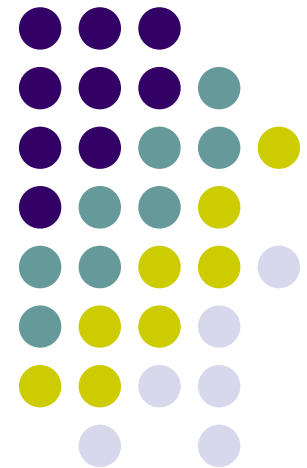


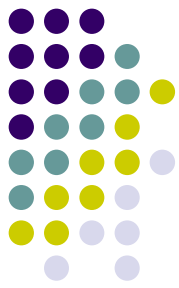
HSM a problémy s bezpečností API

Masarykova univerzita v Brně
Fakulta informatiky



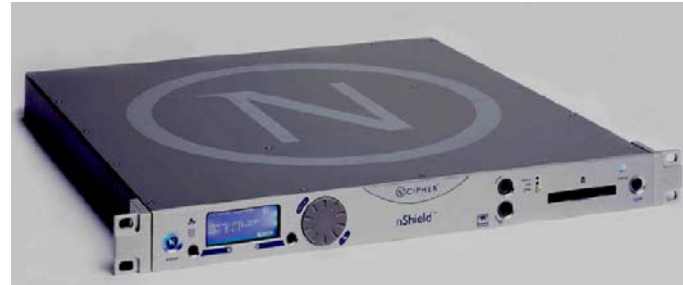
Jan Krhovják
Daniel Cvrček
Vašek Matyáš





Shrnutí

- Úvod
 - Motivace
 - Základní terminologie
- Architektura
- Bezpečnostní požadavky na kryptografické moduly
 - FIPS 140-2
- Útoky na a přes API
 - Úvod do problematiky API pro HSM
 - Klíče a jejich integrita
 - Nedostatečná kontrola parametrů funkcí
 - Nevynucení politiky – PKCS #11

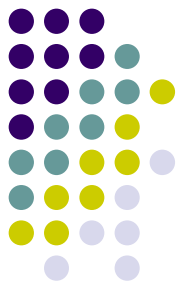


Úvod



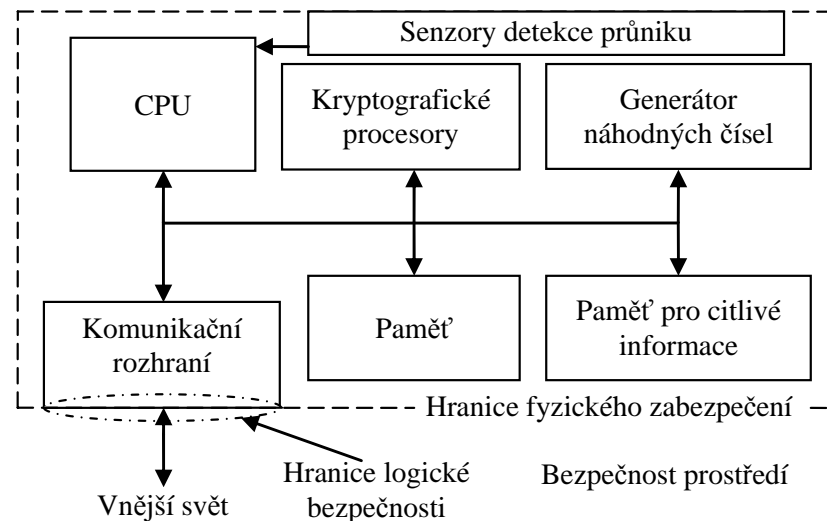
- Motivace
 - Proč/kde se HSM (Hardware Security Module) používají
 - Výhody a nevýhody tohoto přístupu
- Základní terminologie
 - Hardwarová bezpečnostní zařízení (HSM)
 - Koprocesory
 - Akcelerátory
 - Kryptografické čipové karty
 - Hostitelská zařízení
 - Útoky na HSM
 - Fyzické
 - Postraními kanály
 - Na a přes API
 - Hlavní klíč, ukládání klíčů





Architektura HSM

- Vychází z klasické von Neumannovy architektury
 - + Mechanizmy fyzické ochrany
 - + Generátory náhodných čísel
 - + Speciální koprocesory
 - + NVRAM
 - V/V obvody
- Snadná verifikace
- Bezpečnost
 - Fyzická
 - Logická
 - Prostředí
 - Operační
- IBM 4758



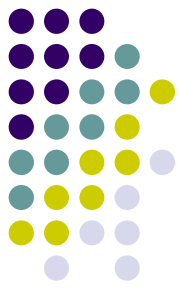


Fyzická bezpečnost

- Evidence průniků (tamper evidence)
 - Průnik => zanechání viditelných stop
 - Chemické či mechanické prostředky
- Odolnost proti průnikům (tamper resistance)
 - Pouze do jisté úrovně!
 - Chemicky odolné látky, ocelové kryty
- Detekce průniků (tamper detection)
- Použitím speciálních el. obvodů (senzorové sítě ...)
- Odpověď/reakce na průniky (tamper response)
 - Důsledek detekce => zabraňuje získání citlivých informací
 - Smazání/přepsání/zničení paměti

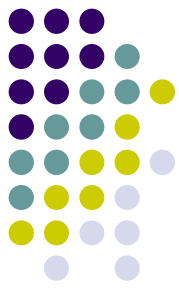


Bezpečnostní požadavky na HSM: FIPS 140-2 (I)



- Vztahují se k bezpečnému návrhu a implementaci modulu
- Celkem 11 oblastí bezpečnostních požadavků – například:
 - Dokumentace kryptografického modulu
 - Porty a rozhraní
 - Role, služby a autentizace
 - Fyzická bezpečnost
 - Operační prostředí
 - Správa klíčů
 - Zmírnění jiných útoků
- Každý modul je vůči nim nezávisle testován
- Výsledná úroveň = minimální hodnota dosažená byť jen v jediném z testů

Bezpečnostní požadavky na HSM: FIPS 140-2 (II)



- Norma definuje 4 úrovně zabezpečení
 - Úroveň 1 – není vyžadována fyzická ochrana
 - Příkladem jsou osobní počítače
 - Úroveň 2 – vyžaduje zajištění evidence průniků
 - Autentizace založená na rolích
 - OS musí splňovat určité bezp. požadavky
 - Příkladem jsou čipové karty
 - Úroveň 3 – již ne pouze pasivní fyzická ochrana
 - V případě detekce průniku musí být citlivá data vymazána
 - Autentizace založená na identitách
 - Příkladem zařízení je Chrysalis-ITS Luna CA³
 - Úroveň 4 – dodržování vnějších provozních podmínek
 - Příkladem zařízení je IBM 4758 či IBM PCIXCC





Logická bezpečnost

- **Kontrola přístupu**
 - Předpokladem je existence důvěryhodného prostředí
- **Kryptografické algoritmy**
 - V podstatě matematické funkce – tajné jsou pouze klíče
 - Zajištění důvěrnosti, integrity, autentizace ...
- **Kryptografické protokoly**
 - V podstatě distribuované algoritmy
 - Jejich jednotlivé kroky „propojeny“ voláním funkcí API
 - Zaměříme se na API (Application Programming Interface)
 - Jediné komunikační rozhraní mezi HSM a vnější aplikací
 - Na základě funkcí API jsou budovány protokoly
 - API HSM obsahuje stovky funkcí s mnoha parametry
=> velmi velký prostor k chybám a vzniku útoků

Útoky na a přes API



- Příklady běžně používaných API
 - Public Key Cryptographic Standard (PKCS) #11
 - Common Cryptographic Architecture (CCA)
 - Příklad – verifikační funkce

```
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  
);
```

- Útoky rozdělené podle typu chyb, kterých využívají
 - Klíče a jejich integrita
 - Nedostatečná kontrola parametrů funkcí
 - Nevynucení politiky – PKCS #11

Klíče a jejich integrita – Meet in the Middle Attack (I)



- Zneužívá
 - Malé velikosti šifrovacích klíčů algoritmu DES
 - Neexistenci omezení na generování klíčů
 - Špatný návrh rozlišování typů klíčů
- Myšlenka útoku
 - Mnoho HSM dokáže generovat desetitisíce klíčů během minut
 - Útočník vygeneruje např. 2^{16} klíčů => jimi zašifruje stejný testovací vzorek => ten si uloží (mimo HSM)
 - Poté systematicky prohledává klíčový prostor
 - Stejný testovací vzorek každým klíčem zašifruje a porovná se všemi uloženými vzorky
 - Shoda = nalezená hodnota jednoho tajného klíče
- Výpočetní složitost nalezení jednoho klíče tak klesne z 2^{55} na 2^{39}

Klíče a jejich integrita – Meet in the Middle Attack (II)



- Je-li nalezený klíč určen k šifrování dalších klíčů (terminální klíč), lze jeho pomocí přešifrovat veškerá jinými terminálními klíči chráněná data a klíče
 - Tímto způsobem lze kompromitovat osm z devíti typů klíčů, které používá Visa Security Module
- Variantu útoku lze aplikovat i na kryptografický modul Prism TSM 200
 - Lze získat hlavní klíč celého zařízení!
 - Žádná omezení na vkládání částí hlavního klíče
 - Po vložení části klíče vždy vrácen zašifrovaný kontrolní vektor

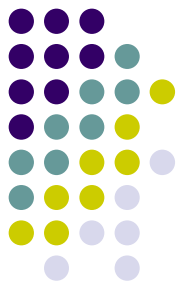


Klíče a jejich integrita – Conjuring Keys From Nowhere



- Neautorizované generování klíčů ukládaných mimo HSM
 - Náhodně vytvořená hodnota zašifrovaného klíče se podstrčí HSM
 - Při dešifrování je hodnota klíče také náhodná
 - V případě DES má s pravděpodobností $1/2^8$ správnou paritu
 - V případě dvou-klíčového 3DES-2 má správnou paritu s pravděpodobností $1/2^{16}$ (stále dosažitelné)
- Takto vložené klíče mohou posloužit k vytvoření dalších a složitějších útoků
- Obrana spočívá v pečlivějším návrhu formátu klíčů
=> např. před zašifrováním přidat kontrolní součet

Nedostatečná kontrola parametrů funkcí – Decimalisation Table Attacks (I)



- Techniky generování a verifikace PINů
 - Generování IBM 3624
 - Založeno na validačních datech (např. číslo účtu – PAN)
 - Validační data jsou zašifrována *PIN generujícím* klíčem
 - Výsledek je zkrácen a decimalizován => PIN
 - Generování IBM 3624 Offset
 - Decimalizovaný výsledek se nazývá IPIN (Intermediate PIN)
 - Zákazník si sám volí PIN
 - Offset = PIN – IPIN (po cifrách mod 10)
 - Verifikace u obou metod probíhá stejně
 - Výsledek se nakonec navíc porovná s PINem získaným z příslušného zašifrovaného EPB (zašifrovaný PIN zaslaný například bankomatem)
 - PIN nemůže být parametrem verifikační funkce!

Nedostatečná kontrola parametrů funkcí – Decimalisation Table Attacks (II)



- Příklad verifikace PINu metodou IBM 3624 Offset
 - PIN v EPB je 7216 (odeslaný z bankomatu)
 - Veřejně přístupný offset (typicky uložen na kartě) – 4344
 - Decimalizační tabulka (DT) – 0123456789ABCDEF
012345678901**2789**
 - Číslo účtu zákazníka (PAN) je 4556 2385 7753 2239
- Verifikační proces
 - PAN je zašifrován \Rightarrow 3F7C 2201 00CA 8AB3
 - Zkrácen na požadovaný počet číslic (typicky 4) \Rightarrow 3**F7C**
 - Decimalizován použitím DT, IPIN \Rightarrow 3**972**
 - Přičten offset 4344, vygenerován PIN \Rightarrow 7216
 - Porovnání tohoto správného PINu s PINem extrahovaným z EPB

Nedostatečná kontrola parametrů funkcí – Decimalisation Table Attacks (III)

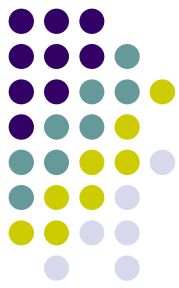


- Příklad triviální manipulace s DT
 - Předpokládejme použití čtyřmístných PINů a offsetu 0000
 - Nastavíme-li decimalizační tabulku (DT) na samé nuly, vždy se vygeneruje PIN roven čtyřem nulám
 - Tímto trikem lze získat EPB obsahující PIN 0000
- Útok využívající známého zašifrovaného PINu 0000
 - Další manipulací s DT dokáže útočník zjistit číslice zákaznickova PINu (ale ne jejich pořadí)
 - Prohledávaný prostor PINů tak omezit z 10 000 na nejvýše 36
- Útok bez známého zašifrovaného PINu
 - Opět manipulací s DT zjistí číslice zákaznickova PINu
 - Jejich pořadí pak dokáže určit manipulací s offsety
 - K určení pozic všech číslic čtyřmístného PINu stačí 6 volání verifikační funkce

Nevynucení politiky u PKCS #11



- Doposud jsme mluvili pouze o útocích na API navržené přímo pro konkrétní HSM
 - Například IBM CCA API
- Nyní se zaměříme na PKCS #11 API
 - Navrženo pouze jako standardní rozhraní mezi aplikacemi a jedinouživatelskými bezpečnostními zařízeními
- Hlavní problém tohoto API
 - Jedná se pouze o sadu funkcí bez jakékoliv politiky
 - Ta by například zajistila konzistentnost vlastností klíčů



Symmetric Key Attacks (I)

- 3DES Key Binding Attack
 - 3DES-2 klíč K a jeho jednotlivé poloviny K_1 a K_2
 - Při exportu $E_{KEK}(K) = (E_{KEK}(K_1), E_{KEK}(K_2))$
- Key Separation Attack
 - Konfliktní nastavení vlastností klíčů
 - Například klíč určený k šifrování klíčů a dešifrování dat
- Weaker Key/Algorithm Attack
 - Šifrování klíčů slabými algoritmy jako RC2 či DES
 - Degradována bezpečnost silných algoritmů



Symmetric Key Attacks (II)

- Reduced Key Space Attack
 - Jedna z funkcí PKCS #11 umožňuje vytvořit klíč z části po sobě jdoucích bitů již existujícího klíče
 - Toho lze využít ke zmenšení prohledávaného prostoru klíčů
 - Útočník například nejprve použitím čtyřiceti po sobě jdoucích bitů z 56bitového DES klíče vytvoří 40bitový RC2 klíč
 - Ten hrubou silou dešifruje => s jeho pomocí najde zbylých 16 bitů původního DES klíče
- Existují také útoky, které se opírají o podporu API pro kryptografické operace s veřejným klíčem
- Množství relativně snadných útoků na PKCS #11 API
 - Zcela nevhodné jako hlavní API pro víceuživatelské HSM

Závěr 😊



- Bezpečnost současné generace bankovních API je s ohledem na **vnitřní útočníky** skutečně špatná
- Parametry funkcí mohou být libovolně měněny – jejich **kontrola je nedostačující**
- Mnoho implementovaných standardů zaručuje **kompatibilitu**, ale také způsobuje **kritické chyby**
- Důkazem je právě množství nově objevených útoků
 - Tyto útoky neznamenaají, že by HSM byly k ničemu
 - Krátkodobě je však nutno posílit administrativní kontrolu přístupu k modulům
 - Dlouhodobě se musí odstranit přímo chyby způsobující zranitelnost modulů