MASARYKOVA UNIVERSITA
FAKULTA INFORMATIKY

Digital Rights Managment

DIPLOMOVÁ PRÁCE

Petr Švenda

Brno 2004

Prohlášení

Prohlašuji, že tato práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování použil nebo ze kterých jsem čerpal, jsou řádně citovány s uvedením plného odkazu na příslušný zdroj.

Ve Křtinách, dne 17. května 2004 ……

Petr Švenda

Poděkování

Rád bych poděkoval především svému vedoucímu během této diplomové práce, panu Vaškovi Matyášovi, za pravidelné a podrobné čtení konceptů, kritiku, připomínky a cenné rady. To vše i přes nepříznivou geografickou vzdálenost. Dále bych rád poděkoval Lukáši Rychnovskému a Ondřeji Ševečkovi za plodné diskuse a přítelkyni Lindě za trpělivost.

Shrnutí

Práce je rozdělena do čtyř logických částí. V první části jsou rozebrány základní charakteristiky a problémy výpočetních prostředí, vznikajících při použití pro potřeby Digital Rights Managment (DRM). Druhá část mapuje techniky použitelné pro ochranu autonomních softwarových agentů s důrazem na aktuální pokročilé metody. V třetí části je navržen autentizační a transportní protokol umožňující bezpečnou komunikaci mezi stranami, z nichž jedna je umístěna v prostředí pod kontrolou útočníka. Tento scénář odpovídá výpočetnímu prostředí, které poskytuje ochranu pouze vybraným operacím a vyžaduje komunikaci mezi chráněnou a nechráněnou částí. Čtvrtá část je věnována popisu praktické programové implementace ochranného rozhranní, určeného pro zvýšení bezpečnosti softwarových agentů vykonávaných ve výpočetním prostředí běžné PC platformy. Zvýšení bezpečnosti je založeno na využití kryptografické čipové karty s podporou rozhranní JavaCard. Ochranné rozhranní poskytuje základ jednoduché DRM architektuře.

Klíčová slova

autentizační a transportní protokol, Digital Rights Managment, JavaCard, kryptografická čipová karta, mobilní kryptografie, NGSCB, ochrana spustitelného kódu, obfuskace, softwarový agent, Trusted Computing Group, White-Box Attack Resistant AES, XML.

Obsah

1 Úvod

S rozvojem digitálních technologií vzniklo i nové prostředí umožňující rozvoj nových obchodních modelů. Digitální zpracování poskytuje možnost tvorby identických kopií s minimálními náklady a snadnost distribuce bez ohledu na geografickou vzdálenost. Přináší možnosti výrazného snížení nákladů na uvedení a distribuci produktů, přesnost statistik jejich využívání a velké množství nových strategií nabízení produktů. Zároveň významně zasahuje časem prověřené a vybalancované vztahy a postupy mezi zúčastněnými stranami podílejícími se na využívání autorských prací, platné pro svět fyzických objektů. Přesun objektů z fyzického do digitálního světa sebou přináší problémy kontroly jejich manipulace tak, aby byly zachovány zájmy autora, nakladatele, distributora, koncového uživatele a zúčastněných. Efektivní správa práv k digitálním objektům (Digital Rights Managment, dále DRM) by měla tento cíl plnit.

Vzhledem k rozsahu celé oblasti DRM zahrnující legislativní aspekty, obchodní modely, způsoby distribuce, ustálené vzorce chování (např. „fair use“), problémy stanovení identity klienta (PC vs. uživatel), zachování soukromí a v neposlední řadě technické aspekty zajištění fungování celého systému se tato práce věnuje pouze vybraným oblastem. Jedná se především o technické prostředky zajištění ochrany autonomní aplikace (dále softwarový agent) odpovědné za využití digitálních dat v souladu s definovanými právy, vykonávané ve výpočetním prostředí uživatele.

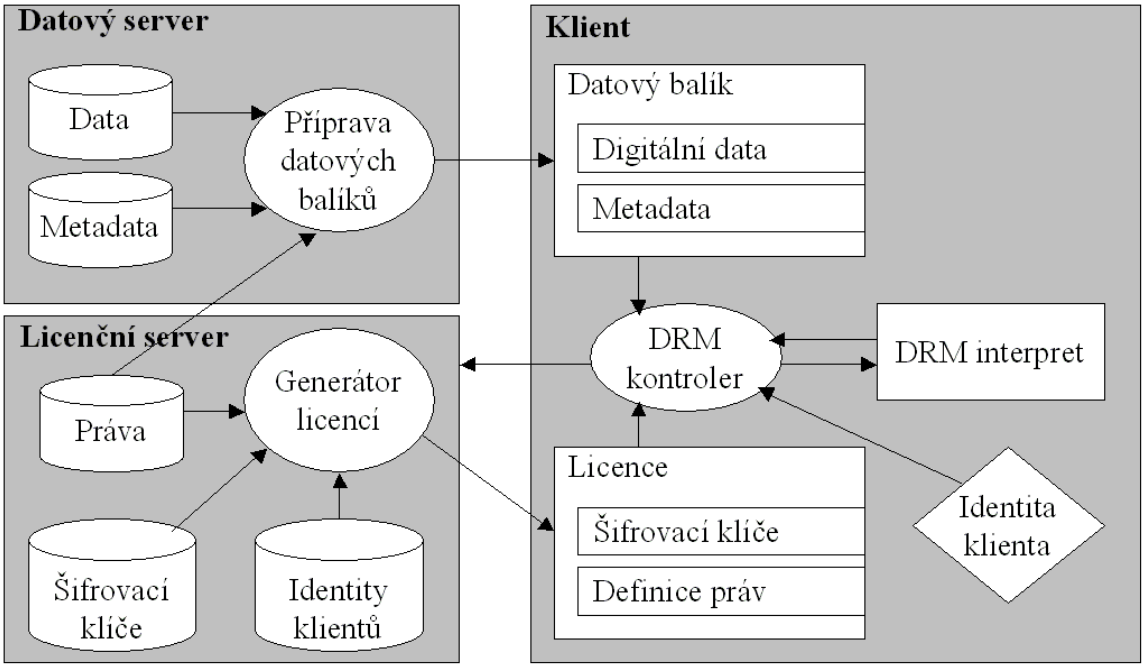
V kapitole 2 je probrána referenční architektura DRM, uveden příklad protokolů pro poskytování digitálních dat a uveden přehled základních typů práv použitelných pro digitální objekty. Kapitola 3 zavádí základní pojmy týkající se ochrany softwarového agenta a poskytuje přehled možných cílů útočníka při jeho napadení. V kapitole 4 je rozděleno prostředí klienta v závislosti míry využití ochranných hardwarových prostředků, které by měli poskytovat větší míru bezpečnosti než čistě softwarová výpočetní platforma. V zavedeném rozdělení jsou rozebrány problémy při použití pro potřeby referenční DRM architektury. Kapitola 5 se věnuje krátce běžným metodám ochrany softwarových agentů bez využití ochranných hardwarových prostředků. Kapitola 6 probírá vybrané pokročilé techniky ochrany softwarových agentů. Uvedené techniky jsou založeny na hlubších teoretických principech nebo využívají ve zvýšené míře možností ochranných hardwarových prostředků. Kapitola 7 se zabývá návrhem vhodného autentizačního a transportního protokolu pro komunikující strany, z nichž jedna je umístěna v potencionálně nebezpečném prostředí. Připravuje tak základ možnosti řízeného využívání chráněných prostředků pro případ, kdy se oprávněný softwarový agent nachází v prostředí pod kontrolou útočníka. V kapitole 8 je popsáno provedení praktické implementace obecně použitelného a snadno začlenitelného ochranného rozhranní. Ochranné rozhranní využívá pro ochranu citlivých částí kódu a dat spolupráce softwarového agenta s programovatelnou kryptografickou čipovou kartou.

2 Model distribuce a poskytování digitálních dat

2.1 Referenční DRM architektura

Pro prezentaci mechanismů a zúčastněných entit při poskytování a zpřístupnění digitálních dat byl vybrán referenční DRM architektura uvedená v [RTM01]. Zavádí základní rozdělení vystupujících entit na poskytovatele, který spravuje Datový server a Licenční server a klienta, který poskytnuté data využívá. Cílem vzájemné komunikace je umožnit klientovi přístup k digitálním datům a zároveň zajistit jejich použití v souladu s podmínkami stanovenými poskytovatelem. Způsob a frekvence komunikace se liší v závislosti na stupni konektivity klienta a poskytovatele („on-line/off-line“ systémy), použitých technologických prostředků a zvoleném obchodním modelu.

Konkrétním implementací prezentovaného modelu je například Windows Media Rights Manager, určený pro zpřístupnění hudby a videa na platformě MS Windows [MSRM03].



Obr. 2.1: Referenční DRM architektura

2.1.1 Datový server (Content Server)

Datový server je umístěn v bezpečné prostředí u poskytovatele digitálních dat. Obsahuje digitální data, příslušná metadata a seznam nabízených služeb. Disponuje funkčností potřebnou pro přípravu zabezpečeného datového balíku do formy použitelné ve zvoleném DRM systému. Repositář digitálních dat (Content Repository) bývá reprezentován

vhodným souborovým nebo databázovým serverem, obdobným způsobem jsou uchovávána i metadata k uloženým objektům.

Příprava zabezpečeného datového balíku se typicky skládá z generování jeho unikátní identifikace, vložení relevantních dat a metadat a přiřazení licence s popisem práv pro manipulaci s balíkem. Licence je vytvořena ve spolupráci s Licenčním serverem a obsahuje identifikace nebo zabezpečené hodnoty šifrovacích klíčů použitých pro zajištění integrity a utajení digitálních dat distribuovaných v datovém balíku.

2.1.2 Licenční server (Licence server)

Licenční server je také umístěn v bezpečném prostředí u poskytovatele. Obsahuje kopie licencí vytvořených během přípravy datových balíků a hodnoty použitých šifrovacích klíčů. Součástí licence je typicky identifikace datového balíku, identifikace klienta (uživatele nebo zařízení), popis přidělených práv a identifikace nebo zabezpečené hodnoty použitých šifrovacích klíčů. Práva definovaná licencí mohou být zachycena vhodným specifikačním jazykem, například XrML [XrML2].

2.1.3 Prostředí klienta

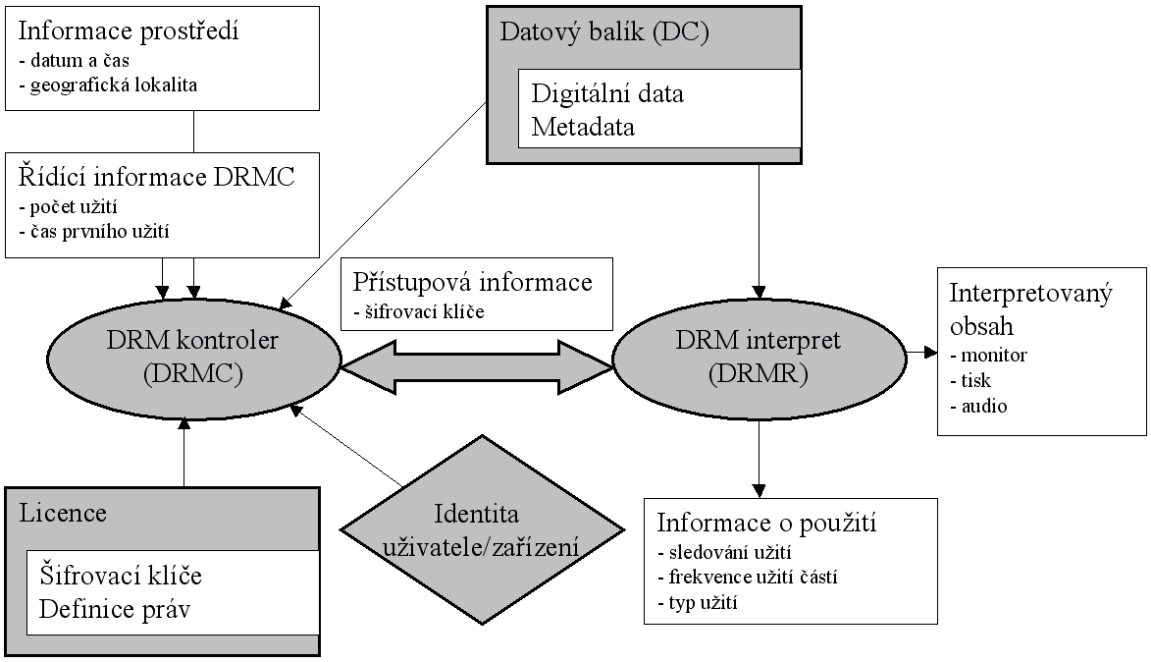
Klientem je označována jednoznačně identifikovatelná entita, které jsou poskytovány služby nebo digitální data spolu s licencí definující oprávnění využívat poskytnuté prostředky. Prostředím klienta je označován soubor hardwarových zařízení, softwarových aplikací a relevantních informací sloužících k využití služby nebo poskytnutých digitálních dat. Z hlediska poskytovatele by prostředí klienta mělo poskytovat dostatečnou míru záruky, že využití služby nebo digitálních dat bude probíhat v souladu s přidělenými právy.

Prostředí klienta obsahuje typicky softwarovou aplikaci odpovědnou za prezentaci digitálních dat v požadované formě uživateli (DRM interpret, dále DRMR) a softwarovou aplikaci odpovědnou za vyhodnocení oprávněnosti požadavku na využití zabezpečeného datového balíku (DRM kontroler, dále DRMC). Příkladem prezentace digitálních dat prostřednictvím DRMR je zobrazení na obrazovku, tisk nebo hudební reprodukce. DRMC provádí kontrolu požadovaných operací nad zabezpečeným datovým balíkem vzhledem k přiřazené licenci a poskytuje informace nutné pro zpřístupnění digitálních dat aplikací DRMR.

Proces zpřístupnění digitálních dat je znázorněn na obrázku 2.2. DRMR slouží jako vstupní bod pro zpřístupnění zabezpečeného datového balíku. Po obdržení požadavku na prezentaci dat nejprve kontaktuje DRMC a požádá o zpřístupnění digitálních dat pro požadovanou operaci. DRMC ověření identitu klienta, nalezne licenci příslušnou zabezpečenému datovému balíku a vyhodnotí oprávněnost požadované operace. DRMC poskytuje v kladném případě zpět přístupové informace (šifrovací klíče) nebo přímo dešifrovaná data aplikaci DRMR. Licence může být získána z lokální kopie u „off-line“ systémů, nebo je vyžádána přímo z Licenčního serveru. Volitelně může být prováděn záznam o proběhlých operacích pro potřebu zvoleného obchodního modelu.

DRMC může být nezávislá na DRMR, nebo naopak její integrální součástí. V případě nezávislosti DRMC a DRMR je nutné provést (vzájemnou) autentizaci s cílem zabránit zpřístupnění digitálních dat útočníkovi nebo naopak prezentaci podvržených dat.

DRMR i DRMC aplikace musí poskytovat dostatečnou úroveň bezpečnosti proti útokům zaměřeným na manipulaci jejich chování útočníkem.



Obr. 2.2: Zpřístupnění zabezpečeného digitálního balíku

DRMR a DRMC mohou být dedikované softwarové aplikace s integrovanou podporou pro požadované chování z hlediska poskytovatele, nebo se může jednat o všeobecně rozšířené aplikace s dodatečně upraveným chováním, například formou zásuvných „plug-in“ modulů. Výhodou dedikovaných aplikací bývá snadnější implementace požadovaného chování a zajištění ochrany vůči útočníkovi. Nevýhodou je nutnost distribuce celé softwarové aplikace k uživateli a nutnost uživatelova návyku na nové prostředí. Obecně lze říci, že dedikovaná forma bývá výhodnější z hlediska poskytovatele, uživatel preferuje použití rozšiřujících modulů.

2.2 Protokoly poskytování digitálních dat

Zavedená referenční DRM architektura používá vhodný protokol pro poskytování digitálních dat v souladu s potřebami zvoleného obchodního modelu a technickými parametry komunikujících stran. Pro ilustraci jsou zde uvedeny dva různé protokoly a krátce rozebrána vhodnost jejich použití v různých scénářích.

2.2.1 Generická příprava datových balíků (Protokol 1)

Zabezpečený datový balík je připraven předem, bez znalosti prostředí klienta a identity cílového uživatele. Distribuce probíhá vhodným distribučním kanálem ke všem potencionálním uživatelům. Při pokusu o zpřístupnění zabezpečených digitálních dat dochází k určení identity uživatele a ve spolupráci s Licenčním serverem k vygenerování příslušné licence. Výhodou je nižší režie přípravy a distribuce zabezpečeného datového balíku. Distribuce může být provedena hromadně s dalšími produkty, například formou

CDROM v časopise. Při pokusu o zpřístupnění dochází pouze ke přenosu licence, nikoli celého datového balíku. Nevýhodou je obtížnější zajištění bezpečnosti datového balíku, neboť ho lze použít principiálně u kteréhokoli uživatele, například vyzrazením hodnoty jednotného šifrovacího klíče. Vhodnou strategií je distribuce velké, ale nefunkční části datového balíku. Při pokusu o zpřístupnění je distribuována licence a relativně malá chybějící část. Chybějící část může být vytvořena na základě konkrétní identity uživatele anebo prostředí klienta.

1. Klient K přijme zabezpečený datový balík DC.
2. Pokusí se použít DC prostřednictvím DRMR operací vyžadujícím práva P. Dochází k aktivaci DRMC.
   2.1. Existuje-li pro P platná licence, pokračuje krokem 9.
   2.2. Neexistuje-li pro P platná licence, pokračuje krokem 3.
3. DRMC zašle Licenčnímu serveru (LS) žádost o licenci L spolu s identifikací K a DC.
4. LS ověří identitu K proti databázi a použije identifikaci DC pro zjištění definovaných práv vztahujících se k DC.
5. LS provede porovnání zjištěných práv s právy požadovanými klientem
6. V případě potřeby je provedena dodatečná operace (finanční transakce, aktivace sledování použití – může záviset na definovaných právech)
7. LS vytvoří zabezpečenou licenci L skládající se z:
   - šifrovacího klíče příslušného k DC získaného z databáze šifrovacích klíčů
   - udělených práv P
   - identifikace klienta K
8. Licence L je zaslána zpět klientovi K.
9. DRMC na základě licence dešifruje obsah DC a poskytne ho bezpečně DRMR.
10. DRMR zobrazí/přehraje/vytiskne obsah uživateli.

2.2.2 Personalizovaná příprava datových balíků (Protokol 2)

Variantou generické přípravy zabezpečených datových balíků je protokol, ve kterém je zabezpečený datový balík vytvořen a distribuován až během žádosti o jeho využití uživatelem. Uživatel nejprve zažádá (prostřednictvím DRMR) o vytvoření zabezpečeného balíku a zároveň poskytne vybrané informace o své identitě a cílovém klientském prostředí. Datový balík tak může být na základě těchto informací personalizován. Další postup je již obdobný předchozímu protokolu. Výhodou je možnost použít rozdílné parametry ochrany pro zabezpečení datového balíku pro každého uživatele, například rozdílné šifrovací klíče nebo digitální vodoznak. Vhodné použití může vést ke zvýšení odolnosti ochrany proti útočníkovi, neboť může snižovat globální dopad úspěšného útoku a zvyšovat náklady na opakovaní útoku u jiného uživatele. Nevýhodou je nutnost přenosu celého balíku až při žádosti o jeho použití.

3 Ochrana softwarového agenta

3.1 Základní pojmy

V této části textu je uveden přehled základních pojmů z oblasti bezpečnosti výpočetního prostředí klienta relevantních vzhledem k zaměření této práce.

3.1.1 Bezpečné prostředí

Bezpečným prostředím rozumíme prostředí, které poskytuje vzhledem k hodnotě zapracovávaných informací dostatečnou záruku, že neumožní provádět útočníkovi akce narušující korektní výkon softwarového agenta umístěného v tomto prostředí. Narušením korektního výkonu je rozuměno úspěšné provedení některého z útoků uvedených v této kapitole. Konkrétní instance bezpečného prostředí nemusí zajišťovat ochranu proti všem typům útoků, postačuje ochrana proti útokům relevantním vzhledem k povaze zpracovávaných informací.

Stanovit úroveň bezpečnosti konkrétního prostředí vzhledem k úrovni předpokládaných útočníků je obtížné. Odhad poskytované bezpečnosti bude záviset na aktuálním stavu teoretických a technických poznatků spolu s odhadem vývoje v budoucnu. Odhad bezpečnosti může být snížen s objevem efektivní metody řešení problémů pokládaných za obtížné, využitím implementačních chyb nebo technickým pokrokem v hrubé výpočetní síle. Příkladem mohou být útoky na procesor se šifrovanou sběrnicí [Ku98], útoky s velmi nízkými náklady na kryptografické čipové karty pomocí ozařování vybraných míst rentgenovým a viditelným zářením [AnKu02, SkAn02] nebo existence zařízení pro nalezení 56bitového klíče algoritmu DES prohledáním prostoru všech možných klíčů.

3.1.2 Uzavřený softwarový agent (BlackBox Security – BBS)

Softwarový agent má vlastnost BBS, pokud jeho kód ani data nemohou být smysluplně čtena nebo modifikována po neomezeně dlouhou dobu. K softwarovému agentovi s vlastností BBS lze přistupovat, jakoby se trvale nacházel v bezpečném prostředí.

Při splnění vlastnosti BBS může útočník ovlivňovat softwarového agenta pouze modifikací jeho vstupů a výstupů. Útokům vedeným proti komunikaci s dalšími softwarovými agenty lze často zabránit použitím standardních mechanismů pro zajištění důvěrnosti a integrity komunikace. Potřebné citlivé informace (šifrovací a podpisové klíče) lze díky splnění vlastností bezpečného prostředí umístit přímo do kódu softwarového agenta.

Pokud dochází ke generování instance softwarového agenta z obecně známé předlohy doplněné o konkrétní citlivé informace (šifrovací klíče), je třeba parametrizovat generování instancí. Výsledný prostor všech možných instancí musí být dostatečně velký pro zabránění slovníkového útoku, při kterém útočník generuje ve vlastní režii instance softwarových agentů a hledá korelaci mezi vstupy a výstupy s napadaným softwarovým agentem.

V současném době není znám žádný algoritmus, který by umožňoval zkonstruovat softwarového agenta s vlastností BBS dle obecné specifikace bez použití bezpečného hardware. Pro některé třídy úloh však lze použít techniku mobilní kryptografie [SaTs98].

3.1.3 Softwarový agent s časově omezeným intervalem uzavřenosti (Time-Limited BlackBox Security – TLBBS)

Softwarový agent má vlastnost TLBBS, pokud jeho kód ani data nemohou být smysluplně čtena nebo modifikována po dobu trvání předem stanoveného časového intervalu. Během tohoto intervalu lze k softwarovému agentovi s vlastností TLBBS přistupovat, jakoby se nacházel v bezpečném prostředí.

Softwarový agent s TLBBS vlastností oslabuje požadavek na splnění vlastnosti BBS z důvodu snazší dosažitelnosti její platnosti. Zároveň ale dochází k omezení množiny scénářů použití, u kterých je vhodné softwarového agenta použít na scénáře, ve kterých úspěšný útok po uplynutí doby stanoveného časového intervalu nezpůsobí relevantní škodu. Konstrukce softwarového agenta splňující podmínky TLBBS je v současné době obecně dosažitelná, například s použitím metod obfuskace.

Problematickou místem je určení očekávané délky časového intervalu. Výhodou je dosažení vlastnosti TLBBS použitím technik založených na problémech s dokazatelnou výpočetní složitostí. Pro některé scénáře může být vhodné použít principu „neinformovaných agentů“ [RiSch98], u kterých lze zajistit délku časového do výskytu specifické události, ne však již poté. Popis některých technik pro konstrukci softwarových agentů s vlastností TLBBS lze nalézt v kapitole 6.

3.1.4 Nechráněný softwarový agent

Softwarový agent je nechráněný, pokud jeho kód i data mohou být smysluplně čtena a modifikována s vynaložením triviálního úsilí. Nechráněný softwarový agent neobsahuje žádnou ochranu, jejíž odstranění by vyžadovalo netriviální úsilí či použití netriviálního množství výpočetních zdrojů. Jedná se o speciální případ vlastnosti TLBBS, u které je délka časového intervalu neúměrně krátká vzhledem k zamýšleném účelu.

3.1.5 Mentální model

Mentální model je soubor pravidel, konceptů a vědomostních struktur vytvářených interně člověkem při řešení určitého problému. Pokud je řešení tohoto problému algoritmizovatelné, lze vytvořený mentální model pomocí vhodného programovacího jazyka vyjádřit v podobě kódu (softwarového agenta), který daný problém řeší. Opačný postup, při kterém je z kódu vytvářen mentálního modelu problému, který je kódem řešen, se nazývá reverzní inženýrství. Cílem použití standardních programovacích vzorů, poznámek a dokumentace kódu je usnadnit zpětné vytvoření mentálního modelu. Cílem některých ochran typu obfuskace je naopak vytváření mentální modelu co nejvíce ztížit. Korektně vytvořený mentální model z předlohového kódu může být použit pro generování alternativní kódu, který se bude na stejných vstupních datech chovat identicky jako předlohový kód. Korektně vytvořený mentální model umožňuje provádět cílené modifikace kódu a používaných dat.

3.1.6 Statická inspekce softwarového agenta

Při použití statické inspekci dochází ke zkoumání softwarového agenta útočníkem bez nutnosti jeho spuštění. Cílem útočníka je vytvoření mentálního modelu prozkoumávaného kódu a zisk dat nesených softwarovým agentem. Vytvořený mentální

model může být využit pro cílenou modifikaci kódu a dat, například odstranění ochranných mechanismů nebo úpravu rozhodovacích podmínek. Pro statickou inspekci lze použít automatizované nástroje, které přiřazují sémantickou informaci prověřovanému kódu, například převodem do vyššího programovacího jazyku nebo zvýraznění volání systémových funkcí [IDA].

3.1.7 Dynamická inspekce softwarového agenta

Při použití dynamické inspekce dochází k prozkoumání softwarového agenta za běhu. Využívá se možnosti krokování jednotlivých operací, detekce práce s vybraným blokem paměti, detekce volání systémových funkcí, použití I/O zařízení nebo záznam frekvence využití částí kódu. Dynamická inspekce umožňuje zisk a modifikaci hodnot aktuálně zpracovávaných softwarovým agentem. Mezi standardní nástroje pro dynamickou inspekci patří „debuggery“ [SOFTICE] a monitory přístupů do souborového systému [RFMON]. Dynamická inspekce bývá často kombinována se statickou inspekcí z důvodu rychlejší lokalizace relevantní části kódu.

3.2 Cíle útoků

Jednotlivé body uvádějí základní přehled útočníka cílů při napadení softwarového agenta.

3.2.1 Zisk kódu

Cílem útoku je vytvoření mentálního modelu (části) softwarového agenta z jeho kódu. Získaný mentální model může být použit pro extrakci obsažených návrhových myšlenek a cílenou modifikaci kódu.

3.2.2 Zisk dat

Cílem útoku je nalezení polohy a zisk citlivých dat nesených, používaných nebo produkovaných softwarovým agentem. Znalost polohy a hodnoty citlivých dat může být útočníkem využita pro jejich cílenou modifikaci, nebo externí použití mimo softwarového agenta. Problém zisku dat je závažný především z důvodu obtížné detekce kompromitace dat v případě, že data mají dlouhodobou použitelnost, například hodnoty šifrovacích klíčů.

3.2.3 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 (Zisk dat).

3.2.4 Modifikace kódu nebo dat

Cílem útoku je záměrná modifikace kódu nebo dat softwarového agenta tak, aby výsledné chování odpovídalo potřebám útočníka. Modifikace kódu může být trvalá nebo pouze dočasná, která je vrácena do původní podoby po vykonání modifikované části. Dočasná modifikace se používá pro ztížení detekce ochrannými technikami softwarového agenta.

Speciálním případem je náhodná modifikace s cílem narušit výkon neřízeným způsobem nebo využít citlivosti kódu na korektnost zpracování.

3.2.5 Manipulace toku dat

Cílem útoku je ovlivnit výkon softwarového agenta modifikací vyhodnocování rozhodovacích podmínek. Dociluje se pomocí modifikace kódu nebo dat.

3.2.6 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.

3.2.7 Podvržení identity hostitele

Cílem útoku je klamání softwarového agenta ohledně identity prostředí hostitele, ve kterém je softwarový agent vykonáván. Podvržení identity může být provedeno modifikací kódu a dat části softwarového agenta odpovědného za ověření identity nebo podvržením hodnot používaných pro určení identity. Podvržení identity může být využito pro přístup k funkčnosti a informací určeným pro jiného hostitele.

3.2.8 Omezení činnosti agenta

Cílem útoku je omezení nebo úplné potlačení některých funkcí softwarového agenta ve snaze neumožnit provedení operací nevýhodných pro útočníka. Příkladem cíle útoku je využití zdroje, u kterého se předpokládá jen občasná dostupnost pro zisk externích dat (revokované klíče) nebo export dat (report provedených akcí).

3.2.9 Přístup a manipulace komunikace softwarového agenta

Cílem útoku je porušení důvěrnosti, autenticity nebo čerstvosti komunikace softwarového agenta s druhou stranou. Útok vede k zisku přenášených dat (citlivé informace, šifrovací klíče) nebo k nekorektnímu chování softwarového agenta následkem manipulace vyměňovaných zpráv v průběhu komunikace.

3.2.10 Podvržení návratových hodnot volání systémových funkcí

Cílem útoku je manipulace údajů získávaných softwarovým agentem o externích podmínkách prostředí. Útok je speciálním případem útoku výkonem v nekorektním prostředí, při kterém nedochází ke změně vlastností prostředí, ale pouze k modifikaci hodnoty popisující příslušnou vlastnost. Příkladem podvržených hodnot je systémový čas, geografická lokace nebo identita přihlášeného uživatele.

4 Bezpečnost výpočetního prostředí klienta

Bezpečnost výpočetního prostředí klienta lze rozdělit vzhledem k rozsahu využití ochrany prostřednictvím zabezpečených hardwarových zařízení. Tato zařízení by měla poskytovat vyšší úroveň zabezpečení v porovnání s běžnou softwarovou platformou typu PC. U každé skupiny jsou uvedena problémová místa při použití prostředí pro potřeby referenční DRM architektury z kapitoly 2.1.

4.1 Nechráněné výpočetní prostředí

Prostředí klienta neposkytuje implicitně žádnou nebo pouze nedostatečnou ochranu procesů a dat. Výkon softwarových agentů v tomto prostředí může být krokován, jejich paměť čtena a modifikována. Veškeré informace poskytované prostředím mohou být modifikovány. Data ukládaná a načítaná z externích médií mohou být modifikována nebo nahrazena staršími verzemi. Tento scénář nastává při distribuci zabezpečených datových balíků a souvisejících softwarových agentů do prostředí, které je pod plnou kontrolou útočníka s administrátorskými právy (pokud jsou práva vůbec zavedena). Útočník má možnost používat prostředky pro kompletní kontrolu systému a je schopen provádět statickou i dynamickou inspekci softwarového agenta. V současné době prostředí klienta tohoto typu výrazně převažují. Výhodou je absence nákladů a omezení plynoucích z nutnosti fyzické distribuce zabezpečeného hardwarového zařízení. Není známa obecná metoda, který by umožňovala zabezpečit softwarového agenta tak, aby v tomto prostředí splňovala vlastnost BBS.

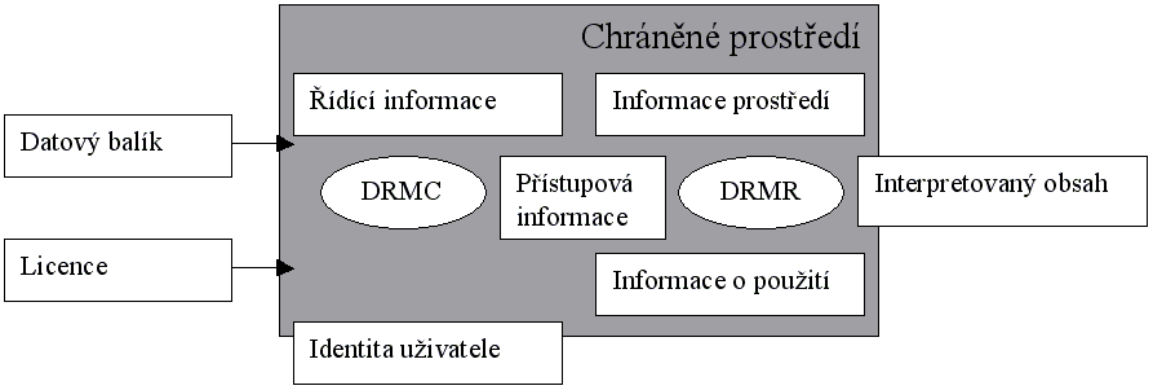
Při použití nechráněného výpočetního prostředí pro potřeby DRM je třeba zajistit ochranu procesů a dat znázorněných na obrázku 2.2. Konkrétně se jedná o splnění následujících požadavků:

- Zajištění integrity a autenticity licence.
- Bezpečný zisk a ověření identity uživatele/zařízení.
- Bezpečný zisk informací prostředí.
- Bezpečný zisk, aktualizace, čerstvost a autenticita řídících informací.
- Bezpečné ověření podmínek licence.
- Bezpečné rozhodnutí o poskytnutí přístupových informací.
- Autentizace mezi DRMR a DRMC.
- Bezpečný přenos přístupových informací mezi DRMC a DRMR.
- Ukrytí hodnot aktuálně používaných šifrovacích klíčů u DRMC a DRMR.
- Bezpečné uchování nechráněné podoby digitálního dat během interpretace.
- Interpretace digitálních dat v souladu s definovanými právy.
- Bezpečný zápis o použití na čerstvé a autentizované médium.

4.2 Hardwarově chráněné výpočetní prostředí

Prostředí klienta poskytuje kompletní ochranu procesů a jejich dat. Stupeň ochrany je závislý na bezpečnosti použitého hardwarového zařízení a měl by odpovídat hodnotě zapracovávaných digitálních dat. Typ hardwarového zařízení se může pohybovat v rozmezí od jednoúčelových zařízení poskytujících pouze omezené přístupové rozhranní bez zabezpečení proti manipulaci na hardwarové úrovni až po vysokou softwarovou a hardwarovou bezpečnost kryptografických čipových karet. Hardwarově chráněné výpočetní prostředí může mít obtížnou rozšiřitelnost a omezený výpočetní výkon. Proces použití pro potřeby DRM je znázorněn na obrázku 4.1.

Problémová místa při použití pro potřeby DRM, uvedená pro nechráněné výpočetní prostředí lze řešit prostřednictvím standardních kryptografických technik, především použitím šifrování a digitálního podpisu. Problémem zůstává zachycení výstupních dat (interpretovaný obsah) útočníkem. Výstupní data by měly zachovat jistou formu „zabezpečení“. Může být dosaženo snížením kvality výstupu oproti originálním digitálním datům nebo prezentace způsobem, který útočníkovi ztěžuje zachycení výstupních dat. Příkladem je převod digitální zvukové nahrávky na analogového výstup, zobrazení vektorového obrazu jako rastrového, s omezenou přesností nebo zobrazení na chráněném vestavěném displeji.



Obr. 4.1: Hardwarově chráněné výpočetní prostředí.

4.3 Výpočetní prostředí s pomocnou hardwarovou ochranou

Výpočetní prostředí uvedené v této skupině využívají v různé míře ochranu pomocí hardwarového zařízení. Některé procesy a data tak mohou být umístěny hardwarově chráněném výpočetním prostředí. Z důvodů výkonnostního a paměťového omezení ochranných hardwarových zařízení je však část požadovaných operací prováděna v nechráněném prostředí pod potenciální kontrolou útočníka. Vzhledem k možnostem použitého hardwarového zařízení pokrýt operace referenční DRM architektury lze míru hardwarové podpory rozdělit do tří úrovní, nacházející se mezi nechráněným a plně chráněným výpočetním prostředím.

4.3.1 Hardwarová ochrana DRMC a DRMR

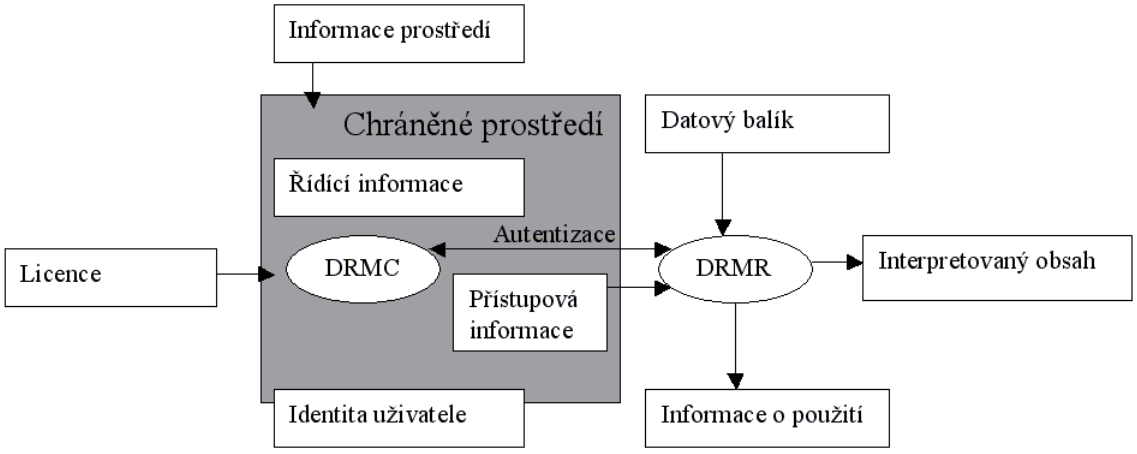
Použité hardwarové zařízení poskytuje dostatečný výkon pro provádění operací DRMC a DRMR i dostatečnou paměťovou kapacitu pro dlouhodobé uložení citlivých informací. Oproti hardwarově chráněnému prostředí jsou informace o prostředí klienta získávány z potencionálně nebezpečného zdroje, například voláním systémových funkcí. Interpretovaný obsah je poskytován nechráněnému prostředí klienta, například zobrazovacímu subsystému nebo tiskárně. Vhodným hardwarovým zařízením pro tento scénář může být dostatečně výkonná kryptografická čipová karta nebo dedikované hardwarové zařízení připojitelné k PC přes PCI sběrnici.

Informace prostředí, které mohou být útočníkem manipulovány, je vhodné získat několika nezávislými metodami. Užití těchto metod by mělo být pro útočníka obtížně rozpoznatelné. Interpretovaný obsah poskytovaný nechráněnému prostředí klienta by měl mít sníženou hodnotu oproti originální předloze. Integritu a utajení dat ukládaných mimo chráněné prostředí lze zajistit jejich šifrováním. Pravidelnou aktualizací hodnoty použitého šifrovacího klíče lze bránit nahrazení šifrovaných dat starší verzí. Identitu uživatele lze určovat například na základě vhodných biometrických metod, prostřednictvím vestavěného čtecího zařízení

4.3.2 Hardwarová ochrana pouze pro DRMC

Použité hardwarové zařízení poskytuje dostatečný výkon pro provádění operací DRMC. Neposkytuje dostatečný výkon nebo jiné parametry pro operace DRMR, především interpretaci zpřístupněných digitálních dat. Typickým příkladem je kryptografická čipová karta připojitelná k PC. Tento scénář je zachycen na obrázku 4.2 a odpovídá situaci řešené projektem popsaným v kapitole 8.

K problémům uvedeným pro prostředí s hardwarovou podporou pro DRMC i DRMR přibývá nutnost vzájemné autentizace a vytvoření bezpečného komunikačního kanálu mezi DRMR a DRMC. Vhodnou dodatečnou ochranou DRMR je třeba zajistit, aby interpretace zpřístupněných digitálních dat probíhala v souladu s pravidly určenými v licenci. Ochrana DRMR musí zajišťovat integritu kódu DRMR, utajení informací používaných pro autentizaci a ochranu zpřístupněných digitálních dat.



Obr. 4.2: Hardwarová ochrana pro DRMC.

4.3.3 Hardwarová podpora pouze pro uchování citlivých informací

Použité hardwarové zařízení poskytuje pouze možnost bezpečného uložení vybraných citlivých informací, například hodnoty klíčů nebo chráněný čítač a specifickou funkčnost typu šifrování a snížení čítače. Hardwarové zařízení nedisponuje dostatečnou paměťovou anebo výkonnostní kapacitou pro provedení celého procesu DRMC, mohou však být procesem DRMC i DRMR částečně využívány.

K problémům z výše uvedených scénářů přibývá nutnost ochrany procesu DRMC. Dodatečná ochrana DRMC musí zajišťovat korektní ověření autenticity a vyhodnocení licence a utajení použitých citlivých dat. Využití hardwarového zařízení pro uložení citlivých informací vyžaduje autentizaci takovým způsobem, aby hardwarové zařízení neposkytovala citlivé informace útočníkovi a zároveň útočník neměl možnost podvrhnout informace pro DRMC (resp. DRMR).

5 Klasické techniky ochrany

Techniky uvedené v této kapitole budou nazývány klasické, oproti technikám pokročilým uvedeným v následujícím kapitole 6. Toto dělení není přesné ani trvalé. Mezi klasické techniky ochran jsou zařazeny techniky, které jsou všeobecně známé a svou bezpečnost staví převážně na utajení své implementace. Techniky ochrany zde uvedené jsou čerpány především z [Ce02, Ze02]. Podrobnější popis některých technik lze nalézt v příloze C.

Základní pravidla ochrany softwarových agentů:

- Nepoužívat smysluplné názvy funkcí obsahující citlivé části kódu a dat. Cílem je ztížit nalezení jejich polohy.
- Neinformovat uživatele ihned o (ochranou) zjištěné chybě. Upozornit se zpožděním nebo způsobit nekorektní výkon softwarového agenta. Ukončení softwarového agenta následkem ochranou zjištěné chyby lze provést sérií instrukcí vyvolávajících výjimku. Cílem je ztížit lokalizaci ochranné funkce.
- Používat kontrolu integrity souborů, nejlépe po malých částech a na různých místech.
- Před ověřením registračního čísla vložit zdržení, případně vyžádat restart. Cílem je zamezit útoku hrubou silou pro zjištění registračního čísla.
- Používat silných šifrovacích algoritmů.
- Používat více kontrolních ochran a náhodně vybírat jejich použití.
- Používat další testy pro kontrolu korektnosti registračního čísla náhodně a později při běhu softwarového agenta.
- Používat sebemodifikující kód využitím metod komprese, šifrování, skoků doprostřed instrukcí a opravných kódů.
- Umísťovat důležité informace na neobvyklá místa a redundantně.
- Nepoužívat výstražné řetězce v otevřené podobě. Cílem je ztížit lokalizaci jejich polohy v kódu softwarového agenta. Řetězce lze generovat za běhu, šifrovat, nebo komprimovat.
- Nepoužívat pro ověření korektnosti hodnoty porovnávacích funkcí, ověřovanou hodnotu raději použít k operaci nutné pro pokračování, například k dešifrování kódu.
- Nespoléhat při zjišťování času (a dalších systémových informací) na běžné funkce poskytované systémem. Získávat i z jiných dostupných zdrojů, například čas modifikace souborů užívaných systémem.
- Decentralizovat časově i prostorově použití ochrany. Používat více kontrolních funkcí, testování provádět náhodně. Důležité podmínky netestovat zároveň.
- V případě, že daná funkce nemá být ve zkušební verzi softwarového agenta přístupná, je vhodné ji úplně odstranit, případně šifrovat klíčem zasílaným spolu s registrací plné verze.
- Zanést umělou toleranci do kontrol zadávaných registračních údajů. Nekontroluje se úplná shoda registračních údajů s očekávanými, kontroluje se pouze jejich část nebo se provádějí jen některé testy. Nepoužité části lze otestovat pozdějšími testy za běhu nebo použít pro dešifrování části kódu softwarového agenta.

6 Pokročilé techniky ochrany

Klasické techniky ochran uvedené v kapitole 5 poskytují pouze časově omezenou bezpečnost, založenou především na utajení implementace zvolené ochrany. V této kapitole budou probrány techniky, které se zakládají svou bezpečnost na využití takových podpůrných prostředků, u kterých znalost implementace útočníkem nemá žádný nebo omezený vliv na bezpečnost celé ochrany. Techniky uvedené v této kapitole budou nazývány pokročilé, i když stanovit přesný rozdíl oproti klasickým technikám je obtížné. Často se jedná o rozvinutí klasické ochrany, případně je pro praktické použití vhodné kombinovat pokročilou a klasickou techniku. Vhodným příkladem je kombinace techniky sebekontrolujícího kódu, pro jejíž odstranění nelze použít statickou inspekci kódu spolu s technikou detekce debuggerů, která ztěžuje inspekci dynamickou.

6.1 Poskytování služby modelem klient-server

Ochrana je založena na přenosu chráněné části softwarového agenta z potenciálně nebezpečného prostředí hostitele do bezpečného prostředí chráněného serveru, který poskytuje službu provedení chráněné části. V případě, že softwarový agent požaduje provedení operace obsažené chráněné části, zašle serveru žádost o využití o provedení operace spolu se vstupními hodnotami. Pokud server vyhodnotí žádost jako oprávněnou, provede operaci chráněné části nad vstupními hodnotami a softwarovému agentovi zpět zasílá výsledek operace. Server může být realizován formou aplikace běžící na vzdáleném zabezpečeném počítači, aplikace vykonávané v kryptografické čipové kartě, nebo softwarovým modulem, který je lépe chráněn než zbylá část softwarového agenta.

Protokol interakce mezi klientem a serverem může pomocí autentizace omezovat množinu agentů, kterým má být služba poskytována a zároveň chránit agenta před podvržením serveru útočníkem. Použití autentizace je podmíněno bezpečným provedením autentizačního procesu u obou stran, především pak u strany umístěné v potenciálně nebezpečném prostředí hostitele. Problémem autentizace v tomto prostředí se více věnuje kapitola 7.

Nevýhodou této ochrany může být zvýšená režie způsobená komunikací mezi agentem a serverem, omezení funkčnosti agenta při nedostupnosti serveru a případná omezení plynoucí z přenosových a výpočetních prostředků serveru v případě, že lokální běh agenta je efektivnější.

6.2 Pozdní detekce útoku a systémové kroky jako reakce

Ochrany tohoto typu využívají systém monitorování a zaznamenávání operací provedených hostitelem softwarového agenta. Tyto záznamy jsou zpětně vyhodnocovány a využitím následných opatření umožňují postih nepovolené akce prováděné hostitelem vůči agentovi. Postih může mít formu materiální sankce, snížení reputace útočníka, nebo dopadů vyplývajících z platné legislativy. Prostředí hostitele musí zajišťovat efektivní, bezpečné a korektní monitorování spolu s bezpečným přenosem získaných údajů zpět k poskytovateli monitorovaného softwarového agenta. Důvěryhodnost získaných údajů je důležitá především pro automatickém zpracování, při kterém by podvržené údaje mohly

sloužit k záměrnému snižování reputace hostitele. Monitorování může být prováděno přímo v prostředí hostitele nebo mohou být monitorovány akce agenta při interakci s okolím. Tato ochrana může být problematicky proveditelná v prostředích bez centrální autority, například při mezinárodní distribuci softwarových agentů nebo při operacích, při kterých není dosaženo limitní částky pro zahájení legislativního procesu.

6.3 Hardwarové výpočetní prostředí

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 (čipové karty), až po komplexní výpočetní platformu v rozsahu současných PC. Rozdělení prostředí v závislosti na míře použití pomocného hardwarového zařízení je podrobně rozebráno v kapitole 4, zde jsou uvedeny vlastnosti a konkrétní představitelé použitelných hardwarových zařízení.

6.3.1 Hardwarový token (dongle, paměťová karta, čipová karta)

Ochrana softwarového agenta pomocí hardwarového tokenu využívá přesunu citlivých dat (paměťové karty) nebo kódu (čipové karty) do fyzicky zabezpečeného prostředí. S tímto prostředím softwarový agent komunikuje a vyžaduje jej pro korektní činnost. Nejjednodušší způsob ochrany umisťuje do hardwarového tokenu chráněnou informaci, která je požadována pro pokračování agentova běhu. Touto informací může být například hodnota klíče, který je použit pro dešifrování části kódu softwarového agenta. Získanou chráněnou informaci může útočník buď přímo použít, nebo vytvořit simulátor hardwarového tokenu, který bude poskytovat informaci namísto originálního hardwarového tokenu.

Pokročilejší verze ochrany nevydává chráněnou informaci softwarovému agentovi, ale pouze zasílá odpověď vytvořenou na základě chráněné informace a přijaté výzvy. Pokud je cílem pouze kontrolovat přítomnost hardwarového tokenu, může být vhodné užití protokolů založených na nulovém šíření znalostí (zero-knowledge protocols). Mezi hardwarové zařízení s touto funkčností patří svého času velmi populární „dongles“ umisťované do LPT portů (Rainbow Sentinel, Alladin HASP) nebo kryptografické USB tokeny s vhodnou funkčností (Rainbow iKey 1000).

Pokud hardwarové zařízení umožňuje vykonávat složitější programovatelné operace, lze do něj přesunout část kódu a dat softwarového agenta. Utajení a integrita výkonu přesunutého kódu je pak fyzicky chráněna proti vybraným útokům. Vhodným příkladem jsou čipové karty podporující rozhraní JavaCard [JC22API], které umožňují vyvíjet, v omezené verzi jazyka Java, aplikace přenositelné mezi čipovými kartami různých výrobců.

Při použití hardwarového tokenu s bezpečností odpovídající hodnotě chráněných informací lze dosáhnout vyšší bezpečnosti oproti samostatnému softwarovému agentovi. Nevýhodou je vyšší cena celkového řešení, problémy plynoucí z nutnosti fyzické distribuce hardwarového tokenu a případná omezení výkonu, použitelnosti a přenositelnosti softwarového agenta. Částečným řešením by mohlo být sdílení jednoho

hardwarového tokenu více softwarovými agenty, jak je možné například při použití čipových karet s rozhranním JavaCard. Zde je implicitně podporováno bezpečné sdílení výpočetních a datových zdrojů mezi nezávislými aplikacemi. V případě existence dostatečně obecné platformy a související infrastruktury umožňující bezpečnou vzdálenou distribuci agentů by se mohlo jednat čipy montované do všeobecně rozšířených zařízení typu mobilní telefony, PDA nebo podpisové čipové karty, což by mělo kladný vliv na výslednou cenu a problémy fyzické distribuce.

6.3.2 Šifrovaná sběrnice

Koncept ochrany agentů pomocí šifrované sběrnice rozšiřuje běžnou architekturu *procesor – vyrovnávací paměť – systémová sběrnice – RAM* o šifrovací hardwarový čip umístěný mezi vyrovávací pamětí a systémovou sběrnicí. Instrukce a data agenta určené pro zpracování procesorem jsou dešifrovány při načítání z paměti RAM a šifrovány během zápisu výsledků zpět. Pokud nemá útočník možnost monitorovat datový tok procházející mezi sběrnicí a procesorem, nemá přístup k nešifrované podobě kódu a používaných dat agenta, a to i v případě, že má plný přístup k obsahu paměti RAM a dalším systémovým zdrojům.

Komerčně nasazený 8bitový procesor Dallas Semiconductor DS5000 užívající koncept šifrované sběrnice byl široce rozšířen v kartách pro předplacenou televizi, šifrovací komunikační zařízení a terminálech pro platební karty. Z důvodu příliš krátkého bloku šifrovaných dat ve vyrovnávací paměti však byl nalezen útok, který umožňoval získat nešifrovanou podobu instrukcí bez znalosti šifrovacího klíče [Ku98].

Koncept kryptoprocesoru TrustNo1 [Ku97] navrhuje princip použití šifrované sběrnice pro oblast 16 a 32bitových počítačů s podporou správu virtuálního paměťového prostoru a multiprocessingu. Šifrovací logika je zde propojena s logikou správy virtuální paměti a vyžaduje použití zvláštního klíče pro každý proces a paměťový segment.

Nevýhodou využití šifrované sběrnice je kromě zvýšení nákladů na potřebný hardware i možnost tvůrců softwarového agenta umisťovat škodlivý kód s minimálním rizikem odhalení. Příkladem je opakovaná inicializace náhodného generátoru čísel užívaného jinými softwarovými agenty náhodného klíčů tak, aby bylo generování klíčů později predikovatelné. Použití šifrované sběrnice by tak mělo být zároveň vázáno na použití operačního systému, který umožňuje kontrolu, omezení a ochranu vykonávaných softwarových agentů. Velmi ztížena je také možnost ladění běhu agenta, pokud se nachází v šifrovaném tvaru.

6.3.3 Hardwarově chráněná výpočetní platforma

Ochrana výpočetní platformy včetně operačního systému umožňuje rozšířit bezpečnost prostředí na celý proces běhu platformy, tedy bezpečné zavádění operačního systému, kontrolu spouštěných agentů a ochranu používaných dat.

6.3.3.1 AEGIS

Návrh bezpečné výpočetní platformy *AEGIS* [AFS97] využívá modifikaci PC platformy pro zajištění startu operačního systému do stavu s očekávanou softwarovou a hardwarovou konfigurací. Výchozím předpoklad je důvěryhodnost základní desky, procesoru a malé části BIOSu sloužícího jako verifikační jádro. Verifikační jádro

umožňuje vytvářet a kontrolovat hodnoty integritních součtů všech komponent, kterým je postupně předáváno řízení. Očekávané hodnoty pro důvěryhodná zařízení jsou uloženy v bezpečném hardwarovém úložišti. Při předávání řízení mezi komponentami vždy komponenta s aktivním řízením nejprve ověří pomocí verifikačního jádra integritu té komponenty, které je řízení předáváno. V případě chybné integrity není řízení předáno a startovací proces je zastaven. Pokud je dostupný důvěryhodný zdroj, může proběhnout obnovovací fáze, během které je získána ověřena důvěryhodnost komponenty, která způsobila zastavení. Startovací proces je opakován. Po zapnutí počítače dojde nejprve ke spuštění verifikačního jádra, které ověří integritu zbylé části BIOSu a předá mu řízení. BIOS následně ověřuje integritu a předává řízení firmwaru rozšiřujících hardwarových komponent. Po úspěšném ukončení kontroly všech rozšiřujících hardwarových komponent dochází k ověření integrity zaváděcího bloku na předvoleném zaváděcím médiu. V případě využití dalších zaváděcích bloků je nejprve ověřena jejich integrita. Poslední zaváděcí blok ověří integritu spouštěného jádra operačního systému a předá mu řízení. A priori důvěryhodné verifikační jádro, spolu s řetězcem integritních kontrol, vede ke startu operačního systému s očekávanou hardwarovou i softwarovou konfigurací.

6.3.3.2 Secure Digital Music Initiative

Z popudu sdružení Recording Industry Association of America (RIAA) a velkých nahrávacích společností (SONY, Warner, BMG, EMI a Universal) byla vyvinuta specifikace ochranného prostředí *Secure Digital Music Initiative* [SDMI01] určená pro omezení možnosti distribuce pirátských audio a video nahrávek v přenosných přehrávačích. Využívá kontrolu přehrávaného obsahu pomocí modulu schopného detekovat v nahrávce přítomnost digitálního vodotisku. Nahrávky s chybějícím nebo nekorektním digitálním vodotiskem mohou být zamítnuty. V druhé fázi vývoje těchto zařízení by měl přibýt modul umožňující tvorbu legálních kopií na základě práv přiřazených k nahrávce, čímž by se celý SDMI systém dal využít pro placenou distribuci. V současné době jediný výrobce (SONY) nabízí zařízení podporující první fázi SDMI a budoucí vývoj je sporný, pravděpodobně byl zastaven.

6.3.3.3 Trusted Computing Group, NGSCB

Aktuální a kontroverzní iniciativou směřující k vytvoření zabezpečené výpočetní platformy je aliance *Trusted Computing Group* (TCG) [TCG04], původně známá pod názvem Trusted Computing Platform Alliance (TCPA). Aliance je tvořena vedoucími firmami průmyslu IT (Intel, Microsoft, AMD, IBM, HP) a její návrh využívá myšlenku bezpečného startu PC platformy dle AEGIS. Cílem specifikace je návrh bezpečnější výpočetní platformy především z pohledu poskytovatelů softwarových agentů a dat zpracovávaných na této platformě. V první fázi implementace je navrženo rozšířením standardní PC platformy o kontrolní, kryptograficky zabezpečený „Fritz“ čip umístěný na základní desce. Takto rozšířené PC platformy jsou již komerčně dostupné [InfTPM02]. V pozdější fázi by měla být obdoba “Fritz“ čipu přímo součástí procesoru. Dalším rozšířením je zavedení nové vlastnosti procesoru umožňující přepínat běh mezi standardním a důvěryhodným režimem. Pouze procesor běžící v důvěryhodném režimu má přístup ke speciálně označeným fyzickým paměťovým oblastem. Tyto oblasti jsou ve standardním režimu nedostupné. Po zapnutí počítače přebírá Fritz čip ihned řízení,

s využitím uložených hodnot integritních součtů ověří očekávaný stav BIOSu, firmwaru přítomných hardwarových zařízení a bezpečnostního jádra operačního systému nazývaného Small Security Kernel (SSK). V případě, že celý proces proběhne úspěšně a PC se nachází ve stavu s očekávanou hardwarovou a softwarovou konfigurací, předá Fritz čip řízení SSK. Zároveň poskytne šifrovací klíče potřebné pro dešifrování kódu a dat softwarových agentů, jejichž běh je pomocí TCG platformy chráněn. Dnes běžné oddělení paměťového prostoru běžících softwarových agentů je rozšířeno o možnost definovat příznak stránek fyzických operační paměti, které jsou přístupné pouze v důvěryhodném režimu procesoru a u kterých je přístup řízen pomocí SSK. Paměť softwarového agenta je tak chráněna před aplikacemi běžnými v současných operačních systémech běžících v režimu Ring 0 s přístupem do paměťového prostoru jiných aplikací.

V současné době vyvíjejí firmy Microsoft a HP operační systém schopný využívat služeb platformy TCG. V případě Microsoftu se jedná o modul budoucí verze operačního systému Windows nazývaný NGSCB (dříve Palladium) [NGSCB04] s bezpečnostním jádrem SSK nazývaným Nexus a chráněnými softwarovými agenty nazývanými NCA (Nexus Computing Agents). Při pokusu o spuštění konkrétního NCA Nexus s pomocí uložených hodnot integritních součtů jeho kódu kontroluje, zda je povolen jeho běh. Následně s využitím důvěryhodného módu procesoru izoluje jeho paměťový prostor a zajistí správu a bezpečné uložení používaných kryptografických klíčů. Na základě hardwarové podpory platformy TCG nabízí NGSCB oproti stávajícím operačním systémům možnost bezpečného uložení kryptografických klíčů v hardwarovém úložišti. Ve srovnání s použitím připojitelné kryptografické čipové karty je řešen i problém integrity a ochrany běhu ovladačů čipové karty. K hardwarovému úložišti mají přístup pouze certifikované NCA, jejichž běh je chráněn proti manipulaci ze strany ostatních NCA. Hardwarové úložiště je využíváno pro poskytování služby bezpečného uložení dat (Sealed Storage) s definováním omezené množiny NCA, kteří mohou s daty manipulovat. Potřebné dešifrovací klíče jsou bezpečnostním jádrem Nexus poskytnuty pouze NCA, s povoleným přístupem. S využitím klíče sdíleným mezi bezpečnostními jádry Nexus lze takto chráněná data zpřístupňovat i pro NCA běžící na dalších TCG platformách. Typickým využitím může být certifikovaná firemní aplikace, určená pro manipulaci s citlivými firemními daty. Data jsou chráněna šifrovacím klíčem poskytovaným Nexy jednotlivých stanic pouze této certifikované aplikaci. Tato aplikace zároveň neumožní export dat do nechráněné podoby. Další důležitou poskytovanou funkčností je bezpečné uchování autentizačních informací, která umožňuje autentizaci (Attestation) jednotlivých NCA mezi sebou nebo vůči vzdáleným žadatelům. HP vyvíjí verzi důvěryhodného operačního systému na bázi Linuxu s obdobnými vlastnostmi.

TCG platformy lze využít jako bezpečné úložiště klíčů namísto kryptografické čipové karty. Oproti současným čipovým kartám je zvýšena bezpečnost komunikace mezi aplikací a úložištěm klíčů. Ačkoli není TCG primárně prezentována jako DRM platforma, je za ni často považována a může být pro tento účel dobře použita. Nabízené funkce umožní distribuci šifrovaných nahrávek, které budou přístupné pouze certifikovaným softwarovým přehrávačům, které neumožní vytváření nelegálních kopií. Potřebný šifrovací klíč je přehrávači poskytnut bezpečnostním jádrem SSK pouze v případě, že přehrávač (NCA) běží v bezpečném prostředí a útočník nemá možnost ovlivňovat jeho běh nebo pozorovat jeho data. Organizace pak také mohou využít TCG platformu pro

hardwarově prosazované povinné řízení přístupu využitelného například pro kontrolu pohybu a přístupu ke klasifikovaným dokumentům. Při vytváření klasifikovaného dokumentu je jeho obsah zašifrován klíčem, který je poskytnut pouze omezené množině certifikovaných aplikací běžících na vybraných počítačích s TCG platformou.

Myšlenku nasazení TCG platformy v navrhované podobě provází výrazná kontroverze vyplývající především z faktu, že cílem iniciativy je vytvořit důvěryhodnou platformu z hlediska poskytovatelů software a dat, nikoli z hlediska samotného uživatele platformy. Nutnost certifikace softwarových aplikací, kterým bude umožněno běžet jako NCA v důvěryhodném režimu, přináší dodatečné náklady, které mohou být neúnosné pro malé poskytovatele software. Možnost distribuce šifrovaných aplikací, bez možnosti jejich inspekce pomocí debuggerů běžných v dnešní době, poskytuje ochranu kódu proti reverznímu inženýrství. Zároveň však brání zisku specifikaci proprietárního formátu pro zajištění kompatibility mezi aplikacemi. Nelze ani prokázat, že poskytovatel aplikace využívá kód původně šířený například pod GNU licencí bez dodržení licenčních podmínek. Možnost hardwarově prosazovaného povinného řízení přístupu zvyšuje možnost kontroly pohybu dokumentů. Zároveň však znesnadňuje úniku informací upozorňující veřejnost na nelegální činnost, páchanou v rámci organizace. Dobrý přehled této problematiky lze nalézt v [An03].

6.4 Šifrování kódu a dat agenta

Ochrana tohoto typu využívá šifrování pro zamezení inspekce a smysluplné modifikace kódu a dat agenta. Jednotlivé varianty ochrany se liší dobou provádění dešifrování kódu a dat a místem ukrytí použitých šifrovacích klíčů.

Nejednodušší metodou je dešifrování celého kódu agenta během jeho nahrávání do paměti. Výkon agenta je tak zpomalen pouze během úvodní fáze, při které je nahráván do paměti a na následný průběh již není výkonnostně ovlivněn. Hodnota klíče může být uložena v nešifrované části agenta, nebo může být odvozena z charakteristik okolního prostředí. Odvození klíče může být provedeno kombinací částí integritních součtů agentových souborů [No00] nebo odvozením ze sériových čísel hardwarových komponent typu harddisk a procesor. Lze tak částečně zamezit statické inspekci kódu v případě dostatečně bezpečného uložení nebo generování šifrovacího klíče, nevýhodou zůstává přítomnost celého dešifrovaného kódu v paměti. Dešifrovaný kód může být čten a ukládán v otevřené podobě.

Použití výpočetní platformy umožňující dešifrovat kód agenta v místě mimo kontrolu útočníka chrání agenta před statickou i dynamickou inspekcí. Spolu s infrastrukturou umožňující bezpečnou distribuci dešifrovacích klíčů může taková platforma poskytovat velmi dobrou bezpečnost, nutnost použití specializovaného hardware však zvyšuje nákladnost celého řešení. Specializovaný hardware se může stát úzkým výpočetním místem s omezenou rozšiřitelností. Výpočetní platformy tohoto typu může být založena na využití šifrované sběrnice a principů popsaných v konceptu procesoru TrustNo1 nebo výpočetní platformě TCG.

Kompromisem mezi dešifrováním agenta v místě mimo kontrolu útočníka a kompletním dešifrováním během nahrávání do paměti, je metoda „plovoucího dešifrovacího okna“. Při běhu agenta dochází k dešifrování do paměti pouze té části spustitelného kódu, která je aktuálně prováděna. Po jejím vykonání je z paměti

odstraněna. Kód agenta se tak nikdy neobjeví v paměti celý v dešifrované podobě. Cenou je snížení výkonu agenta následkem opakovaného dešifrování stejných částí kódu. Vhodnou volbou velikosti plovoucího okna lze nalézt kompromis mezi bezpečností a požadovaným výkonem agenta.

Pokud je dešifrovací klíč odvozován z charakteristik prostředí, je nutné bezpečně ukrýt algoritmus provádějící generování klíče, neboť při jejím zisku je útočník schopen dešifrovací klíč rekonstruovat sám. Pokud je třeba chránit kód agenta pouze do doby jeho první aktivace, lze využít techniku neinformovaných agentů [RiSch98], při které je generování klíče vázáno na výskyt specifické události prostředí, která není pro útočníka předpověditelná ani ze znalosti algoritmu generování hodnoty klíče.

Specifickou verzí této ochrany je přímý výkon šifrovaného kódu pomocí tzv. mobilní kryptografie, viz další část práce a [SaTs98]. V tomto případě nedochází k převodu kódu před jeho výkonem na dešifrovanou podobu. Kód je přímo navržen v proveditelné formě, která ale zároveň neumožňuje zisk uchovávaných dat nebo tvorbu mentálního model kódu.

6.5 Mobilní kryptografie

Tato technika ochrany softwarových agentů vyvrací pro některé třídy výpočetních úloh (uvedené později) domněnku, že nelze zabránit útočníkovi v zisku citlivých data a vytvoření mentálního modelu z kódu softwarového agenta běžící v prostředí pod plnou kontrolou útočníka. Chráněné operace anebo data jsou pomocí matematické transformace převedena na sérii alternativních operací, které dávají nad požadovanou množinou vstupních dat stejná výstupní data jako původní operace. Zároveň je však výpočetně obtížné, bez znalosti náhodného materiálu použitého pro transformaci, určit z výsledné série operací původní chráněnou operaci nebo extrahovat používaná citlivá data. Ochrana citlivých operací a dat nezávisí na utajení implementace použité matematické transformace, ale pouze na utajení náhodného materiálu použitého během ní. Mobilní kryptografie (MC) poskytuje ochranu proti útokům zaměřeným na získání citlivých dat nesených agentem nebo modifikaci kódu a dat za účelem cíleného ovlivnění agentovi funkčnosti. Mobilní kryptografie poskytuje dvě základní varianty ochrany v závislosti na tom, zda je cílem chránit hodnotu zpracovávaných dat nebo algoritmus jejich zpracování.

Výpočtem se šifrovanými daty (Computing with Encrypted Data – CED) je označován proces, při kterém strana B provede výpočet operace F nad daty X poskytnutými stranou A. Strana B nedozví z průběhu výpočtu operace F žádnou podstatnou informaci sloužící k odhalení hodnoty dat X a strana A se nedozví žádnou podstatnou informaci sloužící k odhalení operace F.

Výpočtem s šifrovanou funkcí (Computing with Encrypted Function – CEF) je označován proces, při kterém strana B vykoná nad vlastními vstupními daty X program P, poskytnutý stranou A 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.

V případě, že strany A a B nemusí během provádění operace F spolu komunikovat, s výjimkou předání vstupních dat (CED) nebo předání programu (CEF), je taková forma počítání nazývána neinteraktivní a je vhodná pro použití u softwarových agentů běžících v prostředí pod kontrolou útočníka bez možnosti on-line komunikace s bezpečným

prostředím poskytovatele softwarového agenta. Variantu CED je vhodné použít v případě kdy vyžadujeme, aby měl softwarový agent možnost používat externě poskytnutá citlivá data a zároveň nedošlo k zpřístupnění jejich hodnoty útočníkovi. Varianta CEF je vhodná pro případ, kdy má mít softwarový agent možnost vykonávat citlivou operaci umístěnou v jeho kódu, u které chceme zabránit vytvoření mentálního modelu útočníkem. Operace prováděná v rámci CED nebo CEF může být v současné době pouze polynomiální nebo racionální funkce [SaTs98].

Oproti jiným technikám ochrany typu obfuskace, založeným na „ad hoc“ odhadu očekávaného útočníkova postupu pro hodnocení výsledné bezpečnosti, poskytuje MC možnost matematické dokazatelnosti časové náročnosti odstranění ochrany softwarového agenta. Obtížnost odstranění ochrany je založena na nutnosti řešení problémů, u kterých je prokazatelně známa jejich nepolynomiální výpočetní složitost, případně takový algoritmus nebyl dosud nalezen. MC nevyžaduje použití pomocného hardware, výsledný chráněný kód softwarového agenta však může být oproti nechráněnému kódu pomalejší a objemnější. Při použití CEF je zajištěna ochrana proti vytvoření mentálního modelu operace a tím i zamezení cílených změn kódu, útočník však stále může ovlivňovat běh operace pomocí náhodných změn kódu. Problémem, který může vyžadovat použití dalších ochran jiného typu, je útočníkova možnost vyjmout z těla softwarového agenta kód chráněné operace. Tento kód může být prováděn nad podvrženými daty i bez znalosti obsažené operace.

6.5.1 White-Box Attack Resistant AES

Myšlenka počítání se šifrovanou funkcí je využita pro návrh implementace šifrovacího algoritmu AES. Oproti standardní implementaci umožňuje šifrovat v prostředí pod kontrolou útočníka bez vyzrazení hodnoty použitého šifrovacího klíče. Vzhledem k použití v projektové části této diplomové práce je tato varianta AES rozebrána podrobněji.

Implementaci šifrování pomocí White-Box Attack Resistant AES (WBACR AES) [CEJO02] je vhodná v případě potřeby sdílet symetrický šifrovací klíč mezi bezpečným prostředím a softwarovým agentem běžícím v prostředí pod kontrolou útočníka. Klasická implementace šifrovacích algoritmů, které přijímají na vstupu blok dat a hodnotu klíče, zde nedostačuje. Útočník snadno získá kombinací statické a dynamické inspekce hodnotu sdíleného klíče. Implementace šifrovacího algoritmu by tedy měla být modifikována tak, aby nemusela být celá hodnota klíče v otevřené podobě vložena v jednom místě kódu a jeho utajení tak nezáviselo pouze na utajení místa použití. Zároveň by bylo výhodou, kdyby výstup takto modifikovaného šifrovacího algoritmu odpovídal klasické implementaci. Vstupní a výstupní data by tak mohla být v jiném prostředí, kde není utajení hodnoty klíče problémem, zpracována pomocí klasické implementací daného šifrovacího algoritmu. Implementace šifrovacího algoritmu AES formou WBACR AES obě tyto vlastnosti splňuje.

Hlavní myšlenkou WBACR AES je převedení celého šifrovacího algoritmu přijímajícího na vstupu hodnotu klíče a vstupní data na upravený algoritmus, který přijímá pouze vstupní data. Klíč je v upraveném algoritmu již určitým způsobem „obsažen“. Upravený algoritmus je následně použitelný pouze šifrování konkrétní hodnotou klíče. Upravený algoritmus musí zároveň poskytovat ochranu hodnotě

obsaženého klíče takovým způsobem, aby útočník i při znalosti postupu tvorby upraveného algoritmu nebyl schopen hodnotu klíče pro konkrétní instanci upraveného algoritmu získat. WBACR AES implementuje algoritmus AES tak, že dílčí operace AddRoundKey, SybByte, ShiftRow a MixColumn nahrazuje náhledy do předem připravených tabulek. Předpočtené tabulky jsou vytvářeny ze znalosti hodnoty klíče předem, v bezpečném prostředí mimo kontrolu útočníka. Výsledkem přípravy pro různé hodnoty klíčů jsou tabulky s rozdílnými vnitřními hodnotami, samotný algoritmus náhledů do tabulek je na hodnotě klíče nezávislý a nemění se.

Pro režim zašifrování pomocí WBACR AES v režimu 128 bitů klíč a 128 bitů blok jsou vytvořeny tabulky tří typů o celkové velikosti 262 144 bajtů. Tabulky pro režim dešifrování mají obdobnou strukturu a stejnou velikost. Prvním typem jsou tabulky obsahující předpočtený výsledek operací AddRoundKey, SubByte a části operace MixColumn (ASMT tabulky). Druhým typem jsou tabulky obsahující předpočtený výsledek operací XOR (XT tabulky), nutné pro dokončení operace MixColumn. Třetím typem jsou tabulky pro operace závěrečné rundy obsahující předpočtený výsledek operací AddRoundKey, SubByte a závěrečný AddRoundKey (ASAT tabulky). Podrobnosti tvorby WBACR AES tabulek jsou uvedeny v příloze C.

6.5.1.1 Šifrování pomocí WBACR AES

Blok vstupních dat je modifikován na základě série náhledů do tabulek, po posledním náhledu je vstupní blok zašifrován klíčem, který byl použit pro generování tabulek. Vstupní blok dat je nejprve převeden do tvaru matice 4x4 popsané v algoritmu AES. Pro devět běžných rund se šifrovaní skládá z náhledů do ASMT tabulek pro každý prvek matice. Výsledek náhledu je použit pro sérii šesti náhledů do XT tabulek, provádějící jeho kompozici. Výsledek je uložen zpět do matice a pokračuje se následující rundou. Ve finální rundě je výsledek získán jedním náhledem do ASAT tabulek pro každý prvek matice. Tím je šifrování ukončeno. Během šifrování dojde celkem k 1024 náhledům do předpočtených tabulek. Analogicky probíhá režim dešifrování.

Vzhledem k možnosti použití konkrétních tabulek pouze pro jediný, předem daný klíč, jsou vyloučeny šifrovací režimy provádějící změnu klíče po každém bloku dat. Standardní verze režimů ECB, OFB, CFB a CBC je možno použít. Způsob, jak vhodně definovat vstupní resp. výstupní kódování (6.5.1.4) vstupu resp. výstupu celého algoritmu tak, aby bylo možno použít režim CBC, je navržen v kapitole 7.

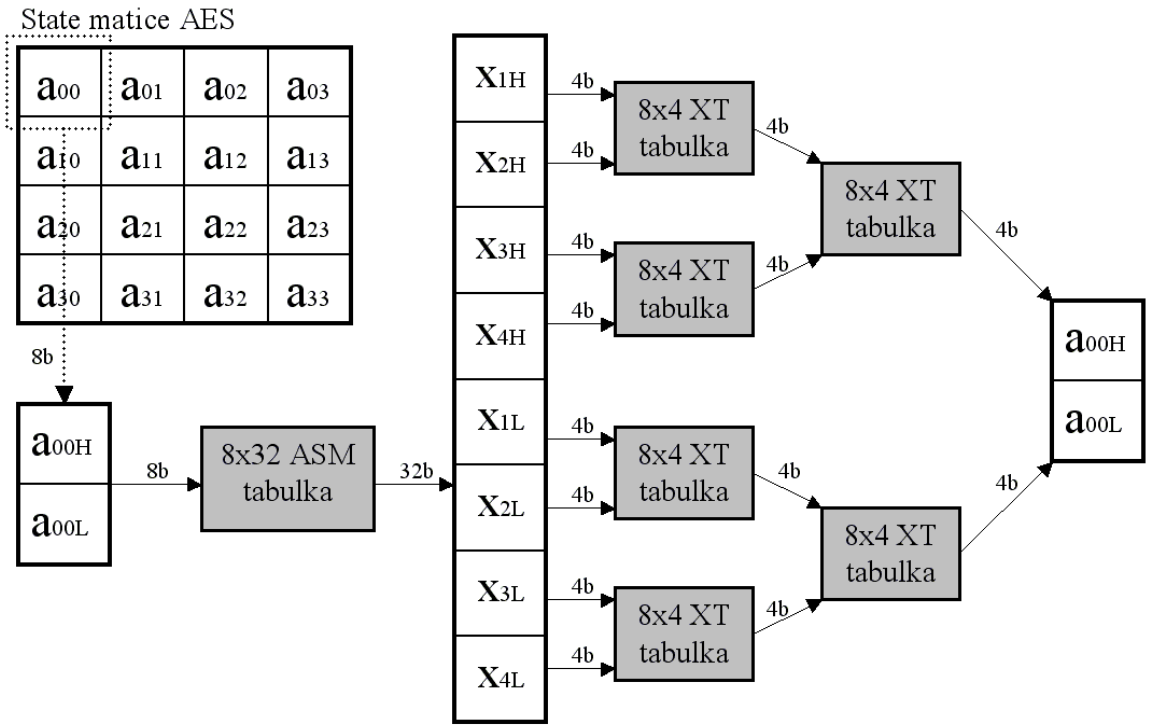
6.5.1.2 Bezpečné ukrytí hodnoty šifrovacího klíče

Hodnota šifrovacího klíče v expandované podobě je použita pro tvorbu ASMT tabulek. Díky použití interního kódování (viz. příloha C) je otevřená podoba hodnoty šifrovacího klíče chráněna před statickou i dynamickou inspekcí. Zisk hodnoty šifrovacího klíče i v případě dostupnosti tabulek ASMT a XT by měl být výpočetně velmi náročný.

6.5.1.3 Oddělitelná šifrovací a dešifrovací funkčnost

Softwarový agent může obsahovat tabulky použitelné pouze pro šifrování nebo dešifrování. Díky této vlastnosti lze vyměňovaná data šifrovat pomocí symetrické kryptografie a zároveň využít všech 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 (nebo naopak). Proces vytváření digitálního podpisu zprávy pak může být simulován pomocí časově méně náročné tvorby autentizačního kódu (MAC).



Obr. 6.1: Běžná šifrovací runda WBACR AES

6.5.1.4 Vstupní a výstupní kódování

Použití vstupního nebo výstupního kódování umožňuje ztížit možnost samostatného použití tabulek extrahovaných z kódu softwarového agenta. Zároveň zvyšuje provázanost mezi tabulkami a softwarovým agentem. Vstupním kódováním je myšlena bijektivní transformace vstupního bloku dat, která je v rámci náhledů do ASMT tabulky první rundy pomocí inverzní transformace odstraněna. Tato transformace může být identita, pak není potřeba vstupní blok nikterak upravovat. V opačném případě je třeba vstupní blok před prvním náhledem příslušně transformovat, jinak výsledkem inverzní transformace nebude původní blok a výsledný zašifrovaný, resp. dešifrovaný blok pak bude nekorektní. Analogicky pro výstupní kódování. Tohoto chování lze s výhodou využít. Softwarový agent může nenápadně a ve velkém úseku kódu provádět transformaci, která odpovídá inverzní transformaci prvního náhledu do tabulek. Pro možnost použití tabulek jako šifrovacího stroje musí útočník kromě extrakce samotných tabulek získat z kódu i popis této transformace. Dalším využitím je personalizace jednotlivých instancí softwarových agentů pomocí různých transformací, které provádí na vstupními daty před prvním náhledem do tabulek. Poskytovatel softwarového agenta pak může při znalosti těchto transformací vzdáleně vyměňovat tabulky za nové. Při generování nových tabulek pouze nastaví vstupní kódování odpovídající cílovému

softwarovému agentovi. Útočník bez znalosti této transformace nebude schopen vygenerovat ve vlastní režii takové tabulky, které by softwarový agent mohl korektně používat.

Je vhodné poznamenat, že vzhledem k nezávislému zpracování jednotlivých bajtů vstupního bloku v prvním (poslednímu) kroku šifrování dle WBACR AES lze použít nezávislé 8bitové kódování pro každý z 16 bajtů vstupního (výstupního) bloku. Celkem lze tedy použít 16 nezávislých 8bitových vstupních kódování a stejný počet pro výstupní kódování.

Body 6.5.1.2 až 6.5.1.4 shrnují klíčové výhody šifrování pomocí WBACR AES, které jsou u standardní implementace algoritmu AES nedostupné. Těchto vlastností je využito při návrhu protokolu SEAUT v kapitole 7.

6.5.1.5 Bezpečnost WBACR AES

Bezpečnost šifrování pomocí WBACR AES je vhodné zvážit ve třech základních kategoriích a dále dle konkrétního způsobu použití.

První kategorií je bezpečnost zašifrovaných dat pomocí WBACR AES proti útočníkovi, který nevlastní WBACR AES tabulky, ani klíč použitý k jejich generování. V tomto případě je bezpečnost ekvivalentní standardní implementaci algoritmu AES se 128bitovým klíčem a 128bitovým blokem.

Druhou kategorií je situace, kdy jsou WBACR AES tabulky umístěné v kódu softwarového agenta a útočník se je snaží extrahovat. Zde bude bezpečnost závislá na použité formě a úrovni obfuskace, obecně však lze říci, že vzhledem k velikosti tabulek a omezení operací pouze na náhledy do tabulek, bude obfuskace poměrně dobře proveditelná. Extrahované tabulky může útočník použít jako šifrovací stroj bez znalosti šifrovacího klíče. Toto použití lze ztížit zavedením vstupního a výstupního kódování.

Třetí kategorií je případ, kdy se již útočníkovi podařilo tabulky extrahovat a jeho cílem je zisk hodnoty klíče, který byl použit pro jejich generování. Hodnota klíče je chráněna pomocí interních kódování hodnot, vznikajících jako meziprodukty šifrovacího procesu během náhledů do tabulek. Na obtížnost odstranění interního kódování má i vliv kvalita použitého generátoru náhodných čísel (RNG). Pokud je kritickým požadavkem ochrana hodnoty použitého klíče, je vhodné využít RNG integrovaný například v některých kryptografických čipových kartách. Pokud pouhá extrakce WBACR AES tabulek z kódu vede k srovnatelnému narušení bezpečnosti, jako zisk samotné hodnoty klíče, použití kvalitního pseudonáhodného RNG může být dostačující. V současné době není znám žádný výpočetně proveditelný útok, který by umožnil odstranit interní kódování a vedl tak ke kompromitaci hodnoty uchovávaného klíče. Bližší rozbor bezpečnosti WBACR AES lze nalézt v [CEJO02].

6.6 Sebekontrolující kód

Ochrana využívající kontrolu integrity agenta je určena pro zamezení změn v kódu a datech. Jednotlivé varianty ochrany se liší v závislosti na velikosti kontrolovaných úseků, frekvenci provádění kontrolní operace a způsobu reakce na zjištěnou modifikaci.

Nejjednodušší varianta je automaticky prováděna operačním systémem před každým spuštěním softwarového agenta. Pomocí integritních součtů typu CRC32 je

jednorázově kontrolována integrita celého kódu softwarového agenta vůči hodnotě nalezené ve standardní hlavičce. Tato ochrana je určena pouze proti náhodným chybám nebo neúmyslné modifikaci a nelze ji použít pro ochranu proti záměrné modifikaci. Údaj v hlavičce není chráněn a postup jeho tvorby je dokumentován. Kontrola je navíc prováděna pouze při spuštění a nezachytí změny provedené v operační paměti po agentově spuštění. Ve skutečnosti se nejedná o sebekontrolující kód, neboť kontrola je prováděna operačním systémem, nikoli agentem samotným.

Na myšlence opakované kontroly integrity částí kódu pomocí vhodných integritních funkcí oproti očekávaným hodnotám jsou založeny ochrany [ChaAt01, HMST01]. Kontrola je prováděna pomocí velkého počtu (řádově stovky) malých kusů kódu nazývaných testery, jejichž úkolem je co nejméně nápadně ověřit integritu přidělené části kódu. V případě zjištěné modifikace je spuštěn reakční mechanismus, který na situaci příslušně zareaguje. Reakcí může být ukončení výkonu agenta nebo jeho chybná funkčnost. Po změně kódu 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. V ideálním případě je útočník donucen deaktivovat takřka všechny umístěné testery. Pokud tuto deaktivaci není schopen provádět automaticky, výrazně se zvyšuje doba nutná na provedení úmyslné změny, která unikne detekci a reakci. Cílem ochrany je donutit útočníka k vytváření mentálního modelu z co největší části agentova kódu, přestože hodlá provést modifikaci pouze vybraného úseku. Podrobnosti tvorby a funkčnosti testerů a reakčních mechanismů lze nalézt v příloze C.

6.7 Obfuskace

Metoda ochrany nazývaná obfuskace (obfuscation, mess-up algorithms) využívá takových transformací kódu a dat, které plně zachovávají původní funkčnost kódu, zároveň však ztěžují jeho srozumitelnost, a to i při použití pomocných automatizovaných prostředků. Cílem ochrany je ztížit útočníkovi vytvoření mentálního modelu použitých algoritmů, ztížit vytvoření diagramu toku dat, zabránit smysluplné modifikaci kódu a extrakci uložených citlivých dat. Různé programovací jazyky mají rozdílné předpoklady pro vhodnost provádění obfuskace. Obecně jsou za vhodné považovány jazyky C, C++ a Perl. Obfuskace může být prováděna na úrovni zdrojového kódu, přeloženého kódu nebo vhodnou kombinací obou přístupů. Obfuskační transformace se dělí na výpočetní, datové a preventivní.

Výpočetní transformace ovlivňují srozumitelnost původního běhu kódu vkládáním mrtvého nebo irelevantního kódu, zaváděním redundantních operacích, paralelizací kódu nebo odstraněním struktur kódu, vznikajících standardně používanými programovacími vzory.

Datové transformace ovlivňují způsob přístupu a zpracování datových částí změnou kódování, rozdělováním a slučováním proměnných, restrukturalizací datových polí, převodem statických struktur na dynamicky generované nebo modifikací dědičných vztahů objektově orientovaného programování.

Preventivní transformace jsou zaměřené proti známým technikám reverzního inženýrství, dekompilátorům a deobfuskačním nástrojům. Využívají znalosti chování a

omezení rozšířených nástrojů používaných pro statickou a dynamickou inspekci kódu nebo nástrojů přímo určených pro odstranění obfuskačních transformací.

Nevýhodou použití obfuskačních transformací může být zvýšení výsledné velikosti přeloženého kódu, snížení rychlostí vykonávání některých operací a výrazné ztížení ladění výsledného kódu. Podrobnější popis výše uvedených transformací, metrik používaných pro hodnocení stupně obfuskace a běžně používaných obfuskačních nástrojů lze nalézt například v [CTL97, CGJZ01, ClkwTr02, NCJ01, CTL98] a příloze C.

6.8 Nutnost pravidelné aktualizace agenta

Pokud je pro korektní běh softwarového agenta nutná pravidelná aktualizace (software aging) a poskytování této aktualizace lze kontrolovat, pak lze tohoto prostředí využít pro zvýšení rizika odhalení člověka, který nelegálně šíří kopie svého legálně získaného softwarového agenta. Cílem této ochrany není ochránit kód ani data softwarového agenta, ale pouze zvýšit nutnost interakce mezi jeho nelegálním uživatelem a člověkem, který nelegální kopie poskytuje.

Nutnost pravidelných aktualizací je buď přirozená vlastnost softwarového agenta jako v případě antivirových programů a bezpečnostních záplat operačních systémů, nebo ji lze vyvolat uměle, jak je popsáno v [JaRe01]. Navrhovaná metoda je určena pro prostředí, ve kterých dochází k výměně dat vytvářených jednotlivými softwarovými agenty náležícím různým uživatelům. Vyměňovaná data jsou šifrována klíči, které se pravidelně mění a jsou distribuovány pomocí pravidelných aktualizací. Obsahem aktualizace je hodnota šifrovacího klíče platného pro aktuální časový interval, ze které lze zároveň odvodit i hodnoty klíčů pro všechny předešlé intervaly. Softwarový agent během produkce výstupních dat vybere šifrovací klíč platný pro aktuální časový interval, do hlavičky dat přidá identifikaci intervalu a výstupní data zvoleným klíčem zašifruje. Takto zašifrovaná data bude moci dešifrovat pouze softwarový agent, který vlastní aktualizaci alespoň pro ten časový interval, ve kterém byla data vytvořena. Softwarový agent, který nezískává pravidelně aktualizace, bude schopen přijímat vstupní data vytvořené pouze do toho časového intervalu, pro který vlastní poslední aktualizaci a postupně bude narůstat množství dat, ke kterým nebude mít přístup. Pro zachování plné funkčnosti bude uživatel nucen obstarat si novější aktualizaci. V případě nelegálního uživatele bude docházet k opakované interakci s poskytovatelem nelegální kopie, což by mělo vést ke zvýšení rizika jeho odhalení.

Kritickým místem celého protokolu pro dosažení nutnosti pravidelné aktualizace vůči nelegálnímu poskytovateli je zajistit, aby nebylo možno aktualizace sdílet prostřednictvím anonymních kanálů typu BBS, FTP a peer-to-peer sítě. Vzhledem k rozšíření PTP sítí se mi jeví možnost zamezení šíření nechráněných aktualizací nereálná. Možným řešením by byla aktualizace šifrovacích klíčů šifrovaným kanálem do hardwarových tokenů, které by pro program prováděli šifrování, resp. dešifrování výstupních a vstupních dat, je tím však pouze zvýšena obtížnost získání hodnoty globálně použitelného klíče.

Šifrováním výstupních dat je výrazně omezena otevřenost vyměňovaných dat a tím i interoperabilita softwarových agentů různých poskytovatelů, což by v praktickém použití vedlo k nutnosti používat pouze produkty omezené množiny poskytovatelů pro možnost sdílet produkovaná data.

6.9 Neinformovaný agent

Ochrana založená na principu neinformovaného agenta [RiSch98] využívá odvození chráněných informací takovým způsobem, aby znalost algoritmu odvození nevedla k zisku informace. Neinformovaný agent typicky disponuje šifrovanou zprávou (proveditelné instrukce, citlivá data) a metodu, pomocí které periodicky prohledává charakteristiky okolního prostředí ve snaze získat data nutná pro generování dešifrovacího klíče k nesené zprávě. V případě získání těchto dat provede dešifrování své zprávy a použije ji. Bez správných hodnot charakteristik poskytovaných prostředím není schopen zprávu dešifrovat (je neinformovaný) a je tedy kryptograficky odolný proti inspekci nesené zprávy až do doby výskytu očekávané charakteristiky prostředí.

Hlavní problém s odvozením šifrovacích klíčů z prostředí spočívá v plné kontrole útočníka nad tímto prostředím a jeho možnosti libovolně modifikovat prostředím poskytované charakteristiky, dále plnou kontrolou nad vstupy agenta a jeho vnitřními stavy. Odvození tak musí být odolné proti inspekci kódu, nesmí tedy obsahovat přímý popis charakteristik prostředí, při kterých je chráněná informace zpřístupněna. Prostor hodnot charakteristik musí být natolik rozsáhlý, aby nebylo možné použít slovníkový útok, při kterém útočník systematicky podvrhuje hodnoty charakteristiky ve snaze určit hodnoty, které vedou k odvození správného klíče. Příkladem takto odolného odvození může být odvození správného šifrovacího klíče pouze v případě, že je spuštěn na počítači s předem známým jménem „Memetika“. Agent nese neinformovaně identifikaci cílového počítače pomocí dvojnásobně hashované hodnoty jeho jména, tedy například CÍL = H(H(Memetika)) s využitím kryptografické hash funkce. Při spuštění na libovolném počítači spočte hodnotu H(H(jméno_ počítače)) a porovná s hodnotou CÍL. V případě odpovídající hodnoty odvodí hodnotu klíče jako H(jméno_počítače) a dešifruje nesenou zprávu. Ze znalosti hodnoty CÍL nelze určit ani cílový počítač, na kterém bude agent schopen dešifrovat nesenou zprávu, ani hodnotu šifrovacího klíče.

Neinformovaný agent může být použit pro konstrukci šifrovacích klíčů ze specifických hodnot prostředí, neinformované testování existence informace, jejíž hodnota nemá být předem jasná, ochranu obsahu množiny akcí, které vedou ke spuštění reakčních mechanismů u systémů pro detekci průniku (IDS) nebo ochranu souboru podmínek prostředí, jejichž splnění aktivuje běh agent. Popis některých technik konstrukce a využití neinformovaných agentů je uveden v příloze C.

7 Protokol autentizace a výměny dat

V této kapitole bude rozebrána problematika volby vhodného autentizačního a komunikačního protokolu pro potřeby bezpečné komunikace mezi dvěma stranami, z nichž jedna není umístěna v bezpečném prostředí. Útočník se tak může snažit narušit probíhající protokol nejen analýzou a manipulací vyměňovanými zprávami, ale i manipulací samotného procesu zpracování zpráv protokolu jednou ze stran.

Nejprve je uveden popis vlastností prostředí komunikujících stran, následuje přehled obecně požadovaných cílů, jejichž dosažení by měly vhodné protokoly zajišťovat. Pokud není uvedeno explicitně jinak, předpokládá se umístění strany A v bezpečném a strany B v potenciálně nebezpečném prostředí. Pro tento případ je proveden rozbor zranitelnosti standardní implementace 3-průchodového protokolu ISO 9798-2 pro vzájemnou autentizaci (dále 3P-ISO9798-2) a na základě určených slabých míst tohoto protokolu je navržena modifikace, která by měla v uvažovaném prostředí lépe zajišťovat bezpečnou autentizaci a komunikaci.

7.1 Vlastnosti prostředí komunikujících stran

Hledaný protokol by měl zajišťovat bezpečnou komunikaci mezi dvěma stranami, z nichž jedna se nachází v bezpečném výpočetním prostředí, ke kterému nemá útočník přístup, zatímco druhá je naopak v prostředí pod plnou kontrolou útočníka. Toto dělení je zjednodušující, neboť bezpečnost výpočetního prostředí je relativní vzhledem k prostředkům, které musí útočník vynaložit na jeho prolomení. Přesto však má toto dělení smysl v případě, že komunikující strany se nacházejí v různých úrovních zabezpečení, z nichž jednu pokládáme za dostačující vzhledem k hodnotě zpracovávaných informací, zatímco druhou nikoli. Příkladem může být program vykonávaný na zabezpečeném serveru poskytovatele služby nebo v programovatelném hardwarovém tokenu (strana A) a softwarový agent na uživatelově PC (strana B). Útočník má přístup ke všem vyměňovaným zprávám, strana A se vůči němu jeví jako černá skříňka, kterou může ovlivňovat pouze manipulací vyměňovaných zpráv. Proti straně B může naopak použít všechny útoky uvedené v kapitole 3.2.

Vzhledem k užití protokolu pro potřeby DRM, při kterém lze zhruba rozdělit komunikující strany na stranu, která poskytuje určitou službu a na stranu, která o využití služby žádá, je vhodné nahlížet na stranu A jako na poskytovatele služby a stranu B jako na uživatele služby. V přehledu požadovaných cílů bude uveden význam cíle právě na základě tohoto dělení.

7.2 Požadované vlastnosti protokolu

Hledaný protokol by měl mít obecné vlastnosti, které mají význam z hlediska použití pro DRM. Problémy zajištění požadované vlastnosti vycházejí z předpokladu, že strana B může být kontrolována útočníkem, strana A se nachází v bezpečném prostředí a zprávy vyměňované v průběhu komunikace mohou být útočníkem zachycovány. Zde uvedené problémy se obecně vztahují ke všem autentizačním a komunikačním protokolům, některé z nich mohou být konkrétním protokolem uspokojivě řešeny. Později v textu

bude uveden podrobný rozbor problémů a útoků na protokol 3P-ISO9798-2 a jeho navrhované modifikace SEAUT.

7.2.1 Autentizace strany B vůči straně A

Cílem je omezit poskytování služby pouze vybraným stranám B, jejichž identita je prokazována pomocí vhodného autentizačního mechanismu, typicky s využitím symetrické nebo asymetrické kryptografie. Pro bezpečné umístění autentizačních informací na straně B je třeba řešit problém ukrytí jejich přímé hodnoty před útočníkem a problém využití autentizačních funkcí pouze způsobem definovaným autentizačním protokolem nad daty, která nebyla podvržena útočníkem. V některých případech může být potřeba zajistit originální autentizační informace proti nahrazení hodnotami podvrženými útočníkem. Problém čerstvosti autentizační komunikace vzhledem k A lze zajistit standardními metodami s využitím keksíků ve formě náhodných čísel nebo časových razítek.

7.2.2 Autentizace strany A vůči straně B

Cílem je zajistit, aby strana B poskytovala data pro zpracování pouze autentizovanému poskytovateli služby. Identita strany A je prokazována pomocí vhodného autentizačního mechanismu, typicky s využitím symetrické nebo asymetrické kryptografie. V případě použití asymetrické kryptografie je nutné zajistit ochranu proti podvržení veřejného klíče A, při použití symetrické kryptografie je nutné navíc ukrýt i hodnotu sdíleného klíče. V obou případech je nutno chránit integritu procesu ověření identity strany A. Dodatečným problémem je samotné uchování indikace stavu, zda je již strana A autentizována vůči B či nikoli. Problém čerstvosti autentizační komunikace vzhledem k A lze zajistit standardními metodami s využitím keksíků ve formě náhodných čísel nebo časových razítek, je však nutné zajistit jejich korektní generování a integritu dočasně uchovávaných hodnot použitých keksíků.

7.2.3 Autentizace dat pocházejících od strany B

Cílem je zajistit poskytování služby stranou A pouze nad daty pocházejícími od autentizované strany B. Při použití symetrické kryptografie je původ dat prokazován hodnotou autentizačního kódu (MAC) vytvořeného s použitím sdíleného symetrického integritního klíče. Může se jednat o dlouhodobě sdílený klíč nebo (lépe) o klíč ustanovený na počátku relace. V obou případech je třeba řešit problém ukrytí hodnoty integritního klíče a zároveň zajistit použití funkcí pro výpočet integritního součtu pouze nad daty, která nebyla podvržena útočníkem. Při použití asymetrické kryptografie je původ dat prokazován digitálním podpisem pomocí privátního klíče strany B. Hodnotu privátního klíče a podepisovací funkce je třeba chránit před útočníkem.

7.2.4 Autentizace dat pocházejících od strany A

Cílem je zajistit, aby strana B zpracovávala pouze data pocházející od autentizované strany A. Situace je obdobná bodu 7.2.3, v případě použití asymetrické kryptografie je třeba zajistit integritu veřejného klíče A umístěného na straně B. Navíc je třeba chránit integritu procesu ověřování autenticity vstupních dat.

7.2.5 Čerstvost dat pocházejících od strany B

Cílem je zajistit ochranu proti duplikaci zachycených zpráv určených pro stranu A. Čerstvost zprávy může být zajišťována vhodným keksíkem ve formě náhodného čísla, pořadového čísla nebo časového razítka spolu s mechanismem pro zajištění jeho integrity. Keksík lze kombinovat s šifrováním zprávy dynamicky ustanoveným klíčem relace. Použití keksíku se lze zcela vyhnout, pokud je každá zpráva šifrovaná unikátním klíčem. Při použití klíče relace je třeba zajistit jeho bezpečné uchování na straně B.

7.2.6 Čerstvost dat pocházejících od strany A

Cílem je zajistit ochranu proti duplikaci zachycených zpráv určených pro stranu B. Situace je obdobná bodu 7.2.5, navíc je třeba chránit integritu procesu ověřování čerstvosti vstupních dat a zároveň zajistit korektní generování keksíků nebo klíče relace.

7.2.7 Utajení obsahu komunikace mezi stranami A a B

Cílem je utajení obsahu komunikace prováděné mezi stranami A a B. Při použití symetrické kryptografie je nutné na straně B ukrýt hodnotu sdíleného klíče používaného pro šifrování výstupní komunikace a zároveň zajistit použití dešifrovacích funkcí pouze pro data, která nebyla podvržena útočníkem. V případě použití asymetrické kryptografie je nutné zajistit ochranu kódu strany B proti podvržení hodnoty veřejného klíče A, utajit hodnotu privátního klíče strany B a zajistit použití dešifrovacích funkcí strany B pouze pro data, která nebyla podvržena útočníkem. Použité šifrovací klíče mohou být dlouhodobě sdílené nebo mohou být dynamicky ustanovené pro každou relaci, případně může jít o kombinaci obou typů. Při použití klíče relace je třeba řešit problém ukrytí jeho hodnoty a korektnosti použití šifrovacích funkcí využívajících jeho hodnotu.

Cíle uvedené v bodech 7.2.1 až 7.2.7 shrnují požadované principy protokolu, který by zajišťoval komunikaci dvou stran A a B pro případ, kdy se pouze strana A nachází v bezpečném prostředí. Z výčtu výše uvedených problémů se jeví jako vhodnější použití symetrické kryptografie, ale pouze v případě existence způsobu ukrytí hodnoty použitého symetrického klíče. Možné řešení tohoto problému nabízí implementace šifrovacího algoritmu s využitím WBACR AES pro potřebu dlouhodobě sdílených šifrovacích klíčů. Klíče relace není možno tímto způsobem implementovat a jejich použití je omezeno na standardní implementaci šifrovacích algoritmů. Utajení hodnoty a integrity dynamicky ustanovených klíčů relace, které jsou pouze dočasně umístěny v paměti strany B, je tedy obtížnější problém. Z tohoto důvodu se jeví jako nevhodné použití asymetrické kryptografie, neboť ve většině případů je nutné z důvodu efektivnosti šifrování provést šifrování dat symetrickým šifrovacím klíčem a pouze jeho hodnotu pak zabezpečit asymetricky. Strana B je pak nucena tento klíč používat stejně jako v případě klíče relace. Možné uplatnění by měla asymetrická kryptografie v případě autentizace stran A a B. Ovšem výhody plynoucí ze vzájemného vztahu veřejného a privátního klíče asymetrické kryptografie lze rovněž simulovat pomocí WBACR AES díky možnosti rozdělit předpočtené tabulky WBACR AES na šifrovací a dešifrovací část.

7.3 Rozbor zranitelnosti protokolu 3P-ISO9798-2

Zde uvedený rozbor zranitelnosti protokolu 3P-ISO9798-2 se zabývá pouze zranitelností protokolu vyplývající z umístění jedné ze stran do potenciálně nebezpečného prostředí. nikoli zranitelností protokolu v případě, že se obě komunikující strany nacházející v bezpečných prostředích zajišťujících integritu a utajení dat procesu autentizace.

Předpokládejme, že pro vzájemnou autentizaci dvou stran A a B je použit protokol 3P-ISO9798-2, který je implementován bez jakéhokoli záměrného ohledu na fakt, že softwarový agent odpovědný za řízení procesu autentizace strany B se nachází v prostředí pod kontrolou útočníka. Útočník má možnost provádět všechny útoky uvedené v kapitole 3.2, především krokovat výkon kódu softwarového agenta strany B až na úroveň instrukcí procesoru a číst nebo modifikovat paměť a kód softwarového agenta strany B. Cílem útočníka je narušit vykonávání autentizačního protokolu takovým způsobem, aby došlo k přijetí autentizace útočníka vydávajícího se za stranu B vůči straně A nebo naopak. Vícekrokový protokol 3P-ISO9798-2 založený na symetrické kryptografii byl vybrán záměrně z toho důvodu, že poskytuje oboustrannou autentizaci vhodnou pro potřeby DRM.

7.3.1 Protokol 3P-ISO9798-2 pro vzájemnou autentizaci

Strany A a B dlouhodobě sdílí předem ustanovený symetrický šifrovací klíč K_{AB}. Strana A, resp. B je jednoznačně identifikována hodnotou id_A, resp. id_B. Zápis $E_{K_{AB}}$ (data) značí šifrování hodnoty data pomocí klíče K_{AB} a vhodného symetrické šifrovacího algoritmu.

1. B zahajuje protokol
   - 1.1. B generuje náhodný keksík N_B a uschová jej;
   - **1.2. $B \rightarrow A \{id_B, N_B\}$.**
2. A zpracovává $\{id_B, N_B\}$;
   - 2.1. A nalezne symetrický klíč K_{AB} sdílený A a B;
   - 2.2. A generuje náhodný keksík N_A a uschová jej;
   - **2.3. $A \rightarrow B\{ id_A, E_{K_{AB}} (N_A, N_B, id_B)\}$.**
3. B zpracovává $\{ id_A, E_{K_{AB}} (N_A, N_B, id_B)\}$;
   - 3.1. B nalezne symetrický klíč K_{AB} sdílený A a B;
   - 3.2. B dešifruje zprávu $\{ id_A, E_{K_{AB}} (N_A, N_B, id_B)\}$;
   - 3.3. B ověří korektnost příjemce dle id_B;
   - 3.4. B ověří čerstvost zprávy porovnáním N_B;
   - *3.5. A je nyní autentizován vůči B;*
   - **3.6. $B \rightarrow A\{ E_{K_{AB}} (N_B, N_A)\}$.**
4. A zpracovává $\{ E_{KAB} (N_B, N_A)\}$;
   - 4.1. A dešifruje zprávu $\{ E_{K_{AB}} (N_B, N_A)\}$;
   - 4.2. A ověří čerstvost zprávy porovnáním N_A;
   - 4.3. A ověří odesílatele zprávy porovnáním N_B;
   - *4.4. B je nyní autentizován vůči A.*

7.3.2 Zranitelnost protokolu 3P-ISO9798-2

V následujícím rozboru je pro konkrétní typ útoku uvedeno, k jakému typu úspěšné autentizace útočníka může dojít a zda provedení útoku vyžaduje modifikaci kódu softwarového agenta příslušné strany. Útočník má tři hlavní směry, kterými se může vydat pro dosažení autentizace vůči vybrané straně.

7.3.2.1 Zisk autentizačních informací

První způsobem je zisk všech privátních informací sloužících k prokázání identity strany, za kterou se útočník chce vydávat a s jejich využitím provést následně kroky definované v autentizačním protokolu ve své vlastní režii. Pokud je autentizace založena pouze na znalosti těchto informací, bude útok po jejich zisku úspěšný.

V protokolu 3P-ISO9798-2 je autentizační informací dlouhodobě sdílený symetrický klíč K_{AB}, který je použit modulem B pro dešifrování a šifrování zpráv v krocích 3.2 a 3.6. Pokud je zde klíč K_{AB} standardně použit spolu se vstupnímu daty jako parametr knihovních funkcí Encrypt() a Decrypt()[1], útočník s pomocí debuggeru nastaví místo přerušení programu na vstupní body těchto funkcí a vyčká na jejich zavolání. Přímým čtením vstupních parametrů získá hodnotu klíče K_{AB} při jeho prvním použití. Pokud je místo volání knihovních funkcí umístěn jejich kód přímo vložen do autentizačního kódu (za cenu zvýšení velikosti softwarového agenta strany B), je útočník nucen krokovat výkon autentizačního kódu, zisk hodnoty klíče je však principiálně stejný, neboť jeho hodnota se objeví nechráněné podobě v paměti softwarového agenta.

V případě znalosti hodnoty klíče K_{AB} se může útočník autentizovat jako strana A vůči straně B a naopak. Útočník nemusí měnit kód softwarového agenta strany B, kromě znalosti použitého autentizačního protokolu nejsou vyžadovány žádné dodatečné podmínky pro úspěšný útok.

7.3.2.2 Modifikace řízení procesu autentizace strany B

Druhou způsobem je modifikace kódu provádějícího řízení procesu autentizace v těle softwarového agenta strany, vůči kterému se chce útočník autentizovat takovým způsobem, aby modul *vyhodnotil* proběhlou autentizaci jako korektní i v případě, že přijaté zprávy protokolu korektní nebudou. Chybného vyhodnocení autentizace může být docíleno mnoha způsoby, mezi které patří přímá změna datové proměnné nesoucí příznak, zda autentizace proběhla úspěšně, modifikace kódu takovým způsobem, aby došlo k odstranění ochrany proti útoku přehráním dříve zachycené komunikace nebo modifikace generátoru náhodných čísel takovým způsobem, který nezaručuje kryptografickou náhodnost výstupních hodnot a umožňuje použít útok založený na této slabosti.

Nejméně náročným zásahem bude pravděpodobně modifikace příznaku ustanoveného v kroku 3.4 takovým způsobem, aby vždy odpovídal korektně proběhlé autentizaci. Toho lze docílit nahrazením instrukce porovnání instrukcí nastavující příznak. Nezávisle na opravdovém průběhu autentizačního protokolu tak bude vždy útočník autentizován vůči straně B jako strana A. Autentizace útočníka jako strany B vůči straně A není možná, neboť útočník nemá možnost provést výše uvedenou modifikaci

[1] Knihovní funkce provedou šifrování a dešifrování zvoleným algoritmem.

softwarovému agentovi strany A. Útočník musí měnit kód softwarového agenta strany B v rozsahu jednotek instrukcí, kromě znalosti použitého autentizačního protokolu nejsou vyžadovány žádné dodatečné podmínky pro úspěšný útok. Výhodou tohoto typu útoku je nezávislost na použitých kryptografických technikách autentizačního protokolu.

Protokol využívá pro zajištění čerstvosti vyměňovaných zpráv kryptograficky náhodné číslo (keksík), jehož generování v kroku 1.1 vyžaduje kryptograficky kvalitní generátor. Útočník se může pokusit o jeho modifikaci takovým způsobem, aby generované hodnoty neměly vlastnost kryptograficky náhodných čísel a umožnit tím útok pomocí opakovaného přehrání zachycené komunikace. V extrémním případě může generátor vracet stále stejnou hodnotu modifikací instrukcí přiřazení výsledku generování na instrukci přiřazení hodnoty zvolené útočníkem. Pro úspěšnou autentizaci útočníka vůči straně B namísto strany A musí útočník měnit kód softwarového agenta strany B v rozsahu jednotek instrukcí, kromě znalosti použitého autentizačního protokolu nejsou vyžadovány žádné dodatečné podmínky pro úspěšný útok. Autentizace vůči modulu A není tímto způsobem možná, neboť modul A má funkční část ověřující čerstvost zpráv.

Autentizace útočníka jako strany A vůči straně B lze dosáhnout nahrazením původní hodnoty klíče K_{AB} hodnotou K_{AB}', která je známa útočníkovi. Pomocí statické nebo dynamické analýzy nalezne útočník v těle softwarového agenta polohu hodnoty klíče K_{AB} a trvale ji zamění za hodnotu K_{AB}'. Proces autentizace útočníka vůči straně B je pak vyhodnocen jako úspěšný podobně jako v případě, kdy útočník získá hodnotu klíče K_{AB}. Vzhledem k nutnosti modifikace kódu je vhodnou obranou kontrola integrity kódu softwarového agenta.

7.3.2.3 Využití autentizačních funkcí strany B

Třetím způsobem pro autentizaci útočníka vůči straně A namísto strany B je zneužití autentizačních funkcí softwarového agenta strany B. Útočník podvrhne vybraná data ve vhodném místě výkonu softwarového agenta strany B takovým způsobem, že dojde k přípravě zpráv potřebných pro autentizaci útočníka. Část kódu obsahující požadované funkce může být ze softwarového agenta útočníkem vyjmuta a umístěna do softwarového agenta vytvořeného útočníkem. Tímto způsobem lze využít funkčnosti šifrování dat klíčem ukrytým v těle softwarového agenta a podobně.

Útočník s pomocí statické a dynamické analýzy identifikuje počátek a konec kódu pokrývající funkčnost kroku 3. Tuto část kódu pak umístí do vlastního softwarového agenta, který se následně může úspěšně autentizovat vůči straně A jako softwarový agent strany B. Tento typ útoku bude významný v případě, že hodnota klíče K_{AB} nebude snadno získatelná způsobem popsaným v 7.3.2.1.

7.3.3 Závěr rozboru zranitelnosti protokolu 3P-ISO9798-2

Implementace protokolu 3P-ISO9798-2 standardním způsobem je naprosto nedostatečná z hlediska zabezpečení proti útočníkovi, který má možnost plné kontroly nad softwarovým agentem strany B. Navrhované útoky se neomezují pouze na protokol 3P-ISO9798-2. Jejich varianty mohou být použity proti obecnému autentizačnímu protokolu, jehož implementace nepředpokládá přístup útočníka k softwarovému agentovi provádějícími autentizační proces jedné ze stran. Při použití asymetrické kryptografie lze částečně odstranit problém utajení autentizační informace strany A, neboť hodnota

veřejného klíče A nemusí být utajována. Zároveň však vzniká problém ochrany veřejného klíče A před jeho nahrazením jiným veřejným klíčem, ke kterému vlastní privátní část útočník.

Výsledkem rozboru jsou následující doporučení, které by měla implementace autentizačního mechanismu splňovat:

- Výsledkem autentizačního protokolu nesmí být na straně B pouze binární rozhodnutí o přijetí autentizace.
- Autentizační informace strany B umístěné v těle softwarového agenta nesmí být snadno čitelné při použití statické nebo dynamické analýzy.
- Kód softwarového agenta by měl obsahovat prostředky pro kontrolu integrity a reakci na modifikaci.
- Mechanismy zajišťující čerstvost vyměňovaných zpráv musí být obzvlášť robustně zajištěny proti modifikaci.
- Kód softwarového agenta strany B by měl být co nejlépe chráněn proti vytvoření mentálního modelu pro zamezení extrakce částí kódu obsahující požadované funkce.
- Softwarový agent strany B by neměl obsahovat funkci generující zprávy zaměnitelné za zprávy pocházející od strany A ani v případě, že útočník manipuluje se vstupními daty této funkce.

7.4 Protokol SEAUT (SEcure AUthenticated Transmision)

Na základě rozboru slabých míst protokolu P3-ISO9798-2 je navržen protokol SEAUT, který si klade za cíl zjištěná slabá místa odstranit. Protokol SEAUT využívá čtyři dlouhodobě sdílené šifrovací klíče pro symetrickou kryptografii. Tyto klíče jsou předem ustanovené mezi stranami A a B. Jeden pár je použit pro autentizační část a druhý pro transportní část protokolu. Použití klíčů je poněkud „netradiční“. Přestože se jedná o klíče používané pro symetrickou kryptografii, s pomocí WBACR AES je dosaženo stavu, kdy strana B může používat jeden z klíčů v každém páru pouze pro šifrování a druhý pouze pro dešifrování. Strana B tak ve výsledku může dvěma klíči šifrovat a dvěma klíči dešifrovat. Strana A má toto rozdělení inverzní vzhledem k rozdělení strany B, tedy klíč určený pro šifrování resp. dešifrování stranou B bude klíčem určeným pro dešifrování resp. šifrování stranou A. Toto rozdělení je předem dané a trvalé po celou dobu sdílení klíčů stranami A a B. Strana A má sdílené klíče realizovány formou datového pole obsahující jejich přímou hodnotu. Strana B formou WBACR AES tabulek takovým způsobem, že pro klíč určený k šifrování je přítomna pouze šifrovací část tabulek WBACR AES, analogicky pro klíč určený k dešifrování. Strana A sice může principiálně použít svůj klíč K určený k dešifrování i pro šifrování, strana B však nebude schopna takto zašifrovanou zprávu dešifrovat, protože pro klíč K vlastní pouze „šifrovací“ část WBACR AES tabulek. Označení zkratek použitých klíčů je vztaženo vzhledem k straně B, neboť zde je dělení klíčů díky WBACR AES jednoznačně dané a nemůže dojít k záměně „šifrovacího“ klíče za „dešifrovací“ a naopak.

Výsledkem autentizační části pro stranu A je binární rozhodnutí o korektnosti proběhlé autentizace a ustanovení základní hodnoty dynamického klíče relace. Strana A (poskytovatel služby) se zde může rozhodnout, zda umožní straně B využití nabízené

služby. Strana B provádí binární rozhodnutí o korektnosti proběhlé autentizace pouze volitelně, neboť toto rozhodnutí může být útočníkem snadno manipulováno. Korektně proběhlá autentizace vede k ustanovení stejné základní hodnoty dynamického klíče relace jako u strany A, nekorektní autentizace vede k chybné hodnotě. Klíč relace je použit pro zajištění čerstvosti vyměňované komunikace během transportní části protokolu. Mění po každé odeslané zprávě a slouží tak bez potřeby dodatečných mechanismů (keksíků) k ochraně zpráv proti útoku přehráním. Utajení vyměňovaných dat v transportní části je zajištěno pomocí dvou dlouhodobě sdílených transportních klíčů používaných obdobným způsobem jako autentizační klíče. Použití klíče relace neslouží k utajení dat, pouze zajišťuje s pomocí autentizačních klíčů čerstvost komunikace. Pokud nedostačuje utajení poskytované pouze transportními klíči, lze použít výpočetně náročnější protokol SEAUT2, využívající klíč relace i pro utajení komunikace. Použití protokolu SEAUT2 je vhodné v situaci, kdy mají přenášená data hodnotu i po odhadované době na vyzrazení transportních klíčů ukrytých ve WBACR AES tabulkách na straně B.

Použité označení:

id_A … Jednoznačná identifikace strany A.

id_B … Jednoznačná identifikace strany B.

$V(id_B)$ … Hodnota proměnné obsahující uloženou hodnotu id_B na straně B.

N_A … Kryptograficky náhodné číslo generované stranou A.

N_B … Kryptograficky náhodné číslo generované stranou B.

$V(N_B)$ … Hodnota proměnné obsahující uloženou hodnotu N_B na straně B.

KE_A … Dlouhodobě sdílený autentizační klíč mezi stranami A a B. Na straně B je klíč realizován částí WBACR AES tabulek použitelných pouze pro šifrování. Na straně A je klíč realizován formou datového pole používaného pouze pro dešifrování.

KD_A … Dlouhodobě sdílený autentizační klíč mezi stranami A a B. Na straně B je klíč realizován částí WBACR AES tabulek použitelných pouze pro dešifrování. Na straně A je klíč realizován formou datového pole používaného pouze pro šifrování.

KE_T … Dlouhodobě sdílený transportní klíč mezi stranami A a B. Na straně B je klíč realizován částí WBACR AES tabulek použitelných pouze pro šifrování. Na straně A je klíč realizován formou datového pole používaného pouze pro dešifrování.

KD_T … Dlouhodobě sdílený transportní klíč mezi stranami A a B. Na straně B je klíč realizován částí WBACR AES tabulek použitelných pouze pro dešifrování. Na straně A je klíč realizován formou datového pole používaného pouze pro šifrování.

K_R … Klíč relace ustanovený během autentizační části protokolu. Na straně A i B je realizován formou datového pole.

E_{KX_X} (data) … Operace šifrování dat pomocí klíče KX_X. Na straně B značí šifrování pomocí WBACR AES, na straně A šifrování standardní verzí algoritmu.

D_{KX_X} (data) … Operace dešifrování dat pomocí klíče KX_X. Na straně B značí dešifrování pomocí WBACR AES, na straně A dešifrování standardní verzí algoritmu.

X_{K_R} (data)	… Operace xor hodnoty K_R nad daty. V případě, že délka dat přesahuje délku hodnoty K_R, probíhá operace xor opakovaně po blocích délky hodnoty K_R.
H(data)	… Operace jednosměrné hash funkce nad daty.
request	… Data reprezentující požadavek strany B na stranu A.
response	… Data reprezentující odpověď strany A na požadavek strany B.
x | y	… Operace zřetězení argumentů x a y.

1. **B zahajuje autentizační část protokolu SEAUT**
   - 1.1. B generuje náhodné číslo N_B a uloží je do $V(N_B)$;
   - **1.2. $B \rightarrow A$ {id_A, id_B, "START", N_B }**.
2. A zpracovává {id_A, id_B, "START", N_B };
   - 2.1. A nalezne symetrický klíč KD_A dlouhodobě sdílený mezi A a B;
   - 2.2. A generuje náhodné číslo N_A a uschová jej;
   - **2.3. $A \rightarrow B$ { id_A, id_B, E_{KD_A} (N_A, N_B, id_B)}**.
3. B zpracovává { id_A, id_B, E_{KD_A} (N_A, N_B, id_B)};
   - 3.1. B nalezne symetrický klíč KD_A dlouhodobě sdílený A a B;
   - 3.2. B dešifruje { E_{KD_A} (N_A, N_B, id_B)};
   - 3.3. Nepovinná kontrola:[2]
     - 3.3.1. B ověří korektnost příjemce porovnáním id_B a $V(id_B)$;
     - 3.3.2. B ověří čerstvost porovnáním N_B a $V(N_B)$;
     - *3.3.3. A je nyní autentizován vůči B;*
   - 3.4. B vytvoří klíč relace $K_R = H(N_A \mid N_B \mid id_B \mid V(N_B) \mid V(id_B))$;
   - 3.5. B nalezne symetrický klíč KE_A dlouhodobě sdílený A a B;
   - 3.6. **$B \rightarrow A${ id_A, id_B, E_{KE_A} (N_B, N_A)}**.
4. A zpracovává { id_A, id_B, E_{KE_A} (N_B, N_A)};
   - 4.1. A nalezne symetrický klíč KE_A dlouhodobě sdílený A a B;
   - 4.2. A dešifruje { E_{KE_A} (N_B, N_A)};
   - 4.3. A ověří čerstvost zprávy porovnáním N_A;
   - *4.4. B je nyní autentizován vůči A;*
   - 4.5. A vytvoří klíč relace $K_R = H(N_A \mid N_B \mid id_B \mid N_B \mid id_B)$;
5. **B zahajuje transportní část protokolu SEAUT**
   - 5.1. B nalezne transportní symetrický klíč KE_T dlouhodobě sdílený mezi A a B;
   - 5.2. B aktualizuje $K_R = H(K_R)$;
   - 5.3. B vytváří zprávu { id_A, id_B, X_{K_R} (E_{KE_T} (request))}, šifrování v režimu CBC s využitím K_R jako inicializačního vektoru;
   - 5.4. **$B \rightarrow A$ { id_A, id_B, X_{K_R} (E_{KE_T} (request))}**.
6. A zpracovává { id_A, id_B, X_{K_R} (E_{KE_T} (request))};
   - 6.1. A nalezne transportní symetrický klíč KE_T dlouhodobě sdílený mezi A a B;
   - 6.2. A aktualizuje $K_R = H(K_R)$;

[2] Útočníkem snadno modifikovatelná

6.3. A aplikuje operaci X_{K_R} na $\{X_{K_R}(E_{KE_T}(\text{request}))\} == \{E_{KE_T}(\text{request})\}$;
6.4. A dešifruje $\{E_{KE_T}(\text{request})\}$ v režimu CBC s využitím K_R jako inicializačního vektoru;
6.5. A zpracuje požadavek request;
6.5.1. Pokud request je roven "STOP", pokračuje bodem 9;
6.6. A nalezne transportní symetrický klíč KD_T dlouhodobě sdílený mezi A a B;
6.7. A aktualizuje $K_R = H(K_R)$;
6.8. A vytváří zprávu $\{ id_A, id_B, X_{K_R}(E_{KD_T}(\text{response}))\}$, šifrování v režimu CBC s využitím K_R jako inicializačního vektoru;
6.9. **$A \rightarrow B \{ id_A, id_B, X_{K_R}(E_{KD_T}(\text{response}))\}$**.

7. B zpracovává $\{ id_A, id_B, X_{K_R}(E_{KD_T}(\text{response}))\}$;
7.1. B nalezne transportní symetrický klíč KD_T dlouhodobě sdílený mezi A a B;
7.2. B aktualizuje $K_R = H(K_R)$;
7.3. B aplikuje operaci X_{K_R} na $\{X_{K_R}(E_{KD_T}(\text{response}))\} == \{E_{KD_T}(\text{response})\}$;
7.4. B dešifruje $\{E_{KD_T}(\text{response})\}$ v režimu CBC s využitím K_R jako inicializačního vektoru;
7.5. B zpracuje odpověď response
7.5.1. Pokud response je roven "STOP", pokračuje bodem 8;
7.6. B rozhoduje pokračování komunikace
7.6.1. B ukončuje komunikaci zprávou "STOP" v kroku 4.5 a provede krok 8.
7.6.2. B pokračuje v komunikaci krokem 4.5.

8. Závěr komunikace pro stranu B
8.1. B bezpečně odstraní hodnotu klíče K_R.

9. Závěr komunikace pro stranu A
9.1. A bezpečně odstraní hodnotu klíče K_R.

7.5 Rozbor protokolu SEAUT

Protokol SEAUT přidává k modifikaci autentizační části převzaté z protokolu P3-ISO9798-2 transportní část, sloužící k utajení a autentizaci komunikace vyměňované po úspěšném dokončení autentizační části. Autentizační část SEAUT je oproti P3-ISO9798-2 je modifikována použitím dvou autentizačních klíčů KE_A a KD_A, reprezentací klíčů KE_A a KD_A na straně B ve formě částečných WBACR AES tabulek a nahrazením binárního rozhodování o přijetí autentizace tvorbou klíče relace K_R využívaného v transportní části protokolu. Transportní část využívá pro utajení a autentizaci vyměňovaných dat dva transportní klíče KE_T a KD_T realizované na straně B formou částečných WBACR AES tabulek. Pro zajištění čerstvosti vyměňovaných dat je použita operace využívající klíč relace K_R, ustanovený během autentizační části a aktualizovaný po každé přijaté nebo odeslané zprávě. V následujícím textu budou rozebrány důvody pro navržení modifikací autentizační části a principy návrhu transportní části.

7.5.1 Autentizační klíče KE_A a KD_A

Protokol SEAUT používá dva sdílené autentizační klíče KE_A a KD_A namísto jednoho sdíleného autentizačního klíče K_{AB} v protokolu P3-ISO9798-2. Klíč K_{AB} je používán pro

zabezpečení všech autentizačních zpráv, na straně A i B je tedy použit pro režim šifrování i dešifrování. Naproti tomu klíče KE_A a KD_A jsou použity pouze „částečně". Klíč KE_A je stranou B používán pouze pro režim šifrování autentizační zprávy určené pro stranu A (krok 3.6), strana A používá klíč KE_A pouze pro režim dešifrování autentizační zprávy pocházející od strany B (krok 4.2). Klíč KD_A je stranou A používán pouze pro režim šifrování autentizační zprávy určené pro stranu B (krok 2.3), strana B používá klíč KD_A pouze pro režim dešifrování autentizační zprávy pocházející od strany A (krok 3.2). V případě, že hodnota klíčů KE_A a KD_A je různá, nemůže dojít k záměně zprávy pocházející od strany A za zprávu pocházející B a naopak ani v případě, kdy má útočník možnost podvrhovat vstup kódu[3] vytvářejícího autentizační zprávu. Při pokusu o takové podvržení dojde k dešifrování zprávy jiným klíčem. Záměně zpráv je běžně, při umístění stran A i B v bezpečném prostředí, bráněno vložením a kontrolou jednoznačné identifikace vytvářející a přijímající strany (id_A, id_B). Tato kontrola na straně B však může být v našem případě útočníkem snadno odstraněna, neboť strana B se nenachází v bezpečném prostředí. Útočník navíc může během tvorby zprávy na straně B prohodit hodnoty vkládaných jednoznačných identifikací. Pokud je použit pouze jeden autentizační klíč, výsledná zpráva určená pro stranu A bude zaměnitelná za zprávu pocházející od strany A. Užití dvou, „částečně" používaných autentizačních klíčů oběma typům útokům zabraňuje.

7.5.2 Reprezentace klíčů KE_A a KD_A formou částečných WBACR AES tabulek

Vyzrazení hodnot klíčů KE_A a KD_A vede, stejně jako v případě vyzrazení klíče K_{AB} protokolu P3-ISO9798-2, ke kompromitaci autentizačního procesu. Při použití standardní verze šifrovacích algoritmů útočník snadno získá hodnoty používaných klíčů inspekcí strany B. Pro ukrytí jejich hodnot a možnost bezpečného užití, jsou na straně B klíče realizovány formou WBACR AES tabulek. Tyto tabulky by měli poskytovat dostatečnou ochranu proti statické i dynamické analýze strany B. Vzhledem k „částečnému" použití autentizačních klíčů a možnosti rozdělit tabulky WBACR AES na šifrovací a dešifrovací část, je možné do kódu strany B umístit pouze šifrovací, resp. dešifrovací část WBACR AES tabulek pro klíč KE_A, resp. KD_A. Velikosti místa zabíraného tabulkami se sníží na polovinu, především však útočník nebude schopen ani po získání tabulek strany B vytvářet zprávy šifrované klíčem KD_A ani dešifrovat zprávy zašifrované klíčem KE_A. Tyto operace bude schopen provádět až po extrakci hodnoty ukrytého klíče, což by mělo být výpočetně obtížné.

7.5.3 Klíč relace K_R

V autentizační části protokolu SEAUT se strana A vůči straně B autentizuje schopností šifrovat klíčem KD_A, čerstvost je zajištěna prostřednictvím náhodné čísla N_B, generovaného stranou B. Na straně B je pouze nepovinná přímá kontrola korektnosti proběhlé autentizace (krok 3.3) z toho důvodu, že útočník snadno kontrolu modifikuje tak, aby i v případě nekorektních autentizačních zpráv strana B považovala proběhlou autentizaci za korektní. Tato nepovinná kontrola je zastoupena operací (krok 3.4), která neporovnává hodnoty autentizačních zpráv oproti očekávaným, ale použije je pro tvorbu klíče relace K_R. V případě, že hodnoty autentizační zpráv nebudou korektní,

[3] Softwarového agenta strany B.

vygenerovaný klíč relace bude neplatný. Útočník sice může modifikací nepovinné kontroly docílit pokračování protokolu, komunikace v transportní části však bude chybná. Hodnota dat ze vstupních zpráv nebude stranou B korektně získána a výstupní zprávy strany B budou chybně generovány.

Počáteční hodnota klíče relace je vytvořena pomocí kryptograficky jednosměrné hash funkce se vstupem skládajícím se z hodnot používaných během procesu autentizace. Jedná se o hodnoty N_A, N_B a id_B, získané dešifrováním zprávy v kroku 3.2 a „očekávané" korektní hodnoty $V(N_B)$ a $V(id_B)$ uložené na straně B. Hodnota N_A slouží jako náhodnostní základ pro utajení klíče relace, neboť všechny ostatní hodnoty jsou útočníkovi dostupné ze zachycené komunikace. Tato hodnota je generována v bezpečném prostředí strany A a lze tak u ní zajistit požadovanou délku i kvalitu. Zbylé hodnoty slouží pro zvýšení odolnosti autentizačního protokolu proti manipulaci strany B. Porovnání dešifrované hodnoty id_B a očekávané hodnoty $V(id_B)$ v kroku 3.3.1 je zastoupeno použitím obou těchto hodnot pro tvorbu klíče relace. Pokud je v kroku 3 přijata nekorektní zpráva, bude dešifrovaná hodnota id_B odlišná od hodnoty $V(id_B)$ a výsledný klíč relace bude odlišný od klíče, který by vznikl z korektní autentizační zprávy. Analogicky tomu bude v případě zastoupení kontroly čerstvosti v kroku 3.3.2 použitím hodnot N_B a $V(N_B)$ pro tvorbou klíče relace.

Je vhodné poznamenat, že tvorba klíče relace a jeho následné použití má význam pouze z důvodu umístění strany B v potencionálně nebezpečném prostředí. Strana A, vyhodnocující autentizační proces v bezpečném prostředí, používá standardní ověření očekávaných hodnot zpráv prostým porovnáním (krok 4.3). Strana B se vůči straně A autentizuje schopností šifrovat klíčem KE_A, čerstvost je zajištěna náhodným číslem N_A. Strana A (poskytovatel služby) tak má možnost odmítnout žádost o využití služby stranou B v případě chybné autentizace a protokol ukončit.

Klíč relace K_R není primárně určen pro utajení vyměňované komunikace v transportní části. Slouží hlavně k zajištění čerstvosti komunikace v transportní části a k zastoupení snadno odstranitelných kontrol korektnosti autentizace. Úroveň utajení komunikace je závislá na transportních klíčích KE_T a KD_T. Pokud dojde k jejich vyzrazení, lze předpokládat, že i autentizační klíče KE_A a KD_A budou vyzrazeny, neboť jsou reprezentovány na straně B stejným způsobem. Útočník tak bude schopen klíč relace rekonstruovat ze zachycené komunikace v autentizační části protokolu. Z tohoto důvodu není slabinou použití klíče relace jako inicializačního vektoru režimu CBC ani opakované xorováním k šifrovanému bloku dat, přestože oba dva způsoby jsou napadnutelné publikovanými útoky vedoucími k zisku hodnoty použitého klíče.

V případě, že je vyžadováno zvýšení bezpečnosti utajení nad úroveň poskytovanou transportními klíči, je třeba použít výpočetně náročnější protokol SEAUT2, který využívá klíč relace pro šifrování vyměňované komunikace. Způsob ustanovení hodnoty klíče relace zároveň zajišťuje, že hodnota nebude zpětně získatelná ze zachycené autentizační komunikace ani po vyzrazení autentizačních klíčů.

7.5.4 Transportní klíče KE_T a KD_T

Utajení a autentizace komunikace probíhající v transportní části SEAUT je zajištěna pomocí dlouhodobě sdílených transportních klíčů KE_T a KD_T. Jejich použití a reprezentace je analogická jako v případě autentizačním klíčům KE_A a KD_A. Oproti

autentizačním klíčům je možné použít výhody vstupního a výstupního kódování WBACR AES tabulek. Důležitou funkcí transportních klíčů je autentizace vyměňovaných zpráv. Útočník není schopen bez znalosti hodnoty klíče KD_T vytvořit zprávu, kterou strana B korektně dešifruje. Pokud dojde na straně B ke korektnímu dešifrování příchozí zprávy, tak tato zpráva byla vytvořena stranou A. Strana B neobsahuje funkčnost, která by umožnila vytvořit takovou zprávu ani při manipulaci vstupních hodnot útočníkem. Klíč KD_T je na straně B přítomen pouze ve formě části WBACR AES tabulek umožňující dešifrování. Příchozí zpráva však může pocházet z útočníkem zachycené předchozí komunikace. Čerstvost zprávy je zajišťována pomocí klíče relace K_R.

7.5.5 Čerstvost komunikace v transportní části pomocí K_R

Způsob použití klíče relace K_R je motivován snahou odstranit porovnávací kontrolu korektnosti autentizace a čerstvosti příchozích dat. Pokračování protokolu založené na binárním rozhodnutí na základě porovnávací kontroly je útočníkem snadno odstranitelné. Porovnávací kontrola je tedy v obou případech nahrazena „implicitní“ formou kontroly. Každý bit vstupních a výstupních dat je v krocích 5.3 a 7.3 „xorován“ s hodnotou klíče relace. Pokud bude hodnota klíče relace chybná, bude s pravděpodobností ½ chybný i výsledný bit dat. Klíč relace, který by byl neměnný po celou dobu jedné relace protokolu SEAUT by zajišťoval čerstvost pouze vzhledem ke zprávám pocházejícím z jiné relace. Neměnný klíč K_R by sloužil pouze k zajištění čerstvosti zpráv mezi různými relacemi. Metody zajištění čerstvosti zpráv v rámci relace pomocí čítače, časového razítka nebo náhodného čísla jako keksíku, se vzhledem k snadné manipulaci útočníkem nejeví jako vhodné. Namísto binárního rozhodnutí na základě porovnávání očekávaných hodnot keksíku je opět použita „implicitní“ kontrola klíčem relace. Klíč relace je pomocí jednosměrné hash funkce aktualizován na novou hodnotu po každé přijaté nebo odeslané zprávě. Opakované zpracování stejné vstupní zprávy tak nevede ke stejným hodnotám dešifrovaného obsahu a je ztížena možnost podvrhnout zprávu zachycenou v rámci stejné relace. Použití dostatečně kvalitní jednosměrné hash funkce zajišťuje, že útočník nebude schopen ze získané aktuální hodnoty klíče relace odvodit hodnoty předešlé. Vhodná metoda aktualizace je uvedena v 7.9.

Je vhodné poznamenat, že přijetí i jediné zprávy mimo očekávanou posloupnost vede k nekorektnímu pokračování celého zbytku protokolu bez možnosti obnovy. Pro korektní pokračování je nutné protokol ukončit a provést znovu od začátku. Snadno tak může být této vlastnosti zneužito k útoku způsobující odmítnutí služby (DoS). K duplikaci zprávy navíc může dojít i v důsledku přirozeného chování transportního média, například při možné duplikaci paketů v rámci protokolu TCP/IP. Je tedy vhodné použít ještě dodatečný mechanismus zabraňující neúmyslným duplikacím zpráv a DoS útokům vzhledem k útočníkovi, který nemá přístup ke straně B. Typ takového mechanismu je závislý na použitém komunikačním médiu a není protokolem SEAUT zohledněn.

7.6 Protokol SEAUT verze 2

Protokol SEAUT verze 2 (SEAUT2) rozšiřuje původní protokol SEAUT o ochranu vyměňovaných dat i v případě, že útočník získá hodnoty transportních klíčů KE_T a KD_T. Vyměňovaná data jsou šifrována dvojitě. Nejprve pomocí klíče KE_T nebo KD_T (v

závislosti na směru komunikace), výsledný šifrovaný text je znovu zašifrován pomocí aktuálního klíče relace K_R. Klíč relace je ustanoven s použitím algoritmu Diffie-Hellman a hodnota klíče tak není dostupná ze zachycené komunikace v autentizační části protokolu SEAUT2 ani v případě pozdější kompromitace autentizačních klíčů KE_A a KD_A. Pokud tedy útočník nemá možnost pozorovat paměť strany B obsahující hodnotu klíče K_R (dynamická inspekce), je vyměňovaná komunikace utajená i v případě (pozdější) kompromitace transportních klíčů.

Strana A se vůči straně B prokazuje schopností šifrovat klíčem KD_A, strana B se vůči straně A prokazuje schopností šifrovat klíčem KE_A a dešifrovat klíčem KD_A. Hodnoty α^x a α^y použité pro ustanovení klíče relace jsou zároveň využity jako keksík pro zajištění čerstvosti autentizačních zpráv. Vzhledem k použití klíče relace pro šifrování, a jeho změně po každé odeslané zprávě z důvodu zajištění čerstvosti vyměňovaných zpráv, jsou kladeny vyšší výpočetní nároky na transportní část protokolu SEAUT2 oproti původnímu protokolu SEAUT. Obzvláště v případě šifrovacích algoritmů s časově náročnou přípravou klíče může být změna hodnoty klíče před každou zprávu výrazně penalizující.

Protokol SEAUT2 tak lze doporučit v případě, že útočník nemá možnost provádět dynamickou inspekci strany B, má možnost dlouhodobě zachycovat vyměňovanou komunikaci mezi stranami A a B a vyměňovaná data mají hodnotu i po časovém intervalu, který je odhadován na vyzrazení klíčů KE_T a KD_T strany B.

Je vhodné poznamenat, že vzhledem k předpokladu, že útočník nemá možnost pomocí dynamické inspekce zjistit hodnotu klíče K_R, se může zdát zbytečné dvojí šifrování pomocí klíče relace a dlouhodobě sdíleného klíče KE_T nebo KD_T. Šifrování pomocí dostatečně dlouhého a kvalitního klíče relace s využitím kvalitního šifrovacího algoritmu by mělo poskytovat požadovanou bezpečnost samo o sobě. Útočník však může pomocí statické analýzy nalézt a modifikovat proces generování klíče relace, například volbou slabého exponentu x. Ustanovený klíč relace tak nebude dostatečně kvalitní a bude útočníkem obnovitelný ze zachycené autentizační části protokolu SEAUT2. Pro ztížení obnovy klíče relace i v tomto případě je použito šifrování zpráv vyměňovaných v krocích 2.3 a 3.8 pomocí autentizačních klíčů, takže útočník nemá bez dynamické inspekce nebo kompromitace jednoho z autentizačních klíčů přístup k hodnotě α^y.

1. **B zahajuje autentizační část protokolu SEAUT2**
   - 1.1. B generuje exponent x a uschová jej;
   - 1.2. B vypočte ($\alpha^x \bmod p$) a uschová;
   - 1.3. **$B \rightarrow A\{id_A, id_B,$ "START", $\alpha^x \bmod p\}$**.
2. A zpracovává { id_A, id_B, "START", $\alpha^x \bmod p$ };
   - 2.1. A nalezne symetrický klíč KD_A dlouhodobě sdílený mezi A a B;
   - 2.2. A generuje exponent y a uschová jej;
   - 2.3. A vypočte ($\alpha^y \bmod p$) a uschová;
   - 2.4. **$A \rightarrow B\{ id_A, id_B, E_{KD_A} (\alpha^x \bmod p, \alpha^y \bmod p)\}$**.
3. B zpracovává { id_A, id_B, $E_{KD_A} (\alpha^x \bmod p, \alpha^y \bmod p)$};
   - 3.1. B nalezne symetrický klíč KD_A dlouhodobě sdílený A a B;
   - 3.2. B dešifruje { $E_{KD_A} (\alpha^x \bmod p, \alpha^y \bmod p)$};

3.3. B ověří čerstvost porovnáním $(\alpha^x \mod p)$[4];
3.4. A je nyní autentizován vůči B;
3.5. B vypočte hodnotu $R = (\alpha^y)^x \mod p$;
3.6. B vypočte klíč relace K_R a bezpečně odstraní hodnotu x.
3.6.1. Varianta 1: $K_R = R$[5];
3.6.2. Varianta 2: $K_R = H(R \mid \alpha^x \mod p)$[6];
3.7. B nalezne symetrický klíč KE_A dlouhodobě sdílený A a B;
3.8. **$B \rightarrow A\{ id_A, id_B, E_{KE_A}(\alpha^y \mod p)\}$**.

4. A zpracovává $\{ id_A, id_B, E_{KE_A}(\alpha^y \mod p)\}$;
4.1. A nalezne symetrický klíč KE_A dlouhodobě sdílený A a B;
4.2. A dešifruje $\{ E_{KE_A}(\alpha^y \mod p)\}$;
4.3. A ověří čerstvost zprávy porovnáním $(\alpha^y \mod p)$;
4.4. B je nyní autentizován vůči A;
4.5. A vypočte hodnotu $R = (\alpha^x)^y \mod p$;
4.6. A vypočte klíč relace K_R a bezpečně odstraní hodnotu y.
4.6.1. Varianta 1: $K_R = R \mod p$;
4.6.2. Varianta 2: $K_R = H(R \mid \alpha^x \mod p)$;

5. B zahajuje transportní část protokolu SEAUT2
5.1. B nalezne transportní symetrický klíč KE_T dlouhodobě sdílený mezi A a B;
5.2. B aktualizuje $K_R = H(K_R)$;
5.3. **$B \rightarrow A\{ id_A, id_B, E_{K_R}(E_{KE_T}(request))\}$** v režimu CBC;

6. A zpracovává $\{ id_A, id_B, E_{K_R}(E_{KE_T}(request))\}$;
6.1. A nalezne transportní symetrický klíč KE_T dlouhodobě sdílený mezi A a B;
6.2. A aktualizuje $K_R = H(K_R)$;
6.3. A dešifruje $\{ E_{K_R}(E_{KE_T}(request))\}$ v režimu CBC;
6.4. A zpracuje požadavek request;
6.4.1. Pokud request je roven "STOP", pokračuje bodem 9;
6.5. A nalezne transportní symetrický klíč KD_T dlouhodobě sdílený mezi A a B;
6.6. A aktualizuje $K_R = H(K_R)$;
6.7. **$A \rightarrow B\{ id_A, id_B, E_{K_R}(E_{KD_T}(response))\}$** v režimu CBC.

7. B zpracovává $\{ id_A, id_B, E_{K_R}(E_{KD_T}(response))\}$;
7.1. B nalezne transportní symetrický klíč KD_T dlouhodobě sdílený mezi A a B;
7.2. B aktualizuje $K_R = H(K_R)$;
7.3. B dešifruje $\{ E_{K_R}(E_{KD_T}(response))\}$ v šifrovacím režimu CBC;
7.4. B zpracuje požadavek request;
7.4.1. Pokud request je roven "STOP", pokračuje bodem 8;
7.5. B aktualizuje $K_R = H(K_R)$;
7.6. B rozhoduje pokračování komunikace.
7.6.1. B ukončuje komunikaci zasláním zprávy "QUIT" v kroku 4.5 a provede krok 8;

[4] Útočník snadno odstraní, ale čerstvost komunikace je zajištěna pomocí klíčem relace
[5] Útočník musí pro napadení čerstvosti vůči B modifikovat hodnotu x
[3] Útočník musí pro napadení čerstvosti vůči B modifikovat hodnotu x a $(\alpha^x \mod p)$

7.6.2. B pokračuje v komunikaci krokem 4.5;

8. Závěr komunikace pro stranu B

8.1. B bezpečně odstraní hodnotu klíče K_R.

9. Závěr komunikace pro stranu A

9.1. A bezpečně odstraní hodnotu klíče K_R.

7.7 Útok na doplnění dat pro režim CBC („CBC padding“)

Otevřená data určená pro šifrování pomocí algoritmu AES musí mít celkovou délku rovnu násobku délky bloku (128 bitů). Běžné způsoby doplnění nastavují chybějící bajty na určitou hodnotu dle zvoleného systému, například na počet doplněných bajtů [PKCS#5]. S. Vaudenay prezentoval prakticky proveditelný útok [Va02], využívající postranního chybového kanálu. Pokud je strana A reprezentována hardwarovým tokenem[7] s umístěnými hodnotami klíčů, je orákulum požadované pro útok dostupné. Zprávy vyměňované v průběhu protokolu SEAUT tedy nemohou používat tento zranitelný způsob doplňování. Vzhledem ke specifickým vlastnostem šifrování pomocí WBACR AES nelze použít metody doplnění využívající derivované klíče [PKCS#5], neboť otevřená hodnota klíče není dostupná. Rovněž nelze použít metody, které brání výše uvedenému útoku použitím dešifrovací operaci pro poslední šifrovaný blok [KlRo02], neboť dešifrovací část nemusí být při oddělené distribuci WBACR AES tabulek dostupná. Použití různých forem doplnění pomocí integritních součtů vede ke výraznému zvětšení velikosti zprávy, což má negativní vliv na rychlost zpracování protokolu. Z výše uvedených důvodů a známé velikosti jednotlivých zpráv vyměňovaných v průběhu SEAUT bylo zvoleno pro konkrétní implementaci následující řešení:

a) Šifrovaná data zpráv v krocích 2.3 a 3.6 nejsou doplněna. Vyžaduje vhodné zvolení délky keksíků (64 bitů) a délky unikátního identifikátoru strany B (128 bitů). Strana A i B neprovádí žádné doplnění resp. odstranění.
b) Šifrovaná data vyměňovaná během transportní části protokolu krocích 5.4 a 6.9 oproti běžnému CBC režimu navíc podrobena opakované operaci XOR s klíčem relace K_R. Pokud není útočníkovi známa hodnota klíče K_R, není V. útok přímo proveditelný. Vlastnosti a síla „ochrany“ poskytované klíčem K_R však nebyly podrobně studovány.

Hledání vhodnějšího obecně použitelného doplnění při šifrovaní v režimu CBC pomocí WBACR AES nebylo provedeno a je předmětem dalšího studia. Je vhodné zvážit použitelnost vybraného doplnění při parciální distribuci WBACR AES tabulek (6.5.1.3) a při použití vstupního a výstupního kódování (7.8).

7.8 Vstupní a výstupní kódování při šifrovacím režimu CBC

Bezpečnost použití WBACR AES tabulek na straně B lze zvýšit zavedením vstupního a výstupního kódování. Toto kódování ztíží útočníkovi extrakci tabulek z kódu softwarového agenta, použití extrahovaných tabulek jako šifrovacího stroje nebo nahrazení jinými (viz kapitola 6). Zvolené vstupní kódování je postupně aplikováno na

[7] Ochranné rozhranní uvedené v kapitole 8.

vstupní data před jejich použitím jako vstupu WBACR AES. Čím větší množství kódu je vykonáno během aplikace kódování, tím obtížnější je pro útočníka jeho rekonstrukce. Analogicky pro odstranění výstupního kódování. Jako vhodné se jeví použít toto kódování u transportních klíčů KE_T a KD_T. Na data „request“ může být vstupní kódování aplikováno pozvolna během jejich přípravy pro šifrování. Analogicky pro data „response“, kdy může být výstupní kódování vzniklé po dešifrování odstraněno až během použití dat „response“, nikoli ihned po dokončení dešifrování. Popis kódování umístěného v softwarovém agentovi strany B tak může být rozložen na velkém prostoru. Autentizační klíče KE_A a KD_A naopak vhodnými kandidáty nejsou. Proces autentizace bude pravděpodobně probíhat v krátkém časovém úseku a aplikaci kódování na autentizační zprávy nelze odložit. Je vhodné poznamenat, že použití vstupního a odstranění výstupního kódování je záležitostí pouze strany B, neboť vzniká následkem použití WBACR AES tabulek. Šifrovaná data vyměňovaná mezi stranami A a B nejsou použitým kódováním jakkoli ovlivněna.

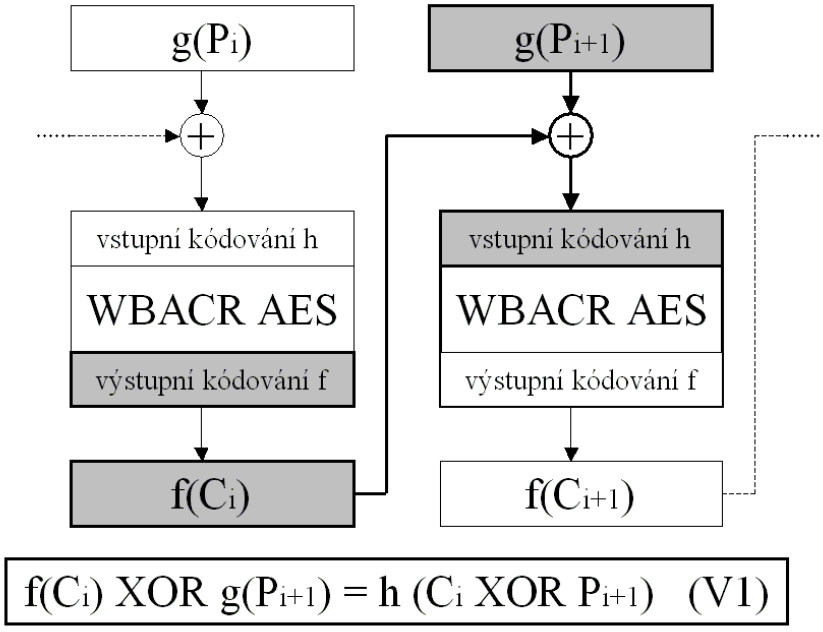
Pokud by byl pro šifrování použit režim ECB, na tvorbu vstupního a výstupního kódování by nebyly kladeny žádné dodatečné omezující požadavky. Platným kódováním by byla libovolná bijektivní funkce se vstupem i výstupem o velikosti n bitů (v našem případě n = 8). Použití režimu CBC si však vynucuje zavedení jistých omezení. Při šifrování dochází k aplikaci šifrované podoby předešlého bloku dat pomocí funkce XOR na následující nešifrovaný blok před samotným použitím šifrovacího algoritmu. Pokud by bylo vstupní i výstupní kódování zvoleno stejně jako pro ECB, výsledkem šifrování několika bloků dat by nebyla stejná šifrovaná data, jako v případě bez použití kódování. Naivním řešením tohoto problému je odstranění výstupního kódování mezi šifrování jednotlivých bloků dle CBC. Následovala by aplikace funkce XOR na dekódovaný šifrovaný blok a následující nešifrovaný blok. Na výsledek by bylo aplikováno vstupní kódování a konečně provedeno šifrování pomocí WBACR AES tabulek. Toto řešení by však nepředstavovalo výrazné zvýšení bezpečnosti, neboť by odstranění výstupního a aplikace vstupního kódování muselo proběhnout na velmi malém prostoru a útočník by tak snadno získal jeho předpis. Navíc by došlo k výraznému zpomalení celého procesu šifrování. Proto je třeba zavést omezující podmínky na tvorbu vstupního a výstupního kódování tak, aby nebylo nutné výše uvedený postup provádět. Návrh takového omezení je znázorněn na obrázku 7.1, tučně je zvýrazněna relevantní část. Vyplývá z něj, jaké vlastnosti jsou požadovány po použitém vstupním a výstupním kódování. Splnění vztahu V1 lze vhodným generováním kódování zajistit. Funkce $\underline{f}$, $\underline{g}$, $\underline{h}$ jsou bijektivní a pro praktické použití zadány formou tabulky. Definiční obor i obor funkčních hodnot těchto funkcí jsou přirozená čísla z intervalu $DOM = \langle 0, 2^k-1 \rangle$, kde $\underline{k}$ je velikost argumentu použitého kódování (v bitech).

Algoritmus generování funkcí $\underline{f}$, $\underline{g}$, $\underline{h}$ splňujících V1 vychází z následujících úvah (uvedeno bez důkazu):

- Pro splnění vztahu V1 musí být splněn i méně obecný vztah V2 vzniklý volbou x = y
  $f(x)\ XOR\ g(x) = h(x\ XOR\ x) = h(0)$ (V2)
- Z podmínky bijekce funkce $\underline{h}$ a V1 plyne:
  $\forall m, n, m', n' \in DOM, \{m\ XOR\ n = m'\ XOR\ n'\} \Rightarrow$

=> { h(m XOR n) = h(m' XOR n') }
=> { f(m) XOR g(n) = f(m') XOR g(n') } (V3)

- Podmínka bijekce funkce f stanovuje, že:
  $\forall m, n \in DOM: \{ m != n \} => \{ f(m) != f(n) \}.$ (V4)
- Znalost funkční hodnoty h(s) a vztah V3 umožňuje stanovit omezující podmínku na rozsah možných hodnot, kterou musejí splňovat funkční hodnoty f(m) a g(n), pokud s = m XOR n:
  $\forall s, m, n \in DOM: (m \text{ XOR } n = s) => f(m) \text{ XOR } g(n) = h(s)$ (V5)
- Pomocí operace XOR a operandů z množiny $P = \{x \mid x = 2^p, p \in \{0, \ldots, k-1\}\}$ lze vyjádřit libovolné číslo z intervalu $<0, 2^k>$. Využitím této vlastnosti, vztahu V5, volby f(0) a h(0) a volby f(x) pro $\forall x \in P$ lze jednoznačně určit všechny funkční hodnoty funkce f.



Obr. 7.1: Vstupní a výstupní kódování pro šifrování v režimu CBC[8].

7.8.1 Generování vstupního/výstupního kódování v režimu CBC

Algoritmus pro generování funkcí f, g a h tak, aby byly použitelné pro potřeby vstupní a výstupního kódovaní probíhá v následujících krocích:

1. Zvolíme náhodně funkční hodnoty f(0) a h(0). Inicializujeme všechny hodnoty tabulek určující funkce f, g a h hodnotou UNASSIGNED. (UNASSIGNED $\notin$ DOM).
2. Využitím V1, f(0) a h(0) vypočteme g(0)

[8] Pro přehlednost je zobrazeno použití kódování f, g, h v případě, kdy celý vstupní i výstupní blok lze zakódovat jediným k bitovým kódováním. Pro konkrétní případ AES s délkou bloku 128 bitů lze díky vlastnostem funkce XOR a způsobu náhledu do ASMT tabulek v první rundě použít 16 různých 8 bitových (16 x 8 = 128) kódování $f_1, \ldots, f_{16}$ a k nim příslušné $g_1, \ldots, g_{16}$ a $h_1, \ldots, h_{16}$. Kódování f_i je použito pro i-tý bit výstupu, analogicky pro g_i a h_i. (viz. 6.5.1.4).

3. Zvolíme náhodně, ale v souladu s V4, funkční hodnoty f(s) pro
   $\forall s: s = 2^p \;\; p \in \{ 0, \ldots, k - 1\}$.
4. Pro všechny zvolené funkční hodnoty f(s) z bodu 3 využitím V1, f(s) a h(0) vypočteme g(s).
5. Pro všechny zvolené funkční hodnoty f(s) z bodu 3 využitím V1, f(s) a g(0) vypočteme h(s).
6. Využitím vztahu V5 a již vypočtených funkčních hodnot pro f(r) a h(s) vypočteme funkční hodnotu g(r XOR s) = f(r) XOR h(s). Z vypočtené hodnoty g(r XOR s) a vztahu V2 vypočteme funkční hodnoty pro f(r XOR s) a h(r XOR s). Bod 6 opakovaně iterujeme přes všechny[9] hodnoty $\underline{t} \in$ DOM, dokud $\exists t$: f(t) = UNASSIGNED.
7. Generujeme WBACR AES tabulky s použitím vstupního kódování $\underline{h}$ a výstupního kódování $\underline{f}$.
8. Před šifrováním v režimu CBC aplikujeme na vstupní bloky dat P_i funkci $\underline{g}$.
9. Provedeme kroky šifrovacího režimu CBC standardním způsobem.

V případě šifrování dle ECB je počet všech možných různých variant vstupních a výstupních kódování v obou případech **$[(2^k)!]$** pro kódování o velikosti $\underline{k}$ bitů. Navržená metoda generování $\underline{f}$, $\underline{g}$, $\underline{h}$ díky nutnosti splnit vztah V1 umožňuje vytvořit 2^k možných voleb pro f(0), 2^k možných voleb pro h(0), a $\underline{k}$ možností volby jednoznačně neurčitelných funkčních hodnot $\underline{f}$ v souladu s V4. Ostatní funkční hodnoty funkcí $\underline{f}$, $\underline{g}$ a $\underline{h}$ jsou plně determinovány. Celkové množství různých variant bijektivních kódování $\underline{f}$, $\underline{g}$ a $\underline{h}$ je rovno **$[2^k \times 2^k \times (2^k\text{-}1) \times \ldots \times (2^k\text{-}k))]$.** Pro 8bitové kódování (k = 8) asi 2^{70} různých variant. Vzhledem k vlastnostem funkce XOR lze stále v případě 128bitového bloku AES použít 16 nezávislých trojic odpovídajících kódování $\underline{f}$, $\underline{g}$, $\underline{h}$ (viz. 6.5.1.4).

Vzhledem k faktu, že předpis funkce $\underline{g}$ musí být obsažen v kódu softwarového agenta využívajícího šifrování pomocí WBACR AES, nemusí být nároky na bezpečnost vstupního a výstupního kódování tak velké, jako v případě interního kódování WBACR AES. Pro zachování celkové úrovně bezpečnosti použití WBACR AES v kódu softwarového agenta postačuje, aby obtížnost odstranění vstupního a výstupního kódování byla srovnatelná s obtížností zisku předpisu funkce $\underline{g}$ z kódu softwarového agenta. Zisk předpisu funkce $\underline{g}$ útočníkem vede snadno k určení $\underline{f}$ a $\underline{h}$.

Podrobnější rozbor dopadu této úpravy na bezpečnost vstupního a výstupního kódování je nad rámec této diplomové práce. Lze však říct, že počet prohledávaných možností při odstraňování vstupního a výstupního kódování útočníkem bude menší, než je absolutní počet uvedený výše. Relevantní rozbor bezpečnosti vstupního a výstupního kódování lze nalézt v [CEJO02], dále zvážení vlastností chování funkce XOR. Rozbor bezpečnosti navrhované úpravy může být námětem pro další studium. Dopad na bezpečnost bude mít i kvalita použitého generátoru náhodných čísel pro volbu funkčních hodnot f(0), h(0) a jednoznačně neurčitelných f(x). Vzhledem k nutnosti umístění předpisu funkce g v kódu softwarového agenta se jeví postačující použití kvalitního pseudonáhodného generátoru.

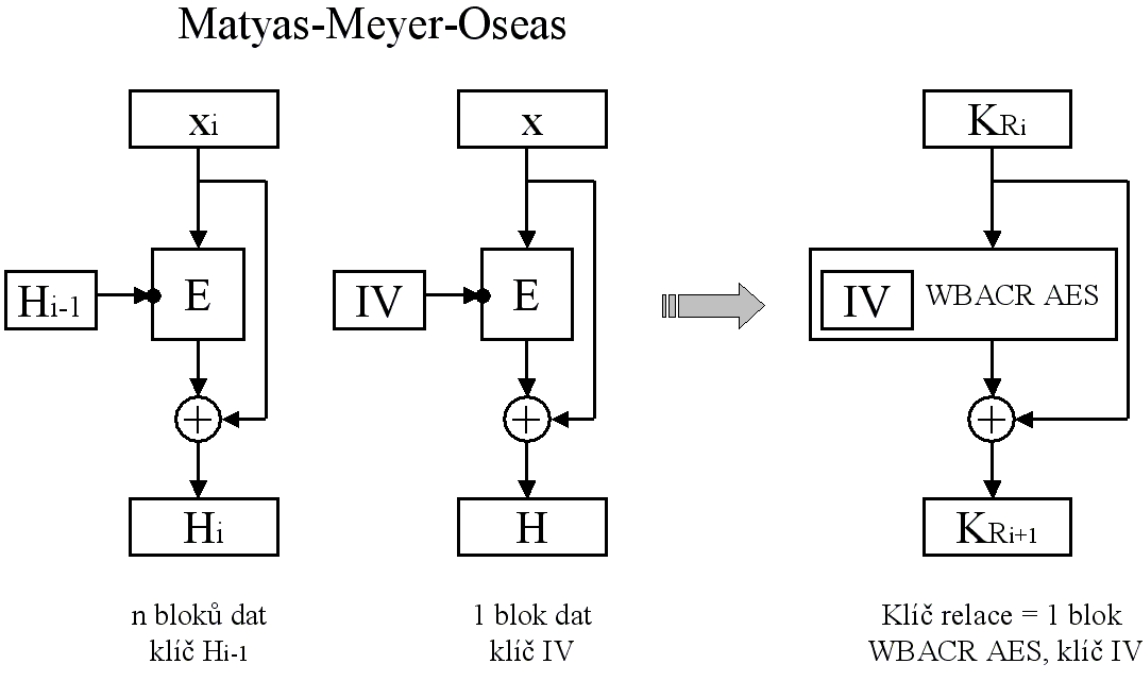
[9] Lze provést optimalizaci a procházet pouze relevantní hodnoty.

7.9 Metoda aktualizace klíče relace K_R

Aktualizace klíče relace K_R po každé zaslané nebo přijaté zprávě slouží pro zajištění čerstvosti probíhající komunikace. V ideálním případě by aktualizační proces měl zajišťovat následující vlastnosti:

a) Útočník není schopen odvodit předchozí hodnoty klíče relace z aktuální hodnoty.
b) Útočník není schopen odvodit následující hodnoty klíče relace z aktuální hodnoty.
c) Útočník není schopen smysluplně[10] modifikovat hash funkci.
d) Útočník není schopen smysluplně modifikovat aktuální hodnotu klíče relace.

Splnění vlastnosti a) je zajištěno, pokud aktualizační proces využívá kryptograficky jednosměrné hash funkce. Splnění vlastnosti b) je zajištěno, pokud je využita klíčovaná hash funkce a hodnotu použitého klíče lze utajit před útočníkem. Splnění vlastnosti c) lze zajistit transformací hash funkce pomocí technik mobilní kryptografie nebo obfuskace. Způsob dosažení vlastnosti d) není vzhledem ke způsobu použití klíče relace zřejmý a je předmětem dalšího studia.



Obr. 7.2: Metoda aktualizace klíče relace K_R.

S ohledem na splnění výše uvedené vlastnosti je navržena metoda aktualizace hodnoty klíče relace využívající WBACR AES. Metoda je založena na využití schématu Matyas-Meyer-Oseas (MMO) pro tvorbu hash funkce z blokového šifrovače, jak je znázorněno na obrázku 7.2. Vzhledem k fixní velikosti klíče relace dochází vždy pouze k jedinému cyklu MMO a hodnota použitého šifrovacího klíče je vždy rovna předem

[10] Smysluplnou modifikací je myšleno takové ovlivnění zpracování vstupní hodnoty, aby výsledná hodnota byla útočníkem využitelná pro útok na čerstvost komunikace.

dané hodnotě IV. Tato vlastnost umožňuje použít blokový šifrovač realizovaný formou šifrovací části WBACR AES tabulek.

Navržená metoda zajišťuje vlastnost kryptografické jednocestnosti nejméně na úrovni MMO s jedním cyklem. Použití WBACR AES umožňuje navíc utajit před útočníkem hodnotu klíče IV. Utajení hodnoty klíče IV zároveň poskytuje ochranu před generováním následných hodnot klíče relace[11]. Útočník není schopen provádět smysluplnou modifikaci hash funkce na úrovni hodnot WBACR AES tabulek. Zůstává však možnost modifikovat kód tabulky používající, proto je vhodné použít dodatečnou ochranu formou obfuskace. Samotný způsob konstrukce hash funkce neposkytuje ochranu aktuální hodnotě klíče relace. Použitím vstupního a výstupního kódování WBACR AES tabulek lze ztížit nalezení otevřené podoby klíče relace. Pouhá lokalizace místa aktualizace tak nevede k vyjevení aktuální hodnoty klíče relace, navíc je ztíženo použití extrahovaných WBACR AES tabulek.

Problém aplikace operace XOR na původní hodnotu klíče relace a její šifrovanou podobu lze řešit vhodnou volbou vstupního a výstupního kódování stejně jako v případě CBC šifrování (viz. 7.8). Možnost vhodné kombinace použitého výstupního kódování se vstupním kódováním WBACR AES tabulek pro klíče KE_T a KD_T tak, aby se klíč relace nacházel po celou dobu v chráněném tvaru a zároveň byla jeho hodnota použitelná způsobem vyžadovaným v kroku krok 5.3 (inicializační vektor, opakovaný XOR) protokolu SEAUT je předmětem dalšího studia.

[11] Útočník však stále může použít extrahované WBACR AES tabulky jako šifrovací stroj.

8 Projekt: Jednoduchá DRM architektura

Cílem projektu je demonstrovat možnosti, které nabízí programovatelný hardwarový token pro zvýšení úrovně ochrany běhu a dat softwarového agenta v potenciálně nebezpečném prostředí hostitele. Tato prostředí lze využít jako základ jednoduché DRM architektury. Zvýšení úrovně ochrany je dosaženo přesunem citlivých části softwarového agenta do prostředí programovatelného hardwarového tokenu. Projekt vytváří do jisté míry obecný, opakovaně použitelný nástroj, který tento přesun usnadňuje. Softwarový agent využívá přesunuté části prostřednictvím služeb hardwarového tokenu. Projekt umožňuje omezit množinu softwarových agentů, kteří mohou služby poskytované hardwarovým tokenem využívat. Toho je dosaženo prostřednictvím autentizace jednotlivých softwarových agentů vůči hardwarovému tokenu. Důležitou součástí projektu je tedy řešení problému, jak ukrýt a bezpečně používat autentizační informace na straně softwarového agenta, který zůstává v potenciálně nebezpečném prostředí.

Zvýšení bezpečnosti je zamýšleno především vzhledem k poskytovateli softwarového agenta. Vhodné použití projektu může zvyšovat bezpečnost i vzhledem k hostiteli, například omezením množiny softwarových agentů, kteří mohou využívat podpisový klíč nacházející se na hardwarovém tokenu. Primárně je však sledován záměr zvýšení důvěryhodnosti hostitelské platformy pro poskytovatele softwarového agenta.

8.1 Ochranného rozhranní

Zvýšení úrovně ochrany je dosaženo vytvořením rozhranní, které umožňuje snadný přesun citlivých částí kódu softwarového agenta do programovatelného kryptografického hardwarového tokenu. Přesunuté části kódu může po autentizaci softwarový agent prostřednictvím nepřímých požadavků využívat. Dynamicky vytvořeným, zabezpečeným komunikačním kanálem poskytuje vstupní data pro přesunutou část kódu. Hardwarový token požadavek vyhodnotí a v kladném případě zasílá zpět zpracovaná data. Kryptografický hardwarový token by měl poskytovat větší bezpečnost běhu a ochranu dat než nechráněný softwarový agent a projekt by měl umožňovat snadné využití této výhody.

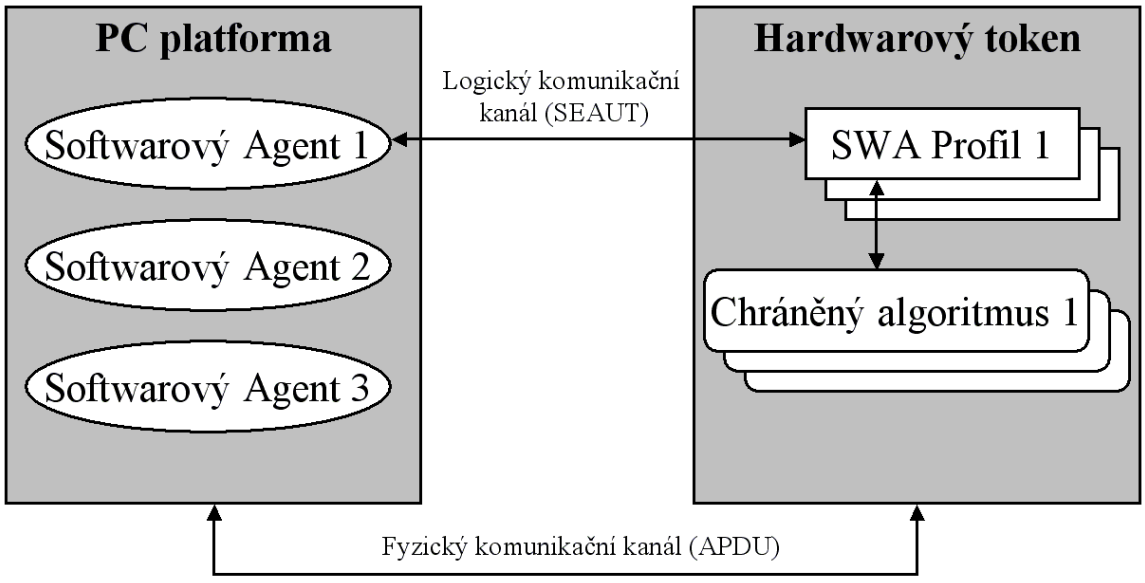
Přesunuté části kódu budou nazývány chráněnými algoritmy. Poskytování funkčnosti chráněných algoritmů je založeno na modelu klient-server popsaném v kapitole 6, klientem je zde softwarový agent, serverem hardwarový token. Chráněné algoritmy jsou softwarovým agentem využívány prostřednictvím komunikačního protokolu SEAUT popsaného v kapitole 7. Prostřednictvím protokolu SEAUT je dosažena autentizace jednotlivých softwarových agentů a utajení vyměňované komunikace. Pro ukrytí hodnot šifrovacích klíčů použitých pro autentizaci a komunikaci je využito alternativní implementace šifrovacího algoritmu AES ve formě WBACR AES tabulek. Hardwarový token může sloužit jako server více nezávislým softwarovým agentům, jejichž množina se může dynamicky měnit. Je umožněna průběžná aktualizace interních hodnot chráněných algoritmů. Projekt obsahuje podporu pro správu takového

prostředí prostřednictvím XML licencí. Výsledkem je programový skelet, který by měl být snadno přizpůsobitelný konkrétním požadavkům programátora.

Příloha A obsahuje popis ukázkového aplikace LightApp, která využívá ochranného rozhranní pro řízené využití Gaussova algoritmu pro výpočet dne, na který připadá v daném roce velikonoční neděle.

8.1.1 Požadovaná funkčnost:

1. Kód chráněného algoritmu je uložen a vykonáván ve výpočetním prostředí s požadovanou úrovní bezpečnosti.
2. Chráněný algoritmus může být používán pouze vybranou množinou autentizovaných softwarových agentů.
3. Chráněný algoritmus lze distribuovat do cílového bezpečného prostředí vzdáleně. Distribuce chráněného algoritmu probíhá nezávisle na distribuci bezpečného prostředí.
4. Interní hodnoty chráněného algoritmu lze ovlivňovat pomocí licence, kterou je možno vzdáleně distribuovat. Formát licence by měl být otevřený a obecně rozšířený standard.
5. Bezpečné výpočetní prostředí lze sdílet více chráněnými algoritmy a softwarovými agenty. Změnu množiny softwarových agentů a oprávnění pro využívání jednotlivých chráněných algoritmů lze provádět dynamicky a vzdáleně.
6. Obtížnost začlenění ochranného rozhranní je pro programátora co nejmenší. Programátor je odstíněn od implementačních záležitostí sloužících pro zajištění výše uvedených podmínek.



Obr. 8.1: Přehled ochranného rozhranní.

8.1.2 Typické scénáře užití ochranného rozhranní

- Potřeba zajistit integritu výkonu algoritmu, například při ověřování splnění podmínek daných DRM licencí, provádění autentizačního protokolu nebo ověřování digitálního podpisu dokumentu.
- Potřeba utajit implementaci algoritmu, například pro ochranu před zcizením myšlenky algoritmu nebo utajením podmínek, které vedou ke spuštění reakčního mechanismu u IDS systémů.
- Potřeba utajit data používaná algoritmem, například hodnoty šifrovacích a podpisových klíčů nebo uživatelových privátních informací.
- Potřeba zajistit integritu dat používaných algoritmu, například digitálně podepisovaného dokumentu, věrohodnému záznamu o použití algoritmu nebo dešifrované podoby zpracovávané DRM licence.
- Potřeba zajistit spuštění softwarového agenta pouze za přítomnosti hardwarového tokenu, například pro znesnadnění distribuce pirátských kopií software, kontroly pohybu softwarového agenta nebo umožnění jeho využití pouze autorizovanými osobami.
- Potřeba omezit množinu subjektů, které mohou algoritmus anebo data využívat, například pro zvýšení ochrany uživatelova podpisového klíče včetně jeho zneužití neoprávněným softwarovým agentem. Dalším příkladem je ochrana užití uživatelových privátních informací všeobecně nebo kontrola množství instalovaných softwarových agentů při množstevně orientované licenční politice software.

8.1.3 Omezující podmínky použití

Omezující podmínkou bránícími nasazení ochranného rozhranní může být relativně omezená rychlost zpracování požadavku a zaslání odpovědi. Snížení doby odezvy lze dosáhnout použitím výkonnějších čipových karet pro zrychlení provedení chráněného algoritmu a volbou čipové karty s rychlým hardwarovým akcelerátorem algoritmu AES. Dalšího zrychlení lze dosáhnout při přechodu na komunikačního rozhranní s vyšší propustností a větší povolenou délkou jednoho příkazu, než poskytují standardní APDU příkazy. Druhou omezující podmínkou je nutnost fyzické distribuce hardwarového tokenu. Díky platformové nezávislosti rozhranní JavaCard lze využít hardwarových tokenů 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.

8.1.4 Použité vývojové prostředky

Ochranné rozhranní je vyvinuto s použitím programovacích jazyků C++ a Java v operačním systému Microsoft Windows 2000. Principiálně je přenositelné i na jiné platformy, vzhledem k využití platformově závislých knihoven pro práci s hardwarovým tokenem však přenos může vyžadovat určité modifikace kódu. Část umístěná v hardwarovém tokenu je napsána v jazyce JavaCard verze 2.1.2, což je varianta jazyka Java s podobnou syntaxí, ale s výrazně omezeným množstvím dostupných objektů v základních balících. Jazyk JavaCard je určen především pro prostředí kryptografických čipových karet. Vývojovým prostředím této části byl Borland JBuilder verze 6, propojený s vývojovým prostředím Gemplus GemXpresso RAD III obsahující podporu pro čipové

karty firmy Gemplus. Kód softwarového agenta a generátor WBACR AES tabulek je napsán v jazyce C++ s použitím vývojového prostředí Microsoft Visual Studio 6. Visuální nadstavba určená pro prezentaci použití ochranného rozhranní využívá služeb knihovny Microsoft Foundation Class.

8.2 Hardwarový token

Hardwarovým tokenem může být libovolná kryptografická čipová karta podporující rozhranní JavaCard. Pro seznámení s principy fungování a používání technologie JavaCard jsou vhodné zdroje [Ru01, JC22API]. Při implementaci tohoto projektu byla použita kryptografická čipová karta Gemplus GXPPro-R3.

8.2.1 Hardwarový token x JavaCard applet

V předchozím textu byl pojem „hardwarový token" používán pro označení fyzicky zabezpečeného prostředku, se kterým lze komunikovat. Rozhranní JavaCard umožňuje nainstalovat na jeden fyzický hardwarový token pomocí vhodného rozhranní typu Open Platform [OP04] několik nezávislých aplikací nazývaných „applety". Jednotlivé „applety" jsou unikátně identifikované svým AID (Application IDentifier) a před použitím vybraného „appletu" je třeba provést pomocí speciálního příkazu jeho aktivaci. Není možné souběžně aktivovat více „appletů". Aktivovaný „applet" pak slouží jako příjemce všech příchozích příkazů. Vzniká tím poněkud nepřehledná situace, která by si vynutila změnu formulací typu „komunikace softwarového agenta s hardwarovým tokenem" na formulaci „komunikaci softwarového agenta s aktivním appletem na hardwarovém tokenu". Z důvodu přehlednosti je tedy zavedeno následující zjednodušení: Pokud je na jednom fyzickém hardwarovém tokenu umístěno více „appletů", je každý tento „applet" považován za „virtuální" hardwarový token. Na jednom fyzickém hardwarovém tokenu může být současně aktivní pouze jeden „virtuální" hardwarový token. Každý z nich je unikátně identifikován a pro potřeby tohoto projektu není podstatné, zda jsou umístěny na stejných nebo rozdílných fyzických hardwarových tokenech. V dalším textu tedy bude termín „hardwarový token" používán poněkud nepřesně pro aktivní „applet" na vybraném fyzickém hardwarovém tokenu, tedy pro „virtuální" hardwarový token. Před samotnou komunikací softwarového agenta s hardwarovým tokenem je tedy nutné nejprve vybrat fyzický hardwarový token a na něm aktivovat požadovaný „applet"[1].

8.2.2 Moduly hardwarového tokenu

Hardwarový token (dle výše zavedeného značení) obsahuje logické oddělené moduly pro správu profilů softwarových agentů, správu chráněných algoritmů, modul pro zpracování XML licencí a modul pro ustanovení bezpečného logického kanálu mezi hardwarovým tokenem a softwarovým agentem. Vzhledem k omezeným výpočetním a paměťovým prostředkům současných čipových karet nebylo možné provést plně objektově orientovaný návrh pro implementaci těchto modulů. V provedené implementaci tak nejsou dodrženy některé doporučované postupy objektového programování, například přístup k členským atributům tříd pouze přes vyhrazené metody a podobně.

[1] Realizováno voláním metody CRemoteAlgManager::Connect().

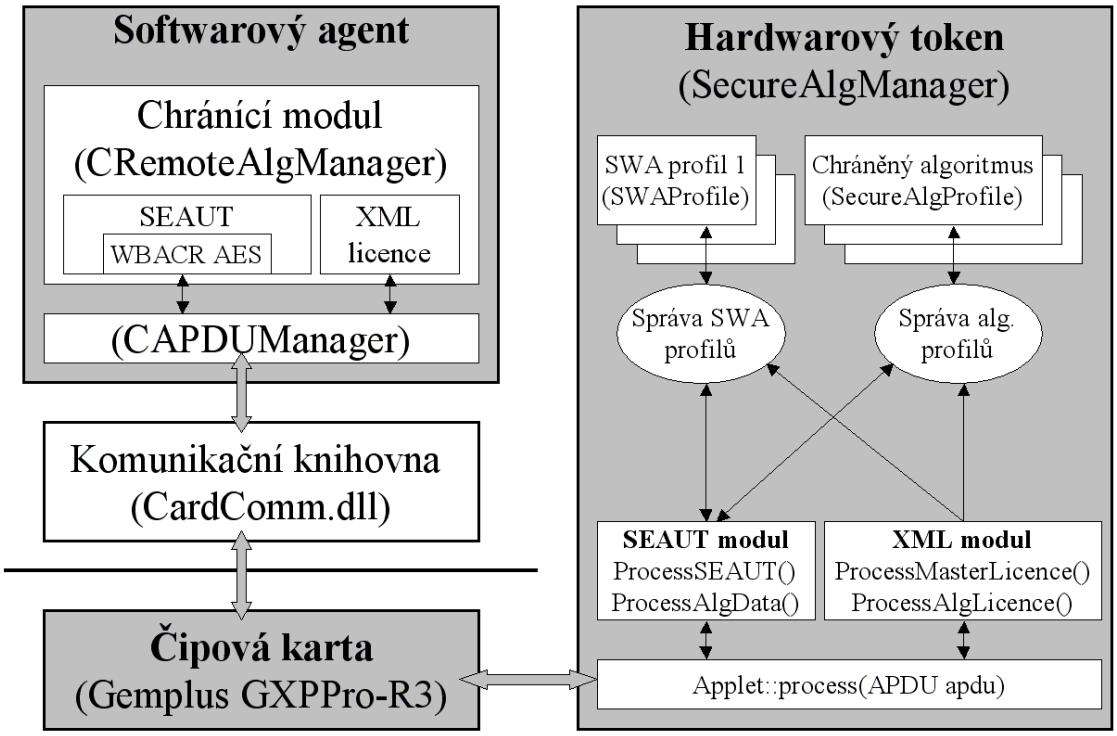
Modul správy softwarových agentů: Modul pro správu profilů softwarových agentů obsahuje samostatnou položku (profil) pro každého softwarového agenta, kterému je umožněno využívat služeb poskytovaných hardwarovým tokenem. Obsahem každého profilu je unikátní identifikace softwarového agenta, hodnota sdílených šifrovacích klíčů používaných pro autentizaci a utajení vyměňované komunikace během protokolu SEAUT, aktuální hodnota dynamicky ustanoveného klíče relace, údaj o stavu autentizace softwarového agenta vůči hardwarovému tokenu a pomocné údaje.

Modul správy chráněných algoritmů: Modul pro správu chráněných algoritmů umožňuje zpracovávat požadavky softwarového agenta, provést nad poskytnutými vstupními daty chráněný algoritmus a vytvořit odpověď zasílanou zpět softwarovému agentovi. V tomto modulu programátor doplňuje kód konkrétního chráněného algoritmu a určuje jeho chování. Je definována základní třída[2], kterou chráněný algoritmus rozšiřuje. Prostřednictvím metod základní třídy jsou chráněné algoritmy genericky používány.

Modul správy XML licencí: Modul pro zpracování XML licencí obsahuje jednoduchý XML parser využívající režim SAX pro zpracování XML dokumentu. Události jsou obsluhovány na základě typu licence rozdílnými obslužnými třídami. Projekt umožňuje zpracování dvou typů XML licencí. Prvním typem je řídící licence určená pro aktualizaci údajů souvisejících s profily softwarových agentů a chráněných algoritmů. Způsob zpracování řídící licence nemusí být měněn, s výjimkou obsluhy elementu pro přidání nového algoritmu. V současné podobě lze pomocí ní přidávat profily pro nové softwarové agenty, modifikovat existující profily softwarových agentů (sdílené šifrovací klíče…), odebírat profily softwarových agentů, přidávat a odebírat profily pro instance chráněných algoritmů a určovat množinu softwarových agentů s povoleným využíváním konkrétní instance chráněného algoritmu. Druhým typem je licence určená pro aktualizaci hodnot konkrétní instance chráněného algoritmu. Zpracování této licence může být programátorem upraveno tak, aby umožňovalo aktualizaci hodnot specifických pro chráněný algoritmus. Příkladem takových hodnot může být povolený počet a typ použití chráněného algoritmu nebo hodnoty šifrovacích klíčů používaných algoritmem. Původním záměrem bylo implementovat ukázkový příklad XML licence k algoritmu pomocí syntaxe jazyka XrML [XrML2], používanému pro specifikaci práv k digitálním datům. Vzhledem k bohatosti struktury jazyka XrML a paměťovému omezení současných hardwarových tokenů to nebylo možné. Za ukázkový příklad licence k algoritmu tak byla nakonec zvolena pouze jednoduchá syntaxe umožňující ovlivňovat počet použití cílového algoritmu. Pro utajení a integritu obou typů licencí je použit šifrovací algoritmus DES implementovaný použitou čipovou kartou GXPPro-R3. Dle konkrétního typu použité čipové karty lze tento algoritmus snadno nahradit jiným, pokud by nebyl čipovou kartou implementován. Hodnoty klíčů licencí jsou sdíleny mezi hardwarovým tokenem a jeho poskytovatelem a mohou být vzdáleně aktualizovány prostřednictvím řídící licence. Pro ochranu před útokem na běžné formy doplnění šifrovaného textu [Va02] byl zvolen odlišný způsob šifrování posledního bloku dle [KlRo02]. V příloze A lze nalézt DTD obou typů licencí.

[2] Třída SecureAlgProfile.

Modul ustanovení komunikačního kanálu: Modul pro ustanovení bezpečného komunikačního kanálu implementuje funkčnost požadovanou protokolem SEAUT. Využívá hodnot šifrovacích klíčů získaných z profilů jednotlivých softwarových agentů. Umožňuje souběžné ustanovení a udržování komunikačního kanálu pro více softwarových agentů. Při zabezpečení komunikačního kanálu je použit šifrovací algoritmus AES, na straně softwarového agenta implementovaný formou WBACR AES. V době tvorby projektu nebyla dostupná vhodná čipová karta, která by algoritmus AES hardwarově implementovala. Pro demonstraci výhod WBACR AES bylo třeba algoritmus AES na straně čipové karty implementovat pomocí prostředků nabízených rozhranním JavaCard. Výsledná implementace je velmi pomalá, neboť z paměťových důvodů nebylo možné použít běžnou optimalizaci AES pomocí předpočtených tabulek. Pro praktické použití je nutno zvolit čipovou kartu implementující AES na hardwarové úrovni.



Obr. 8.2: Podrobný přehled ochranného rozhranní.

8.3 Softwarový agent

Softwarovým agentem je běžná PC aplikace běžící v uživatelském režimu. Způsob implementace softwarového agenta není určen a závisí na potřebách programátora. Pouze část kódu, která by obsahovala algoritmus přesunutý do prostředí hardwarového tokenu, je nahrazena chránícím modulem[3] a voláním jeho metod.

[3] Třída CRemoteAlgManager.

8.3.1 Chránící modul

Chránící modul poskytuje funkčnost umožňující využití přesunuté algoritmu bezpečným způsobem. Nejprve dojde k ustanovení komunikačního kanálu dle protokolu SEAUT. Softwarový agent a hardwarový token se oboustranně autentizují s využitím autentizačních klíčů realizovaných na straně softwarového agenta formou WBACR AES tabulek. Ustanoveným kanálem lze zasílat žádosti o provedení chráněného algoritmu nad vstupními daty a přijímat odpovědi obsahující zpracovaný výstup chráněného algoritmu. Přijatá data jsou chránícím modulem podstoupena zpět hlavní části softwarového agenta. Integrita a utajení vyměňovaných dat je zajišťováno pomocí transportních klíčů, na straně softwarového agenta opět realizovaných formou WBACR AES tabulek.

Chránící modul umožňuje zasílání XML licencí obou typů určených pro hardwarový token. Podrobnější popis těchto licencí je uveden výše. Tato funkčnost může být odstraněna, nebo přesunuta do speciální aplikace odpovědné za distribuci XML licencí. Z důvodu paměťových omezení současných kryptografických hardwarových tokenů je formát zápisu logických operací realizovatelných licencí navržen tak, aby velikost zápisu každé z nich nepřesahovala povolenou velikost jednoho APDU příkazu. Není tak třeba vytvářet na hardwarovém tokenu paměťově náročné pole, které by bylo postupně plněno několika APDU příkazy. Rozsáhlou licenci lze vždy rozdělit na sérii jednodušších licencí zasílaných samostatnými APDU příkazy. Zároveň se tak snižuje velikost kódu nutného správu licencí na straně hardwarového tokenu a zvyšuje rychlost zpracování licence. Licence jsou vytvářeny a distribuovány poskytovatelem hardwarového tokenu.

8.3.2 Dodatečné ochranné prostředky

Je vhodné zabezpečit část softwarového agenta obsahující chránící modul dodatečnými ochrannými prostředky. Vhodnými kandidáty jsou obfuskační systémy pro ztížení extrakce nebo nahrazení WBACR AES tabulek, použití sebekontrolujícího kódu pro zajištění integrity a ochrany zaměřené proti standardním statickým a dynamickým inspekčním nástrojům. Tyto dodatečné ochranné prostředky nejsou v projektu použity.

8.4 Postup začlenění

Tato část textu popisuje posloupnost logických kroků, které je nutné vykonat pro využití ochranného rozhranní. Navržené dělení zachycuje životní cyklus použití ochranného rozhranní a zvýrazňuje skupiny operací, které mohou být prováděny nezávislými stranami. V příloze D je uvedeno podrobnější, a lehce odlišné „programátorské“ dělení. „Programátorské“ dělení vychází z typického použití a jeví se jako vhodnější pro popis, jak začlenit ochranného rozhranní na programátorské úrovni.

a) Distribuce fyzického hardwarového tokenu: Prvním krokem je distribuce fyzického hardwarového tokenu k cílovému uživateli. S výhodou lze využít hardwarových tokenů, které jsou již distribuovány, například formou SIM karet do mobilních telefonů nebo podpisových čipových karet (cílový hardwarový token musí mít požadované vlastnosti). Tento krok je časově nejnáročnější, není ho však třeba opakovat, neboť fyzický hardwarový token lze pro potřeby ochranného rozhranní opakovaně využít.

b) Implementace chráněných algoritmů: Druhým krokem je implementace chráněných algoritmů, překlad a (vzdálená) instalace hardwarového tokenu. Druhý krok může být prováděn stranou, která provedla fyzickou distribuci hardwarového tokenu, nebo nezávislou stranou, které je zpřístupněna možnost využívat zabezpečující klíče používané pro vzdálenou instalaci „appletů“ na fyzický hardwarový token. Instalovaný „applet“, který přestane postačovat požadavkům může být vzdáleně odstraněn a opakováním druhého kroku nahrazen novým. Hardwarovému tokenu je během instalace přiřazena unikátní identifikace.

c) Implementace softwarového agenta: Třetím krokem je implementace softwarového agenta. Dochází k začlenění volání metod ochranného modulu, přiřazení unikátní identifikace softwarového agenta a generování WBACR AES tabulek pro použité autentizační a transportní 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 hardwarového tokenu pro jednotlivé softwarové agenty.

d) Správa licencí: Čtvrtým krokem je tvorba řídící licence pro správu profilu softwarového agenta na cílovém hardwarovém tokenu. Možnost využití služeb hardwarového tokenu je podmíněno existencí profilu softwarového agenta, který definuje přístupová práva k jednotlivým chráněným algoritmům a obsahuje údaje nutné pro vytvoření bezpečného komunikačního kanálu. Během čtvrtého kroku lze vytvářet a distribuovat licence pro aktualizaci interních hodnot chráněných algoritmů, umístěných na hardwarovém tokenu. Licence k chráněnému algoritmu umožňuje při vhodné implementaci řídit využití chráněného algoritmu a poskytuje tak základ pro DRM. Čtvrtý krok je dle potřeby opakován. Licence jsou distribuovány v šifrované podobě zabezpečené klíčem sdíleným mezi hardwarovým tokenem a jeho poskytovatelem. Distribuce licence nevyžaduje speciální komunikační kanál, může být provedena přímo softwarovým agentem s využitím funkcí ochranného rozhranní, nebo jiným vhodným nástrojem pro komunikaci s hardwarovým tokenem.

e) Využití chráněného algoritmu: Pátým krokem je samotné využití služeb hardwarového tokenu softwarovým agentem. Softwarový agent vytváří nejprve bezpečný komunikační kanál a zasílá hardwarovému tokenu požadavky na využití chráněného algoritmu. Hardwarový token 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.

8.5 Výkonnostní charakteristiky

Přenesení algoritmů do bezpečného prostředí hardwarového tokenu zvyšuje jejich ochranu, zároveň však má vliv na celkový výkon softwarového agenta. Nemusí se nutně jednat o snížení výkonu. Hardwarový token může poskytovat akceleraci operací chráněného algoritmu. Při použití běžné kryptografické čipové karty však výkon omezen

bude a je tedy nutno vhodně zvolit, která část kódu softwarového agenta bude chráněna. Následující faktory mají vliv na změnu výkonu:

a) Rychlost vytvoření autentizovaného kanálu dle SEAUT.
b) Propustnost datové vrstvy mezi softwarovým agentem a hardwarovým tokenem.
c) Propustnost komunikační vrstvy dle SEAUT.
d) Rychlost přípravy resp. zpracování dat chráněného algoritmu na straně softwarového agenta.
e) Rychlost provedení chráněného algoritmu nad vstupními daty.

Pro konkrétní typ hardwarového tokenu se liší vliv jednotlivých faktorů. V současné době je významným faktorem především b) díky omezené rychlosti datového rozhranní prostřednictvím APDU příkazů. V závislosti na vlastnostech chráněného algoritmu může být významným i faktor e). Pro zanedbatelný vliv faktoru c) je na straně hardwarového tokenu vyžadována hardwarová podpora šifrovacího algoritmu AES.

Tabulka 8.3 zachycuje výsledky orientačního testu, provedeného za účelem odhadu doby trvání základní operací ochranného rozhranní na čipové kartě s hardwarovou podporou algoritmu AES. Pro odhad byla použita záměna šifrovacího algoritmu AES za algoritmus DES, hardwarově podporovaný čipovou kartou Gemplus GXPPro-R3 (HW DES). Z porovnání rychlosti hardwarových implementací algoritmů DES a AES [GaCho01, SaMo03] lze očekávat zlepšení oproti hodnotám ve sloupci HW DES. Celkové zrychlení bude závislé na poměru šifrování a ostatních operací. Pro srovnání je uveden i výkon při použití softwarové implementace AES (SW AES), použité v projektu.

Operace	HW DES	SW AES
Zaslání 1 APDU (0 B vstup, 0B výstup)	0,07 s	0,07 s
Zaslání 1 APDU (256 B vstup, 0B výstup)	0,16 s	0,16 s
Zaslání 1 APDU (256 B vstup, 256B výstup)	0,3 s	0,3 s
Použití chráněného algoritmu (0 B dat = 1 APDU)[4]	0,36 s	25 s
Použití chráněného algoritmu (240 B dat = 1 APDU)	0,61 s	N/A[5]
Použití chráněného algoritmu (1 kB dat = 4 APDU)	2,4 s	N/A
Navázání autentizovaného spojení dle SEAUT	0,52 s	24 s
Zpracování řídící licence [NewAlg] (220 B = 1 APDU)	13,7 s	N/A
Zpracování řídící licence [RemoveAlg] (130 B = 1 APDU)	6 s	N/A

Tab. 8.3: Trvání vybraných operací ochranného rozhranní.

[4] Chráněný algoritmus pro potřeby testu nad vstupními daty neprovádí žádnou operaci, pouze je předá v nezměněné podobě na výstup.

[5] Pokud operace probíhají na čipové kartě Gemplus GXPPro-R3 není ukončena do cca 60 sekund, je firemním ovladačem přerušena. Z tohoto důvodu nelze měřit čas operací trvajících déle než 60 sekund.

9 Závěr

Diplomová práce zmapovala základní charakteristiky výpočetních prostředí použitelných pro potřeby Digital Rights Managment (DRM). Těžištěm zájmu byla výpočetní prostředí s částečnou podporou hardwarových ochranných prostředků. Tato prostředí lze v současné době vytvářet s poměrně malými náklady přidáním programovatelné kryptografické čipové karty k běžné PC platformě. Vybraným operacím je tak umožněn výkon v bezpečném prostředí, zbývající část je stále prováděna v prostředí pod potencionální kontrolou útočníka. Pro zajištění dodatečné ochrany nechráněných částí operací bylo provedeno zmapování běžně používaných technik ochrany softwarových agentů. Tyto „klasické“ techniky byly doplněny o podrobné studium vybraných pokročilých ochranných technik, především metod obfuskace, sebekontrolujícího kódu, neinformovaného agenta a mobilní kryptografie. Konkrétní využití poslední jmenované metody, implementace šifrovacího algoritmu AES umožňující ukrytí použité hodnoty klíče (WBACR AES), bylo prakticky realizováno do podoby všeobecně použitelné programového nástroje.

Klíčová část práce se zabývá návrhem autentizačního a transportního protokolu, který by umožnil bezpečnou komunikaci mezi stranami, z nichž jedna je umístěna v prostředí pod kontrolou útočníka. Tento scénář odpovídá výše uvedenému výpočetnímu prostředí, které poskytuje (hardwarovou) ochranu pouze vybraným operacím a vyžaduje komunikaci mezi chráněnou a nechráněnou částí. Na základě rozboru slabých míst 3-průchodového protokolu ISO 9798-2 pro vzájemnou autentizaci v kontextu tohoto prostředí byla navržena modifikace a rozšíření stávajícího protokolu. Navržený protokol SEAUT by měl poskytovat vyšší stupeň ochrany použitých autentizačních informací na straně, která se nachází v prostředí pod kontrolou útočníka, a zvyšovat obtížnost záměrné modifikace jejího kódu za účelem napadení protokolu. Hlavním bodem použitým během návrhu je využití vlastností WBACR AES umožňujících utajení hodnoty klíče, oddělenou distribuci šifrovací a dešifrovací části a použití vstupního a výstupního kódování. Dále je zavedeno speciální využití klíče relace, které nahrazuje všechny snadno modifikovatelné porovnávací operace určené pro kontrolu očekávaných hodnot během protokolu. Případný útok tak není detekován přímo, ale způsobí nekorektní zpracování vyměňovaných dat. Tato vlastnost by měla výrazně ztěžovat útočníkovi záměrnou modifikaci kódu softwarového agenta implementujícího chování během protokolu.

Popis protokolu je doplněn o návrh způsobu generování vstupního a výstupního kódování WBACR AES tak, aby ho bylo možné použít v šifrovacím režimu CBC. Využití tohoto kódování umožňuje zvýšit odolnost proto záměrné modifikaci a poskytuje možnost personalizace softwarových agentů. Dále je uveden návrh (v kontextu práce) robustní metody tvorby kryptograficky jednosměrné hash funkce využitím blokového šifrovače realizovaného pomocí WBACR AES. Zapojení této metody dále zvyšuje bezpečnost implementace protokolu SEAUT.

Práce je uzavřena programovou realizací ochranného rozhranní, které umožňuje snadné využití kryptografické čipové karty s podporou JavaCard, pro zvýšení bezpečnosti softwarových agentů vykonávaných ve výpočetním prostředí běžné PC platformy. Ochranné rozhranní využívá některých technik a principů probíraných v průběhu práce,

především implementaci protokolu SEAUT. Pro možnost vzdálené a rozšiřitelné správy celého prostředí je v prostředí čipové karty implementována podpora zpracování XML licencí. Výsledkem je jednoduchá DRM architektura, přizpůsobitelná konkrétním požadavkům.

Námětem pro další studium může být podrobný rozbor bezpečnosti principů použitých pro návrh protokolu SEAUT. Jedná se především o způsob použití klíče relace (7.5.3 a 7.5.5), návrh generování vstupního a výstupního kódování WBACR AES pro šifrovací režim CBC (7.7) a vlastnosti jednosměrné hash funkce (7.9) použitelné pro aktualizaci klíče relace. Zranitelnost hodnoty klíče relace K_R vůči modifikaci by mohla být částečně řešena použitím vhodné kombinace vstupního/výstupního kódování WBACR AES tabulek určených pro jeho aktualizaci (7.9) a vstupního/výstupního kódování WBACR AES tabulek pro klíče KE_T a KD_T. Vhodná kombinace by mohla zajistit trvalou ochranu otevřené hodnoty klíče relace proti zisku útočníkem a zároveň umožnit použití způsobem definovaným dle SEAUT.

Podrobnější studium vyžaduje také nalezení vhodné metody doplnění dat před šifrováním na požadovanou délku bloku tak, aby byl vhodně použitelný pro WBACR AES a zároveň byl odolný proti útoku od S. Vaudenaye [Va02]. Úvodní rozvaha je uvedena v kapitole 7.7.

Literatura

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

[AFS97] Arbaugh, W., Farber, D., Smith, J.: A Secure and Reliable Bootstrap Architecture, IEEE Security and Privacy Conference, 1997.
Dokument dostupný na URL
http://www.cis.upenn.edu/~waa/aegis.ps (březen 2004)

[An03] Anderson, R.: 'Trusted Computing' FAQ version 1.1. Srpen 2003.
Dokument dostupný na URL
http://www.cl.cam.ac.uk/ftp/users/rja14/toulouse.pdf (březen 2004)

[AnKu02] Anderson, R., Kuhn, M.: Low Cost Attacks On Tamper Resistant Devices. Cambridge University, 2002.
Dokument dostupný na URL
http://www.cl.cam.ac.uk/~mgk25/tamper2.pdf (leden 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 (únor 2003)

[BSMS01] Microsoft's Digital Rights Management Scheme 2.0 - Technical Details by Beale Screamer. 2001.
Dokument dostupný na URL
http://apache.dataloss.nl/~fred/beale-screamer/657.zip (duben 2004)

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

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

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

[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.

[Cl02] Clark, D.: How Copyright Become Controversial. 2002.
Dokument dostupný na URL
http://www.cfp2002.org/proceedings/proceedings/clark.pdf (červenec 2003)

[ClkwTr02] Cloakware Corp., Ottawa: An Introduction to Cloakware Code Transformation Technology. Květen 2002.
Dokument dostupný na URL
http://206.191.60.52/resources/pdf/Intro_to_Transcoder.Litva.0703.pdf (duben 2004)

[ClkwDiv02] Cloakware Corp., Ottawa: Diversity As A Defence Against Automated Attacks On Software. Červen 2002.

[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 (duben 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 (červenec 2003)

[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 (říjen 2003)

[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.

[Ha02] Halderman, J.: Evaluating New Copy-Prevention Techniques for Audio CDs, Princeton University, 2002. Dokument dostupný na URL http://www.cs.princeton.edu/~jhalderm/papers/drm2002.pdf (duben 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.

[IDA] Interactive Dissassembler IDA, http://www.ida.com

[InfTPM02] Infineon Technologies, : Trusted Platform Module FAQ. 2002. Dokument dostupný na URL http://www.infineon.com/cmc_upload/documents/048/003/TPM_FAQ_1.pdf (duben 2004)

[JaRe01] Jakobsson, M., Reiter, M.: Discouraging Software Piracy Using Software Aging. Springer LNCS 2320, Berlin, 2001, s. 1-12.

[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 (duben 2004)

[KlRo02] Klíma, S. Rosa, T.: Strengthened Encryption in the CBC Mode. 2002. Dokument dostupný na URL ftp://ftp.decros.cz/pub/Archiv/Publications/2002/cbc.pdf (květen 2004)

[Ku97] Kuhn, M.: TrustNo1 Cryptoprocessor Concept, duben 1997 Dokument dostupný na URL http://www.cl.cam.ac.uk/~mgk25/trustno1.pdf (duben 2004)

[Ku98] Kuhn, M.: Cipher Instruction Search Attack on the Bus-Encryption Security Microcontroller DS5002FP. IEEE Transactions on Computers, 47, říjen 1998.

Dokument dostupný na URL http://www3.informatik.uni-erlangen.de/Publications/Articles/kuhn_ToC.pdf (duben 2004)

[MSRM03] Microsoft Corp., Redmond: Architecture of Windows Media Rights Manager. 2003. Dokument dostupný na URL http://www.microsoft.com/windows/windowsmedia/wm7/drm/architecture.aspx (květen 2003)

[NGSCB04] NGSCB, http://www.microsoft.com/resources/ngscb/default.mspx

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

[No00] Novotný, P.: Ochrana spustitelných souborů v OS Windows [Diplomová práce]. VUT Brno, 2000.

[OP02] Open Platform specification, Dokument dostupný na URL http://www.globalplatform.org/showpage.asp?code=specifications (březen 2004)

[PKCS#5] RSA Laboratories: PKCS#5 v. 2.0: Password-Based Cryptography Standard, 1999, Dokument dostupný na URL http://www.rsasecurity.com/rsalabs/pkcs/pkcs-5/index.html (květen 2004)

[RFMON] Winternals Software: Regmon, Filemon documentation, Dokument dostupný na URL http://www.winternals.com (březen 2004)

[RiDa99] Daemen, J., Rijmen, V.: AES Proposal: Rijndael. Belgium září 1999. Dokument dostupný na URL http://csrc.nist.gov/encryption/aes/rijndael/Rijndael.pdf (listopad 2003)

[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/campa/teaching/ruuskanen-final.pdf (září 2003)

[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.

[SDMI01] Secure Music Digital Initiative, Specification Overview, Dokument dostupný na URL http://www.sdmi.org/port_device_spec_overview.htm (únor 2004)

[SkAn02] Skorobogatov, S., Anderson, R.: Optical Fault Induction Attack. Cambridge University, 2002. Dokument dostupný na URL http://www.cl.cam.ac.uk/ftp/users/rja14/faultpap3.pdf (duben 2004)

[SOFTICE] SOFT-Ice, http://www.numega.com

[St01] Studzinsky, A.: Bezpečnost ByteCode [Ročníkový projekt]. VUT Brno, 2001.

[TCG04] Trusted Computing Group, http://www.trustedcomputinggroup.org/home/

[Va02] Vaudenay, S.: Security Flaws Induced by CBC Padding. EUROCRYPT 2002. Springer LNCS 2332, Berlin, 2002, s. 534-545.

[XrML2] Content Guard, Inc.: XrML 2.0 Specification. Listopad 2001. Dokument dostupný na URL http://www.xrml.org/get_XrML.asp (duben 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 2003)

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