = C_t$ dan is $E(S, B_r \oplus C_{r-1}) = E(S, B_t \oplus C_{t-1})$, dus $B_r \oplus C_{r-1} = B_t \oplus C_{t-1}$, dus $B_r \oplus B_t = C_{r-1} \oplus C_{t-1}$, wat aan de afluisteraar bekend is. Zoals boven al aangestipt is de kans op zulke coincidenties niet verwaarloosbaar als zeer grote hoeveelheden gegevens met dezelfde sleutel vercijferd worden.

b) Cipher-block Chaining (CBC)

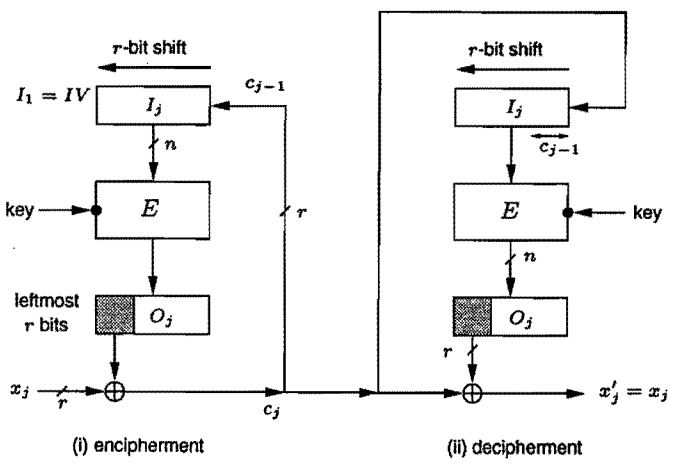


CFB

Een andere manier om terugkoppeling in te bouwen is het genereren van een 'one-time pad' door telkens de vorige code weer te versleutelen. Dit wordt CFB (Cipher Feedback) genoemd. Dit levert vercijfering $C_t = B_t \oplus E(S, C_{t-1})$ en ontcijfering $B_t = C_t \oplus E(S, C_{t-1})$. Omdat hier de IDEA operatie niet op B_t wordt toegepast maar alleen op de voorafgaande cijfertekst, is het makkelijk om varianten te maken die met brokken brontekst werken van een andere lengte dan 64 bits. De IDEA chip van Ascom kent dit protocol toegepast op stukken brontekst van 1, 8, 16, 32 of 64 bits, waarbij dan telkens de laatste 64, 8, 4, 2 of 1 stukken code als invoer voor de encryptiefunctie gebruikt worden (en code oplevert die evenlang is als de brontekst). Ook hier moet weer een random beginboodschap gekozen worden. Een nadeel van dit protocol is de directe samenhang van boodschap en code. Als een aanvaller de boodschap, of het formaat van de boodschap, kent, en hij bovendien de transmissie kan onderscheppen en vervangen door een eigen brouwsel, kan hij eenvoudig de codeboodschap op een plaats wijzigen zodat daar komt te staan wat hij wil; de ontvanger merkt pas verderop dat er iets is

misgegaan. Meer in het bijzonder kan het eind van een boodschap ongestraft worden gewijzigd. (Kortom, een boodschap moet altijd een serienummer en een tijdstempel vooraan hebben, en een checksum achteraan, zodat het veranderen van de staart van een boodschap geen schade kan doen.)

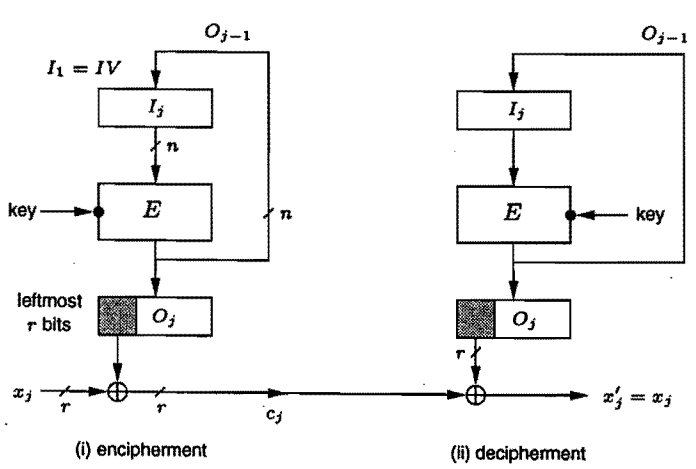
c) Cipher feedback (CFB), r-bit characters/r-bit feedback



OFB

Bij CFB wordt de codeboodschap teruggekoppeld om nieuwe bits voor een one-time pad te maken. Je kunt ook de bits die het pad recentelijk heeft afgeleverd terugkoppelen. Dit heet OFB (Output Feedback) en levert iets als $P_t = E(S, P_{t-1})$ (bits van het one-time pad), $C_t = B_t \oplus P_t$ (vercijfering), $B_t = C_t \oplus P_t$ (ontcijfering). In tegenstelling tot CFB is OFB zwak als de feedback korte stukken pad gebruikt ([21, 24]).

d) Output feedback (OFB), r-bit characters/n-bit feedback



Appendix A: Broncode

```
/* idea.c - C source code for IDEA block cipher.
 * IDEA (International Data Encryption Algorithm), formerly known as
 * IPES (Improved Proposed Encryption Standard).
 * Algorithm developed by Xuejia Lai and James L. Massey, of ETH Zurich.
 * This implementation modified and derived from original C code
 * developed by Xuejia Lai.
 * Zero-based indexing added, names changed from IPES to IDEA.
 * CFB functions added.  Random number routines added.
 *
 *  Optimized for speed 21 Oct 92 by Colin Plumb <colin@nsq.gts.org>
 *
 * There are two adjustments that can be made to this code to
 * speed it up.  Defaults may be used for PCs.  Only the -DIDEA32
 * pays off significantly if selectively set or not set.
 * Experiment to see what works better for you.
 *
 * Multiplication: default is inline, -DAVOID_JUMPS uses a
 * different version that does not do any conditional
 * jumps (a few percent worse on a SPARC), while
 * -DSMALL_CACHE takes it out of line to stay
 * within a small on-chip code cache.
 * Variables: normally, 16-bit variables are used, but some
 * machines (notably RISCs) do not have 16-bit registers,
 * so they do a great deal of masking.  -DIDEA32 uses "int"
 * register variables and masks explicitly only where
 * necessary.  On a SPARC, for example, this boosts
 * performace by 30%.
 *
 * The IDEA(tm) block cipher is covered by a patent held by ETH and a
 * Swiss company called Ascom-Tech AG.  The Swiss patent number is
 * PCT/CH91/00117.  International patents are pending. IDEA(tm) is a
 * trademark of Ascom-Tech AG.  There is no license fee required for
 * noncommercial use.  Commercial users may obtain licensing details
 * from Dieter Profos, Ascom Tech AG, Solothurn Lab, Postfach 151, 4502
 * Solothurn, Switzerland, Tel +41 65 242885, Fax +41 65 235761.
 *
 * The IDEA block cipher uses a 64-bit block size, and a 128-bit key
 * size.  It breaks the 64-bit cipher block into four 16-bit words
 * because all of the primitive inner operations are done with 16-bit
 * arithmetic.  It likewise breaks the 128-bit cipher key into eight
 * 16-bit words.
 *
 * For further information on the IDEA cipher, see these papers:
 * 1) Xuejia Lai, "Detailed Description and a Software Implementation of
 *       the IPES Cipher", Institute for Signal and Information
 *       Processing, ETH-Zentrum, Zurich, Switzerland, 1991
 * 2) Xuejia Lai, James L. Massey, Sean Murphy, "Markov Ciphers and
 *       Differential Cryptanalysis", Advances in Cryptology- EUROCRYPT'91
 *
 * This code assumes that each pair of 8-bit bytes comprising a 16-bit
 * word in the key and in the cipher block are externally represented
 * with the Most Significant Byte (MSB) first, regardless of the
 * internal native byte order of the target CPU.
 */

#include "idea.h"

#ifdef TEST
#include <stdio.h>
#include <time.h>
#endif

#define ROUNDS 8 /* Don't change this value, should be 8 */
#define KEYLEN (6*ROUNDS+4) /* length of key schedule */

typedef word16 IDEAkey[KEYLEN];

#ifdef IDEA32 /* Use >16-bit temporaries */
#define low16(x) ((x) & 0xFFFF)
typedef unsigned int uint16; /* at LEAST 16 bits, maybe more */
#else
#define low16(x) (x) /* this is only ever applied to uint16's */
typedef word16 uint16;
#endif

#ifdef _GNUC_
/* __const__ simply means there are no side effects for this function,
 * which is useful info for the gcc optimizer */
#define CONST __const__
#else
#define CONST
#endif

static void en_key_idea(word16 userkey[8], IDEAkey Z);
static void de_key_idea(IDEAkey Z, IDEAkey DK);
```

```
static void cipher_idea(word16 in[4], word16 out[4], CONST IDEAkey Z);

/*
 * Multiplication, modulo (2**16)+1
 * Note that this code is structured like this on the assumption that
 * untaken branches are cheaper than taken branches, and the compiler
 * doesn't schedule branches.
 */
#ifdef SMALL_CACHE
CONST static uint16 mul(register uint16 a, register uint16 b)
{
register word32 p;

if (a)
{ if (b)
{ p = (word32)a * b;
b = low16(p);
a = p>>16;
return b - a + (b < a);
}
else
{ return 1-a;
}
}
else
{ return 1-b;
}
}       /* mul */
#endif /* SMALL_CACHE */

/*
 * Compute multiplicative inverse of x, modulo (2**16)+1,
 * using Euclid's GCD algorithm.  It is unrolled twice to
 * avoid swapping the meaning of the registers each iteration,
 * and some subtracts of t have been changed to adds.
 */
CONST static uint16 inv(uint16 x)
{
uint16 t0, t1;
uint16 q, y;

if (x <= 1)
return x; /* 0 and 1 are self-inverse */
t1 = 0x10001 / x; /* Since x >= 2, this fits into 16 bits */
y = 0x10001 % x;
if (y == 1)
return low16(1-t1);
t0 = 1;
do
{ q = x / y;
x = x % y;
t0 += q * t1;
if (x == 1)
return t0;
q = y / x;
y = y % x;
t1 += q * t0;
} while (y != 1);
return low16(1-t1);
} /* inv */

/* Compute IDEA encryption subkeys Z */
static void en_key_idea(word16 *userkey, word16 *Z)
{
int i,j;

/*
 * shifts
 */
for (j=0; j<8; j++)
Z[j] = *userkey++;

for (i=0; j<KEYLEN; j++)
{ i++;
Z[i+7] = Z[i & 7] << 9 | Z[i+1 & 7] >> 7;
Z += i & 8;
i &= 7;
}
}       /* en_key_idea */

/* Compute IDEA decryption subkeys DK from encryption subkeys Z */
/* Note: these buffers *may* overlap! */
static void de_key_idea(IDEAkey Z, IDEAkey DK)
{
int j;
uint16 t1, t2, t3;
IDEAkey T;
word16 *p = T + KEYLEN;
```

```
t1 = inv(*Z++);
t2 = -*Z++;
t3 = -*Z++;
*--p = inv(*Z++);
*--p = t3;
*--p = t2;
*--p = t1;

for (j = 1; j < ROUNDS; j++)
{
t1 = *Z++;
*--p = *Z++;
*--p = t1;

t1 = inv(*Z++);
t2 = -*Z++;
t3 = -*Z++;
*--p = inv(*Z++);
*--p = t2;
*--p = t3;
*--p = t1;
}
t1 = *Z++;
*--p = *Z++;
*--p = t1;

t1 = inv(*Z++);
t2 = -*Z++;
t3 = -*Z++;
*--p = inv(*Z++);
*--p = t3;
*--p = t2;
*--p = t1;
/* Copy and destroy temp copy */
for (j = 0, p = T; j < KEYLEN; j++)
{
*DK++ = *p;
*p++ = 0;
}
} /* de_key_idea */

/*
 * MUL(x,y) computes x = x*y, modulo 0x10001.  Requires two temps,
 * t16 and t32.  x must me a side-effect-free lvalue.  y may be
 * anything, but unlike x, must be strictly 16 bits even if low16()
 * is #defined.
 * All of these are equivalent - see which is faster on your machine
 */
#ifdef SMALL_CACHE
#define MUL(x,y) (x = mul(low16(x),y))
#else
#ifdef AVOID_JUMPS
#define MUL(x,y) (x = low16(x-1), t16 = low16((y)-1), \
t32 = (word32)x*t16+x+t16+1, x = low16(t32), \
t16 = t32>>16, x = x-t16+(x<t16) )
#else
#define MUL(x,y) ((t16 = (y)) ? (x=low16(x)) ? \
 t32 = (word32)x*t16, x = low16(t32), t16 = t32>>16, \
 x = x-t16+(x<t16) : \
 (x = 1-t16) : (x = 1-x))
#endif
#endif

/* IDEA encryption/decryption algorithm */
/* Note that in and out can be the same buffer */
static void cipher_idea(word16 in[4], word16 out[4], register CONST IDEAkey Z)
{
register uint16 x1, x2, x3, x4, t1, t2;
register uint16 t16;
register word32 t32;

int r = ROUNDS;

x1 = *in++;  x2 = *in++;
x3 = *in++;  x4 = *in;
do
{
MUL(x1,*Z++);
x2 += *Z++;
x3 += *Z++;
MUL(x4, *Z++);

t2 = x1^x3;
MUL(t2, *Z++);
t1 = t2 + (x2^x4);
MUL(t1, *Z++);
t2 = t1+t2;
```

```
x1 ^= t1;
x4 ^= t2;

t2 ^= x2;
x2 = x3^t1;
x3 = t2;
} while (--r);
MUL(x1, *Z++);
*out++ = x1;
*out++ = x3 + *Z++;
*out++ = x2 + *Z++;
MUL(x4, *Z);
*out = x4;
} /* cipher_idea */

/*---------------------------------------------------------------*/

#ifdef TEST
/*
 * This is the number of Kbytes of test data to encrypt.
 * It defaults to 1 MByte.
 */
#ifndef KBYTES
#define KBYTES 1024
#endif

void main(void)
{ /* Test driver for IDEA cipher */
int i, j, k;
IDEAkey Z, DK;
word16 XX[4], TT[4], YY[4];
word16 userkey[8];
clock_t start, end;
long l;

/* Make a sample user key for testing... */
for(i=0; i<8; i++)
userkey[i] = i+1;

/* Compute encryption subkeys from user key... */
en_key_idea(userkey,Z);
printf("\nEncryption key subblocks: ");
for(j=0; j<ROUNDS+1; j++)
{
printf("\nround %d:   ", j+1);
if (j==ROUNDS)
for(i=0; i<4; i++)
printf(" %6u", Z[j*6+i]);
else
for(i=0; i<6; i++)
printf(" %6u", Z[j*6+i]);
}

/* Compute decryption subkeys from encryption subkeys... */
de_key_idea(Z,DK);
printf("\nDecryption key subblocks: ");
for(j=0; j<ROUNDS+1; j++)
{
printf("\nround %d:   ", j+1);
if (j==ROUNDS)
for(i=0; i<4; i++)
printf(" %6u", DK[j*6+i]);
else
for(i=0; i<6; i++)
printf(" %6u", DK[j*6+i]);
}

/* Make a sample plaintext pattern for testing... */
for (k=0; k<4; k++)
XX[k] = k;

printf("\n Encrypting %d KBytes (%ld blocks)...", KBYTES, KBYTES*64l);
fflush(stdout);
start = clock();
cipher_idea(XX,YY,Z);       /* encrypt plaintext XX, making YY */
for (l = 1; l < 64*KBYTES; l++)
cipher_idea(YY,YY,Z); /* repeated encryption */
cipher_idea(YY,TT,DK);      /* decrypt ciphertext YY, making TT */
for (l = 1; l < 64*KBYTES; l++)
cipher_idea(TT,TT,DK); /* repeated decryption */
end = clock() - start;
l = end * 1000. / CLOCKS_PER_SEC + 1;
i = l/1000;
j = l%1000;
l = KBYTES * 1024. * CLOCKS_PER_SEC / end;
printf("%d.%03d seconds = %ld bytes per second\n", i, j, l);
```

```
printf("\nX %6u   %6u  %6u  %6u \n",
  XX[0], XX[1],  XX[2], XX[3]);
printf("Y %6u   %6u  %6u  %6u \n",
  YY[0], YY[1],  YY[2], YY[3]);
printf("T %6u   %6u  %6u  %6u \n",
  TT[0], TT[1],  TT[2], TT[3]);

/* Now decrypted TT should be same as original XX */
for (k=0; k<4; k++)
if (TT[k] != XX[k])
{
printf("\n\07Error!  Noninvertable encryption.\n");
exit(-1); /* error exit */
}
printf("\nNormal exit.\n");
exit(0); /* normal exit */
}        /* main */


#endif /* TEST */


/*********************************************************************/


/*
 * xorbuf - change buffer via xor with random mask block
 * Used for Cipher Feedback (CFB) or Cipher Block Chaining
 * (CBC) modes of encryption.
 * Can be applied for any block encryption algorithm,
 * with any block size, such as the DES or the IDEA cipher.
 */
static void xorbuf(register byteptr buf, register byteptr mask,
register int count)
/* count must be > 0 */
{
if (count)
do
*buf++ ^= *mask++;
while (--count);
} /* xorbuf */


/*
 * cfbshift - shift bytes into IV for CFB input
 * Used only for Cipher Feedback (CFB) mode of encryption.
 * Can be applied for any block encryption algorithm with any
 * block size, such as the DES or the IDEA cipher.
 */
static void cfbshift(register byteptr iv, register byteptr buf,
register int count, int blocksize)
/*  iv is the initialization vector.
 * buf is the buffer pointer.
 * count is the number of bytes to shift in...must be > 0.
 * blocksize is 8 bytes for DES or IDEA ciphers.
 */
{
int retained;
if (count)
{
retained = blocksize-count; /* number bytes in iv to retain */
/* left-shift retained bytes of IV over by count bytes to make room */
while (retained--)
{
*iv = *(iv+count);
iv++;
}
/* now copy count bytes from buf to shifted tail of IV */
do *iv++ = *buf++;
while (--count);
}
} /* cfbshift */



/* Key schedules for IDEA encryption and decryption */
static IDEAkey Z, DK;
static word16 *iv_idea; /* pointer to IV for CFB or CBC */
static boolean cfb_dc_idea; /* TRUE iff CFB decrypting */


/* initkey_idea initializes IDEA for ECB mode operations */
void initkey_idea(byte key[16], boolean decryp)
{
word16 userkey[8]; /* IDEA key is 16 bytes long */
int i;
/* Assume each pair of bytes comprising a word is ordered MSB-first. */
for (i=0; i<8; i++)
```

```
{
userkey[i] = (key[0]<<8) + key[1];
key++; key++;
}
en_key_idea(userkey,Z);
if (decryp)
{
de_key_idea(Z,Z); /* compute inverse key schedule DK */
}
for (i=0; i<8; i++) /* Erase dangerous traces */
userkey[i] = 0;
} /* initkey_idea */


/* Run a 64-bit block thru IDEA in ECB (Electronic Code Book) mode,
using the currently selected key schedule.
*/
void idea_ecb(word16 *inbuf, word16 *outbuf)
{
/* Assume each pair of bytes comprising a word is ordered MSB-first. */
#ifndef HIGHFIRST /* If this is a least-significant-byte-first CPU */
word16 x;

/* Invert the byte order for each 16-bit word for internal use. */
x = inbuf[0]; outbuf[0] = x >> 8 | x << 8;
x = inbuf[1]; outbuf[1] = x >> 8 | x << 8;
x = inbuf[2]; outbuf[2] = x >> 8 | x << 8;
x = inbuf[3]; outbuf[3] = x >> 8 | x << 8;
cipher_idea(outbuf, outbuf, Z);
x = outbuf[0]; outbuf[0] = x >> 8 | x << 8;
x = outbuf[1]; outbuf[1] = x >> 8 | x << 8;
x = outbuf[2]; outbuf[2] = x >> 8 | x << 8;
x = outbuf[3]; outbuf[3] = x >> 8 | x << 8;
#else /* HIGHFIRST */
/* Byte order for internal and external representations is the same. */
cipher_idea(inbuf, outbuf, Z);
#endif /* HIGHFIRST */
} /* idea_ecb */


/*
 * initcfb - Initializes the IDEA key schedule tables via key,
 * and initializes the Cipher Feedback mode IV.
 * References context variables cfb_dc_idea and iv_idea.
 */
void initcfb_idea(word16 iv0[4], byte key[16], boolean decryp)
/*  iv0 is copied to global iv_idea, buffer will be destroyed by ideacfb.
key is pointer to key buffer.
decryp is TRUE if decrypting, FALSE if encrypting.
*/
{
iv_idea = iv0;
cfb_dc_idea = decryp;
initkey_idea(key,FALSE);
} /* initcfb_idea */


/*
 * ideacfb - encipher a buffer with IDEA enciphering algorithm,
 * using Cipher Feedback (CFB) mode.
 *
 * Assumes initcfb_idea has already been called.
 * References context variables cfb_dc_idea and iv_idea.
 */
void ideacfb(byteptr buf, int count)
/* buf is input, output buffer, may be more than 1 block.
 * count is byte count of buffer.  May be > IDEABLOCKSIZE.
 */
{
int chunksize; /* smaller of count, IDEABLOCKSIZE */
word16 temp[IDEABLOCKSIZE/2];

while ((chunksize = min(count,IDEABLOCKSIZE)) > 0)
{
idea_ecb(iv_idea,temp);  /* encrypt iv_idea, making temp. */

if (cfb_dc_idea) /* buf is ciphertext */
/* shift in ciphertext to IV... */
cfbshift((byte *)iv_idea,buf,chunksize,IDEABLOCKSIZE);

/* convert buf via xor */
xorbuf(buf,(byte *)temp,chunksize); /* buf now has enciphered output */

if (!cfb_dc_idea) /* buf was plaintext, is now ciphertext */
/* shift in ciphertext to IV... */
cfbshift((byte *)iv_idea,buf,chunksize,IDEABLOCKSIZE);

count -= chunksize;
```

```
buf += chunksize;
}
} /* ideacfb */


/*
close_idea function erases all the key schedule information when
we are all done with a set of operations for a particular IDEA key
context.  This is to prevent any sensitive data from being left
around in memory.
*/
void close_idea(void) /* erase current key schedule tables */
{
short i;
for (i = 0; i < KEYLEN; i++)
Z[i] = 0;
} /* close_idea() */

/*********************************************************************/

/*
 * These buffers are used by init_idearand, idearand, and close_idsarand.
 */
static word16 dtbuf_idea[4] = {0}; /* buffer for enciphered timestamp */
static word16 randseed_idea[4] = {0}; /* seed for IDEA random # generator */
static word16 randbuf_idea[4] = {0}; /* buffer for IDEA random # generator */
static byte randbuf_idea_counter = 0; /* # of random bytes left in randbuf_idea */

/*
 * init_idearand - initialize idearand, IDEA random number generator.
 * Used for generating cryptographically strong random numbers.
 * Much of the design comes from Appendix C of ANSI X9.17.
 * key is pointer to IDEA key buffer.
 * seed is pointer to random number seed buffer.
 * tstamp is a 32-bit timestamp
 */
void init_idearand(byte key[16], byte seed[8], word32 tstamp)
{
int i;
initkey_idea(key, FALSE); /* initialize IDEA */

for (i=0; i<4; i++) /* capture timestamp material */
{ dtbuf_idea[i] = tstamp; /* get bottom word */
tstamp = tstamp >> 16; /* drop bottom word */
/* tstamp has only 4 bytes-- last 4 bytes will always be 0 */
}
/* Start with enciphered timestamp: */
idea_ecb(dtbuf_idea,dtbuf_idea);

/* initialize seed material */
for (i=0; i<8; i++)
((byte *)randseed_idea)[i] = seed[i];

randbuf_idea_counter = 0; /* # of random bytes left in randbuf_idea */

} /* init_idearand */


/*
 * idearand - IDEA pseudo-random number generator
 * Used for generating cryptographically strong random numbers.
 * Much of the design comes from Appendix C of ANSI X9.17.
 */
byte idearand(void)
{
int i;
if (randbuf_idea_counter==0) /* if random buffer is spent...*/
{ /* Combine enciphered timestamp with seed material: */
for (i=0; i<4; i++)
randseed_idea[i] ^= dtbuf_idea[i];
idea_ecb(randseed_idea,randbuf_idea); /* fill new block */

/* Compute new seed vector: */
for (i=0; i<4; i++)
randseed_idea[i] = randbuf_idea[i] ^ dtbuf_idea[i];
idea_ecb(randseed_idea,randseed_idea); /* fill new seed */

randbuf_idea_counter = 8; /* reset counter for full buffer */
}
/* Take a byte from randbuf_idea: */
return(((byte *)randbuf_idea)[--randbuf_idea_counter]);
} /* idearand */


void close_idearand(void)
{ /* Erase random IDEA buffers and wipe out IDEA key info */
int i;
for (i=0; i<4; i++)
```

```
{ randbuf_idea[i] = 0;
randseed_idea[i] = 0;
dtbuf_idea[i] = 0;
}
close_idea(); /* erase current key schedule tables */
} /* close_idearand */

/* end of idea.c */

/* idea.h - header file for idea.c     */

#include "usuals.h"  /* typedefs for byte, word16, boolean, etc. */

#define IDEAKEYSIZE 16
#define IDEABLOCKSIZE 8


void initcfb_idea(word16 iv0[4], byte key[16], boolean decryp);
void ideacfb(byteptr buf, int count);
void close_idea(void);

void init_idearand(byte key[16], byte seed[8], word32 tstamp);
byte idearand(void);
void close_idearand(void);


/* prototypes for passwd.c */

/* GetHashedPassPhrase - get pass phrase from user, hashes it to an IDEA key. */
int GetHashedPassPhrase(char *keystring, char *hash, boolean noecho);

/* hashpass - Hash pass phrase down to 128 bits (16 bytes). */
void hashpass (char *keystring, int keylen, byte *hash);
```

Appendix B: Snelheid

Bovenstaande code refereert aan "usuals.h". Een mogelijke invulling van dit bestand (in een omgeving met 32-bit integers) is

```
/* usuals.h */
typedef unsigned int word32;
typedef unsigned short int word16;
typedef unsigned char byte, boolean;
typedef byte *byteptr;

#define FALSE 0
#ifndef CLOCKS_PER_SEC
#define CLOCKS_PER_SEC 1000000
#endif
#define min(a,b) (((a)<(b)) ? (a) : (b))
```

Let op—het 'unsigned' hier is belangrijk!

Nu kan een test-versie van de code gecompileerd worden met het commando `cc -DTEST -o idea idea.c` , eventueel voorzien van een vlag die de compiler laat optimaliseren, zoals `-O4` (de toepasselijke vlaggen zijn machineafhankelijk), en mogelijk voorzien van vlaggen die alternatieve implementaties van stukken algoritme kiezen, zoals `-DIDEA32` of `-DSMALL_CACHE` of `-DAVOID_JUMPS` (hieronder aangegeven met '32', 'sc' en 'aj'). Uitproberen levert op de machines die ik hier in de buurt heb:

machine	snelheid (in bytes/sec)	snelheid met optimalisatie	type
Sparc IPC	49391	81411	32,sc
Sparc5 (sun4m)	129725	408552	32
66MHz 486/DX2	129613	240499	32
166MHz Pentium	388361	1059167	32
275MHz DEC Alpha	731930	1414679	32,sc
SGI Indy 100 MHz R4000	419430	794375	32,aj
SGI Indy 200 MHz R4400	852500	1747626	32,aj
SGI Indy 250 MHz R4400	1048576	2139951	32,aj
SGI Challenge 194 MHz R10000	1807889	4559026	32

Het blijkt dat optimalisatie de code op veel machines meer dan 50% sneller maakt, en dat de juiste keuze voor 32 of aj of sc vaak nog eens 50% scheelt. Op alle machines waarop ik dit geprobeerd heb bleek `-DIDEA32` het programma te versnellen. (Dit is niet zo verbazingwekkend: al deze machines hebben 32-bit of 64-bit integers; het werken met een datatype word16 is dan inefficiënt—voortdurend moet er gemaskeerd worden—en met 32 bits werken en alleen maskeren wanneer dat echt nodig is maakt veel uit. Dit betekent dat dit beter de default had kunnen zijn, met een `-DIDEA16` vlag om echt met word16's te werken.) Schneier [36] vermeldt tijden die iets hoger ligger, en vindt 110kB/sec op een 33MHz 386, en 300kB/sec op een 66MHz 486, 25% sneller dan wat ik vind op mijn 486. Zelfs in software is IDEA dus al redelijk snel—mijn oude DEC Alpha haalt al 1.4 MB/sec, en een snelle SGI haalt 4.5 MB/sec.

Er zijn verschillende hardware implementaties gemaakt. Schneier [36] vermeldt een chip die 22 MB/sec haalt, zie ook [14]. Ascom verkoopt chips die 300 Mb/sec (37 MB/sec) halen, zie [10]; de tekst van deze web pagina is de inhoud van de volgende appendix, gereproduceerd zonder permissie.

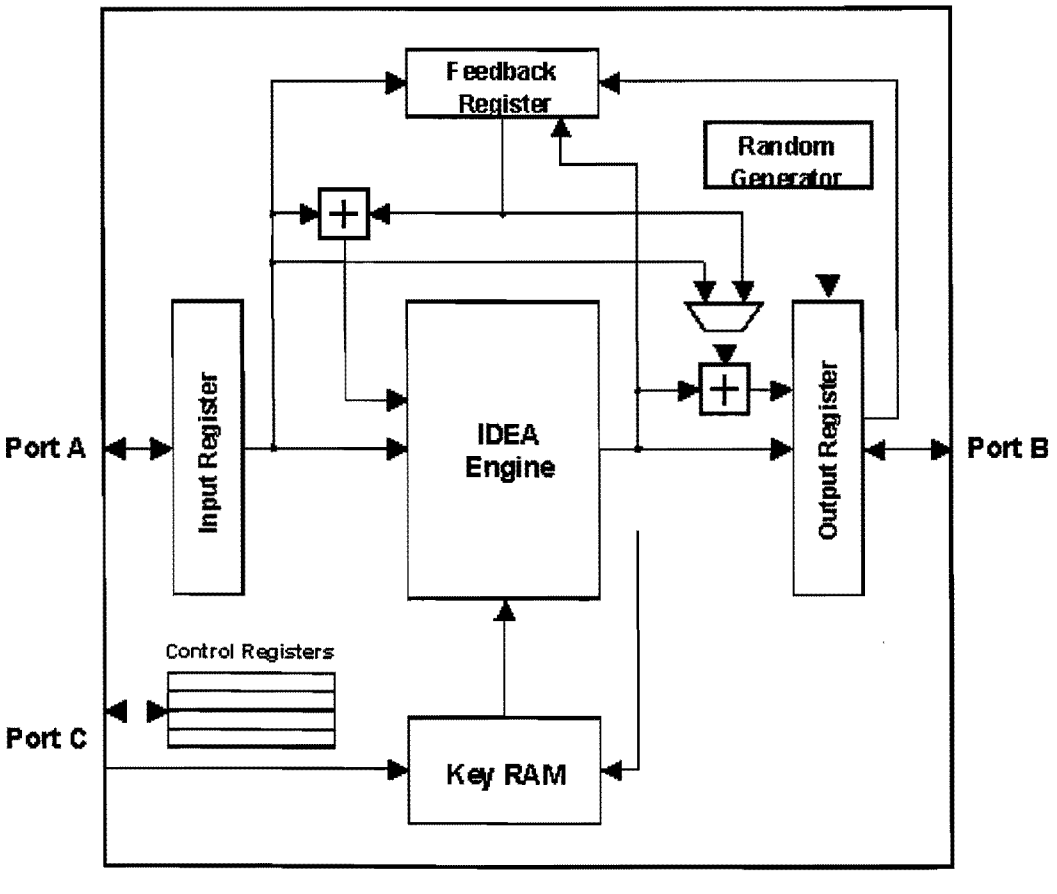
Appendix C: Ascom's IDEA chip informatie pagina

```
Preliminary Data Sheet

Overview

* High speed cryptographic device implementing the
  IDEA Algorithm
* Up to 300 Mbit/sec data rate (at 40 MHz clock rate)
* Multiple modes:
  ECB (Electronic Codebook)
  CBC (Cipher Block Chaining)
```

CFB (Cipher Feedback) 1,8,16,32,64 bit
OFB (Output Feedback)
Key Stream Generator
MAC computation

* Built-in random number generator
* Interface for several microprocessor families
* 2 data ports (8, 16, or 32 bit wide)
* Independent security control port (8 bit wide)
* Key storage for 32 keys
* Flexible key management for both master-session key and asymmetric key manage-ment environments
* Storage for 32 initial vector /saved contexts of 64 bit length along with the corresponding session keys
* Available in low profile TQFP 144 package

Block Diagram



Description

The IDEACrypt is a powerful cryptographic chip designed for system flexibility to ease secure system implementations. It is based on the IDEA algorithm which features very high security due to its proven design and the key length of 128 bit. It supports all common operating modes for block ciphers (ECB, CBC, CFB, and OFB) and can also be configured as a key stream generator for stream ciphers and for MAC computation.

The IDEACrypt chip implements the IDEA algorithm using a 3-stage pipeline. At a clock rate of 40 MHz, it provides a throughput of 300 Mbit/sec in ECB mode, and a throughput of 100 Mbit/sec in the other modes.

Interfaces

The IDEACrypt chip is extremely flexible with regard to its interfaces. It features two data ports which are optimized with respect to performance and can be configured with respect to their width (8, 16, or 32 bit). The configuration of the data ports is controlled via the port control register.

For security reasons, a separate 8-bit control port is provided for control and key management functions.

The IDEACrypt chip can be configured for direct interfacing with the following microprocessor families:

* Intel i960
* Intel 80xx
* Motorola 683xx
* MPC 860

The chip is also easily interfaced with PCI and PCMCIA/card bus controllers.

Key Management

The IDEACrypt chip provides storage for 32 keys. The keys are stored in a RAM which may be buffered for key retention when the chip is in power-down mode. The keys are protected and cannot be read out of the chip.

The keys may be designated as either master keys or session keys. Master keys are always loaded via the C Port. In a master-session key environment encrypted session keys are loaded via the A Port, decrypted under the master key, and stored in the key RAM. In an asymmetric key management environment, the unencrypted session key (e.g., after decrypting with RSA) is loaded via the C port.

Each key may be stored either for encryption or decryption, with the subkey generation being carried out automatically. The expanded key (consisting of 52 16-bit subkeys) may also be loaded directly via the C port. Key management is controlled via the key control register.

Initial Vectors / Context Saving

In CBC, CFB, and OFB modes, initial vectors are required. At the beginning of a session, initial vectors are loaded via the A Port. When a session is interrupted (e.g., for handling another session), the context is automatically stored at the same address where the session key is stored. Upon continuation of the session, the context is thus available immediately without reloading.
IV management is also controlled via the key control register.

Applications

Typical applications of the IDEACrypt are as follows:

* PC market: file encryption, encryption of data stored on disk
* Electronic commerce, financial applications
* Telecommunication networks: ATM, SONET
* Computer networks : Routers, Firewalls, WAN, LAN
* High speed cable modems
* High speed secure communication in general

Specifications

```
DC Operating Conditions
Supply voltage Core 3.3 V
Pads 3.3 / 5 V
Backup RAM 2.6 V

Temperature Range
Operating: 0 to 70 C
Storage: -25 to 85 C

Package

TQFP 144 (20x20x1.6 mm), 0.5 mm pitch

Evaluation Kit

   * PC adapter card with PCI bus
   * Evaluation software (drivers, etc.)
   * User guide

 Ascom Systec Ltd 1998
Ascom Systec reserves the right to make changes to this
specification at any time and without notice. 250298
------------------------------------------------------

Last modified: March 15, 1998, webmaster
Copyright  1997 by Ascom. All rights reserved
```

Appendix D: Zwakke sleutels

Samenvatting

Nummer de bits in een IDEA sleutel van 0 tot 127, waarbij bit 0 het meest significante bit is. Joan Daemen [15] beschrijft drie klassen van zwakke sleutels:

1. Sleutels waarbij versleuteling een lineaire factor heeft, d.w.z., waarbij een zekere lineaire combinatie van uitvoerbits op een lineaire manier van een lineaire combinatie van invoerbits afhangt. De grootste groep van deze sleutels heeft nullen op plaatsen 0-25, 29-71 en 75-110, terwijl bits 26-28, 72-74, 111-127 vrij gekozen kunnen worden, zodat er 2^{23} zwakke sleutels van dit type zijn. Dit is de enige klasse die expliciet in [15] gegeven wordt. Voor de volledige lijst, zie hieronder.
2. Sleutels waarbij invoerteksten die op weinig plaatsen verschillen leiden

tot uitvoerteksten die herkenbare overeenkomst vertonen. De grootste groep van deze sleutels heeft nullen op plaatsen 0-25, 41-71, 77-107 en 123-127, terwijl bits 26-40, 72-76, 108-122 vrij gekozen kunnen worden, zodat er 2^{35} zwakke sleutels van dit type zijn. Dit is de enige klasse die expliciet in [15] gegeven wordt. Voor de volledige lijst, zie hieronder.

3. Sleutels waarbij invoerteksten die op weinig plaatsen verschillen leiden tot tussenresultaten na zeven ronden die herkenbare overeenkomst vertonen. Aannemende dat zo'n sleutel gebruikt is kunnen dan de sleutelbits gebruikt in de laatste ronde bepaald worden, en met weinig extra moeite volgt de hele sleutel. De grootste groep van deze sleutels heeft nullen op plaatsen 0-25, 41-71, 84-98 en 123-127 terwijl bits 26-40, 72-83, 99-122 vrij gekozen worden, zodat er 2^{51} zwakke sleutels van dit type zijn. Alle zwakke sleutels van type 2 zijn ook van type 3.

Een heel eenvoudige test die voldoende is om al deze zwakke sleutels te vermijden is nagaan of ten minste een van de bits 8-23 niet nul is, dwz of de tweede en derde byte van de sleutel niet beide NUL zijn.

Lineaire factoren

IDEA gebruikt drie verschillende groupsstructuren door elkaar, maar als we alleen naar het laatste bit van een 16-bit waarde kijken, dan doen $+$ en $\oplus$ hetzelfde. De vermenigvuldiging doet nog steeds iets obscuurs, behalve dan als het een vermenigvuldiging met 1 of -1 is. Aangezien alleen vermenigvuldigingen met sleutelbits voorkomen, geeft de eis dat we met 1 of -1 vermenigvuldigen een voorwaarde op de sleutel.

Schrijf x voor het laatste bit van een variabele X. Merk op dat als $K = 1$ dan is $k = 1$ en als $K = 0$ (dit is de -1 van de multiplicatieve groep) dan is $k = 0$, terwijl $-A = \bar{A} + 2 \pmod{65537}$, als $\bar{A}$ het bitsgewijze complement van A is, zodat in beide gevallen het laatste bit van $K.A$ gelijk aan $a \oplus k \oplus 1$ is.

Druk voor elk van de 16 keuzen van $a = (a_1, a_2, a_3, a_4)$, een binaire vector ter lengte 4, de statistiek $(a, y) = a_1y_1 \oplus a_2y_2 \oplus a_3y_3 \oplus a_4y_4$ op de uitvoer van een ronde uit in de invoer bits x_1, x_2, x_3, x_4, en de deelsleutelbits $k_1, ..., k_6$, zeg als $(a, y) = (b, x) \oplus (c, k) \oplus d$, en noteer welke deelsleutels 1 of -1 moeten zijn om dat te laten werken. Dit levert de tabel

a	b	c	d	voorwaarde op
0000	0000	000000	0	
0001	0100	010001	0	K_6
0010	1101	110111	0	K_1, K_4, K_5, K_6
0011	1001	100110	0	K_1, K_4, K_5
0100	0001	000101	0	K_4, K_6
0101	0101	010100	0	K_4
0110	1100	110010	0	K_1, K_5
0111	1000	100011	0	K_1, K_5, K_6
1000	0111	011111	1	K_4, K_5, K_6
1001	0011	001110	1	K_4, K_5
1010	1010	101000	1	K_1
1011	1110	111001	1	K_1, K_6
1100	0110	011010	1	K_5
1101	0010	001011	1	K_5, K_6
1110	1011	101101	1	K_1, K_4, K_6
1111	1111	111100	1	K_1, K_4

Dat was voor één ronde. Voor acht-en-een-halve ronde moeten we deze voorwaarden combineren. (Voor de laatste halve ronde vinden we restricties op K_1 en/of K_4 als de lineaire combinatie waar we naar kijken y_1 en/of y_4 bevat. Eerst een overzicht welke bits van de sleutel in welke deelsleutels gebruikt worden.

ronde	K_1	K_2	K_3	K_4	K_5	K_6
1	0-15	16-31	32-47	48-63	64-79	80-95
2	96-111	112-127	25-40	41-56	57-72	73-88
3	89-104	105-120	121-127, 0-8	9-24	50-65	66-81
4	82-97	98-113	114-127, 0-1	2-17	18-33	34-49
5	75-90	91-106	107-122	123-127, 0-10	11-26	27-42
6	43-58	59-74	100-115	116-127,0-3	4-19	20-35
7	36-51	52-67	68-83	84-99	125-127,0-12	13-28
8	29-44	45-60	61-76	77-92	93-108	109-124
9	22-37	38-53	54-69	70-85		

We vinden nu bij elke keuze van een lineaire combinatie van de invoerbits (preciezer: de vier laatste bits van de vier groepjes van 16 bits waarin in invoerblok verdeeld wordt) een lineaire combinatie van de uitvoerbits van IDEA mits alle sleutelbits behalve de hieronder aangegeven bits nul zijn.

invoer	uitvoer	aantal zwakke sleutels	vrije sleutelbits
0001	1101	2	108
0010	1110	2^8	56-63
0011	0011	2^5	111-115
0100	0100	2^{13}	65, 97-108
0101	1001	2^{12}	37-40, 108-115
0110	1010	2^{14}	111-124
0111	0111	1	
1000	1000	2	108
1001	0101	2^{17}	99-115
1010	0110	2^{23}	26-28, 72-74, 111-127
1011	1011	2	65
1100	1100	2^{20}	33-35, 108-124
1101	0001	2^{11}	63, 99-108
1110	0010	1	
1111	1111	2^{12}	63-69, 111-115

Al deze zwakke sleutels beginnen met drie NUL bytes (het eerste bit dat ergens vrij voorkomt is bit 26), dus bij een sleutelkeuze die niet met drie NUL bytes begint is deze zwakte vermeden.

Correlaties

Hierboven werd het gedrag van het laatste bit van de vier 16-bit groepjes onderzocht. Voor het voorste bit kan iets dergelijks gedaan worden. Hier is niet direct iets over de uitvoer te zeggen als alleen de voorste bits bekend zijn—tenslotte kunnen die door de gewone optelling bedorven worden—maar als twee invoerteksten gelijk zijn behalve misschien wat betreft de vier voorste bits van de vier 16-bit groepjes, dan blijft deze eigenschap behouden onder $+$ en $\oplus$, en als alleen met 1 of -1 vermenigvuldigd wordt kan een pad ontstaan via hetwelk een bekend verschil van invoer tekst resulteert in een bekend verschil van uitvoer tekst. De tabel hieronder geeft bij elk van de 16 mogelijke combinaties van de voorste bits van de vier 16-bit groepjes in $X \oplus X'$, waarbij X en X' overigens identiek zijn, het corresponderende verschil in de uitvoer $Y \oplus Y'$ van een ronde, en welke deelsleutels 1 of -1 moeten zijn om deze conclusie te kunnen trekken.

$X \oplus X'$	$Y \oplus Y'$	voorwaarde op
0000	0000	
0001	1110	K_4, K_6
0010	1000	K_5, K_6
0011	0110	K_4, K_5
0100	1011	K_6
0101	0101	K_4
0110	0011	K_5
0111	1101	K_4, K_5, K_6
1000	0010	K_1, K_5, K_6
1001	1100	K_1, K_4, K_5
1010	1010	K_1
1011	0100	K_1, K_4, K_6
1100	1001	K_1, K_5
1101	0111	K_1, K_4, K_5, K_6
1110	0001	K_1, K_6
1111	1111	K_1, K_4

Propagatie door alle ronden levert een verband tussen invoer-XOR en uitvoer-XOR op voorwaarde dat alle sleutelbits behalve de aangegeven vrije bits nul zijn.

invoer	uitvoer	aantal zwakke sleutels	vrije sleutelbits
0001	0001	2^3	63-65
0010	0010	1	
0011	0011	2^{24}	37-47, 79-83, 108-115
0100	0111	2^6	0-1, 124-127
0101	0110	2^{35}	26-40, 72-76, 108-122
0110	0101	2^{28}	37-40, 92-115
0111	0100	1	
1000	1011	2^5	104-108
1001	1010	2^{14}	111-124
1010	1001	2^5	111-115
1011	1000	2	108
1100	1100	2^{12}	111-122
1101	1101	2^2	108, 124
1110	1110	2^{15}	56-65, 104-108
1111	1111	2^{12}	63-69, 111-115

Al deze zwakke sleutels beginnen met drie NUL bytes (het eerste bit dat ergens vrij voorkomt is bit 26), behalve bij invoer 0100, waar de tweede en derde byte NUL zijn, dus bij een sleutelkeuze waarbij niet zowel de tweede als de derde byte NUL zijn is deze zwakte vermeden. Deze zelfde eis is afdoende om te vermijden dat een correlatie als hier beschreven geldt voor het cumulatief effect van de eerste zeven ronden:

invoer	aantal zwakke sleutels	vrije sleutelbits
0001	2^{15}	63-65, 111-122
0010	2^{5}	111-115
0011	2^{37}	33-47, 79-83, 99-115
0100	2^{19}	0-1, 111-127
0101	2^{51}	26-40, 72-83, 99-122
0110	2^{45}	33-40, 79-115
0111	2^{5}	111-115
1000	2^{19}	104-122
1001	2^{17}	33-35, 111-124
1010	2^{18}	26-35, 72-74, 111-115
1011	2^{12}	104-115
1100	2^{15}	33-35, 111-122
1101	2^{21}	104-124
1110	2^{22}	56-65, 104-115
1111	2^{29}	24-35, 63-74, 111-115

Appendix E: Andere aanvallen op IDEA

Nu IDEA met zijn 8,5 ronden bij normaal gebruik onbreekbaar lijkt, heeft menigeen gepoogd een gekortwiekte versie van IDEA te kraken. Meier [33] beschrijft een aanval die werkt bij gekortwiekt IDEA met 2 ronden, maar geen voordelen meer biedt bij 3 of meer ronden. Daemen [16] beschrijft hoe IDEA met 2,5 ronden aangepakt kan worden. Borst, Knudsen en Rijmen [13] beschrijven twee verschillende aanvallen die soms werken bij 3 of 3,5 ronden. Lai [28], de ontwerper van IDEA, was al tot de conclusie gekomen dat IDEA met 4 of meer ronden bestand is tegen differentiële cryptanalyse (de techniek die in [13] gebruikt wordt), en de genoemde publicaties lijken deze conclusie te bevestigen.

Appendix F: Review van [25]

In Zentralblatt für Mathematik 777.94013 staat het volgende review van [30]:

Xuejia Lai & James L. Massey, Markov ciphers and differential cryptanalysis, Advances in Cryptology, Proc. Workshop, EUROCRYPT '91, Brighton/UK 1991, Lect. Notes Comput. Sci. 547, 17-38 (1991).

The paper considers the security of iterated block ciphers against the differential cryptanalysis developed by Biham and Shamir. Especially, it is investigated if the so-called Proposed Encryption Standard (PES), introduced by the authors in 1991, is resistant to differential cryptanalysis.

The mentioned cryptanalysis is a chosen-plaintext attack on secret-key ciphers that are based on iterating a cryptographically weak function several times. The

iterations are called rounds. Differential cryptanalysis analyzes the effect of the difference of a pair of plaintexts on the difference of succeeding round outputs in a iterated cipher.

The authors use the concept of Markov ciphers to describe the probability of success of differential cryptanalysis on an r-round cipher depending on the existence of $(r-1)$-round differentials with high probability. (An i-round differential is a couple (α, β) such that a pair of plaintexts with difference α can result in a pair of i-th round outputs that have difference β.)

It is proved that PES including also its mini-versions is immune to differential cryptanalysis after sufficiently many (≥ 7) rounds. As a result of these investigations a minor modification of PES is proposed. This modified cipher called Improved PES (IPES) is shown to be highly resistant against differential cryptanalysis.

[G.Eigenthaler (Wien)]

Citations: Zbl.756.00008 Keywords: security of iterated block ciphers; differential cryptanalysis; Proposed Encryption Standard; chosen-plaintext attack; secret-key ciphers; Markov ciphers

Classification: *94A60 Cryptography

Appendix G: Nieuwe export regels?

De New York Times schreef op 9 juli 1998:

ADMINISTRATION RELENTS (A LITTLE) ON ENCRYPTION EXPORTS
Bowing to pressure from the software and banking industries, the Clinton administration is easing controls over the export of strong encryption programs, but only for banks and financial institutions in nations that are deemed to have acceptable money-laundering laws. The announcement was made Tuesday by Commerce Secretary William Daley, who noted, "This action gives our nation's financial institutions the flexibility they need to remain globally competitive. Importantly, it balances those needs with law enforcement, national security and foreign policy concerns. Through steps like this we can continue to encourage the development of an electronic commerce system users can trust." Under the new rules, which will take effect later this summer, companies will not need to submit a plan for creating law-enforcement keys before exporting encryption products to financial institutions headquartered in any of the 45 qualifying nations. Once they receive approval, the financial institutions will be able to share the technology with branches worldwide, except those located in "terrorist" states.

References

[1] Mark Ames, *Saving dollars makes sense of crypto export controls*, Cryptography: Policy and Algorithms, Proc. Brisbane 1996, E. Dawson & J. Golić (eds.), Lecture Notes in Comp. Sci. 1029, Springer 1996, pp. 90-97.

[2] `http://www.ascom.ch`

[3] `http://www.ascom.ch/products.html`

[4] `http://www.ascom.ch/systec/infosec.html`

[5] `http://www.ascom.ch/systec/IDEA.html`

[6] `http://www.ascom.ch/systec/security/Policy/Exhibit/ideapatent.html`

[7] `http://www.ascom.ch/systec/security/Policy/idea_licmain.html`

[8] `http://www.ascom.ch/systec/security/Policy/idea_lictypes.html`

[9] `http://www.ascom.ch/systec/security/Policy/Exhibit/idea_exhfees.html`

[10] `http://www.ascom.ch/Web/systec/security/ideachip.html`

[11] M. Bellare et al., *A concrete security treatment of symmetric encryption*, Proc. of the 38-th Symposium on the Foundations of Computer Science, IEEE, 1997.

[12] E. Biham & A. Shamir, *Differential cryptanalysis of DES-like cryptosystems*, J. of Cryptology 4 (1991) 3-72.

[13] Johan Borst, Lars R. Knudsen & Vincent Rijmen, *Two attacks on reduced IDEA*, Eurocrypt '97, pp. 1-13.

[14] A. Curiger, H. Bonnenberg, R. Zimmermann, N. Felber, H. Kaeslin & W. Fichtner, *VINCI: VLSI Implementation of the New Block Cipher IDEA*, Proc. IEEE CICC '93, San Diego, CA, May 1993, pp. 15.5.1-15.5.4.

[15] Joan Daemen, René Govaerts & Joos Vandewalle, *Weak keys for IDEA*, Advances in Cryptology—Crypto '93 Proceedings, D.R. Stinson (ed.), Lecture Notes in Comp. Sci. 773, Springer 1994, pp. 224-230.

[16] Joan Daemen, René Govaerts & Joos Vandewalle, *Cryptanalysis of 2.5 rounds of IDEA*, ESAT-COSIC Report 94-1, Dept. of Electrical Engin., Univ. Leuven, March 1994.

[17] Electronic Frontier Foundation (and John Gilmore, Cryptography Research, Paul Kocher, Advanced Wireless Technologies), *Cracking DES: Secrets of Encryption Research, Wiretap Politics & Chip Design*, O'Reilly, May 1998. ISBN: 1-56592-520-3

[18] `http://www.eff.org/descracker.html`

[19] `http://www.replay.com/mirror/cracking_des/chap-11.html`

[20] W. Diffie & M.E. Hellman, *Exhaustive cryptanalysis of the NBS Data Encryption Standard*, Computer **10** (1977) 74-84.

[21] D.W. Davies & G.I.P. Parkin, *The average size of the key stream in Output Feedback Encipherment*, Cryptography, Proc. Burg Feuerstein, !982, Springer 1983, pp. 263-279.

[22] Lov K. Grover, *A fast quantummechanical algorithm for database search*, Proc. 28th ACM Symp. on the Theory of Computing, 1996, pp. 212-219.

[23] M.E. Hellman, R. Merkle, R. Schroeppel, L. Washington, W. Diffie, S. Pohlig & P. Schweitzer, *Results of an initial attempt to Cryptanalyze the NBS Data Encryption Standard*, Technical Report SEL 76-042, Information Systems Lab, Dept. of Electrical Engineering, Stanford University, 1976.

[24] R.R. Jueneman, *Analysis of Certain Aspects of Output-Feedback Mode*, Advances in Cryptology—Crypto 82 Proceedings, Plenum, 1983, pp. 99-127.

[25] John Kelsey, Bruce Schneier & David Wagner, *Key-schedule Cryptanalysis of IDEA, G-DES, SAFER, and Triple-DES*, Advances in Cryptology—Crypto '96, Proceedings Santa Barnara Aug 1996, N. Koblitz (ed.), Lecture Notes in Comp. Sci. 1109, Springer 1996, pp. 237-251.

[26] Paul C. Kocher, *Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems*, Advances in Cryptology—Crypto '96, Proceedings Santa Barnara Aug 1996, N. Koblitz (ed.), Lecture Notes in Comput. Sci. 1109, Springer 1996, pp. 104-113.

[27] Bert-Jaap Koops, `http://cwis.kub.nl/~frw/people/koops/lawsurvy.htm`

[28] X. Lai, *On the design and security of block ciphers*, Ph.D. Thesis, ETH Zürich, 1992. Uitgegeven in ETH Series in information processing **1**, Hartung-Gorre, Konstanz, 1992.

[29] X. Lai & J. Massey, *A Proposal for a New Block Encryption Standard*, Advances in Cryptology, Proc. Workshop, EUROCRYPT '90, Aarhus/Denm. 1990, Lect. Notes Comput. Sci. 473, Springer-Verlag, 1991, pp. 389-404. Zbl 764.94017

[30] Xuejia Lai, James L. Massey & S. Murphy, *Markov ciphers and differential cryptanalysis*, Advances in Cryptology, Proc. Workshop, EUROCRYPT '91, Brighton/UK 1991, Lect. Notes Comput. Sci. 547, 17-38 (1991). Zbl 777.94013

[31] M. Matsui, *Linear Cryptanalysis method for DES cipher*, Advances in Cryptology—Eurocrypt '93 Proceedings, Springer 1994, pp. 386-397.

[32] Mitsuru Matsui, *The first experimental cryptanalysis of the Data Encryption Standard*, Advances in Cryptology—Crypto '94, Proceedings Santa Barbara 1994, Yvo G. Desmedt (ed.), Lecture Notes in Comp. Sci. 839, Springer 1994, pp. 1-11.

[33] W. Meier, *On the security of the IDEA block cipher*, Advances in Cryptology—Eurocrypt '93 Proceedings, T. Helleseth (ed.), Lect. Notes in Comput. Sci. 765, Springer, 1993, pp. 371-385.

[34] R. Morris, N.J.A. Sloane & A.D. Wyner, *Assessment of the NBS Proposed Data Encryption Standard*, Cryptologia **1** (1977) 281-291.

[35] National Bureau of Standards, NBS FIPS PUB 46, *Data Encryption Standard*, Jan 1977.

[36] Bruce Schneier, *Applied Cryptography*, Wiley, 1996.

[37] A. Shamir, *On the security of DES*, Advances in Cryptology—Crypto '85 Proceedings, Springer, 1986, pp. 280-281.

[38] P. Shor, *Algorithms for Quantum Computation: Discrete Log and Factoring*, Proc. 35th IEEE Symp. on Foundations of Computer Science, 1994.

[39] Daniel R. Simon, *On the power of Quantum Computation*, preprint.

[40] M.J. Wiener, *Efficient DES key search*, TR-244, School of Computer Science, Carleton University, May 1994.

[41] P.R. Zimmermann, *The Official PGP User's Guide*, MIT, Boston, 1995.

Management Summary

This report contains a description of IDEA (International Data Encryption Standard), together with an analysis of the cryptographic strength thereof.

IDEA is an encryption system proposed by X. Lai and J. Massey in 1991 as a safer alternative for DES (Digital Encryption Standard), the well-known US government standard for sensitive, but not classified information. Partly because of this origin, we shall use DES or its variant Triple DES as a point of reference when discussing IDEA in this summary. Triple DES, also referred to as EDE-DES, consists of applying three DES encryptions with three different keys in a special way.

IDEA, DES and Triple DES have the following similarities

- All three are a *symmetric* encryption system: the keys required for encryption and decryption coincide.
- All three are *block* encryption systems with a blocklength of 64 bits (8 byte): a message is divided in blocks of 64 bits and is then encrypted block by block. This means, amongst other things, that IDEA can be used in the same "modes of operation" (ECB, CBC, CFB, OFB) in which DES can be used. With IDEA one can also create Message Authentication Codes (MAC's) as one can with DES.

IDEA, DES and Triple DES have the following differences:

- The most important difference between IDEA and DES is the used key-length: IDEA uses cryptographic keys of 128 bits and DES uses cryptographic keys of only 56 bits. Triple DES, however, has an (effective) keylength of 112 bits (and not of 3x56=168 bits as one might expect).
- In software, IDEA is about twice as fast as DES, which is three times as fast as Triple DES.
- Unlike the use of DES, the use of IDEA is not free: the costs for commercial use are between $2 and $15 for each user. Non-commercial use of IDEA is free, this is one of the reasons that IDEA is used in PGP (Pretty Good Privacy), a popular tool in the public domain, for encrypting electronic mail on the Internet.
- IDEA is specially developed for implementations in software on personal computers; it uses algebraic operations, such as multiplications, that are standard available on these platforms.

In the report the following attacks on IDEA are discussed:

- "Exhaustive Key Search"
  Here, on basis of one or more encrypted messages, the attacker simply tries out all possible keys: 2^{128} with IDEA, 2^{112} with Triple DES and 2^{56} with DES. With current technology, this attack is not practically feasible with both IDEA and Triple DES. However, with DES this attack indeed is practically feasible: recently, in commission of the Electronic Frontier Foundation a "DES-cracker" has been build of $250.000 that can decrypt a DES encrypted message in less than three days. Moreover, in principle anyone can construct a DES-cracker as the specifications of it have been publicized in a book.
- "Birthday Attacks"
  If one uses IDEA (or (Triple) DES) to encrypt large quantities of information with the same key, than on a certain moment one "leaks" secret (plaintext) information. This phenomenon occurs with large probability if one encrypts $2^{32} \cdot 64 = 2^{38}$ bits (=32 Gigabyte) of information with the same key. Given the current communication and storage capacities, these quantities are not unthinkable large. The quantity of information enciphered with IDEA using the same key, must be well under this limit. With respect to this limit: if the blocklength of IDEA was 128 bits (instead of 64), then this limit would become $2^{64} \cdot 128 = 2^{71}$ bits, which is currently an unrealistic quantity of information to encounter in practice.
- Differential Cryptanalysis
  Here it is assumed that the attacker is capable to encrypt any message ("chosen plaintext") using an IDEA (or (Triple) DES) "box" with a secret key unknown by the attacker. The attacker then tries to find the secret key by slightly changing chosen messages and then analyzing the resulting differences in the enciphered messages. Both IDEA and (Triple) DES are resistant to these types of attacks. In fact, resistance to differential cryptanalysis was one of the design criteria of IDEA; the publications so far indicate that this goal is achieved.
- "Timing Attacks"

Here the same assumption holds as in the previous attack, but the attacker does not analyze the differences in the enciphered messages but analyses the required time for encryption under different messages. In theory, IDEA and (Triple) DES are not completely resistant to these type of attacks, but these type of attacks do not seem to be practically feasible.

- Weak Keys
  Also in this type of attack the assumption holds as in the previous two attacks. Some type of keys ("weak keys") give a specific patron of enciphered messages with certain (chosen) messages. In this fashion, the use of the weak keys can be determined, which is undesirable. As the probability of choosing a weak key at random is very small (10^{-20}) the risk of such keys is negligible. Moreover, as weak keys are easily detectable one can completely eliminate the risk of choosing a weak key by testing keys on weaknesses in the key generation before using them.

Summarizing, IDEA is resistant to all known attacks, but by its relatively short blocklength (64 bits), IDEA is not really suitable for encryption large quantities of information (multiple gigabytes) under the same key. It is not a coincidence that an important requirement in the AES (Advanced Encryption Standard, the official US government successor of DES) is that the blocklength must be 128 bits. In *that* sense, IDEA can not be considered as the successor of DES as meant by Lai and Massey: at most it can be seen as an intermediate solution. However, the same can be said about Triple DES. Although IDEA has a few advantages over Triple DES (speed, slightly more secure), it depends of the application if this is worth the extra cost.