Bezpečnost RSA (1)

RSA v novém světle

Nechval dne před večerem – a platí to i v kryptografii. Šifra RSA, která svými teoretickými bezpečnostními vlastnostmi dokázala ukolébat řadu expertů, nedávno utrpěla nikoli bezvýznamný šrám týkající se její implementace...

Je tomu právě rok, co jsme v Chipu 11/00 vyjadřovali uspokojení nad tím, že nová verze formátu RSA pro šifrování (PKCS#1, ver. 2.0) zabrání útoku, který byl úspěšně ověřen na protokolu SSL používajícím formát PKCS#1, ver. 1.5, kde se pomocí RSA šifrují symetrické klíče. Nešlo o maličkost, vždyť tento formát využívá většina internetového bankovnictví...

Letos v srpnu na konferenci Crypto'2001 James Manger ukázal, že jásot byl předčasný. Nově navrhovaný formát (verze 2.0 a 2.1), který ani nestačil proniknout do masového použití (ba ani do zmiňovaného e-bankovnictví), obsahuje chybu. Jaké jsou důsledky tohoto objevu a co to konkrétně znamená pro bezpečnost, si řekneme v právě začínajícím miniseriálu.

JE RSA V POŘÁDKU?

V minulém Chipu jsme se věnovali problému faktorizace RSA a loučili jsme se s ulehčujícím a perspektivním závěrem (nejen pro mnohé banky), že faktorizace 1024bitového modulu RSA nám v nejbližší době nehrozí. RSA je v pořádku – proč bychom se tedy měli znepokojovat?

Jenže ono nezáleží jen na šifře samé, ale také na způsobu jejího použití, a dokonce, jak vyplynulo i z článků o postranních kanálech (viz infotipy), také na způsobu její realizace. Aby byl celý systém bezpečný, musí být použita **kvalitní šifra** (to snad už máme zaručeno – neradi bychom po roce zase psali, že ne...), musí být použita **správným způsobem** (o tom budeme právě hovořit) a správně to musí být také **realizováno** („zadrátováno" nebo naprogramováno – tady je to pro tvůrce a bezpečnostní obranáře hodně složité). Do všech těchto oblastí dnes také hovoří kryptologové snažící se o důkazy bezpečnosti, které implementátoři požadují.

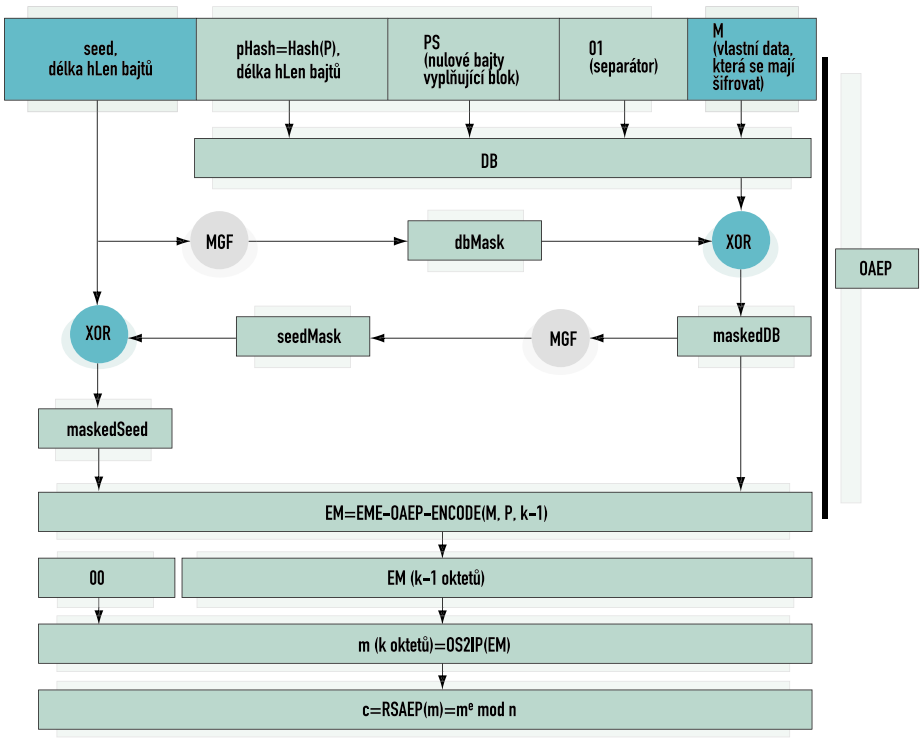
V roce 1998 Bleichenbacher (viz infotipy) ukázal, jak zaútočit na dosud nejrozšířenější formát PKCS#1 verze 1.5 pro šifrování symetrických klíčů. Proto byl na bázi metody OAEP *(Optimal Asymmetric Encryption Padding)* navržen formát PKCS#1 verze 2.1, který se zdál být bezpečný. Tím měla být nepříjemnost zažehnána a radovali jsme se, že konečně máme kvalitní formát dat. Jenže ouha, ukázalo se, že, jak uvidíme dále, přidáním jednoho nenápadného nulového bajtu k OAEP se toho dost pokazilo – Mangerův útok je tak průzračný a jednoduše realizovatelný, až z toho jde mráz po zádech (na 1024bitovém modulu RSA asi tisíckrát efektivnější než útok Bleichenbacherův...).

Při precizním pohledu naštěstí můžeme konstatovat, že pokud by byl současný formát verze 2.1 použit striktně tak, jak je napsán, nově popsaný Mangerův útok by nefungoval. Jak jsme však řekli, je tu ještě realizační hledisko – a Manger poukazuje na spoustu míst, kde se dá udělat chyba v implementaci. Tolik z ryze praktického hlediska.

Pokrok však nastal i na poli čistě teoretickém a bylo ukázáno, že metoda OAEP není ani po této stránce to „pravé ořechové". O všem si povíme podrobněji. Nejprve si však připomeneme uvedený formát, potom se podíváme, jak do něj byla metoda OAEP nevhodně začleněna, jaký to umožnilo útok a jaká možná řešení a bezpečnostní opatření se navrhují.

FORMÁT DAT EME-OAEP DLE PKCS#1 V. 2.1

Norma PKCS#1 verze 2.1 se týká jak formátu dat pro šifrování, tak pro digitální podpis. Zde se teď budeme věnovat jen části zašifrování a odšifrování dat této normy a příslušnému formátu budeme →



Kódování zprávy M pomocí metody OAEP a její následné zašifrování RSA: Přidání nulového bajtu k výsledku kódování OAEP vnáší potenciální chybu – při nevhodné implementaci to pak může vést k vyzařování informace o M, vedoucí až k získání celé zprávy!

zkráceně říkat *formát* PKCS#1. Pod daty si představte zejména náhodné symetrické klíče, tak jak je používá např. protokol SSL. Nebudeme se zabývat formátem pro digitální podpis, protože v něm tak zásadní chyby nebyly nalezeny ani v předchozí verzi 1.5. této normy.

Nejprve si zaveďme označení: *M* je zpráva, kterou šifrujeme, *n* je modul RSA, *e* je veřejný exponent, *k* je počet bajtů modulu. RSAES-OAEP je označení pro šifrovací schéma, které kombinuje šifrovací transformaci RSAEP a odšifrovací transformaci RSADP s metodou kódování EME-OAEP, založenou na [BERO94]. Metoda kódování používá maskování zprávy, přesněji funkci generující masku (MGF) ve dvou fázích (tzv. Feistelových rundách) a MGF využívá vhodnou bezpečnou hašovací funkci (například SHA-1). Vstup SHA-1 může být libovolný řetězec, výstupem je haš o délce *hLen* bajtů (norma používá pojem *oktet*, tj. osm bitů).

Přesnou definici MGF naleznete v článku z Chipu 11/00 (viz infotipy), její využití v kódování EME-OAEP znázorňuje připojené schéma. Jak vidíte, zpráva je doplněna *separátorem* (bajt 0x01), poté 20 bajty hodnoty *pHash(P)* – přičemž P je implicitně definován jako konstantní (prázdný) řetězec, takže jeho *pHash* je vlastně také konstanta – a poté doplněna *hLen* náhodnými bajty *seed* (protože hašovací funkcí je SHA-1, je *hLen* = 20). Náhodný seed pak funkce generující masku MGF „roztáhne“ na potřebný počet náhodně vyhlížejících bajtů, kterými tak maskuje zbytek celého bloku. A naopak – nyní obdržený znáhodněný výsledek zase zpětně maskuje začátek. Tato dvourundová mixáž naruší původní strukturu dílčích bloků vstupu, včetně zprávy *M*.

PLAINTEXT-AWARENESS

Zdá se tedy, že výsledek *EM* dokonale maskuje *M*. To je důležité, protože je známo, že u RSA lze využít její multiplikativní vlastnosti k podsouvání sofistikovaně generovaných šifrových textů. Násobení ale nejde příliš dohromady s bitovým maskováním (navíc dvourundovním), takže by mělo být možné prokázat, že RSA kombinované s OAEP (což je právě popsané kódování) je tzv. *plaintext-aware encryption scheme,* tj. šifrovací schéma, u něhož nemůžeme vytvořit platný šifrový text, aniž bychom předtím znali odpovídající otevřený text.

Obyčejná RSA tuto vlastnost nemá, jak dokládá následující příklad. Pokud zachytíme nějaký šifrový text *c*, postačí vytvořit šifrový text $c' = (c * 2^e) \bmod n$ a víme, že je platný – po jeho odšifrování nutně vznikne otevřený text, který je „dvojnásobkem“ původního otevřeného textu! Tuto úvahu snadno dokážeme odšifrováním *c'* takto: $c'^d \bmod n = (c * 2^e)^d \bmod n = c^d * 2^{ed} \bmod n = m * 2 \bmod n$.

Trochu to připomíná dětský „kouzelnický“ trik „Mysli si číslo, vynásob je dvěma, přičti 10, vyděl dvěma, odečti číslo, které jsi si myslel, a vyjde ti 5“. (Princip této magie snad není třeba vysvětlovat.) A, stejně jako zmíněný kouzelník, umíme díky vlastnostem RSA někomu podstrčit určitý šifrový text, o jehož odšifrovaném tvaru *přece jen něco víme*, aniž bychom ho znali celý! O šifrovém textu *c'* můžeme totiž s jistotou tvrdit, že po jeho odšifrování vyjde dvojnásobek původního nám neznámého vstupu do RSA (vše je v modulu *n*). Vidíme tedy, že obyčejné schéma RSA bez vhodného maskování zprávy není *plaintext-aware,* a využil toho jak Bleichenbacher v roce 1998, tak Manger před třemi měsíci.

CO ZAVINÍ NULA NAVÍC

Jak je ale možné, že se útok vydařil, když se RSA-OAEP účinně brání (je tzv. CCA2 odolné, jak si řekneme příště) právě popsaným kouzelnickým trikům složitým maskováním a zabraňuje také Bleichenbacherovu útoku z roku 1998 (viz infotipy)? Do implementace se totiž vloudila chyba – po OAEP se k výsledku ještě přidává na nejvýznamnější místo jeden nulový oktet. Proč? Z toho prostého důvodu, že číslo *m*, které jde do šifrovací transformace $c = m^e \bmod n$ (RSAEP), musí být menší než modul *n*.

Pokud by byl současný formát verze 2.1 použit striktně tak, jak je napsán, nový Mangerův útok by nefungoval.

Kódování OAEP se tedy dělá jen na výslednou délku *k-1* oktetů, aby se mohla tato nula přidat a výsledek byl *k*-oktetový, tedy v délce modulu. Tím, že takto vytvořené číslo *m* má *k*-tý bajt nulový, zatímco modul *n* ho má nenulový, je triviálně splněna podmínka $m < n$, a proto se může později regulérně odšifrovat transformací RSADP ($m = c^d \bmod n$).

Takto definované schéma už ovšem není čisté OAEP, ale jakýsi „slepenec“ z OAEP, přidané nuly a šifrovací funkce RSAEP. Odšifrovací funkce RSADP, která na takto vytvořených šifrových textech pracuje, musí pak po odšifrování nutně dostat číslo (*m*), které má v nejvyšším bajtu nulu! Původní vlastnost OAEP, která vše rozmixovala na náhodná čísla, se tady vytrácí, i když jen na jediném bajtu – a Mangerův útok toho mistrně využil.

MANGERŮV ÚTOK

Mangerův útok (viz infotipy) se týká procedury odšifrování – její postup vidíte v rámečku na str. 175. →

Přímo v textu normy je uvedena tato závažná poznámka, která jistě o mnohém svědčí:

„Je důležité, aby chyby v krocích 4 a 5 byly nerozlišitelné, jinak útočník může být z typu chyby schopen získat užitečnou informaci. Zejména v krocích 4 a 5 musí být identické. Navíc čas provedení dešifrovací procedury nesmí odhalit, zda se chyba vyskytla. Jedna z možností, jak toho dosáhnout (stejného času, pozn. autorů)*, je následující: V případě chyby v kroku 4 postup do kroku 5 s EM nastavenou na řetězec nulových oktetů. Chybová zpráva byla použita v útoku s volbou šifrového textu na zprávy, zašifrované podle PKCS #1 v. 1.5 , viz [BLEI98].“*

Tvůrci normy si tedy jisté slabiny byli vědomi. Manger ale založil svůj útok na předpokladu, že u mnoha implementací této normy je možné rozlišit, zda se při odšifrování výpočet dostal až na konec procedury RSAES-OAEP-DECRYPT(*K, C, P*), nebo jen do jejího kroku 4, kde vznikla chyba vrácená z funkce I2OSP „*celé číslo je příliš velké*“, tj. že zpráva *m* neměla vedoucí bajt nulový. Argumenty, že to zjistit lze (povšimneme si jich příště), jsou natolik přesvědčivé, že společnost RSA bude na Mangerův útok reagovat vydáním přepracovaného návrhu normy...

Ale pojďme už k podstatě útoku. Předpokládejme, že jsme v protokolu SSL mezi stanicí uživatele a serverem zachytili šifrový text *c* obsahující v *M* symetrický šifrovací klíč na dané sezení. Pokud je možné odlišit typ chyby, jak jsme popsali výše, můžeme server bombardovat různými šifrovými texty c_1, c_2, c_3 ... (budou to vhodné modifikace *c*) a dostaneme od něj skoro vždy odezvu „*dešifrovací chyba*“. Z dalších projevů serveru dále obdržíme zmíněnou informaci (o její podstatě hodně napovídá seriál o postranních kanálech) a zjistíme, zda příčinou chyby byla nepřítomnost nuly v nejvyšším bajtu čísla po odšifrování, nebo zda chyba vznikla až ve fázi kontroly v bodě 9 a 10 dekódování EME-OAEP-DECODE(*EM, P*).

Jinými slovy: server nám k danému šifrovému textu c_i vždy prozradí, zda po odšifrování je, nebo není u čísla $m_i = c_i^d \bmod n$ jeho nejvyšší bajt nulový. Takový „orákulus“ Mangerovi stačí, aby u 1024bitového modulu RSA po cca 1100 volbách šifrových textů c_i už zjistil tajnou informaci *M*! Konkrétní postup si ukážeme příště.

CO SI O TOM MYSLET?

Mangerův útok je v první řadě veden na PKCS#1. Pokud se autoři současného návrhu této normy (verze 2.1, draft 2) uchýlí k alibismu, potom spor, zda jejich norma je, nebo není Mangerovým útokem dotčena, patrně vyhrají. V normě je totiž jasně řečeno, že informace o tom, zda první bajt je nulový, je potenciálně nebezpečná a že systém musí být navržen tak, aby tuto informaci „nevyzařoval“. Přitom však norma uvádí jen velmi chabý náznak, jak tuto podmínku splnit. Manger pak ve své práci ukazuje několik ryze pragmatických důvodů, proč se lze domnívat, že běžné aplikace této normy (na něž zřídka dohlížejí fundovaní kryptologové) budou zranitelné.

Alibisticky vzato tedy standard PKCS#1 v současné verzi nemá problém. Z praktického hlediska se však dá namítnout, že norma, která chce dávat přímý návod pro implementaci, by neměla realizátory nechávat takto napospas osudu. Závěrem tedy lze říci, že k průlomu normy jako takové nedošlo, avšak bylo ukázáno, že není příliš dobře zpracována. O tom, kolik praktických systémů je v důsledku toho zranitelných, lze jen spekulovat.

Vlastimil Klíma | vlastimil.klima@i.cz
Tomáš Rosa | tomas.rosa@i.cz

INFOTIPY

O použití RSA, normě PKCS#1, v. 2.0 a 2.1 (draft 1) a Bleichenbacherově útoku:
Klíma, V.: Bezpečné použití RSA, Chip 11/00, str. 52 – 56

Bleichenbacherův útok z roku 1998:
[BLEI98] Bleichenbacher, D.: Chosen Ciphertext Attacks against Protocols Based on the RSA Encryption Standard PKCS #1, *Crypto '98*, str. 1 – 12 Springer Verlag, 1998

O postranních kanálech:
Rosa, T.: Kryptografie v klidu a bezpečí (seriál článků), Chip 2/01 až 9/01, viz archiv

Mangerův útok:
[MANG01] Manger, J.: Chosen Ciphertext Attack on RSA Optimal Asymmetric Encryption Padding (OAEP) as Standardized in PKCS#1 v2.0, Crypto'2001, Springer Verlag, 2001, str. 230 – 238

Nové a připravované úpravy:
http://www.rsalabs.com/pkcs/pkcs-1/index.html

Metoda OAEP:
[BERO94] Bellare, M., Rogaway, P.: Optimal Asymmetric Encryption – How to Encrypt with RSA. *Eurocrypt '94*, str. 92 – 111, Springer Verlag, 1994

Znovuposouzení bezpečnosti OAEP:
[SHOUP01] Shoup, V.: OAEP Reconsidered, Crypto'2001, str. 239 – 259, Springer Verlag, 2001

Archiv článků z Chipu:
http://www.decros.cz/bezpecnost/_kryptografie.html

Postup při odšifrování

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–2–2hLen*, kde *hLen* je délka výstupu hašovací funkce použité v EME-OAEP

Chyby: „*dešifrovací chyba*"

Postup:
1. Není-li délka šifrového textu *C k* oktetů, vrať „*dešifrovací chyba*" a skonči.
2. Konvertuj řetězec *C* na celé číslo *c* pomocí funkce 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.
4. Konvertuj číselnou reprezentaci zprávy *m* na oktetový řetězec *EM* o délce *k–1* oktetů pomocí funkce I2OSP: *EM* = I2OSP(*m*, *k–1*). Je-li jejím výstupem „*číslo je příliš velké*", pak vrať „*dešifrovací chyba*" a skonči. *Pozn. autorů:* to nastane, pokud *k*-tý nejvyšší bajt zprávy *m* není nulový (viz specifikace funkce I2OSP), což je klíčové pro Mangerův útok.
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.
6. Výstup je *M*.

Konvertující funkce I2OSP
I2OSP je zkratka pro *Integer-to-octet-string-primitive* a konvertuje nezáporné celé číslo na řetězec oktetů specifikované délky. Všimněte si, že po odšifrování v kroku 4 transformace RSAES-OAEP-DECRYPT(*K*, *C*, *P*) je právě volání této konvertující funkce.

I2OSP(*x*, *l*)

Vstup:
x nezáporné celé číslo, které má být konvertováno
l délka výsledného řetězce

Výstup:
X odpovídající řetězec *l* oktetů

Chyby: „*celé číslo je příliš velké*"

Postup:
1. Je-li $x \geq 256^l$, vrať „*celé číslo je příliš velké*" a skonči.
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 $0 \leq x_i < 256$ (jedno nebo více vedoucích cifer může být nula, pokud $x < 256^{l-1}$).
3. Nechť oktet X_i má číselnou hodnotu x_{l-i} for $1 \leq i \leq l$. Potom výstupem je řetězec $X = X_1\,X_2 \ldots X_l$.

Postup při dekódování

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*,
P kódovací parametr, oktetový řetězec

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

Chyby: „*dekódovací chyba*"

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.
2. Je-li *emLen* < *2hLen*+1, vrať „*dekódovací chyba*" a skonči.
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.
10. Je-li *pHash'* různá od *pHash*, vrať „*dekódovací chyba*" a skonči.
11. Výstup je *M*.