Masarykova univerzita
Fakulta informatiky

Analýza útoků na aplikační programovací rozhraní pro hardwarová bezpečnostní zařízení

Dokument vycházející z diplomové práce

Verze 1.0

Jan Krhovják

15. září 2004

Shrnutí

Cílem této práce, která vznikla rozšířením a vylepšením mé diplomové práce, je popis architektury a principů fungování hardwarových bezpečnostních zařízení, analýza útoků na jejich aplikační programovací rozhraní a podrobná studie jednoho z těchto zařízení. V první části uvedeme základní požadavky na bezpečnost kryptografických modulů a několik technik útoků, kterým musí být schopna tato zařízení odolávat. Druhá část se již bude zabývat útoky na jejich aplikační programovací rozhraní, přičemž zvláštní pozornost bude věnována vysoce zabezpečeným kryptografickým modulům, které podporují provádění finančních transakcí a jsou určeny především k nasazení v bankovním sektoru. Třetí část práce se bude týkat programovatelných kryptografických koprocesorů firmy IBM.

Poděkování

Mé poděkování patří Dr. Václavu Matyášovi, jehož odborné vedení a cenné rady významně přispěly při vzniku této práce. Také děkuji Danielu Cvrčkovi, který práci několikrát velmi pečlivě pročetl, a jehož rady pomohly odstranit spoustu nejasností a nepřesností.

Obsah

Kapitola 1

Úvod

Zajištění bezpečné komunikace především u distribuovaných systémů a aplikací vyžaduje použití vhodných mechanizmů a prostředků pro zajištění integrity jejich kryptografických funkcí a důvěrnosti příslušných šifrovacích klíčů. U běžně používaných výpočetních systémů toho nelze dosáhnout, 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 bankovnictví a nutnost zabezpečení stále se zvyšujícího počtu elektronických transakcí, které probíhají především v bankovních sítích (např. VISA, MasterCard, American Express) při vzájemné komunikaci jednotlivých bank a při komunikaci s jejich peněžními bankomaty (ATM). Tyto rozsáhlé ATM sítě[1], 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), čímž by se mělo zcela předejít podvodům ze strany klientů i zaměstnanců bank.

Hardwarová bezpečnostní zařízení používaná k ochraně finančních transakcí musí splňovat velmi přísné bezpečnostní požadavky, a jedinou možností, jak manipulovat s chráněnými citlivými informacemi uvnitř zařízení, je použití přesně definovaného softwarového rozhraní – tzv. aplikačního programovacího rozhraní (API). To by mělo být navrženo tak, aby nemohlo dojít k jakémukoliv zneužití či kompromitování chráněných dat. 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 této práci podrobně zabývat.

Zcela odlišnou třídu tvoří programovatelná hardwarová bezpečnostní zařízení, jejichž software a firmware může být nezávisle na výrobci snadno aktualizován či

[1] V této práci 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.

přeprogramován. Tím je na jednu stranu zcela vyřešen problém oprav případných bezpečnostních chyb, které se ve firmware a software (tj. i v API) mohou objevit, ale je zase nutné zajistit, aby jejich změny nemohly nijak ohrozit důvěrnost a integritu již chráněných dat.

Rozvržení kapitol

V druhé kapitole se nejprve blíže seznámíme se základní architekturou HSM, s požadavky kladenými na jejich bezpečnost, a demonstrujeme několik technik fyzických útoků, kterým musí být schopna tato zařízení odolávat. V třetí kapitole se již budeme zabývat problematikou bezpečnosti API a budou představeny útoky na API starších i současných HSM. Zvláštní pozornost bude věnována útokům umožňujícím získání PINů. Ve čtvrté kapitole se nakonec podrobněji seznámíme s návrhem programovatelných kryptografických koprocesorů firmy IBM.

Tato práce se opírá o základní znalost kryptografického koprocesoru IBM 4758, jehož popis je obsažen v příloze A, a o pochopení základních principů CCA API, které je zpracováno v příloze B. V případě, že čtenář není s tímto systémem nebo problematikou nijak obeznámen, je doporučeno přednostní nastudování těchto příloh. V příloze C a D jsou pak uvedeny detaily útoků na CCA API a PKCS #11.

Kapitola 2

Bezpečný hardware

Tato část práce je koncipována jako úvod do problematiky hardwarových bezpečnostních zařízení. Jejím cílem je seznámit se s jejich základními typy, architekturou, požadovanou úrovní zabezpečení a také se známými útoky, jimž by měla odolávat.

2.1 Základní terminologie a pohled na bezpečnost

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

HSM 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*, zvlášť pokud je umístěno na veřejně přístupném místě a je bez dostatečné fyzické ochrany.

Jako *útok* na hardwarové bezpečnostní zařízení lze označit postup, pomocí něhož lze získat[1] data, která mají být pro útočníka nepřístupná, nebo postup, kterým útočník provede operaci, ke které není autorizován.

Protože kryptografické moduly jsou určeny především k nasazení v nedůvěryhodných prostředích, je již při jejich návrhu nezbytné zvážit a vyřešit veškerá s tímto související bezpečnostní rizika. Jak je popsáno v [48], lze obecně požadavky na bezpečnost kryptografických modulů rozdělit do kategorií *fyzická bezpečnost*, *logická bezpečnost* a *bezpečnost prostředí*.

2.1.1 Fyzická bezpečnost

Pojem fyzická bezpečnost v souvislosti s informační bezpečností zahrnuje technologie použité k zabezpečení informace proti fyzickému útoku [48]. Jedná se o vytvoření

[1] V kontextu této práce není tedy útokem chápáno ani vyřazení zařízení z provozu ani žádná forma DoS (Denial of Service) útoku.

ochranné vrstvy, která obklopuje výpočetní systém a zabezpečuje jej proti neautorizovanému fyzickému přístupu (tj. vytváří bezpečný prostor). V současné době, kdy dochází k masovému nasazení výpočetních systémů v nedůvěryhodném prostředí (např. bankomaty a platební terminály), se stává fyzická bezpečnost naprosto klíčovou. S rostoucí cenou dat a informací dochází také k častým pokusům o průnik do těchto systémů a vznikají nové typy útoků, na něž je třeba včas reagovat.

Mezi přístupy běžně používané k fyzickému zabezpečení kryptografických modulů patří:

- Evidence průniků (tamper[2] evidence) – zajišťuje zanechání stop signalizujících vniknutí do zařízení a/nebo narušení jeho bezpečnosti. Je většinou realizována chemickými či mechanickými prostředky (např. speciální barviva, holografické pásky, pečetě, zámky). Tento systém je však účinný pouze je-li součástí nějaké bezpečnostní politiky[3] (pokud nikdo stopy po průniku nehledá, nebudou nikdy nalezeny).

- Odolnost proti průnikům (tamper resistance) – zajišťuje (do jisté úrovně) odolnost proti fyzickému vniknutí do zařízení. Typicky je realizována chemicky odolnými látkami či masivními ocelovými kryty, jejichž odstranění zpomalí útok a je k němu potřeba vynaložit značné úsilí. V případě čipových karet je odolností proti průniku míněno použití ochranných vrstev na čipu, plastového obalu čipu apod.

- Detekce průniků (tamper detection) – zajišťuje detekci průniků, jíž bývá dosaženo použitím speciálních elektrických obvodů připojených k vnějšímu krytu či k jiným ochranným prvkům.

- Odpověď na průniky (tamper response) – jsou mechanismy vyvolávané při detekci průniku a zabraňující získání citlivých informací a dat chráněných uvnitř modulu po narušení fyzické ochrany. Typicky jsou realizovány jejich zničením (např. vymazáním pamětí či zničením čipu chemickými látkami).

Mírná významová odlišnost zde spočívá v tom, že evidence a odolnost proti průnikům zajišťují pasivní ochranu modulů, zatímco detekce a odpověď na průniky zajišťují aktivní ochranu modulů (tj. jsou to dva různé pohledy). Základní útoky na výše uvedené techniky a metody obrany proti nim jsou zdokumentovány v [48].

2.1.2 Logická bezpečnost

Logická bezpečnost zahrnuje mechanizmy, jimiž se operační systémy či jiný software snaží předejít neautorizovanému přístupu k citlivým informacím či datům [48]. Obecně lze tyto mechanizmy rozdělit na tři části:

- Kryptografické algoritmy – jsou matematické funkce, jejichž hlavním cílem je zajištění důvěrnosti, integrity, autentizace a nepopiratelnosti. Na jejich bezpečnosti a správném použití stojí podstatná část soudobé kryptografie.

[2] Tampering – neautorizovaná modifikace zařízení, která může ovlivnit jeho bezpečnost a funkčnost.

[3] Tímto není myšlena bezpečnostní politika zařízení (jejíž součástí je evidence průniků vždy), ale bezpečnostní politika celého systému.

- Kryptografické protokoly – jsou metody popisující vzájemnou komunikaci mezi jednotlivými zařízeními. V podstatě se jedná o distribuované algoritmy, kdy jsou jednotlivé kroky „propojeny“ komunikací uskutečňovanou přes prostředí bez fyzické bezpečnosti (tj. pomocí funkcí API).

- Kontrola přístupu – má oproti algoritmům a protokolům do značné míry omezenu použitelnost. Nutným předpokladem je existence důvěryhodného prostředí.

Podrobná studie kryptografických algoritmů a protokolů však převyšuje rámec této práce a je dobře zdokumentována například v [14, 39].

2.1.3 Bezpečnost prostředí

Oproti fyzické bezpečnosti, kde hlavním aktivem je informace, zajišťuje bezpečnost prostředí ochranu samotného zařízení a jejím hlavním cílem je omezení možnosti provést útok (tj. například odcizení či zničení zařízení) [48]. Typicky je jí dosaženo kontrolou či omezením fyzického přístupu k zařízení (např. použitím ozbrojených stráží, bezpečnostních kamer či identifikačních karet, jejichž pomocí může být umožněn/omezen přístup do místnosti se zařízením), a je tedy závislá na jeho umístění (tj. na prostředí).

Ačkoliv je z pohledu naší analýzy nejméně podstatná, je právě ona jednou z hlavních příčin vedoucích k selhání bezpečnostních mechanizmů, a proto by měla být vždy součástí bezpečnostní politiky.

2.1.4 Vnitřní architektura HSM

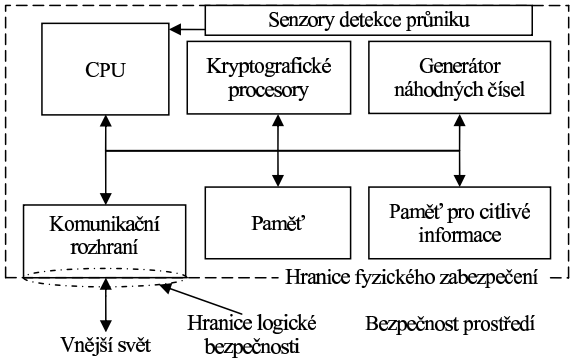
Architektura HSM je do značné míry závislá na jejich typu. Základní architektura vycházející z klasické von Neumannovy architektury je oproti běžným počítačům rozšířena o mechanizmy fyzické ochrany, generátory náhodných čísel, či speciální koprocesory pro urychlení specifických kryptografických operací. Na druhou stranu neobsahují HSM například běžné vstupní a výstupní (V/V) obvody, což značně snižuje složitost operačního prostředí.

Je zřejmé, že odolnost proti průnikům nebude u čipových karet nikdy realizována stejným způsobem jako u kryptografických koprocesorů či bankomatů. Jestliže ovšem porovnáme funkční schémata, obsahují stejné základní bloky. V následující části textu budeme vycházet z [42, 49].

Kryptografické koprocesory a akcelerátory

Kryptografické koprocesory a akcelerátory jsou většinou rozšiřující přídavné karty. Typicky obsahují procesor, víceúčelovou paměť, komunikační rozhraní, paměť pro citlivé informace, generátory náhodných čísel, procesor určený na urychlení kryptografických operací a systém pro detekci průniků pomáhající zajistit fyzickou bezpečnost. Poslední čtyři komponenty odlišují koprocesory a akcelerátory od víceúčelových zařízení. Pro zajištění ochrany před vnějším prostředím mohou být také vybaveny dalšími obvody pro zajištění předepsaných provozních podmínek na V/V portech.

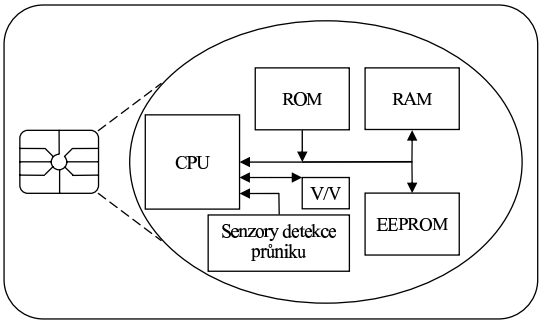
Jádrem celého zařízení je procesor (CPU), který řídí veškeré V/V operace, zpracovává přerušení a stará se o správu paměti. Navíc úzce spolupracuje s dalšími procesory sloužícími k provádění specializovaných kryptografických operací. Paměť určená pro uchování citlivých informací je většinou napájena přídavnými bateriemi a obsahuje tajné klíče, které mohou být v případě detekce průniku vymazány. Nezbytnou součástí kryptografických modulů je i hardwarový generátor náhodných čísel, s jehož pomocí lze tyto klíče generovat. Vnitřní architektura těchto zařízení je spolu s vyznačením jednotlivých oblastí zabezpečení zobrazena výše na obrázku 2.1. Typickým příkladem programovatelného kryptografického koprocesoru je IBM 4758 (viz příloha A).



Obr. 2.1: Vnitřní architektura HSM.

Čipové karty

Na kryptografické čipové karty (smart cards) lze pohlížet jako na levné kryptografické moduly s menší výpočetní silou. Jejich hlavní oblastí nasazení je vykonávání kryptografických operací vyžadujících tajný klíč, který nesmí být odhalen. Hlavní výhodou čipových karet oproti kartám s magnetickým proužkem je, že uložená data mohou být chráněna proti neautorizovanému přístupu a modifikaci. Čipové karty se typicky skládají (viz obr. 2.2) z 8, 16, nebo 32bitového procesoru, pamětí ROM, EPROM, z malého množství RAM potřebného k provádění výpočtů a z komunikačního V/V rozhraní. Operační systém je trvale uložen v ROM a řídí například ukládání či načítání dat z energeticky nezávislé EEPROM. V závislosti na typu komunikačního rozhraní se současná generace čipových karet dělí na:



Obr. 2.2: Vnitřní architektura čipové karty.

- Kontaktní – jednočipové karty, jejichž komunikační rozhraní vyžaduje přímý (tj. fyzický) kontakt s „čtečkou“ karet.
- Bezkontaktní – jednočipové karty obsahující anténu, jejíž prostřednictvím je na krátkou vzdálenost realizována komunikace s „čtečkou“ karet (tj. není vyžadován přímý fyzický kontakt).
- Kombinované – jednočipové karty s kontaktním i bezkontaktním komunikačním rozhraním. Toto řešení spojuje výhody předchozích dvou typů čipových karet.

- Hybridní – karty obsahující dva vzájemně nepropojené čipy a případně i magnetický proužek. Jeden z čipů je většinou s kontaktním a druhý s bezkontaktním komunikačním rozhraním. Tímto způsobem lze kombinovat různé typy karet, čímž je dosaženo flexibilnějšího řešení.

Ostatní hardwarová bezpečnostní zařízení

Existuje spousta dalších typů hardwarových bezpečnostních zařízení, avšak většinou se jedná o různé modifikace či kombinace zařízení popsaných výše. Patří mezi ně například:

- Kryptografické switche a routery – jedná se o síťové prvky, které jsou navrženy především k zabezpečení finančních transakcí v bankovních sítích.
- USB čipy – mají stejnou funkcionalitu jako čipové karty, avšak k přenosu dat využívají rozhraní USB.
- Super čipové karty (super smart cards) – mají také stejnou funkcionalitu jako čipové karty, ale navíc mají integrovánu klávesnici, display a případně i solární panel.

Mezi nejvýznamnější výrobce hardwarových bezpečnostních zařízení patří například HP-Atalla [26], Cisco, Chrysalis, Eracom [21], IBM [32], nCipher [41], Racal a Thales.

2.2 Bezpečnostní požadavky na kryptografické moduly

Bezpečnostní požadavky na kryptografické moduly jsou specifikovány v normě FIPS[4] 140-2 [23], která od 25. listopadu 2001 zcela nahradila starší normu FIPS 140-1 [22] z roku 1994. Tento standard 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 2.2.1) vztahujících se k bezpečnému návrhu a implementaci modulu. Vyšší úrovni 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 mechanizmy 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, které bývá nejčastěji dosaženo použitím kvalitních zámků či pečetí. Požadována je přinejmenším autentizace založená na *rolích*[5]. Tato úroveň povoluje spouštění softwarových a firmwarových částí kryptografického modulu na obecném počítačovém systému pouze s operačním systémem[6], který

[4] Federal Information Processing Standard.

[5] Při tomto typu autentizace nemusí být ověřena identita operátora.

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

splňuje alespoň úroveň EAL2 ohodnocenou podle CC[7]. Typickými příklady zařízení spadajících do této úrovně jsou čipové karty.

Úroveň 3 – dále rozšiřuje požadavky na fyzické zabezpečení. Útočníkovi již musí být zabráněno získat přístup k citlivým informacím udržovaným uvnitř modulu. Toho bývá dosaženo použitím silných ochranných krytů a speciálních obvodů, které pokus o fyzický průnik detekují a 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 s jinými zařízeními. Softwarové a firmwarové části modulu mohou být spouštěny pouze na operačním systému, 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 nedodržení 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 musí projít sérií přísných testů, jež by prokazovaly, že je schopen chránit citlivá data i při práci mimo rozsah jeho operačních hodnot. Softwarové a firmwarové části modulu mohou být spouštěny pouze na operačním systému, který splňuje alespoň úroveň EAL4 ohodnocenou podle CC. Příkladem zařízení splňujícího FIPS 140-1 a spadajícího do této úrovně je IBM 4758.

2.2.1 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

Tato oblast explicitně definuje požadavky na dokumentaci všeho co je uvnitř kryptografické hranice modulu. Dokumentace by měla specifikovat hardwarové, firmwarové a softwarové komponenty kryptografického modulu, kryptografickou hranici modulu a fyzickou bezpečnost modulu. Dále by pak měla specifikovat fyzické porty, logická

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

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 vztahující se 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 fyzickým portům a logickým rozhraním tím, že budou definovány všechny vstupní a výstupní body modulu a rozhraní budou alespoň kryptograficky odděleny. Norma specifikuje čtyři logická rozhraní pro vstup a výstup. První je určené pro vstup příkazů, signálů a kontrolních dat. Druhé pro výstup stavových dat, signálů či fyzických indikátorů stavu (tj. LED diody a displeje). Zbylé 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 operátora[8] podporovat autorizované *role*, s přiřazenými službami. Pokud jsou podporovány paralelní sezení, musí být logicky zcela odděleny (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. Navíc může být požadována také *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.

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

2. Autentizace založená na identitách – oproti předchozímu případu je navíc testována i identita operátora.

Při implementaci těchto mechanizmů 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 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 mechanizmy 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

Operační prostředí kryptografického modulu se vztahují na správu softwarových, firmwarových a/nebo hardwarových součástí, které jsou požadovány pro správné fungování modulu. Operační prostředí jsou zde rozdělena na modifikovatelná (např. firmware v RAM) a nemodifikovatelná (např. firmware v ROM), přičemž zvláštní důraz je kladen na operační systém, který je důležitou součástí operačních prostředí modulu (EAL pro operační systém musí být vždy doložen).

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 pomocí kryptografického modulu chráněny před neautorizovaným čtením, modifikací či záměnou.

Elektromagnetické interference a kompatibilita

V této sekci jsou definovány požadavky na elektromagnetické interference a kompatibilitu pro FCC[9].

Auto-testování

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.

Zaručení 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). Některé z těchto útoků jsou podrobněji popsány dále v části 2.3.

2.2.2 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 SP[10] 800-29 [24]. 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 mechanizmy a na testování modulu. V důsledku vzniku nových norem je již také operační systém ohodnocován podle CC (namísto TCSEC[11]).

[9] Federal Communications Commission – americká nezávislá vládní agentura zodpovědná za regulaci federálních komunikací.

[10] National Institute of Standards and Technology Special Publication.

[11] Trusted Computer System Evaluation Criteria – někdy též označováno jako „oranžová kniha“.

2.3 Útoky na fyzickou bezpečnost

Tato část pojednává o některých pokročilých útocích na fyzickou bezpečnost a o jejich snadné aplikaci na konkrétní kryptografické moduly[12]. Nejdříve se ovšem pokusíme o jednoduchou klasifikaci útočníků. V [1] byla navržena taxonomie útočníků následovně:

Třída I (clever outsiders): Jsou velmi často inteligentní, ale mají nedostatečnou znalost systému a mají přístup pouze k méně sofistikovanému vybavení. Často se raději pokoušejí zneužít existující nedostatky systému, než vytvořit nové.

Třída II (knowledgeable insiders): Oproti předchozí třídě mají speciální technické vybavení, analytické nástroje a dostatek zkušeností. Mají odlišné znalosti různých částí systémů, ke kterým většinou mají přístup.

Třída III (funded organisations): Jsou většinou schopné sestavit týmy specialistů se vzájemně se doplňujícími schopnostmi a velkou finanční podporou. Ti jsou pak schopni hluboké analýzy systému, navrhování důmyslných útoků a používání nejpokročilejších analytických nástrojů. Mohou využívat útočníků z třídy II.

Dále si popíšeme několik technik útoků, které v současné době patří k největším hrozbám fyzické bezpečnosti kryptografických modulů a které mohou být provedeny třídou útočníků II či dokonce I.

2.3.1 TEMPEST

Existuje celá škála útoků pracujících na principu odchytávání elektromagnetického záření z elektronických zařízení. Jedná se o velmi obecné útoky, jejichž možnosti nejsou zdaleka prozkoumány, a to ani přesto, že první z nich jsou známy již od počátku šedesátých let minulého století. Obecnost a obtížnost obrany je činí dodnes velmi nebezpečnými, a proto se jimi zabývají i mnohé vládní a vojenské organizace. Program TEMPEST byl prvním, který se touto problematikou začal zabývat, a důsledkem toho je, že většina souvisejících materiálů je utajována. V této části vycházíme především z [7], přičemž další informace je možné nalézt v [38, 47].

Postiženy mohou být téměř všechna elektronická zařízení. Podařilo se například odposlouchávat síťový provoz pomocí telefonního kabelu, který byl veden paralelně, celé 2 metry daleko od síťového kabelu [51]. Se znalostí frekvence, kterou vyzařuje kabel od klávesnice, šlo odposlechnout a určit stisknuté klávesy [44]. Podobným způsobem lze odposlouchávat i procesor (např. na čipové kartě) určený k provádění známého algoritmu. Z našeho pohledu se však jako nejnebezpečnější jeví možnost vzdálené rekonstrukce obrazu monitoru. Poznamenejme, že ačkoliv mnohé z monitorů splňují požadavky sníženého vyzařování (normy MPR a TCO), neznamená to, že jsou chráněny proti útokům založeným na TEMPESTu. Tyto normy specifikují pouze měření do 400kHz, zatímco vyzařování vztahující se k obsahu obrazovky se nachází vysoce nad 30MHz. Adekvátní ochranu nám neposkytují dokonce ani moderní

[12] Jedná se především o útoky, ke kterým ve FIPS 140-2 neexistují testovatelné požadavky bezpečnosti, a o ilustrující útoky znázorňující snadnou zranitelnost konkrétních bezpečnostních zařízení.

LCD displeje. Výhoda všech těchto útoků je, že mohou být prováděny nepozorovaně z velké vzdálenosti a obrana proti nim, která spočívá ve stínění počítače či celé budovy, je velmi nákladná.

V posledních letech bylo pozorováno [7], že lze poměrně snadno zachytit vyšší frekvence obrazového signálu, na něž není lidské oko natolik citlivé. Na základě toho vznikly TEMPEST fonty, které při svém zobrazení omezují vyzařování na těchto frekvencích a podstatně nám tak za přijatelnou cenu zvyšují bezpečnost. Vzhledem k tomu, že uživatel na první pohled nepozná rozdíl mezi různými fonty, může ale dojít i k vývoji speciálních virů, které nahradí běžné fonty za fonty usnadňující útok.

2.3.2 Útoky postranními kanály

Útoky postranními kanály využívají informací, které uniknou během provádění kryptografických operací (např. doba trvání těchto operací, množství spotřebované energie, nebo i výše zmíněné elektromagnetické záření). Typicky se jedná o velmi jednoduché a účinné metody, které často vedou až k odhalení zašifrovaných dat či tajných klíčů. Mezi nejvýznamnější patří časová analýza, výkonová analýza a chybová analýza. Všechny vznikly v průběhu posledních deseti let, a otevřely tak nový pohled jak na návrh kryptosystémů, tak na samotnou kryptoanalýzu. Podrobně se jimi zabývá například [40].

Časová analýza (timing analysis) – byla poprvé publikována v roce 1995, kdy Paul Kocher v [34] popsal, jak může u kryptosystémů jako RSA, DSS či Diffie-Hellman čas korelovat s hodnotami jejich tajných klíčů. Obrana proti časové analýze není náročná a je založena buď na použití operací, jejichž provádění zabere stejné množství času, nebo na přidání „šumu“, který naopak zajistí různé délky provádění těchto operací.

Chybová analýza (fault analysis) – byla poprvé představena v roce 1996 a je popsána v [13]. Vychází z předpokladu, že hardwarové chyby, které se mohou vyskytnout během výpočtu kryptografického modulu mohou také ovlivnit jeho bezpečnost. Útočník se tak pokouší cíleně či náhodně vkládat do průběhu výpočtu kryptografické operace chyby. Pokud se podaří změnit výsledek operace, je možné v některých případech provést úspěšnou kryptoanalýzu. Jako vhodné protiopatření se může jevit kontrola výstupu výpočtů, avšak ta je náročná a navíc, jak bylo ukázáno v [50], nemusí být dostačující.

Výkonová analýza (power analysis) – tato metoda, popsána v [35, 36], byla poprvé publikována v roce 1998 a znamenala další významný pokrok v kryptoanalýze. Spočívá ve využití informace o množství spotřebované energie během provádění daného kryptografického algoritmu. Dělí se na SPA (simple power analysis) a DPA (differential power analysis). SPA je založena na přímém vyhodnocování množství spotřebované energie, zatímco DPA navíc využívá statistických metod, čímž je pak na rozdíl od SPA schopna odhalit i nepatrné výkyvy spotřeby energie (a eliminovat šum a chyby vzniklé při měřeních). DPA je považována za účinnější a nebezpečnější útok než SPA, ale například pro mnoho současných čipových karet je SPA naprosto dostatečná. Protiopatření

proti výkonové analýze mohou být buď softwarová (např. speciálně navržené algoritmy či přidání náhodného maskování) nebo hardwarová (např. fyzické stínění či nepřímé napájení čipů uvnitř kontaktních čipových karet).

Implementace těchto útoků jasně dokazují, že hranice mezi logickou a fyzickou bezpečností se vzájemně prolínají a že již na kryptografické algoritmy nelze nahlížet jako na „černé skříňky“, na jejichž bezpečnost (byť i formálně dokázanou) se můžeme vždy a za všech okolností spolehnout. Jak ukážeme ve třetí kapitole, podobné tvrzení se vztahuje i na kryptografické protokoly, které jsou vybudovány pomocí funkcí API.

2.3.3 Útoky na zařízení odolná proti průniku

Jedná se o útoky na zařízení, která většinou splňují nějaké bezpečnostní požadavky (např. jednu z úrovní ohodnocenou podle FIPS 140-2 či CC), a měla by být proto fyzickým útokům dostatečně odolná. Ukážeme si metody útoků využívající například výše popsaných postranních kanálů, a seznámíme se i s útoky na konkrétní kryptografické moduly. V této části vycházíme z [5, 6].

Proveditelný útok chybovou analýzou

Ačkoliv již bylo popsáno a navrženo mnoho útoků využívajících chybové analýzy založené na změně obsahu paměti (a tím algoritmu nebo klíče), hlavní problém spočívá v jejich proveditelnosti (resp. v proveditelnosti jejich chybového modelu). V mnoha koprocesorech jsou totiž tajné klíče udržovány v EEPROM, společně s několika kilobajty spustitelného kódu, a pokus o vložení chyby tedy může způsobit spíše krach výpočtu či nějakou neinformativní chybu než vyprodukování chybného výstupu, potřebného pro tyto útoky. Popíšeme si tedy odlišný a mnohem realističtější chybový model, jehož pomocí pak lze mnohé útoky provádět snadněji a lépe.

Útok využívající tento chybový model byl poprvé použit v hackerské komunitě k narušení bezpečnosti čipových karet placených televizních stanic. Jeho hlavní myšlenka spočívá v aplikaci chyby, která je v tomto případě realizována jako krátkodobá změna dodávky energie či rychlosti hodinového taktu. Nejde tedy o vytvoření permanentní chyby, ale chyby provádění konkrétního výpočtu. Tento typ chyb ovlivní pouze některé signály a tranzistory, což může způsobit přeskočení či provedení jiných instrukcí (dokonce takových, které nepodporuje ani mikrokód). Zařízení ovšem zůstane nepoškozeno. Celý útok je závislý na přesném načasování a délce trvání indukované chyby. Zvažme například krátký cyklus, jehož cílem je vypsání části paměti na sériový port:

```
1 b = answer_address
2 a = answer_length
3 if (a == 0) goto 8
4 transmit(*b)
5 b = b + 1
6 a = a - 1
7 goto 3
8 ...
```

V takovém případě se snažíme systematicky najít takovou chybu, jejíž výsledek by neovlivnil čítač programu, ale změnil by podmíněný skok na řádku tři nebo čítač cyklu na řádku šest. Jestliže se nám to podaří, pak zkopírujeme celou paměť. Podobně mohou být cílem útoku systémy kontroly hesel, přístupových práv a vůbec všechny rutiny, kde odstranění jediné instrukce může narušit jejich správnou funkci. Jako rozumné softwarové protiopatření se jeví takovýmto instrukcím předcházet, zatímco hardwarová ochrana může být realizována například použitím vnitřních nezávislých generátorů hodinového taktu nebo asynchronními obvody (tj. bez hodin).

Ukázalo se, že útoky založené na vkládání chyb do instrukčního kódu jsou lehčí a účinnější než útoky založené na vkládání chyb do dat, a navíc, jsou-li aplikovány namísto řídícího kódu přímo na algoritmus, jsou velmi efektivní. V [6] je také popsáno použití tohoto chybového modelu k útoku na RSA, DES či k provedení zpětné analýzy[13] neznámé blokové šifry.

μABYSS

μABYSS je koprocesor vyvinutý firmou IBM, který je spolu s procesorem, pamětí, baterií a dalšími obvody umístěn v chromoniklovém drátěném obalu zalitý neprůhlednou epoxidovou pryskyřicí s příměsí křemíku. Tato ochrana se jevila jako dostačující a učinila μABYSS odolným proti většině pokusů o mechanické průniky. I přesto však byla nalezena slabá místa.

Návrháři předpokládali, že při detekci průniku bude dostatečnou ochranou citlivých informací jejich smazání odstraněním napájení pamětí. Ty si však náboj při pokojové teplotě ponechávají řádově několik sekund a při zchlazení (např. tekutým dusíkem nebo héliem) i několik minut či dokonce hodin, což již dává útočníkovi příležitost k odstranění fyzických překážek a přečtení všech informací z paměti. Navíc později bylo v [25] ukázáno, že paměti, které uchovávají delší dobu stejný bitový vzorek (např. tajný klíč), mají i po odstranění napájení tendenci zůstat ve stejném stavu a tento efekt může trvat až několik dní.

Dallas DS5002FP

Tento Intel 8051 kompatibilní procesor, vyvinutý firmou Dallas Semiconductor, se používá v mnoha platebních terminálech k provádění zabezpečených operací. Tajný klíč je uložen ve speciálním, bateriemi napájeném registru uvnitř čipu a jeho pomocí probíhá šifrování („za běhu“). Má-li se do koprocesoru založeného na DS5002FP nahrát nová aplikace, vymaže se nejprve bezpečnostní zámek, což má vždy za následek smazání tajného klíče. Poté se ustanoví nový tajný klíč, alokuje se potřebné množství paměti a provede se nahrání aplikace. Nakonec se opět nastaví původní bezpečnostní zámek. Takto nahraná aplikace je pomocí tajného klíče zašifrována a uložena ve vnější nechráněné paměti, odkud je spouštěna (t.j. musí být opět v reálném čase zpětně dešifrována). Protože se vždy šifrují jak přenášená data, tak také hodnoty adres na než mají být uložena, nejsou zašifrovaná data či instrukční kódy aplikací uloženy v paměti ve správném pořadí. Šifrování adres probíhá pouze pomocí tajného klíče, zatímco šifrování dat je navíc závislé i na hodnotě adresy. Zašifrovaná data

[13] Reverse engineering.

mohou tedy oproti adresám nabývat více různých hodnot. Tento a spousta dalších bezpečnostních mechanizmů, jako například náhodné přístupy k vnější sběrnici a paměti, jejichž cílem je zamaskování operací probíhajících uvnitř koprocesoru, jsou podrobněji popsány v [19].

Celkové fyzické zabezpečení procesoru patřilo podle výrobce k jednomu z nejdůmyslnějších, a přesto se podařilo provést jednoduchý a efektivní útok, jež vedl k odhalení všech citlivých informací. Myšlenka tohoto útoku spočívá v podstrčení zašifrovaných instrukcí procesoru, na základě jehož chování pak lze určit jejich skutečnou funkci. Pokusme se najít například tříbajtovou instrukci `mov 90h, #42h`, zakódovanou jako `75h 90h 42h`, která by měla po dvou přístupech na sběrnici vracet na paralelním portu na adrese `90h` hodnotu `42h`. Zkoušíme tedy systematicky všech 2^{16} kombinací prvních dvou bajtů, dokud není instrukce provedena, a tím i spolu s příslušnou hodnotou adresy portu rozpoznána. Následným prohledáváním všech 2^8 kombinací třetího bajtu instrukce a pozorováním adresy `90h`, lze pro adresu, z níž byl tento bajt načten, získat dešifrovací funkci datové sběrnice. Nyní celý proces zopakujeme a budeme hledat jednobajtovou instrukci `NOP`, následovanou stejnou instrukcí `mov` jako v předchozím případě. Časová složitost se nezvýší, protože již známe správnou zašifrovanou hodnotu třetího bajtu (tj. v tomto případě hodnotu adresy portu `90h`). Protože se nám však o jedničku zvýšila adresa, na níž má instrukce `mov` třetí bajt, můžeme pro tuto adresu, podobně jako výše, získat dešifrovací funkci datové sběrnice. Pomocí dostatečného počtu takto získaných a uložených hodnot zašifrovaných instrukcí, dat a adres již snadno můžeme procesoru poslat sekvenci vhodných instrukcí, které vypíší celou paměť.

V [6] nazývají tento útok *Cipher Instruction Search Attack* a je vidět, že snadnost jeho návrhu a realizace je způsobena pouze tím, že nikdo z návrhářů něco podobného prostě neočekával, a tedy nevzal ani v úvahu. Pro svou povahu může být již také považován za zvláštní druh útoku na protokoly, a tedy na logickou bezpečnost.

Kapitola 3

Popis útoků na a přes API

Cílem této části práce je podrobný rozbor útoků na a přes API a studie jejich praktické proveditelnosti. Všechny útoky z této kapitoly se řadí mezi útoky na logickou bezpečnost a jejich význam spočívá především v demonstraci nedostatků a chybného návrhu API současných kryptografických modulů. Ukazuje se, že evoluční vývoj těchto produktů a snaha o univerzálnost návrhu těchto zařízení, a tudíž i podpora mnoha standardů či norem, může vyústit v až příliš rozsáhlé API, jehož správnou funkčnost lze jen stěží zaručit. Než přistoupíme k analýze jednotlivých útoků, poznamenejme, že veškerý výzkum týkající se bezpečnosti API odstartovala práce [3].

3.1 Úvod do problematiky API pro HSM

Funkce API tvoří většinou jediné komunikační rozhraní mezi kryptografickým koprocesorem a vnější aplikací. Jejich pomocí je realizován přístup k veškerým operacím, které koprocesor provádí, a jsou tedy nezbytné jak pro správu samotného zařízení, tak také pro komunikaci s jinými zařízeními.

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, čímž poskytuje velmi velký prostor ke zneužití. Tuto skutečnost dokládají nedávno objevené útoky na API hardwarových bezpečnostních zařízení, jejichž podstata většinou spočívá v korektním zadávání příkazů koprocesoru, avšak v neočekávaném pořadí. Výsledkem těchto útoků je obejití zamýšlené bezpečnostní politiky zařízení.

V této práci se budeme zabývat výhradně kryptografickými API. Ty lze dále rozdělit na standardní a finanční kryptografická API[1]. 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

[1] V praxi jsou však většinou součástí pouze jednoho společného API.

podpora SET), které jsou vyžadovány bankovním sektorem. Mezi běžně komerčně používané API patří například The Common Cryptographic Architecture (CCA) od IBM, The HP(dříve Compaq)-Atalla API, Public Key Cryptography Standard (PKCS) #11 od RSA nebo The Thales-Zaxus-Racal API. Protože se tato práce do značné míry zabývá také bezpečností kryptografického koprocesoru IBM 4758 (viz příloha A), bude nejvíce odkazovaným právě IBM CCA API a v závěru kapitoly i PKCS #11. CCA (viz příloha B) je příkladem typického API, které kromě standardních funkcí a operací podporuje i finanční služby.

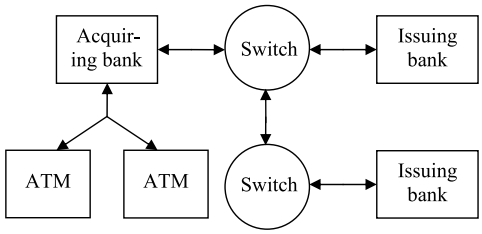
3.1.1 Finanční bezpečnost a kryptografická API

Protože význam většiny funkcí obsažených ve standardním kryptografickém API je zřejmý, zaměříme se nyní na finanční kryptografická API. V této části vycházíme především z [18] a naším cílem je vysvětlení významu některých funkcí obsažených ve finančním kryptografickém API. Nejprve popíšeme základní terminologii nezbytnou k porozumění architektury ATM či EFTPOS[2] sítí:

- Vydávající banka (issuing bank) – banka, kde má vlastník (uživatel, zákazník) svůj účet, a která mu vydala kartu a PIN.
- Poskytující banka (acquiring bank) – banka, která je počátečně zodpovědná za transakci uživatele (tj. provozuje například použitý bankomat).
- Osobní identifikační číslo (PIN) – číslo, které používá vlastník účtu k ověření jeho identity vůči vydávající bance.
- Číslo účtu (PAN) – číslo asociované s každým uživatelským účtem.
- Čistý PIN-blok (CPB) – PIN formátovaný[3] do 8bajtového bloku.
- Zašifrovaný PIN-blok (EPB) – zašifrovaný CPB.

Architektura ATM či EFTPOS sítí

Jednoduchá reprezentace ATM či EFTPOS sítí je zobrazena na obrázku 3.1. V levém dolním rohu jsou bankomaty či jiná zařízení (např. webové prohlížeče či mobilní telefony), které používá zákazník k provádění transakcí nebo ke komunikaci s bankou. Ne vždy však musí mít zákazník účet v poskytující bance, která provozuje použitý bankomat. V těchto případech musí banka přeposlat transakci bance vydávající, ve které má daný zákazník svůj účet. Pro komunikaci mezi bankami se používají „mezilehlé“ switche.



Obr. 3.1: Reprezentace EFT sítě.

Důvodem použití této architektury (resp. switchů) byly šifrovací metody použité k ochraně PINů. V době, kdy se začaly vyvíjet bankovní sítě, se používalo pouze

[2]Electronic Funds Transfer at Point of Sale (EFTPOS) je metoda elektronických plateb umožňující nákup zboží a služeb pomocí kreditní nebo debetní karty.

[3]Existuje mnoho formátů CPB, z nichž asi nejvýznamnější je ANSI X9.8.

symetrické kryptografie založené typicky na DES. Libovolné dvě entity, které spolu chtěly zabezpečeně komunikovat, si proto nejprve musely ustanovit sdílený tajný klíč (někdy též označovaný jako zónový klíč). Tento model vzájemné komunikace však obnášel tři zásadní problémy. Předně to byla komunikace s novými bankami, které ještě neměly ustanoven zónový klíč, dále proces ustanovování těchto klíčů, který byl drahý a zdlouhavý a nakonec samotná správa velkého množství sdílených klíčů, jež musely být bezpečně uloženy a chráněny. Aby se těmto problémům předešlo, byly použity mezilehlé switche. Komunikace bank pomocí switchů je popsána v následujícím odstavci.

Operace prováděné na PINech

Před provedením vlastní finanční transakce se musí uživatel vydávající bance autentizovat. Nejprve tedy zadá do komunikačního zařízení (např. ATM) PIN, který je ihned zašifrován. Šifrování probíhá pomocí klíče sdíleného mezi ATM a s poskytující bankou, které je poté EPB zaslán. Nyní musí banka poslat EPB switchi, s kterým však sdílí jiný zónový klíč. Musí jej tedy přešifrovat tj. pomocí prvního klíče dešifrovat a pomocí zónového klíče sdíleného se switchem opět zašifrovat. Celý tento proces se opakuje do doby než EPB dorazí k vydávající bance. Ta pak pouze rozhodne, je-li PIN (spojený s číslem účtu) správný.

Je tedy zřejmé, že použité finanční API musí přinejmenším obsahovat funkce určené k šifrování PINů, překladu PINů (tj. přešifrování jinými zónovými klíči či přeformátování PIN-bloku), generování PINů a verifikaci PINů. Šifrování PINů je jednoduchá operace, při níž je PIN formátován do 8bajtového CPB a poté zašifrován (typicky pomocí DES či 3DES). Poté může být EPB dále překládán či přeformátováván. Překlad PINů mezi zónovými klíči pouze dešifruje EPB s použitím vstupního klíče a zašifruje jej s použitím výstupního klíče. Přeformátování je pak jen rozšířením překladové funkce umožňující specifikovat vstupní a výstupní formátování PINů. Generování PINů je operace, při níž se na základě dat vztažených k účtu či osobě (a někdy označovaných jako validační data) vypočítá/vygeneruje PIN, který je pak vrácen jako EPB.

3.1.2 Kategorizace útoků na kryptografická API

Odlišnost a především rozmanitost jednotlivých útoků způsobuje, že jejich rozdělení je velmi obtížné. Existují útoky natolik obecné, že je lze použít na většinu námi zmiňovaných zařízení (resp. jejich API), jiné jsou naopak aplikovatelné jen na konkrétní API. Některé by se daly označit jako útoky na standardní kryptografická API, jiné jako útoky na finanční kryptografická API. Jejich cílem může být získávání tajných klíčů, PINů, ale i narušení integrity dat nebo obcházení řízení přístupu (např. neautorizované generování či změna typů klíčů).

Naše rozdělení se pokouší zohledňovat generaci dotčeného zařízení, ale především cíl útoku. Většina teoretických útoků je proto nezávisle na konkrétním API popsána v částech 3.2, 3.3 a 3.4. Cílem útoků v 3.2 a 3.3 je obcházení řízení přístupu nebo získání tajných klíčů (což samozřejmě vede i k získání PINů a ostatních citlivých informací). Cílem útoků v 3.4 je získání PINů bez znalosti jakéhokoliv klíče. Mož-

nostmi praktické aplikace útoků se pak zabývá část 3.5 a nakonec v části 3.6 jsou demonstrovány útoky na konkrétní API (PKCS #11).

3.2 Útoky na API starších kryptografických modulů

Tato část práce pojednává o útocích na API starších kryptografických modulů. Ty byly určeny převážně pro použití v bankovním sektoru a jejich hlavním cílem bylo chránit kryptografické klíče či jiné citlivé informace před personálem banky. Dále v této části vycházíme z [3, 4, 10].

3.2.1 Known Key Attack

Tento nedávno objevený útok lze aplikovat na mnohá bezpečnostní zařízení, která byla používána bankami ke správě bankomatů kolem roku 1980. Patří mezi ně také značně rozšířený Visa Security Module (VSM), jehož hlavní funkcí bylo chránit v bankovních ATM sítích přenášený PIN.

Protože se VISA snažila do své sítě začlenit co nejvíce bank, bylo potřeba zajistit, aby zaměstnanci kterékoliv banky neměli přístup k PINům zákazníků nejen své, ale i ostatních bank. Každý uzel v síti se tedy vybavil schváleným kryptografickým zařízením, které mělo přenášené PINy chránit. Klíče mezi těmito zařízeními a bankomaty se nastavovaly manuálně. Bezpečnostní zařízení umístěné v bance vygenerovalo příslušné části hlavního klíče, které se vytiskly na bezpečné tiskárně[4] připojené přímo k HSM a předaly bezpečnostním úředníkům. Ti je pak doručili a nainstalovali do příslušného bankomatu, kde z nich byl logickým součtem (operace $\oplus$) vytvořen hlavní terminální klíč (K_{MT}). Pomocí tohoto klíče byly později bankomatu důvěrnou cestou zasílány ostatní šifrovací klíče. Podobným způsobem se nastavovaly i klíče mezi jednotlivými bankami.

Mnoho bankovních kryptografických modulů (např. VSM) tedy obsahuje funkce k vygenerování a vytištění částí hlavního terminálního klíče (pro zjednodušení tyto funkce voláme z hostitelského zařízení bez parametrů). Kromě toho je hodnota části K_{MT} navrácena i volajícímu programu a zašifrována pomocí příslušného hlavního klíče (K_M) zařízení. Dále má VSM další funkci, která jednotlivé části klíče kombinuje a produkuje tak výsledný K_{MT} (tato funkce má dva parametry a to jednotlivé části klíče zašifrované pomocí hlavního klíče K_M). Podívejme se nyní na předpokládané volání těchto funkcí:

1. `Host -> VSM:` GenerateKeyPart();
   `VSM -> Printer:` K_{MT1}
   `VSM -> Host:` $E_{K_M}(K_{MT1})$

2. `Host -> VSM:` GenerateKeyPart();
   `VSM -> Printer:` K_{MT2}
   `VSM -> Host:` $E_{K_M}(K_{MT2})$

[4] Výstupem tisku je speciální uzavřená obálka, podobná těm, v nichž je zákazníkům bank doručován PIN.

3. `Host -> VSM:` CombineKeyParts($E_{K_M}(K_{MT1}), E_{K_M}(K_{MT2})$);
   `VSM -> Host:` $E_{K_M}(K_{MT1} \oplus K_{MT2}) = E_{K_M}(K_{MT})$

Jak je vidět, po provedení těchto funkcí je $K_{MT} = K_{MT1} \oplus K_{MT1}$. Selhání protokolu spočívá v tom, že nečestný programátor může vzít libovolný zašifrovaný klíč (či jeho část) a podstrčit jej dvakrát funkci, která části klíče kombinuje.

1. `Host -> VSM:` CombineKeyParts($E_{K_M}(K_{MT1}), E_{K_M}(K_{MT1})$);
   `VSM -> Host:` $E_{K_M}(K_{MT1} \oplus K_{MT1}) = E_{K_M}(0)$

Volá-li tyto funkce výše uvedeným způsobem například hostitelský program v bankomatu, zná pak útočník (např. nečestný bankovní programátor) hodnotu příslušného terminálního klíče.

Nyní stačí pouze použít funkci, která zašifruje *PIN generující klíč* (K_{PG}) pomocí klíče K_{MT}. PIN generující klíč a PAN používá bankomat k verifikaci zákazníkova PINu v případě výpadku sítě. Tento klíč si pak útočník může známým K_{MT} dešifrovat a s jeho pomocí je schopen vypočítat libovolný zákazníkův PIN (zjednodušeně $PIN = E_{K_{PG}}(PAN)$).

3.2.2 A 'two-time type' Attack

Podobně jako jiná hardwarová bezpečnostní zařízení používá i VSM rozlišování jednotlivých typů klíčů. Toho bývá dosaženo jejich šifrováním odlišnými hlavními klíči. Klíče stejného typu jsou šifrovány vždy stejným hlavním klíčem. VSM má hlavních klíčů (a tedy i typů) devět (např. starší IBM 3848 má pouze tři) a jak ukážeme, ani to není dostačující.

Jak již bylo zmíněno u předchozího útoku, jedno z použití K_{MT} je, aby chránil přenos ostatních šifrovacích klíčů. Ty jsou někdy označovány jako komunikační klíče (K_C) a používají se k zajištění důvěrnosti a integrity zpráv jdoucích z/do bankomatu. Pro používání K_C k šifrování či dešifrování nejsou žádná omezení. Navíc existuje funkce umožňující vložit do systému čistý K_C, který je pak zašifrován odpovídajícím hlavním klíčem – označme jej K_{MC} (tato funkce má jediný parametr a tím je komunikační klíč K_C určený k zašifrování klíčem K_{MC}). Kromě toho je v systému obsažena funkce, která umožňuje dešifrovat $E_{K_{MC}}(K_C)$ a přešifrovat jej pod jiným K_{MT} (tato funkce má dva parametry – prvním je příslušným hlavním klíčem K_{MC} zašifrovaný komunikační klíč K_C a druhým pak hlavním klíčem K_M zašifrovaný nějaký terminální klíč K_{MT}). Podívejme se nyní na předpokládané volání těchto funkcí:

1. `Host -> VSM:` InsertKey(K_C);
   `VSM -> Host:` $E_{K_{MC}}(K_C)$

2. `Host -> VSM:` ReEncrypt($E_{K_{MC}}(K_C), E_{K_M}(K_{MT})$);
   `VSM -> Host:` $E_{K_{MT}}(K_C)$

Problémem je, že klíče K_{MT} a K_{PG} (PIN generující klíče) mají stejný typ (tj. jsou šifrovány stejným hlavním klíčem K_M), což umožňuje použít v libovolné funkci $E_{K_M}(K_{PG})$ namísto $E_{K_M}(K_{MT})$. Útočníkovi nyní jen stačí namísto K_C vložit do

první funkce PAN a v druhé funkci namísto $E_{K_M}(K_{MT})$ podstrčit $E_{K_M}(K_{PG})$. Volání těchto funkcí pak bude vypadat následovně:

1. `Host -> VSM:` InsertKey(PAN);
   `VSM -> Host:` $E_{K_{MC}}(PAN)$
2. `Host -> VSM:` ReEncrypt($E_{K_{MC}}(PAN), E_{K_M}(K_{PG})$);
   `VSM -> Host:` $E_{K_{PG}}(PAN) = PIN$

Jak je vidět, útočník může touto cestou získat libovolný PIN. Tento útok je důsledkem nejen příliš složité manipulace s klíči, ale také přirozeného předpokladu, že jakákoliv zašifrovaná data již nejsou z hlediska informační bezpečnosti citlivá. To však v případě použití K_{PG} není pravda a výsledek šifrování (zákazníkův PIN) je vždy citlivou informací.

3.2.3 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). Nalezení jednoho klíče pak obvykle umožní kompromitování i všech ostatních klíčů stejného typu, což statisticky vede k přirozenému omezení prohledávaného prostoru klíčů. Mnoho kryptografických koprocesorů dokáže vygenerovat 2^{16} DES klíčů daného typu řádově během minut a jejich pomocí pak lze redukovat nalezení jednoho klíče z původních 2^{55} na 2^{39}. Prohledání takto redukovaného prostoru klíčů zabere i na domácím PC pouze několik dnů.

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 například typu terminální (K_{MT}), lze jím přešifrovat veškerá jinými terminálními klíči chráněná data, která se pak známým klíčem snadno dešifrují. Tímto způsobem je možné kompromitovat osm z devíti typů klíčů, které používá VSM. Podrobně je tento útok popsán v [10].

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 klíč ustanovuje manuálně z jednotlivých částí, které jsou v modulu v příslušném registru XORovány. Po vložení části klíče je vždy vrácen celým obsahem registru zašifrovaný kontrolní vektor zajišťující korektní nahrání klíče. Chybou v API tohoto zařízení je, že jakýkoliv uživatel může pokračovat v přidávání částí hlavního klíče, a tím i v získávání dalších variant zašifrovaného kontrolního vektoru. Tímto způsobem se vytvoří 2^{16} variant hlavních klíčů společně s množinou zašifrovaných kontrolních vektorů a dále se postupuje analogicky jako v předchozím případě.

3.2.4 Conjuring Keys From Nowhere

Kouzlení klíčů je považováno za útok[5] umožňující neautorizované generování klíčů ukládaných mimo kryptografický koprocesor. Ty pak mohou být využity k dalším druhům útoků na API. 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 pravděpodobností $1/2^8$ správnou paritu[6]. 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), které používají formáty klíčů bez jakéhokoliv doplnění. Obrana spočívá v důmyslnějším návrhu formátu klíčů (např. před jeho zašifrováním přidat kontrolní součet a časové razítko).

3.3 Útoky na API současných kryptografických modulů

Tato část práce pojednává o útocích na API současných kryptografických modulů, které se dnes běžně používají. Útoky se týkají především koprocesorů využívajících IBM CCA API (viz příloha B) a demonstrují techniky, jimiž lze obejít bezpečnostní politiku HSM. K mnohým útokům lze použít i některé metody aplikovatelné na starší kryptografické moduly. To je důsledkem toho, že všechny útoky obsažené v celé této kapitole byly objeveny až po roce 2000, kdy se na univerzitě v Cambridge začali problematikou bezpečnosti API zabývat.

Podrobné informace týkající se nutné konfigurace zařízení a použitých funkcí CCA API jsou obsaženy v příloze C. Připomeňme ještě, že veškeré klíče opouštějící koprocesor jsou vždy zašifrovány nějakým transportním klíčem (viz B.3.3), což pro zjednodušení popisu útoku není vždy uvedeno.

3.3.1 A Chosen Key Difference Attack on Control Vectors

Následující útok je také někdy označován jako *Key Import Attack* a je detailně popsán v [4, 9, 10]. Je to jeden z nejjednodušších útoků na CCA API a jeho výsledkem je neautorizovaná změna kontrolního vektoru[7] (CV) při importu klíčů.

Modifikace transportního klíče

Transportní resp. klíče-šifrující klíče (K_{EK}) bývají ve finančních systémech mezi jednotlivými koprocesory přenášeny po částech, které nejsou nijak šifrovány. Klíč se zpětně vytvoří logickým součtem několika takových částí. Chybou CCA je, že vlastník kterékoliv části klíče ji může libovolně modifikovat a změnit tak nepozorovaně hodnotu výsledného klíče.

Označme si původní transportní klíč K_{ORIG}, jeho tři části KP_A, KP_B, KP_C, modifikovaný klíč K_{MOD} a vnášenou modifikaci kontrolního vektoru CV_{MOD}. Aby nebylo při modifikaci transportního klíče vzbuzeno podezření, je potřeba, aby byl K_{ORIG} řádně nahrán a až později držitel poslední části klíče vložil znovu svou vhodně

[5] I přesto, že některé starší moduly používaly tuto techniku k regulérnímu generování klíčů.

[6] DES používá lichou paritu a jako paritní je považován nejméně významný bit každého oktetu.

[7] V této části práce nebudeme rozlišovat mezi pojmy „kontrolní vektor“ a „typ klíče“.

modifikovanou část ($\mathrm{KP_C} \oplus \mathrm{CV_{MOD}}$). To je také jedna z mála praktických možností, jak v systému vytvořit zároveň originální i modifikovanou verzi klíče a zajistit tak, aby byl systém stále funkční. Druhou možností je, že držitel poslední části klíče vloží nejprve modifikovanou část klíče a vytvoří tak modifikovaný klíč. Poté jej zkopíruje a ostatním vlastníkům částí klíče bude tvrdit, že se při vkládání přepsal. Zopakováním celého procesu se pak vytvoří již originální klíč a systém bude opět funkční.

$$\mathrm{K_{ORIG}} = \mathrm{KP_A} \oplus \mathrm{KP_B} \oplus \mathrm{KP_C}$$

$$\mathrm{K_{MOD}} = \mathrm{K_{ORIG}} \oplus \mathrm{CV_{MOD}} = \mathrm{KP_A} \oplus \mathrm{KP_B} \oplus (\mathrm{KP_C} \oplus \mathrm{CV_{MOD}})$$

Změna typu importovaného klíče

Uvažme nyní bezpečný bankovní systém využívající IBM 4758 s nainstalovaným CCA. Jako jediná šifrovací funkce je použit algoritmus 3DES. Datové klíče se využívají k šifrování běžného bankovního provozu, a jsou tedy dostupné v mnoha uživatelských rolích. PIN generující klíče slouží k výpočtu PINů z veřejně dostupného čísla účtu, a jejich použití je tedy omezeno jen na několik málo autorizovaných rolí. Typickým příkladem útoku je importování PIN generujícího klíče ($\mathrm{K_{PG}}$) jako datového klíče, čímž se umožní uživatelům neautorizovaných rolí provádět výpočty PINů. Nechť je tedy do koprocesoru přenášen $\mathrm{K_{PG}}$ typu $\mathrm{CV_{PINKEY}}$, chráněný pomocí klíče $\mathrm{K_{ORIG}}$. Správně fungující proces importu pak vypadá následovně.

`Klíč vysílajícího modulu:` $\mathrm{K_{EK1}} = \mathrm{K_{ORIG}}$
`Klíč přijímajícího modulu:` $\mathrm{K_{EK2}} = \mathrm{K_{ORIG}}$
`Přijatý klíč a jeho typ:` $\mathrm{E_{K_{EK1} \oplus CV_{PINKEY}}(K_{PG})}, \mathrm{CV_{PINKEY}}$
`Proces importu:` $\mathrm{D_{K_{EK2} \oplus CV_{PINKEY}}(E_{K_{EK1} \oplus CV_{PINKEY}}(K_{PG})) = K_{PG}}$
`Zašifrování hlavním klíčem:` $\mathrm{E_{K_M \oplus CV_{PINKEY}}(K_{PG})}$

Útočník však může zneužít modifikovaného transportního klíče, a změnit tak typ $\mathrm{CV_{PINKEY}}$ na $\mathrm{CV_{DATAKEY}}$. Stačí pouze importovat $\mathrm{K_{PG}}$ pomocí $\mathrm{K_{MOD}}$ (v našem případě $\mathrm{K_{EK2}}$), jehož $\mathrm{CV_{MOD}} = \mathrm{CV_{PINKEY}} \oplus \mathrm{CV_{DATAKEY}}$. Celý proces vypadá následovně.

`Klíč vysílajícího modulu:` $\mathrm{K_{EK1}} = \mathrm{K_{ORIG}}$
`Klíč přijímajícího modulu:` $\mathrm{K_{EK2}} = \mathrm{K_{ORIG}} \oplus (\mathrm{CV_{PINKEY}} \oplus \mathrm{CV_{DATAKEY}})$
`Přijatý klíč a jeho typ:` $\mathrm{E_{K_{EK1} \oplus CV_{PINKEY}}(K_{PG})}, \mathrm{CV_{PINKEY}}$
`Proces importu:` $\mathrm{D_{K_{EK2} \oplus CV_{DATAKEY}}(E_{K_{EK1} \oplus CV_{PINKEY}}(K_{PG})) = K_{PG}}$
`Zašifrování hlavním klíčem:` $\mathrm{E_{K_M \oplus CV_{DATAKEY}}(K_{PG})}$

Poznamenejme, že praktická realizace tohoto útoku není tak přímočará a může vyžadovat obcházení četných bezpečnostních opatření bank. Útok je navíc značně závislý na aktuální konfiguraci celého zařízení a optimální nastavení řízení přístupu či řádné testování integrity vkládaných klíčů mu zcela zabrání (viz C.2.2). Asi nejjednodušším protiopatřením je však zcela přestat používat distribuci klíčů po částech a nahradit ji technikami založenými na používání asymetrické kryptografie.

3.3.2 Import/Export Loop Attack

Tento v [10] popsaný útok je rozšířením předcházejícího útoku na CCA API. Jeho cílem je opět neautorizovaná změna kontrolního vektoru, ale již se neomezuje pouze na import klíčů poslaných z jiných koprocesorů. Demonstruje možnost změny typu klíčů během jejich exportu a následného importu.

K provedení tohoto útoku je třeba mít v koprocesoru k dispozici stejný importní a exportní klíč lišící se o přesně danou hodnotu typu (tj. $\mathrm{CV}_{\mathrm{OLDTYPE}} \oplus \mathrm{CV}_{\mathrm{NEWTYPE}}$). Uvažme následující situaci. Útočník po částech importuje do koprocesoru klíče IMPORTER1 a IMPORTER2, lišící se o rozdíl typů $\mathrm{CV}_{\mathrm{IMPORTER}} \oplus \mathrm{CV}_{\mathrm{EXPORTER}}$. Brání-li bezpečnostní politika vložení první části klíče, lze ji vždy získat *kouzlením*.

`IMPORTER1:` $\mathrm{K}_{\mathrm{EK1}} = \mathrm{K}_{\mathrm{RANDOM}}$
`IMPORTER2:` $\mathrm{K}_{\mathrm{EK2}} = \mathrm{K}_{\mathrm{RANDOM}} \oplus (\mathrm{CV}_{\mathrm{IMPORTER}} \oplus \mathrm{CV}_{\mathrm{EXPORTER}})$

Dále, opět metodou *kouzlení* klíčů, vytvoří část klíče KP typu $\mathrm{CV}_{\mathrm{IMPORTER}}$. Toho lze docílit například neustálým zkoušením importu této zašifrované[8] části pomocí klíče IMPORTER1. Má-li útočník KP vytvořen, importuje jej použitím klíče IMPORTER2, čímž se jeho typ neautorizovaně změní na $\mathrm{CV}_{\mathrm{EXPORTER}}$.

`Kouzlením získaná část klíče:` $\mathrm{E}_{\mathrm{K}_{\mathrm{EK1}} \oplus \mathrm{CV}_{\mathrm{IMPORTER}}}(\mathrm{KP}), \mathrm{CV}_{\mathrm{IMPORTER}}$
`Proces importu:` $\mathrm{D}_{\mathrm{K}_{\mathrm{EK2}} \oplus \mathrm{CV}_{\mathrm{EXPORTER}}}(\mathrm{E}_{\mathrm{K}_{\mathrm{EK1}} \oplus \mathrm{CV}_{\mathrm{IMPORTER}}}(\mathrm{KP})) = \mathrm{KP}$

V tuto chvíli má tedy v koprocesoru k dispozici klíč KP, který je typu $\mathrm{CV}_{\mathrm{EXPORTER}}$ a klíč KP, který je typu $\mathrm{CV}_{\mathrm{IMPORTER}}$. Nyní již stačí pomocí funkce určené k importu částí nahrát ke klíči typu $\mathrm{CV}_{\mathrm{IMPORTER}}$ požadovaný rozdíl mezi typy. Těchto modifikovaných importních klíčů lze samozřejmě vytvořit i několik.

`Požadovaný exportní klíč:` $\mathrm{K}_{\mathrm{E}} = \mathrm{KP}$
`Požadovaný importní klíč:` $\mathrm{K}_{\mathrm{I}} = \mathrm{KP} \oplus (\mathrm{CV}_{\mathrm{OLDTYPE}} \oplus \mathrm{CV}_{\mathrm{NEWTYPE}})$

Pomocí exportního klíče pak může útočník z koprocesoru exportovat libovolný klíč s příznakem umožňujícím export. Ten následně pomocí vhodného klíče importuje, čímž dojde k neautorizované změně jeho typu. Příklad demonstrující import PIN generujícího klíče jako datového klíče (tj. $\mathrm{CV}_{\mathrm{OLDTYPE}} = \mathrm{CV}_{\mathrm{PINKEY}}$ a $\mathrm{CV}_{\mathrm{NEWTYPE}} = \mathrm{CV}_{\mathrm{DATAKEY}}$) je uveden níže.

`Požadovaný exportní klíč:` $\mathrm{K}_{\mathrm{E}} = \mathrm{KP}$
`Požadovaný importní klíč:` $\mathrm{K}_{\mathrm{I}} = \mathrm{KP} \oplus (\mathrm{CV}_{\mathrm{PINKEY}} \oplus \mathrm{CV}_{\mathrm{DATAKEY}})$
`Proces exportu:` $\mathrm{E}_{\mathrm{K}_{\mathrm{E}} \oplus \mathrm{CV}_{\mathrm{PINKEY}}}(\mathrm{K}_{\mathrm{PG}}), \mathrm{CV}_{\mathrm{PINKEY}}$
`Proces importu:` $\mathrm{D}_{\mathrm{K}_{\mathrm{I}} \oplus \mathrm{CV}_{\mathrm{DATAKEY}}}(\mathrm{E}_{\mathrm{K}_{\mathrm{E}} \oplus \mathrm{CV}_{\mathrm{PINKEY}}}(\mathrm{K}_{\mathrm{PG}})) = \mathrm{K}_{\mathrm{PG}}$
`Zašifrování hlavním klíčem:` $\mathrm{E}_{\mathrm{K}_{\mathrm{M}} \oplus \mathrm{CV}_{\mathrm{DATAKEY}}}(\mathrm{K}_{\mathrm{PG}})$

Bezpečnostní protiopatření jsou stejná jako u předchozího útoku.

[8] Uvědomme si, že podobně jako u předchozího útoku neznáme klíče K_{PG} a $\mathrm{K}_{\mathrm{ORIG}}$, tak ani u tohoto útoku neznáme klíče $\mathrm{K}_{\mathrm{RANDOM}}$ a KP, a tedy ani klíče vzniklé jejich rozšířením o další části. Tyto klíče jsou typicky uloženy mimo koprocesor a zašifrovány nějakým transportním klíčem.

3.3.3 3DES Key Binding Attack

Tento útok popsaný v [4, 10] je důsledkem nedostatečné vazby jednotlivých částí 3DES klíčů a jeho výsledkem je kompromitování většiny koprocesorem chráněných klíčů. CCA sice rozlišuje[9] mezi levou a pravou částí klíče, ale již nespecifikuje příslušnost ke konkrétnímu klíči. Tím je umožněna neoprávněná manipulace s jeho jednotlivými polovinami. Útočníkovi pak stačí vygenerovat velké množství klíčů se stejnými polovinami 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[10]. Byl-li požadovaným typem exportní klíč, můžeme nyní jejich pomocí exportovat a poté dešifrovat všechny klíče 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 nemá útočník povoleno vytvářet klíče se stejnými polovinami, musí vygenerovat standardní DES klíč a změnit jeho kontrolní vektor (viz B.3.3) na LEFT HALF OF A 3DES KEY. K této neautorizované změně typu muže využít jednoho z předcházejících útoků.

3.4 Útoky na API vedoucí k získání PINů

Samostatnou třídu útoků na současné kryptografické moduly tvoří útoky vedoucí k získání PINů, někdy označované jako *PIN Recovery Attacks*. 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 zašifrovaný PIN-blok (EPB). Kromě toho, že jsou extrémně rychlé a snadno implementovatelné, lze je většinou aplikovat i na více druhů běžně používaných API[11], čímž postihnou mnohem více hardwarových bezpečnostních zařízení. Dále v této části vycházíme především z [11, 12, 16, 18].

3.4.1 Decimalisation Table Attacks

Následující útoky zneužívají špatného návrhu a implementace funkcí používaných při generování a verifikaci PINů. Typická verifikační funkce na základě validačních dat vygeneruje zadanou metodou PIN a porovná jej s PINem získaným z EPB[12]. Bezpečnostním problémem těchto funkcí jsou právě metody generování PINů vycházející ze starých metod používaných bankomatem IBM 3624. Ty většinou k převodu čísel používají funkce s decimalizační tabulkou jako jedním z parametrů, a tu mnohá

[9] Pro zjednodušení uvažujme pouze dvouklíčový 3DES.

[10] Z důvodů vnitřního omezení pro používání klíčů.

[11] Například IBM CCA API, HP-Atalla API a Thales-Zaxus-Racal API.

[12] Kdyby byl místo identifikátoru metody generování PINů a validačních dat parametrem funkce čistý PIN, mohl by bankovní programátor postupným prohledáváním všech kombinací PINů snadno určit hodnotu PINu v EPB.

API nepovažují za citlivý vstup. V této části ukážeme, jak lze vhodnou manipulací s tabulkou (případně s pomocí několika známých zašifrovaných PINů) z EPB extrahovat hodnotu PINu. Tento druh útoků lze efektivně aplikovat například na CCA API (viz C.2.5) či na VSM.

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 generování/verifikaci PINů zvolených zákazníkem. Generování zde probíhá stejně jako v předchozím případě, ale výsledek se nazývá IPIN (intermediate PIN) či A-PIN a offset je získán odečtením IPINu od zvoleného PINu. Při verifikaci se pak zvolený PIN získá z IPINu přičtením offsetu. 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 této hodnotě je následně přičten offset, čímž se získá původní zákazníkův PIN. Ten se nakonec porovná s PINem, který byl bankomatu manuálně zadán.

```
                    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
```

Tyto metody byly použity v nejstarších typech bankomatů, a jsou tedy dosti rozšířeny a implementovány i v nových HSM. Jejich hlavním cílem bylo umožnit „offline“ bankomatům verifikovat zákazníkův PIN. Validační data byla spolu 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 a podporuje je například i IBM CCA API (viz B.5). Jedinou změnou je, že 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. První varianta útoku umožňuje získat číslice obsažené v zákazníkově PINu. Ten je z důvodů verifikace uvnitř funkce vygenerován, avšak jeho hodnota může být změnou decimalizační tabulky ovlivněna. Nastavíme-li například decimalizační tabulku na samé nuly, bude bez použití offsetu vždy po decimalizaci každý vygenerovaný PIN roven čtyřem nulám. Stejným trikem můžeme pomocí funkce generující zašifrované PINy získat EPB obsahující stejný nulový PIN. Když pak použijeme tento EPB a stejnou decimalizační tabulku jako parametry verifikační funkce, proběhne verifikace úspěšně (tj. z EPB dešifrovaný PIN je roven vygenerovanému PINu).

Nechť D_{orig} je původní decimalizační tabulka a D_i je nová binární decimalizační tabulka. Pak D_i má jedničku právě na těch pozicích, kde D_{orig} měla i. Je-li například $D_{orig} = \texttt{0123 4567 8901 2345}$, pak $D_5 = \texttt{0000 0100 0000 0001}$ a $D_9 = \texttt{0000 0000 0100 0000}$. Nyní stačí, aby útočník pro každou číslici i podstrčil verifikační funkci zašifrovaný nulový PIN a decimalizační tabulku D_i. Není-li v zákazníkově 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 nejvýše[13] 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 banky[14].

Celý proces určení originálního PINu p je pak reprezentován procházením binárního vyhledávacího stromu, jehož uzly v obsahují nějakou decimalizační tabulku D_v a jeden z pěti známých PINů p_v. Stromem se vždy prochází od začátku a rozhodnutí, zda z uzlu v jít doleva či doprava, závisí na výsledku volání verifikační funkce s parametry D_v a p_v. S každým uzlem v také může být asociován seznam originálních PINů P_v, přičemž $p \in P_v$ právě tehdy, když se do uzlu v dostane výše popsaným průchodem stromu. Kořenový uzel pak obsahuje seznam všech PINů a listy tvoří vždy jediný originální PIN. K efektivnímu vyhledávání je však ještě potřeba, aby byl strom co nejvíce vyvážený. Při jeho budování tedy volíme pro každý uzel v hodnoty D_v a p_v tak, aby pravděpodobnost úspěšné verifikace byla pro nějaké $p \in P_v$ co nejblíže 1/2. Tím je zaručeno, že oba podstromy budou přibližně stejně velké a celý strom bude dostatečně vyvážený.

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

[14] To je zároveň nejjistější 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).

Útok bez známého zašifrovaného PINu

V této části demonstrujeme, jak provést útok i bez jakékoliv známé zašifrované hodnoty PINu. 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 je nová decimalizační tabulka. Pak D_i má hodnotu $i-1$ právě na těch pozicích, kde D_{orig} měla 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 podstrčil verifikační funkci zachycený EPB, správný offset (tj. `0000`) a decimalizační tabulku 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 zákazníkův 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 [12].

3.4.2 ANSI X9.8 Attacks

Následující útoky zneužívají špatného návrhu a implementace funkcí používaných při verifikaci (viz C.1) a překladu PINů. Jejich bezpečnostní problémy souvisí především s formátováním PINu před jeho zašifrováním (PIN-blok), jehož vhodnou manipulací pomocí validačních dat (např. PANů) a překladové funkce může útočník z EPB získat hodnotu PINu.

Formáty PIN-bloků

Protože po zašifrování může samotný čtyřmístný PIN nabývat pouze 10 000 hodnot, hrozí nebezpečí útoků označovaných jako *Code Book Attacks*. Ty zneužívají malého počtu všech zašifrovaných a dešifrovaných PINů k snadnému vytvoření jednoznačného seznamu jejich dvojic – tzv. *kódové knihy*. Dešifrování je pak jen hledáním v tabulce. Z toho důvodu je vždy před zašifrováním PIN formátován do 8bajtové struktury zvané PIN-blok, kde jsou k němu většinou přidány náhodné doplňující hodnoty, které těmto útokům v ideálním případě zcela zamezí[15]. Jako příklad uveďme formáty:

[15] Formátování se samozřejmě provádí i pro PINy tvořené více než čtyřmi číslicemi.

- IBM 3624;
- ISO-0 (stejný jako ANSI X9.8, VISA-1 a ECI-1);
- ISO-1 (stejný jako ECI-4);
- ISO-2;
- VISA-2, VISA-3, VISA-4.

Některé formáty však právě z důvodů proměnných délek PINů nepoužívají doplnění náhodnými hodnotami a snaží se vyřešit zvýšení entropie v PIN-bloku jinými způsoby.

Zaměřme se nyní na formáty VISA-3 a ANSI X9.8, které jsou nezbytné k aplikaci několika dalších útoků. Oba jsou určeny pro PINy délky 4–12 číslic, přičemž delší PIN může být z pravé strany zkrácen. U formátu VISA-3 začíná PIN vlevo a končí oddělovačem, za nimž následují doplňující číslice. Ty mají v rámci jednoho PIN-bloku vždy stejné hodnoty a ztíží[16] útočníkovi případné budování kódové knihy. Popis VISA-3 pro čtyřmístný PIN je uveden níže; použité symboly reprezentují 4bitové hexadecimální číslice.

```
P je číslice nabývající hodnot '0'-'9' a udává PIN.
F je číslice hodnoty 'F' a slouží jako oddělovač.
X je doplňující číslice stejné hodnoty.
VISA-3 Clear PIN Block (CPB) = PPPPFXXXXXXXXXX.
```

U ANSI X9.8 je nejprve PIN formátován do bloku P1, PAN do bloku P2 a výsledný CPB vznikne následně jejich XORováním. Použitím PANu se předejde[17] budování kódové knihy a jeho svázání s PINem poskytne dostatečnou ochranu i proti postupnému zkoušení falešných PANů. Obecný popis formátu ANSI X9.8 je uveden níže.

```
Z je číslice hodnoty '0'.
L je číslice nabývající hodnot '4'-'C' a udává délku PINu.
f je hodnota, která je v závislosti na délce PINu buď P nebo F.
A je číslice nabývající hodnot '0'-'9' a udává PAN.
P1 = ZLPPPPfffffffFF.
P2 = ZZZZAAAAAAAAAAA.
ANSI X9.8 Clear PIN Block (CPB) = P1 xor P2.
```

Útok proti funkcím vyžadujícím PAN

Tento útok lze aplikovat na všechny překladové a verifikační funkce, které k extrahování PINu z CPB využívají PAN. Ten je vyžadován především těmi funkcemi, které podporují formátování PINu podle ANSI X9.8. Významné vstupní parametry volaných funkcí jsou:

[16] Počet položek kódové knihy se z 10^4 zvýší pouze na 10^5.

[17] V tomto případě zvýšení entropie plně závisí na číslu účtu, pokud ho jsme schopni získat, tak se entropie nezvýší.

- Tajný klíč, kterým byl PIN-blok zašifrován (K)
- Zašifrovaný PIN-blok (EPB)
- Číslo účtu (PAN)

Základní myšlenka útoku pak spočívá ve sledování změn způsobených postupnými modifikacemi PANu. Předpokládejme, že se útočníkovi podařilo zachytit ANSI X9.8 EPB obsahující čtyřmístný PIN. Při správném volání některé z těchto funkcí je po dešifrování klíčem K tento PIN extrahován z P1 = CPB $\oplus$ P2 = `04PPPPFFFFFFFFFF` jako `PPPP`. Během tohoto procesu je navíc proveden test, ověřující, zdali všechny jeho číslice nabývají hodnot '`0`'–'`9`'. Pokud tomu tak není, končí volání dané funkce s chybou. Podívejme se nyní, co se stane, zavolá-li útočník tutéž funkci s modifikovanou hodnotou první číslice PANu. Protože nyní P2' = P2$\oplus$`0000x00000000000`, je teď PIN z $P1' = CPB \oplus P2' = CPB \oplus P2 \oplus$ `0000x00000000000` = `04PPPPFFFFFFFFFF` $\oplus$ `0000x00000000000` extrahován jako `PPPP`$\oplus$`00x0`. V závislosti na zvolené hodnotě `x` pak, pokud `P` $\oplus$ `x` $<$ `10`, funkce proběhne vpořádku, jinak skončí s chybou. Tohoto testu lze využít k vytvoření posloupnosti úspěšných a neúspěšných volání funkce, která umožní částečně identifikovat `P` jako číslici z množiny[18] {`P`, `P`$\oplus$`1`}. Protože však první čtyři číslice bloku P2 jsou vždy nuly, není možné tímto způsobem blíže identifikovat hodnoty prvních dvou číslic PINu. Útok sníží množství možných kombinací PINu z 10^4 na 400.

Útok proti funkcím překládajícím PINy

Tento útok je sice rozšířením útoku předcházejícího, ale lze jej již aplikovat pouze na překladové funkce. Ty umožňují kromě PANů modifikovat i formát vstupního či výstupního PIN-bloku, čímž dávají útočníkovi mnohem větší prostor ke zneužití. Pozorujme například, co se stane při přeformátování zachyceného ANSI X9.8 EPB, je-li vstupní formátování specifikováno jako VISA-3 a výstupní jako ANSI X9.8. Pro jednoduchost nechť je PAN použitý v EPB roven nulám, čehož lze přeformátováním vždy dosáhnout. Stejně tak nechť je i výstupní PAN roven nulám. Po dešifrování je tedy CPB = `04PPPPFFFFFFFFFF` $\oplus$ `0000000000000000` = `04PPPPFFFFFFFFFF`, ale PIN je extrahován podle pravidel VISA-3 jako `04PPPP`. Poté je formátován do nového ANSI X9.8 PIN-bloku jako CPB = `0604PPPPFFFFFFFF`$\oplus$`0000000000000000` = `0604PPPPFFFFFFFF` a znova zašifrován. Tímto se uvnitř EPB rozšířil původní čtyřmístný PIN tvaru `PPPP` na šestimístný PIN tvaru `04PPPP`. Aplikací předchozího útoku nyní již útočník může částečně identifikovat všechny číslice původního PINu a prostor prohledávaných PINů tak snížit z 10^4 na 16.

K jejich jednoznačnému určení je však nezbytné zároveň s výše uvedeným přeformátováním modifikovat i PAN použitý v zachyceném EPB. To při vstupním formátování VISA-3 není možné (a verifikační funkce tento vstup ignoruje). Požadované modifikace PANu je tedy nutno provést předem pomocí přeformátování EPB s vstupním i výstupním formátem ANSI X9.8.

Použijeme-li k přeformátování z VISA-3 do ANSI X9.8 například EPB s takto předem změněnou druhou číslicí PANu, pak podle pravidel VISA-3 je PIN z CPB

[18] Nechť $x, n \in \mathbb{Z}$ a n je sudé. Pak platí, že $x < n \Leftrightarrow x \oplus 1 < n$.

extrahován jako `04PPPP`⊕`00000x`. V případě, že `x` = `P`⊕`F` (tj. `x`⊕`P` = `F`) je však extrahován pouze jako `04PPP`, což lze detekovat převedením zpět do původního formátu a porovnáním obou zašifrovaných PIN-bloků. Tato metoda již umožňuje jednoznačně identifikovat všechny číslice PINu jako `P` = `x` ⊕ `F`, přičemž k odhalení prvních dvou číslic je opět nutno nejprve PIN rozšířit.

3.4.3 Key Separation Attacks

Jak jsme ukázali, principy oddělování jednotlivých typů klíčů ještě nebyly zcela pochopeny. To lze také tvrdit o metodách, určujících, které z klíčů mohou či nesmí mít společný typ, a demonstrujeme proto útoky, které nám to jasně dokazují.

Nedostatečná separace PIN šifrujících a PIN generujících klíčů

Tento útok zneužívá nedostatečného oddělení typů PIN šifrujících a PIN generujících klíčů, čímž je v některých API umožněno použití šifrujícího klíče namísto generujícího a naopak. Lze jej implementovat pomocí verifikační funkce a významné vstupní parametry jsou:

- Tajný klíč, kterým byl PIN-blok zašifrován (K)
- Tajný PIN generující klíč (K_{PG})
- Zašifrovaný PIN-blok (EPB)
- Číslo účtu (PAN) nutné k získání PINu z ANSI X9.8 PIN-bloku
- Validační data (VAL) – tj. většinou zase PAN
- Offset

Pro jednoduchost předpokládejme, že jsme v situaci, kdy hledáme čtyřmístný PIN a offset je roven `0000`. Protože má útočník v koprocesoru k dispozici také zašifrovaný klíč K, jimž byl zachycený PIN-blok zašifrován, může pomocí překladové funkce tento EPB převést do formátu ANSI X9.8 s nulovým[19] PANem (formát CPB je pak tvaru `04PPPPFFFFFFFFFFFF`). Kromě tohoto EPB a offsetu může útočník verifikační funkci podstrčit namísto K_{PG} klíč K a namísto validačního řetězce VAL obsahujícího PAN validační řetězec VAL tvaru `04UVWXFFFFFFFFFF`, kde UVWX je zkoušený PIN.

Verifikační funkce napřed VAL zašifruje klíčem K, zkrátí na čtyři číslice, decimalizuje, přičte offset a výsledný vygenerovaný PIN porovná s PINem extrahovaným (opět pomocí klíče K) z EPB. Pokud se útočníkovi povede uhodnout číslice PINu ve validačních datech (tj. hodnotu UVWX), tak ví, že jejich zašifrováním klíčem K se uvnitř funkce vygenerovala stejná hodnota, jakou má EPB. Zkrácením a decimalizací této hodnoty pak vznikne IPIN, který tedy lze získat i z EPB. Toho lze využít k nastavení offsetu tak, aby v případě uhodnutí číslic PINu proběhla verifikace úspěšně (což se zatím nestalo). Protože PIN=IPIN+offset, a tedy i offset=PIN–IPIN

[19] K úspěšnému provedení útoku mohou některá API vyžadovat i použití konkrétního nenulového PANu, nutného k získání PINu z ANSI X9.8 PIN-bloku (je-li například požadováno, aby validační data VAL byla tvořena pouze dekadickými ciframi).

(všechny operace samozřejmě po cifrách a modulo 10), pak pokud útočník jako IPIN použije hodnotu získanou zkrácením a decimalizací známého EPB, tak v okamžiku uhodnutí PINu ve validačních datech proběhne volání verifikační funkce úspěšně.

Není-li z důvodů decimalizace určení PINu jednoznačné a verifikační funkce proběhne úspěšně vícekrát, tak je nutno s odlišnými PANy celý proces i několikrát zopakovat (změna PANu pak samozřejmě ovlivní kromě CPB i hodnotu VAL). Zjednodušený pseudokód tohoto útoku je uveden níže.

```
K=PIN_sifrujici_klic;
KPG=K;
EPB=zachyceny_EPB
IPIN=zkrat_a_decimalizuj(EPB);

for(i=0;i<9999;i++) {
  ZKOUSENY_PIN=ZAROVNEJ_NA_CTYRI_CIFRY(i);
  VAL=04||ZKOUSENY_PIN||FFFFFFFFFF;
  OFFSET=ZKOUSENY_PIN - IPIN;
  if (verify(K,EPB,KPG,VAL,OFFSET)) {
    printf("Hledany PIN je pravdepodobne %s", ZKOUSENY_PIN);
  }
}
```

Nedostatečná separace klíčů pro odlišné verifikační algoritmy

V této části ukážeme, že zkombinováním dvou samostatně naprosto bezpečných verifikačních algoritmů lze zcela narušit bezpečnost daného API. Nutným předpokladem je povolení používání stejných typů klíčů ve dvou různých verifikačních funkcích.

VISA PIN Validation Value (PVV) generující algoritmus[20] pomocí verifikujícího klíče nejprve zašifruje tzv. „transaction security parametr“ (TSP), který je vytvořen zřetězením dvanácti nejlevějších číslic PANu a čtyř číslic PINu (extrahovaného z EPB). Výsledek je pak decimalizován tak, že se prochází zleva doprava a první čtyři dekadické číslice jsou v nezašifrované podobě vráceny jako PVV. Pokud je těchto dekadických číslic méně, tak se v druhém průchodu použijí hexadecimální číslice modulo 10.

Pravděpodobnost, že první čtyři číslice jsou dekadické a utvoří tak PVV, je $(10/16)^4 \doteq 0,1526$. Vyzkoušením sedmi modifikací PANu (a tím i TSP) lze tedy statisticky očekávat získání jedné takové PVV. Pokud použijeme správný TSP jako validační data algoritmu IBM 3624 Offset, bude získaný IPIN roven hodnotě PVV. Tím se umožní výpočet offsetu pomocí PVV, čehož lze podobně jako u předchozího útoku využít k nalezení hodnoty PINu. Útočníkovi pak stačí při vytváření validačních dat zkoušet všechny offsety, které získá odečtením cifer PVV od cifer zkoušeného PINu (modulo 10) na vstupu verifikační funkce (používající metodu IBM 3624 Offset). Uhodl-li správný PIN, pak verifikace proběhne úspěšně. Zjednodušený pseudokód tohoto útoku je uveden níže.

[20] Až doposud jsme vždy k verifikaci PINů používali jen algoritmy IBM 3624 a IBM 3624 Offset.

```
K=PIN_sifrujici_klic;
KPG=K;
EPB=zachyceny_EPB;

for(i=0;i<7;i++) {
  ZKOUSENY_PAN=i;
  PVV=vypocti_PVV(EPB,ZKOUSENY_PAN); \\ TSP=ZKOUSENY_PAN||PIN

  for(j=0;j<9999;j++) {
    ZKOUSENY_PIN=ZAROVNEJ_NA_CTYRI_CIFRY(j);
    VAL=ZKOUSENY_PAN||ZKOUSENY_PIN;
    OFFSET=ZKOUSENY_PIN - PVV;
    if (verify(K,EPB,KPG,VAL,OFFSET)) {
      printf("Hledany PIN je pravdepodobne %s", ZKOUSENY_PIN);
    }
  }
}
```

Ne vždy je však v praxi útočník autorizován hodnotu PVV generovat a musí ji tedy nalézt testováním (tj. vyzkoušením všech 10 000 hodnot PVV).

3.4.4 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. Je-li tento vzorek binárních nul podstrčen verifikační funkci jako validační data, tak po zašifrování PIN generujícím klíčem (K_{PG}) vznikne hodnota, z níž je zkrácením a decimalizací vytvořen IPIN. To však umožňuje útočníkovi vypočítat IPIN pomocí testovací funkce (stačí jen zkrátit a decimalizovat klíčem K_{PG} zašifrované nuly). Použitím verifikační funkce pro nalezení offsetu k danému EPB pak získá PIN.

3.4.5 Recent PIN Recovery Attacks

Následující dva útoky byly objeveny Mikem Bondem, který se společně s Jolyonem Clulowem problematikou API na univerzitě v Cambridge zabývá. V době psaní této práce však ještě nebyly veřejně publikovány, a vycházíme zde tedy pouze z osobní e-mailové korespondence [11].

PIN Derivation Attack

Předpokládejme, že použité API je již navrženo tak, aby odolávalo všem předchozím útokům vedoucím k získání PINů (tj. například decimalizační tabulkou již nelze manipulovat apod.). K aplikaci tohoto útoku je potřeba pouze funkce generující PINy. Hlavním vstupem funkce je číslo účtu a zvolený offset, výstupem je EPB ve formátu ANSI X9.8. Použitím offsetů může útočník pro daný účet snadno vygenerovat všech 10 000 hodnot EPB a k určení jejich PINů pak musí získat alespoň jeden PIN a

jemu odpovídající EPB. V nejjednodušším případě stačí, aby pro daný účet zadal do bankomatu PIN a pokusil se jeho EPB zachytit.

Nevýhodou celého útoku je, že vyžaduje cestu k bankomatu a zpět. V současné době pracuje Bond na statistické metodě, která tuto nevýhodu odstraňuje a měla by umožnit provést útok s využitím *kouzlení* klíčů.

Collision Attack

Předpokládejme použití stejného „vylepšeného“ API jako v předchozím případě. Tento útok i přesto umožňuje pomocí funkce generující PINy (např. metodou IBM 3624 Offset) částečné identifikování posledních dvou číslic čtyřmístného PINu uloženého ve formátu ANSI X9.8. V původní verzi útoku sice Bond uvedl, že tyto dvě číslice PINu lze určit jednoznačně, ale po podrobnější analýze jsme dospěli k názoru, že celý útok umožňuje pouze jejich částečnou identifikaci (podobně jako jeden z předchozích Clulowových ANSI X9.8 útoků).

Uvažme nejprve pro jednoduchost PIN skládající se z jedné cifry. Níže jsou pro dvě odlišná čísla účtu uvedeny dvě množiny všech deseti EPB, které byly vygenerovány například opět pomocí offsetů.

PAN	PIN	xor	EPB	PAN	PIN	xor	EPB
0	0	0	21A0	7	0	7	2F2C
0	1	1	73D2	7	1	6	345A
0	2	2	536A	7	2	5	0321
0	3	3	FA2A	7	3	4	FF3A
0	4	4	FF3A	7	4	3	FA2A
0	5	5	0321	7	5	2	536A
0	6	6	345A	7	6	1	73D2
0	7	7	2F2C	7	7	0	21A0
0	8	8	4D0D	7	8	F	AC42
0	9	9	21CC	7	9	E	9A91

Jediné, co pro dané číslo účtu útočník vidí, je EPB. Sledováním obou množin však může navíc pozorovat, že v levé množině chybí EPB s hodnotou `AC42` a `9A91`. Jednoduchým výpočtem pak snadno zjistí, že jim odpovídá hodnota PINu `8` nebo `9`.

Základní myšlenka útoku tedy spočívá ve využití malého počtu hodnot vzniklých XORováním číslic '0'–'9'. Hledáním kolizí mezi výstupy generující funkce pak lze pomocí offsetu a čísla účtu částečně odhalit číslice PINu v EPB. Připomeňme, že generující funkce nejprve vypočítá z validačních dat IPIN a přičte k němu offset (modulo 10). Tím je vytvořen PIN, který je společně s PANem formátován do ANSI X9.8 CPB a zašifrován. Během tohoto procesu se poslední dvě cifry PINu XORují s prvními dvěmi ciframi PANu. Nyní se pokusme generující funkci více formalizovat. Většina parametrů funkce pro generování PINů bude mít během útoku stejnou hodnotu a neovlivní tedy generovaný PIN. Díky tomu na ni můžeme nahlížet jako na čtyři pseudonáhodné funkce (F_a, F_b, F_c, F_d), z nichž každá bude mít jako parametry první dvě cifry PANu (označme je e a f). To je dáno tím, že poslední dvě cifry čtyřmístného PINu (uloženého ve formátu ANSI X9.8) se XORují pouze s prvními dvěmi ciframi PANu, jejichž jakákoliv změna však ovlivní i celý vygenerovaný

IPIN. Výsledkem těchto funkcí je tedy vždy jedna dekadická číslice IPINu, který má tvar $F_a(e,f)||F_b(e,f)||F_c(e,f)||F_d(e,f)$, kde symbol || označuje zřetězení. K IPINu je dále přičten offset (a, b, c, d) tvořený číslicemi '0'–'9' a poslední dvě číslice právě vytvořeného PINu jsou XORovány s prvními dvěmi číslicemi PANu.

$$\begin{aligned} U_a &= (F_a(e,f)+a) \mod 10 \\ U_b &= (F_b(e,f)+b) \mod 10 \\ U_c &= ((F_c(e,f)+c) \mod 10) \oplus e \\ U_d &= ((F_d(e,f)+d) \mod 10) \oplus f \end{aligned}$$

EPB se pak vypočítá jako $Encrypt(Pad(U_a,U_b,U_c,U_d))$. Celkově tedy obdržíme funkci $Generate(a,b,c,d,e,f)$, která ze vstupních čtyř číslic offsetu a prvních dvou číslic PANu vrací EPB. Její pomocí je pak útočník schopen k danému PANu částečně identifikovat dvě číslice IPINu odpovídající hodnotám $F_c(e,f)$ a $F_d(e,f)$. K získání $F_c(e,f)$ si nejprve zvolí hodnotu DELTA a modifikací offsetu hledá kolize tak, aby platilo $Generate(a,b,c,d,e,f) = Generate(a',b',c',d',e \oplus \text{DELTA},f)$. Je-li kolize nalezena (tj. oba EPB se rovnají), tak platí $(U_a,U_b,U_c,U_d) = (U_{a'},U_{b'},U_{c'},U_{d'})$ a zejména $U_c = U_{c'}$. Z této rovnosti dále dostáváme, že DELTA $= ((F_c(e,f)+a) \mod 10) \oplus ((F_c(e \oplus \text{DELTA},f)+a') \mod 10)$. Dané hodnoty DELTA však lze získat pouze pomocí XORování omezeného počtu dekadických číslic, jejichž seznam je uveden níže.

```
DELTA=1: {0,1} {2,3} {4,5} {6,7} {8,9}
DELTA=2: {0,2} {1,3} {4,6} {5,7}
DELTA=3: {0,3} {1,2} {4,7} {5,6}
DELTA=4: {0,4} {1,5} {2,6} {3,7}
DELTA=5: {0,5} {1,4} {2,7} {3,6}
DELTA=6: {0,6} {1,7} {2,4} {3,5}
DELTA=7: {0,7} {1,6} {2,5} {3,4}
DELTA=8: {0,8} {1,9}
DELTA=9: {0,9} {1,8}
DELTA=A: {2,8} {3,9}
DELTA=B: {2,9} {3,8}
DELTA=C: {4,8} {5,9}
DELTA=D: {4,9} {5,8}
DELTA=E: {6,8} {7,9}
DELTA=F: {6,9} {7,8}
```

Řekněme například, že byla detekována kolize pro DELTA=F. To znamená, že $(F_c(e,f)+a) \mod 10$ má hodnotu 6, 7, 8 nebo 9. Další kolize pro DELTA=7 tuto množinu přípustných hodnot omezí na 6 a 7 a protože hodnota a je známá, lze již snadno určit i hodnotu $F_c(e,f)$. Tímto způsobem dokáže útočník pro dobře volené DELTA částečně identifikovat $F_c(e,f)$ pomocí průměrně dvou kolizí. Určení hodnoty $F_d(e,f)$ pak provede analogicky.

3.5 Praktická aplikace útoků na API

V této části práce se zabýváme problematikou praktické aplikace útoků na API. Jako příklad demonstrujeme, jak může útočník navrhnout útok umožňující získání PIN generujícího klíče (K_{PG}), aniž by byl odhalen bezpečnostními kontrolami bank. Vycházíme zde z [15].

3.5.1 Bankovní bezpečnost

Útok na skutečnou banku vyžaduje obcházení četných bezpečnostních procedur a opatření, která by mu v ideálním případě měla zcela zabránit. Patří mezi ně například bezpečnostní opatření v oblasti kontroly prostředí, procedurální kontroly či pravidelné bezpečnostní audity. Protože tyto důmyslné bankovní procedury zcela znemožňují provedení útoku na centrální server, je jakákoliv neautorizovaná změna účtu velmi obtížná a případná nekonzistentnost by byla ihned odhalena. Vhodnou alternativou útoku jsou tedy právě kryptografické moduly, které jsou sice většinou velmi dobře fyzicky zabezpečeny (např. splňují normu FIPS 140-1 na úrovni 4), ale přístup k jejich API a jeho používání je sledován nedostatečně. I přesto je však vhodné provést útok pokud možno najednou a co nejrychleji, protože jedině tak se sníží riziko odhalení na minimum.

Ukažme si tedy, jak lze navrhnout reálný útok na CCA API a optimalizovat jej tak, aby byl proveditelný rychle a hlavně jednorázově. Útok na získání PIN generujícího klíče se skládá ze tří základních částí.

Vytvoření testovacího vzorku – v této části je nutno vytvořit klíč určený k šifrování dat, který pak bude sloužit jako testovací vzorek pro útok na exportní klíč. Získáme ho tak, že pomocí náhodně vygenerovaných datových klíčů zašifrujeme vzorek binárních nul a hodnotu jednoho z nich pak odhalíme útokem *Meet in the Middle* (viz 3.2.3). Tento krok je nezbytný, protože exportní klíče mají povoleno pouze šifrování jiných klíčů a pro další krok tedy potřebujeme znát alespoň jeden klíč, který bude sloužit jako „data pro šifrování“. K neautorizovanému generování klíčů je použito *kouzlení*.

Nalezení exportního klíče – datový klíč z předchozího kroku zašifrujeme pomocí náhodně vygenerovaných exportních klíčů (se stejnými polovinami). Nyní opět aplikujeme útok *Meet in the Middle* a jeho pomocí získáme dva exportní 3DES klíče (se stejnými polovinami).

Export požadovaného klíče – v této fázi z těchto exportních klíčů pomocí *3DES Key Binding* útoku vytvoříme dva exportní klíče s rozdílnými polovinami. Výsledný 3DES klíč pak použijeme k exportu požadovaného PIN generujícího klíče.

Takto navržený útok však vyžaduje tři neautorizované přístupy k CCA API. Tím vystavuje útočníka značnému riziku, protože ho v případě odhalení útoku v první či druhé fázi umožňuje přistihnout při činu.

Pokusme se tedy celý útok optimalizovat tak, aby byl proveditelný jednorázově a délka neautorizovaného přístupu nebyla větší než půl hodiny. Aby toho bylo možno

dosáhnout, musí být druhá množina zašifrovaných testovacích vzorků vygenerována ihned po první. To v předchozím případě nebylo možné, protože útočník nevěděl, který z vygenerovaných datových klíčů se mu v první fázi podaří pomocí *Meet in the Middle* útoku odhalit. Generování této množiny pro každý možný klíč ale také není možné, protože počet operací kryptografického modulu by narostl exponenciálně, což by vyžadovalo řádově dny neautorizovaného přístupu.

Řešením je použití množiny souvisejících klíčů, které lze vytvořit pomocí funkce určené k importování částí klíčů. Tato množina může být získána například vygenerováním jedné neznámé hodnoty klíče a jejím postupným XORováním s hodnotami `0`–`16383`. Nalezení jednoho klíče pak umožní i kompromitování ostatních. Tím je umožněno použít v druhé fázi útoku jako testovací vzorek kterýkoliv klíč i přesto, že jeho hodnotu zatím neznáme.

Stejný problém vznikne v případě exportních klíčů, které mají být použity v třetí fázi a řešením je opět použití souvisejících exportních klíčů. Výsledkem těchto modifikací je, že samotná aplikace útoku *Meet in the Middle* se již neprovádí mezi jednotlivými fázemi, ale až na konci po nashromáždění všech potřebných dat. Celý útok má sice stále tři logické části, ale může již být proveden jednorázově a s poměrně krátkou délkou přístupu k API. V [15] je také demonstrována optimalizace prohledávání prostoru klíčů pomocí FPGA[21].

3.6 Útoky na Public Key Cryptography Standard #11

Až doposud jsme se zabývali útoky, které byly aplikovatelné především na IBM CCA či jemu podobná kryptografická API. Ta byla většinou navržena pro konkrétní kryptografické moduly a přesto se jejich bezpečnost ukázala jako nedostatečná. Nyní se zaměříme na velmi oblíbené rozhraní PKCS #11 [43], které je také často používáno jako hlavní API pro kryptografické moduly. Oproti předchozím však bylo navrženo pouze jako standardní rozhraní mezi aplikacemi a jednouživatelskými bezpečnostními zařízeními. Srovnání základních rysů CCA API a PKCS #11 pro IBM 4758 je uvedeno v [27]. Dále v této části budeme vycházet z [17, 43].

3.6.1 Základní informace o PKCS #11

Podle terminologie PKCS #11 jsou hardwarová bezpečnostní zařízení uchovávající objekty (např. data, klíče či certifikáty) a provádějící kryptografické operace nazývána *tokeny*. 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,

[21] Field Programmable Gate Array – víceúčelový programovatelný čip, určený k hardwarovému provádění specifických funkcí.

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. Pouze 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. Podrobné informace týkající se použitých funkcí PKCS #11 jsou obsaženy v příloze D.

3.6.2 Symmetric Key Attacks

V de facto nezměněné podobě lze na PKCS #11 aplikovat útoky *Key Conjuring* i *Meet in the Middle*. Protože není nijak omezeno použití 3DES klíčů se stejnými polovinami, je také možno aplikovat útok popsaný v 3.5. 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 K_{EK} a platí tedy, že $E_{K_{EK}}(K) = (E_{K_{EK}}(K_1), E_{K_{EK}}(K_2)$. 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. Abychom předešli tomuto útoku, API by nemělo umožňovat, aby byl exportovaný klíč modifikován. Toho lze dosáhnout například použitím MAC[22].

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 (např. šifrování/dešifrování dat, šifrování/dešifrování klíčů, podepisování/verifikace). Chybou API je, že umožňuje konfliktní nastavení těchto vlastností a umožňuje tak kompromitování klíčů. Je-li například klíč označen jako klíč určený k šifrování klíčů 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,

[22]Message Authentication Codes – tvoří podtřídu klíčovaných hašovacích funkcí a jejich cílem je zajištění integrity a autenticity zpráv.

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 K_{EK} (tj. $E_{K_{EK}}(K)$). Poté exportuje K_{EK} pomocí sebe sama (tj. $E_{K_{EK}}(K_{EK})$) a útokem hrubou silou zjistí jeho hodnotu. Pomocí známého K_{EK} 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=($K_A \oplus$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:

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

Tím zcela izoloval část klíče 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 (tato metoda je popsána v D.1). 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.

3.6.3 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 $C = K^e \mod n$. Tato metoda je však v případě použití malé hodnoty veřejného exponentu napadnutelná, protože pokud $K^e < n$, tak je klíč možno jednoduše dešifrovat jako $K = C^{\frac{1}{e}}$. Je-li například požadováno, aby měl veřejný klíč z důvodů zvýšení rychlosti umocňování nízkou Hammingovu váhu, je vygenerování klíče s malou hodnotou exponentu poměrně pravděpodobné[23]. 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, $d \mod p-1$, $d \mod q-1$ a $q-1 \mod p$). 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 [13]. V ideálním případě by tedy u soukromých klíčů měla být zajištěna integrita (např. použitím MAC nebo prováděním základních aritmetických testů typu $d \equiv e^{-1} \mod n$ a $n = pq$).

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

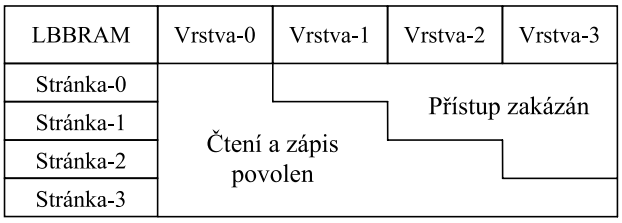
Kapitola 4

Programovatelné HSM

V této kapitole se blíže seznámíme s některými detaily programovatelných kryptografických modulů firmy IBM. Typickým příkladem takového zařízení je kryptografický koprocesor IBM 4758, jehož základní architekturou se zabývá příloha A. Vycházíme zde z [45].

4.1 Přehled základní architektury

Software a firmware je u IBM 4758 rozdělen do čtyř nezávislých vrstev, které jsou uloženy ve čtyřech různých paměťových segmentech (viz A.2.1) a postupně si od nejnižší vrstvy směrem k nejvyšší předávají řízení – tj. po vykonání kódu Vrstvy-0 je řízení předáno Vrstvě-1 (viz A.3), která jej pak předá Vrstvě-2 atd. Kód nahrávaný do těchto vrstev musí být vždy digitálně podepsán, a proto má každá vrstva vlastníka s příslušným podepisovacím klíčem. Vlastník dané vrstvy ustanovuje vlastníka vrstvy následující a poskytuje mu certifikát k jeho počátečnímu veřejnému klíči[1]. Citlivá data jednotlivých vrstev jsou uložena v příslušných chráněných stránkách LBBRAM a přístup k nim je omezen obvodem stavové kontroly (viz obr. 4.1). Pomocí těchto mechanizmů lze zaručit, že do koprocesoru je nahráván pouze autorizovaný kód, jehož integrita je navíc vždy před spuštěním ověřena.



Obr. 4.1: Přístup k chráněným stránkám LBBRAM.

Celkově tento návrh umožňuje, aby operační systém i aplikační programy mohly pocházet od navzájem různých výrobců a aby veškerá konfigurace uživatelského software probíhala až po dodání zařízení. Z hlediska údržby mohou být také přesně definovaným postupem, který využívá oddělení vrstev programového vybavení, bezpečně odstraněny případné chyby v software a firmware. Tyto metody jsou popsány dále. Poznamenejme, že vlastníkem vrstev 0 a 1 je většinou ustanoven výrobce zařízení.

[1] Segment 0 je určen pouze pro čtení, a vnitřní hierarchie klíčů tedy začíná až od Vrstvy-1.

4.2 Inicializace a certifikace zařízení

Během procesu inicializace zařízení je uvnitř koprocesoru pomocí generátoru náhodných čísel vytvořen asymetrický podepisovací klíč a několik symetrických tajných klíčů. Soukromý klíč je bezpečně uložen ve Stránce-1 v LBBRAM, zatímco veřejný klíč je exportován a vnější certifikační autorita (CA) k němu přidá informace o zařízení a aktuální softwarové konfiguraci. Takto vytvořený kořenový certifikát následně podepíše a uloží zpět do koprocesoru. Aby konečný vlastník věřil, že zařízení je skutečně autentické a neporušené, probíhá inicializace a certifikace bezprostředně na závěr jeho výroby (tj. ještě v továrně před jakýmkoliv převozem či uskladněním).

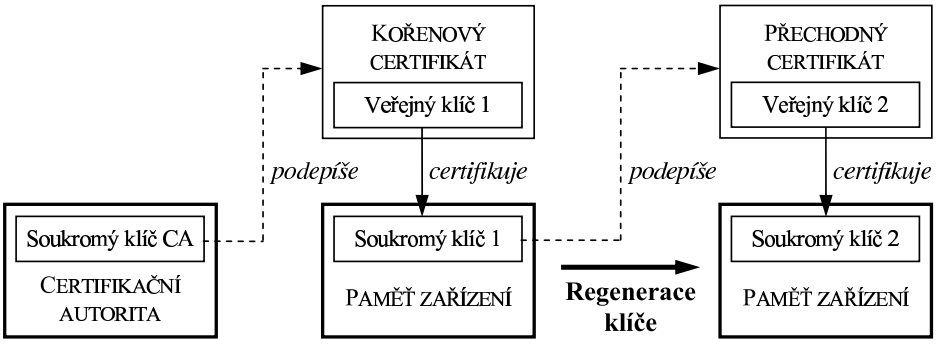
Během používání koprocesoru ovšem může dojít k situaci, kdy je potřeba kořenový certifikát změnit. Tento proces se nazývá *regenerace* klíčů a skládá se ze tří částí:

1. Vygenerování nového páru klíčů – proběhne uvnitř koprocesoru a opět za pomoci generátoru náhodných čísel.

2. Vytvoření *přechodného certifikátu* – pomocí starého soukromého klíče je podepsán certifikát, který kromě nového veřejného klíče může obsahovat například i důvody změny klíčů.

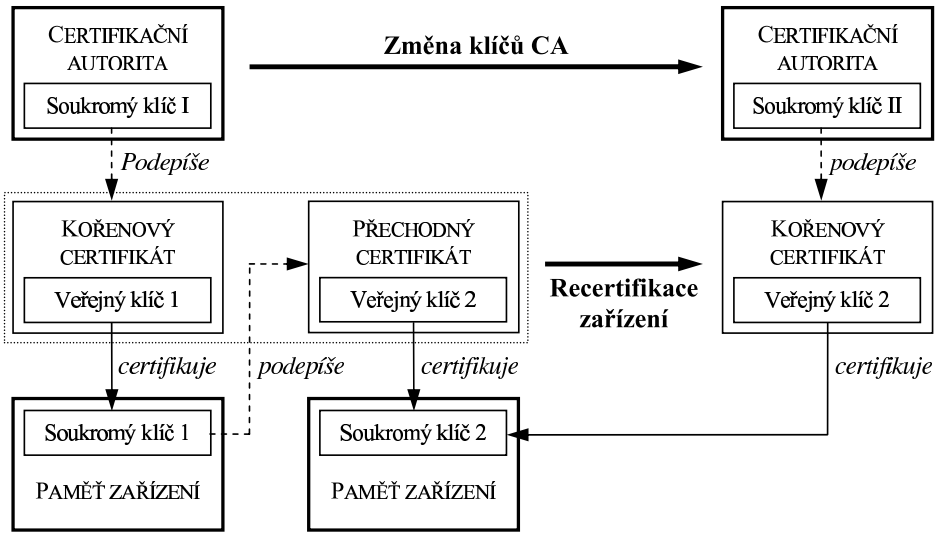
3. Ustanovení nového páru klíčů – během této atomické operace dojde k ustanovení nového páru klíčů a jejich certifikátu za oficiální. Starý soukromý klíč je automaticky vymazán.

Tento postup nám zaručuje, že seznam přechodných certifikátů spolu s kořenovým certifikátem bude vždy potvrzovat pravost aktuálního podepisovacího klíče zařízení (viz obr. 4.2). Celkově činí regenerace koprocesor nezávislý na jednom jediném páru klíčů, délce klíčů či dokonce zvoleném kryptografickém algoritmu. Její provádění je z bezpečnostního hlediska doporučeno vždy společně s nahráváním nového kódu.



Obr. 4.2: Regenerace klíče.

CA může také seznam přechodných certifikátů a kořenový certifikát nahradit jediným zcela novým certifikátem (viz obr. 4.3). Tento proces se nazývá *recertifikace* zařízení a umožňuje, aby ani CA nebyla závislá na jednom jediném páru klíčů, délce klíčů či zvoleném kryptografickém algoritmu.



Obr. 4.3: Recertifikace zařízení.

S platným kořenovým certifikátem již lze snadno vybudovat vnitřní hierarchii klíčů, začínající certifikovaným párem klíčů pro Vrstvu-1. Připomeňme, že v případě koprocesoru IBM 4758 je vlastníkem Vrstvy-1 firma IBM.

4.3 Integrita a nahrávání kódu

Výše popsaná architektura koprocesoru zajišťuje, že veškeré citlivé informace jsou přístupné pouze důvěryhodnému kódu, který je prováděn na neporušeném zařízení. Nyní si ukážeme, jakým způsobem je řešen problém zachování integrity a nahrávání nového kódu.

4.3.1 Integrita kódu

Miniboot-0 obsahuje kód umožňující hardwarové provádění DES, autentizaci pomocí tajného klíče a operace nezbytné k opravě zařízení (např. chyba kódu ve Vrstvě-1). Miniboot-1 pak zajišťuje podporu pro kryptografii s veřejným klíčem, hašování a podporu nahrávání či případných oprav dalšího kódu. Celkově IBM 4758 obsahuje tři mechanizmy související s integritou kódu:

1. Ochrana proti chybnému přepsání flash – tato ochrana se vztahuje pouze na Vrstvu-1 a je popsána v A.3. Chybné přepsání kódu této vrstvy by způsobilo vyřazení podpory pro kryptografii s veřejným klíčem, na níž je založeno ověřování vlastníků vrstev a nahrávání kódu. Tím by se stal koprocesor nadále nepoužitelný.

2. Kontrola náhodných hardwarových chyb – je prováděna pro každou vrstvu zvlášť a je založena na 64bitovém MAC[2]. Oproti 32bitovému CRC je použití

[2]Technická zpráva [45] však neobsahuje žádné bližší informace týkající se použitých klíčů.

a implementace MAC založeného na DES (v modu CBC) snazší a hardwarová podpora DES je navíc dostupná i ve Vrstvě-0. Použití 64bitového MAC je v případě segmentů, do nichž není povolen zápis, zcela dostačující. Kdyby případný útočník získal do některého segmentu právo zápisu, mohl by kromě kódu snadno měnit i jeho kontrolní součet.

3. Bezpečný boot – je proces, který je popsán opět v A.3 a během něhož dochází k postupným kontrolám integrity jednotlivých vrstev.

Tyto mechanizmy společně zaručují, že integrita důvěryhodného kódu zůstane zachována.

4.3.2 Nahrávání nového kódu

Základní princip nahrávání kódu do jednotlivých vrstev je ve zjednodušené podobě uveden v A.3.1. Podívejme se nyní na tuto problematiku podrobněji.

Vlastnictví vrstvy

Pro $0 < N < 3$ může vlastník vrstvy N vydat příkaz k ustanovení nového vlastníka vrstvy $N + 1$. Pro $2 \leq N \leq 3$ může vlastník vrstvy N vydat příkaz, že se jejího vlastnictví vzdává[3]. V obou případech se jedná o speciální příkazy Minibootu. Aby však bylo možno dosáhnout nezbytné flexibility při konfiguraci zařízení, mají vrstvy určené pro operační systém a aplikace navíc několik dalších parametrů, které určují, v jakém stavu se jejich obsah či kód nachází.

- Každá z těchto vrstev může být *vlastněna* či *nevlastněna*.
- Vlastněná vrstva může mít *spolehlivý* či *nespolehlivý* obsah.
- Spolehlivá vlastněná vrstva může obsahovat *spustitelný* či *nespustitelný* kód.

Vrstva může být nespolehlivá z několika důvodů, například vždy při prvním nahrávání kódu či při její opravě po chybném zápisu do flash. Spolehlivá vrstva může zase obsahovat nespustitelný kód například z bezpečnostních důvodů. Kompletní přehled změn stavů jednotlivých vrstev je uveden v [45].

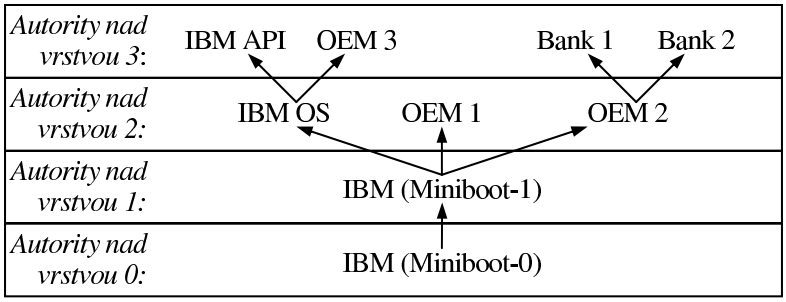
Aby mohly být příkazy k ustanovení nového vlastníka vrstvy či k vzdání se vlastnictví vrstvy provedeny, musí mít vrstva N spolehlivý obsah (tj. především veřejný klíč).

Autentizace softwarových autorit

Na obrázku 4.4 je ilustrováno, jakým způsobem jsou do stromové struktury organizovány jednotlivé *softwarové autority* – tj. někdo, kdo může autorizovat nahrávání nového firmware či software. Kořen tvoří jediný vlastník Minibootu, další generaci pak vlastníci různých druhů operačních systémů a poslední generaci vlastníci aplikací, které běží nad příslušným operačním systémem.

[3] Vzdáním se vlastnictví by pravděpodobně měly být odstraněny i veškeré citlivé informace dané vrstvy (a asi i vrstev vyšších).

K autentizaci zpráv od softwarových autorit používá koprocesor většinou metody založené na asymetrické kryptografii. Každá zpráva je nejprve podepsána pomocí soukromého klíče dané autority a později v koprocesoru pomocí jejího veřejného klíče zase verifikována. Tento veřejný klíč je uložen v segmentu příslušné vrstvy spolu s nahraným kódem a dalšími parametry[4], čímž je také v případě potřeby umožněna jeho snadná změna.



Obr. 4.4: Softwarové autority.

Nevýhodou tohoto procesu verifikace je, že vyžaduje aby byly splněny následující dvě podmínky:

- Příslušná vrstva kódu již musí být korektně nahrána – jinak koprocesor nezná odpovídající veřejný klíč.
- Miniboot-1 musí být stále funkční – jinak koprocesor ztratí podporu kryptografie s veřejným klíčem.

V důsledku toho vznikly dva podporované mechanizmy nahrávání nového kódu:

- *Ordinary loading* – v případě, že obě podmínky platí.
- *Emergency loading* – v případě, že alespoň jedna podmínka neplatí.

Při nahrávání nového kódu Vrstvy-1 či v případě poškození Vrstvy-1[5] je nutno k autentizaci zpráv použít metody založené na symetrické kryptografii. Sdílená tajemství jsou v tomto případě bezpečně uložena ve Stránce-0 v LBBRAM.

Ordinary loading

V tomto případě pro $1 \leq N \leq 3$ může vlastník vrstvy N vydat podepsaný příkaz k normálnímu nahrávání (resp. aktualizaci) kódu vrstvy N. Koprocesor pak pouze verifikuje jeho podpis přímým použitím veřejného klíče uloženého ve vrstvě N. Každý příkaz se skládá z nového kódu, nového veřejného klíče a informace k identifikaci cílového prostředí. Jako veřejný klíč může být použit samozřejmě i starý klíč – to závisí pouze na bezpečnostní politice příslušné softwarové autority. Identifikace

[4] Ty mohou sloužit například k identifikaci dané softwarové autority.

[5] Jedná se tedy o speciální případ pohotovostního nahrávání kódu do vrstvy s nespolehlivým obsahem.

cílového prostředí umožní softwarové autoritě zaručit, že její příkaz bude proveden pouze v příslušném důvěryhodném prostředí.

Pokud byla například v druhé verzi operačního systému nějakého výrobce nalezena závažná bezpečnostní chyba, je takto umožněno výrobcům aplikací zajistit, že jejich kód bude možno nahrát pouze do zařízení s operačním systémem verze tři a vyšší.

Výše popsaným způsobem lze aktualizovat i kód nižších vrstev K, které mají plnou kontrolu nad vyššími vrstvami a jejich citlivými informacemi. Tyto vyšší vrstvy však nemohou spoléhat na to, že aktualizace nižších vrstev budou vždy bezpečné a kompatibilní. V horším případě nemohou spoléhat ani na to, že příslušné softwarové autority nižších vrstev budou dostatečně chránit své soukromé podepisovací klíče. Vlastník vrstvy N má tedy také prostředek, jak se vyjádřit k případným budoucím změnám přepisovatelných vrstev $K < N$. K tomuto účelu jsou zavedeny tři parametry: věř *vždy*, nevěř *nikdy* a věř pouze, je-li příkaz pro vrstvu $K < N$ *spolupodepsán* vlastníkem vrstvy N.

Aktualizace kódu vrstvy N je tedy úspěšná, pokud zachová všechny citlivé informace této vrstvy a ponechá její kód nadále spustitelný. Navíc pokud $N < M \leq 3$, musí aktualizace kódu zachovat i všechny citlivé informace vrstvy M, což je právě tehdy, když je obsah vrstvy M spolehlivý a parametr vyjadřující důvěru vrstvě N je nastaven buď na věř vždy nebo věř pouze spolupodepsanému příkazu (a podpis vlastníka vrstvy M je v použitém příkazu obsažen). V opačném případě jsou během aktualizace všechny citlivé informace vrstvy M zničeny.

Emergency loading

Pro $2 \leq N \leq 3$ umožňuje tato metoda nahrávání kódu do vrstvy N bez znalosti jejího obsahu. Toho lze využít především v případech, kdy její obsah není spolehlivý (např. při prvním nahrávání kódu). Vlastník vrstvy N opět vydá podepsaný příkaz, podobný jako v případě normálního nahrávání kódu do vrstvy N. Veřejný klíč, který je součástí tohoto příkazu, musí být ale navíc podepsán vlastníkem vrstvy $N - 1$, čímž vznikne tzv. *pohotovostní certifikát*. Koprocesor pak nejprve pomocí veřejného klíče vrstvy $N - 1$ ověří podpis tohoto pohotovostního certifikátu a je-li v pořádku, ověří pomocí již důvěryhodného veřejného klíče vrstvy N podpis původního příkazu.

Tato metoda nahrávání kódu však přináší riziko, že vlastník vrstvy $N - 1$ může vytvořit pohotovostní certifikát libovolnému veřejnému klíči. Koprocesor si pak nemůže být jist, zda příkaz k nahrání kódu do vrstvy N pochází skutečně od jejího právoplatného vlastníka. To bylo důvodem implementace následujících opatření:

- Citlivé informace vrstvy N jsou vymazány, ale kód v této vrstvě je nadále spustitelný (přinejmenším mu věří údajný vlastník této vrstvy).
- Citlivé informace vyšších vrstev jsou vymazány a kód těchto vrstev je označen jako nespustitelný.

Obě tyto akce jsou provedeny automaticky jako součást úspěšného nahrání kódu.

4.4 Autentizace zařízení

V předchozích částech jsme si ukázali metody autentizace nahrávaného kódu. Aby však koprocesor mohl bezpečně fungovat v nedůvěryhodném prostředí, musí být možné navíc vzdáleně rozlišit mezi zprávou od reálného koprocesoru a zprávou od chytrého podvodníka. Toto je třeba zajistit vždy, kdy je požadována bezpečná komunikace alespoň dvou HSM – zařízení musí být schopna se navzájem autentizovat. K tomuto účelu lze použít vnější autentizaci, jejíž princip je popsán v A.3.2 a formální model v [46].

Kapitola 5

Závěr

V druhé kapitole jsme se seznámili s bezpečnostní problematikou a základní architekturou HSM. Byly představeny bezpečnostní požadavky na zařízení splňující normu FIPS 140-2 a několik typů útoků na fyzickou bezpečnost HSM. Pro některé z těchto útoků ještě ani v době vzniku FIPS 140-2 neexistovaly žádné testovatelné požadavky a podrobné informace týkající se například programu TEMPEST jsou dodnes přísně utajovány.

Hlavním cílem celé práce byla analýza útoků na aplikační programovací rozhraní. Tyto útoky patří do oblasti logické bezpečnosti a jsou detailně popsány ve třetí kapitole. Nejprve jsme se seznámili s problematikou kryptografických API a postupně jsme analyzovali útoky na API starších a současných HSM. Největší prostor byl věnován bezpečnosti IBM CCA API, které je proto dále popsáno v příloze B. Poté jsme se věnovali útokům vedoucím k získání PINů a kapitolu jsem zakončili praktickou aplikací útoků na skutečný bankovní systém. Během analýzy těchto útoků došlo k osobní e-mailové komunikaci s Mikem Bondem [11] a byly zpřesněny některé jeho příliš optimistické závěry týkající se určení číslic PINů. Naší pozornosti neuniklo ani velmi oblíbené aplikační programovací rozhraní PKCS #11, ale jeho specifikace bohužel ponechává mnoho implementačních detailů na konkrétním výrobci HSM, což analýzu dosti ztížilo.

Součástí práce byla také podrobná studie jednoho konkrétního zřízení, kterým se stal programovatelný kryptografický koprocesor IBM 4758. S jeho architekturou a problematikou návrhu programovatelných HSM jsme se podrobně seznámili ve čtvrté kapitole a v příloze A. Viděli jsme také, že kromě CCA API podporuje tento koprocesor i PKCS #11. Bezpečností těchto dvou API jsme se však již do značné míry zabývali v předcházející části, a detaily útoků na ně jsou proto obsaženy v přílohách C a D.

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

Reference

[1] D. G. Abraham, G. M. Dolan, G. P. Double, and J. V. Stevens. Transaction Security System. In *IBM Systems Journal*, volume 30, pages 206–229. IBM, 1991. Available at: `http://www.research.ibm.com/journal/sj/302/ibmsj3002G.pdf`.

[2] C. Adams and S. Lloyd. *Understanding PKI: Concepts, Standards, and Deployment Considerations*. Addison-Wesley, second edition, November 2002. ISBN 0-672-32391-5.

[3] 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. Available at: `http://www.cl.cam.ac.uk/ftp/users/rja14/protocols00.pdf`.

[4] R. J. Anderson and M. Bond. API-Level Attacks on Embedded Systems. In *IEEE Computer*, volume 34, pages 67–75, October 2001. Available at: `http://www.cl.cam.ac.uk/~mkb23/research/Attacks-on-Crypto-TS.pdf`.

[5] R. J. Anderson and M. G. Kuhn. Tamper Resistance – a Cautionary Note. In *Proceedings of the Second USENIX Workshop on Electronic Commerce*, pages 1–11. USENIX Association, November 1996. Available at: `http://www.cl.cam.ac.uk/~mgk25/tamper.pdf`.

[6] R. J. Anderson and M. G. Kuhn. Low Cost Attacks on Tamper Resistant Devices. In *Proceedings of 5th International Workshop on Security Protocols*, volume 1361 of *Lecture Notes in Computer Science*, pages 125–136. Springer, 1997. Available at: `http://www.cl.cam.ac.uk/~mgk25/tamper2.pdf`.

[7] R. J. Anderson and M. G. Kuhn. Soft Tempest: Hidden Data Transmission Using Electromagnetic Emanations. In *Cryptographic Hardware and Embedded Systems*, volume 1525 of *Lecture Notes in Computer Science*, pages 124–142. Springer, 1998. Available at: `http://www.cl.cam.ac.uk/~mgk25/ih98-tempest.pdf`.

[8] V. Austel, R. Perez, S. Smith, and S. Weingart. Validating a High-Performance, Programmable Secure Coprocessor. In *22nd National Information Systems Security Conference (NISSC)*, October 1999. Available at: `http://csrc.nist.gov/nissc/1999/proceeding/papers/p16.pdf`.

[9] M. Bond. A Chosen Key Difference Attack on Control Vectors, November 2001. Available at: http://www.cl.cam.ac.uk/~mkb23/research/CVDif.pdf.

[10] 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. Available at: http://www.cl.cam.ac.uk/~mkb23/research/Attacks-on-Crypto-TS.pdf.

[11] M. Bond. Osobní e-mailová korespondence, 2004.

[12] M. Bond and P. Zieliński. Decimalisation Table Attacks for PIN Cracking. Technical Report 560, University of Cambridge, Computer Laboratory, February 2003. Available at: http://www.cl.cam.ac.uk/TechReports/UCAM-CL-TR-560.pdf.

[13] 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.

[14] M. Burrows, M. Abadi, and R. Needham. A Logic of Authentication. In *Proceedings of the Royal Society of London*, pages 233–271, 1989.

[15] R. Clayton and M. Bond. Experience Using a Low-Cost FPGA Design to Crack DES Keys. In *Cryptographic Hardware and Embedded Systems*, volume 2523 of *Lecture Notes in Computer Science*, pages 579–592. Springer, 2002. Available at: http://www.cl.cam.ac.uk/users/rnc1/descrack/DEScracker.pdf.

[16] J. S. Clulow. PIN Recovery Attacks. Technical Report 0520 00296, Prism, October 2001. Available at: http://www.flurnet.org/archive/research/Clulow.pdf. Revised October 2002.

[17] 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. Available at: http://www.cl.cam.ac.uk/~jc407/SecurityOfPKCS11.pdf.

[18] J. S. Clulow. The Design and Analysis of Cryptographic Application Programming Interfaces for Security Devices. Master's thesis, University of Natal, January 2003. Available at: http://www.cl.cam.ac.uk/~jc407/Dissertation.pdf.

[19] Dallas Semiconductor. *Secure Microprocessor Chip DS5002FP*. Available at: http://pdfserv.maxim-ic.com/en/ds/DS5002FP.pdf.

[20] J. G. Dyer, M. J. Lindemann, R. Perez, R. Sailer, L. P. van Doorn, S. Smith, and S. Weingart. Building the IBM 4758 Secure Coprocessor. In *IEEE Computer*, volume 34, pages 57–66, October 2001. Available at: http://www.cs.dartmouth.edu/~sws/papers/comp01.pdf.

[21] Eracom. Eracom Security Products. Available at: http://www.eracom-tech.com/productshome.htm.

[22] Federal Information Processing Standards Publication 140-1, Security Requirements for Cryptographic Modules, National Institute of Standards and Technology, January 1994. Available at: http://csrc.nist.gov/publications/fips/fips140-1/fips1401.pdf.

[23] Federal Information Processing Standards Publication 140-2, Security Requirements for Cryptographic Modules, National Institute of Standards and Technology, May 2001. Available at: http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf.

[24] 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. Available at: http://csrc.nist.gov/publications/nistpubs/800-29/sp800-29.pdf.

[25] P. Gutman. Secure Deletion of Data from Magnetic and Solid-State Memory. In *Sixth USENIX Security Symposium Proceedings*, pages 77–90, July 1996. Available at: http://www.usenix.org/publications/library/proceedings/sec96/full_papers/gutmann/index.html.

[26] Hewlett Packard. HP's Atalla Security Products. Available at: http://h20138.www2.hp.com/page/NSPAnnment.html.

[27] IBM. Comparison of CCA and PKCS #11 v2.01 for the IBM 4758. White paper. Available at: ftp://www6.software.ibm.com/software/cryptocards/FAQCCA_and_PKCS11.pdf.

[28] IBM. IBM Comment on A Chosen Key Difference Attack on Control Vectors, 2001. Available at: http://www.cl.cam.ac.uk/~mkb23/research/CVDif-Response.pdf.

[29] IBM. *IBM 4758 CCA Installation Manual*, August 2002. Available at: http://www-3.ibm.com/security/cryptocards/html/library.shtml.

[30] IBM. *IBM 4758 General Information Manual*, May 2002. Available at: http://www-3.ibm.com/security/cryptocards/html/library.shtml.

[31] IBM. *IBM 4758 CCA Basic Services Reference and Guide, Release 2.41*, September 2003. Available at: http://www-3.ibm.com/security/cryptocards/html/library.shtml.

[32] IBM. IBM Cryptocards. Available at: http://www-3.ibm.com/security/cryptocards/.

[33] IBM. IBM Resource Access Control Facility. Available at: http://www-1.ibm.com/servers/eserver/zseries/zos/racf/.

[34] P. Kocher. Timing Attacks on Implementation of Diffie-Hellman, RSA, DSS and Other Systems. In *Advances in Cryptology – CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer,

December 1996. Available at: http://www.cryptography.com/resources/whitepapers/TimingAttacks.pdf.

[35] P. Kocher, J. Jaffe, and B. Jun. Introduction to Differential Power Analysis and Related Attacks, 1998. Available at: http://www.cryptography.com/resources/whitepapers/DPATechInfo.PDF.

[36] P. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In *Advances in Cryptology – Crypto 99 Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999. Available at: http://www.cryptography.com/resources/whitepapers/DPA.pdf.

[37] J. Krhovják. Kontrola integrity dat hašovacími funkcemi. Bachelor's thesis, Masaryk University Brno, 2003. Available at: http://www.fi.muni.cz/~xkrhovj/projekt/bakalarska_prace_final.pdf.

[38] M. G. Kuhn. Compromising emanations: eavesdropping risks of computer displays. Technical Report 577, University of Cambridge, Computer Laboratory, December 2003. Available at: http://www.cl.cam.ac.uk/TechReports/UCAM-CL-TR-577.pdf.

[39] A. J. Menezes, S. A. Vanstone, and P. C. van Oorschot. *Handbook of Applied Cryptography*. CRC Press, August 2001. ISBN 0-8493-8523-7.

[40] J. A. Muir. Techniques of side channel cryptanalysis. Master's thesis, University of Waterloo, June 2001. Available at: http://www.math.uwaterloo.ca/~jamuir/papers/mmthesis-side-channel.pdf.

[41] N-Cipher. N-Cipher Security Products. Available at: http://www.ncipher.com/products/.

[42] W. Rankl and W. Effing. *Smart Card Handbook*. John Wiley & Sons, second edition, 2000. ISBN 0-471-98875-8.

[43] RSA. *Cryptographic Token Interface Standard – PKCS #11, Version 2.11*, November 2001. Available at: ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-11/v211/pkcs-11v2-11r1.pdf.

[44] Schutzmaßnahmen gegen Lauschangriffe [Protection against eavesdropping attacks], Faltblätter des BSI 5, German Information Security Agency, Bonn, 1997.

[45] S. Smith and S.Weingart. Building a High-Performance, Programmable Secure Coprocessor. Research Report RC 21102, IBM, February 1998. Available at: http://www.research.ibm.com/secure_systems/papers/arch.pdf.

[46] S. W. Smith. Outbound Authentication for Programmable Secure Coprocessors. In *7th European Symposium on Research in Computer Security*, pages 72–89, 2002. Available at: http://www.cs.dartmouth.edu/~sws/papers/esorics02.pdf.

[47] The Complete, Unofficial TEMPEST Information Page. Available at: `http://www.eskimo.com/~joelm/tempest.html`.

[48] S. H. Weingart. Physical Security Devices for Computer Subsystems: A Survey of Attacks and Defenses. In *Cryptographic Hardware and Embedded Systems*, volume 1965 of *Lecture Notes in Computer Science*, pages 302–317. Springer, 2000.

[49] B. Yee. *Using Secure Coprocessors*. PhD thesis, Carnegie Mellon University, May 1994. Available at: `ftp://www.cs.ucsd.edu/pub/bsy/pub/th.ps.gz`.

[50] S.-M. Yen and M. Joyce. Checking Before Output May Not Be Enough Against Fault-Based Cryptanalysis. In *IEEE Transactions on Computers*, volume 49, pages 967–970, 2000.

[51] U̇berkoppeln auf Leitungen [Cross-talk on cables], Faltblätter des BSI 4, German Information Security Agency, Bonn, 1997.

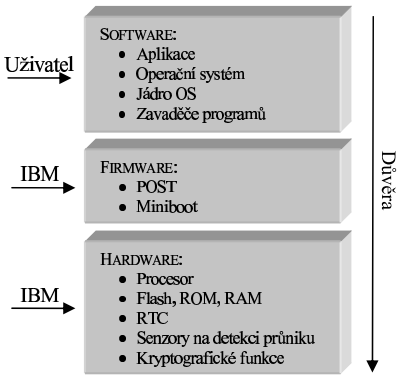
Příloha A

Koprocesor IBM 4758

IBM 4758 PCI kryptografický koprocesor je programovatelné hardwarové bezpečnostní zařízení, které umožňuje provádět symetrické a asymetrické kryptografické operace a poskytuje i podporu pro bankovní protokoly. Jeho pomocí mohou být náročné kryptografické operace vykonávány bezpečně i v méně bezpečném či nedůvěryhodném prostředí, které nám poskytují dnešní počítače. Aby takové zřízení mohlo bezpečně pracovat v nedůvěryhodném prostředí, musí být odolné nejen proti logickým útokům, ale hlavně proti útokům fyzickým. V současné době se IBM 4758 dodává ve dvou provedeních. Jsou to modely 002 a 023, které již plně nahradily[1] starší modely 001 a 013. Rozdíl mezi jednotlivými provedeními je pouze v ochraně citlivých informací uvnitř koprocesoru. Model 002 (stejně jako starší 001) splňuje normu FIPS 140-1 na úrovni 4, zatímco model 023 (stejně jako starší 013) splňuje tuto normu pouze na úrovni 3. IBM nabízí a doporučuje koprocesor jak pro použití v klasických osobních počítačích, tak také ve všech řadách IBM *Ecommerce* Serverů (tj. xSeries, pSeries, iSeries, zSeries). Dále v této části textu budeme vycházet především z [30, 45], případně i z [20, 29].

A.1 Základní informace

Celkovou strukturu koprocesoru lze rozčlenit na hardware, firmware a software (viz obr. A.1), z nichž pouze hardware a firmware jsou certifikovány podle FIPS 140-1 na úrovni 4. Dodávaný software je podle IBM „ukázkovou“ aplikací, kterou používá drtivá většina zákazníků. Hardware a firmware jsou plně pod správou IBM, která jediná je může měnit, a software je pod správou uživatele, který jej může programovat, měnit či konfigurovat dle vlastních potřeb a zájmů. Software je rozdělen do nezávislých vrstev, kde vyšší vrstvy spoléhají na danou úroveň zabezpečení, kterou jim poskytují vrstvy nižší.



Obr. A.1: Struktura koprocesoru.

[1] V tomto textu se budeme zabývat výhradně novými modely 002 a 023.

Je tedy zřejmé, že aplikace nemůže být bezpečnější než funkce jádra, které volá, a stejně tak nemůže být ani operační systém bezpečnější než hardware, který provádí jeho instrukce.

A.1.1 Typická nasazení

Výrobce se snažil o takový návrh, který umožňuje co nejširší oblast použití. Obecně lze říci, že je možné jej použít pro veškeré aplikace, které vyžadují pro svůj běh vysoce bezpečné prostředí, zaručující důvěrnost a integritu. Mezi nejvýznamnější využití IBM 4758 patří aplikace finančního průmyslu, jako například generování a verifikace PINů v ATM a POS transakčních serverech. Uplatnění naleznou i PKI aplikace jako certifikační autority, které mohou využít vysoké fyzické bezpečnosti koprocesoru, například na ukládání soukromých klíčů. Další možnosti využití zahrnují inicializaci čipových karet, šifrování dat, digitální podepisování či různé formy elektronického obchodování (např. SET).

A.2 Hardware

Po hardwarové stránce je IBM 4758 standardní rozšiřující karta (viz obr. A.2) podporující rozhraní PCI verze 2.1[2]. Na kartě je umístěn samotný koprocesor, dvě baterie a v zadní části karty také devítipinový konektor RS-232. Obvody koprocesoru jsou chráněny proti průniku (např. kovovým krytem) a jakýkoliv pokus o průnik má za následek smazání všech citlivých informací.

Obr. A.2: IBM 4758.

Vnitřní architektura koprocesoru je založena na procesoru třídy 486 a na speciálních obvodech určených pro rychlé výpočty SHA-1, DES a urychlení speciálních algoritmů využívajících modulární aritmetiky. Podstatnou součástí je i hardwarový generátor náhodných čísel a řídící obvody zajišťující ochranu proti fyzickým útokům. Nyní si popíšeme jednotlivé části zařízení podrobněji:

1. Senzory detekce průniku – jak již bylo zmíněno výše, právě v tomto bodě se od sebe liší modely 002 a 023. Zatímco u modelu 002 je vnitřní elektronika obklopena polyuretanovou směsí[3], model 023 používá pouze elektrický obvod připojený ke kovovému krytu. K dalším technikám detekce průniku patří senzory monitorující stav okolního prostředí (např. detekce nízké či vysoké teploty, radiace, tlaku apod.). Veškeré senzory jsou od okamžiku výroby neustále napájeny.

2. CPU (Central Processing Unit) – jádrem celého zařízení je procesor Intel třídy 486 s taktem 99MHz.

[2]Sběrnice PCI podobně jako koprocesor podporuje napětí buď 3.3V nebo 5V.

[3]Tvrdá a vůči agresivnímu prostředí (teplotní výkyvy, kyseliny apod.) vysoce odolná látka.

3. Rozhraní sběrnice PCI – pomocí něj je prováděna veškerá komunikace mezi koprocesorem a počítačem (zejména tedy DMA operace mezi FIFO buffery a pamětí hostitelského systému).

4. Vnitřní sběrnice – šířka této sběrnice je 32 bitů a mimo jiné podporuje například obousměrné DMA operace mezi FIFO buffery.

5. FIFO (First-In First-Out) buffery – koprocesor obsahuje dva buffery napojené na interní a externí DMA kanály a na prováděcí jednotku SHA-1 a DES. Oba podporují vysoké přenosové rychlosti a pomáhají vyrovnávat asynchronní operace.

6. Prováděcí jednotka SHA-1 a DES – tato jednotka podporuje 56bitové šifrování DES v modech ECB a CBC a hašovací funkci SHA-1. Šifrovat lze také za pomocí tříklíčového 3DES.

7. RNG (Random Number Generator) – generátor náhodných čísel je založen na všudypřítomném elektronickém šumu, který je zdrojem nepřetržitého proudu náhodných bitů. Tyto náhodné bity pak slouží jako základ (semínko, násada) pro PRNG (Pseudo-Random Number Generator).

8. Procesor na výpočty modulární aritmetiky – tento čip urychluje asymetrické kryptografické algoritmy založené většinou na modulární aritmetice. Podporuje například algoritmy RSA, DSA a Diffie-Hellman. Maximální délka klíčů je 2048 bitů.

9. RTC (Real-Time Clock) – hodiny reálného času poskytují přesné datum a čas. Jsou pod výhradní kontrolou software běžícího uvnitř koprocesoru.

10. Obvod stavové kontroly – tento „ochranný“ obvod tvoří část implementace bezpečnostní architektury a pomocí něj je uplatňována silná bezpečnostní politika. Jeho prostřednictvím přechází IBM 4758 mezi různými stavy.

11. Flash paměť – koprocesor obsahuje 4MB flash paměti, v níž je uložen firmware, operační systém a aplikační programy. Zápis do ní je však pomalý a může být uskutečněn nejvýše asi tisíckrát. Při detekci průniku obsah flash není vymazán.

12. RAM (Random Acces Memory) – koprocesoru jsou k dispozici 4MB DRAM paměti. Při detekci průniku jsou zastaveny obnovovací paměťové cykly a její obsah je takto vymazán.

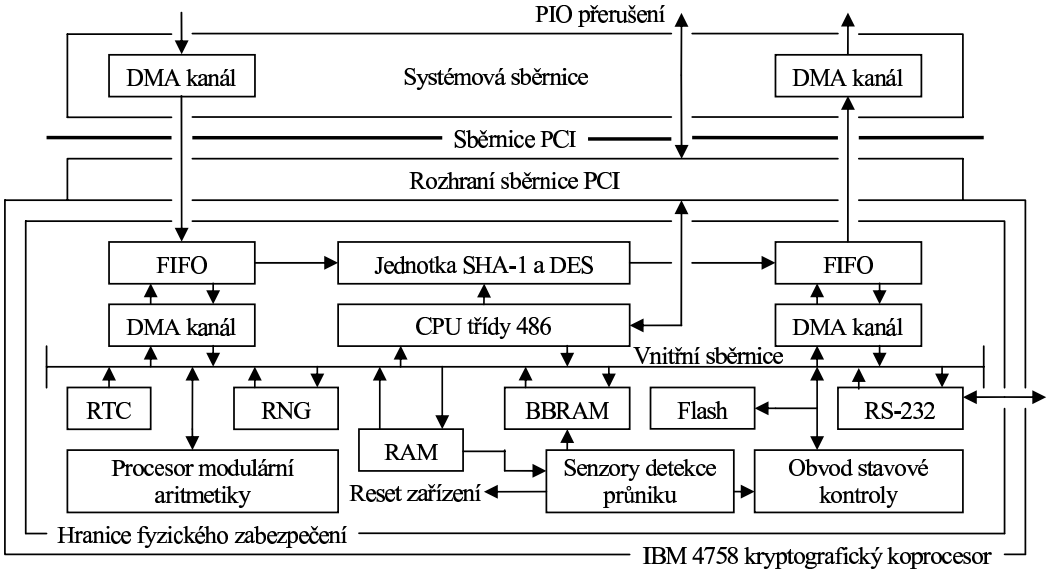
13. BBRAM (Battery-Backed Random Acces Memory) – paměť napájená bateriemi a uchovávající citlivé informace. Přístup do ní může být omezen řídícím programem. Při detekci průniku je obsah BBRAM vymazán.

14. LBBRAM (Lockable BBRAM) – tato paměť chrání citlivá data pro firmware a může být zpřístupněna pouze přes obvod stavové kontroly. Při detekci průniku je obsah LBBRAM vymazán.

15. Sériové rozhraní (RS-232) – v koprocesoru je RS-232 používáno jako alternativní rozhraní ke sběrnici PCI.

16. Baterie – zajišťují neustálý přísun energie do koprocesoru a napájejí senzory detekce průniku, BBRAM a LBBRAM. Odstranění baterií a systémového napájení způsobí okamžité vymazání obsahu těchto pamětí a zařízení zůstane navíc nadále nepoužitelné.

Veškeré přesuny dat se v IBM 4758 uskutečňují prostřednictvím vnitřní sběrnice, čímž je zajištěno, že hostitelský počítač (resp. operační systém) k nim nemá přístup. Tato separace od systémové sběrnice má však za následek celkové zpomalení toku dat. Vnitřní architektura zařízení je schematicky znázorněna na obrázku A.3.



Obr. A.3: Vnitřní architektura IBM 4758.

A.2.1 Paměťové segmenty

Aby bylo možno u programovatelného zařízení dosáhnout rozumné implementace bezpečnostní politiky při zavádění kódu (ať již firmware či software), bylo jej potřeba rozdělit na několik nezávislých *vrstev*, kde aktuálně běžící vrstva nemůže nikdy číst či měnit citlivá data nižších vrstev[4]. Jednotlivé vrstvy jsou pak uloženy do odlišných, předem stanovených, paměťových oblastí (segmentů), pomocí nichž je paměťový podsystém koprocesoru rozdělen následovně:

Segment 0 – je malá část flash paměti. Zde je uložen základní kód, který je spouštěn obvodem stavové kontroly během výrobního procesu, po jehož ukončení[5] je tato část paměti učiněna „nezměnitelnou“ (tj. chová se jako ROM). Je zde uložena také část firmware.

[4]Toho je dosaženo pomocí obvodu stavové kontroly, který k těmto datům blokuje přístup.

[5]Kromě toho přejde obvod stavové kontroly za inicializační stav (a tato změna stavu je již nevratná).

Segment 1 – je část flash paměti, ve které je uložen zbytek firmware. Ačkoliv byla data do tohoto segmentu nahrána během výrobního procesu, mohou být nahrazena použitím utility CLU (Coprocessor Load Utility).

Segment 2 – v této oblasti je umístěn, spolu s podpůrným programem, operační systém CP/Q^{++}. Tento systém podporuje aplikace uložené v segmentu 3.

Segment 3 – je určen pro aplikační programy. Při koupi zařízení je prázdný (stejně jako segment 2). K nahrání kódu do těchto segmentů se používá také CLU.

A.3 Firmware

Firmware je tvořen nízkoúrovňovým samozaváděcím programem (Miniboot) a diagnostickým programem Power-On Self-Test (POST). Tyto programy tvoří *bootovací kód* a jsou rozčleněny do dvou logických vrstev (0 a 1). Miniboot-0 a POST-0 jsou uloženy v paměťovém segmentu 0, zatímco Miniboot-1 a POST-1 jsou uloženy v segmentu 1. Zbylou oblast paměti využívá řídící program (segment 2), aplikační programy (segment 3) a data jimi používaná.

Vrstva-1 je uložena ve dvou 256KB oblastech ve flash paměti (spolu s kontrolními daty), protože chyba během zápisu do flash by mohla způsobit vyřazení podpory asymetické kryptografie a tím znemožnit další nahrávání kódu. Speciální adresovací elektronika a obvod stavové kontroly zaručují, že kód samozaváděcího programu je autentizovaný (tj. je ověřen jeho původ a integrita). Jakákoliv změna tohoto programu je zapsána do *neaktivní* oblasti paměti, což je oblast, z níž zrovna neprobíhá spouštění kódu Vrstvy-1. Tato oblast se stává *aktivní* až v okamžiku, kdy je její obsah autorizován. Výše popsaný mechanizmus tedy umožňuje, že samotné nahrávání nízkoúrovňového programu může být kdykoliv přerušeno, aniž by byla ovlivněna následná dostupnost zařízení.

Rozdělení POST na dvě vrstvy zaručuje, že POST-0 bude jednoduchý, malý a relativně bezchybný, zatímco POST-1 bude obsahovat zbylé obsáhlejší testy a v případě potřeby bude moci být přepsán či vylepšen. Po zapnutí či resetu zařízení je vždy nejprve spuštěn POST-0, který zkontroluje hardware, nezbytný pro bezpečné spuštění samozaváděcího programu Miniboot-0. Proběhne-li kontrola úspěšně, spustí se Miniboot-0, po jehož ukončení je provedena kontrola autenticity Vrstvy-1 a je jí předáno řízení. Následuje spuštění POST-1 (tj. kontrola zbylého hardware) a Miniboot-1. Tato spolupráce firmware a hardware, při níž koprocesor do flash paměti ukládá pouze autorizovaný kód, jehož integrita je vždy před podstoupením kontroly další vrstvě testována, se nazývá *bezpečný boot*.

Jako firmware by se dala také označit citlivá data, která byla do koprocesoru nahrána či vygenerována ve fázi výroby (v inicializačním stavu). Jedná se například o jedinečné sériové číslo zařízení, popis zařízení, RSA klíče, či výrobcem podepsaný kořenový certifikát potvrzující pravost těchto[6] údajů a umožňující také *vnější autentizaci* zařízení (popsána v A.3.2).

[6] S výjimkou soukromých klíčů, které jsou tajné.

A.3.1 Nahrávání kódu do segmentů 2 a 3

Kód určený pro tyto segmenty musí být digitálně podepsán, proto má každá vrstva (resp. s ní spjatý segment) vlastníka a klíčový pár, kterým může autentizovat nahrávaný kód. Vlastník dané vrstvy ustanovuje vlastníka vrstvy následující a poskytuje mu certifikát k jeho počátečnímu veřejnému klíči. Tento nový vlastník má plnou kontrolu nad následujícími nahráváními kódu, či nad případnou změnou svých klíčů. IBM je vlastníkem firmware (tj. vrstvy 0 a 1). Pro nahrání kódu do segmentu 2 je tedy potřeba mít od IBM digitálně podepsaný příkaz pro získání jeho vlastnictví. Samozaváděcí program nejprve prověří tento příkaz, a je-li vše v pořádku, ustanoví nového vlastníka. Při nahrávání samotného kódu, vždy ověřuje vlastnictví segmentu, certifikát digitálního podpisu a poté použije nyní již důvěryhodný veřejný klíč k ověření digitálního podpisu tohoto kódu. Analogicky probíhá také zápis kódu do segmentu 3.

A.3.2 Vnější autentizace

Vnější autentizace umožňuje poskytnout vnějšímu okolí přesnou informaci o koprocesoru, aplikacích běžících v koprocesoru, a o jejich historii. Tím by se kromě autentizace zařízení mělo předejít i potenciálnímu kompromitování citlivých informací za pomocí k tomuto účelu nainstalovaného, byť již dávno smazaného, software. Z tohoto důvodu je potřebný pevně daný mechanizmus zaznamenávání těchto informací a důvěryhodný kořenový certifikát. Veškeré významné změny v software jsou pak spolu s dalšími údaji (např. sériové číslo či popis daného zařízení) ukládány do speciálního *řetězce*, který je digitálně podepsán soukromým klíčem koprocesoru a je k dispozici všem aplikacím. Tento řetězec je bezpečně uložen ve Vrstvě-1. Vnější autentizace může být provedena lokálně i vzdáleně a pravost získaných informací lze ověřit pomocí IBM certifikovaného veřejného klíče zařízení.

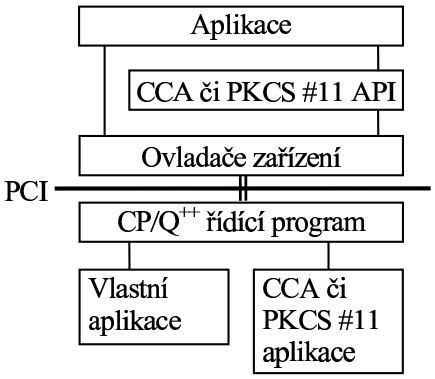
A.4 Software

Po proběhnutí bezpečného bootu předává Miniboot-1 kontrolu Vrstvě-2. Tato vrstva obsahuje operační systém[7] CP/Q^{++} a případně další podpůrné programy (CCA či PKCS #11). Spolu s CP/Q^{++} nabízí IBM dva softwarové balíčky SDT (Software Development Toolkit). První podporuje vývoj aplikací běžících v koprocesoru a druhý rozšiřuje implementaci a funkčnost CCA podpůrného programu.

Custom SDT – tento balíček obsahuje například ovladače zařízení a utility, umožňující vytváření, podepisování, nahrávání a debugování vlastních aplikací běžících ve Vrstvě-3.

CCA User Extensions SDT – používá se pouze spolu s předchozím balíčkem a kromě rozšíření funkčnosti IBM CCA aplikací poskytuje například možnost vytvořit CCA API pro vnější operační systém. Umožňuje také plný přístup k CP/Q^{++} API.

[7] V některých publikacích bývá „operační systém“ označován jako „řídící program“. V tomto textu používáme oba výše zmíněné pojmy a považujeme je za ekvivalentní.



Obr. A.4: Struktura SW.

Samotný operační systém (OS) CP/Q^{++} vznikl ze systému CP/Q jeho rozšířením o hardwarovou podporu koprocesoru. Provádí například správu paměti, multi-tasking, synchronizaci úloh a poskytuje i standardní knihovnu jazyka C. Součástí OS je také komunikační protokol zajišťující ustanovení spojení mezi IBM 4758 a hostitelským systémem. Jeho prostřednictvím probíhá veškerá V/V komunikace – tj. přenosy dat, požadavků a jejich výsledků. Vzájemná komunikace mezi jednotlivými částmi software je znázorněna na obrázku A.4.

A.4.1 PKCS #11 podpůrný program

Tento program je určen pro operační systémy AIX, Windows NT a Windows 2000. Implementuje podmnožinu kryptografických funkcí aplikačního programovacího rozhraní Cryptoki 2.01 firmy RSA™. Patří mezi ně například algoritmy DES, 3DES, RSA, DSA, SHA-1, MD5, MD2 a SSL. Poskytuje také podporu pro více zařízení či pro bezpečný současný přístup více aplikací ke koprocesoru.

A.4.2 CCA podpůrný program

Tento program je určen pro operační systémy AIX, Windows NT, Windows 2000 a OS/2. Implementuje kryptografické funkce CCA (Common Cryptographic Architecture) firmy IBM. Po změnách americké legislativy týkající se exportu kryptografických funkcí se již této aplikace netýkají žádná vývozní omezení. Pro modely 002 a 023 je určen podpůrný program CCA verze 2.x (zatímco pro starší modely 001 a 013 je určena verze 1.32). Jeho API je navrženo pro použití aplikacemi napsanými v jazyce C. Mezi podporované funkce patří například:

- Šifrování a dešifrování dat pomocí DES s využitím modu CBC a ANSI X9.23 pro zpracování posledního bloku.
- ANSI X9.9 a X9.19 DES a 3DES generování a verifikace MAC.
- Hašování pomocí SHA-1, MD5, RIPEMD-160, MDC-2 a MDC-4.
- Podepisování a verifikace podpisu algoritmem RSA s podporou formátování podle norem ISO 9796 či ANSI X9.31.
- Podpora protokolu SET.
- Generování a verifikace PINů s podporou mnoha formátů a algoritmů.
- Přešifrování a případné přeformátování PINů.

Tento software byl navíc nezávisle ověřen německou organizací ZKA[8] a byl certifikován pro použití ve specifických finančních systémech.

[8] Zentraler Kreditausschuss – německý centrální úřad pro kontrolu bank.

Příloha B

IBM 4758 CCA API verze 2.41

V této příloze se budeme zabývat základním popisem vlastností CCA verze 2.41. Nebudeme se pokoušet o výčet a podrobný popis procedur obsažených v tomto API a přehledně zdokumentovaných v publikaci [31], z níž v této části vycházíme. Nebudeme se zabývat ani speciálními technikami či algoritmy, které umožňují vzájemnou spolupráci více koprocesorů.

B.1 Volání služeb API

CCA API je navrženo tak, aby podporovalo aplikace napsané v jazyce C. Jsou-li aplikačním programem procedury (resp. funkce) API zavolány, ztrácí tento program kontrolu nad během, dokud není procedura ukončena a požadavek zodpovězen. Procedury mohou být volány i souběžně, a to jak více aplikacemi běžícími jako různé procesy, tak také jednotlivými vlákny obsaženými v jednom procesu (náležícímu jedné aplikaci). Windows NT, Windows 2000 a OS/2 omezují počet souběžných volání na 32 a lze jej zvýšit pouze použitím více koprocesorů. Na AIX se výše zmíněné omezení nevztahuje. Uvnitř koprocesoru je software CCA organizován do více vláken s odděleným paměťovým prostorem.

B.2 Řízení přístupu založené na rolích

Řízení přístupu je proces, určující které služby CCA budou mít různí uživatelé[1] v daném čase k dispozici. CCA používá, řízení přístupu založené na rolích. Názvosloví IBM vychází z definice RBAC (Role-based Access Control), která pracuje jak s konkrétními uživateli, tak s rolemi, které jsou uživatelům přiřazovány. Nejde tedy o definici použitou ve FIPS 140-1 a 2, kde jednotliví uživatelé nejsou rozeznatelní. Přístupová práva přiděluje pro jednotlivé uživatele systémový administrátor, který vytvoří množinu rolí odpovídající různým třídám uživatelů. Ti pak vlastní svůj *uživatelský profil*, ve kterém jsou přiřazeni k jedné z několika definovaných rolí.

Administrátor je většinou ustanoven pouze jeden. Je-li již systém nastaven, může být smazán i jeho profil, čímž se předejde případným změnám v řízení přístupu.

[1] V kontextu této přílohy je uživatelem míněn jak lidský uživatel, tak také automatizovaný výpočetní proces.

Obecně vzato je systém řízení přístupu založený na rolích efektivnější než systém s individuálním řízením přístupu, což je dáno především jednoduchou správou pouze několika odlišných rolí. Každý koprocesor s nainstalovaným CCA musí mít alespoň jednu *standardní* (default) roli, kterou smí používat nepřihlášení uživatelé. Chce-li uživatel využít nadstandardních služeb koprocesoru, jako je například správa klíčů, je povinen nejprve se přihlásit. Tím je aktivován jeho uživatelský profil včetně s ním spjaté role a je to zároveň jediný způsob jak aktivovat jinou roli než standardní. Ve většině aplikací však používají uživatelé koprocesor bez přihlášení (tj. nemají uživatelský profil a jsou přiřazeni k standardní roli s omezenými právy). Profily jsou nutné pouze pro bezpečnostní úředníky či jiné speciální uživatele.

Při přihlašování je identita uživatele ověřena[2] pomocí, až 64 znaků dlouhé, *přístupové fráze*. Tato fráze není nikdy v otevřené formě přenesena do koprocesoru. Je z ní vytvořen haš, který je použit jako klíč pro zašifrování přihlašovací informace. Ta je následně poslána koprocesoru, kde je pomocí (v profilu uložené) kopie haše dešifrována a následně ověřena. Po přihlášení je v koprocesoru vygenerován 192bitový *klíč sezení* (K_s), který je pomocí haše zašifrován a poslán zpět uživateli. Integrita i autentičnost (a případně i důvěrnost) další komunikace s koprocesorem je zaručena klíčovanou hašovací funkcí HMAC (a případně i pomocí DES) s klíčem K_s. Po dokončení práce s koprocesorem je nutné odhlášení a ukončení sezení, což má mimo jiné za následek smazání klíče K_s.

B.3 Správa klíčů

V podstatě veškeré kryptografické operace jsou závislé na jednom či více klíčích, kterých však uvnitř koprocesoru může být udržováno jen omezené množství. Proto mohou být při použití CCA veškeré *pracovní klíče*, včetně soukromého klíče RSA a klíče sezení, uloženy mimo koprocesor zašifrované algoritmem 3DES. Jako šifrovací klíč je použit *hlavní klíč*, který je bezpečně uložen v otevřené podobě uvnitř koprocesoru a není možné ho nijak exportovat.

Výhodou tohoto systému správy klíčů je, že množství spravovaných klíčů není závislé na omezené paměťové kapacitě koprocesoru, ale na paměti daného počítače. Navíc tyto, mimo koprocesor uložené klíče, lze použít i jinými kryptografickými uzly[3], které mají stejný hlavní klíč (např. při použití více zařízení v jednom počítači). Nevýhodou je však například složitější systém změny hlavního klíče.

B.3.1 Ustanovení a změna hlavního klíče

CCA umožňuje systémovému administrátorovi (resp. speciálnímu uživateli) vyvolat změnu či ustanovení nového hlavního klíče. Ten může být vytvořen třemi procesy:

Sestavením z více částí – tato metoda spočívá v postupném vkládání jednotlivých částí klíče, které jsou uvnitř koprocesoru ukládány do speciálního registru a XORovány. Tím je zaručeno, že znalost kterékoliv části klíče nedává žádnou informaci o výsledném hlavním klíči. Tato technika vyžaduje[4] alespoň dvě

[2] Podle terminologie FIPS 140-1 a [8] se tedy jedná o řízení přístupu založené na identitách.

[3] Nainstalovanému koprocesoru s CCA podpůrným programem se někdy říká CCA *uzel*.

[4] To je potřeba zajistit administrativní/procedurální bezpečnostní politikou.

důvěryhodné osoby (tj. bezpečnostní úředníky) takové, které nikomu neposkytnou svou část klíče. Tito bezpečnostní úředníci mají svůj profil a s ním spjatou roli, která umožňuje zadávání částí klíče. Navíc, jak je zmíněno výše, je ještě potřeba alespoň třetí důvěryhodné osoby, jejíž role povolí ustanovení či změnu hlavního klíče vyvolat. Tím dochází k rozdělení zodpovědnosti mezi nejméně tři bezpečnostní úředníky.

Náhodným vygenerováním – v tomto případě je hlavní klíč za pomocí hardware náhodně vygenerován a uložen uvnitř koprocesoru. Takto vytvořený klíč není (s výjimkou níže popsaného *klonování*) mimo koprocesor nikomu dostupný.

***Klonováním* z jednoho uzlu na druhý** – je to proces, kdy je hlavní klíč rozdělen[5] na n částí, z nichž m (kde $1 \leq m \leq n \leq 15$) je vyžadováno pro jeho znovusložení v jiném uzlu. Jednotlivé části klíče jsou pomocí 3DES zašifrovány a zaslány daným uzlům. Nový klíč, kterým bylo šifrování provedeno, je těmto uzlům bezpečně předán, zašifrován pomocí jejich veřejného RSA klíče. Tento postup se uplatní při současném použití více koprocesorů, kdy je nutné používat stejné klíče. Ty jsou v zašifrované podobě uloženy mimo koprocesor a je tedy potřeba, aby spolupracující koprocesory měly stejný i hlavní klíč.

Všechny klíče, které jsou pomocí hlavního klíče zašifrovány, musí být navíc během jeho změny přešifrovány. Toho lze dosáhnout pomocí CCA poměrně jednoduše. Je-li však hlavní klíč používán více uzly, je již celý systém jeho změny složitější.

B.3.2 Ustanovení a správa RSA klíčů

Pro vytvoření RSA klíčů je nutné zadat jejich bitovou délku. Dále je třeba stanovit, zdali (a jak) má být soukromý klíč zašifrován a má-li zůstat uchován uvnitř koprocesoru (určité množství klíčů může být trvale uchováno v koprocesotu, což zrychluje příslušné kryptografické operace). Je-li zvoleno, že má zůstat uložen uvnitř, je vrácen (ve formě čistého textu) pouze veřejný klíč. V opačném případě je nutné určit, v jaké formě má být vrácen soukromý klíč. CCA podporuje následující tři možnosti:

1. Čistý text – v tomto případě je soukromý klíč (stejně jako veřejný) vrácen v otevřené podobě a je na uživateli, aby mu poskytl dostatečnou ochranu. Tato možnost umožňuje komunikaci s aplikacemi vyžadujícími vložení soukromého klíče v otevřené podobě.

2. Zašifrován hlavním klíčem – tímto je soukromý klíč dostatečně chráněn i mimo koprocesor.

3. Zašifrován *transportním klíčem* – tato metoda umožňuje zašifrovat soukromý klíč pomocí *importního* či *exportního* DES klíče a bude popsána později v části B.3.3 týkající se algoritmu DES.

U soukromého klíče lze navíc také určit, má-li být ve formě modulo-exponent či CRT (Chinese-Remainder Theorem[6]). Obecně je použití CRT rychlejší, což je způsobeno

[5] Rozdělení je založeno na Shamirově algoritmu pro sdílení tajemství, který je také popsán v [26].

[6] Čínská zbytková věta – jeden ze základních poznatků teorie čísel. Jedná se o systém lineárních kongruencí jedné proměnné, který má právě jedno řešení.

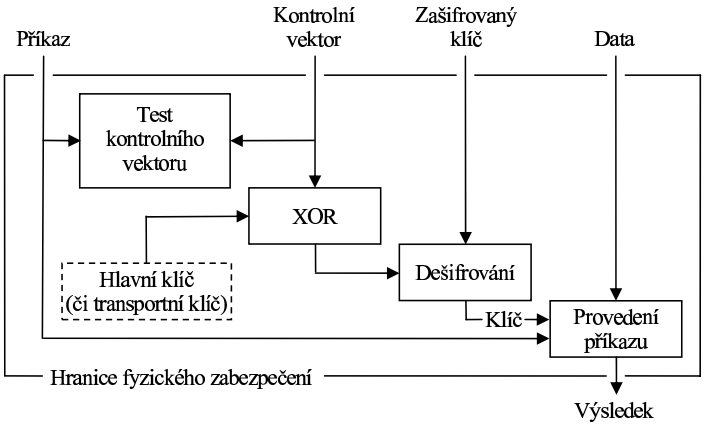
efektivnější prací s modulární aritmetikou. Výsledkem je náhodné vygenerování prvočísel[7] p a q v souladu s požadavky ANSI X9.31 a následné ustanovení RSA klíčů. S každým soukromým klíčem je také uloženo i jeho jméno. Samotná správa RSA klíčů je pak vykonávána buď jednotlivými aplikacemi, čímž je k daným soukromým klíčům omezen přístup, nebo také pomocí speciálních *přístupových monitorů*, které dohlížejí nad autorizovaným používáním těchto soukromých klíčů (příkladem může být RACF – Resource Access Control Facility systém pro kontrolu přístupu k datům, viz [33]).

B.3.3 Ustanovení a správa DES klíčů

Většina důvěrných dat, která jsou přenášena mimo koprocesor, je šifrována kryptografickým algoritmem 3DES, jehož spolehlivost závisí na utajení a správě jeho šifrovacích klíčů. CCA API umožňuje kromě generování, instalace, verifikace či distribuce těchto klíčů i jejich vzájemné rozdělení na jednotlivé typy a omezení použití na vykonávání pouze specifických úloh. Toho je dosaženo prostřednictvím *kontrolních vektorů* (datový záznam spojený s klíčem určující typ příslušného klíče). Obecně lze DES klíče rozdělit na dva základní typy:

- Vnitřní (internal) – ty jsou určeny pro použití v operacích probíhajících uvnitř koprocesoru a jsou vždy zašifrovány hlavním klíčem. Někdy se zašifrovaným vnitřním klíčům, které jsou již kompletní a připraveny k použití, říká také *operační klíče*.
- Vnější (external) – ty jsou určeny pro komunikaci s jinými uzly a jsou zašifrovány transportním klíčem, který je jiný než hlavní klíč. Lze za ně považovat i RSA klíče.

Výše zmíněné transportní klíče, někdy též označované jako *klíče-šifrující klíče*, jsou určeny výhradně k šifrování jiných klíčů (tj. nikoliv dat). Jejich speciálním případem je i hlavní klíč.



Obr. B.1: Provedení kryptografického příkazu.

[7] K ověření prvočíselnosti je použit Rabin-Millerův iterativní pravděpodobnostní test.

Kontrolní vektory

Kontrolní vektor je hodnota, která je přenášena v otevřené formě spolu s zašifrovaným klíčem a pomocí níž je specifikováno k čemu je klíč určen. Kryptograficky je kontrolní vektor s daným klíčem spojen tak, že je XORován s hlavním klíčem (či jiným transportním klíčem), a výsledkem této operace je poté daný klíč zašifrován. Tím je zaručeno, že kontrolní vektor nebude moci být modifikován, neboť jeho případná změna by způsobila nesprávné dešifrování původního klíče. Před vykonáním kryptografického příkazu, který má jako parametr zašifrovaný klíč (viz obr. B.1), je vždy zkontrolován kontrolní vektor spjatý s tímto klíčem; jeho pomocí je potvrzeno oprávněné užití klíče v daném příkazu. Je-li použití povoleno, je klíč korektně dešifrován a příkaz proběhne správně. V opačném případě nebude příkaz proveden.

Instalace a verifikace klíčů

Samotné vytvoření a instalace klíčů probíhá podobně jako u hlavního klíče. Pouze není podporováno klonování a v případě, že je klíč sestavován z více částí, jsou jednotlivé části ukládány mimo koprocesor a zašifrovány hlavním klíčem XORovaným s jejich kontrolním vektorem. Ačkoliv samotný klíč či jeho části neznáme, umožňuje CCA testovat[8], jsou-li jejich hodnoty korektní. V případě neúspěšného testu pak může být například znemožněno vkládání dalších částí klíčů.

Export a import vnějších klíčů

Aby mohly být operace na zašifrovaných datech prováděny na více uzlech, je mezi nimi potřeba bezpečně přenést příslušný šifrovací klíč. Toho může být dosaženo buď metodami asymetrickými (tj. RSA) nebo metodami symetrickými (tj. DES spolu s kontrolními vektory). Použijeme-li k přenosu klíče RSA, je před zašifrováním potřeba rozlišit dva případy:

1. Klíč je určen k operacím prováděných na důvěrných datech. Tato informace je uložena v kontrolním vektoru, což v tomto případě znamená, že všechny jeho bity jsou rovny nule. Při XORování se pak tento vektor chová jako kdyby neexistoval, což umožňuje kompatibilitu i se zařízeními nevyužívajícími CCA. K formátování klíče je použito metod RSAES-OAEP[9] a RSAES-PKCS-v1_5 definovaných v standardu PKCS #1 v2.0.

2. Klíč je určen k operacím prováděných na klíčích (tj. je to transportní klíč). V tomto případě je již kontrolní vektor nenulový a je zajištěna kompatibilita pouze se zařízeními s nainstalovaným CCA. K formátování klíče a kontrolního vektoru je použita OAEP metoda PKA92.

Má-li být k přenosu klíče použit rychlejší DES, je nejprve potřeba na dané uzly bezpečně přenést (např. pomocí RSA) a nainstalovat transportní klíč[10]. Asymetrické vlastnosti DES jsou zajištěny kontrolním vektorem, jehož pomocí je transportní klíč

[8] CCA podporuje pět verifikačních algoritmů, které jsou použity v závislosti na typu klíče.
[9] OAEP – Optimal Asymmetric Encryption Padding.
[10] To může být i hlavní klíč, což není vždy žádoucí, a proto se touto možností nebudeme zabývat.

označen buď jako *exportní* nebo jako *importní*, čímž je omezeno jeho použití pouze na operace, k nimž je určen. Je-li tedy například klíč označen jako exportní, nelze s ním žádné klíče dešifrovat, protože k tomu je určen s ním spjatý importní klíč, který je nainstalován na jiných uzlech. Kromě exportu či importu vnějších klíčů lze analogicky asymetrických vlastností DES využít buď při šifrování/dešifrování dat či PINů nebo při generování/verifikaci MACů založených na DES. Rozdíl u těchto klíčových párů je pouze v informaci uložené v kontrolním vektoru.

Obměňování klíčů

Obměňování klíčů je metoda často využívaná k inicializaci smart karet[11]. K tomuto účelu je používán *klíč-generující klíč*, který spolu s veřejnými daty (např. sériové číslo) uloženými na kartě odvodí (resp. obmění) touto kartou používaný(é) klíč(e).

B.4 Další kryptografické požadavky

Patří k nim například zajištění důvěrnosti, integrity a nepopiratelnosti dat[12]. V části týkající se zajištění integrity budeme navíc vycházet i z [37].

B.4.1 Zajištění důvěrnosti zpráv

Máme-li na kryptografických uzlech bezpečně přeneseny a nainstalovány tajné klíče, můžeme přistoupit k zabezpečení důvěrnosti zpráv. Jedná se o problematiku převedení čistého textu na zašifrovaný text a k tomuto účelu se používá algoritmus DES. Ten je řazen mezi blokové algoritmy a pracuje po blocích délky 64 bitů (8 bajtů). Kromě DES lze zvolit i jeho silnější variantu 3DES, kterou CCA používá vždy, když jsou šifrovány klíče či PINy.

Podporováno je také použití DES v ANSI X3.106 CBC modu, což však vyžaduje, aby data byla násobky osmi bajtů. Tento požadavek obecně není zaručen, a je tedy potřeba přijmout nějakou strategii zpracování (resp. doplňování) posledního bloku. CCA používá metodu ANSI X9.23. V některých aplikacích může také nastat situace, kdy lze předpovědět hodnotu prvního bloku (např. to bude část hlavičky nějakého souboru). V tomto případě je pak nutno předejít tomu, aby útočník znal část čisté i zašifrované zprávy (a znemožnit mu tímto případnou kryptoanalýzu). Toho lze dosáhnout následujícími metodami:

- Použitím inicializačního vektoru (IV) – ten je při šifrovacím procesu XORován s prvním blokem zprávy ještě před jeho zašifrováním a u dešifrovacího procesu pak analogicky po jeho dešifrování. Samozřejmě je potřeba IV bezpečně přenést mezi danými uzly, čehož je většinou dosaženo společně s přenosem klíče.

- Předřazením zprávy osmi bajty náhodných dat – této metody se využívá, pokud nelze IV bezpečně přenést. Po dešifrování je pak nutno těchto osm bajtů dat smazat. IV je v tomto případě nastaven na nulu.

[11] Jedná se o typ karet používajících namísto magnetického proužku elektronický čip.
[12] Pojem „data“ budeme v této části považovat za ekvivalentní pojmu „zprávy“.

B.4.2 Zajištění integrity a nepopiratelnosti zpráv

Zajištění integrity může být pomocí CCA dosaženo následujícími třemi metodami:

- Hašováním.
- Použitím MAC založeného na DES.
- Hašováním spolu s digitálním podpisem.

Avšak pouze hašování spolu s digitálním podpisem nám může potvrdit i originální původ[13] dat. V případě použití jedné či více certifikačních autorit (CA) může být navíc zaručena i jejich nepopiratelnost (tj. původce digitálního podpisu nemůže popřít, že jej vytvořil, a proto nemůže popřít ani data, která podepsal). K ověření pravosti veřejného klíče původce podpisu se pak používá certifikátu, což je metoda založená také na digitálním podpisu. Problematika certifikátů a CA však již převyšuje rámec našeho zájmu a je dobře zdokumentována například v [2].

Hašování

Výsledkem použití kryptografické hašovací funkce je reprezentativní obraz dat, nazývaný též haš[14], který bývá většinou 128 či 160 bitů dlouhý. Obecně dělíme hašovací funkce do dvou základních skupin podle toho, mají-li vstup s jedním či dvěma parametry. *Bezklíčové* hašovací funkce dostávají na vstup pouze data, zatímco *klíčované* hašovací funkce dostávají na vstup navíc ještě tajný klíč. CCA podporuje následující bezklíčové hašovací funkce:

- SHA-1 – je definována ve FIPS PUB 180-1 a je také specifikována pro použití s DSS (Digital Signature Standard). Tato funkce produkuje 160bitový haš; je-li možnost volby, je většinou preferována před MD5. Nejvyššího výkonu dosahuje na RISCových procesorech.
- RIPEMD-160 – i tato funkce produkuje 160bitový haš. Vznikla jako bezpečná náhrada 128bitových funkcí MD4, MD5 a RIPEMD.
- MD5 – tato funkce je specifikována v RFC 1321 a produkuje 128bitový haš. Nejvyššího výkonu dosahuje na CISCových procesorech.
- MDC – je založena na algoritmu DES a také produkuje 128bitový haš. Tato funkce je označována za docela silnou, avšak rozumné rychlosti lze dosáhnout až s pomocí speciálního hardware.

Z klíčovaných hašovacích funkcí podporuje:

- MAC – tato funkce je založena na algoritmu DES pracujícím v modu CBC. V závislosti na délce klíče pak podporuje normy ANSI X9.9 a ANSI X9.19.

[13] Mechanizmy založené na sdílení tajných klíčů (tj. MAC) nepovolují rozlišovat původ dat mezi stranami sdílejícími klíč, a tudíž nemohou jejich originální původ potvrdit.

[14] Někdy označován jako hash value, hash code, hash result, message digest, digitální otisk či kryptografický haš.

Existuje mnoho způsobů, jak pomocí bezklíčových hašovacích funkcí zajistit a ověřit integritu dat. V nejjednodušším případě stačí pouze vytvořit haš dat a kdykoliv je pak třeba ověřit jejich integritu, stačí vytvořit nový haš a porovnat jej se starým. Pokud byla integrita dat zachována, hodnoty hašů se rovnají. Jiné metody mohou využít těchto hašovacích funkcí společně s šifrováním, čímž je pak také zajištěna důvěrnost dat. Podobně lze využít i klíčované hašovací funkce, avšak v případě společného zajištění integrity a důvěrnosti dat je bez podrobné znalosti problematiky doporučeno používat odlišné tajné klíče.

Hašování s použitím digitálního podpisu

Tato metoda zajišťuje integritu dat použitím logického komunikačního kanálu, který je vytvořen digitálním podpisem. Ten je založen na asymetrické kryptografii, a v podstatě se nejedná o nic jiného než o vytvoření haše a jeho následné zašifrování soukromým klíčem odesílatele. Kdokoliv kdo má příslušný veřejný klíč pak může podpis ověřit. Pouze vytvoří nový haš dat a pomocí veřejného klíče dešifruje haš původní. Jsou-li hodnoty rovny, znamená to, že integrita podepsaných dat zůstala zachována.

CCA vytváří digitální podpisy pomocí algoritmu RSA, který pracuje s bloky stejné bitové délky jako je hodnota modulu jeho klíčů. Podporováno je formátování kratšího haše do těchto bloků podle ANSI X9.31, ISO-9796, PKCS #1 či jeho doplnění nulami.

B.5 Finanční služby

K velmi důležitým vlastnostem CCA patří podpora finančních služeb, jako jsou operace prováděné na PINech či komunikace pomocí protokolu SET. Použití PINů spočívá v autorizaci osobních finančních transakcí, které uživatel provádí například z ATM či jiných podobných zařízení. Bezpečná manipulace s PINem je zajištěna pomocí jeho formátování do jednoho z podporovaných formátů PIN-bloku a následným zašifrováním algoritmem 3DES. PIN-blok je většinou 64 bitů dlouhý a kromě čísel[15] PINu obsahuje také doplňující hodnoty. Jejich význam, kromě doplnění PIN-bloku do 64 bitů, spočívá také ve zvýšení jeho „proměnlivosti“[16].

Při manipulaci s bankomatem vloží většinou zákazník kartu s magnetickým proužkem a zadá PIN určený k jeho identifikaci. ATM pak z karty získá číslo účtu (PAN) a případně další potřebné informace. Následně je PIN zformátován do PIN-bloku, zašifrován a spolu s dalšími daty odeslán k verifikaci. Obecně se používají dvě základní metody verifikace:

- Výpočet PINů – při použití této metody se nejprve dešifruje PIN-blok a získá se z něj uživatelem zadaný PIN. Poté se spočítá s čísla účtu odvozený PIN, který se případně opraví pomocí *offsetu*[17]. Nakonec se hodnoty těchto PINů porovnají a je vrácen výsledek. PIN však nemusí být vždy odvozen z čísla

[15] PIN se může skládat ze 4–12 číslic, přičemž každá z nich je uložena ve 4 bitech a je ohodnocena jako '0'–'9' v desítkové soustavě nebo někdy jako '0'–'9' a 'A'–'F' v šestnáctkové soustavě.

[16] Typický čtyřmístný PIN může nabývat pouze 10 000 hodnot, což by bez použití doplňujících dat platilo i pro PIN-blok, do něhož byl tento PIN formátován.

[17] Vygenerovaná hodnota, která je uložena na kartě v otevřené podobě.

účtu, ale může být přiřazen bankou či zvolen zákazníkem. V tomto případě pak může být použit k výpočtu nějaké hodnoty, která se uloží na magnetickém pásku karty a je později použita k verifikaci.

- Databáze PINů – tato metoda využívá zašifrovaných PIN-bloků zákazníků, které jsou uloženy ve verifikační databázi. Zadaný PIN je pak v závislosti na použitém formátování buď přešifrován či přímo porovnán se zašifrovaným PIN-blokem uloženým v databázi.

Pro popis různých typů PINů užívá IBM následující terminologie:

- A-PIN – je PIN odvozený z čísla účtu, z PIN-generujícího klíče a případně dalších vstupů (např. decimalizační tabulka).
- C-PIN – je PIN, který by měl zákazník použít k vlastní identifikaci. Může být buď přidělen nějakou institucí či přímo zvolen zákazníkem.
- O-PIN – je většinou nazývaný *offset* a je odvozený z A-PINu a C-PINu.
- T-PIN – je zákazníkem zadávaný PIN, který je určen pro verifikaci.

Kromě šifrování umožňuje CCA také generování, verifikaci či přešifrování PINů a podporuje i více metod výpočtů PINů (např. IBM 3624 PIN, IBM 3624 PIN Offset, VISA PIN Validation Value, Netherlands PIN, Interbank PIN, IBM German Bank Pool Institution PIN), PIN-blok formátů (např. IBM 3624, ISO-0, ISO-1, ISO-2) či metod extrahování PINů.

Příloha C

Detaily útoků na CCA API

Cílem této přílohy je podrobný rozbor jednotlivých útoků na CCA API, a to především z hlediska použitých funkcí a aktuální konfigurace zařízení. Nejprve se s funkcemi CCA API blíže seznámíme a pak přejdeme k rozboru jednotlivých útoků. Vycházíme zde z [4, 9, 10, 12, 31].

C.1 Funkce CCA API

Veškeré informace vyměňované mezi aplikačním programem a funkcemi CCA API se předávají ve formě parametrů. Každá funkce má pevně daný počet parametrů a i když nejsou použity, musí být vždy při volání funkce přítomny. Všechny tyto parametry jsou ukazatele na proměnné používané aplikačním programem. První čtyři parametry mají všechny funkce CCA API stejné. Jedná se o návratové hodnoty funkcí, jejichž význam je popsán v [31]. Podívejme se nyní na to, jak vlastně taková funkce CCA API vypadá. Jako příklad nám bude sloužit funkce `Encrypted_PIN_Verify`, jejíž cílem je extrahovat ze zašifrovaného PIN-bloku zákazníkem zadaný T-PIN a porovnat jeho hodnotu s A-PINem. Tato funkce má celkově třináct parametrů, jejichž výčet a stručný popis je uveden níže.

název parametru	v/v	typ	délka [bajty]
return_code	výstup	integer	
reason_code	výstup	integer	
exit_data_length	vstup/výstup	integer	
exit_data	vstup/výstup	string	exit_data_length
PIN_encrypting_key_identifier	vstup	string	64
PIN_verifying_key_identifier	vstup	string	64
PIN_profile	vstup	string_array	24 nebo 48
PAN_data	vstup	string	12
encrypted_PIN_block	vstup	string	8
rule_array_count	vstup	integer	1, 2 nebo 3
rule_array	vstup	string_array	8*rule_array_count
PIN_check_length	vstup	integer	
data_array	vstup	string_array	3*16

Protože první čtyři parametry mají všechny funkce CCA API stejné, popišme si tedy význam pouze zbylých devíti parametrů:

1. $PIN\_encrypting\_key\_identifier$ je ukazatel na klíč, jímž je PIN zašifrován;
2. $PIN\_verifying\_key\_identifier$ je ukazatel na klíč, jehož pomocí je z validačních dat vypočítán A-PIN;
3. $PIN\_profile$ je proměnná skládající se z tří 8bajtových řetězců (což je ekvivalentní jednomu 24bajtovému). Definuje formát PIN-bloku, kontrolu formátu (pro IBM 4758 vždy nastaveno na NONE a čtyři mezery) a doplňující číslice;
4. PAN_data je řetězec se zákazníkovým číslem účtu, které je použito k získání PINu z dešifrovaného PIN-bloku. Funkce jej použije pouze v případě, že je v $PIN\_profile$ formát PIN-bloku nastaven na ISO-0;
5. $encrypted\_PIN\_block$ je řetězec obsahující zašifrovaný PIN-blok;
6. $rule\_array\_count$ je počet elementů $rule\_array$;
7. $rule\_array$ je pole obsahující klíčová slova, která určují metody výpočtu a extrahování PINu;
8. PIN_check_length je počet číslic PINu, které by funkce měla ověřit;
9. $data\_array$ je proměnná skládající se z tří 16bajtových řetězců (což je ekvivalentní jednomu 48bajtovému). Význam jejich hodnot je závislý na použité metodě výpočtu PINu.

Poznamenejme ještě, že některé implementace CCA mohou mít změněné pořadí prvků v poli $rule\_array$. Mírně zjednodušený příklad kódu skutečného volání této funkce je uveden níže.

```
Encrypted_PIN_Verify(
    A_RETRES, A_ED,                 // návratové kódy
    trial_pin_kek_in, pinver_key,   // šifrovací klíče
    (UCHAR*)"3624    " "NONE    "   // formát PIN-bloku
                       "       F",  // doplňující číslice
    (UCHAR*)"            ",         // PAN pro získání PINu z ISO-0
    trial_pin,                      // zašifrovaný PIN-blok
    I_long(2),                      // počet elementů rule_array
    (UCHAR*)"IBM-PINO" "PADDIGIT",  // verifikační metoda
    I_long(4),                      // počet číslic PINu
    "0123456789012345"              // decimalizační tabulka
    "123456789012    "              // validační data (PAN)
    "0000            "              // offset
 );
```

C.2 Implementační detaily útoků na CCA API

Všechny útoky na CCA API jsou do jisté míry závislé na konfiguraci řízení přístupu. Aby uživatel mohl danou funkci CCA API použít, musí k tomu mít ve své roli oprávnění. Tím se přinejmenším omezí skupina potenciálních útočníků na ty, kteří mají oprávnění k použití funkcí nezbytných pro příslušný útok. V lepším případě může dokonce správná konfigurace řízení přístupu některým útokům zcela zabránit.

C.2.1 Conjuring Keys From Nowhere

K aplikaci útoku *kouzlení* klíčů není vyžadována žádná z funkcí CCA API. Útočník je schopen mimo koprocesor neautorizovaně generovat klíče, a chová se tedy jako uživatel, který má ve své roli povoleno použití funkce `Key_Generate`.

C.2.2 A Chosen Key Difference Attack on Control Vectors

Cílem tohoto útoku je neautorizovaná změna kontrolního vektoru při importu klíčů. K jeho úspěšnému provedení potřebuje mít útočník povoleno použití dvou funkcí CCA API. První z nich je `Key_Part_Import` sloužící k nashromažďování jednotlivých částí klíče. Ty jsou pak v zašifrované podobě uloženy jako částečný klíč, a musí mít tedy v kontrolním vektoru nastaven KEY-PART bit na hodnotu jedna. Tento částečný klíč vzniká postupným vkládáním jednotlivých částí klíče, které jsou spolu vzájemně XORovány. Je-li při volání této funkce jedno z klíčových slov v *rule_array* nastaveno na LAST, změní se v kontrolním vektoru KEY-PART bit na nulu. Tím je vkládání klíče ukončeno. Aby mohl útočník funkci `Key_Part_Import` použít, musí mít ve své roli aktivovány příkazy LOAD_FIRST_KEY_PART pro použití s klíčovým slovem FIRST nebo COMBINE_KEY_PARTS pro použití s klíčovými slovy MIDDLE či LAST. Druhou funkcí je `Key_Import`, používající se k importu DES klíčů zašifrovaných pomocí transportního klíče typu IMPORTER. Po importu je vždy daný DES klíč zašifrován hlavním klíčem zařízení. Útočník musí mít k použití této funkce ve své roli aktivován příkaz REENCIPHER_TO_MASTER_KEY a musí mít povolen import požadovaného typu klíče (na původním typu klíče však nezáleží).

Ochrana navrhovaná IBM

Po objevení tohoto útoku vydala IBM souhrn doporučení [28], kterak mu předejít. Nejjednodušší a nejúčinnější možnost je zcela přestat používat distribuci klíčů po částech a využít technik založených na asymetrické kryptografii. Veškerá s klíčem přenášená data pak budou při přenosu vždy zašifrována veřejným RSA klíčem příjemce, čímž se podobným útokům zcela zabrání. V CCA API je k tomuto účelu obsažena dvojce funkcí `PKA_Symetric_Key_Export` a `PKA_Symetric_Key_Import`. První z nich při exportu zašifruje symetrický DES klíč veřejným RSA klíčem příjemce a druhá naopak při importu dešifruje přijatý klíč svým soukromým RSA klíčem. Distribuce požadovaného klíče pak lze uskutečnit pouze pomocí těchto dvou funkcí.

Druhou možností ochrany je správná konfigurace řízení přístupu. Žádná osoba by neměla mít oprávnění ke spouštění funkcí `Key_Part_Import` a `Key_Import` zároveň. Dokonce i při použití asymetrických technik by neměl mít nikdo právo spouštět

funkci `Key_Part_Import`, čímž se předejde nahrávání klíčů v nezašifrované podobě a útoku se tak opět zabrání.

Posledním doporučením je použití mechanizmů fyzické a administrativní bezpečnostní politiky. Jejich cílem je maximální omezení přístupu k zařízení a dodržování speciálních bezpečnostních procedur při vkládání částí klíčů. Například jednu část klíče sdílejí dvě důvěryhodné osoby a při jejím vkládání jedna kontroluje, co druhá píše. Jinou možností může být ustanovení další osoby, která po vložení poslední části klíče použitím funkce `Key_Test` ověří jeho integritu.

C.2.3 Import/Export Loop Attack

Tento útok je rozšířením útoku předcházejícího, a k jeho realizaci jsou tedy nezbytné i stejné funkce. Kromě funkcí `Key_Part_Import` a `Key_Import` vyžaduje navíc funkci `Key_Export`, která se používá k exportu DES klíčů. Každý DES klíč je po exportu vždy zašifrován transportním klíčem (typu EXPORTER) XORovaným s příslušným kontrolním vektorem. Aby útočník mohl tuto funkci použít, musí mít ve své roli aktivován příkaz REENCIPHER_FROM_MASTER_KEY. Funkce `Key_Part_Import` se zde opět používá ke XORování rozdílů typů k již existujícím klíčům (tj. z klíče K_{EK1} se její pomocí vytvoří K_{EK2} a z klíče K_E zase K_I). Ochrana proti tomuto útoku je stejná jako v předchozím případě.

C.2.4 3DES Key Binding Attack

Podobně jako v případě *kouzlení* klíčů zneužívá i tento útok spíše chybu v návrhu CCA API. Ta umožňuje neautorizovaně manipulovat s mimo koprocesor uloženými 3DES klíči, což spolu s aplikací *Meet in the Middle* útoku umožní kompromitování většiny koprocesorem chráněných klíčů. K efektivnímu provedení útoku však útočník potřebuje funkci `Key_Generate`, která umožňuje mimo jiné i generování 3DES klíčů se stejnými polovinami. Toho je dosaženo nastavením parametru *key_length* na hodnotu SINGLE-R. K tomuto použití `Key_Generate` musí mít útočník ve své roli aktivován příkaz REPLICATE_KEY. K exportování klíčů je pak opět potřeba funkce `Key_Export` spolu s aktivovaným příkazem REENCIPHER_FROM_MASTER_KEY.

C.2.5 Decimalisation Table Attacks

K aplikaci těchto útoků je nezbytná funkce `Encrypted_PIN_Verify`, která slouží k verifikaci PINů a je detailně popsána v části C.1. K jejímu použití musí mít útočník ve své roli aktivován například příkaz VERIFY_ENCRYPTED_3624_PIN (pro verifikaci PINů vypočtených metodou IBM 3624). Z hlediska útoků jsou podstatné parametry *PAN_data*, *encrypted_PIN_block* a *data_array*, přičemž z tohoto pole nás nejvíce zajímá právě decimalizační tabulka. Manipulace s PAN daty i s decimalizační tabulkou je snadná, protože v obou případech se jedná o nezašifrované hodnoty. Mnohem obtížnější je získání zašifrovaného PIN-bloku obsahujícího námi zvolený PIN. Někdy je sice umožněno vkládání nezašifrovaného PINu pomocí příkazu `Clear_PIN_Encrypt`, avšak s tím jsou spojena další bezpečnostní rizika[1].

[1] Není-li například při formátování do PIN-bloku použito náhodné doplnění, může útočník snadno zašifrovat všechny možné kombinace PINů, a tím získat jejich kompletní dešifrovací funkci.

Příloha D

Detaily útoků na PKCS #11

Cílem této přílohy je podrobný rozbor funkcí použitých k jednotlivým útokům na PKCS #11. Vycházíme zde z [17, 43].

D.1 Funkce PKCS #11

PKCS #11 dělí své funkce do třinácti základních kategorií, z nichž se budeme zabývat především funkcemi určenými pro správu klíčů:

- `C_GenerateKey` – generuje tajný klíč;
- `C_GenerateKeyPair` – generuje veřejný a soukromý klíč;
- `C_WrapKey` – šifruje tajný či soukromý klíč;
- `C_UnwrapKey` – dešifruje tajný či soukromý klíč;
- `C_DeriveKey` – vytváří ze základního klíče nový klíč.

Jediným podstatným rozdílem oproti CCA API je, že výsledek volání funkce je předáván přímo pomocí její návratové hodnoty. K deklaraci jednotlivých funkcí je v PKCS #11 použita `CK_DECLARE_FUNCTION(`*returnType*`,`*name*`)`, která je následována závorkami s výčtem parametrů deklarované funkce. *returnType* je typ návratové hodnoty a *name* je jméno deklarované funkce. Podívejme se například na prototyp funkce `C_WrapKey`:

```
CK_DEFINE_FUNCTION(CK_RV, C_WrapKey)(
    CK_SESSION_HANDLE hSession,
    CK_MECHANISM_PTR pMechanism,
    CK_OBJECT_HANDLE hWrappingKey,
    CK_OBJECT_HANDLE hKey,
    CK_BYTE_PTR pWrappedKey,
    CK_ULONG_PTR pulWrappedKeyLen
);
```

hSession je identifikátor sezení; *pMechanism* je ukazatel na metodu šifrování; *hWrappingKey* je identifikátor šifrovacího klíče; *hKey* je identifikátor klíče, který má být zašifrován; *pWrappedKey* je ukazatel na místo, kde bude uložen zašifrovaný klíč; a *pulWrappedKeyLen* je ukazatel na místo, kde bude uložena délka zašifrovaného klíče.

Nyní se podrobněji podívejme na funkci `C_DeriveKey`, která má prototyp:

```
CK_DEFINE_FUNCTION(CK_RV, C_DeriveKey)(
    CK_SESSION_HANDLE hSession,
    CK_MECHANISM_PTR pMechanism,
    CK_OBJECT_HANDLE hBaseKey,
    CK_ATTRIBUTE_PTR pTemplate,
    CK_ULONG ulAttributeCount,
    CK_OBJECT_HANDLE_PTR phKey
);
```

Parametr *pMechanism* je v případě této funkce ukazatel na metodu vytváření klíčů. `C_DeriveKey` podporuje následující mechanizmy:

- `CKM_CONCATENATE_BASE_AND_KEY` – vytváří tajný klíč spojením dvou existujících tajných klíčů;
- `CKM_CONCATENATE_BASE_AND_DATA` – vytváří tajný klíč přidáním nějakých dat na konec již existujícího tajného klíče;
- `CKM_CONCATENATE_DATA_AND_BASE` – vytváří tajný klíč přidáním nějakých dat na začátek již existujícího tajného klíče;
- `CKM_XOR_BASE_AND_DATA` – vytváří tajný klíč XORováním existujícího tajného klíče a nějakých dat;
- `CKM_EXTRACT_KEY_FROM_KEY` – vytváří tajný klíč z bitů jiného tajného klíče.

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

D.2 Implementační detaily útoků na PKCS #11

V následující části uvedeme, které funkce jsou použity k implementaci jednotlivých útoků na PKCS #11. Poznamenejme také, že případ kdy má normální uživatel přístup pouze ke čtení token objektů není omezující, protože pomocí funkce `C_CopyObject` lze každý z nich zkopírovat jako session object, který má pro danou aplikaci vždy i právo zápisu.

D.2.1 Symmetric Key Attacks

Key Conjuring a *3DES Key Binding Attack* využívají špatně navrženého formátu ukládání tajných klíčů, který umožňuje, aby byly neautorizovaně mimo token modifikovány. Export klíčů se provádí pomocí funkce `C_WrapKey` a import pomocí `C_UnWrapKey`. Těchto funkcí využívá i *Key Separation Attack*, který však zneužívá implementační chyby funkce `C_SetAttributeValue`. Ta umožňuje konfliktní nastavení vlastností klíče (tj. v našem případě `CKA_WRAP` a `CKA_DECRYPT`) a exportovaný klíč pak lze snadno pomocí funkce `C_Decrypt` dešifrovat jako normální data. *Weaker Key/Algorithm Attack* zneužívá implementační chyby funkce `C_WrapKey`, která umožňuje k exportu klíčů využít i algoritmů používajících krátký šifrovací klíč. *Meet in the Middle Attack* a *Related Key Attack* využívají k manipulaci s klíči funkci `C_DeriveKey` a její metodou `CKM_XOR_BASE_AND_DATA`. S využitím metody `CKM_CONCATENATE_BASE_AND_DATA` lze také zvětšit délku klíče a *Related Key Attack* tak rozšířit i na dvouklíčový 3DES. Funkci `C_DeriveKey` využívá také *Reduced Key Space Attack*, který je založen na její metodě `CKM_EXTRACT_KEY_FROM_KEY`.

D.2.2 Public Key API Attacks

Small Public Exponent with No Padding Attack zneužívá asymetrickou šifrovací metodu `CKM_RSA_X_509` funkce `C_WrapKey`. *Trojan Public Key Attack* je založen na špatně navrženém formátu ukládání veřejných klíčů, který umožňuje použít jako šifrovací klíč funkce `C_WrapKey` libovolný veřejný klíč. *Trojan Wrapped Key Attack* využívá podobné chyby i v případě tajných klíčů, které pak lze libovolně importovat pomocí funkce `C_UnWrapKey`. *Private Key Modification Attack* oproti tomu umožňuje pomocí funkce `C_UnWrapKey` pouze importovat pozměněný soukromý klíč.