Zvýšení bezpečnosti softwarových aplikací pomocí kryptografické čipové karty

Petr Švenda
Masarykova univerzita v Brně
Fakulta informatiky
xsvenda@fi.muni.cz

Vašek Matyáš
Masarykova univerzita v Brně
Fakulta informatiky
matyas@fi.muni.cz & t-matyas@microsoft.com
(Microsoft Research Ltd., Cambridge, UK)

Abstrakt

Příspěvek se zabývá možností zvýšení bezpečnosti běhu softwarového agenta využitím programovatelné kryptografické čipové karty s podporou JavaCard a kombinací s metodami mobilní kryptografie. Popisuje ochranné rozhraní, umožňující přesun citlivých částí kódu do prostředí čipové karty, vzdáleně spravovatelné příkazy ve formátu XML. Je navržena speciální implementace autentizačního a transportního protokolu pro řízené využívání přesunutého kódu, která poskytuje ochranu autentizačním informacím uchovávaným na straně softwarového agenta a kódu provádějícího kroky protokolu i v případě, že je výpočetní prostředí pod kontrolou útočníka. Je využita implementace šifrovacího algoritmu AES, která umožňuje ukrýt hodnotu používaného klíče. Jsou navržena rozšíření zvyšující použitelnost této implementace v šifrovacím režimu CBC a umožňující větší provázanost s kódem softwarového agenta ve stylu mobilní kryptografie. Výsledný systém lze použít pro implementaci DRM architektur nebo kontroly přístupu k uživatelovým privátním informacím umístěným na čipové kartě, například při využití podpisového klíče.

1. Úvod

Digitální zpracování dat poskytuje mimo jiné možnost snadné tvorby identických kopií a rychlou distribuci bez ohledu na geografickou vzdálenost. Přináší s sebou však také problém kontroly jejich použití tak, aby byly zachovány zájmy všech zúčastněných stran. Efektivní správa práv k digitálním objektům (Digital Rights Managment, dále DRM) by měla tento cíl plnit. DRM, prováděná ve výpočetním prostředí koncového uživatele, bývá typicky zajištěna pomocí autonomní softwarové aplikace (dále softwarový agent) odpovědné za kontrolu využití digitálních dat v souladu s definovanými právy. Problémem běžných výpočetních prostředí typu PC je možnost útočníka plně kontrolovat běh tohoto softwarového agenta, především možnost čtení kódu a zpracovávaných dat a jejich následná modifikace. Útočník tak může docílit nežádoucí změny chování softwarového agenta úpravou jeho kódu anebo využít získaných dat k přímému přístupu k chráněným objektům.

Problémy související se zajištěním bezpečnosti běhu softwarové agenta lze částečně řešit použitím programovatelné kryptografické čipové karty, pokud poskytuje vyšší stupeň bezpečnosti než původní prostředí. Citlivý kód a data lze přenést a vykonávat na čipové kartě a využívat tak její hardwarové i softwarové ochranné mechanismy pro obranu před útočníkem. Vzhledem k omezeným výpočetním, paměťovým i funkčním (zobrazování dat apod.) prostředkům čipové karty však není možné ve většině případů přesunout celou funkčnost softwarového agenta, ale jen jeho vybrané části. Softwarový agent následně může využívat přesunuté části (dále chráněný algoritmus) pouze nepřímo, zasíláním vstupních dat a přijímáním zpracovaných výstupních dat. Pokud není požadováno omezení množiny softwarových agentů, které mohou chráněný algoritmus využívat a jde pouze o zajištění integrity výkonu nebo utajení implementace a důvěrnost použitých dat, je toto řešení postačující. Pokud je zapotřebí omezit množinu softwarových agentů oprávněných k využívání chráněného algoritmu, je třeba přistupující softwarové agenty vhodným způsobem autentizovat. Klíčovým problémem se stává uchování autentizačních informací na straně softwarového agenta, neboť ten se nachází v prostředí pod možnou kontrolou útočníka. Navíc musí být zajištěna i integrita celého procesu autentizace, neboť dochází i k autentizaci bezpečnostní proxy vůči softwarovému agentovi. V závislosti na povaze zpracovávaných dat může být navíc požadováno důvěrnost, integrita a čerstvost vyměňovaných zpráv.

V tomto příspěvku se věnujeme popisu systému adresujícího tento problém (dále ochranného rozhraní), pracujícímu v prostředí běžného PC s připojitelnou čipovou kartou podporující technologii JavaCard.

V dalším textu bude používáno následující označení zúčastněných stran: *Poskytovatelem* budeme označovat stranu,

která implementuje funkčnost softwarového agenta, integruje ochranné rozhraní a vzdáleně ovládá celý systém. *Uživatelem* pak budeme označovat tu stranu, v jejímž výpočetním prostředí je softwarový agent vykonáván, a k němuž je připojena čipová karta. *Útočníkem* bude označen takový uživatel, který využívá (neomezené) kontroly nad svým výpočetním prostředím k ovlivnění chování softwarového agenta v rozporu se záměrem poskytovatele.

Dělení textu je následující: Kapitola 2 nastiňuje problematiku ochrany softwarového agenta a uvádí vybrané typy ochran. Kapitola 3 popisuje základní strukturu ochranného rozhraní, poskytovanou funkčnost, možnosti vzdálené správy a modelový postup nasazení. V kapitole 4 jsou shrnuty základní vlastnosti implementace šifrovacího algoritmu AES navrženého v [CEJO02]. Kapitola 5 popisuje autentizační a transportní protokol navržený pro bezpečnou komunikaci mezi softwarovým agentem a čipovou kartou. V kapitole 6 jsou podrobněji rozebrány stavební prvky použité při implementaci tohoto protokolu. Kapitola 7 uvádí příklady scénářů, ve kterých je možné ochranné rozhraní využít.

2. Ochrana softwarového agenta

2.1. Typy útoků

Poskytovatel stojí před úkolem zabezpečit softwarového agenta tak, aby nedošlo k jeho zneužití útočníkem, který může své (prakticky) neomezené kontroly nad výpočetním prostředím softwarového agenta využít k následujícím typům útoků:

Zisk a modifikace kódu nebo dat – cílem je vytvoření mentálního modelu (části) softwarového agenta z jeho kódu pro následnou modifikaci jeho chování, zisk citlivých dat (šifrovací klíče) nebo extrakci obsažených návrhových myšlenek.

Tvorba diagramu toku dat – cílem útoku je rekonstrukce diagramu toku dat pro vybrané operace prováděné softwarovým agentem. Z vytvořeného diagramu a znalosti vstupních dat může útočník dedukovat informace o vnitřním stavu softwarového agenta a tento stav záměrně modifikovat. Získaný diagram toku dat může být použit pro usnadnění tvorby mentálního modelu nebo modifikaci rozhodovacích podmínek.

Výkon kódu v nekorektním prostředí – cílem útoku je ovlivnění softwarového agenta jeho výkonem v prostředí s nekorektními vnějšími podmínkami. Příkladem nekorektních vnějších podmínek je nedostupnost požadovaných zdrojů (paměť), nekorektní formát vstupních dat nebo chybná funkčnost systémových funkcí včetně podvržení identity hostitelského systému. Může být dosaženo negativního omezení činnosti agenta potlačením funkcí kritických z hlediska poskytovatele (kontrola revokace, logování).

Manipulace komunikace – cílem útoku je narušení důvěrnosti, autenticity nebo čerstvosti komunikace softwarového agenta s druhou stranou. Útok je zaměřen na zisk přenášených dat (citlivé informace, šifrovací klíče) nebo na nekorektní chování softwarového agenta následkem manipulace vyměňovaných zpráv v průběhu komunikace.

2.2. Ochranné techniky

Pro ochranu softwarového agenta proti výše uvedeným útokům lze využít široké spektrum technik. Zde uvedené jsou děleny na čistě softwarové, které nevyužívají pomocná hardwarová zařízení a na ty, které výhod hardwarově chráněných zařízení využívají.

2.2.1. Softwarové ochrany

Z běžně rozšířených lze uvést využití registračního čísla měnícího se v závislosti na registračních údajích uživatele (jméno, firma) nebo hodnotách prostředí (sériová čísla harddisků, síťových karet, CPU), kontrola originálního média na očekávané vlastnosti (fyzické chyby, softwarové chyby, identifikační znaky). Pro detekci a eliminaci nástrojů využívaných útočníkem při provádění statické a dynamické analýzy (disassemblery, debuggery) nebo monitorování aktivit softwarového agenta (Regmon, Filemon) lze do jeho kódu umístit metody využívajících znalosti konkrétních nástrojů nebo obecných principů jejich chování. Dobrý přehled použitelných technik lze nalézt v [Ce02, Ze02].

Mezi pokročilejší techniky této skupiny patří nástroje pro automatickou transformaci kódu na sémanticky ekvivalentní, zároveň však ztěžující zpětnou tvorbu mentálního modelu (obfuskace). Výpočetní transformace vkládají do funkčního kódu mrtvý nebo irelevantní kód, rozšiřují podmínky skoků o tautologie, odstraňují standardní programovací vzory, zavádějí redundantní operace, paralelizují sekvenční kód, agregují nesouvisející bloky kódu nebo nahrazují volání knihovních funkcí přímo jejich kódem. Datové transformace mění kódování dat, rozdělují a spojují proměnné, restrukturalizují datová pole a dynamicky generují statická data. Zároveň zavádějí transformace, které zvyšují paměťovou a výpočetní náročnost u nástrojů určených pro automatické odstranění provedené obfuskace. Principy základních obfuskačních metod lze nalézt v [CTL97, CTL98, CGJZ01, NCJ01].

Pro zamezení inspekce kódu a cílené modifikace lze použít šifrování kódu agenta klíčem ukrytým v jeho nešifrované části nebo odvozeným z předem známých charakteristik výpočetního prostředí. Dešifrování může být prováděno najednou během zavádění do operační paměti, nebo je dešifrována pouze aktuálně vykonávaná část (metoda plovoucího dešifrujícího „okna“). V prvním případě je vliv na rychlost běhu zanedbatelný, útočník však může využít nástrojů pro ukládání paměti a získat tak dešifrovanou podobu softwarového agenta. Ve druhém případě je v paměti vždy jen malá část

celkového kódu, cenou je však snížení celkového výkonu následkem opakovaného dešifrování stejných částí kódu. Vhodnou volbou velikosti plovoucího okna lze nalézt kompromis mezi bezpečností a požadovaným výkonem softwarového agenta. Kritickým místem je v obou případech ukrytí metody pro generování a použití šifrovacího klíče. Slibnou metodu, umožňující výkon „šifrovaného“ kódu bez nutnosti jeho dešifrování, poskytuje pro některé třídy funkcí technika mobilní kryptografie prezentovaná v [SaTs98]. Je využívána popisovaným ochranným rozhraním a více popsána v kapitole 4. Při použití mobilní kryptografie lze přesněji stanovit výpočetní obtížnost odstranění ochrany.

Pokud je třeba chránit kód softwarového agenta pouze do doby jeho prvního použití, lze využít techniku neinformovaných agentů. Softwarový agent aplikuje dvakrát jednosměrnou hashovací funkci H na zvolenou událost v prostředí (jméno cílového počítače) a porovnává ji s nesenou hodnotou I. V případě shody použije hodnotu získanou pouze jednou aplikací funkce H k odvození šifrovacího klíče K. Při použití bezpečné funkce H nemůže útočník ze znalosti I odvodit hodnotu K. Další varianty použití lze nalézt v [RiSch98].

Na myšlence opakované kontroly integrity částí kódu jsou založeny ochrany proti modifikaci kódu prezentované v [ChaAt01, HMST01]. Kontrola je prováděna pomocí velkého počtu (řádově stovky) malých kusů kódu nazývaných testery, které co nejméně nápadně ověřují integritu přidělené části kódu. Při zjištění modifikace spouštějí reakční mechanismus, který na situaci příslušně zareaguje, například násilným ukončení běhu nebo chybnou funkčností. Při změně kódu pak musí útočník deaktivovat i tester nebo reakční mechanismus odpovědný za kontrolu modifikované části. Při násobném překrývání testovacích oblastí je nutno deaktivovat hned několik testerů nebo reakčních mechanismů. Kód testeru a reakčního mechanismu se zároveň nachází také v kontrolované oblasti a testery se tak navzájem chrání proti modifikaci.

2.2.2. Pomocná hardwarová zařízení

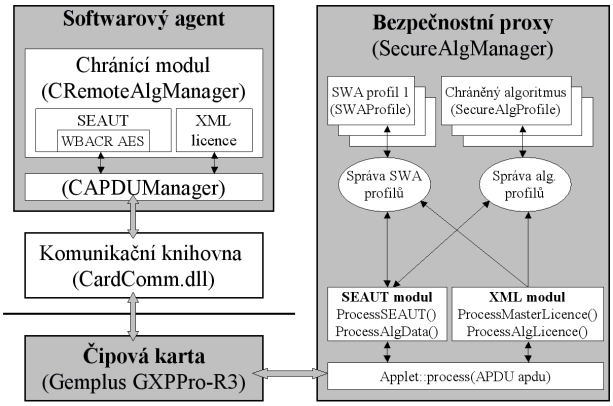
Ochrany tohoto typu spoléhají na využití pomocného hardwarového zařízení, u kterého náklady na zisk chráněných informací nebo vytvoření kopie překračují hodnotu informace, která má být chráněna před útočníkem. Nabízená funkčnost a rozsah nasazení hardwarových zařízení se pohybuje od zabezpečeného datového nosiče (paměťové karty), přes zařízení schopné provádět vybrané operace na vloženými daty (hardwarové klíče, čipové karty), až po komplexní výpočetní platformu v rozsahu současných PC (Trusted Computing Group).

Do této kategorie spadá i navrhované ochranné rozhraní využívající čipovou kartu s podporou JavaCard. Ve srovnání s jednoduchými hardwarovými klíči typu „dongles“ (Rainbow Sentinel, Alladin HASP) přináší větší výpočetní výkon, možnost bezpečné vzdálené aktualizace umístěných programových balíků prostřednictvím rozhraní OpenPlatform [OP02] a bezpečné sdílení výpočetního prostředí více stranami. Ochranné rozhraní neposkytuje mnohé z možností nabízených výpočetní platformou TCG [TCG04, An03], narozdíl od ní však nevyžaduje od uživatele nákladný přechod na specializovaný hardware. S výhodou lze využít běžně rozšířené kryptografické čipové karty distribuované za jiným účelem.

3. Ochranné rozhraní

Ochranné rozhraní využívá spolupráce softwarového agenta s čipovou kartou s podporou JavaCard a skládá se ze dvou částí. Část umístěná na čipové kartě obsahuje jednoduchý XML parser, základní kostru (rodičovský objekt) pro implementaci chráněných algoritmů a především funkčnost bezpečnostní proxy. Bezpečnostní proxy je jediný objekt, se kterým mohou softwaroví agenti přímo komunikovat prostřednictvím APDU příkazů. Veškerá komunikace mezi softwarovým agentem a chráněným algoritmem je zprostředkována přes tuto proxy, která na základě definovaných pravidel povoluje nebo zamítá jejich využití a zajišťuje zabezpečení komunikace. Část integrovaná do kódu softwarového agenta obsahuje funkčnost nutnou pro ustanovení komunikačního kanálu s bezpečnostní proxy, vzájemnou autentizaci a zasílání/zpracování dat vyměňovaných s chráněným algoritmem. Jako samostatná aplikace nebo součást softwarového agenta je přítomen modul pro předávání řídících příkazů ve formátu XML zasílaných poskytovatelem pro řízení bezpečnostní proxy. Implementace tohoto modulu nevyžaduje žádné zvláštní zabezpečení před útočníkem, neboť pouze předává komunikaci zabezpečenou na vyšší úrovni a nedisponuje žádnou tajnou informací.
Klíčové prvky ochranného rozhraní:

- Využití chráněného výpočetního prostředí programovatelné kryptografické čipové karty.
- Návrh způsobu autentizace a komunikace mezi čipovou kartou (bezpečnostní proxy) a softwarovým agentem zohledňující fakt, že softwarový agent je vykonáván v prostředí pod možnou kontrolou útočníka.
- Sdílení prostředí čipové karty více chráněnými algoritmy.
- Dynamická definice množiny softwarových agentů oprávněných k využívání konkrétního chráněného algoritmu.
- Vzdálená aktualizace interních hodnot chráněných algoritmů.



Obrázek 1: *Základní schéma ochranného rozhraní.*

- Vzdálené správa celého systému pomocí zpráv ve formátu XML.

3.1. Komunikace od softwarového agenta

Přenos citlivé části z kódu softwarového agenta je realizován obdobným způsobem, jakým je postupováno při tvorbě a využití běžné knihovní funkce. Požadovaná funkčnost je implementována v prostředcích jazyka JavaCard [JC22API] přetížením metody, kterou volá bezpečnostní proxy po vyhodnocení oprávněnosti žádosti softwarového agenta o využití chráněného algoritmu. Původní funkčnost obsažená v těle softwarového agenta je nahrazena voláním metodu ochranného rozhraní, která zajistí ustanovení komunikačního kanálu, zaslání vstupních dat a zpracování výstupní odpovědi.

3.2. Vzdálená správa

Správa bezpečnostní proxy a interních hodnot jednotlivých chráněných algoritmů je prováděna pomocí dávkových příkazů ve formátu XML. Utajení a integrita dávek je zabezpečena pomocí šifrovacích klíčů sdíleného mezi poskytovatelem a bezpečnostní proxy. Zpracování dávky probíhá přímo na čipové kartě pomocí jednoduchého XML parseru vycházejícího z [Br02] (režim SAX) z důvodu zajištění integrity parsování a utajení zpracovávaných hodnot.

3.2.1. Bezpečnostní proxy

Řídícími příkazy je prováděna kompletní správa bezpečnostní proxy. Zajišťují obecnou funkčnost nutnou pro komunikaci mezi softwarovým agentem a bezpečnostní proxy, nezávislou na konkrétní implementaci chráněných algoritmů. Jejich zpracování by nemělo být nutné měnit. Řídící příkazy umožňují:

- Vytvoření/modifikaci/zrušení profilu softwarového agenta.
- Vytvoření/modifikaci/zrušení profilu chráněného algoritmu.
- Modifikace množiny softwarových agentů oprávněných využívat konkrétní chráněný algoritmus.
- Modifikace šifrovacích klíčů používaných pro zabezpečení dávkových zpráv.

Pomocí řídících příkazů lze dynamicky vytvářet vytvářet a rušit unikátně identifikované instance implementovaných tříd chráněných algoritmů. Každý softwarový agent musí mít v bezpečnostní proxy vytvořen svůj profil. Ten obsahuje jeho unikátní identifikaci, hodnoty šifrovacích klíčů sdílených s bezpečnostní proxy a seznam chráněných algoritmů, které mohou být tímto softwarovým agentem využívány. Bezpečnostní proxy pomocí unikátní identifikace jednotlivých instancí zajišťuje předání vstupních dat cílovým chráněným algoritmům. Profilů pro softwarové agenty i chráněné algoritmy může být vytvořeno více, až do předem definovaného počtu daného paměťovými možnosti použité čipové karty. Je umožněno souběžné využití chráněného algoritmu více softwarovými agenty.

3.2.2. Chráněné algoritmy

XML parser je přístupný i pro aktualizaci interních hodnot jednotlivých chráněných algoritmů. Implementace zpracování je plně prováděna poskytovatelem, ochranné rozhraní pouze zajišťuje bezpečný přenos a předání XML parseru. Možným využitím je vzdálená definice dodatečných podmínek, za kterých je umožněno použití chráněného algoritmu i pro oprávněné softwarové agenty. Typickým příkladem je použití čítače, který je při každém využití chráněného algoritmu snižován. Chráněný algoritmus je proveden pouze tehdy, pokud je hodnota čítače kladná. Čítač lze zvyšovat pouze prostřednictvím XML licence distribuované poskytovatelem, čehož lze využít pro základ DRM architektury.

3.3. Otázka výkonu

Přenos citlivých částí do prostředí čipové karty má vliv na celkový výkon softwarového agenta. Nemusí se nutně jednat o snížení výkonu, neboť čipová karta může poskytovat hardwarovou akceleraci některých operací chráněného algoritmu. Při použití běžné kryptografické čipové karty však výkon pravděpodobně omezen bude a je tedy nutno vhodně zvolit, která část kódu softwarového agenta bude chráněna. Na změnu výkonu mají vliv následující faktory: a) rychlost vytvoření autentizovaného kanálu dle SEAUT, b) propustnost datové vrstvy mezi softwarovým agentem a čipovou kartou, c) propustnost komunikační vrstvy dle SEAUT, d)

Operace	Čas
Zaslání 1 APDU (0 B vstup, 0B výstup)	0,07 s
Zaslání 1 APDU (256 B vstup, 0B výstup)	0,16 s
Zaslání 1 APDU (256 B vstup, 256B výstup)	0,3 s
Použití algoritmu (0 B dat = 1 APDU)	0,36 s
Použití algoritmu (240 B dat = 1 APDU)	0,61 s
Použití chráněného algoritmu (1 kB dat = 4 APDU)	2,4 s
Navázání autentizovaného spojení dle SEAUT	0,52 s
Zpracování příkazu [NewAlg] (220 B = 1 APDU)	13,7 s
Zpracování příkazu [RemoveAlg] (130 B = 1 APDU)	6 s

Tabulka 1: *Trvání vybraných operací ochranného rozhraní na čipové kartě Gemplus GXPPro-R3. Uvedené hodnoty jsou pouze orientační, nebot'použitá karta nemá implementovánu hardwarovou podporu algoritmu AES. Hodnoty vycházejí z měření při použití algoritmu DES. Pro potřeby testu chráněný algoritmus nad vstupními daty neprovádí žádnou operaci, pouze je předá v nezměněné podobě na výstup. Příkaz [NewAlg] vytvoří nový profil chráněného algoritmu. Příkaz [RemoveAlg] odstraní profil chráněného algoritmu.*

rychlost přípravy dat pro chráněný algoritmus na straně softwarového agenta a e) rychlost provedení chráněného algoritmu nad vstupními daty. Pro konkrétní typ čipové karty se liší vliv jednotlivých faktorů. V současné době je významným faktorem především propustnost datové vrstvy díky omezené rychlosti komunikace prostřednictvím APDU příkazů. V závislosti na používaných operacích může být významným faktorem e). Pro zanedbatelný vliv faktorů a) a c) je na straně hardwarového tokenu vyžadována hardwarová podpora šifrovacího algoritmu AES.

Tabulka 1 zachycuje výsledky orientačního testu, provedeného za účelem odhadu doby trvání základní operací ochranného rozhraní. Pro odhad byla použita záměna šifrovacího algoritmu AES za algoritmus DES, hardwarově podporovaného použitou čipovou kartou Gemplus GXPPro-R3. Z porovnání rychlosti hardwarových implementací algoritmů DES a AES uvedených v [GaCho01, SaMo03] lze při hardwarové podpoře AES očekávat mírně lepší výsledky oproti uvedeným. Celkové zrychlení bude závislé na poměru šifrování a ostatních operací.

3.4. Dodatečné ochranné techniky

Konstrukce části ochranného rozhraní integrovaného v kódu softwarového agenta (především implementace komunikačního protokolu SEAUT) je navržena s cílem ztížit útočníkovi provedení smysluplné modifikace kódu, získání vyměňovaných dat a podvržení dříve zachycených zpráv vyměňovaných s bezpečnostní proxy. Cílem použitých technik je donutit útočníka vytvářet mentální model z co největší části kódu, v případě pokusu o modifikaci pak zvýšit počet míst nutných pro změnu chování. Representace kryptografických klíčů u softwarového agenta pomocí WBACR AES tabulek (viz kapitola 4) chrání jejich otevřenou hodnotu, nezabraňuje však útočníkovi v jejich použití pokud dojde k jejich extrakci. Proto je vhodné doplnit ochranné rozhraní o další ochranné prostředky. Vhodnými kandidáty jsou obfuskační nástroje pro ztížení extrakce nebo nahrazení WBACR AES tabulek, použití sebekontrolujícího kódu pro zajištění integrity a ochrany zaměřené proti standardním statickým a dynamickým inspekčním nástrojům. Použitelnost konkrétních nástrojů nebyla zkoumána.

3.5. Postup nasazení

Následujících pět kroků shrnuje logické operace, které je potřeba učinit pro integraci a nasazení ochranného rozhraní.

1. Fyzická distribuce čipové karty – distribuovanou čipovou kartu lze pro potřeby ochranného rozhraní využít opakovaně. S výhodou lze použít čipových karet, které již uživatel vlastní, například SIM karty nebo podpisové čipové karty. Podmínkou je samozřejmě splnění funkčních a bezpečnostních požadavků.

2. Implementace chráněných algoritmů – druhým krokem je implementace chráněných algoritmů, překlad a (vzdálená) instalace bezpečnostní proxy na čipovou kartu. Instalovaný aplet, který přestane postačovat požadavkům může být vzdáleně odstraněn a opakováním druhého kroku nahrazen novým. Vzdálenou instalaci lze provést bezpečným způsobem prostřednictvím vhodného rozhraní podporovaného kartou, například specifikace OpenPlatform [OP02] podporující instalaci podepsaných a šifrovaných apletů. Bezpečnostní proxy je během instalace přiřazena unikátní identifikace.

3. Implementace softwarového agenta – dochází k začlenění volání metod ochranného modulu, přiřazení unikátní identifikace softwarového agenta a generování WBACR AES tabulek pro použité šifrovací klíče. Softwarový agent je přeložen do spustitelné podoby a distribuován k uživateli. Třetí krok může být dle potřeby opakován, pomocí následného čtvrtého kroku lze povolovat a omezovat využití služeb bezpečnostní proxy pro distribuované softwarové agenty.

4. Správa XML příkazy – čtvrtým krokem je tvorba řídících příkazů, které vytvoří profil softwarového agenta u cílové bezpečnostní proxy a povolí využití zvolených chráněných algoritmů. Lze vytvářet a distribuovat příkazy pro aktualizaci interních hodnot chráněných algoritmů. Čtvrtý krok je dle potřeby opakován. Příkazy jsou distribuovány v šifrované podobě zabezpečené klíčem sdíleným mezi bezpečnostní proxy a poskytovatelem.

5. Využití chráněného algoritmu – pátým krokem je využití chráněného algoritmu softwarovým agentem. Softwarový agent vytváří bezpečný komunikační kanál a zasílá

bezpečnostní proxy požadavky. Bezpečnostní proxy rozhoduje o oprávněnosti požadavku na základě autentizace softwarového agenta a seznamu chráněných algoritmů, které může softwarový agent využívat. V kladném případě zasílá zpracovaný požadavek zpět softwarovému agentovi.

4. WhiteBox Attack Resistant AES

Pro ochranu hodnot šifrovacích klíčů a dat zpracovávaných softwarovým agentem je využito konceptu mobilní kryptografie, navrženém v [SaTs98]. Výpočtem s šifrovanou funkcí (Computing with Encrypted Function) je zde označován proces, při kterém strana B vykoná nad vlastními vstupními daty X program P, poskytnutý stranou A, realizující operaci F. Program P realizuje operaci F takovým způsobem, že strana B se během výkonu P nad X nedozví žádnou podstatnou informaci, která by sloužila k odhalení operace F.

Tato myšlenka je využita pro implementace šifrovacího algoritmu AES navrženém v [CEJO02] (dále WBACR AES). Umožňuje šifrovat v prostředí pod kontrolou útočníka bez vyzrazení hodnoty použitého šifrovacího klíče. Narozdíl od běžných implementací, přijímajících na vstupu data a hodnotu šifrovacího klíče, je přijímán pouze blok vstupních dat. Ten je modifikován pouze pomocí série náhledů do předpočtených tabulek. Po posledním náhledu je vstupní blok zašifrován klíčem, který byl použit pro generování tabulek. Šifrování využívající WBACR AES má oproti standardní implementaci několik předností:

1. Utajení hodnoty použitého klíče – zpětný zisk hodnoty klíče je výpočetně obtížný i při znalosti vygenerovaných tabulek. Průběžné výsledky během série náhledů jsou chráněny pomocí náhodné bijektivní transformace generované během tvorby tabulek.

2. Oddělitelná šifrovací a dešifrovací funkčnost – vygenerované tabulky pro šifrování a dešifrování jsou navzájem nezávislé. Softwarový agent tak může pro daný klíč obsahovat tabulky použitelné pouze pro šifrování nebo dešifrování. Díky této vlastnosti lze vyměňovaná data šifrovat pomocí symetrické kryptografie a zároveň využít výhod plynoucích ze dvou oddělených klíčů asymetrické kryptografie. Dešifrovací část tabulek může sloužit jako „veřejný“ klíč distribuovaný všem softwarovým agentům. Šifrovací část vlastní pouze jeden softwarový agent a slouží mu jako „privátní“ klíč. Proces vytváření digitálního podpisu zprávy může být nahrazen pomocí časově méně náročné tvorby autentizačního kódu (MAC).

3. Vstupní a výstupní kódování – vstupním kódováním je myšlena bijektivní transformace vstupního bloku dat, která je v rámci náhledů do předpočtených tabulek pro první rundu pomocí inverzní transformace (zahrnuté v tabulkách) odstraněna. Tato transformace je volena náhodně během generování tabulek. V případě identické transformace jsou vstupní data přijímána přímo, v opačném případě je třeba na ně před začátkem šifrování aplikovat tuto transformaci. Analogicky pro výstupní kódování aplikované v rámci náhledů poslední rundy, které je nutno před použitím výstupních dat inverzní transformací odstranit. Použití vstupního anebo výstupního kódování ztěžuje možnost samostatného použití tabulek extrahovaných z kódu softwarového agenta, neboť útočník musí získat i předpis použitých bijektivních transformací a zvyšuje celkovou provázanost kódu. Navíc poskytuje základ pro „napojení“ dalších operací ve stylu mobilní kryptografie.

5. SEcure AUthenticated Transport protokol

Pro zajištění vzájemné autentizace, důvěrnosti a čerstvosti vyměňované komunikace mezi bezpečnostní proxy a softwarovým agentem byl navržen protokol (Secure Authenticated Transport protokol, dále SEAUT), který zohledňuje umístění softwarového agenta v prostředí pod možnou kontrolou útočníka (prostředí bezpečnostní proxy je považováno za bezpečné). Běžné autentizační protokoly na bázi symetrické nebo asymetrické kryptografie nelze použít, neboť útočník má možnost na straně softwarového agenta pomocí ladících nástrojů číst hodnoty autentizačních informací, modifikovat hodnoty proměnných, narušovat integritu kroků použitého protokolu nebo využít autentizačních funkcí ve vlastní režii. Obdobné problémy vyvstávají při potřebě důvěrnosti a čerstvosti vyměňované komunikace.

Vhodná implementace použitého protokolu by měla splňovat následující podmínky: a) strana B neprovádí žádné porovnávací operace vzhledem k očekávaným hodnotám, b) autentizační informace strany B a informace sloužící k utajení vyměňovaných dat nejsou čitelné při použití statické a dynamické inspekce, c) strana B obsahuje robustní mechanismus kontroly čerstvosti, d) strana B je chráněna proti vytvoření mentálního modelu a modifikaci a e) strana B neobsahuje funkci generující zprávy zaměnitelné za zprávy pocházející od strany A, a to ani v případě, že útočník manipuluje se vstupními daty této funkce.

Navržený protokol SEAUT se skládá ze dvou částí, autentizační a transportní. Stranu umístěnou v bezpečném prostředí (bezpečnostní proxy) označme A, stranu umístěnou v prostředí pod možnou kontrolou útočníka (softwarový agent) označme B.

5.1. Autentizační část

Autentizační část navrženého protokolu je založena na 3-průchodovém protokolu ISO9798-2 (dále P3-ISO9798-2) pro vzájemnou autentizaci [ISO9798-2], využívajícím sdílený klíč K_S pro symetrickou kryptografii a náhodné číslo jako keksík pro zajištění čerstvosti.

Vyměňované zprávy (autentizační část):
1. $A \leftarrow B : \{id_B, N_B\}$.
2. $A \rightarrow B : \{id_A, E_{KD_A}(N_A, N_B, id_B)\}$.
3. $A \leftarrow B : \{E_{KE_A}(N_B, N_A)\}$.
4. $K_{R_{init}} = H_1(N_A|N_B|id_B|V(N_B)|V(id_B))$

Protokol SEAUT používá na místo jednoho klíče K_S klíče KE_A a KD_A. Pro šifrování 2. zprávy je použit klíč KD_A, pro 3. zprávu KE_A. Lze snadno nahlédnout, že strana B využívá klíč KE_A pouze pro zašifrování a klíč KD_A pouze pro dešifrování (opačně pro stranu A). Cílem této úpravy je zajistit, aby na straně B nemusela být přítomna funkčnost pro dešifrování klíčem KE_A. Klíč KE_A tak může být na straně B realizován pouze šifrovací částí WBACR AES tabulek (viz 4). Analogicky pro klíč KD_A.

Další modifikací je způsob vyhodnocení korektnosti provedené autentizace na straně B. Kontrola shody keksíku N_B a identifikace id_B vůči očekávaným hodnotám $V(N_B)$ a $V(id_B)$ z 1. a 2. zprávy je prováděna jen nepovinně, neboť ji útočník může snadno modifikovat nebo odstranit. Namísto toho je ze zaslaných i obdržených hodnot pomocí jednosměrné klíčované hashovací funkce H_1 (viz 6.2) vytvořena iniciální hodnota klíče relace K_R. Klíč relace K_R bude ustanoven i v případě nekorektní autentizace strany A vůči straně B, bude však odlišný od klíče vzešlého z korektní autentizace. Použití klíčované hashovací funkce (s využitím WBACR AES) pro jeho tvorbu zabraňuje útočníkovi odvodit jeho přímou hodnotu. Vzhledem ke způsobu použití K_R v transportní části pak nekorektní K_R povede chybnému zpracování vyměňovaných zpráv v případě, že útočník použije zprávy zachycené během předchozí (korektní) komunikace mezi A a B.

5.2. Transportní část

Transportní část protokolu zajišťuje důvěrnost a čerstvost vyměňovaných zpráv. K zachování důvěrnosti je využita dvojice šifrovacích klíčů KE_T a KD_T, realizovaných a používaných obdobně jako klíče KE_A a KD_A v autentizační části. Pokud strana B pomocí klíče KD_T korektně dešifruje příchozí zprávu, může si být jista, že pochází od strany A, neboť sama nedisponuje funkčností pro zašifrování tímto klíčem a útočník ji tak nemůže zneužít pro vytvoření podvržené zprávy. Ze stejného důvodu je chráněn před útočníkem obsah zachycených zpráv určených pro stranu A, neboť B nedisponuje funkčností pro jejich dešifrování.

Vyměňované zprávy (transportní část):
5. A, B vypočtou: $K_{R_i} = H_2(K_{R_{i-1}})$.
6. $A \leftarrow B : \{idA, idB, X_{K_{R_i}}(E_{KE_T}(request))\}$.
7. A, B vypočtou: $K_{R_{i+1}} = H_2(K_{R_i})$.
8. $A \rightarrow B : \{idA, idB, X_{K_{R_{i+1}}}(E_{KD_T}(response))\}$.

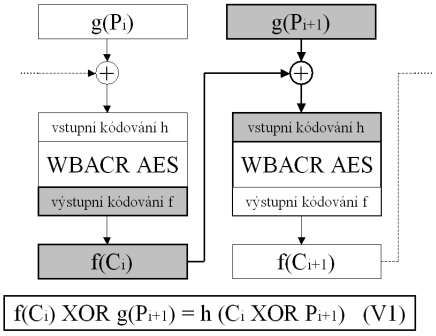
Problémem tak zůstává robustní zajištění čerstvosti (na straně B). Metody založené na porovnávacích kontrolách různých typů keksíku (náhodné číslo, čas, pořadové číslo) nelze použít z důvodu snadné manipulaci kontrolního kódu. Vzhledem k implementaci šifrovacího algoritmu pomocí WBACR AES nelze využít ani pravidelnou změnu šifrovacího klíče. Navržená implementace využívá klíče relace K_R, aktualizovaného pomocí klíčované hashovací funkce H_2 po každé odeslané i přijaté zprávě. Klíč K_R je použit takovým způsobem (operace X_{K_R}), aby ovlivnil každý bit odesílané i přijímané zprávy (viz 6.2). Jeho nekorektní hodnota vzhledem ke zpracovávané zprávě tak vede k jejímu poškození. Při použití vstupního a výstupního kódování pro aktualizaci K_R není na straně B jeho otevřená hodnota použita a je tak ztížena jeho lokalizace nebo modifikace.

WBACR AES tabulky pro šifrovací klíče KE_T a KD_T jsou generovány s využitím vstupního a výstupního kódování (narozdíl od KE_A a KD_A). Data získaná dešifrováním zprávy pomocí klíče KD_T tak nejsou ihned přístupná v otevřené podobě čitelné útočníkem. Odstranění použitého kódování může být provedeno postupně v následujícím kódu nebo může sloužit k napojení další operace ve stylu mobilní kryptografie. Zvyšuje se tím provázanost jednotlivých částí kódu, útočník má ztíženo získání dat v otevřené podobě a extrakci nebo nahrazení WBACR AES tabulek. Volbou různého vstupního a výstupního kódování lze provádět personalizaci softwarového agenta strany B nezávisle na straně A.

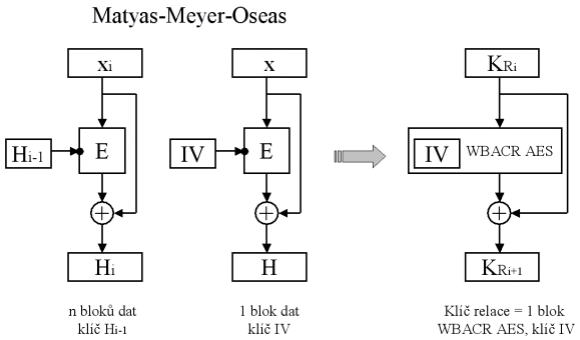
6. Stavební prvky SEAUT

6.1. I/O kódování pro CBC režim

Vstupní a výstupní kódování (dále IOC), jak je popsáno v [CEJO02], poskytuje způsob, jak ztížit použití WBACR AES tabulek, pokud dojde k jejich extrakci. Bez dodatečných úprav však IOC nelze použít pro jiný šifrovací režim než ECB. Použití pro režim CBC by vyžadovalo odstranění resp. aplikaci IOC před každou jeho iterací, což by výrazně snížilo rychlost šifrování. Zároveň by byl kód aplikující kódování s využitím dynamické inspekce snadno lokalizovatelný a jeho použití by ztratilo význam. Pro praktickou možnost využití IOC pro režim CBC byla navržena modifikace znázorněná na obrázku 6.1. Ke dvojici vstupního h a výstupního kódování f je přidáno datové kódování g. Narozdíl od původního však nejsou jednotlivá kódování nezávislá, ale jsou generována tak, aby byl splněn vztah $f(C_i)\ XOR\ g(P_{i+1}) = h(C_i\ XOR\ P_{i+1})$. Po aplikaci funkce XOR na výstup předchozí iterace (kódování f) a následujícího vstupu (kódování g) získáme hodnotu $h(C_i\ XOR\ P_{i+1})$. Použití IOC během celého procesu šifrování (resp. dešifrování) je transparentní a nevyžaduje žádnou změnu oproti běžnému režimu CBC. Datové kódování g



Obrázek 2: *Vstupní a výstupní kódování použitelné pro šifrovací režim CBC. P_i značí i-tý blok vstupních dat, C_i značí i-tý blok zašifrovaných dat.*



Obrázek 3: *Aktualizace klíče relace K_R s využitím WBACR AES.*

je aplikováno v libovolném místě před počátkem šifrování a může být součástí předchozí operace navržené ve stylu mobilní kryptografie. Analogicky pro výstupní kódování f.

Interní struktura WBACR AES prakticky umožňuje použít IOC o maximální velikosti 8 bitů. Nelze tedy použít jediné kódování pro celý 128 bitový šifrovaný blok. Operace XOR je ale „blokovatelná“ (výsledek i-tého bitu nezávisí na j-tém bitu), lze tedy použít 16 nezávislých IOC_1 až IOC_{16} pro 1 až 16 bajt bloku.

6.2. Použití a aktualizace K_R

Klíč relace K_R je vytvářen klíčovanou hashovací funkcí z očekávaných a obdržených hodnot během autentizační části protokolu SEAUT. Tvorba hashovací funkce z blokového šifrovače realizovaného s využitím WBACR AES zvyšuje bezpečnost K_R, neboť útočník nemůže bez extrakce klíče určit ve vlastní režii výslednou hodnotu K_R. Hashovací funkce má výstupní kódování odpovídající vstupnímu kódování K_R.

Hodnota K_R je použita dvěma způsoby:

1. Inicializační vektor pro režim CBC – cílem je zajistit, aby chybně ustanovený klíč relace K_R vedl k vytváření a zpracování nekorektních zpráv vyměňovaných během transportní části. Pokud útočník modifikuje běh softwarového agenta tak, aby pokračoval i přes nekorektní autentizaci, bude hodnota K_R odlišná a vyměňované zprávy nebudou zpracovány korektně.

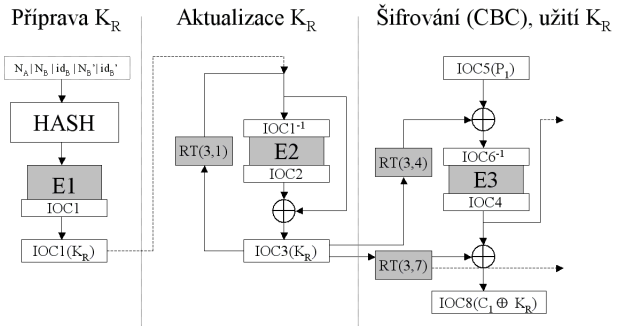
2. Opakovaná aplikace pomocí operace XOR na zašifrovaná data – cílem je zajistit, aby zpracování příchozí zprávy vytvořené pomocí jiné hodnoty K_R vedlo k chybnému zpracování celé zprávy, ne pouze prvního bloku jako v případě použití pro inicializační vektor.

Hodnota K_R je aktualizována po každé přijaté nebo odeslané zprávě. Navržená metoda aktualizace je založena na využití schématu Matyas-Meyer-Oseas (MMO) pro tvorbu hashovací funkce z blokového šifrovače, jak je znázorněno na obrázku 6.2. Vzhledem k fixní velikosti K_R dochází vždy pouze k jedinému cyklu MMO a hodnota použitého šifrovacího klíče se nemění. Tato vlastnost umožňuje realizovat blokový šifrovač šifrovací část9 WBACR AES tabulek. Navržená metoda zajišťuje vlastnost kryptografické jednocestnosti nejméně na úrovni MMO s jedním cyklem. Použití WBACR AES umožňuje utajit před útočníkem hodnotu klíče IV. Lze použít IOC generované dle 6.1. Před další aktualizační iterací je třeba změnit kódovanou hodnotu K_R z výstupního na vstupního kódování. Pokud je IOC pro K_R použito, útočník nemá v paměti přístupnou jeho otevřenou hodnotu.

6.3. Provázanost IOC

Pro zajištění čerstvosti komunikace je klíčová ochrana hodnoty K_R proti podvržení. Z tohoto důvodu je přítomna pouze v chráněné podobě s aplikovaným IOC. Hodnota K_R je využívána jako argument funkce XOR a její použití i v kódované podobě lze provést při vhodně generovaným IOC základních bloků využívajících WBACR AES (5.2, 6.2) dle 6.1. Hodnota K_R se tak nikdy neobjeví v paměti softwarového agenta v otevřené podobě. Provázanost korespondujících IOC je zachycena na obrázku 6.3. Generování IOC kompatibilního s více jak jednou operací XOR příliš omezuje počet možných různých kódování a zvyšuje útočníkovu šanci na odhalení předpisu použitého kódování. Změnu kódování z IOC_1 na IOC_2 lze provést pomocí předpočtené tabulky pro operaci identity se vstupním kódováním IC_1 a výstupním kódováním OC_2. V případě n-bitového IOC se jedná o tabulku velikosti $2^n * n$ bitů.

Alternativou k změně kódování z důvodu použitelnosti pro operaci XOR je použití předpočtených tabulek pro tuto operaci s odpovídajícím IOC. V případě dvou n-bitových argumentů se jedná o tabulku velikosti $2^{2n} * n$ bitů. Předpočtené tabulky umožňují nahradit XOR libovolnou jinou



Obrázek 4: *Provázanost vstupních a výstupních kódování základních bloků. E1, E2 a E3 označují šifrovací část WBACR AES tabulek pro různé hodnoty klíče. RT(x,y) značí tabulku pro změnu kódování z IOCx na IOCy.* P_1 *označuje první blok otevřeného textu.* C_1 *značí první blok zašifrovaného textu.*

„blokovatelnou“ funkcí, pokud by byla vhodnější například z hlediska bezpečnosti.

7. Možnosti využití

Prosazení DRM u koncového uživatele – vlastností ochranného rozhraní, které zajišťují důvěrnost kódu a dat chráněného algoritmu, integritu výkonu a možnost bezpečného řízení prostřednictvím XML licencí, lze využít jako základ DRM architektury ve výpočetním prostředí uživatele. XML parser využívaný pro řízení bezpečnostní proxy je dostupný i pro zpracování příkazů aktualizaci hodnot chráněných algoritmů.

Kontrola využití podpisového klíče umístěného na čipového kartě - pro ochranu před viry, trojskými koni, nebo i samotným uživatelem lze rozšířit ochranu podpisového klíče. I po zadání PINu uživatelem bude moci pouze oprávněný softwarový agent zasílat data určená k podpisu.

Důvěrnost algoritmu/dat – lze použít při potřebě utajit data používaná algoritmem, například hodnot šifrovacích a podpisových klíčů nebo uživatelových privátních informací. Použití kryptografické čipové karty ztěžuje reverzní inženýrství prováděné útočníkem se záměrem odhalit funkčnost poskytovanou algoritmem nebo extrahovat použité návrhové myšlenky.

Omezující podmínkou bránící nasazení ochranného rozhraní může být relativně malá rychlost zpracování požadavku a zaslání odpovědi. Snížení doby odezvy lze dosáhnout použitím výkonnějších čipových karet zrychlující chráněný algoritmus a volbou čipové karty s rychlým hardwarovým akcelerátorem pro algoritmus AES. Dalšího zrychlení lze dosáhnout přechodem na komunikačního rozhraní s vyšší propustností a větší povolenou délkou jednoho příkazu, než poskytují standardní APDU příkazy. Omezující podmínkou je i nutnost fyzické distribuce čipové karty. Díky platformové nezávislosti rozhraní JavaCard lze využít čipových karet distribuovaných za jiným účelem, pokud mají požadované funkční a bezpečnostní vlastnosti. Příkladem mohou být kryptografických čipové karty používané pro digitální podpis nebo SIM karty mobilních telefonů.

8. Závěr

Příspěvek popsal systém pro ochranu vybraných částí kódu s využitím kryptografické čipové karty s podporou JavaCard. Ukazuje, že současné čipové karty s použitelnou pamětí cca 16kB lze využít pro implementaci systému pro řízené využívání kódu více softwarovými agenty, zahrnujícím XML parser pro vzdálenou správu prostřednictvím příkazů ve formátu XML. Dále je popsána implementace autentizačního a transportního protokolu navrženého s využitím principů mobilní kryptografie, který by měl zajišťovat autentizaci, důvěrnost a čerstvost i v případě, že se jedna strana nachází ve výpočetním prostředí kontrolovaném útočníkem. Pro použití v tomto protokolu byl navržen mechanismus generování vstupního a výstupního kódování pro WBACR AES tak, aby ho bylo možno prakticky využít pro šifrovací režim CBC. Dále byla navržena metoda aktualizace a využití klíče relace K_R používaného pro zajištění čerstvosti tak, aby byla chráněna jeho otevřená podoba.

Návrh a implementace ochranného rozhraní byl proveden P. Švendou pod vedením V. Matyáše v rámci diplomové práce s názvem Digital Rights Managment [Sv04]. Oproti původnímu textu je rozšířen návrh využití vstupního a výstupního kódování WBACR AES. Na [Sv04] je dostupná plná verze textu diplomové práce i zdrojové kódy pod GPL licencí.

Reference

[AES00] NIST: Advanced Encryption Standard AES, 2000. Dokument dostupný na URL `http://www.nist.com/aes/` (srpen 2003)

[An03] Anderson, R.: ‘Trusted Computing’ FAQ version 1.1. Dokument dostupný na URL `http://www.cl.cam.ac.uk/~rja14/tcpa-faq.html` (srpen 2004)

[Br02] Brandt, S.: Create a quick-and-dirty XML parser, JavaWorld, 2002. Dokument dostupný na URL `http://www.javaworld.com/javatips/jw-javatip128_p.html` (srpen 2004)

[Ce02] Červeň, P.: Cracking a jak se proti němu bránit. Praha, ComputerPress, 2002, ISBN 80-7226-382-X.

[CEJO02] Chow, S., Eisen, P., Johnson, H., van Oorschot, P.C.: White-Box Cryptography and an AES implementation. Cloakware Corporation, 2002. Dokument dostupný na URL http://web.archive.org/web/20040205092333/http://206.191.60.52/resources/pdf/SAC2002-CW.pdf (srpen 2004)

[CGJZ01] Chow, S., Gu, Y., Johnson, H., Zakharov, V. A.: An Aproach to the Obsfucation of Control-Flow of Sequential Computer Programs. Springer LNCS 2200, Berlin, 2001, s. 144-155.

[CTL97] Collberg, Ch., Thomborson, C. Low, D.: A Taxonomy Of Obsfuscating Transformations. New Zeland, University Of Aucland, 1997. Dokument dostupný na URL http://www.cs.arizona.edu/∼collberg/Research/Publications/CollbergThomborsonLow97a/A4.pdf (srpen 2004)

[CTL98] Collberg, Ch., Thombson, C., Low, D.: Breaking Abstraction and Unstructuring Data Structures. University Of Aucland, New Zeland, 1998. Dokument dostupný na URL http://www.cs.arizona.edu/∼collberg/Research/Publications/CollbergThomborsonLow97d/A4.ps.gz (srpen 2004)

[DhFe01] Dhem, J-F., Feyt, N.: Present and Future Smart Cards. Gemplus, France, 2001. Dokument dostupný na URL http://www.itu.dk/courses/DSK/E2002/smart2.pdf (srpen 2004)

[HMST01] Horne, B., Matheson, L., Sheehan, C., Tarjan, R.: Dynamic Self-Checking Techniques for Improved Tamper Resistance. Springer LNCS 2320, Berlin, 2001, s. 141-159.

[Ho98] Hohl, F.: Time Limited Blackbox Security: Protecting Agents From Malicious Hosts. Springer LNCS 1419, Berlin, 1998, s. 92-113.

[ChaAt01] Chang, H. Attalah, M.: Protecting Software Code by Guards. Springer LNCS 2320, Berlin, 2001, s. 160-175.

[GaCho01] Gaj, K., Chodowiec, P.: Fast Implementation and Fair Comparison of the Final Candidates for AES Using FPGA, Springer LNCS 2020, Berlin, 2001, s. 84-99.

[ISO9798-2] ISO/IEC 9798-2:1999 Information technology – Security techniques – Entity authentication – Part 2: Mechanisms using symmetric encipherment algorithms. Popis protokolu je také dostupný v [MOV].

[JC22API] Sun Microsystems, Inc., Palo Alto: JavaCard 2.2.1 Platform Specification. 2003. Dokument dostupný na URL http://www.java.sun.com/products/javacard/specs.html (srpen 2004)

[MOV] Menezes, A., van Oorschot, P., Vanstone, S.: Handbook of Applied Cryptography. CRC Press 1996/2001. Dostupné také na URL http://www.cacr.math.uwaterloo.ca/hac/ (srpen 2004).

[NCJ01] Nickerson, J., Chow, S., Johnson, H.: Tamper Resistant Software: Extending Trust In Hostile Environment. Říjen 2001. Dokument dostupný na URL http://web.archive.org/web/20040205080524/http://206.191.60.52/resources/pdf/ACM-01-Trust_in_Hostile_Environments.pdf (srpen 2004)

[NGSCB04] NGSCB, Dokument dostupný na URL http://www.microsoft.com/resources/ngscb/default.mspx (srpen 2004)

[OP02] Open Platform specification, Dokument dostupný na URL http://www.globalplatform.org/ (srpen 2004)

[RiSch98] Riordan, J., Scheider, B.: Environmental Key Generation Towards Clueless Agents. Springer LNCS 1419, Berlin, 1998, s. 15-24.

[RTM01] Rosenblatt, B. Trippe, B. Mooney, S.: Digital Rights Management: Bussines and Technology. Indianapolis, Hungry Minds, Inc., listopad 2001, ISBN 0-7645-4889-1.

[Ru01] Ruuskanen, J-P.: JAVACARD. University of Helsinky, 2001. Dokument dostupný na URL http://www.cs.helsinki.fi/u/ca

mpa/teaching/ruuskanen-final.pdf (srpen 2004)

[SaMo03] Satoh, A., Morioka, S.: Hardware-Focused Performance Comparison for the Standard Block Ciphers AES, Camellia, and Triple-DES. Springer LNCS 2851, Berlin, 2003, s. 252-266.

[SaTs98] Sander, T., Tschudin, Ch.: Protecting Agents From Malicious Hosts. Springer LNCS 1419, Berlin, 1998, s. 44-60

[Sv04] Švenda, P.: Digital Rights Managment. Diplomová práce. Fakulta informatiky, Masarykova universita, Brno, 2004. Dokument dostupný na URL http://www.fi.muni.cz/∼xsvenda/mst/index.html. (srpen 2004)

[TCG04] Trusted Computing Group, http://www.trustedcomputinggroup.org/home/ (srpen 2004)

[Za02] Zanero, S.: Smart Card Content Security. Dipartimento di Elettronica e Informazione, 2002. Dokument dostupný na URL http://www.elet.polimi.it/upload/zanero/papers/scsecurity.pdf (srpen 2004)

[Ze02] Zemánek, J.: Cracking bez tajemství. Praha, ComputerPress, 2002, ISBN 80-7226-703-5.