

Hardwarové bezpečnostní moduly – API a útoky

Jan Krhovják <xkrhovj@informatics.muni.cz>,
Daniel Cvrček¹ <Daniel.Cvrcek@cl.cam.ac.uk>,
Vašek Matyáš² <matyas@informatics.muni.cz>.

Fakulta informatiky Masarykovy univerzity v Brně

Abstrakt: Příspěvek pojednává o vybraných aspektech hardwarových bezpečnostních modulů a zaměřuje se na útoky na a přes aplikační programovací rozhraní (API). Nejdříve se kromě architektury kryptografických modulů seznámíme se základními požadavky na jejich bezpečnost a s americkými normami FIPS 140-1 a FIPS 140-2, jimiž jsou tyto požadavky obvykle specifikovány. Dále se pak budeme věnovat samotné problematice bezpečnosti API, a také popisu útoků, které jsou často umožněny chybným návrhem mnohých běžně používaných aplikačních programovacích rozhraní.

1 Úvod

Zajištění bezpečné komunikace především u distribuovaných systémů a aplikací vyžaduje použití vhodných mechanismů a prostředků pro zajištění integrity jejich kryptografických funkcí a důvěrnosti příslušných šifrovacích klíčů. U běžně používaných výpočetních systémů toho lze dosáhnout jen velmi obtížně, protože jejich hardware je typicky zcela fyzicky nezabezpečen a software obsahuje velké množství chyb. To bylo hlavním důvodem návrhu a vytvoření hardwarových bezpečnostních zařízení (HSM – Hardware Security Module), do jejichž fyzicky zabezpečeného prostředí se přesunulo provádění veškerých kryptografických operací.

Oblastí, jež odstartovala vývoj hardwarových bezpečnostních zařízení, bylo bankovníctví. A hlavní skupinou aplikací pak elektronické transakce v bankovních sítích (např. VISA, MasterCard, American Express) zahrnující komunikaci jednotlivých bank a komunikaci s jejich peněžními bankomaty (ATM). Tyto rozsáhlé ATM sítě³, do kterých je v dnešní době sdružováno stále více bank, umožňují zákazníkům provádět finanční transakce téměř z kteréhokoliv místa na světě a bankám také samozřejmě přinášejí větší obrat a zisky. Jednotlivé banky ani zákazník si však nemohou vzájemně důvěřovat a úlohou HSM je, kromě zabezpečení důvěrného přenosu dat, také bezpečná správa kryptografických klíčů a jiných citlivých dat (např. PINů) tak, aby se předešlo podvodům ze strany klientů i zaměstnanců bank.

Hardwarová bezpečnostní zařízení poskytují bezpečné prostředí pro provádění citlivých operací, ke kterým je možné přistupovat za použití přesně definovaného softwarového rozhraní – tzv. aplikačního programovacího rozhraní (API). To by mělo být navrženo tak, aby nemohlo dojít ke zneužití či kompromitování chráněných dat uložených v HSM. Snaha o co nejflexibilnější návrh těchto zařízení, a podpora mnoha standardů a norem, však činí jednotlivá API příliš složitá a jejich správnou funkčnost lze pak jen stěží zaručit. Důkazem toho je stále narůstající počet útoků na API různých HSM, jejichž analýzou se budeme v tomto příspěvku zabývat.

¹ Práce na příspěvku byly uskutečněny především během působení v Computer Laboratory, University of Cambridge, UK.

² Práce na příspěvku byly také prováděny během předchozího pobytu na University College Dublin a nyní u Microsoft Research Ltd. (Cambridge, UK).

³ V tomto příspěvku bude pojem *ATM síť* značit vždy síť peněžních bankomatů, a význam zkratky ATM je tedy nutno interpretovat nikoliv jako Asynchronous Transfer Mode, ale jako Automatic Teller Machine.

1.1 Základní terminologie a pohled na bezpečnost

Hardwarová bezpečnostní zařízení, někdy též označovaná jako *kryptografické koprocesory* či *kryptografické moduly*, jsou bezpečnostní zařízení určená k bezpečnému provádění kryptografických operací v jinak nedůvěryhodném prostředí. Lze mezi ně zařadit také *čipové karty*, které se používají k autentizaci nebo k uchovávání citlivých dat, či *kryptografické akcelerátory*, které slouží k urychlování kryptografických operací.

Koprocesory i akcelerátory bývají většinou nainstalovány v tzv. *hostitelských zařízeních*, což mohou být například osobní počítače, velké výpočetní servery nebo specializované bankovní systémy. Z bezpečnostního hlediska může být hostitelské zařízení *nedůvěryhodným prostředím*, zvláště pokud je umístěno na veřejně přístupném místě a je bez jakékoliv fyzické ochrany.

Převážná část příspěvku popisuje útoky na HSM. Jako *útok* označujeme postup, pomocí něhož získáme data, která mají být pro útočníka nepřístupná, nebo postup, kterým provede operaci, ke které není autorizován.

1.2 Architektura kryptografických modulů

Architektura kryptografických modulů je do značné míry závislá na jejich typu. Základní architektura vychází z klasické von Neumannovy architektury. Oproti běžným počítačům je ovšem množina základních bloků rozšířena o mechanismy fyzické ochrany (odolnost proti fyzickým útokům), generátory náhodných čísel či speciální koprocesory pro urychlení specifických kryptografických operací. Na druhou stranu neobsahují např. běžné V/V obvody, což značně snižuje složitost operačního prostředí. Je zřejmé, že odolnost proti průnikům (fyzickým útokům) nebude u čipových karet nikdy realizována stejným způsobem jako u kryptografických koprocesorů či bankomatů. Jestliže ovšem porovnáme funkční schémata, základní bloky jsou stejné.

Jádrem celého zařízení je procesor, který řídí veškeré vstupní a výstupní operace, zpracovává přerušování a stará se o správu paměti. Navíc úzce spolupracuje s případnými dalšími procesory sloužícími k provádění specializovaných kryptografických operací.

Paměť určená pro uchování citlivých informací je většinou napájena přídavnými bateriemi a obsahuje tajné klíče, které mohou být v případě detekce průniku vymazány. Generátor náhodných čísel je obvyklou součástí kryptografických modulů, protože kryptograficky bezpečný zdroj náhodných dat je pro tato zařízení naprostou nutností.

Na kryptografické čipové karty (smart cards) lze pohlížet jako na levné kryptografické moduly s menší výpočetní silou.

2 Bezpečnostní požadavky na kryptografické moduly

Bezpečnostní požadavky na kryptografické moduly jsou specifikovány v normě FIPS⁴ 140-2 [11], která od 25. listopadu 2001 zcela nahradila starší normu FIPS 140-1 [10] z roku 1994. Norma definuje čtyři úrovně zabezpečení, které pokrývají široké spektrum možností potenciálního použití kryptografického modulu v různých aplikacích a prostředích. Jednotlivé úrovně jsou specifikovány požadavky v jedenácti oblastech (viz 2.1) vztahujících se k bezpečnému návrhu a implementaci modulu. Vyšší úrovně odpovídají i vyšší požadavky na zabezpečení.

Úroveň 1 – definuje nejnižší stupeň ochrany. Specifikovány jsou pouze základní bezpečnostní požadavky, jako je například používání schválených algoritmů. Nejsou vyžadovány žádné specifické mechanismy fyzické ochrany ani speciálně ohodnocený operační systém. Typickými příklady zařízení spadajících do této úrovně jsou osobní počítače.

Úroveň 2 – rozšiřuje fyzické zabezpečení modulu tím, že vyžaduje zajištění evidence průniků, která bývá nejčastěji realizována použitím kvalitních zámků či pečeti. Nutností je také autentizace založená na *rolích*⁵. Tato úroveň povoluje spouštění softwarových a firmwarových částí krypto-

⁴ Federal Information Processing Standard.

⁵ Při tomto typu autentizace nemusí být ověřena identita operátora.

grafického modulu na obecném počítačovém systému pouze s operačním systémem⁶, který splňuje alespoň úroveň EAL2 ohodnocenou podle CC⁷. Typickými příklady zařízení spadajících do této úrovně jsou čipové karty.

Úroveň 3 – dále rozšiřuje požadavky na fyzické zabezpečení. Přístup k citlivým informacím udržovaným uvnitř modulu již není pouze pasivně chráněn, např. za použití silných ochranných krytů, ale musí existovat mechanismy, které v případě detekce průniku citlivá data vymažou. Požadována je i autentizace založená na ověřování identity, což je rozšíření autentizace založené na rolích. Citlivá data, která opouštějí modul v nezašifrované podobě, by měla používat speciálních fyzicky oddělených portů či rozhraní, s jejichž pomocí lze vytvořit důvěryhodný kanál. Softwarové a firmwarové části modulu mohou být spouštěny pouze operačním systémem, který splňuje alespoň úroveň EAL3 ohodnocenou podle CC. Příkladem zařízení splňujícího FIPS 140-1 a spadajícího do této úrovně je Luna CA.

Úroveň 4 – definuje nejvyšší stupeň zabezpečení. Zařízení na této úrovni bývají určena k provozování v zcela nechráněných prostředích, a proto by měla být schopna detekovat a reagovat na všechny známé neautorizované pokusy o fyzický průnik. Navíc je také požadována specifikace vnějších provozních podmínek modulu, které musí být za provozu dodrženy (např. povolený rozsah napětí či teplot). Důvodem je existence útoků, které využívají překročení provozních podmínek k získání citlivých informací. Kryptografický modul tedy musí buď obsahovat senzory, s jejichž pomocí je možné vnější podmínky kontrolovat a případně citlivá data smazat, nebo provedení série přísných testů, jež prokážou, že je schopen chránit citlivá data i při práci mimo rozsah jeho operačních hodnot. Softwarové a firmwarové části modulu mohou být spouštěny pouze operačním systémem, který splňuje alespoň úroveň EAL4 ohodnocenou podle CC. Příkladem zařízení splňujícího FIPS 140-1 a spadajícího do této úrovně je IBM 4758.

2.1 Oblasti bezpečnostních požadavků

V této části jsou v jednotlivých oblastech specifikovány bezpečnostní požadavky, vůči kterým je kryptografický modul nezávisle testován. Po ukončení testů získá modul celkové ohodnocení tak, že jeho výsledná úroveň zabezpečení je rovna minimální úrovni dosažené i třeba jen v jediném testu. V mnoha oblastech klade standard požadavky i na obsah povinné, prodejcem poskytované dokumentace. Následující odstavce popisují rozsah požadavků kladených na jednotlivé aspekty bezpečnosti.

Dokumentace kryptografického modulu – by měla specifikovat hardwarové, firmwarové a softwarové komponenty kryptografického modulu, kryptografickou hranici modulu a fyzickou ochranu modulu. Je třeba specifikovat fyzické porty, logická rozhraní a všechny definované datové vstupy a výstupy. Dokumentace by měla také obsahovat výčet všech bezpečnostních funkcí modulu včetně všech operačních modů a specifikovány by měly být i veškeré k bezpečnosti se vztahující informace (např. kryptografické klíče, autentizační data) a bezpečnostní politika modulu.

Porty a rozhraní – kryptografický modul by měl omezovat veškerý datový tok a fyzický přístup k logickým rozhraním, která definují vstupní a výstupní body modulu. Rozhraní musí být alespoň logicky odděleny, ale mohou sdílet společný fyzický port. Norma specifikuje čtyři logická rozhraní pro vstup a výstup. První je určeno pro vstup příkazů, signálů a kontrolních dat. Druhé pro výstup stavových dat, signálů či fyzických indikátorů stavu (tj. LED diody a displeje). Zbývá dvě pak slouží pro vstup a výstup všech dat (včetně kryptografických klíčů či nezašifrovaných dat). Za speciální vstup do kryptografického modulu je považován i přívod napájení, či zdroj hodinového kmitočtu.

Role, služby a autentizace – kryptografický modul by měl pro operátory⁸ podporovat autorizované role, s přiřazenými službami. Pokud jsou podporovány paralelní sezení, musí být logicky zcela oddě-

⁶ Tímto je míněn operační systém uvnitř hardwarového modulu, nikoliv na hostitelském zařízení.

⁷ Common Criteria – mezinárodní kritéria ohodnocení IT bezpečnosti. Standard FIPS se opírá o CC všude tam, kde je vyžadována validace funkčních vlastností.

⁸ Jedinec či proces, který má přístup ke kryptografickému modulu a jeho službám.

leny (procesorový čas, paměť) operace prováděné v jednotlivých sezeních. Jeden operátor může mít přiděleno i více rolí, z nichž následující tři by měly být vždy modulem podporovány:

1. Uživatelská role – umožňuje přístup k bezpečnostním službám, a provádění kryptografických funkcí či jiných operací.
2. Role bezpečnostního úředníka – umožňuje inicializaci a konfiguraci zařízení (např. vkládání či změnu kryptografických klíčů a funkce auditu).
3. Role pro údržbu – je určena pro služby fyzické a logické údržby (např. diagnostika hardware či software). Při použití této role jsou veškeré citlivé informace bez ochrany automaticky vymazány. Tato role ale nemusí existovat, pokud není podporována údržba zařízení.

Pojem *služby* pokrývá všechny operace a funkce, které mohou být kryptografickým modulem prováděny. Operátor by měl mít vždy k dispozici služby poskytující informace o stavu zařízení a provádějící testování modulu a alespoň jednu schválenou bezpečnostní funkci. Dále může být také požadována *autentizace* operátora, jejíž pomocí modul ověří, zda je operátor autorizován k osvojení požadované role a k používání služeb s ní spjatých. Podporována by měla být alespoň jedna z následujících metod:

1. Autentizace založená na rolích – modul vyžaduje, aby si operátor vybral jednu či více rolí a autentizuje její (resp. jejich) převzetí. Identita samotného operátora není testována.
2. Autentizace založená na identitách – oproti předchozímu případu je navíc testována i identita operátora.

Při implementaci těchto mechanismů se jako autentizační data využívají například hesla, klíče, PINy, tokeny či biometriky (kdy náhodný pokus o přístup má šanci úspěchu menší než $1:10^6$). Tato data by měla být uvnitř modulu chráněna proti modifikaci, záměně či odhalení. Po restartu zařízení je vždy nutná nová autentizace.

Konečně stavový model - každý modul (na úrovni 4) by měl být specifikován pomocí konečně stavového přechodového diagramu, který obsahuje všechny operační a chybové stavy modulu, přechody mezi těmito stavy a události, které přechody způsobují nebo jsou jimi způsobeny.

Fyzická bezpečnost – tato oblast definuje požadavky na fyzické bezpečnostní mechanismy kryptografického modulu, které jsou zvlášť specifikovány pro tři základní typy modulů:

1. Jednočipové – tyto moduly obsahují jeden integrovaný obvod a nemusí být nijak fyzicky chráněny (předpokládá se ochrana hostitelským zařízením). Typickým příkladem jsou čipové karty.
2. Vícečipové vestavěné (embedded) – obsahují jeden a více integrovaných obvodů, které však také nemusí být fyzicky chráněny krytem. Příkladem jsou rozšiřující karty.
3. Vícečipové autonomní – integrované obvody jsou v tomto případě již zcela fyzicky chráněny svým krytem. Tyto moduly se používají v nechráněném a v nedůvěryhodném prostředí. Příkladem jsou kryptografické routery.

V závislosti na typu modulu jsou pak stanoveny požadavky pro jednotlivé úrovně zabezpečení.

Operační prostředí – má smysl pouze u zařízení, která umožňují přihrávání a spouštění nových částí programového vybavení. Podstatnou součástí prostředí je operační systém, ten je rozdělen na modifikovatelnou (např. firmware v RAM) a nemodifikovatelnou (např. firmware v ROM) část.

Správa klíčů – tato oblast definuje požadavky na celý životní cyklus kryptografických klíčů: generování, ustanovení, verifikaci, distribuci, uložení či vymazání. Její součástí je také specifikace generátoru náhodných čísel, jehož pomocí mohou být tajné klíče vytvářeny. Pro všechny úrovně zabezpečení je vyžadováno, aby tajné či soukromé klíče byly kryptografickým modulem chráněny před neautorizovaným čtením, modifikací či záměnou.

Elektromagnetické interference a kompatibilita – v této sekci jsou definovány požadavky na elektromagnetické interference a kompatibilitu pro FCC⁹.

Auto-testy – zde jsou definovány požadavky na testy, které musí být provedeny modulem při svém spuštění či restartu, a které ověřují jeho správnou funkčnost a integritu. Proběhne-li testování neúspěšně, musí modul vstoupit do chybového stavu a chybu ohlásit. V tomto stavu nelze provádět žádné kryptografické operace. Kromě testování integrity software/firmware jsou testovány například i kryptografické algoritmy, generátor náhodných čísel či korektnost soukromého a veřejného klíče.

Záruka návrhu – tato oblast se týká použití nejlepších technik a postupů pro vývoj, správu konfigurace, nasazení a používání. Poskytuje také záruky řádného testování, doručení a instalace. Důraz je kladen i na uživatelskou dokumentaci.

Zmírnění jiných útoků – tato část pojednává o zmírnění dalších útoků, na něž může být kryptografický modul náchylný. Jedná se především o známé útoky, pro něž v době vydání tohoto standardu nebyly ještě ustanoveny testovatelné požadavky (např. časová analýza, výkonová analýza), nebo které převyšují rámec této normy (např. program TEMPEST).

2.2 Srovnání norem FIPS 140-1 a 140-2

FIPS 140-2 je relativně nový standard, a proto je mnoho v současné době používaných modulů certifikováno pouze podle FIPS 140-1. Obě normy mají podobnou strukturu, avšak k drobným změnám či upřesněním došlo ve všech částech. Jejich podrobné srovnání lze nalézt v NIST SP¹⁰ 800-29 [12]. Norma FIPS 140-2 je v některých částech jinak strukturována a došlo také k celkovému upřesnění a sjednocení použité terminologie. Asi nejpatrnějšími změnami oproti FIPS 140-1 je přidání nové části týkající se zmírnění jiných útoků a fyzická/logická separace portů. K dalším změnám patří například zesílení požadavků na autentizační mechanismy a na testování modulu. V důsledku vzniku nových norem je již také operační systém ohodnocován podle CC (namísto TCSEC¹¹).

3 Popis útoků na a přes API

Cílem této části je podrobný rozbor útoků na a přes API, jejichž význam spočívá především v demonstraci nedostatků a chybného návrhu API současných kryptografických modulů. Ukazuje se, že postupný vývoj těchto produktů a snaha o univerzálnost návrhu těchto zařízení, a tudíž i podpora mnoha standardů či norem, vyústila v až příliš rozsáhlá API, jejichž správnou funkčnost lze jen stěží zaručit. Výzkum týkající se bezpečnosti API odstartoval Ross Anderson v [1].

3.1 Úvod do problematiky API pro HSM

API tvoří jediné komunikační rozhraní mezi kryptografickým koprocesorem a vnější aplikací. Jejich pomocí je realizován přístup k veškerým operacím, které koprocesor provádí. Na základě funkcí API jsou budovány protokoly, které bývají typicky tvořeny posloupností tří až pěti zpráv, které si předávají (resp. přeposílají) jednotlivé strany protokolu.

Návrh protokolů je již sám o sobě velmi obtížnou záležitostí, a to i přes malý počet vyměňovaných zpráv, které mohou být útočníkem zmanipulovány (HSM nepodporují sezení, což znamená, že každá zpráva musí být samopopisná). API kryptografického koprocesoru obsahuje typicky 30–500 funkcí s mnoha parametry, což poskytuje velmi velký prostor k chybám a vzniku útoků. Výsledkem útoků, které byly objeveny, je vždy obejít definované bezpečnostní politiky zařízení.

⁹ Federal Communications Commission – americká státní agentura zodpovědná za regulaci federálních komunikací.

¹⁰ National Institute of Standards and Technology Special Publication.

¹¹ Trusted Computer System Evaluation Criteria – někdy též označovány jako *Oranžová kniha*.

V tomto příspěvku se budeme zabývat výhradně kryptografickými API. Ty lze dále rozdělit na standardní a finanční kryptografická API¹². Standardní kryptografická API poskytují pouze základní funkcionalitu, která je požadována po bezpečnostním zařízení (např. správa klíčů či šifrování). Oproti tomu finanční kryptografická API poskytují navíc funkce pro specifické finanční operace (např. manipulace s PINy či podpora SET), které jsou vyžadovány bankovním sektorem. Mezi běžně komerčně používané API patří například The Common Cryptographic Architecture (CCA) od IBM, The Compaq-Atalla API, Public Key Cryptography Standard (PKCS) #11 od RSA nebo The Thales-Zaxus-Racal API.

Abychom vytvořili alespoň trochu přehledný seznam existujících útoků, rozdělili jsme je do skupin podle typů chyb, kterých využívají. Těmi základními oblastmi jsou klíče a jejich integrita, nedostatečná kontrola parametrů funkcí, nevynucování bezpečnostní politiky.

3.2 Klíče a jejich integrita

Problém evoluce HSM a snaha o dodržení zpětné kompatibility je nejpatrnější u pokračujícího používání slabého kryptografického algoritmu DES. Ačkoliv nové moduly už dokáží pracovat např. s algoritmem 3DES (se dvěma klíči), tak bylo nalezeno velké množství útoků, které využívají nedostatečné ochrany před změnou typu klíče z 3DES na DES a relativně snadného útoku na slabší algoritmus.

3.2.1 The Meet in the Middle Attack

Malá velikost šifrovacích klíčů DES umožňuje využít špatný návrh rozlišování typů klíčů a neexistenci omezení na generování klíčů (např. limity omezující počet vygenerovaných klíčů). Tento fakt umožňuje snížit výpočetní náročnost hledání DES klíče. Mnoho kryptografických koprocesorů dokáže vygenerovat desetitisíce klíčů řádově během minut. Jestliže útočník vygeneruje 2^{16} klíčů, sníží tak výpočetní náročnost nalezení alespoň jednoho klíče z původních 2^{55} na 2^{39} . Prohledání takto redukováného prostoru klíčů zabere i na domácím PC pouze několik dnů, s použitím speciálního zařízení využívající FPGA technologii několik hodin.

Myšlenka celého útoku je rozdělit výpočetní složitost na výpočetní a paměťovou složitost. Nejprve se každým z vygenerovaných 2^{16} klíčů zašifruje stejný testovací vzorek a výsledek se uloží. Poté se systematicky prohledává klíčový prostor a stejný testovací vzorek se každým klíčem zašifruje a porovná se všemi uloženými vzorky. Dojde-li při porovnávání ke shodě, znamená to, že byla nalezena hodnota jednoho tajného klíče.

Je-li tento klíč oprávněn k šifrování dalších klíčů uložených v HSM (tzv. terminální klíč), lze jím přešifrovat veškerá jinými terminálními klíči chráněná data a klíče, které pak útočník snadno dešifruje. Tímto způsobem je možné kompromitovat osm z devíti typů klíčů, které používá Visa Security Module (VSM). Podrobně je tento útok popsán v [4].

Velmi pěknou variantu útoku je možno aplikovat i na kryptografický modul Prism. Při vynaložení stejného úsilí jako v předchozím případě lze získat dokonce hlavní klíč celého zařízení. V tomto kryptografickém modulu se hlavní (DES) klíč ustanovuje manuálně z jednotlivých částí, které jsou v modulu postupně XORovány (operace \oplus). Po vložení každé části klíče je vrácen kontrolní vektor (pro ověření korektního nahrání nové části klíče) zašifrovaný nově vytvořeným klíčem. Chybou v API tohoto zařízení je, že jakýkoliv uživatel může pokračovat v přidávání nových částí hlavního klíče, a tím i v získávání dalších variant zašifrovaného kontrolního vektoru. Jestliže tak vytvoří 2^{16} variant zašifrovaného kontrolního vektoru postupuje dále analogicky s předchozím případem.

3.2.2 Conjuring Keys From Nowhere

Kouzlení klíčů je považováno za útok¹³ umožňující neautorizované generování klíčů ukládaných mimo kryptografický koprocesor. V podstatě se jedná o náhodné vytvoření zašifrovaného klíče, který se podstrčí koprocesoru. Po dešifrování je hodnota klíče také náhodná a v případě DES má s pravděpo-

¹² V praxi jsou však většinou součástí pouze jednoho společného API.

¹³ I přesto, že některé starší moduly používaly tuto techniku k regulárnímu generování klíčů.

dobností $1/2^8$ správnou paritu¹⁴. V případě 3DES má správnou paritu s pravděpodobností $1/2^{16}$, což je stále dosažitelné. Tento útok lze aplikovat i na mnohé současné kryptografické moduly (např. IBM 4758 s CCA API), které používají formáty klíčů bez jakéhokoliv doplnění.

Na první pohled toto nevypadá přímo jako útok, protože útočník hodnotu odšifrovaného klíče nezná a není schopen dešifrovat data tímto klíčem zašifrovaná, ale takto vložené klíče mohou být využity ke složitějším útokům na API. Obrana spočívá v pečlivějším návrhu formátu klíčů obsahujícím větší množství entropie (např. před jeho zašifrováním přidat kontrolní součet a časové razítko).

3.2.3 3DES Key Binding Attack

Tento útok popsáný v [2, 4] je důsledkem nedostatečné vazby jednotlivých částí 3DES klíčů a jeho výsledkem je kompromitování většiny koprocesorem chráněných klíčů. CCA sice rozlišuje¹⁵ mezi levou a pravou částí klíče, ale již nespecifikuje příslušnost ke konkrétnímu klíči. Tím je umožněna nekorektní manipulace s jeho jednotlivými polovinami. CCA vyžaduje u některých typů klíčů, aby byly nejen typu 3DES, ale aby se také obě 64 bitové části klíče lišily. Pro generování klíčů ovšem druhá podmínka není vyžadována (z důvodu zpětné kompatibility).

Útočníkovi tedy tímto útokem stačí vygenerovat velké množství klíčů se stejnými polovinami (replicated keys) a stejného typu jako požadovaný klíč, pomocí *Meet in the Middle* útoku nalézt hodnoty dvou z těchto klíčů (prohledávání 2^{41} možností) a výměnou jejich polovin vytvořit dva 3DES klíče s odlišnými polovinami. Je-li nalezeným klíčem exportní klíč, můžeme nyní s jeho pomocí exportovat a poté dešifrovat všechny klíče v HSM určené k exportu.

Získat však lze i klíč, který nemá povolen export. Stačí jen zaměnit jednu jeho polovinu s polovinou známého klíče. Tím vzniknou dva klíče, jejichž jedna polovina je známá a druhou získáme prohledáváním prostoru 2^{56} (hledáme oba klíče současně). To je již samozřejmě práce pro speciální hardware či distribuované systémy. Pokud je uplatňována přísnější kontrola přístupu a útočník nemá povoleno vytvářet klíče se stejnými polovinami, musí vygenerovat standardní DES klíč a změnit obsah jeho kontrolního vektoru (struktura obsahující informace o klíči) na LEFT HALF OF A 3DES KEY. K tomu je možné použít útok na import klíčů [2, 3, 4].

3.3 Nedostatečná kontrola parametrů funkcí – PINy

U bankovních HSM je jednou z nejcitlivějších oblastí práce s PINy bankovních karet. Ověřování PINů bylo ovšem hlavní motivací pro zavádění HSM v bankovníctví, protože umožňovaly klientům bank používat své bankovní karty kdekoli po světě. Je pozoruhodné, jak špatně jsou chráněny citlivé parametry právě těch funkcí zajišťujících operace s PINy. PINy jsou formátovány do tzv. PIN-bloků (CPB), což jsou 8 bajtové struktury. Ty se po zašifrování nazývají EPB (encrypted PIN block).

Útoky k získání PINů - *PIN Recovery Attacks* tvoří snad největší třídu útoků na současné HSM. Tyto útoky demonstrují techniky, jejichž pomocí lze ze zašifrovaného PIN-bloku bez znalosti klíče získat hodnotu PINu. K jejich provedení mnohdy stačí pouze přístup ke kryptografickému finančnímu API daného zařízení (s řádně nainstalovanými klíči) a jeden zašifrovaný PIN-blok (EPB).

Kromě toho, že jsou útoky extrémně rychlé a snadno implementovatelné, lze je většinou aplikovat i na více druhů běžně používaných API¹⁶, čímž postihnou mnohem více HSM. Dále v této části vycházíme především z [5, 7, 9].

3.3.1 Decimalisation Table Attacks

Typická verifikační funkce na základě validačních dat (obvykle je to číslo účtu) vygeneruje PIN a porovná jej s PINem získaným z EPB¹⁷. Bezpečnostním problémem těchto funkcí jsou právě metody generování PINů vycházející ze starých metod používaných bankomatem IBM 3624. Jedním z parametrů těchto funkcí je decimalizační tabulka umožňující převod čísel ze šestnáctkové na desít-

¹⁴ DES používá lichou paritu a jako paritní je považován nejméně významný bit každého oktetu.

¹⁵ Pro zjednodušení uvažujeme pouze dvouklíčový 3DES.

¹⁶ Například IBM CCA API, Compaq-Atalla API a Thales-Zaxus-Racal API.

¹⁷ Vstupem funkce nemůže být přímo PIN, protože bankovní programátor by mohl snadno vyzkoušet všechny PINy (10^4 možností) a určit hodnotu PINu v EPB.

kovou soustavu. S touto tabulkou lze ovšem dělat zajímavé věci. Podívejme se ale nejdříve, jak se vlastně PIN generuje.

Techniky generování a verifikace PINů

Existuje mnoho používaných metod generování a verifikace PINů, jejichž typickými příklady jsou metody IBM 3624 a IBM 3624 Offset. IBM 3624 generování PINů je založeno na validačních datech (např. číslo účtu – PAN), která jsou zašifrována PIN generujícím klíčem a požadovaný počet číslic je převeden do desítkové soustavy (decimalizován) a zvolen jako PIN.

Verifikace pak probíhá analogicky, avšak PIN generující klíč se nazývá PIN verifikující klíč a vypočítaný PIN je nakonec porovnán s PINem získaným z EPB. Metoda IBM 3624 Offset navíc použitím offsetů umožňuje změnu PINu zákazníkem. Generování zde probíhá stejně jako v předchozím případě, ale výsledek se nazývá IPIN (intermediate PIN) a offset je získán odečtením IPINu od zvoleného PINu. Všechny operace sčítání a odčítání se provádějí na jednotlivých číslicích (modulo 10) a k decimalizaci se používá decimalizační tabulka. Názorný příklad výpočtu zákazníkem zvoleného PINu při verifikaci metodou IBM 3624 Offset je uveden níže.

Číslo účtu je reprezentováno pomocí ASCII číslic v dekadické soustavě a poté interpretováno jako hexadecimální vstup blokové šifry DES (resp. 3DES). Po zašifrování PIN generujícím klíčem je výsledek opět převeden do hexadecimální soustavy a zkrácen na první čtyři cifry. Ty však mohou obsahovat hodnoty 'A'-'F', které nejsou obsaženy na numerické klávesnici bankomatu, a proto jsou pomocí decimalizační tabulky převedeny na číslice dekadické soustavy. K výsledku je přičten offset, čímž se získá hodnota pro porovnání s PINem, který byl zadán bankomatu.

```
PAN: 4556 2385 7753 2239
Zašifrovaný PAN: 3F7C 2201 00CA 8AB3
Zkrácený zašifrovaný PAN: 3F7C
Hexadecimální číslice: 0123456789ABCDEF
Decimalizační tabulka: 0123456789012345
Decimalizovaný IPIN: 3572
Veřejně přístupný offset: 4344
Zvolený čtyřmístný PIN: 7816
```

Tento postup byl použit v nejstarších typech bankomatů, a jsou dosti rozšířeny a implementovány i v nových HSM. Validační data byla tehdy společně s offsetem uložena na kartách s magnetickým proužkem, takže jediné, co musel bankomat chránit, byl PIN generující klíč. V současné době jsou tyto metody stále používány (podporuje je například i IBM CCA API), ale verifikace PINů již neprobíhá v bankomatu, ale ve vydávající bance.

Útoky využívající známých zašifrovaných PINů

Bez újmy na obecnosti předpokládejme, že jsou používány pouze čtyřmístné PINy. Nejjednodušším útokem je využití decimalizační tabulky ke zjištění číslic, které se vyskytují v PINu. Nastavíme-li například decimalizační tabulku na samé nuly, bude (bez použití offsetu) vždy po decimalizaci vygenerovaný PIN roven čtyřem nulám.

Tímto trikem můžeme pomocí funkce generující zašifrované PINy získat EPB obsahující PIN „0000“. Jestliže nyní při verifikaci použijeme jako parametry tento EPB a „nulovou“ decimalizační tabulku, proběhne verifikace úspěšně (tj. z EPB dešifrovaný PIN je roven vygenerovanému PINu). Nechť D_{orig} je korektní decimalizační tabulka a D_i jsou nové binární decimalizační tabulky, které mají jedničku právě na těch pozicích, kde D_{orig} měla hodnotu i . Je-li například $D_{orig} = 0123 4567 8901 2345$, pak $D_5 = 0000 0100 0000 0001$ a $D_0 = 0000 0000 0100 0000$.

Nyní stačí, aby útočník pro i od 0 do 9 zavolal verifikační funkci s EPB nulového PINu a decimalizační tabulkou D_i . Není-li v hledaném PINu číslice i obsažena, změna v decimalizační tabulce se neprojeví a verifikace proběhne úspěšně. V opačném případě je i jedna z hledaných číslic PINu. K určení

všech číslic vyskytujících se v PINu je potřeba provést verifikaci maximálně desetkrát. Celkově se tak počet možných kombinací PINů omezí z 10 000 na nejvýše¹⁸ 36.

Druhá varianta útoku je efektivnější a navíc umožňuje přesně určit pořadí číslic v PINu. K její aplikaci je ale potřeba získat pět zašifrovaných hodnot známých PINů (0000, 0001, 0010, 0100, 1000). Protože bankovní systémy většinou neumožňují vkládání nezašifrovaných PINů a metodu z předchozího útoku nelze k vygenerování všech pěti hodnot PINů použít, je nutno získat je jinou cestou. Asi nejjednodušší je zadat tyto PINy bankomatu a zachytit je zašifrované poté, co dorazí do banky¹⁹. Jestliže se samotné hledání PINu implementuje, např. pomocí vhodně zkonstruovaného binárního stromu, který určí jakou decimalizační tabulku použít v dalším kroku, je možné nalézt PIN nejhůře na 24 pokusů, ale průměrně již na 15 pokusů [5].

Útok bez známého zašifrovaného PINu

Nutnou podmínkou předchozího útoku byla znalost EPB pro vybrané PINy. Následující útoky již toto nevyžadují. Předpokládejme, že se nám podařilo zachytit zákazníkův EPB obsahující správný PIN a že hodnota tohoto PINu ještě nikdy nebyla změněna (tj. offset je stále 0000). Nechť D_{orig} je původní decimalizační tabulka a D_i jsou nové decimalizační tabulky. Platí, že D_i mají hodnotu $i-1$ právě na těch pozicích, kde D_{orig} měla hodnotu i . Je-li například $D_{orig} = 0123\ 4567\ 8901\ 2345$, pak $D_5 = 0123\ 4467\ 8901\ 2344$ a $D_9 = 0123\ 4567\ 8801\ 2345$. Nyní stačí, aby útočník pro každou číslici i zavolal verifikační funkci se zachyceným EPB, správným offsetem (tj. 0000) a decimalizační tabulkou D_i . Tímto způsobem, podobně jako u prvního útoku, zjistí číslice obsažené v zákazníkově PINu. Jejich pořadí pak dokáže určit vhodnou manipulací s offsety.

Uvažme běžný případ, kdy má zákazník PIN všechny číslice odlišné. Jako příklad zvolme PIN s hodnotou 1492 a pokusme se určit pozici číslice 2. Hodnota PINu v EPB je vždy 1492, ale hodnotu generovaného PINu lze použitím decimalizační tabulky D_2 změnit na 1491. Tím se však docílí toho, že verifikace PINu neproběhne úspěšně. Nyní postupným voláním verifikační funkce s offsety 1000, 0100, 0010, 0001 budeme naopak zvyšovat jednotlivé číslice generovaného PINu. Pouze v případě offsetu 0001 se však jeho hodnota vrátí zpět na 1492 a verifikace proběhne úspěšně. Použitím offsetu je pak jednoznačně určeno, která číslice PINu byla upravena. Ve skutečnosti se dokonce poslední verifikace ani nemusela provádět, protože nebyla-li hledaná číslice na předchozích třech pozicích, musela být na čtvrté.

Tímto způsobem může útočník určit pozice všech číslic, k čemuž v případě čtyřmístného PINu složeného z čtyř různých číslic potřebuje nejvýše 6 volání verifikační funkce (tři pro nalezení pozice první číslice, dvě pro nalezení pozice druhé číslice a jedno pro nalezení pozice třetí číslice). Poznamenejme, že tento útok je mírnou modifikací původního útoku z [5].

3.3.2 Check Value Attack

Jedná se o útok, který k získání PINů zneužívá funkci standardního API určenou k testování korektnosti zašifrovaných DES klíčů. Ta mimo jiné umožňuje šifrování 64bitového vzorku binárních nul. Zopakujme, že verifikační funkce zašifruje validační data PIN generujícím klíčem a výslednou hodnotu zkrátí a decimalizuje – tím je vytvořen IPIN. Testovací funkce tak umožňuje útočníkovi získat IPIN (stačí jen zkrátit a decimalizovat výsledek získaný testovací funkcí). Použitím verifikační funkce a manipulací s offsetem k danému EPB pak získá PIN.

3.3.3 ANSI X9.8 Attacks

Existence PIN-bloků dává možnost definovat příslušný formát. Výsledkem je, že existuje několik formátů, které jsou široce používány. Pro výrobce HSM to ovšem znamená nutnost implementovat několik funkcí pro verifikaci PINů pro jednotlivé formáty a překladové funkce mezi formáty.

¹⁸ V případě čtyřmístného PINu složeného z tří různých číslic. U čtyř různých číslic by kombinací bylo pouze 24 a u dvou číslic jen 14.

¹⁹ To je zároveň nejjednodušší metoda, jak známý zašifrovaný PIN získat, protože použití funkce generující PINy bývá do značné míry omezeno (a její využití v prvním útoku bylo spíše ilustrativní).

Z tohoto faktu vznikly útoky zneužívající špatného návrhu a implementace těchto funkcí používaných při verifikaci a překladu (změně formátů) PINů. Základním předpokladem těchto útoků je nízká entropie v PIN-blocích, jejímž důsledkem je nemožnost rozpoznat, který formát byl pro vytvoření daného PIN-bloku použit. To umožňuje zbavit se validačních dat v PIN bloku pomocí překladových funkcí (některé formáty je neobsahují), vložit validační data podle vlastního výběru, nebo se dostat k částem PIN-bloku, které by při použití jednoho formátu byly pro útočníka nedosažitelné. Detaily útoků jsou popsány v [14].

3.3.4 Další rozpracované útoky

Další dva útoky (PIN Derivation Attack a Collision Attack) byly objeveny Mikem Bondem, který se společně s Jolyonem Clulowem problematikou API na univerzitě v Cambridge zabývá. V době psaní tohoto příspěvku však ještě nebyly veřejně publikovány, byly s autory diskutovány prostřednictvím osobní e-mailové korespondence a informace o nich budou případně podány při prezentaci na konferenci EurOpen.

3.4 Nevynucení politiky – PKCS #11

Tato závěrečná část příspěvku se zabývá možnostmi, které útočníkovi nabízí nevhodné použití API, které neobsahuje a nevynucuje žádnou bezpečnostní politiku. Až doposud jsme se zabývali útoky, které byly aplikovatelné především na IBM CCA či jemu podobná kryptografická API. Ta byla navržena pro konkrétní kryptografické moduly a implementovala konkrétní bezpečnostní politiku. Přesto se jejich bezpečnost ukázala jako nedostatečná.

Nyní se zaměříme na velmi oblíbené rozhraní PKCS #11 [15], které je také často používáno jako hlavní API pro kryptografické moduly (implementace je i součástí operačních systémů). Oproti předchozím však bylo navrženo pouze jako standardní rozhraní mezi aplikacemi a jednorázovými bezpečnostními zařízeními. Hlavní problém tohoto API je, že je to pouze sada funkcí a neobsahuje žádnou politiku, která by například zajistila konzistentnost vlastností klíčů. Srovnání základních rysů CCA API a PKCS #11 pro IBM 4758 je uvedeno v [13]. Dále v této části budeme vycházet z [8, 15].

3.4.1 Základní informace o PKCS #11

Podle terminologie PKCS #11 jsou hardwarová bezpečnostní zařízení uchováající objekty (např. data, klíče či certifikáty) a provádějící kryptografické operace nazývána *tokens*. K jejich použití je nutné vždy vytvořit logické spojení s aplikací, což vyžaduje, aby se uživatel nejprve řádně přihlásil. Proběhne-li autentizace úspěšně, může pak prostřednictvím funkcí API s tokenem komunikovat. Uchovávané objekty se dělí podle jejich *viditelnosti* a *životnosti*. *Token objects* jsou stálé objekty, které jsou viditelné všem přihlášeným aplikacím s dostatečnými právy. *Session objects* jsou oproti tomu pouze dočasné objekty, které přetrvávají během vytvořeného spojení a jsou viditelné jen pro aplikaci, která je vytvořila.

Každý objekt je dále asociován s množinou vlastností, které popisují jeho typ a určují jeho použití. Například objekt klíč je vždy typu veřejný, soukromý či tajný, přičemž poslední dva z těchto typů mohou být navíc označeny jako *citlivé* či *neextrahovatelné*. Klíč, který je označen jako citlivý, nemůže být nikdy v otevřené podobě exportován mimo token. Neextrahovatelný klíč pak nemůže být exportován ani když je zašifrován. Tyto vlastnosti však nejsou s klíčem nijak kryptograficky svázány a importující aplikace si je tedy může libovolně upravit.

Oproti CCA API definuje PKCS #11 jen dva typy uživatelů: normální uživatele a bezpečnostní úředníky. Normální uživatel má po autentizaci možnost přistupovat k jednotlivým objektům a využívat kryptografických funkcí tokenu. Bezpečnostní úředník je zodpovědný za inicializaci tokenu a počáteční nastavení uživatelského hesla či PINu. Na rozdíl od normálních uživatelů nemůže provádět žádné kryptografické operace. Cílem PKCS #11 je poskytnout uchovávaným objektům dostatečnou ochranu před odhalením (např. označením klíčů jako citlivé a neextrahovatelné), ale není záměrem chránit objekty jednoho uživatele před použitím jinými uživateli.

3.4.2 Symmetric Key Attacks

V de facto nezměněné podobě lze na PKCS #11 aplikovat útoky *Key Conjuring* i *Meet in the Middle*. Další útoky pak většinou zneužívají podobných nedostatků jako útoky na CCA API.

3DES Key Binding Attack

Nedostatečná vazba mezi jednotlivými polovinami 3DES klíče umožňuje provést útok na každou jeho polovinu zvlášť. Označme požadovaný klíč jako K a jeho jednotlivé poloviny jako K_1, K_2 . Při exportu klíče je každá polovina nezávisle zašifrována pomocí KEK (key encryption key) a platí tedy, že $E_{KEK}(K) = (E_{KEK}(K_1), E_{KEK}(K_2))$. Jednotlivé poloviny pak lze nezávisle na sobě importovat jako standardní DES klíč a zašifrovat jimi nějaký testovací vzorek. K jejich nalezení (hledá-li útočník oba klíče současně) pak stačí prohledat prostor přibližně 2^{56} klíčů.

Oproti předchozím API se v tomto případě útočník nemusí obávat problémů s kontrolou přístupu. Abychom předešli tomuto útoku, API by nemělo umožňovat, aby byl exportovaný klíč modifikován. Toho lze dosáhnout například použitím MAC²⁰.

Key Separation Attack

Jak již bylo zmíněno dříve, PKCS #11 specifikuje pro každý objekt typu klíč množinu vlastností, které určují k čemu smí být použit. Bezpečnostní chybou API je, že umožňuje konfliktní nastavení těchto vlastností a umožňuje tak kompromitování klíčů. Je-li například klíč označen jako KEK a zároveň jako klíč určený k dešifrování dat, lze jeho pomocí exportovat z tokenu libovolný extrahovatelný chráněný klíč a poté jej jednoduše jako data dešifrovat. Abychom předešli tomuto útoku, měla by být v API volba vlastností objektů mnohem více omezující. Tyto vlastnosti by navíc měly být s daným objektem nějak kryptograficky svázány.

Weaker Key/Algorithm Attack

PKCS #11 umožňuje zašifrování libovolného klíče pomocí algoritmů používajících klíč krátké délky (např. RC2 či DES). Útočník pak například nejprve exportuje požadovaný 3DES klíč K zašifrovaný pomocí standardního DES KEK (tj. $E_{KEK}(K)$). Poté exportuje KEK pomocí sebe sama (tj. $E_{KEK}(KEK)$) a útokem hrubou silou zjistí jeho hodnotu. Pomocí známého KEK pak již snadno získá hodnotu klíče K . Tímto je zcela degradována bezpečnost silných kryptografických algoritmů, a API by proto vůbec nemělo k exportu klíčů použití slabších algoritmů umožňovat.

Related Key Attack

Podobně jako útok *Meet in the Middle* lze i tento útok aplikovat na klíče, které jsou označeny jako neextrahovatelné. Podívejme se nejprve, jak lze pomocí páru souvisejících klíčů $K_1=(K_A, K_B, K_C)$ a $K_2=(K_A \oplus \text{DELTA}, K_B, K_C)$ učinit tří-klíčový 3DES jen nepatrně silnější než standardní DES. Útočník pouze zašifruje testovací vzorek P klíčem K_1 a dešifruje klíčem K_2 , čímž získá:

$$P' = D_{K_A \oplus \text{DELTA}}(E_{K_B}(D_{K_C}(E_{K_C}(D_{K_B}(E_{K_A}(P)))))) = D_{K_A \oplus \text{DELTA}}(E_{K_A}(P)).$$

Tím zcela izoloval část K_A , kterou teď může hrubou silou hledat nezávisle na částech K_B a K_C . Prohledávaný klíčový prostor se tím redukoval průměrně na 2^{55} a k nalezení K_A bude potřeba provést průměrně 2^{56} operací DES. Použije-li útočník 2^{16} souvisejících párů klíčů, pak lze aplikací *Meet in the Middle* útoku prohledávaný klíčový prostor redukovat až na 2^{39} .

Reduced Key Space Attack

Jednou z možností, jak z existujícího klíče vytvořit nový klíč, je výběr části jeho bitů (jedna z funkcí umožňuje vytvořit nový klíč výběrem části bitů z klíče existujícího). Ukažme například, jak lze použitím této metody z 32bitového tajného klíče s hodnotou $0x329F84A9$ vytvořit nový 16bitový tajný klíč. $0x329F84A9$ je binárně zapsáno jako 0011 0010 1001 1111 1000 0100 1010

²⁰ Message Authentication Code – tvoří podtřídou klíčovaných hašovacích funkcí a jejich cílem je zajištění integrity a autenticity zpráv.

1001 a jako počáteční pozici, od níž začne výběr nového klíče, stanovme bit b_{21} . Následujících 16 bitů $b_{21} \dots b_{31} b_0 \dots b_4$ pak vytvoří binární řetězec 1001 0101 0010 0110, kterým je jednoznačně určena hodnota nově vytvořeného klíče jako $0x9526$.

Toho lze snadno využít ke zmenšení prohledávaného prostoru klíčů. Útočník například nejprve použitím čtyřiceti po sobě jdoucích bitů z 56bitového DES klíče vytvoří 40bitový RC2 klíč (taková změna typu klíče je možná). Ten pak hrubou silou dešifruje a s jeho pomocí najde zbývajících 16 bitů původního DES klíče. Tento útok lze opět aplikovat i na neextrahovatelné klíče.

3.4.3 Public Key API Attacks

V následující části demonstrujeme útoky, které se opírají o podporu API pro kryptografické operace s veřejným klíčem a umožňují kompromitování soukromých nebo tajných klíčů.

Small Public Exponent with No Padding Attack

Problémem tohoto API je, že umožňuje používání funkcí, které už jsou zastaralé. Byl-li k šifrování klíče použit asymetrický algoritmus RSA bez doplnění (tzv. X.509 Raw RSA), je tento klíč z řetězce znaků pouze překonvertován na číslo, zašifrován a tento výsledek je opět převeden zpět na řetězec znaků. Je-li tedy veřejný RSA klíč dvojice m a e , lze operaci šifrování zapsat jako $C=K^e \bmod n$. Tato metoda je však v případě použití malé hodnoty veřejného exponentu napadnutelná, protože pokud $K^e < n$, tak je klíč možno jednoduše dešifrovat jako $K=C^{1/e}$. Je-li například požadováno, aby měl veřejný klíč z důvodů zvýšení rychlosti umocňování nízkou Hammingovu váhu, je vygenerování klíče s malou hodnotou exponentu poměrně pravděpodobné²¹. Tomuto útoku se dá snadno předejít zakázáním používání malých hodnot exponentu nebo použitím RSA s doplněním (tzv. PKCS #1 RSA).

Trojan Public Key Attack

Protože PKCS #11 ukládá veřejné klíče bez jakýchkoliv dalších integritních či autentizačních informací, může útočník snadno do tokenu vložit vlastní veřejný klíč. Jeho pomocí zašifruje a vyexportuje klíče, které pak pomocí svého soukromého klíče lehce dešifruje. Tímto jednoduchým útokem lze kompromitovat symetrické i asymetrické klíče, které nejsou označeny jako neextrahovatelné. API by tedy před použitím veřejného klíče k exportu citlivých informací mělo být schopno přinejmenším ověřit jeho původ.

Trojan Wrapped Key Attack

Podobně jako u předchozího útoku neumožňuje PKCS #11 zjistit ani původ libovolného zašifrovaného klíče. Obsahuje-li příslušný token řádný veřejný a soukromý klíč, může útočník jednoduše importovat svůj vlastní symetrický tajný klíč. Stačí jej nejprve zašifrovat veřejným klíčem a následně importovat. Tento klíč pak lze použít k exportu jiných klíčů ze zařízení a následně k jejich dešifrování. API by tedy mělo umožňovat ověřit i původ zašifrovaných klíčů určených k importu.

Private Key Modification Attack

V PKCS #11 mohou být soukromé klíče exportovány či importovány pouze tehdy, obsahují-li kromě soukromého exponentu a modulu také veřejný exponent a koeficienty CRT (tj. n , p , q , e , d , $d \bmod p-1$, $d \bmod q-1$ a $q-1 \bmod p$). Nyní uvažme situaci, kdy je soukromý RSA klíč zašifrován nějakým symetrickým tajným klíčem a exportován. Šifrování probíhá pomocí modu CBC a modifikace jednoho zašifrovaného bloku tedy způsobí změnu dvou bloků zašifrovaných dat. Protože celková délka zašifrovaných dat závisí především na velikosti asymetrických klíčů (typicky 512, 1024 nebo 2048 bitů), je pravděpodobné, že modifikace zašifrovaného bloku ovlivní nezávisle na ostatních datech pouze hodnotu samotného klíče. Jeho importováním pak získá útočník v tokenu částečně změněný klíč, který může použít k provedení útoku analýzou chyb [6]. V ideálním případě by tedy u soukromých klíčů měla být zajištěna integrita (např. použitím MAC nebo prováděním základních aritmetických testů typu $d \equiv e^{-1} \bmod n$ a $n=pq$).

²¹ Například CCA API umožňuje generování asymetrických klíčů přímo s hodnotami veřejného klíče 3 nebo $2^{16}+1$.

4 Závěr

Příspěvek vznikl na základě (a podrobnější informace lze získat v) [14] – diplomové práce J. Krhovjáka, vedené V. Matyášem a oponované D. Cvrčkem.

Seznámili jsme se s bezpečnostní problematikou a základní architekturou HSM a byly představeny bezpečnostní požadavky na zařízení splňující stěžejní normu pro tuto oblast – FIPS 140-2 [11].

Hlavním cílem celého příspěvku byla prezentace útoků na a přes aplikační programovací rozhraní. Mnohé z těchto útoků byly aplikovatelné i na v dnešní době běžně používané IBM CCA API. Naši pozornosti neuniklo ani velmi oblíbené aplikační programovací rozhraní PKCS #11, ale jeho specifikace bohužel ponechává mnoho implementačních detailů na konkrétním výrobcí HSM, což analýzu jednotlivých útoků dosti ztížilo.

I přesto, že cílem tohoto příspěvku nebylo stanovit, jak by měl bezpečný návrh kryptografických API vypadat, je porozumění chybám ve stávajících API prvním krokem, jak toho dosáhnout.

Reference

- [1] R. J. Anderson. The Correctness of Crypto Transaction Sets. In *Proceedings of 8th International Workshop on Security Protocols*, volume 2133 of *Lecture Notes in Computer Science*, pages 125–127. Springer, April 2000.
- [2] R. J. Anderson, M. Bond. API-Level Attacks on Embedded Systems. In *IEEE Computer*, volume 34, pages 67–75, October 2001.
- [3] M. Bond. A Chosen Key Difference Attack on Control Vectors, November 2001.
- [4] M. Bond. Attacks on Cryptoprocessor Transaction Sets. In *Cryptographic Hardware and Embedded Systems*, volume 2162 of *Lecture Notes in Computer Science*, pages 220–234. Springer, 2001.
- [5] M. Bond, P. Zielinski. Decimalisation Table Attacks for PIN Cracking. Technical Report 560, University of Cambridge, Computer Laboratory, February 2003.
- [6] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults. In *EUROCRYPT*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer, December 1997.
- [7] J. S. Clulow. PIN Recovery Attacks. Technical Report 0520 00296, Prism, October 2001. Revised, October 2002.
- [8] J. S. Clulow. On the Security of PKCS #11. In *Cryptographic Hardware and Embedded Systems*, volume 2779 of *Lecture Notes in Computer Science*, pages 411–425. Springer, 2003.
- [9] J. S. Clulow. The Design and Analysis of Cryptographic Application Programming Interfaces for Security Devices. Master's thesis, University of Natal, January 2003.
- [10] Federal Information Processing Standards Publication 140-1, Security Requirements for Cryptographic Modules, National Institute of Standards and Technology, January 1994.
- [11] Federal Information Processing Standards Publication 140-2, Security Requirements for Cryptographic Modules, National Institute of Standards and Technology, May 2001.

- [12] Federal Information Processing Standards Special Publication 800-29, A Comparison of the Security Requirements for Cryptographic Modules in FIPS 140-1 and FIPS 140-2, National Institute of Standards and Technology, June 2001.
- [13] IBM. Comparison of CCA and PKCS #11 v2.01 for the IBM 4758. White paper.
- [14] J. Krhovják. Analýza útoků na aplikační programovací rozhraní pro hardwarová bezpečnostní zařízení. Master's thesis, Masaryk University Brno, 2004. Available at http://www.fi.muni.cz/~xkrhovj/apinf/sdipr/DP_upravena_v1.pdf.
- [15] RSA Security Inc. Cryptographic Token Interface Standard – PKCS #11, Version 2.11, November 2001.