BEZPEČNOST RSA (4)

RSA v novém světle

Reálná hrozba Mangerova útoku na některé implementace algoritmu RSA volá po jejich revizi. Minule jsme naznačili teoretické východisko možné obrany, nyní si ji ukážeme prakticky přímo na napadeném standardu PKCS#1.

Navržený postup obrany má ochránit implementace RSA proti potenciálním útokům využívajícím postranních kanálů. Jde o originální teoretický výsledek, který má jednoduché a praktické využití. Zopakujme si stručně jeho princip: Proti postranním kanálům využíváme maskování funkcí tak, že do nich zavedeme nový umělý parametr, který funkci významově neovlivňuje. Tím, že tomuto parametru přiřazujeme náhodné hodnoty, „zastíníme" skutečné hodnoty významových parametrů a „zašumíme" tak postranní kanál. Útočníkovi, který by jej chtěl využít, se pak dění na postranním kanálu jeví podobně jako náhodný proces.

Tento princip aplikujeme i na návratové kódy z implementačních procedur a podprocedur. Běžně se vracejí hodnoty 0 nebo 1 (resp. -1), podle toho, zda v proceduře došlo k chybě. Pokud ovšem chyba nastane v procedurách přímo zpracovávajících citlivé informace (např. otevřený text po odšifrování), není vhodné, aby byť i jen chybovým hlášením útočníkovi přes postranní kanál prozrazovaly, co se dělo. Zejména u RSA lze totiž vhodnými „triky" se šifrovým textem docílit záměrných změn v odšifrovaném textu. Tím se dostáváme „pod kůži" té privátní operaci odšifrování, na kterou jinak nemáme nic, protože neznáme privátní klíč.

Jak jsme již také ukázali, jednoduchá chybová informace o tom, zda výsledek odšifrování obsahuje na nejvyšším místě nulu nebo ne, vede k odhalení celého otevřeného textu. (Ostatně je to jen důsledek obecného tvrzení o individuálních bitech RSA, viz [2].) Právě proto doporučujeme maskovat návratové kódy – místo obvyklého chybového kódu 1 (-1) vracíme náhodné nenulové číslo, které označujeme RNZ *(random nonzero)*.

Další doporučení shrnujeme v následujících všeobecných zásadách, které dále demonstrujeme přímo na standardu PKCS#1. Pokud „revizoři" kódů realizujících RSA podle PKCS#1 budou tyto zásady dodržovat, zařadí se rozhodně mezi bezpečnostní smetánku. Po zkušenostech z praxe můžeme totiž takřka s jistotou prohlásit, že většina implementací zůstane v současném stavu a žádné opravy se dělat nebudou, dokud nedojde k nějakému „průšvihu"...

Kódy, které realizují RSA podle PKCS#1, určitě zaslouží revizi. Několik vhodných zásad zde navrhujeme.

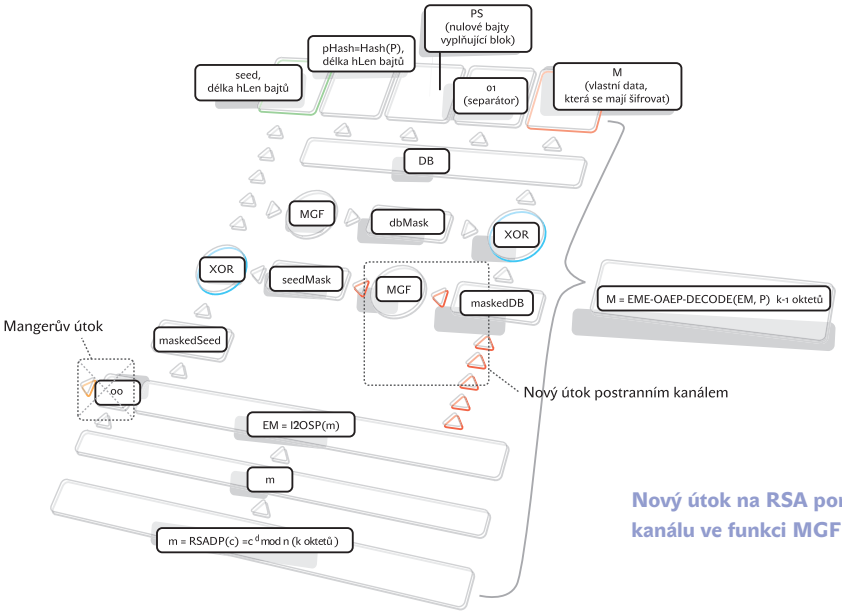
Připomeňme ještě, že z hlediska Mangerova útoku na PKCS#1 má klíčový význam procedura RSAES-OAEP-DECRYPT, která se volá k odšifrování šifrového textu *c*. Ta pak volá další procedury I2OSP a EME-OAEP-DECODE, viz první díl. Pro naše obecná pravidla lze všechny tyto procedury považovat za dílčí. Uvažujme nyní obecně systém, který se skládá z posloupnosti několika dílčích výpočtů (procedur) *comp_1(...), comp_2(...), ..., comp_n(...)* a formulujme naše doporučení.

MINIMÁLNÍ OBECNÁ PRAVIDLA

1. Nepoužívejte příkazy *stop* nebo *break*. Zdá se to samozřejmé, ale formálně jsou tyto příkazy v PKCS#1 použity a měly by být vyloučeny.
2. Abyste zabránili časovému útoku, posloupnost výpočtů *comp_1(...), comp_2(...),...* by se měla navenek jevit „spojitá" a „stejná" i pro různé datové vstupy. Z teoretického hlediska je to sice nemožné, ale v praxi byste se měli tomuto principu co nejvíce přiblížit, ať už tvoříte nějaké fyzické zařízení nebo píšete program.
3. Každá dílčí procedura (*comp_X*) by měla vždy vracet **standardní výstupy**, např. chybový kód (*error_X*), ukazatel na výstupní data (*pointer_X*) a délku těchto dat (*length_X*). Tyto proměnné by měly být pro různé datové vstupy vždy počítány pokud možno **stejným procesem** (byť to není vždy triviální úloha).
4. Dílčí návratové kódy by měly být nulové, jestliže vše proběhlo v pořádku, a **náhodné nenulové** (RNZ), pokud nastala nějaká chyba. Přitom je důležité zvážit, kam zařadit volání procedury pro tvorbu RNZ, aby její používání samo o sobě nevětvilo dílčí proceduru. V některých případech, například v jednoduchých smart kartách, může být získání náhodné hodnoty obtížné. Pak ji můžeme odvodit přímo z nějakých částí dešifrované zprávy *m* – avšak velmi obezřetně, abychom tak nevytvořili další postranní kanál.
5. V každém dílčím výpočtu (kromě posledního) lze využít výstupy (*pointer_X* a *length_X* pro různá X) z předchozích dílčích výpočtů, ale raději ne předchozí návratové chybové kódy *error_X*. Pravděpodobně by to totiž způsobilo větvení čitelné na postranních kanálech. Tyto návratové kódy se využijí až v závěrečném vyhodnocení celého výpočtu.
6. Po ukončení všech dílčích výpočtů vypočítáme závěrečný chybový kód jako logický OR všech dílčích chybových kódů (*final_error* = *error_1* OR *error_2* OR ... OR *error_n*) a podle něj teprve rozhodneme, zda výstupní data z posledního výpočtu jsou platná. Zlepšíme tak plynulost výpočtu a zabráníme datově závislému větvení.
7. Návratové RNZ by měly mít co největší rozsah, např. 8- až 32bitová slova (čím „širší", tím lepší je náhodné maskování). Např. při osmibitovém RNZ náhodně vybíráme jeden z 255 kanálů.
8. S citlivými daty (kam patří zejména všechny bity otevřeného textu, mezivýsledky procedur *comp_X* a chybové kódy) provádíme co nejméně přesunů. Každý přesun totiž na postranních kanálech (hlavně napěťově-proudových) vyzařuje informaci o přenášených datech. Proto je vhodné vyhradit pro tato data paměťové oblasti, kam mají následující výpočetní procesy přístup. Je-li to nezbytně nutné, pak přesouvat naráz co nejdelší bitové řetězce, což je efektivnější a pomáhá maskovat vyzařovanou informaci (viz minulý díl).

PŘÍKLAD

Vraťme se k Mangerovu útoku. Jako aplikaci uvedených principů můžeme z procedury I2OSP vracet (místo hlášení *Číslo je příliš veliké*) chybový kód rovný přímo nejvyššímu bajtu (X) vstupního čísla, tj. *error_I2OSP* = X (když X nebude nulový, nastala chyba, což tedy *error_I2OSP* korektně odráží; totéž v případě nuly, kdy chyba nenastala). Tuto hodnotu však (jak jsme doporučili) nikam nepřesouváme, nejlépe je zanechat ji po



Nový útok na RSA pomocí postranního kanálu ve funkci MGF

odšifrování v nějakém registru, odkud ji později použijeme pro operaci závěrečného vyhodnocení platnosti odšifrovaných dat.

Nastavení *error_I2OSP* = X zajišťuje kontinuální výpočet bez větvení i jistou míru náhodnosti. Pak je ale nutné pokaždé předat zbytek vstupního čísla k dalšímu zpracování procedurou EME-OAEP-DECODE a nedat přitom jakkoli najevo, zda jde o smysluplná data nebo ne (v případě, že X není nula, došlo k náhodné chybě na kanálu nebo k útoku). Stav zjistíme z jejího návratového kódu *error_DECODE*, neboť k detekci platnosti předložených dat má silné nástroje (viz bod 9 této procedury dále). Chybový kód celé procedury odšifrování RSAES-OAEP-DECRYPT se vypočítá jako *final_error* = *error_I2OSP* OR *error_DECODE* a až podle něj se rozhodne, zda výsledná data jsou platná.

Proč náhodné chybové kódy

Výhody RNZ v roli chybových kódů si ukážeme na příkladu PKCS#1, v němž bychom pro *error_I2OSP* a *error_DECODE* použili jen hodnoty 0 a 1. Nechť x_{k-1} je nejvýznamnější bajt odšifrovaného celého čísla *m* (m = c^d mod n). Je-li x_{k-1} nenulové, šifrový text je špatný, takže *error_I2OSP* = 1; při nulovém x_{k-1} bude *error_I2OSP* = 0. Předpokládejme, že útočník, jenž může volit šifrové texty, například využitím napěťově-proudové analýzy, dokáže studovat chování systému v době, kdy je počítána hodnota *final_error* = *error_I2OSP* OR *error_DECODE*. Mohou nastat pouze tyto případy:
(0 OR 0) – správný šifrový text;
(1 OR 1) – špatný šifrový text a x_{k-1} není nula;
(0 OR 1) – špatný šifrový text a x_{k-1} je náhodně nula;
(1 OR 0) – v běžném případě skoro vyloučenonemožná situace, je-li integritní kontrola *m* v pořádku, ale bajt x_{k-1} je špatně.

Jestliže útočník posílá správný šifrový text, poznává chování systému v prvním případě (0 OR 0). Když posílá náhodný špatný šifrový text, poučí se o chování systému v druhém (1 OR 1) nebo ve třetím případě (0 OR 1), přičemž pravděpodobnosti těchto jevů jsou cca 255/256 a 1/256, takže je snadno odliší.. Po této „fázi učení" bude tak útočník schopen jasně rozlišit případy, kdy *error_final* reprezentuje stavy (**1** OR 1) nebo (**0** OR 1), tedy zjistit *error_I2OSP* v případě, že odesílá falešný šifrový text. To je však přesně táž informace jako původní chybové hlášení *Číslo je příliš veliké*. Mangerův útok tu máme znovu, tentokrát napěťově-proudovou analýzou.

Jak vidíme, musíme se obecně vyhnout situacím, kdy citlivá proměnná X vstupuje do N-árních operací, v nichž zbývajících N-1 operandů je známo útočníkovi. Zejména to platí pro případ analýzy spotřeby energie. Například musíme vyloučit operace jako *X + const* nebo *if (X != 0) then* atd. Proto doporučujeme, aby nejen *error_DECODE*, ale **všechny** chybové kódy používaly RNZ, pokud možno připravené už před voláním odpovídajících procedur (*comp_X*).

Konkrétní doporučení pro realizaci vybraných dílčích procedur PKCS#1 [3] vidíte na následujících stránkách ve formě komentářů (K:) tučným písmem v originálním textu.

Shrnutí

V závěrečném dílu miniseriálu věnovaného problematice správného používání schématu RSA, kterou po čase znovu otevřel tzv. Mangerův útok, jsme doporučili konkrétní zásady pro implementaci tohoto schématu. Při jejich dodržení lze očekávat snížení rizika zmíněného útoku na minimum. Otázka správného používání RSA však jde daleko za rámec konkrétních formátovacích metod – v práci [2] je ukázáno, že náchylnost k takovýmto druhům útoků vychází přímo ze základních vlastností RSA. Formátovací metody mohou tuto náchylnost více či méně skrýt, ale nikdy ne zcela vyloučit – to může dokázat teprve precizní implementace, která bude brát zvýšený ohled na riziko postranních kanálů.

Ale Mangerův útok zdaleka nemusí být jediný – v [2] jsme poukázali na jiný možný způsob napadení RSAES-OAEP, který zde kvůli omezenému rozsahu článku uvádíme pouze jako ilustrační obrázek. Je z něj zřejmé, že útočíme přes zcela jinou část otevřeného textu než v případě Mangerova útoku. Jak plyne z [2], máme právo se domnívat, že budou existovat implementace, které informaci o této části otevřeného textu poskytnou a umožní tak na RSA zaútočit s cílem získat celou přenášenou informaci.

Čím je šifra RSA jednodušší co do svého základního popisu, tím složitější je její bezpečná implementace.

Popisem nového útoku nechceme naznačit, že by se šifra RSA neměla používat. Důrazně však upozorňujeme, že jakkoli jsou účinná protiopatření proti Mangerovu útoku na místě, bylo by velmi krátkozraké se domnívat, že tím jsou všechny problémy s postranními kanály vyřešeny. Je to vlastně paradoxní – jak je RSA co do svého základního popisu jednoduchá, tak je nakonec její bezpečná implementace složitá. ■ ■ ■

Vlastimil Klíma, *autor@chip.cz* | Tomáš Rosa, *autor@chip.cz*

Literatura:

[1] Archiv článků:
http://www.decros.cz/bezpecnost/_kryptografie.html
[2] Nový útok na RSA postranním kanálem a jiné souvislosti:
ftp://ftp.decros.cz/pub/Archiv/Publications/2001KlimaRosa_kryptobesidka_2001_CZ.pdf,
ftp://ftp.decros.cz/pub/Archiv/Publications/2001KlimaRosa_kryptobesidka_2001_EN.ppt
[3] PKCS#1 ver. 2.1 draft 2, nové a připravované úpravy:
http://www.rsalabs.com/pkcs/pkcs-1/index.html

RSAES-OAEP-DECRYPT(*K, C, P*)

Vstup:

K příjemcův privátní klíč

C přijatý šifrový text k odšifrování, řetězec *k* oktetů, kde *k* je délka modulu RSA *n* v oktetech

P parametr pro OAEP, řetězec, který může být prázdný

Výstup:

M otevřený text (zpráva pro zašifrování), řetězec o délce nejvýše *k*–2–2*hLen*, kde *hLen* je délka výstupu hašovací funkce použité v EME-OAEP

Chyby:

„dešifrovací chyba" **K: Obecnou zásadou je, že každá procedura (X) vrací chybovou hodnotu *error_X* ∈ <0, 255> a eventuálně pointer na výstupní data *pointer_X* a jejich délku *length_X*. Je-li *error_X* nula, je vše v pořádku, jinak nastala nějaká chyba a vracená data jsou chápána jako neplatná.**

Postup:

1. Není-li délka šifrového textu *C k* oktetů, vrať *„dešifrovací chyba"* a skonči. **K: Tento výstup neříká útočníkovi nic nového, neboť zná šifrový text a modul RSA. Pro čistotu programování a kvůli obecné zásadě „žádné přerušení, spojité zpracování" je lepší tuto kontrolu přesunout jinam, například před volání procedury RSAES-OAEP-DECRYPT.**
2. Konvertuj řetězec *C* na celé číslo *c* pomocí procedury OS2IP, *c* = OS2IP(*C*).
3. Použij dešifrovací transformaci RSADP s privátním klíčem *K* na šifrový text *c*, vyjde číslo *m* = RSADP(*K*, *c*) = $c^d \bmod n$. Je-li výstupem z transformace RSADP chyba typu *„číselná reprezentace šifrového textu mimo rozsah"* (což by nastalo, kdyby c ≥ n), pak vrať *„dešifrovací chyba"* a skonči. **K: Kontrolu *c* ≥ *n* je vhodné zařadit před proceduru RSADP nebo RSAES-OAEP-DECRYPT, protože *c* i *n* známe předem.**
4. Konvertuj číselnou reprezentaci zprávy *m* na oktetový řetězec *EM* o délce *k–1* oktetů pomocí transformace I2OSP: *EM* = I2OSP(*m*, *k–1*). Je-li jejím výstupem *„číslo je příliš velké"*, pak vrať *„dešifrovací chyba"* a skonči. **K: Toto je klíčový bod Mangerova útoku. Nesmí zde být žádné přerušení. Opravená procedura I2OSP nevydává hlášení *„číslo je příliš velké"* , ale vrací *error_I2OSP* a ukazatel na data *pointer_I2OSP*. Data předáme do dalšího kroku a návratový kód *error_I2OSP* zpracujeme později, i kdyby indikoval nesmyslná data.**
5. Na *EM* s využitím parametru *P* použij dekódovací transformaci EME-OAEP-DECODE k rekonstrukci zprávy *M* = EME-OAEP-DECODE(*EM*, *P*). Je-li výstupem dekódovací transformace *„dekódovací chyba"*, vrať *„dešifrovací chyba"* a skonči. **K: Také procedura EME-OAEP-DECODE nově vrací *error_DECODE*, *pointer_DECODE* a *length_DECODE* a není tu ani přerušení, ani chybová zpráva.**
6. Výstup je *M*. **K: Teprve nyní definujeme celkový chybový kód *error_DECRYPT* = *error_I2OSP* OR *error_DECODE*, ukazatel na výstupní data *pointer_DECRYPT* = *pointer_DECODE* a délku dat *length_DECRYPT* = *length_DECODE*. Je-li výsledný chybový kód 0, *M* jsou platná data, je-li nenulový, v dílčích procedurách nastala chyba, a proto *M* považujeme za neplatné.**

I2OSP(*x, l*)

Vstup:

x nezáporné celé číslo, které má být konvertováno. **K: Vstupem je vždy celé číslo o *k* bajtech, kde *k* je globální proměnná, známá před voláním I2OSP.**

l délka výsledného řetězce **K: V kontextu PKCS#1 je *l* vždy rovné *k*-1.**

Výstup:

X odpovídající řetězec *l* oktetů

Chyby:

„celé číslo je příliš velké" **K: Takové chybové hlášení tu nesmí v žádném případě být, neboť je podstatou Mangerova útoku. Místo toho definujeme návratový kód podle obecných doporučení a data vždy necháme zpracovat, i kdyby byla nesmyslná.**

Postup:

1. Je-li $x \geq 256^l$, vrať *„celé číslo je příliš velké"* a skonči. **K: Tento krok vynecháme, neboť vytváří postranní kanál.**
2. Zapiš celé číslo *x* jedinečným způsobem jako *l*-ciferné číslo v bázi 256: $x = x_{l-1}\,256^{l-1} + x_{l-2}\,256^{l-2} + \ldots + x_1\,256 + x_0$, kde $0 \leq x_i < 256$ (jedna nebo více vedoucích cifer může být nula, pokud $x < 256^{l-1}$). **K: V kontextu PKCS#1 voláme tuto proceduru tak, že jejím vstupem je *k*-bajtové číslo a výstupem (*k*-1)-bajtový (oktetový) řetězec. Dále využijeme toho, že pokud je nejlevější bajt vstupu nenulový, nastala chyba. Vstupní číslo proto vyjádříme ve tvaru $x = x_{k-1}\,256^{k-1} + x_{k-2}\,256^{k-2} + \ldots + x_1\,256 + x_0$ a přímo nastavíme *error_I2OSP* = x_{k-1} a *pointer_I2OSP* tak, aby ukazoval na x_{k-2}. Pokud je bajt x_{k-1} nenulový, definuje (do jisté míry náhodný) nenulový chybový kód, což odpovídá tomu, že někde nastala chyba. Je-li x_{k-1} (a tedy i návratový kód) nulový, korektně indikuje správný průběh I2OSP.**
3. Nechť oktet X_i má číselnou hodnotu x_{l-i} pro $1 \leq i \leq l$. Potom výstupem je řetězec $X = X_1\,X_2 \ldots X_l$. **K: Délka vracených dat je vždy *k*-1 a pointer na ně je také znám předem; nejdůležitější je tedy návratový kód *error_I2OSP*, viz předchozí krok.**

EME-OAEP-DECODE(*EM, P*)

Volby:

Hash hašovací funkce (*hLen* označuje délku jejího výstupu v oktetech)

MGF funkce generující masku

Vstup:

EM zakódovaná zpráva, oktetový řetězec délky nejméně 2**hLen* + 1, její délku v oktetech označme *emLen*. **K: V kontextu PKCS#1 je emLen vždy rovna *k*-1.**

P kódovací parametr, oktetový řetězec. **K: Ve skutečnosti nepotřebujeme parameter P, ale pHash = Hash(P). Funkci Hash i parametr P známe předem, a proto pHash kvůli vyloučení časového postranního kanálu pro Mangerův útok počítáme raději před touto procedurou.**

Výstup:

rekonstruovaná (odkódovaná) zpráva, oktetový řetězec délky nejvýše *emLen*-1-2**hLen*

Chyby:

dekódovací chyba". **K: „*dekódovací chyba*" bude nahrazena standardními výstupy *error_DECODE*, *pointer_DECODE* a *length_DECODE*.**

Postup:

1. Je-li délka *P* větší než vstupní limit hašovací funkce (2^{61}–1 oktetů pro SHA-1), pak vrať „*dekódovací chyba*" a skonči. **K: Tuto kontrolu je vhodné (jako opatření proti časovému postrannímu kanálu) přesunout před proceduru RSADP nebo RSAES-OAEP-DECRYPT. Je to možné, protože před voláním EME-OAEP-DECODE i RSAES-OAEP-DECRYPT známe hašovací funkci i délku P.**
2. Je-li *emLen* < *2hLen*+1, vrať „*dekódovací chyba*" a skonči. **K: Stejný jako v předchozím kroku.**
3. Nechť *maskedSeed* je prvních *hLen* oktetů *EM* a nechť *maskedDB* je zbývajících *emLen-hLen* oktetů.
4. Nechť *seedMask* = *MGF*(*maskedDB*, *hLen*).
5. Nechť *seed* = *maskedSeed* ⊕ *seedMask*.
6. Nechť *dbMask* = *MGF*(*seed*, *emLen-hLen*).
7. Nechť *DB* = *maskedDB* ⊕ *dbMask*.
8. Nechť *pHash* = *Hash*(*P*), řetězec *hLen* oktetů.
9. Rozděl *DB* na řetězec *pHash´* , který se skládá z prvních *hLen* oktetů *DB*, dále na (eventuálně prázdný) řetězec *PS* obsahující nulové oktety, následující za *pHash´*, a zprávu *M* tak, že *DB* = *pHash´* || *PS* || 01 || *M* . Jestliže v *DB* není žádný oktet 01, který odděluje *PS* od *M*, vrať „*dekódovací chyba*" a skonči. **K: Zpráva „dekódovací chyba" bude opět nahrazena standardními *error_DECODE*, *pointer_DECODE* a *length_DECODE*, nebude zde ani žádný příkaz "stop". Při ochraně proti postranním kanálům je také vhodnější sem zároveň přesunout i kontrolu z bodu 10. Nově tedy, když v DB se pHash' nerovná pHash nebo zde není obsažen žádný oktet 01, oddělující PS od M, nebo když PS neobsahuje samé nuly, nastavíme *error_DECODE* na RNZ, jinak na 0. Hodnoty *pointer_DECODE* a *length_DECODE* musí být nastaveny tak, aby předcházely časovému útoku, jak bylo popsáno v obecné části výše.**
10. Je-li *pHash*' různá od *pHash*, vrať „*dekódovací chyba*" a skonči. **K: Spojit s bodem 9.**
11. Výstup je *M*. **K: Výstupem je nyní *error_DECODE*, *pointer_DECODE* a *length_DECODE*.**