



BEZPEČNOST RSA

RSA v novém světle

(2)

Minule jsme se dotkli ožehavého tématu: šifra RSA, donedávna považovaná za zcela bezpečnou, se v případech ne zcela korektní implementace může změnit v nevěrného zrádce. Nyní se podíváme na další detaily a zkusíme se i zamyslet nad otázkou „Co s tím?“.

Zmíněná implementační chyba může mít za následek i takovou „malíčkovost“, jakou je odhalení kompletního otevřeného textu – tedy právě toho, co mělo být skryto! Útok je velmi reálný, a bude proto nezbytné přezkoumat všechny implementace RSA s cílem zjistit, zda jsou vůči němu imunní, a eventuálně je opravit tak, aby mu zabránily. Jak uvidíme v závěru, je totiž vysoce pravděpodobné, že velmi mnoho implementací tento útok umožní. Spolehnout se na délku modulu RSA zde nepomůže – na 1024bitový modul postačí zhruba 1100 útočníkem vyslaných „sond“ (obecně jich je na n -bitový modul zapotřebí přibližně $n + 100$).

JAK UŽ BYLO ŘEČENO...

V úvodním dílu jsme si „připravili půdu“ pro to, abychom postup, předpoklady a úspěšnost útoku mohli vysvětlit; zde už tedy pojmy a označení zavedené v minulém dílu nebudeme opakovat. Připomeneme jen, že po Bleichenbacherově útoku na formát PKCS#1 verze 1.5 (viz infotypy) z roku 1998 byla pro šifrování dat prostřednictvím RSA doporučována nová procedura RSAES-OAEP v PKCS#1 verze 2.1. Metoda OAEP navíc do zprávy vnáší integritní informaci a poté celkově zamaskování struktury (čímž brání Bleichenbacherově útoku).

Aby se však taková zpráva „vešla“ pod modul RSA (n), byla k výsledku tohoto zakódování přidána ještě levostranná nula – takto vzniklá formáto-

vaná zpráva (m) už potom splňuje podmínku $m < n$, nutnou pro správné odšifrování pomocí RSA. Avšak ona připojená nula se ukázala být základem pro „postranní kanál“ vyzařující informaci o m . Při odšifrování totiž některé implementace RSA zastaví výpočet, když tuto nulu na začátku nenajdou. Zdá se to logické – proč pokračovat, když další výpočet by byl beztak na nic (zpráva bez nuly na nejvyšším místě nevyhovuje předpokladům dešifrovacího algoritmu, protože algoritmus zašifrování ji tam vždy umístí). Na první pohled zcela oprávněně chybové hlášení je však postranním kanálem, vynášejícím informaci o otevřeném textu (blíže viz [ROSA]). A jak ukázal Mangerův útok ze srpna t. r., pokud se některá implementace takto chová, kýženě utajené je ztraceno a m lze získat.

DETAILLY

Po krátkém osvěžení paměti přejdeme k podrobnostem. Jako k si označíme počet bajtů modulu n a pro jednoduchost si útok ukážeme pro n o délce přesně $8 \cdot k$ bitů (tj. $n \geq 2^{8k-1}$). Ostatně v praxi se používají téměř výhradně takovéto moduly (většinou je dokonce počet jejich bitů násobkem 64). Útok, i když mírně modifikovaný, ale funguje na moduly jakékoliv délky. Číslo, které je bajtově stejně dlouhé jako modul n , ale v k -tém bajtu má nastaven pouze nejnižší bit, označme B , tedy $B = 2^{8k-1}$, viz obr. 1. Protože m má k -tý bajt nulový, platí vždy $m < B$.

Dejme tomu, že naše vyhlédnutá „oběť“ komunikuje s bankou pomocí protokolu SSL a že máme zájem tuto komunikaci rozšifrovat. V reklamách na e-bankovníctví je sice SSL vždy vyzdvihováno jako záruka bezpečnosti, ale jak uvidíme, při nevhodné implementaci tomu tak být nemusí.

Vytvoření spojení prostřednictvím SSL probíhá tak, že před vlastní komunikací je na straně klienta vygenerováno náhodné tajemství (označíme je zde M) určené k odvození komunikačních klíčů, které je serveru banky zasláno zašifrované algoritmem RSA. V tento okamžik na síti →

INFOTYPY

O 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

O postranních kanálech ▶ [ROSA] 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

O významu jednotlivých bitů RSA
▶ [HANA98] Håstad, J. a Näslund, M.: The Security of Individual RSA Bits, FOCS '98, IEEE 1998, str. 510 – 521

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

Odpověď orákula	k-tý bajt	(k-1)-tý bajt	(k-2)-tý bajt	1. bajt	popis
	11010100	10011101	1101	101	n
	1	00000000	0000	000	B
		11011	0101	001	m
" < B "		110110	101	010	2*m
" < B "		1101101	01	100	4*m
" < B "		11011010	1	1000	8*m
" ≥ B "	1	10110101	10000	16*m

↳ nejvýznamnější bit m je 5. bit v (k-1)-tém bajtu

Obr. 1. Postup určení nejvýznamnějšího bitu m

→ zachytíme odpovídající šifrový text c i celou následující šifrovanou komunikaci (kterou na konci naší činnosti budeme schopni odšifrovat). Server odšifruje c , čímž získá m , jehož dekódováním určí M . Další komunikace už probíhá pod ochranou symetrické šifry s klíčem odvozeným od M . K zašifrování M je použit právě formát PKCS#1 (metoda EME-OAEP), tedy M se doplní a zakóduje do délky $k-1$ bajtů a přidá se levostranná nula. Taktov vzniklý řetězec se převede na číslo (m), které se zašifruje na $c = m^e \bmod n$. Naším cílem je získání m , protože z něj už triviálně dekódujeme M .

$f^*m \bmod n$ nedostane nulu; předpokládáme, že nám to prozradí přímo chybovým hlášením, nebo se to dozvíme třeba časovým postranním kanálem (pokud nula vyjde, poznáme to obvykle podle delší odezvy).

Máme tedy k dispozici „orákulum“, které pro námi zvolené f odpoví, zda $f^*m \bmod n \geq B$ (a že tedy v nejvyšším bajtu není nula), nebo zda $f^*m \bmod n < B$ (nula tam je). Promyslíme-li si to důkladně, zjistíme, že **na každý dotaz položený orákulu dostáváme přibližně jeden bit informace o f^*m .** Z toho lze usoudit, že na N -bitový modul RSA postačí řá-

Většina implementací RSAES-OAEP není imunní proti postranním kanálům.

Naši „špionáž“ můžeme zahájit například po skončení sledované komunikace. Příslušnému serveru (dešifrovacímu stroji) po nezbytné úvodní komunikaci místo skutečného zašifrovaného klíče začínáme postupně posílat naše „sondy“ – šifrové texty typu $c' = c^{*f} \bmod n$ pro vhodná f a zachycené c . Dešifrovací stroj je zkouší odšifrovat, a tak obdrží $c^d \bmod n = (c^{*f})^d \bmod n = c^d * f^{fd} \bmod n = f^*m \bmod n$. Přitom občas v nejvyšším bajtu čísla

dově N dotazů orákulu; konkrétně pro 1024bitové n nám postačí cca 1100 dotazů.

Na základě tohoto orákula můžeme zosnovat náš útok. Jakmile jsme schopni určit jeden bit m , z teorie (viz [HANA98]) víme, že existuje algoritmus, který v náhodném polynomiálním čase určí i všechny ostatní bity. Mangerův postup ale využívá specifické vlastnosti daného orákula, a je tak mnohem jednodušší, než tato oklika přes teorii. Formálně je tento postup rozložen do tří kroků, které vidíte v přípojných rámečcích.

Hlavní myšlenky teď objasníme jen neformálně; pokud bychom je totiž precizovali, postup by se příliš větvil a princip by zanikal ve formalismech. (Z tohoto pohledu je nutno brát i následující intervalové odhady, které nejsou vždy zcela přesné.)

V kroku 1 jde o určení pozice nejvýznamnějšího bitu m . Postup můžete sledovat na obrázku 1 – jak je vidět, naše m má jako nejvýznamnější pátý bit v $(k-1)$ -tém bajtu. Jako útočníci to nevíme, ale dokážeme na to přijít. Postačí nám posílat jako šifrové texty hodnoty $f_1^e * c$ postupně pro $f_1 = 2, 4, 8, 16, \dots$. Orákulum po odšifrování dostává hodnoty $2^*m, 4^*m, 8^*m, \dots$ (což je m posouvane o 1, 2, 3, ... bity doleva) a reaguje odpo-

KROK 1

Jako f_1 zkoušejme po řadě 2, 4, 8, ... 2^i tak dlouho, dokud orákulum nevrátí " $\geq B$ ". Konkrétně:

1.1 Víme, že $m \in \langle 0, B \rangle$. Nechť $f_1 = 2$.

1.2 Platí tedy $f_1^e * m \in \langle 0, 2B \rangle$. „Zkusme f_1^e “, tj. zašleme orákulu sondu $f_1^e * c \pmod n$. Orákulu po odšifrování vyjde $f_1^e * m \pmod n$. Modulo n můžeme ale nyní vynechat, protože $f_1^e * m \in \langle 0, 2B \rangle$ a $2B < n$ (n má 8k bitů).

1.3a Jestliže orákulum vrátí "< B", znamená to, že $f_1^e * m \in \langle 0, B \rangle$, takže $2f_1^e * m \in \langle 0, 2B \rangle$. Proto zvolíme nově $f_1 := 2^*f_1$ a jdeme na bod 1.2.

1.3b Jestliže orákulum vrátí " $\geq B$ ", znamená to, že $f_1^e * m \in \langle B, 2B \rangle$, takže $(f_1/2)^e * m \in \langle B/2, B \rangle$, a přejdeme na krok 2.

vědí „< B“, protože chvíli trvá, než nejnvýznamnější bit m takto „docestuje“ až na první místo k-tého bajtu. V tomto okamžiku ale orákulum odpovídá „≥ B“ (na obrázku 1 to nastalo při 16*m). Tak určíme nejnvýznamnější bit m.

Z obrázku 1 je dále zřejmé, že s číslem f_1*m jsme se dostali na nastavení nejnižšího bitu v k-tém bajtu. Proto $256*f_1*m$ je určitě větší než n (a tedy $512*f_1*m$ určitě větší než n+B) a $64*f_1*m$ je zase určitě menší než n. V **kroku 2** se (postupným zvyšováním) právě hledá nejvyš-

ší možné f_2 mezi $64*f_1$ a $512*f_1$ tak, aby f_2*m ještě bylo menší než n+B. Proč to děláme? Jednoduše proto, že potřebujeme násobek f_2*m dostat do oblasti <n, n+B), abychom s ním mohli v dalším kroku pracovat.

Hlavní myšlenka **kroku 3** je tato. Na počátku volíme $f_3 = f_2$, takže máme z minulého kroku zaručeno, že $f_3*m \in <i*n, i*n+B)$ pro vhodné i (na počátku $i=1$), neboli rozsah omezujícího intervalu pro f_3*m je cca B hodnot. Pokud volíme v dalších krocích vždy nové f_3 (zjednodušeně)

může odlišit chybu vrácenou funkcí I2OSP (která nastane při konverzi odšifrovaného celého čísla na řetězec, pokud nemá levostrannou nulu) od chyby v dekódovací transformaci EME-OAEP-DECODE (která vznikne, pokud po dekódování neobdržíme tvar `seed || pHash || PS || 01 || M`), viz minulý díl. Norma PKCS#1 v 2.1 sice upozorňuje, že možnosti odlišení těchto chyb by mělo být zabráněno, nicméně se ukázalo, že mnoho implementací to nerespektuje. Podívejme se na nejčastější implementační prohřešky.

KROK 2

Hledáme maximální možné f_2 tak, že $f_2*m \in <n, n+B)$. To se nám podaří najít díky vhodné počáteční volbě f_2 a poté jejím zvyšováním tak dlouho, dokud orákulum vrací „< B“, tj. platí, že $f_2*m \bmod n$ je menší než n+B. Konkrétně:

2.1 Máme $(f_1/2)*m \in <B/2, B)$. Necht $f_2 = \lfloor (n+B)/B \rfloor * (f_1/2)$, přičemž symbol $\lfloor x \rfloor$ znamená zaokrouhlení x dolů na celé číslo a $\lceil x \rceil$ zaokrouhlení nahoru.

2.2 Máme $f_2*m \in <n/2, n+B)$. Zkusme f_2 s orákulem.

2.3a Jestliže orákulum vrátí „≥ B“, znamená to, že $f_2*m \in <n/2, n)$, takže $(f_2+f_1/2)*m \in <n/2, n+B)$. Zvolme nové $f_2 := f_2 + f_1/2$ a přejděme zpět na bod 2.2.

2.3b Jestliže orákulum vrátí „< B“, znamená to, že $f_2*m \in <n, n+B)$ pro známý násobek f_2 , takže můžeme přejít na krok 3.

KROK 3

Zkoušíme násobky f_3 , pro něž je rozsah hodnot f_3*m široký přibližně 2B a v němž se nalézá jedna „hraniční“ hodnota tvaru $i*n+B$. Každá odpověď orákulu tento interval hodnot opět zužuje na interval šířky B. Každý následující násobek f_3 je přibližně dvojnásobkem předchozího. Konkrétně:

3.1 Mějme $f_3*m \in <n, n+B)$. Takže máme násobek f_2 a rozsah $\langle m_{\min}, m_{\max} \rangle$ možných hodnot m, kde $m_{\min} = \lfloor n/f_2 \rfloor$, $m_{\max} = \lfloor (n+B)/f_2 \rfloor$ a $f_2*(m_{\max}-m_{\min}) \approx B$.

3.2 Zvolme $f_{\text{tmp}} = \lfloor 2B/(m_{\max}-m_{\min}) \rfloor$, tj. šířka rozsahu hodnot $f_{\text{tmp}}*m$ je cca 2B.

3.3 Zvolme $i = \lfloor m_{\min}*f_{\text{tmp}}/n \rfloor$, tj. hraniční bod $i*n+B$ je uvnitř rozsahu hodnot $f_{\text{tmp}}*m$.

3.4 Zvolme f_3 tak, že rozsah hodnot f_3*m obsahuje jediný hraniční bod tvaru $i*n+B$, tj. $f_3 = \lfloor i*n/m_{\min} \rfloor$. To dává $f_3*m \in <i*n, i*n+B)$, ačkoliv horní hranice je pouze přibližná; f_3 je přibližně rovno f_{tmp} . Zkusme f_3 s orákulem.

3.5a Jestliže orákulum vrátí „≥ B“, znamená to, že $f_3*m \in <i*n+B, i*n+B)$. Nastavme $m_{\min} := \lfloor (i*n+B)/f_3 \rfloor$ a vraťme se na bod 3.2.

3.5b Jestliže orákulum vrátí „< B“, znamená to, že $f_3*m \in <i*n, i*n+B)$. Nastavme $m_{\max} := \lfloor (i*n+B)/f_3 \rfloor$, a pokud v intervalu $\langle m_{\min}, m_{\max} \rangle$ nezbyvá už jen jediná hodnota, vraťme se na bod 3.2.

I dobrá norma se může stát bezpečnostně křehkou, pokud se implementuje bez hlubších znalostí.

jako dvojnásobek starého f_3 , pak nové f_3*m leží v intervalu $\langle 2i*n, 2i*n+2B)$, jehož šířka je 2B hodnot. Orákulum ovšem vypočítá $f_3*m \bmod n$, což je hodnota z intervalu $\langle 0, 2B)$, a vrátí „< B“ nebo „≥ B“, takže se z jeho odpovědi dozvíme, zda f_3*m je z intervalu $\langle 2i*n, 2i*n+B)$ nebo $\langle 2i*n+B, 2i*n+2B)$. V obou případech orákulum zúží rozsah nové množiny hodnot f_3*m z 2B opět na B hodnot.

Jestliže však zvyšujeme f_3 vždy zhruba dvojnásobně, a přesto šíře možných hodnot f_3*m zůstává přibližně stále stejná (B), je zřejmé, že prostor $\langle m_{\min}, m_{\max} \rangle$ pro možné hodnoty m se zužuje v každém kroku asi na polovinu. Postup v kroku 3 proto v každé iteraci vypočítává tento nový (zúžený) interval pro m, až v něm zbude jen jedna možná hodnota. Zhruba je možné říci, že buď se zvyšuje m_{\min} na $(m_{\min}+m_{\max})/2$, nebo se m_{\max} snižuje na $(m_{\min}+m_{\max})/2$, což vám jistě připomene metodu půlení intervalů. Postup v kroku 3 názorněji ilustruje obrázek 2.

Jak jsme poznamenali výše, kroky 1 až 3 popisují pouze *ideu* zužování možných hodnot. Podrobný důkaz a přesný postup by vyžadovaly více formalismu; na druhé straně však vidíme, že zbytek je ryze technická záležitost, a proto tento postup a jeho úspěch nikdo nepopírá.

PRAKTICKÁ PROVEDITELNOST ÚTOKU

Kroky 1 a 3 dohromady vyžadují přibližně $\log_2 B = 8(k-1)$ dotazů orákulu a krok 2 typicky cca $n/B \approx 100$ dotazů. Pro 1024bitový modul RSA tedy vychází cca 1100 dotazů, pro 2048bitový modul asi 2200 dotazů. Mangerův útok je typu *Adaptive Chosen Ciphertext*, tj. adaptivní útok s možností volby šifrovaného textu. Předpokládá, že útočník

CHYBOVÁ HLÁŠENÍ

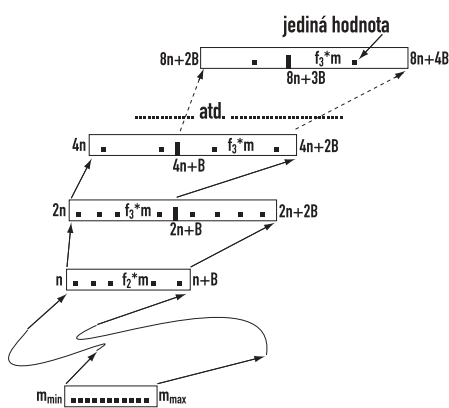
Chybová hlášení z obou možných míst mají být stejná, což mnohdy z „praktického“ hlediska splněno je, ale fakticky tomu tak není. Postačí například, když se chybová hlášení liší v tečce na konci, mezeře nebo velkém/malém písmenu na začátku hlášky. Pro (nepěčlivého) programátora jsou hlášení shodná, pro kryptologa odlišná.

LOGOVACÍ SOUBORY

Dalším možným zdrojem odlišení jsou logovací soubory. I když aplikace vydává stejná chybová hlášení (tj. programátor skutečně použije stejný výstupní řetězec metodou copy-paste), do logovacích souborů jsou při vzniku chyby mnohdy zapisovány podrobnější údaje. Praktická dostupnost logu je obecnější bezpečnostní problém, který závisí na různých okolnostech, ale právě tam se může objevit zpřesnění, jak, kdy nebo kde „decoding error“ nastala. (Že kryptoanalytik v logovacím souboru nadšeně přivítá hlášení „Integer too large“, snad není třeba zdůrazňovat...)

MÉNĚ VIDITELNÉ IMPLEMENTAČNÍ ASPEKTY

Při praktické implementaci musí programátor řešit různé situace. Útočník může například nastavit identifikátor hašovací funkce nebo funkce MGF na nepodporovanou hodnotu. Pokud implementace začne identifikátor vyhodnocovat až v okamžiku použití, je to přesně onen typ chybového hlášení, na jaký útočník čeká. Pokud takto napsaná aplikace *nevzdá* hlášení „nepodporovaná hašovací funkce (resp. MGF apod.)“, útočník ví, že program *nepošel* už přes kontrolu I2OSP na levostrannou nulu, což je totéž, jako kdyby dostal hlášení „Integer too large“. →



Obr. 2. Zužování možných hodnot pro m ve třetím kroku algoritmu

→ ČASOVÝ POSTRANNÍ KANÁL

Na informaci, kterou poskytuje časový postranní kanál, se v běžných realizacích RSA velmi často zapomíná. Z doby trvání celé dešifrovací operace může útočník zjistit, zda byla přerušena velmi brzo funkcí I2OSP (odpovídá hlášce „Integer too large“), nebo postoupila dále do časově náročnější transformace EME-OAEP-DECODE k dekódování (odpovídá hlášce „Decoding error“).

Zajímavá je i myšlenka, jak prodloužit (a tím odlišit) dobu dekódování tím, že se transformaci EME-OAEP-DECODE vnutí ohromný parametr P (třeba 10 MB), který slouží k vytvoření kontrolního řetězce *phash* (viz minulý díl). Pokud se P začne zpracovávat až při dekódování, doba dekódování neúměrně naroste, čímž útočník získá informaci na úrovni hlášení „De-

coding error“ (k jejímuž rozpoznání postačí náramkové hodinky). Proto je nezbytné operaci přípravy *phash* provádět před vlastním dekódováním. Který z programátorů na to ale před několika měsíci (ne-li roky) vůbec pomyslel?

rozhodování nevyzařovalo postranní informaci. Obě cesty mají své slabiny, nicméně první se zdá být z hlediska dlouhodobého vývoje lepší. Podstatnou nevýhodou druhého přístupu je totiž obtížná prokazatelnost účinku provedených úprav. Konstruktor se sice domnívá, že jím navržený postup žádnou postranní informaci nevyzařuje, ale dokázat tuto domněnku formálně může být značně obtížné. Snadno tak obdržíme systém charakterizovaný známým úslovím „security by obscurity“.

Nevýhodou první cesty je zase nízká teoretická rozpracovanost tohoto přístupu vůbec, ačkoliv po jeho zvládnutí lze očekávat srozumitelný důkaz bezpečnostních vlastností. Problémem zde je, že OAEP a jemu podobná schémata jsou studována s předpokladem, že použitá šifrovací funkce má blokový charakter. To však u RSA neplatí – například při délce modulu 1024 bitů nelze využít všechny bloky délky 1024 bitů, aniž bychom se dostali do kolize (díky operaci modulo n). Pokud máme provést rozšíření kódovacího schématu tak, aby byla využita celá doména otevřených textů RSA, bude nutné podmínku blokového charakteru opustit. To je úkol, který v nejbližší době čeká na teoretickou kryptologii. Praxe však tak dlouho čekat nemůže, takže zde po nějakou dobu chtě nechtě budou muset přijít ke slovu úpravy prvního druhu.

SHRNUTÍ

Mangerův útok ukazuje, jak křehká může být i dobrá norma (o jakou se jistě PKCS#1 v. 2.1 snažila), pokud se implementuje bez všech potřebných znalostí. Je možné, že některé imple-

RSA umožňuje dosáhnout změnou šifrového textu zajímavých triků v otevřeném textu.

mentace RSAES-OAEP jsou imunní proti popsaným postranním kanálům, ale z toho, co jsme uvedli, je jisté cítit, že u mnoha tomu tak nebude. V dalším pokračování vám nabídneme náš pohled na věc a pokusíme se navrhnout změny v normě PKCS#1. Vydeme přitom ze snahy co nejméně stávající normu měnit, zachovat plnou datovou kompatibilitu a současně se vyhnout nepodloženým bezpečnostním konstrukcím.

CO S TÍM?

Útok využívá odlišných reakcí dešifrovacího stroje (orákula) vyplývajících z rozhodnutí o struktuře obdrženého otevřeného textu. Nabízejí se tedy dvě možnosti – buď vyloučit rozhodování přímo závislé na struktuře otevřeného textu RSA (m), nebo zajistit, aby toto