JavaCard Security

Rickard Bondesson David Skoghem
Email: {ricbo974,davsk996}@student.liu.se
Supervisor: Anna Vapen, { x07annva@ida.liu.se }
Project Report for Information Security Course
Linköpings universitet, Sweden

Sammanfattning

JavaCard är en teknologi som gör det möjligt att köra Javaprogram på ett smartcard. JavaCard bygger på programmeringsspråket Java, och har därmed ärvt flera av dess säkerhetsfunktioner. JavaCard erbjuder bland annat säker överföring av applikationer till ett smartcard, atomära funktionsanrop och de vanligaste krypteringsalgoritmerna. Genom att tillåta ”native methods”, kan Javas säkerhetsfunktioner kringgås. JavaCard har inget inbyggt stöd för ”garbage collection”, vilket ställer höga krav på att programmen är felfria. De flesta hårdvaruattacker mot smartcards verkar fungera även mot JavaCard, dock har ofta nya kort inbyggda säkerhetsfunktioner som kan hindra flera av dessa attacker. Förutom att lägga till ”garbage collection” så vill vi att JavaCard implementerar CFB (Cipher Feedback) som krypteringsläge.

1. Inledning

Vårt projekt har gått ut på att undersöka JavaCard och dess säkerhetsfunktioner.

1.1 Frågeställningar

Vilka säkerhetsfunktioner erbjuder JavaCard?
- Hur fungerar dessa säkerhetsfunktioner?
- Hur är säkerhetsfunktionerna tillämpbara?
- Finns det enkla möjligheter för krypterade överföringar?

Hur kan JavaCard förbättras?
- Var går det att tillämpa JavaCard?
- Är det några fördelar med JavaCard jämfört med andra smartcards?

Ger JavaCard något extra skydd mot yttre hot, så som analys av strömförbrukning eller analys av kortets minnesceller?

1.2 Avgränsningar

Vi kommer att fokusera på säkerhetsfunktionerna hos JavaCard, vilket gör att vi bland annat utesluter kostnader och utvecklingsproblem. Vi kommer delvis att gå i genom uppbyggnaden av ett smartcard, detta för att ge mer förståelse för vår studie. I och med att vi inte har tillgång till programmeringsutrustning och smartcards, kommer vi inte att göra några praktiska tester.

1.3 Förväntade resultat

Vi förväntar oss att JavaCard är en förhållandevis säker plattform för hantering av applikationer på ett smartcard. Detta på grund av Javaspråkets uppbyggnad, exempelvis hanteringen av buffer-over-flow. Vi förväntar oss också att det kommer finnas färdiga säkerhetslösningar som enkelt går att implementera i applikationer, till exempel färdiga säkerhetsalgoritmer.

1.4 Metod

Den information som kommer att inhämtas kring funktionella egenskaper blir primärt från utvecklaren av JavaCard, Sun Microsystems. För att få fler synvinklar inhämtas information även från andra källor, så som artiklar och rapporter. Vi ska också hämta fakta från tillgängliga böcker, JavaCard Technology for Smart Cards och Smartcard Handbook.

2. Bakgrund

JavaCard är en teknologi som gör det möjligt att köra Javaprogram på ett smartcard.

2.1 Smartcard

Genom att kombinera plastkort med elektroniska kretsar fås något som kallas för smartcard. Det vanligaste är att bädda in dem i plastkort av samma storlek som ett kreditkort. Det finns två typer av kort, minneskort och processorkort. Minneskorten kan endast lagra information medan processorkorten kan även behandla information. Vi definierar smartcards som processorkort för att enklare kunna göra en jämförelse med JavaCard.

Smartcards kom till världen i slutet på 1960-talet genom två forskare från Tyskland, Jürgen Dethloff och Helmut Grötrupp. Den första stora användningen utav smartcards kom dock inte förrän i början på 1980-talet i Frankrike som förbetalda telefonkort. [1]

Den enklaste typen av smartcard har ett minne och en processor med förhållandevis lite beräkningskraft, medan de mer avancerade kan ha dynamiskt minne och filhantering. Det finns tre typer av minnen, persistent non-mutable memory, persistent mutable memory och non-persistent mutable memory. Vanliga varianter på dessa minnestyper är: ROM, EEPROM och RAM. [3, 4]

Smartcards kan innehålla operativsystem. Dessa operativsystem kan delas upp i två olika kategorier, bestämd filstruktur (fixed file structure) och dynamiskt applikationssystem (dynamic application system). Vilken typ av operativsystem som används beror på vilka applikationer de är designade för och vad de har för krypterings-möjligheter.

Smartcards kan delas in i två typer, med eller utan kontaktyta. Den första typen har kontaktytor på ovansidan av kortet. Genom att föra in kortet i en läsare kan en kommunikationskanal bildas mellan kortet och mottagaren. Hur dessa ser ut och fungerar specificeras av ISO 7816 och ISO 7810. [2]

Den andra typen kommunicerar trådlöst genom induktion med hjälp av RFID teknologin. Att det krävs induktion är för att smartcards saknar egen strömkälla för att driva de interna kretsarna. Denna typ specificeras med hjälp av ISO 14443. [2]

Gemensamt för dessa standarder är att de ger en specifikation på storlek, form och kommunikations-protokoll, men det som saknas är hur ett smartcard fungerar internt. Det blir alltså upp till varje tillverkare att avgöra vilken hårdvara som finns i ett smartcard. Denna skillnad i hårdvara leder till att olika typer av assemblerspråk måste användas. Det blir då svårt att utveckla mjukvara som fungerar för flera olika typer av smartcards. [1]

2.2 JavaCard

JavaCard utvecklades som ett svar på den problematik som beskrivits ovan. JavaCard kom till år 1996 när ett antal ingenjörer ville göra smartcards mer tillgängliga för allmänheten, samtidigt som säkerheten skulle hållas intakt på korten. Deras lösning på problemet var att använda Java som högnivåspråk i utvecklingen av programvara för smartcards. Resultatet blev JavaCard 1.0, men de hade enbart specificerat API:et. 1999 släppte Sun Microsystem JavaCard 2.1 som har tre typer av specifikationer: API, Runtime Environment och Virtual Machine. Genom dessa specifikationer kan JavaCard emulera en virtuell miljö trots val av olika hårdvara. En utvecklare behöver därmed bara skriva ett program som kan användas på flera plattformar istället för att behöva skriva ett program för varje plattform. [1]

Språket som används vid programmeringen är i grunden Java, men innehåller endast ett urval av de metoder som specificeras i Javas standard-API. [1]

2.3 IBM Jcop

Jcop är ett smartcard utvecklat av IBM som bygger på specifikationerna för Java och implementerar stöd för JavaCard-API:et. Jcop kan då användas som ett vanligt JavaCard, men har utökade funktioner. Ett Jcop-kort har bland annat möjligheten att använda ”garbage collection”, köra applets i en sandlådemiljö och är designat för att vara enklare att programmera än JavaCard. [13]

2.4 Tillämpningar

JavaCard används i samma tillämpningar som för vanliga smartcards, bland annat som SIM-kort (Subscriber Identity Module) i mobiltelefoner, som betallösningar i kontokort och som passerkort.

3. Analys

JavaCard erbjuder ett antal säkerhetsfunktioner/ egenskaper och kryptologiska algoritmer.

3.1 Säkerhetsfunktioner

JavaCard använder ett utval av metoder från Javas API. Detta ger bland annat att den får nedärvda säkerhetsegenskaper från Javaplattformen, vilket innebär bland annat typsäkra variabler, att den inte tillåter pekare till platser i minnet, verifiering av bytekoden under körning och inkapsling av data i objekt. Vidare erbjuder JavaCard funktioner för säker överföring av applikationer till ett JavaCard. De ser till att paketet (CAP-fil), som innehåller applikationen, inte utför några förbjudna åtgärder enligt en specificerad kontrollista. Paketet kan även vara krypterat för att skydda viktig information från att läcka till omvärlden när det inte ligger på kortet. Själva paketet kan även vara signerat för att påvisa att det kommer från rätt avsändare, för att skydda mot modifierad kod. När väl applikationen körs görs ytterligare kontroller av bytekoden för applikationen, vilket ger en säkrare körning av programmet. [1]

JavaCard använder atomära funktioner vid kommunikationen med omvärlden. Detta innebär att funktioner antingen utförs fullt ut eller inte alls. En funktion kan alltså inte komma halvvägs med alla åtgärder och sedan avbryta transaktionen med behållna ändringar. Detta innebär bland annat att om kortet tas ur läsaren under exekveringen av en funktion, avbryts transaktionen och kortet återgår till sitt tidigare tillstånd.

Samtidigt meddelas applikationen i mottagaren att transaktionen avbröts. [6]

I API:et finns det ytterligare metoder för säkerhetshantering, bland annat generering av kryptonycklar, nyckelöverenskommelse och hantering av PIN. [6]

3.2 Applets

En JavaCard-applikation går under benämningen applet. Den laddas ifrån det paket som ligger lagrat i kortet och registreras i JavaCard Runtime Environment (JCRE). JCRE kan bara hantera en tråd i taget, vilket innebär att bara en applet i taget kan köra. Alla applets på kortet ligger och väntar på anrop utifrån. När väl ett anrop kommer till den enskilda appleten, låter JCRE den göra sina beräkningar och lämna tillbaka ett svar. När väl detta är gjort får appleten återgå till vänteläge. Anropen och svaren sker i form av APDU:er (Application Protocol Data Unit). Dessa går ifrån användaren via kortet till JCRE, som skickar det vidare till rätt applet. [1]

JavaCard använder en så kallad brandvägg, men den är inte som en klassisk brandvägg som filtrerar trafik utan den är till för att hålla applets skilda från varandra. Detta löses bland annat genom att ge dem tillgång till olika delar av minnet och genom olika åtkomstregler. En implikation av detta är att en felande applet inte kan påverka andra applets eller själva kortet. [6]

På applikationsnivå kan inte JavaCard ge extra skydd, eftersom att det är upp till utvecklarna att skriva bra program som inte gör några misstag gällande säkerhetskriteriet CIA. Confidentiality, att inte känslig information så som nycklar och kontonummer läcker ut till obehöriga. Integrity, att modifiering av applikationsdata sker på rätt sätt. Authentication, att det är rätt avsändare för de APDU:er som kommer in. Det kan också vara viktigt att utnyttja digtala signaturer för att få non-repudiation, alltså att en part inte kan neka att de utfört en handling eller skickat specifik information. [1]

Ett sätt att skydda information som skickas mellan mottagare och kortet är att kryptera överföringen. JavaCard erbjuder vissa krypteringsalgoritmer som då kan användas till att skydda den data som skickas med APDU:erna och tillåter även att avsändaren signerar datan. [1]

3.3 Algoritmer

JavaCard 2.2.2 stödjer ett antal kryptologiska algoritmer, vilka finns tillgängliga genom dess API. Dessa är AES (CBC, ECB), DES (CBC, ECB), KOREAN SEED (CBC, ECB), Elliptic Curves och RSA. [5]

De tre första är symmetriska krypteringsalgoritmer, vilket innebär att man använder samma nyckel för både kryptering och dekryptering. Det som står inom parentes är de lägen, som är implementerade i JavaCard, vilka dessa algoritmer kan arbete med. ECB är Electronic CodeBook, vilket innebär att varje block data krypteras oberoende av de andra blocken av data. Detta beteende har gett upphov till namnet på detta läge, eftersom att en uppsättning av alla möjliga varianter av datablock ger en så kallad kodbok, vilken då innehåller alla lösningar. Det andra läget CBC är Cipher Block Chaining, vilket innebär att krypteringen inte enbart beror av det aktuella blocket utan också den tidigare chiffertexten.

De två sista algoritmerna (Elliptic Curves och RSA) är asymmetriska krypteringsalgoritmer. För att kryptera information används en publik nyckel, vilken är känd för alla, medan för att dekryptera används en privat nyckel, som bara är känd för mottagaren. Detta innebär att det är bara mottagaren som kan få tag på den information som krypterades.

Vidare stödjer JavaCard ett antal signeringsalgoritmer som kan användas för att signera innehållet i ett meddelande. Dessa är de algoritmer som stödjs i JavaCard 2.2.2: AES MAC, DES MAC4, DES MAC8, DSA SHA, ECDSA SHA, HMAC MD5, HMAC RIPEMD160, HMAC SHA (256 - 512), HMAC SHA1, KOREAN SEED MAC, RSA MD5, RSA RIPEMD160 och RSA SHA. [5]

För att använda digital signering används en hashfunktion, som beräknar en checksumma på meddelandet. Denna checksumma krypteras sedan med hjälp av en krypteringsalgoritm. Detta hjälper till att identifiera att det är rätt avsändare, samt om innehållet i meddelandet har ändrats.

3.4 Säkerhetsrisker

JavaCard har vissa medfödda säkerhetsproblem som vi kommer att ta upp, och som eventuellt kan komma att utvecklas till stora nackdelar för plattformen.

Först och främst har JavaCard inget stöd för garbage collection. Detta leder till att även små fel i programkod kan leda till stora problem. Dessa fel kan till exempel göra slut på minne vilket gör kortet obrukbart. Om pekaren till en minnesplats ligger kvar även efter att informationen på den platsen byts ut så blir det minnesfel och utan ett system som frigör allokerade minnesplatser blir kortet mottagligt för DOS-attacker. [7]

JavaCards VM (Virtual Machine) översätter bytekoden till körbar kod och kontrollerar att applets inte försöker utnyttja andra applets minnesområde. I och med att JavaCards VM arbetar ovanför operativsystemet på kortet, är dess säkerhet beroende av säkerheten för operativsystemet. Kortet behöver kunna hemlighålla kryptografiska primtal från alla typer av attacker och även ha säker minneshantering som klarar av strömstörningar eller minnesfel. [8]

Eftersom JavaCard är en öppen multiapplikations-plattform, vilket innebär att det går att lägga in egna program på kortet, så finns det en risk för attacker mellan programmen på kortet. För att minska dessa risker har JavaCard en så kallad inbyggd applet firewall. Den gör så att objekt kan utnyttja endast de metoder och data inom det virtuella områden som begränsas av brandväggen. Det gäller alltså att det inte finns några luckor i implementationen av brandväggen. [7]

Objekthanteringen på JavaCard gör det möjligt för program att utbyta information med varandra. Den fungerar så att om en applet1 som ligger i ett virtuellt utrymme A och försöker få åtkomst till en applet2 i virtuellt utrymme B så behöver applet1 göra en förfrågan till JCRE. JCRE frågar då applet2 om det är ok för applet1 att använda applet2. De applets som ligger i samma virtuella utrymme behöver dock inte göra dessa förfrågningar till JCRE utan kan interagera direkt med varandra. Det säkerhetsproblem som uppstår är att applets som har behörighet till ett objekt har också möjligheten att föra över den behörigheten till en annan applet. [7] [9]

JavaCard tillåter användandet av ”native methods” vilket innebär att programmet kan göra anrop till funktioner som finns tillgängliga i hårdvaran, men som inte stöds av Javas API. Detta innebär att om JCRE tillåter en native method att köra så försvinner det skydd som erbjuds av Javamiljön, eftersom att koden körs direkt mot hårdvaran och inte via JCRE. [7] [8]

3.5 Attacker mot JavaCard

Det finns generellt två typer av attacker mot JavaCard och andra typer av smartcards, hårdvaruattacker och mjukvaruattacker.

3.5.1 Hårdvaruattacker

Differential power analysis (DPA) är en metod som fungerar på de flesta smartcards. Det är en ganska avancerad attack då den utnyttjar komplicerade algoritmer för att mäta hur strömmen förändras när processorn jobbar. Olika operationer på kortet drar olika mycket ström och gör att det ofta går att få fram hemlig data från kortet. För att kunna göra en sådan attack krävs en smartcardläsare och en dator. Den komplicerade biten är att tolka resultaten av mätdatan, men med hjälp av algoritmer och modeller kan en bra bild skapas av vad som händer på kortet. [10]

Tester har visat att DPA kan läsa ut de flesta instruktioner som används av en applet under körning, men det finns fall då det blir avsevärt svårare. JavaCard-instruktioner som utför liknande operationer är svåra att skilja på, flera instruktioner är inte tidsbestämda vilket gör det svårt att veta vad det är för instruktion. Vissa nya kort har lagt in skiftbrus vilket gör att en DPA blir svårare och tar mycket längre tid.

Strömspiksattacker, ”glitch”-attacker, optiska attacker och elektromagnetiska störningar kan introducera eller hitta fel och kan utnyttjas på de flesta smartcards. En strömspikattack utförs genom att variera strömmen till kortet. Det görs genom att ansluta en strömkontakt direkt på kortet utan någon modifikation, vilket gör det till en non-invasive-attack. En strömspik går att variera på ett flertal olika sätt, exempelvis går det att ändra form, höjd, varaktighet, snabbhet till maxhöjd och snabbhet tillbaka till noll. Detta görs för att hitta fel på kortet som sedan kan utnyttjas.

En ”glitch”-attack försöker ändra processor-hastigheten och liknar på det viset strömspikmetoden. Om det går att öka upp klockcykeln så kommer vissa instruktioner att bli påverkade medan andra inte. Det kan då ligga kvar gammal data som inte borde göra det, som då eventuellt går att läsa av.

Det går även att göra en optisk attack med till exempel en laserstråle. Genom att utnyttja speciella våglängder kan det gå att ändra tillståndet på minnescellerna i ett minne. Denna attack är semi-invasive då det krävs att man tar bort allt skyddslager som ligger ovanför den minnescell som ska ändras.

En sista attack är elektromagnetisk störning som utförs genom att ha en stark elektromagnetisk kraft nära minnet på ett smartcard så att joner som kontrollerar tillståndet av minnet kan flyttas runt. Denna attack är non-invasive då den endast kräver att den som utför attacken är nära processorn. [11]

3.5.2 Mjukvaruattacker

De risker som nämnts tidigare i rapporten gällande JavaCard är något som kan användas i mjukvaruattacker. Ett exempel är att JavaCard tillåter flera applets på ett och samma kort, vilket medför flera risker. En risk är att JCRE måste vara felfritt implementerad då alla applets förlitar sig på att JCRE fungerar korrekt. JCRE har nämligen hela kontrollen över kortet och full åtkomst till resurserna på kortet. Finns det då några fel i JCRE blir det mycket svårt att kunna bibehålla säkerheten på ett JavaCard. [8]

Vidare finns det mjukvaruattacker som är mer generella. Fuzzing är sätt att attackera ett system, inte bara JavaCard utan alla system som har någon form av input. Attacken går ut på att man testar massor av olika kombinationer av indata till ett system. Målet är att framkalla ett beteende som inte var tänkt av utvecklaren. Genom avancerade datorprogram kan man på ett smart sätt välja de indata som ska testas och behöver därmed inte testa alla möjliga kombinationer. Exempelvis genom att testprogrammet känner till det protokoll (APDU) som används vid överföring av instruktioner till kortet.

Om programmet på kortet inte är utformat på ett väl genomtänkt sätt, kan det möjliggöra replay-attacks. Detta är också något som inte är unikt för JavaCard utan gäller för alla system som kommunicerar. Ett exempel på detta är om programmet på kortet kräver att användaren loggar in och sedan skickar över inloggningsuppgifter till kortet. Någon som avlyssnar överföringen kan spela in denna sändning och spela upp den vid ett senare tillfälle och på så sätt också få tillgång till kortet. Detta skulle kunna till exempel lösas genom serienummer på meddelandet och kryptering.

4. Slutsatser

Från analysen och de frågeställningar som ställts kan vi dra ett antal slutsatser kring säkerheten för JavaCard.

4.1 Krypteringsmoder

De lägen som JavaCard erbjuder till sina symmetriska krypteringsalgoritmer är Electronic CodeBook (ECB) och Cipher Block Chaining (CBC), där ECB krypterar blocken oberoende av varandra och chiffertexten i CBC beror på det aktuella blocket och den tidigare chiffertexten. Det kan ibland vara önskvärt att ha flera lägen tillgängliga, vilka saknas i JavaCard. Ett av dem är Cipher Feedback (CFB), som krypterar 8 bitar i taget vilket kan vara önskvärt om en liten mängd data behöver krypteras. Det finns ytterligare två lägen, Output Feedback (OFB) och Counter Mode. Dessa är mer lika så kallade stream ciphers vilka endast gör en addition på varje bit som skickas. Detta gör att integriteten förloras, men att felpropageringen elimineras.

Om man utgår från JavaCard och vilka tillämpningar den är till för, ser man att läget CFB kan vara nyttigt att implementera i framtiden. Detta eftersom att man i många fall kommunicerar med en liten mängd data, vilket gör det lämpligt att ha tillgång till 8-bitars kryptering utan att behöva fylla ut texten till 64 bitar.

4.2 Applets

JavaCard ger en bra grund kring en säker hantering av en applet. Bland annat genom att stänga inne appleten i en virtuell maskin, verifiering av innehållet i appleten, brandvägg mellan applets och möjligheten till signering av applets från tillverkarens sida. På ren applikationsnivå är det dock upp till utvecklaren att göra ett så bra program som möjligt. Även om miljön kring appleten är säker, kan dåliga program fortfarande skrivas. Genom hårda testningar och kontroller kan många av bristerna upptäckas och korrigeras.

4.3 Säkerhet

JavaCard saknar vissa säkerhetsfunktioner från vanliga Java. De hårdvaruattacker som fungerar på andra smartcards verkar fungera även mot JavaCard, dock så finns det nyare kort som bland annat lägger på brus för att ge en jämn strömkonsumtion för att hindra analyser, har bättre skydd mot magnetfält och till exempel bättre skydd som utlöses vid intrångsförsök på kortet. Dessa typer av skydd gäller dock alla typer av moderna avancerade smartcards och är inte specifika för JavaCard.

4.4 Fördelar

Fördelarna med JavaCard är att det går att uppdatera mjukvaran efter ett kort har släppts och då även lägga till nya funktioner. Detta kan även göras på vissa andra smartcards, men JavaCard ger en bra infrasturktur för säker uppdatering av ny mjukvara. JavaCard som öppen multiapplikationsplattform är förhållandevis säker, även med den markant ökade risken att köra flera program, från olika tillverkare, på samma kort.

4.5 Förbättringar

Tillverkaren Sun förespråkar JavaCard på grund av enkelheten att utnyttja vanlig Javakod, vad vi har förstått så krävs det ganska mycket förändring för att det ska fungera med Javacards API och den virtuella maskin som existerar i JavaCard. Inlärningskurvan som ändå verkar finnas var något vi trodde skulle vara en fördel med JavaCard. Att ett smartcard ändå använder en semantik som många personer känner till ser vi som en fördel. Vi skulle dock vilja se JavaCard-miljön, med exempelvis den virtuella maskinen, ännu mer lik Java. Detta för att slippa behöva tänka om när man ska gå ifrån Java till JavaCard.

5. Liknande arbeten

Vårt arbete bygger på de källor som omnämns i referenslistan. Flera av dem behandlar säkerheten kring JavaCard, men inom olika områden. Vi har försökt att sammanställa en så bra bild som möjligt av JavaCard. För vidare information är dessa artiklar och böcker att rekommendera.

Referenser

[1] Zhiqun Chen. *Java Card Technology for Smart Card*. Addison-Wesley, 2004.

[2] Wikipedia. *Smart Card*. http://en.wikipedia.org/wiki/Smart_card, 2008-04-07.

[3] Java World. *Understanding Java 2.0.* http://www.javaworld.com/javaworld/jw-03-1998/jw-03-javadev.html?page=1, 2008-04-07.

[4] Smart Card Basics. *Smart Card Basics.* http://smartcardbasics.com/, 2008-04-07.

[5] Sun. *Java Card Platform Specifikation 2.2.2* http://java.sun.com/javacard/specs.html, 2008-04-20

[6] Sun. *Java Card Security Platform.* http://java.sun.com/javacard/reference/docs/JavaCard SecurityWhitePaper.pdf, 2008-04-20

[7] Gary McGraw, Edward Felten. *Securing Java.* http://www.securingjava.com/chapter-eight/chapter-eight-7.html, 2008-04-21.

[8] Pierre Girard Jean-Louis Lanet. *Java Card or How to Cope with the New Security Issues Raised by Open Cards?* http://www.cert.fr/francais/deri/wiels/Pacap/rapports-pres/GDC99.pdf, 2008-04-21.

[9] C. Enrique Ortiz. *An introduction to Java Card technology – Part 1.* http://developers.sun.com/mobility/javacard/articles/javacard1, 2008-04-21

[10] Gary McGraw, Edward Felten. *Securing Java.* http://www.securingjava.com/chapter-eight/chapter-eight-5.html, 2008-04-21

[11] K.O. Gadellaa. *Fault Attacks on Java Card.* http://alexandria.tue.nl/extra1/afstversl/wsk-i/gadellaa2005.pdf, 2008-04-21

[12] Dennis Vermoen. *Reverse engineering of Java Card applets using power analysis.* http://ce.et.tudelft.nl/publicationfiles/1162_634_thesis_Dennis.pdf, 2008-04-24

[13] IBM Zurich Research Laboratory. *JCOP embedded software.* http://www.zurich.ibm.com/csc/infosec/jetz.html#jee, 2008-04-24.