Jak pašáci ověřují podpis RSA

Možná si vzpomenete na příběh O pašákovi, který chránil hesla, jenž vyšel v ST 3/2006. Kromě jisté parodie na hašovací funkci jsme si tehdy také zavedli speciální výraz *pašák* jako označení pro sebevědomé všeuměly na poli kryptografickém. Nyní si ukážeme, jak v jejich rukou může dopadnout tak nevinná záležitost, jako je ověřování podpisu RSA. Pašáků schopných následovat níže popsané příklady je určitě dost, takže doporučujeme věnovat této na první pohled banální záležitosti patřičnou pozornost.

Co praví kryptografie

Zaměříme se na podpisové schéma RSA podle dominujícího standardu PKCS#1 s formátováním EMSA-PKCS1-v1_5 [1] (viz také ST 10/2003 v [4]). Nechť (*N, e*) je veřejný klíč (modul, exponent) RSA a (*N, d*) je klíč soukromý (modul, exponent). Při výpočtu podpisu *S* zprávy *M* nejprve vypočteme hašový kód *h(M)*, kde *h* značí použitou hašovací funkci. Poté na něj aplikujeme formátovací pravidla uvedeného standardu, čímž vznikne řetězec $\boldsymbol{m = 00 \| 01 \| FF \| \ldots \| FF \| 00 \| ID_h \| h(M)}$, kde operátor ‖ značí zřetězení. Počet výplňových bajtů FF je volen tak, aby binární délka *m* byla stejná jako délka použitého modulu RSA. Řetězec ID_h je identifikační prefix, který závisí na konkrétní zvolené hašovací funkci, viz [1]. Na takto vzniklé *m* pak aplikujeme odšifrovací neboli podepisovací transformaci RSA. Tím získáme hledanou hodnotu podpisu $S = m^d \bmod N$. Při ověřování podpisu nejprve aplikujeme šifrovací neboli ověřovací transformaci RSA (výklad o názvosloví viz ST 8/2003 v [4]), čímž získáme $x = S^e \bmod N$. Nyní začíná stěžejní část postupu – je nutné zkontrolovat, že získaná hodnota *x* je ve stejném tvaru, v jakém byla hodnota *m* při vytváření podpisu. Nejspolehlivější je přitom přímočarý způsob, který doporučuje i standard [1], kdy pro předloženou zprávu *M* znovu sestavíme hodnotu m a jednoduše ověříme, jestli *x* = *m*. Pokud ano, uznáme podpis za platný, v opačném případě ho odmítneme. Standard na druhou stranu nezakazuje (a proč by také měl) postup, řekněme, *per partes*, kdy shodu *x* a *m* ověříme jaksi po částech: Zvlášť zkontrolujeme přítomnost prefixu 00 01, zvlášť ověříme pozici a délku výplně FF… FF, zkontrolujeme separátor 00 atd. Střídmě uvažující jedinec se však do takového martyria sotva pustí, neboť tím mimo jiné evidentně stoupá riziko, že na něco důležitého zapomene. Co je však podle nás nerozum, to je pro pašáky moderně řečeno „výzva".

Stalo se v zahraničí

První vlaštovku vyslal Daniel Bleichenbacher na konferenci CRYPTO 2006. V sekci krátkých příspěvků prezentoval výsledky ad hoc inspekce kódu knihoven projektu OpenSSL [2], [3]. Nějaký pašák se tam rozhodl aplikovat přístup per partes, jenže bohužel přecenil svou pozornost. Zalíbilo se mu, že řetězec $ID_h \| h(M)$ lze také interpretovat jako datovou strukturu ASN.1, což je obecně uznávaný, platformově nezávislý jazyk pro popis datových objektů [4]. Zde konkrétně platí následující syntaxe:

```
DigestInfo ::= SEQUENCE {
    digestAlgorithm AlgorithmIdentifier,
    digest OCTET STRING
    }.
```

Vidíme, že se jedná o strukturu (typ SEQUENCE), jejímiž prvky jsou jednak identifikátor a případné parametry hašovacího algoritmu (zabalené do struktury AlgorithmIdentifier), jednak bajtový řetězec (OCTET STRING) představující vlastní hašový kód. Místo prostým srovnáním zakódované podoby, jak radí standard, se zde pašák rozhodl uvedenou strukturu kontrolovat přes její zevrubnou syntaktickou analýzu. Zda je však ona struktura v rámci řetězce *x* vůbec na správném místě, to už mu bohužel unikalo. Za platný podpis tak byla uznána i hodnota *S*, pro kterou platilo $S^e \bmod N = x$, kde $x = 00 \| 01 \| FF \| \ldots \| FF \| 00 \| ID_h \| h(M) \| \boldsymbol{GRB}$, kde GRB je nějaký neprázdný řetězec volený útočníkem. Tolerance přítomnosti *GRB* je natolik zásadní slabina, že pro určité klíče s veřejným exponentem 3 (sice jsou na ústupu, ale stále se vyskytují mj. v oblasti platebních karet) lze konstruovat „správné" hodnoty podpisu bez znalosti soukromého klíče! Slovo správné jsme dali do uvozovek, neboť podle standardu takový podpis platný pochopitelně není, jenže chybná ověřovací procedura jej za platný považuje, a to je v daném okamžiku směrodatné. Na to ostatní se přijde patrně až u soudu, přijdeli se na to vůbec. Dodejme, že OpenSSL bylo už opraveno, přičemž tato lekce mu spíš prospěla, než uškodila. Asi žádný kód není úplně bezchybný, a tak podstatné je, že někdo má zájem chyby hledat a opravovat. Víme o spoustě dalších kódů, kde podobné chyby patrně přetrvávají, protože jejich kontrola nikoho příliš nezajímá.

Naši drží laťku

Ve vlastní praxi jsme narazili na případ, kdy bylo ověřováno jen a pouze to, jestli se na konci celého řetězce *x* vyskytuje hodnota *h(M)*. Za správný podpis zprávy *M* tak byla považována každá hodnota *S* splňující $S^e \bmod N = x$, kde $x \bmod 2^L = h(m)$ a *L* je délka hašového kódu. Pro ilustraci si ukážeme jeden útok vytvořením „správného" podpisu *S* bez znalosti soukromého klíče.

Předpokládejme veřejný exponent *e* = *3*, modul *N* délky 1024 bitů a hašovací funkci SHA-1 (*L* = 160). Hledáme číslo *S*, splňující $(S^3 \bmod N) \bmod 2^{160} = h(M)$. Definujme $w = (3*2^{54})^{-3} \bmod N$ a $D = (h(m) - w) \bmod 2^{160}$. Nyní můžeme položit $S = (1 + 9*2^{162}*D)* (3*2^{54})^{-1} \bmod N$ a „správný" podpis je na světě. Dosazením se o tom snadno přesvědčíme: $S^3 \bmod N = (1 + 9*2^{162}*D)^3*(3*2^{54})^{-3} \bmod N = [(27*2^{324}*D^3 + 9*2^{162}*D^2 + D)\ 27*2^{162} + 1]*w \bmod N = (27*2^{164}*D^3 + 9*4*D^2)*2^{160} + D + w \bmod N$. S ohledem na velikost jednotlivých členů výrazu lze ukázat, že poslední redukci mod *N* už lze vynechat, tj. máme přímo $S^3 \bmod N = (27*2^{164}*D^3 + 9*4*D^2)*2^{160} + D + w$. Odtud už vidíme $(S^3 \bmod N) \bmod 2^{160} = (D + w) \bmod 2^{160} = h(m)$, takže náš podpis je skutečně „správný". Z pohledu zkušeného kryptoanalytika se jedná o přímočarý útok.

Závěrem

Chuť udělat něco po svém a neohlížet se na striktní předpisy je lidem asi vlastní. Zjevně ne každý ale dokáže vidět důsledky své originality. Ověřování digitálních podpisů je však dnes už vážná věc. Domníváme se proto, že u kritických aplikací neuškodí provést kontrolu výskytu výše popsaných slabin včetně jejich triviálních derivátů.

Při kontrole je nutné rozlišovat kryptografické procedury, které jsou připojovány až za běhu. Jedna a táž aplikace pak může být podle konfigurace systému buď v pořádku, nebo zranitelná. V reakcích na [2] jsme viděli řadu nesmyslných výroků o imunitě určitých aplikací, které však volají dynamicky připojované knihovny třetích stran.

Vlastimil Klíma, Tomáš Rosa,
v.klima@volny.cz, trosa@ebanka.cz

LITERATURA

[1] PKCS#1:RSA Cryptography Standard, http://www.rsasecurity.com/rsalabs/pkcs/index.html

[2] http://www.openssl.org/news/secadv_20060905.txt

[3] http://www.mail archive.com/cryptography@metzdowd.com/msg06537

[4] E-archivy http://cryptography.hyperlink.cz, http://crypto.hyperlink.cz