Kryptológia 2

Martin Stanek

Dokument voľne nadväzuje na predchádzajúci dokument *Kryptológia*, pričom postupne prehlbuje informácie o kryptografických konštrukciách. Napriek tomu, že obmedzíme matematickú stránku výkladu a v texte použijeme zjednodušenia, nevyhneme sa niektorým matematickým pojmom a zápisom. Predpokladáme, že čitateľ je oboznámený s poznatkami prezentovanými v dokumente *Kryptológia*. Záujemcom o podrobnejší a širší pohľad na túto problematiku možno znova odporučiť špecializovanú odbornú literatúru.

1 Symetrické konštrukcie

Symetrické šifry možno rozdeliť na blokové a prúdové. V praxi sú väčšinou používané blokové šifry. Dôvodom je fakt, že najdôležitejšie štandardy (napr. vydané NIST) primárne štandardizujú blokové šifry (v minulosti DES, v súčasnosti AES). Navyše, voľbou vhodného módu možno blokovú šifru používať aj ako prúdovú šifru.

1.1 Blokové šifry

Blokové šifry sú definované pre bloky bitov pevnej dĺžky, teda pre ľubovoľný kľúč šifra zobrazuje blok vstupných dát na rovnako dlhý blok zašifrovaných dát. Napríklad AES má dĺžku bloku 128 bitov, pre ľubovoľný variant dĺžky kľúča (teda 128, 192 alebo 256 bitov). Blokové šifry sú najčastejšie konštruované viacnásobnou iteráciou jednoduchšej transformácie (nazývanej „kolo“ algoritmu). Pre každé kolo sa používa špecifický kľúč, ktorý je odvodený presne definovaným spôsobom zo šifrovacieho kľúča.

Najpoužívanejšou blokovou šifrou súčasnosti je AES. Z hľadiska používania AES sú dôležité nasledujúce fakty:

- Viaceré procesory majú v hardvéri implementované špeciálne inštrukcie pre AES algoritmus. To znamená výrazné urýchlenie šifrovania a dešifrovania pre aplikácie/knižnice, ktoré takúto implementáciu vedia využiť. Ilustráciu výkonových rozdielov možno vidieť v časti 6.1.

Verzia 2b (november 2020)

- Použitie AES s dlhšími kľúčmi znamená mierne spomalenie šifrovania a dešifrovania, keďže AES-128 má 10 kôl, AES-192 má 12 kôl a AES-256 má 14 kôl. Pre praktické použitie je toto spomalenie zanedbateľné.

V starších systémoch sa možno stretnúť aj so šifrou 3DES (dĺžka bloku 64 bitov), niekedy označovaná ako Triple DES alebo TDEA. Šifra má varianty s kľúčmi dĺžky 56, 112, alebo 168 bitov. Pokiaľ je možné, odporúča sa 3DES v ľubovoľnom variante nepoužívať a migrovať na AES [25].

Operačné módy blokových šifier

Pre šifrovanie dát dlhších ako jeden blok je potrebné použiť šifru vo vhodnom operačnom móde. Nielen dĺžka dát však motivuje použitie rôznych módov. Tie sa navzájom odlišujú účelom použitia (napr. šifrovanie komunikácie alebo diskov), implementačnými vlastnosťami (napr. možnosť paralelizovať šifrovanie a/alebo dešifrovanie), bezpečnostnými požiadavkami (dôvernosť a autentickosť) a pod. NIST rozdeľuje operačné módy podľa účelu použitia do nasledujúcich kategórií[1]:

- dôvernosť (celkovo 5 módov: ECB, CBC, OFB, CFB, CTR)
- autentickosť (CMAC)
- autentizované šifrovanie (CCM) a autentizované šifrovanie s vysokou priepustnosťou (GCM)
- dôvernosť pre blokovo-orientované úložiská dát, napr. disky (XTS)
- dôvernosť a integrita kryptografických kľúčov (KW, KWP, TKW)
- formát zachovávajúce šifrovanie (FF1, FF3)

Poznamenajme, že uvedené módy nevyčerpávajú možnosti a rôznorodosť spôsobov použitia blokových šifier.

Pokiaľ ide o operačné módy určené výlučne pre dôvernosť údajov, najčastejšie sa v praxi možno stretnúť s CBC (Cipher Block Chaining) a CTR (Counter). Napríklad AES-128 v CBC móde je povinnou súčasťou v implementáciách TLS 1.2[2] a CTR mód zabezpečuje dôvernosť dát v GCM móde pre autentizované šifrovanie, pričom AES-128 v GCM móde je povinnou súčasťou implementácií TLS 1.3[3].

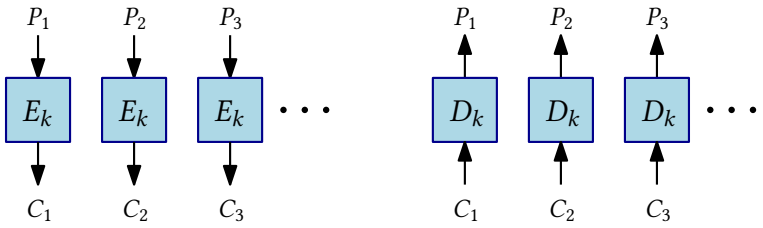
Schematické zobrazenie ECB, CBC a CTR módov je na obrázkoch 1, 2 a 3. Symboly P_1, P_2, P_3 označujú prvé tri bloky vstupných dát a C_1, C_2, C_3 zodpovedajúce bloky zašifrovaných dát. Šifrovací, resp. dešifrovací algoritmus pre jeden blok s využitím kľúča k je označený E_k, resp. D_k. V prípade CBC módu je IV inicializačný vektor a $\oplus$ označuje operáciu XOR dvojice blokov po jednotlivých bitoch (sčítanie modulo 2). CTR mód využíva pri šifrovaní a dešifrovaní počítadlo, inkrementované pre každý nasledujúci blok. Na obrázku 3 označuje ctr_i hodnotu počítadla pri šifrovaní bloku P_i.

Z praktického hľadiska je pri implementácii CBC ale aj ECB potrebné doriešiť spôsob šifrovania potenciálne neúplných posledných blokov v prípade, keď dĺžka otvoreného textu nie je násobkom dĺžky

[1] NIST Special Publication 800-38A, …, 800-38G (Recommendation for Block Cipher Modes of Operation), resp. https://csrc.nist.gov/Projects/Block-Cipher-Techniques/BCM/Current-Modes (november 2020)

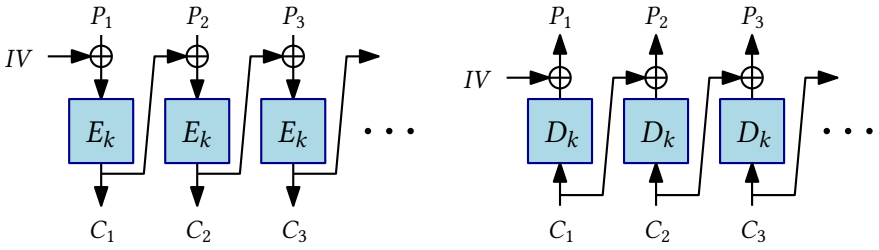
[2] The Transport Layer Security (TLS) Protocol, Version 1.2, RFC 5246, 2008.

[3] The Transport Layer Security (TLS) Protocol, Version 1.3, RFC 8446, 2018.



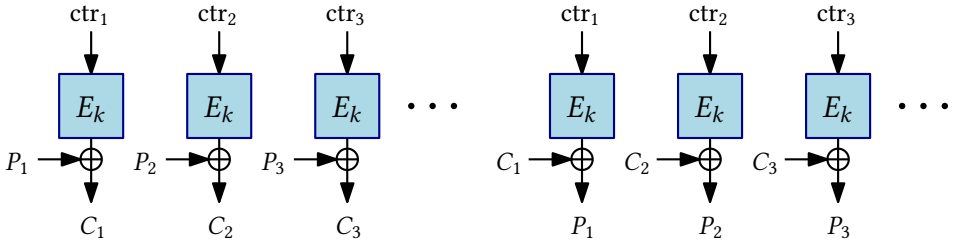
šifrovanie v ECB móde dešifrovanie v ECB móde

Obr. 1: ECB (Electronic Codebook) mód



šifrovanie v CBC móde dešifrovanie v CBC móde

Obr. 2: CBC (Cipher Block Chaining) mód



šifrovanie v CTR móde dešifrovanie v CTR móde

Obr. 3: CTR (Counter) mód

bloku. Obvyklé riešenie je použitie vhodnej výplne/zarovnania (tzv. padding). Ďalšou implementačnou otázkou pri CBC je voľba a prenos inicializačného vektora. Ten síce nemusí byť tajný, možno ho posielať spolu so zašifrovaným textom, ale má byť pre útočníka nepredikovateľný.

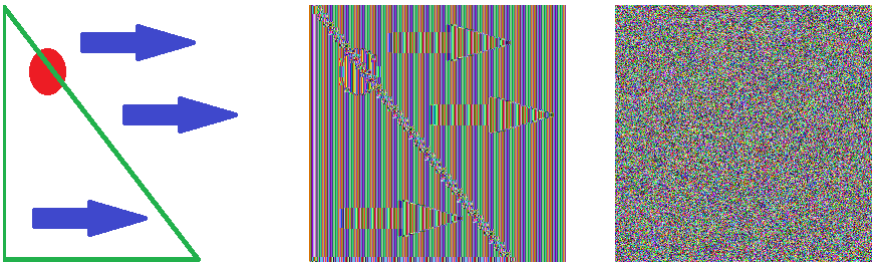
Pri implementácii CTR módu síce nie je potrebné riešiť šifrovanie neúplných posledných blokov (stačí pre XOR použiť len príslušnú časť výstupu z E_k), avšak pozornosť si vyžaduje voľba iniciálnej hodnoty počítadla. Požadujeme, aby sa hodnoty ctr_i neopakovali nielen v rámci šifrovania jednej správy, ale aj medzi všetkými správami šifrovanými rovnakým kľúčom. Preto sú niekedy bloky ctr_i rozdelené na dve časti – prvá je inicializačný vektor unikátny pre každú správu a druhá samotné počítadlo.

Správne implementovaný CBC alebo CTR mód sú bezpečné spôsoby použitia blokovej šifry pre zabezpečenie dôvernosti dát. Nevhodná implementácia v konkrétnej aplikácii však môže viesť k prob-

lémom. Príkladom je tzv. BEAST útok na staršie verzie protokolov SSL/TLS [8]. BEAST útok využíva nevhodný spôsob prenášania inicializačných vektorov v CBC móde medzi samostatnými správami (paketmi) v protokole. Podobne aj ostatné módy vyžadujú starostlivú implementáciu pre dosiahnutie želaných bezpečnostných vlastností. Iným príkladom je nevhodné použitie variantu CFB módu v Netlogon protokole, ktoré viedlo až ku kompromitácii Windows doménových radičov, tzv. Zerologon zraniteľnosť [23].

Nie každý operačný mód blokovej šifry je vhodný na ľubovoľné použitie. Príkladom je použitie ECB módu pri šifrovaní používateľských hesiel spoločnosťou Adobe v roku 2013. Odhliadnime teraz od skutočnosti, že heslá je potrebné ukladať inak (viac v časti 5.2). Použitie ECB módu bolo jednou z príčin, že po úniku databázy používateľov aj so zašifrovanými heslami, bolo možné heslá mnohých používateľov určiť. Zhoda dvoch blokov otvoreného textu v ECB móde totiž vedie k zhode príslušných blokov zašifrovaného textu. To znamená, že rovnaké bloky znakov hesiel rôznych používateľov, sú v zašifrovanom tvare ľahko rozpoznateľné. Druhým významným faktorom pre následné určenie hesiel bol uniknutý obsah „nápovied“, ktoré si používatelia zadávali pre prípad zabudnutia hesla.

Obrázok 4 vizuálne ilustruje vyššie uvedený fakt o ECB móde. Pri šifrovaní obrázka pomocou AES-256 (v tomto prípade na konkrétnej šifre až tak nezáleží) v ECB móde sú všetky bloky obsahujúce iba bielu zašifrované rovnako (hoci nie nutne na monochromatický blok). Podobne pre „modré“ bloky, atď. To znamená, že zo zašifrovaného obrázka je možné získať aj bez kľúča nejaké informácie o pôvodom obrázku. Na ilustráciu je priložený aj ten istý obrázok šifrovaný v CTR móde.



vstupné dáta | AES-256 v ECB móde | AES-256 v CTR móde

Obr. 4: Šifrovanie obrázka v ECB a CTR móde

Autentizované šifrovanie

Autentizované šifrovanie je taký spôsob symetrického šifrovania, ktorý okrem dôvernosti zabezpečuje súčasne aj autentickosť údajov. Tradičný spôsob dosiahnutia oboch požiadaviek je kombinácia šifrovania a autentizačných kódov správ. Existuje viacero spôsobov, ako tieto konštrukcie kombinovať, z ktorých niektoré nie sú (teoreticky) bezpečné alebo vhodné v konkrétnej situácii.

Autentizované šifrovanie spája obe operácie do jednej. Výhodou autentizovaného šifrovania je fakt, že algoritmus je popísaný jednoznačne a nie je potrebné ho ďalej kombinovať ani sa pri implementácii rozhodovať medzi viacerými možnosťami. Niektoré operačné módy blokových šifier sú navrhnuté práve pre autentizované šifrovanie. Najznámejšími módmi tohto typu sú GCM (Galois/Counter Mode) a CCM (Counter with CBC-MAC).

GCM používa na zabezpečenie dôvernosti dát interne CTR mód a na výpočet autentizačného tagu funkciu GHASH. Pri dešifrovaní sa následne overuje aj autentizačný tag a nesúlad medzi prijatým a vypočítaným autentizačným tagom znamená, že zašifrovaný text bol náhodne alebo úmyselne

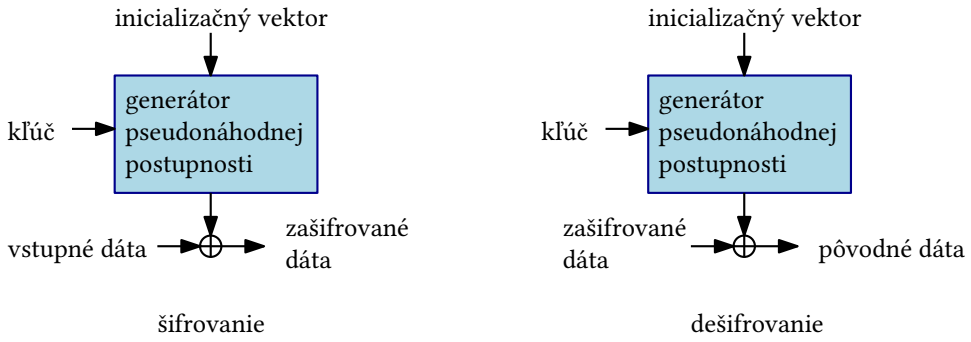
modifikovaný. Výhodou GCM je aj vyššia priepustnosť oproti generickej kombinácii CTR módu so štandardnou konštrukciou autentizačných kódov (ako napr. HMAC). Podobne ako iné módy, aj GCM si vyžaduje starostlivú implementáciu. Jedným zo vstupných parametrov módu je vektor, ktorý musí byť rôzny pre všetky správy šifrované tým istým kľúčom. V opačnom prípade je možné falšovať autentizačné tagy. Tento útok sa nazýva zakázaný útok (forbidden attack), keďže špecifikácia GCM voľbu opakujúcich sa vektorov zakazuje. V prípade TLS komunikácie s AES-GCM bola reálna situácia testovaná v roku 2016 [4], kde boli identifikované zraniteľné servery, pri ktorých dochádzalo k opakovaniu vektorov.

CCM používa na zabezpečenie dôvernosti dát, podobne ako GCM, interne CTR mód a na výpočet autentizačného tagu konštrukciu CBC-MAC. CCM je povinnou súčasťou implementácie štandardu IEEE 802.11i-2004 (WPA2) a je aj súčasťou WPA3 Personal. Poznamenajme, že WPA3 Enterprise pripúšťa v rámci EAP (Extensible Authentication Protocol) výlučne AES v GCM móde.

Niekedy je možné v praxi stretnúť pojem autentizované šifrovanie s asociovanými /dodatočnými dátami (AEAD – authenticated encryption with associated data). Taká konštrukcia pripúšťa dva typy vstupných dát – jeden, pre ktorý zabezpečí dôvernosť aj autentickosť údajov, a druhý, pre ktorý zabezpečí iba autentickosť bez dôvernosti.

1.2 Prúdové šifry

Prúdové šifry sú konštruované najčastejšie ako generátor pseudonáhodného prúdu bitov pripočítavaného modulo 2 (teda operácia XOR) k bitom vstupných dát do zašifrovaného textu. Pri dešifrovaní je pripočítaný rovnaký prúd bitov k zašifrovanému textu, pozri obrázok 5.



Obr. 5: Synchrónna prúdová šifra

Špecializované prúdové šifry majú obvykle jednoduchšiu štruktúru ako blokové šifry a sú vhodné najmä pre hardvérovú implementáciu. Príkladom prúdovej šifry je Snow 3G, ktorý je základom pre niektoré algoritmy zabezpečujúce dôvernosť a integritu údajov v mobilných LTE sieťach. Prúdová šifra optimalizovaná z pohľadu softvérovej implementácie je ChaCha, ktorá je najmä vo variante ChaCha20 používaná vo viacerých aplikáciách (napr. aj ako jedna z možností v TLS protokole). Atraktivita ChaCha20 spočíva v tom, že pri porovnateľnej úrovni bezpečnosti môže dosahovať vyšší výkon ako AES na platformách, ktoré nemajú k dispozícii hardvérovú akceleráciu pre AES.

Keďže blokové šifry je možné vhodným módom (napr. OFB, CTR alebo CFB) použiť aj ako prúdové šifry, v praxi sú väčšinou používané práve blokové šifry, na ktoré sa zameriavajú aj rôzne štandardizačné aktivity.

1.3 Hašovacie funkcie

Od univerzálne použiteľnej hašovacej funkcie požadujeme dve základné bezpečnostné vlastnosti:

1. Odolnosť vzoru: k danému odtlačku nie je efektívne možné vypočítať vstup s takýmto odtlačkom.
2. Odolnosť voči kolíziám: nie je efektívne možné vypočítať dva rôzne vstupy s rovnakým odtlačkom.

Kolízie pre ľubovoľnú hašovaciu funkciu možno hľadať tzv. „narodeninovým“ útokom. Tento útok vytvorí odtlačky veľkého počtu rôznych správ/dokumentov a následne hľadá medzi odtlačkami aspoň jednu dvojicu rovnakých. V prípade, že odtlačok hašovacej funkcie má dĺžku n bitov, tak zložitosť útoku je $\sim 2^{n/2}$. Pripomeňme, že pre ľubovoľnú symetrickú šifru možno hľadať kľúče úplným preberaním v najhoršom prípade so zložitosťou $\sim 2^k$ (kde k je dĺžka kľúča v bitoch). Preto majú štandardizované hašovacie funkcie dĺžky odtlačkov zodpovedajúce dvojnásobku dĺžok kľúčov štandardizovaných symetrických šifier. Typickým príkladom je AES-128, AES-192, AES-256 vs. SHA-256, SHA-384, SHA-512.

V súčasnosti najpoužívanejšie hašovacie funkcie sú tie, ktoré patria do sady hašovacích funkcií SHA-2 [19], resp. SHA-3 [21]:

sada	hašovacie funkcie
SHA-2	SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256
SHA-3	SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE128, SHAKE256

Špecifické postavenie majú funkcie SHAKE128 a SHAKE256, ktoré majú voliteľnú dĺžku výstupu. Flexibilita konštrukcie, z ktorej vychádza SHA-3 štandard, umožňuje jej elegantné použitie aj pre výpočet autentizačných kódov správ (funkcia KMAC), paralelizovateľný výpočet odtlačkov dlhých vstupov (ParallelHash) a pod. [20]. Otázne je, či sa niektoré z týchto konštrukcií presadia výrazne v praxi.

Z hľadiska bezpečnosti je dôležité spomenúť, že v súčasnosti nie je potrebná migrácia z SHA-2 na SHA-3.

V rôznych aplikáciách možno stále naraziť na použitie predchádzajúceho štandardu SHA-1. Ide o hašovaciu funkciu s dĺžkou výstupu 160 bitov. Problém s hašovacou funkciou SHA-1 je ten, že je možné prakticky konštruovať kolízie. Prvá kolízia bola publikovaná v roku 2017, v podobe dvoch rôznych PDF dokumentov s rovnakými odtlačkami[4]. Neskôr boli objavené ďalšie vylepšenia útokov na SHA-1 vedúce ku kolíziám s útočníkom zvoleným prefixom [13]. Momentálne by sa hašovacia funkcia SHA-1 nemala používať v akýchkoľvek konštrukciách, ktoré vyžadujú odolnosť voči kolíziám. Jednou z takých konštrukcií sú podpisové schémy, používané napr. pri certifikátoch verejných kľúčov. Všetky hlavné webové prehliadače ukončili podporu SHA-1 v certifikátoch verejných kľúčov v priebehu roku 2017. Nasledujúca tabuľka sumarizuje vývoj podielu certifikátov webových serverov, ktorých podpisová schéma využíva SHA-1 resp. SHA-256[5]:

[4] https://shattered.io/ (november 2020)

[5] SSL Pulse, https://www.ssllabs.com/ssl-pulse/ (november 2020)

	01/2015	01/2016	01/2017	01/2018	01/2019
SHA-1	66.7%	13.2%	1.5%	0.0%	0.0%
SHA-256	33.3%	86.8%	98.4%	99.8%	99.8%

Z uvedeného možno konštatovať, že v súčastnosti sa pri podpisovaní certifikátov verejných kľúčov na webe takmer výlučne používa hašovacia funkcia SHA-256.

1.4 Autentizačné kódy správ

Najčastejšou konštrukciou autentizačných kódov správ je HMAC, ktorú je možné skonštruovať z ľubovoľnej hašovacej funkcie H takto:

$$\mathrm{HMAC}(k, m) = H((k \oplus \mathrm{opad}) \,||\, H((k \oplus \mathrm{ipad}) \,||\, m)),$$

kde k označuje symetrický kľúč a m správu, ktorej autentizačný kód počítame. Hodnoty ipad a opad sú konštanty, obvykle definované v konkrétnom štandarde. Operácia $\oplus$ označuje XOR a operácia $||$ označuje zreťazenie hodnôt. Napriek dvojitej aplikácii hašovacej funkcie H je výpočet HMAC v podstate (pre dlhšie správy) rovnako rýchly ako výpočet odtlačku správy, keďže vonkajšia aplikácia H sa vykoná už len na krátkom vstupnom reťazci.

Pre bezpečnosť HMAC nie je podstatná odolnosť použitej hašovacej funkcie (s „klasickou“ konštrukciou) voči kolíziám, takže napriek nájdeniu kolízií v SHA-1 je HMAC konštrukcia s touto hašovacou funkciou (momentálne) bezpečná.

Dĺžka výstupu HMAC konštrukcie je rovnaká ako dĺžka výstupu hašovacej funkcie. V praxi sa výstup HMAC niekedy skracuje tak, že sa zoberie len definovaný počet výstupných bitov. Výhodou takéhoto prístupu je menší objem prenášaných dát v situáciách, keď sa autentizačný kód počíta ku každému (potenciálne krátkemu) paketu. Napríklad IPsec umožňuje použiť HMAC-SHA1-96, čo je HMAC počítaný s použitím hašovacej funkcie SHA-1, kde zo 160 bitov dlhého výsledku je na výstup daných prvých 96 bitov. Skracovanie výstupu nemá vplyv na rýchlosť výpočtu HMAC, hoci objektívne znižuje bezpečnostné parametre algoritmu.

Autentizačné kódy správ je možné konštruovať aj z blokových šifier pomocou špecifických autentizačných módov (napr. CMAC). Taktiež niektoré hašovacie funkcie umožňujú konštruovať MAC jednoduchšie ako HMAC konštrukciou. Napríklad SHA-3 dovoľuje vypočítať MAC ako odtlačok s tým, že kľúč sa pripojí na začiatok správy (KMAC – KECCAK Message Authentication Code). Poznamenajme, že takáto konštrukcia s hašovacou funkciou SHA-1 alebo s ľubovoľnou funkciou zo sady SHA-2 by bola triviálne napadnuteľná[6].

2 Asymetrické konštrukcie

Bezpečnosť asymetrických konštrukcií je postavená na matematických problémoch, o ktorých predpokladáme, že nie sú efektívne riešiteľné. V súčasnosti sú najčastejšie používané problémy:

[6] K správe m a jej autentizačnému kódu by bolo možné dopočítať aj bez kľúča korektný autentizačný kód k ľubovoľnému predĺženiu pôvodnej správy, teda pre akúkoľvek správu $m \,||\, m'$.

1. Faktorizácia veľkých čísel – pre zadané n, ktoré je súčinom dvoch prvočísel p a q, je úlohou nájsť tieto prvočísla. O zložitosť tohto problému pre dostatočne veľké n sa opiera RSA schéma.

2. Diskrétny logaritmus – pre zadanú hodnotu $g^x \bmod p$[7], kde p je prvočíslo, g je vhodný prvok z $\{2, 3, \ldots, p-2\}$ a x je náhodné, je úlohou vypočítať x. O zložitosť tohto problému pre dostatočne veľké p sa opiera konštrukcia napr. DSA (Digital Signature Algorithm), Diffieho-Hellmanov protokol (pozri časť 3) a pod. Problém diskrétneho logaritmu možno sformulovať aj na iných matematických objektoch, nielen v modulárnej aritmetike. Častou, prakticky používanou oblasťou sú eliptické krivky a operácie s bodmi na eliptických krivkách. Algoritmy na výpočet diskrétneho logaritmu na eliptických krivkách majú väčšiu zložitosť ako tie, ktoré problém riešia v modulárnej aritmetike. To znamená, že na dosiahnutie rovnakej bezpečnosti stačí pri eliptických krivkách používať kratšie kľúče (pozri časť 5.1).

Shorov algoritmus objavený v roku 1996 umožňuje efektívne počítať faktorizáciu veľkých čísel aj riešiť problém diskrétneho logaritmu na kvantových počítačoch. Pre používané parametre asymetrických schém presahuje veľkosť potrebných kvantových počítačov technické možnosti súčasnosti. Také veľké kvantové počítače jednoducho zatiaľ nemáme k dispozícii. Keďže technický pokrok je ťažké predvídať a prechod na nové algoritmy si vyžiada nejaký čas, NIST sa rozhodol štandardizovať tzv. postkvantové kryptografické algoritmy postupom[8], ktorý bol úspešne použitý pri výbere algoritmov pre štandardy AES a SHA-3. V súčasnosti prebieha analýza, spresňovnie a výber vhodných algoritmov v kategóriách asymetrické šifrovanie (šifrovanie s verejným kľúčom) určené na zapúzdrenie symetrických kryptografických kľúčov (KEM – key encapsulation mechanism) a podpisové schémy. Zavŕšenie výberu sa predbežne očakáva v roku 2022. Matematické problémy používané pri konštrukciách postkvantových schém sú z oblasti mriežok, teórie kódovania a pod.

2.1 Asymetrické šifrovanie

Asymetrické šifrovanie je najčastejšie používané v hybridných schémach na šifrovanie symetrických kľúčov. Často používanou asymetrickou schémou je RSA (navyše sa najjednoduchšie prezentuje), ktorej „učebnicovú“ verziu uvádzame:

- Inicializácia: zvolíme dve veľké rôzne prvočísla p, q a vypočítame verejný modul $n = p \cdot q$. Následne zvolíme hodnotu e a dopočítame d tak, aby platil vzťah $e \cdot d \equiv 1 \pmod{(p-1)(q-1)}$[9].
- Verejný kľúč je dvojica (e, n). Súkromný kľúč je hodnota d, niekedy nazývaná súkromný exponent.
- Šifrovanie je definované pre m z množiny $\{0, 1, \ldots, n-1\}$ takto: $E(m) = m^e \bmod n$.
- Dešifrovanie zašifrovaných údajov c sa vykoná s pomocou súkromného exponentu nasledovne: $D(c) = c^d \bmod n$.

Bez dopadu na bezpečnosť schémy je možné hodnotu e, nazývanú aj verejný exponent, zvoliť ako krátke číslo, čo má priaznivý vplyv na rýchlosť verejnej operácie v RSA. Najčastejšou voľbou

[7] Operácia mod p označuje výpočet celočíselného zvyšku po delení p, napr. $17 \bmod 5 = 2$, $2^{11} \bmod 13 = 2048 \bmod 13 = 7$.

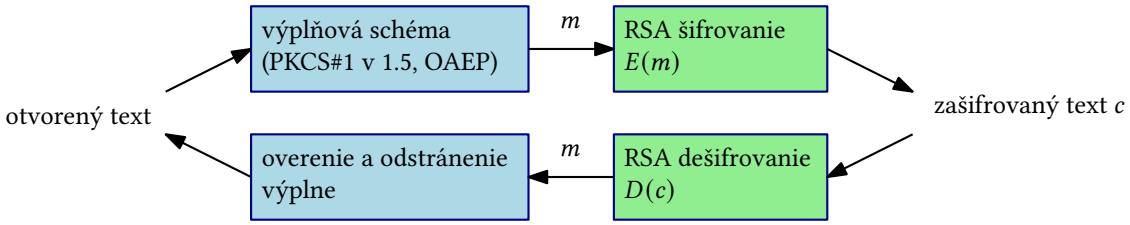
[8] https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization (november 2020)

[9] Symbol $\equiv$ znamená, že $e \cdot d$ má po celočíselnom delení hodnotou $(p-1)(q-1)$ zvyšok 1.

býva $e = 65537 = (10000000000000001)_2$, vďaka vhodnej binárnej reprezentácii čísla. Pokiaľ sa hovorí o dĺžke RSA kľúča, napr. 2048 alebo 4096 bitov, myslí sa dĺžka n. Navyše, rovnakú dĺžku má takmer vždy aj súkromný exponent d.

Dôsledkom rozlične dlhých exponentov je podstatne rýchlejšia verejná RSA transformácia ako súkromná transformácia (pozri časť 6.1). V praxi sa dešifrovanie urýchľuje alternatívnym výpočtom s využitím matematických vlastností modulárnej aritmetiky. To si vyžaduje pamätať dodatočné hodnoty ako súčasť súkromného kľúča, preto je dátová štruktúra obsahujúca súkromný RSA kľúč obvykle „bohatšia“. Príklad vytvorenia RSA inštancie ako aj jej použitie na asymetrické šifrovanie je uvedený v prílohe.

Priamočiare použitie RSA schémy vyššie popísaným spôsobom sa z bezpečnostných dôvodov neodporúča. Problémy sú napríklad determinizmus schémy, umožňujúci komukoľvek testovať kandidátske otvorené texty, výpočet krátkeho otvoreného textu pre malé e, tzv. „meet in the middle“ útok pre urýchlenie hľadania otvoreného textu oproti úplnému preberaniu a pod. Praktické RSA šifrovanie používa vhodnú výplňovú schému (padding). Obvykle používanými schémami sú staršia PKCS#1 v1.5 a novšia OAEP (Optimal Asymetric Encryption Padding)[10]. OAEP má lepšie bezpečnostné vlastnosti a pri spracovaní otvoreného textu pred samotnou verejnou RSA transformáciou využíva ďalšie kryptografické konštrukcie (hašovaciu funkciu, pseudonáhodný generátor). Samozrejme, pri dešifrovaní je po súkromnej RSA transformácii ešte potrebné na získanie pôvodných dát odstrániť vplyv výplňovej schémy, pričom sa správnosť výplne skontroluje (pozri obrázok 6).



Obr. 6: Použitie RSA s výplňovou schémou

2.2 Podpisové schémy

Podobne ako v prípade asymetrického šifrovania, učebnicová RSA podpisová schéma je jednoduchá. Použijeme rovnaké označenia pre RSA schému ako v predchádzajúcej časti. Podpisovaný dokument označíme m a hašovaciu funkciu H. Podpisovanie odtlačku dokumentu $H(m)$ je realizované s pomocou súkromného kľúča takto: $s = H(m)^d \bmod n$, kde s je výsledný podpis. Keďže ide o súkromnú RSA transformáciu, možno využiť rovnaké urýchľovanie ako pri RSA dešifrovaní. Overenie podpisu s k dokumentu m spočíva v porovnaní hodnôt $H(m)$ a $s^e \bmod n$, pričom používame verejný kľúč tvorcu podpisu. Podpis je korektný, ak sú obe hodnoty rovnaké.

V praxi sa opäť používajú vhodné výplňové schémy. Najznámejšími a najpoužívanejšími výplňovými schémami sú PKCS#1 v1.5 pre podpisy (zdôraznime, že je to iná schéma ako pri šifrovaní) a novšia PSS (Probabilistic Signature Scheme)[11]. RSA-PSS opäť využíva aj ďalšie kryptografické konštrukcie

[10] Obe sú definované napr. v Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.2, RFC 8017, 2016.

[11] Obe definované v RFC 8017.

(hašovaciu funkciu, pseudonáhodný generátor). Príklad použitia podpisovej RSA schémy je uvedený v prílohe.

Napriek tomu, že z matematického pohľadu nič nebráni používať rovnakú inštanciu RSA (teda hodnoty e, n, d a ďalšie) v podpisovej schéme aj na účely asymetrického šifrovania, takéto použitie sa neodporúča.

Podpisová schéma RSA (s oboma výplňovými schémami spomínanými vyššie) je spolu s DSA a ECDSA súčasťou štandardu [5]. V súčasnosti sú v praxi používané najmä RSA a ECDSA. Za zmienku stojí fakt, že v návrhu nového štandardu [6] je schéma DSA vypustená a sú doplnené niektoré ďalšie varianty podpisových schém na eliptických krivkách.

Nielen v akademickej sfére pri výskume, ale aj pri štandardizácii kryptografických schém je zreteľný príklon ku konštrukciám, ktorých bezpečnosť je možné matematicky dokázať. Samozrejme, dôkazy majú svoje predpoklady (o zložitosti niektorých problémov alebo o tom, aké vlastnosti majú komponenty použité v analyzovanej konštrukcii, napr. hašovacie funkcie alebo pseudonáhodné generátory) a počítajú s konkrétnym modelom útočníka. To znamená, že aj takéto konštrukcie môžu byť napadnuté, ak sa ukáže nejaký predpoklad nepravdivý, napr. vinou nevhodnej implementácie, alebo útočník postupuje iným spôsobom ako predpokladal model. Na druhej strane konštrukcie s explicitne sformulovanými predpokladmi a dôkazmi vzbudzujú väčšiu dôveru ako ad-hoc konštrukcie, kde takéto dôkazy absentujú. V tejto súvislosti je povšimnutiahodné, že prvý dôkaz bezpečnosti RSA podpisovej schémy s výplňou PKCS#1 v1.5 bol nájdený v roku 2018 [11] (RFC 2313, ktoré pôvodne PKCS#1 v1.5 definovalo, je z roku 1998).

3 Protokoly na dohodnutie kľúča

Úlohou protokolov na dohodnutie kľúča (niekedy nazývaných aj protokoly na výmenu, prípadne distribúciu kľúča) je ustanoviť medzi komunikujúcimi stranami kryptografické kľúče a iné parametre, ktoré budú následne použité na ochranu prenášaných údajov šifrovaním, výpočtom autentizačných kódov a pod. V praxi majú tieto protokoly obvykle aj cieľ vzájomne autentizovať jedného alebo oboch účastníkov. Kľúče dohodnuté pomocou týchto protokolov sú označované ako kľúče spojenia (session keys), keďže ich platnosť je zvyčajne obmedzená na jedno komunikačné spojenie a pri ďalšom spojení v budúcnosti si účastníci dohodnú nové kľúče spojenia.

Diffieho-Hellmanov protokol (ďalej „DH protokol“) slúži na dohodnutie kľúča a vo svojej pôvodnej podobe je bez autentizácie. Priebeh protokolu pre účastníkov A a B je nasledujúci:

1. $A \rightarrow B$: $p, g, g^x \bmod p$,
   kde p je veľké prvočíslo, g je vhodné číslo z množiny $\{2, 3, \ldots, p-2\}$ a x je náhodne zvolené. V prípade, ak sú p a g vopred dohodnuté parametre, nie je potrebné ich prenášať.

2. $B \rightarrow A$: $g^y \bmod p$,
   kde y je náhodne zvolené.

3. A vypočíta hodnotu $K = (g^y)^x = g^{xy}$ a B vypočíta rovnakú hodnotu $K = (g^x)^y = g^{xy}$ (v oboch prípadoch rátajúc modulo p), z ktorej následne môžu obaja odvodiť symetrické kľúče pre šifrovanie, pre výpočet autentizačných kódov a pod.

DH protokol je okrem počítania v modulárnej aritmetike možné sformulovať a implementovať aj na eliptických krivkách. Bezpečnosť DH protokolu pri pasívnom útočníkovi, ktorý odpočúva ale nezasahuje do komunikácie medzi A a B, sa opiera o predpoklad, že pre vhodné p a g nie je možné z hodnôt $g^x \bmod p$ a $g^y \bmod p$ efektívne vypočítať hodnotu K. Pokiaľ však uvažujeme o útočníkovi, ktorý môže prenášané správy v protokole meniť, je možné na DH protokol útočiť tzv. MITM „man in the middle“ útokom (útočníka v popise označíme M):

1. $A \longrightarrow M(B)$: $p, g, g^x \bmod p$
   M zachytí správu určenú pre B
2. $M \longrightarrow B$: $p, g, g^z \bmod p$
   M pošle B namiesto toho inú správu, kde si z zvolil sám
3. $B \longrightarrow M(A)$: $g^y \bmod p$
   M zachytí správu určenú pre A
4. $M \longrightarrow A$: $g^w \bmod p$
   M pošle A namiesto toho inú správu, kde si w zvolil sám
5. A vypočíta hodnotu $K_A = (g^w)^x = g^{xw}$ a B vypočíta hodnotu $K_B = (g^z)^y = g^{yz}$ (v oboch prípadoch modulo p), a teda s vysokou pravdepodobnosťou každý odvodí iné kľúče. Útočník M vie vypočítať obe hodnoty takto: $K_A = (g^x)^w = g^{xw}$ a $K_B = (g^y)^z = g^{yz}$. Následne dokáže M komunikovať s A aj B, v ich vzájomnú komunikáciu môže čítať a prešifrovávať, prípadne aj do nej zasahovať.

DH protokol je základom pre dohodnutie kľúča v mnohých prakticky používaných protokoloch, pričom tieto obvykle používajú varianty DH protokolu tak, aby zamedzili MITM útoku:

- TLS 1.2 (RFC 5246 a nadväzné RFC) je v súčasnosti najpoužívanejšia verzia TLS protokolu[12]. Špecifikácia obsahuje nasledujúce varianty DH protokolu:
  - `DH_anon` – anonymný DH protokol bez autentizácie, zraniteľný na MITM útok. Pri konfigurácii webových služieb je tento variant štandardne zakázaný.
  - `DHE_RSA`, `DHE_DSS` – server generuje parametre p, g, pričom tieto spolu so svoju hodnotu $g^x \bmod p$ (pre náhodné x) podpíše s použitím podpisovej RSA schémy alebo s použitím DSA algoritmu (ten je súčasťou štandardu DSS). Server v takomto prípade má certifikát verejného kľúča, ktorý pošle klientovi a ten následne môže podpis overiť.
  - `DH_RSA`, `DH_DSS` – v tomto prípade sú parametre DH protokolu súčasťou certifikátu servera. Suffixy `_RSA` a `_DSS` sú uvádzané z historických dôvodov, klient môže prípustné podpisové schémy pre certifikát servera signalizovať v rozšírení TLS protokolu. Podotknime, že tieto metódy sú v praxi podporované málokedy.

  Okrem uvedených variantov DH protokolu existujú aj varianty postavené na eliptických krivkách – v takom prípade sú označené prefixom `EC`, napr. `ECDHE_RSA`.

[12] Podľa štatistiky SSL Pulse (https://www.ssllabs.com/ssl-pulse/) bol podiel web serverov podporujúcich TLS 1.2 v októbri 2020 99% a podiel web serverov s podporou TLS 1.3 približne 40%.

Ďalšou možnosťou pri dohodnutí kľúča v TLS 1.2 je RSA metóda, kde klient zašifruje náhodne zvolenú hodnotu s použitím verejného RSA kľúča servera (verejný kľúč servera je súčasťou certifikátu). Hlavným nedostatkom je fakt, že tento spôsob nezabezpečuje pre dohodnuté kľúče vlastnosť dopredného utajenia (forward secrecy, niekedy perfect forward secrecy). Diskusia k nej je na konci tejto časti.

- TLS 1.3 (RFC 8446) – oproti TLS 1.2 bol zredukovaný počet metód, akými je možné dohodnúť kľúče spojenia. Vo väčšine bežných situácií bude pri nadväzovaní nového spojenia použitý nejaký variant DH protokolu (v modulárnej aritmetike alebo nad eliptickými krivkami), s podpísaním parametrov serverom.

- IPsec – pre vzájomnú autentizáciu a dohodnutie kľúča sa používa protokol IKE (Internet Key Exchange), v staršej a novšej verzii: IKEv1 a IKEv2. V oboch prípadoch prebieha dohodnutie kľúča pomocou DH protokolu, pričom autentizácia je vykonaná prostredníctvom šifrovania alebo podpisov s využitím verejných kľúčov/certifikátov, autentizačných kódov s využitím spoločného tajomstva (tzv. „preshared secret“) alebo v prípade IKEv2 aj prostredníctvom vhodnej EAP metódy (EAP – Extensible Authentication Protocol).

- SSH 2 (Secure Shell, RFC 4253 a nadväzné RFC) – jednou z metód na dohodnutie kľúča je použitie DH protokolu s tým, že server svoje parametre podpisuje, čím sa zároveň zabezpečuje ich autentickosť.

- WPA3 (Wi-Fi Protected Access) – štandard pre bezpečnosť Wi-Fi publikovaný v roku 2018 obsahuje v rámci WPA3-Personal protokol SAE (Simultaneous Authentication of Equals, variant Dragonfly protokolu), ktorý rieši dohodnutie kľúčov spojenia a vzájomnú autentizáciu komunikujúcich strán. SAE využíva modifikovný DH protokol, s úpravami potrebnými pre zabezpečenie autentickosti komunikácie prostredníctvom spoločného kľúča (hesla) a zároveň ochrany pred offline slovníkovým útokom.

Forward secrecy. Táto vlastnosť protokolov na dohodnutie kľúča znamená, že súčasné kľúče spojenia nie sú bezprostredne kompromitované ani keď útočník v budúcnosti získa tajné parametre – či už budúce kľúče spojenia alebo súkromné kľúče účastníkov. Samozrejme, forward secrecy nechráni pred nájdením nových útokov na použité kryptografické konštrukcie, napr. symetrické šifry, alebo na riešenie problémov využitých pri konštrukcii protokolu.

Ilustrujme vlastnosť forward secrecy na príkladoch. Nech v protokole účastník A vygeneruje kľúč spojenia a pošle ho zašifrovaný RSA schémou druhému účastníkovi B (s využitím verejného kľúča B). Tento postup nemá forward secrecy vlastnosť. Útočník, ktorý odpočuje zašifrovaný kľúč spojenia, ho dokáže ľahko dešifrovať, ak kedykoľvek v budúcnosti získa súkromný kľúč účastníka B. Na druhej strane v DH protokole je kľúč spojenia odvodený z hodnoty $K = g^{xy}$, pričom x aj y sú v každom behu protokolu volené nanovo a náhodne. To znamená, že hodnoty K v jednotlivých behoch DH protokolu sú navzájom nezávislé a prezradenie niektorej z nich nevedie ku kompromitácii ostatných. Súkromné kľúče, ktoré sú prípadne použité v podpisovej schéme na zabezpečenie autentickosti niektorých správ DH protokolu, nemajú žiadny súvis s dôvernosťou výsledných kľúčov spojenia. Teda ani prezradenie takýchto súkromných kľúčov nevedie ku kompromitácii predchádzajúcich kľúčov spojenia.

4 Infraštruktúra verejných kľúčov

Základom pre získanie certifikátu verejného kľúča (ďalej len „certifikát“) je vytvorenie páru kryptografických kľúčov pre asymetrickú schému a súboru s požiadavkou na vydanie certifikátu. Požiadavka je vo formáte CSR (Certificate signing request) a pripája sa k žiadosti o vydanie certifikátu. Súčasťou CSR sú informácie o subjekte a ďalšie aribúty potrebné pre následné využitie certifikátu, napríklad doménové meno pre web server. Certifikačné autority poskytujú vlastné nástroje uľahčujúce konštrukciu CSR, prípadne návody na správne vygenerovanie CSR pre najčastejšie používané serverové platformy. Napríklad v IIS 10 možno použiť IIS Manager, v Exchange 2019 Exchange Admin Center, pre Apache obvykle openssl, pre Tomcat nástroj keytool, atď. CSR okrem verejného kľúča a atribútov, ktoré sa majú ocitnúť v certifikáte, obsahuje aj podpis týchto dát vytvorený s použitím súkromného kľúča. Vďaka tomu je zrejmé, že tvorca CSR pozná súkromný kľúč a atribúty neboli zmenené (identitu subjektu však musí overiť registračná/certifikačná autorita inak). Príklad vytvorenia CSR pomocou openssl je uvedený v prílohe.

Certifikačné autority zverejňujú pravidlá a postupy svojej činnosti v tzv. certifikačnom poriadku (Certification Practice Statement – CPS). Popis vydávania certifikátov, archivácie záznamov, zneplatňovania a obnovy certifikátov, bezpečnostných opatrení a ďalších činností napomáhajú zvyšovať dôveru v certifikačnú autoritu (ďalej CA). Dôvera býva obvykle umocnená nezávislým auditom CA[13].

Certifikáty sa líšia aj tým, aké údaje a akým spôsobom CA overuje. Obvykle sa možno stretnúť s nasledujúcimi typmi certifikátov:

- Certifikáty s overením domény (Domain Validation, DV) – CA overí len vlastníctvo doménového mena, pre ktorý má vydať certifikát. Známou CA, ktorá vydáva výlučne certifikáty tohto typu je Let’s Encrypt.
- Certifikáty s overením organizácie (Organization Validation, OV) – CA overí doménové meno, názov spoločnosti a prípadne ďalšie náležitosti, aby sa uistila, že certifikát bude obsahovať správne informácie.
- Certifikáty s rozšíreným overením (Extended Validation, EV) – CA dôkladnejšie overuje identitu o spoločnosti, doménových menách a pod. Pri EV certifikátoch sa napríklad overuje aj vlastníctvo ku každému doménovému menu, preto nemôžu byť vydané „hviezdičkové“ certifikáty s EV. Príslušné pravidlá pre vydávanie EV certifikátov definuje CA/Browser Forum, dobrovoľná organizácia združujúca certifikačné autority a tvorcov webových prehliadačov[14]. EV certifikáty sa od OV certifikátov formálne líšia špecifickým atribútom v certifikáte (OID 2.23.140.1.1).

Webové prehliadače v stavovom riadku pri URL adrese v minulosti vizuálne odlišovali TLS spojenia, podľa toho, či má server EV alebo iný certifikát. Samozrejme hovoríme o dôveryhodnom certifikáte, overiteľnom prehliadačom až po explicitne dôveryhodnú CA, v opačnom prípade skončí pokus o nadviazanie spojenia chybou. TLS spojenia s EV certifikátom zobrazovali okrem symbolu zámku aj názov organizácie, pre ktorú bol certifikát vydaný. Ukázalo sa, že takéto bezpečnostné indikátory používatelia ignorujú a na ich správanie nemajú reálny vplyv [24]. Vzhľadom na otázny prínos pre bezpečnosť všetky významné webové prehliadače postupne upustili od používania týchto indikátorov.

[13] Napr. zoznam koreňových CA pre Mozilla Firefox (prirodzene, prienik so sadou CA v iných prehliadačoch je značný), vrátane odkazov na výsledky príslušných auditov týchto CA možno nájsť na https://ccadb-public.secure.force.com/mozilla/IncludedCACertificateReport (november 2020).

[14] Pravidlá sú k dispozícii na https://cabforum.org/extended-validation (november 2020).

CRL a OCSP. Štandardné spôsoby ako overiť, či bol certifikát zneplatnený počas intervalu platnosti sú CRL (Certificate Revocation List) a OCSP (Online Certificate Status Protocol). V ideálnom prípade by pred použitím certifikátu mala byť jeho platnosť okrem ostatných testov overená aj voči CRL alebo pomocou OCSP. Aplikácia potom musí riešiť situáciu, keď tieto mechanizmy nie sú dostupné – pokračovať bez overenia alebo nepokračovať vôbec?

Adresy, na ktorých je možné nájsť CRL alebo službu OCSP sú uvedené v certifikáte. CRL sú pre koncové (klientske) certifikáty vydávané zvyčajne denne – interval je definovaný v certifikačnom poriadku konkrétnej CA. Zoznam sériových čísel zneplatnených certifikátov v CRL obsahuje aj dôvody zneplatnenia. Na zabezpečenie autentickosti je CRL podpísaný certifikačnou autoritou. Pri využívaní CRL je podstatné mať aktuálnu verziu CRL a overiť jeho autentickosť.

OCSP je alternatívny spôsob overenia predčasného zneplatnenia certifikátov. Výhodou oproti CRL je menší objem prenášaných údajov (keďže klient sa pýta na jeden konkrétny certifikát) a teoreticky čerstvá[15], prakticky takmer čerstvá odpoveď o stave certifikátu. Odpoveď je podpísaná certifikačnou autoritou.

Problematika overovania certifikátov je najmä prehliadaní webu zložitejšia. Niektoré servery implementujú tzv. OCSP stapling, kde pridávajú už pri nadväzovaní spojenia odpoveď z OCSP (ktorú pravidelne aktualizujú), takže klient nemusí realizovať vlastný OCSP. Z hľadiska ochrany súkromia je ďalšou výhodou tohto prístupu to, že CA nie je informovaná, kam klient pristupuje. Prehliadače pristupujú k OCSP a CRL rôzne, napr. Chrome štandardne nevaliduje prostredníctvom CRL a OCSP[16], ale používa vlastný zoznam zneplatnených certifikátov CRLSets. Firefox okrem štandardných mechanizmov udržuje aj samostatný zoznam zneplatnených certifikátov OneCRL, určený najmä pre certifikáty nekoreňových CA.

HPKP a Certificate Transparency. Používanie infraštruktúry verejných kľúčov v prostredí webu prinieslo viaceré praktické problémy. Kompromitácia CA alebo tzv. registračných autorít, prípadne podvodné konanie pri vydávaní certifikátov umožní útočníkovi nechať si vydať z pohľadu používateľov platné certifikáty pre webové servery. Takéto bezpečnostné problémy sú častokrát detegované s veľkým oneskorením. Server s platným certifikátom, spravovaný útočníkom, je využiteľný na odchytenie prihlasovacích údajov používateľov alebo na odpočúvanie kompletnej komunikácie s cieľovým serverom.

HTTP Public Key Pinning (HPKP, RFC 7469) rieši tento problém pomocou „prišpendlenia“ verejných kľúčov. Základná myšlienka je nasledovná: klient (webový prehliadač) je inštruovaný serverom, aby si na definované obdobie zapamätal odtlačok jedného alebo viacerých verejných kľúčov vyskytujúcich sa v reťazi certifikátov od certifikátu servera až po certifikát koreňovej CA. Počas tohto obdobia klient odmietne pripojenie na tento server, ak by reťaz certifikátov obsahovala verejný kľúč s iným odtlačkom. HPKP ponúka ochranu za predpokladu, že pri prvom prístupe komunikujeme s legitímnym serverom. Praktické nasadenie HPKP je pomerne zložité a musí brať do úvahy potrebu meniť certifikát servera. Nevhodná konfiguračná operácia, napr. prišpendlenie nevhodného certifikátu, môže mať za následok nedostupnosť webového servera pre používateľov. Ďalším problémom je nepriateľské prišpendlenie, keď útočník s podvodne získaným platným certifikátom oznámi klientom ním zvo-

[15] pokiaľ klient aj OCSP server podporujú špecifické rozšírenie protokolu (nonce extension, RFC 6960)

[16] „Online (i.e. OCSP and CRL) checks are not, generally, performed by Chrome.“ https://dev.chromium.org/Home/chromium-security/crlsets (november 2020)

lenú konfiguráciu HPKP. Aj z týchto dôvodov napr. Chrome upustil od HPKP v roku 2018, ostatné prehliadače neskôr tento krok nasledovali.

V súčasnosti je hlavným spôsobom detekcie podvodne alebo chybne vydaných certifikátov Certificate Transparency. Ide o systém verejne prevádzkovaných služieb spravujúcich zoznamy vydaných certifikátov (certificate logs[17]), do ktorých je možné certifikáty len pridávať a nie je ich možné dodatočne meniť. Tieto vlastnosti sú zabezpečená kryptograficky (s využitím tzv. Merkleho hašovacích stromov) a navyše je možné verejne overiť, že server prevádzkujúci takúto službu sa správa v súlade s predpísanými pravidlami. Primárnymi prispievateľmi certifikátov do týchto zoznamov sú certifikačné autority. Používatelia, presnejšie webové prehliadače, môžu pri nadviazaní TLS spojenia požadovať, aby sa serverom prezentovaný certifikát nachádzal v jednom alebo viacerých zoznamoch – presvedčí ich o tom časová pečiatka certifikátu podpísaná prevádzkovateľom služby (Signed Certificate Timestamp, SCT). Tým sú CA a prevádzkovatelia webových serverov tlačení do publikovania certifikátov v zoznamoch. Následne môžu CA, prevádzkovatelia alebo ktokoľvek iný aktívne monitorovať svoje certifikáty a pružnejšie reagovať pri identifikácii podvodných certifikátov. Poznamenajme, že Certificate Transparency je primárne o včasnej identifikácii problematických certifikátov a nenahrádza štandardné mechanizmy pre zneplatňovanie certifikátov. Autorom a hlavným proponentom projektu Certificate Transparency je Google a podporu v súčasnosti možno nájsť vo viacerých prehliadačoch[18].

5 Kryptoanalýza a bezpečnosť kryptografických konštrukcií

Každá kryptografická konštrukcia je náchylná na tzv. generické útoky, ktoré nezávisia na podrobnostiach a kvalitách konštrukcie. Typickým príkladom je útok úplným preberaním (niekedy nazývaný aj útok hrubou silou) na symetrické šifrovanie, keď útočník vyskúša postupne všetky potenciálne kľúče. V takom prípade nezáleží na tom, či je použitá šifra AES alebo iná – útok sa dá realizovať vždy. Navyše, čím rýchlejšia šifra, tým rýchlejšie bude aj preberanie kľúčov. Zdôraznime, že generický útok je z hľadiska útočníka najhorší možný. Pri ľubovoľnej slabine kryptografickej konštrukcie, nevhodnej implementácii alebo napríklad pri zlom spôsobe voľby kľúčov môže byť útok efektívnejší. Teda kvalitné kryptografické konštrukcie a ich implementácie sa snažia dosiahnuť, aby bol generický útok zároveň najlepším útokom, ktorý má útočník k dispozícii. Tabuľka 1 sumarizuje generické útoky na základné kryptografické konštrukcie a v niektorých prípadoch uvádza aj ich asymptotickú časovú zložitosť.

Kryptológia pri definícii bezpečnosti konštrukcií a ich analýze obvykle uvažuje s čo najsilnejším útočníkom. To znamená, že napríklad (neformálne a zjednodušene):

- Pri šifrovacích schémach očakávame, že útočník sa zo zašifrovaných dát c nedozvie o ich dešifrovanej podobe nič, napriek tomu, že budeme predpokladať schopnosť útočníka nechať si dešifrovať akýkoľvek zašifrovaný text (samozrejme s výnimkou c), resp. nechať si zašifrovať ľubovoľné vstupné dáta. Prirodzene, pri asymetrických šifrách môže ktokoľvek, nielen útočník, šifrovať ľubovoľné dáta, keďže tam je príslušný kľúč verejný.

- Pri podpisových schémach predpokladáme schopnosť útočníka nechať si podpísať ľubovoľnú, ním zvolenú správu; napriek tomu očakávame, že útočník nedokáže vytvoriť korektný podpis k nejakej správe na ktorej podpis sa nepýtal.

[17] Na stránkach Googlu sú preložené ako „verejné denníky transparentnosti certifikátov“.
[18] Podrobnosti o Certificate Transparency možno nájsť na https://www.certificate-transparency.org/ (november 2020).

Konštrukcia	Generický útok (k dĺžka kľúča, n veľkosť odtlačku/výstupu)
Symetrická šifra	Prehľadávanie priestoru všetkých kľúčov $\sim 2^k$
Hašovacia funkcia	Hľadanie kolízií: narodeninový útok $\sim 2^{n/2}$ Hľadanie vzoru: prehľadanie a vyskúšanie vzorov $\sim 2^n$
Autentizačné kódy	Prehľadávanie priestoru všetkých kľúčov $\sim 2^k$, resp. uhádnutie korektného autentizačného kódu k správe $\sim 2^n$.
Asymetrická šifra	Riešenie konkrétneho ťažkého problému (faktorizácia, výpočet diskrétneho logaritmu a pod.)
Podpisová schéma	Riešenie konkrétneho ťažkého problému, resp. útok na hašovaciu funkciu.

Tabuľka 1: Generické útoky na základné kryptografické konštrukcie

Konštrukcie bezpečné pri veľmi silnom útočníkovi potom zostanú bezpečné aj v prípade scenára so slabším útočníkom.

5.1 Ekvivalentné dĺžky kľúčov

Pri používaní viacerých kryptografických konštrukcií je vhodné používať také dĺžky kľúčov, aby zložitosť útoku na každú použitú konštrukciu bola približne rovnaká. Existujú rôzne analýzy a odporučenia popisujúce ekvivalentné dĺžky kľúčov[19] a odporučenia na voľbu dĺžok kľúčov podľa citlivosti chránených údajov alebo potrebnej doby ich ochrany. Uveďme odporúčané dĺžky zo správy projektu ECRYPT CSA [1]. Všetky údaje v tabuľke sú uvedené v bitoch a sú to minimálne, navzájom ekvivalentné dĺžky parametrov.

Bezpečnosť	Symetrický kľúč	Hašovacia funkcia	RSA modul	Eliptická krivka
krátkodobá (aspoň 10 rokov)	128	256	3072	256
dlhodobobá (30–50 rokov)	256	512	15360	512

V praxi je použitá dĺžka kľúčov diktovaná hlavne tým, aké algoritmy a dĺžky kľúčov podporujú štandardné kryptografické knižnice/aplikácie. V prípade certifikátov verejných kľúčov sú dĺžky ovplyvnené tým, aké verejné kľúče je ochotná certifikovať vybraná certifikačná autorita. Predlžovanie kľúčov v certifikátoch zároveň zvyšuje výpočtovú zložitosť operácií a teda nároky na server, ktorý nadväzuje spojenia s veľkým počtom klientov (ilustračné výkonové charakteristiky sú uvedené v časti 6.1).

Ak sa pozrieme na 10 najnavštevovanejších web stránok v doméne .sk[20], nájdeme v certifikátoch 8 krát 2048 bitové RSA a 2 krát verejný kľúč pre schémy nad eliptickými krivkami (v oboch prípadoch pre štandardizovanú krivku P-256). Analogický zoznam Top 10 pre USA ukáže 6 krát použitý RSA-2048 a 4 krát verejný kľúč na eliptickej krivke P-256.

[19] Prehľad možno nájsť na stránke https://www.keylength.com/ (november 2020).
[20] podľa rebríčka popularity stránok spoločnosti Alexa k novembru 2020 (https://www.alexa.com/topsites)

5.2 Ukladanie hesiel a kľúčov

Heslá sú stále najčastejším spôsobom autentizácie používateľov. Pre zníženie dopadov incidentov ako sú napr. kompromitácia servera, zraniteľnosť aplikácie, získanie zálohy databázy informačného systému útočníkom, nevhodné aktivity „zvedavého“ správcu a pod., nie je vhodné používateľské heslá v aplikácii/systéme ukladať v otvorenom tvare. V opačnom prípade potom uvedené hrozby môžu viesť k získaniu celého zoznamu používateľských hesiel. Myšlienka bezpečného ukladania hesiel spočíva v použití vhodnej transformačnej funkcie na spracovanie hesla, označme ju T, pričom následne uložíme v aplikácii jej výsledok $T(\text{heslo})$. Často sa používa terminológia, že heslo sa „hašuje“ a $T(\text{heslo})$ je odtlačok hesla. Pri autentizácii sa používateľom zadané heslo transformuje danou funkciou a výsledok sa porovná s uloženou hodnotou. V prípade zhody je autentizácia používateľa úspešná.

Požadujeme, aby T mala vlastnosť odolnosti vzoru[21], v opačnom prípade by bolo možné z uniknutého zoznamu rekonštruovať pôvodné heslá alebo ich ekvivalentné náhrady. Na ukladanie hesiel sa zvyknú používať špeciálne navrhnuté funkcie ako napr. PBKFD2 (táto funkcia je pôvodne navrhnutá na odvodenie symetrických kryptografických kľúčov z hesla), bcrypt, scrypt alebo Argon2. Nezriedka sa možno stretnúť aj s viacnásobne iterovanými hašovacími funkciami – typickým príkladom sú súčasné distribúcie Linuxu. Dôležité bezpečnostné parametre funkcií pre ukladanie hesiel sú:

- Počítadlo iterácií – slúži na jednoduché riadenie rýchlosti transformačnej funkcie T. Tá je iteratívna a nastavením počítadla je možné výpočet spomaľovať na želanú úroveň. Ukladanie hesiel je jedným z príkladov, keď je vysoký výkon kryptografickej konštrukcie prekážkou bezpečnosti. Pokiaľ totiž útočník získa hodnotu $T(\text{heslo})$, dokáže testovať potenciálne heslá opakovaným výpočtom funkcie T pre rôzne heslá a následným porovnaním výsledku s $T(\text{heslo})$. Samozrejme, zvyšovanie počítadla spomaľuje aj bežné overovanie hesla v aplikácii. Avšak kým povedzme 1000 násobné spomalenie z 0,5 ms na 0,5 sekundy je pre používateľa pri prihlásení určite akceptovateľné, spomalenie útoku preberaním hesiel napríklad z 1 mesiaca na 1000 mesiacov (83,3 roka) robí tento útok nerealistickým.
- Soľ – zvyčajne náhodný reťazec pridávaný pri spracovaní hesla. Soľ je volená pre každého používateľa zvlášť a zabezpečuje, že rovnaké heslá sa pre rôznych používateľov transformujú do rôznych výsledkov. V opačnom prípade, teda bez soli, útočník dokáže testovať heslá paralelne pre všetky získané hodnoty $T = \{T(\text{heslo}_1), \ldots, T(\text{heslo}_r)\}$. Alternatívne dokáže útočník predvypočítať hodnoty často používaných hesiel a po získaní množiny T paralelne vyhľadávať zhodu. Použitie soli tieto útoky redukuje opäť na útoky na individuálne heslá, navyše bez možnosti predvýpočtu.

Hľadanie hesla zo získanej hodnoty $T(\text{heslo})$ skúšaním rôznych hesiel je úloha, ktorá sa ľahko rieši paralelne. V ostatnom období sa na takúto úlohu používajú grafické karty, disponujúce tisíckami jadier na jednej karte[22]. Tabuľka 2 ilustruje rýchlosť preskúšania všetkých hesiel z danej množiny pri orientačnej rýchlosti GPU Nvidia RTX 2080 Ti a rôznych transformačných funkciách. V tabuľke označuje [a-z] množinu 26 malých písmen bez diakritiky a [a-9] množinu malých písmen, veľkých písmen a cifier (dokopy 62 znakov).

Na sťaženie použitia grafických kariet, prípadne programovateľných hradlových polí (FPGA – Field-programmable gate array) alebo špecifických zákazníckych integrovaných obvodov (ASIC –

[21] Pripomeňme: z výstupu y nie je možné efektívne vypočítať x také, že $T(x) = y$.

[22] napr. Nvidia RTX 2080 Ti má 4352 jadier

	Funkcia			
	SHA-1 (1 iterácia)	SHA-512 (5000 iterácií)	NTLM	bcrypt (32 iterácií)
Rýchlosť	16000 MH/s	220 kH/s	76000 MH/s	28000 H/s
Zloženie hesla				
[a-z] dĺžka 8	13 sekúnd	11 dní	3 sekundy	86 dní
[a-9] dĺžka 8	3,8 hodiny	31 rokov	48 minút	247 rokov
[a-9] dĺžka 10	607 dní	121000 rokov	127 dní	950000 rokov

Tabuľka 2: Ilustrácia rýchlosti prebratia hesiel pre rôzne transformačné funkcie

Application-specific integrated circuit) sú navrhované transformačné funkcie, ktoré vyžadujú pri výpočte použitie dostatočne veľkej pamäte. Príkladom funkcií s parametrizovateľným využitím pamäte sú scrypt a Argon2.

Na záver pripomeňme, že ľubovoľný spôsob uloženia nezvýši odolnosť slabých hesiel. Databázy uniknutých hesiel umožňujú modelovať voľbu najčastejších používateľských hesiel nielen z hľadiska slovníkov hesiel ale aj z hľadiska používania rôznych „modifikácií“, zreťazovania slov, prefixov, sufixov a pod. Prax ukazuje, že väčšina používateľov volí heslá, ktorých nájdenie z uniknutej hodnoty $T(\text{heslo})$ je otázkou niekoľkých hodín – s technickým vybavením dostupným individuálnemu útočníkovi.

Ukladanie kľúčov

Bezpečnosť kryptografických konštrukcií podstatne závisí na uložení a používaní kľúčov. Jednou z možností je použitie tzv. hardvérových bezpečnostných modulov (Hardware Security Module), ktoré zároveň vykonávajú kryptografické operácie bez toho, aby súkromné alebo symetrické kľúče opustili modul v otvorenom tvare. Inak sú kľúče zvyčajne uložené v súbore, pričom jedným zo štandardných formátov je PKCS#12 (RFC 7292). Tento formát umožňuje ukladanie používateľských súkromných kľúčov, certifikátov, symetrických kľúčov a pod., pričom podporuje rôzne kombinácie módov pre dosiahnutie súkromia a integrity:

- Mód pre súkromie – použité šifrovanie prostredníctvom asymetrickej schémy, resp. použitím symetrického algoritmu s kľúčom odvodeným z hesla.
- Mód pre integritu – použitý autentizačný kód prostredníctvom HMAC s kľúčom odvodeným z hesla, resp. digitálny podpis prostredníctvom asymetrickej schémy.

Obvyklá kombinácia využíva symetrické mechanizmy odvádzajúce kľúče z používateľského hesla. Poznamenajme, že aj v týchto prípadoch sú pri odvodení kľúčov využívané soľ a počítadlo iterácií.

5.3 Implementačné a prevádzkové slabiny

Príčinou väčšiny útokov na kryptografické konštrukcie sú slabiny v správe kľúčov a slabiny v implementácii. V prípade protokolov je zvyčajne problém v samotnom protokole, teda v štruktúre a postupnosti prenášaných správ, bez ohľadu na kryptografickú kvalitu použitých algoritmov.

Bezpečná implementácia kryptografických konštrukcií nie je triviálna úloha. Vo všeobecnosti stačí jedna implementačná chyba (nedostatok) na narušenie bezpečnostných požiadaviek, kompromitáciu údajov alebo kľúčov. Situácia je komplikovaná tým, že niektoré implementačné nedostatky neovplyvňujú funkčnosť, teda nie sú „vidieť“ a používateľ ich nevníma. Uvedieme niekoľko príkladov.

Útoky postrannými kanálmi (side-channel attacks). Útoky tohto typu využívajú informácie získané z prostredia ovplyvneného kryptografickou operáciou na získanie kľúča alebo chránených údajov. Napríklad tzv. „timing“ útok využíva situáciu, keď čas výpočtu závisí na hodnote kľúča a spracúvaných údajoch. Ak použijeme štandardnú implementáciu RSA[23], tak štatistickým spracovaním väčšieho množstva vzoriek typu ⟨zašifrovaný text, čas dešifrovania⟩ možno získať súkromný RSA kľúč. Iné typy postranných kanálov môžu využívať elektrický príkon, elektromagnetické vyžarovanie, prístupy do vyrovnávacej pamäte (cache) a pod. Jednoduchší príklad postranného kanála je zvuk. Existujú experimenty, ktoré rekonštruujú text (heslo) na základe zvuku vydávaného stlačením jednotlivých kláves na klávesnici (napr. [9]), PIN kódy zadávané na platobných termináloch (napr. [18]), text písaný na virtuálnej klávesnici mobilného telefónu alebo tabletu (napr. [22]) a mnohé ďalšie. Iným príkladom sú útoky využívajúce vlastnosti pamätí DRAM, kde sa v istých prípadoch ovplyvňujú susedné pamäťové bunky (zraniteľnosť Rowhammer). Využitie tejto zraniteľnosti na získanie citlivých dát bolo demonštrované na získaní súkromného RSA kľúča v OpenSSH a bolo publikované pod názvom RAMBleed [12].

Zraniteľnosti spôsobené nesplnením predpokladov. Bezpečnosť kryptografických konštrukcií závisí na predpokladoch. Typickým predpokladom je napríklad náhodnosť niektorých parametrov v konštrukciách, počnúc generovaním samotných kľúčov, pokračujúc inicializačnými vektormi, parametrami výplňových schém a pod. Výskum v roku 2012 zistil, že 0,5% verejných RSA kľúčov v TLS certifikátoch na webe je možné ľahko faktorizovať a získať súkromný kľúč vďaka tomu, že mali spoločný faktor s iným verejným kľúčom [10] – išlo zväčša o „embedded“ zariadenia s nedostatočne inicializovaným generátorom náhodných čísel. Ďalším príkladom, ktorý sa dotkol aj občianskych preukazov s elektronickým čipom (eID) používaných v SR, je zraniteľnosť ROCA [15]. Nevhodná implementácia generovania prvočísel pre RSA schému viedla k možnosti vypočítať súkromný kľúč z verejného kľúča.

Slabiny v kryptografických protokoloch. Minulosť aj súčasnosť je dokladom toho, že bezpečné kryptografické protokoly je ťažké navrhnúť aj implementovať. Ilustratívnym príkladom je najpoužívanejší kryptografický protokol súčasnosti na webe: SSL/TLS. Od úvodnej verzie SSL v roku 1994 prešiel protokolu viacerými iteráciami a verziami, ktoré odstraňovali aj bezpečnostné slabiny. Prehľad niektorých slabín do roku 2013 možno nájsť v práci [14]. Útok s názvom ROBOT, umožňujúci vykonávať dešifrovanie alebo podpisovanie so súkromným kľúčom servera [3], je ukážkou toho, ako sa dá v niektorých implementáciách TLS zneužiť 19 rokov známa zraniteľnosť (len využitá trocha iným spôsobom). Ani novšie protokoly na dohodnutie kľúča nie sú imúnne voči slabinám. Napríklad Dragonblood útok na WPA3 [26] umožňujúci off-line slovníkový útok na heslo, KNOB útok na Bluetooth BR/EDR vedúci k dohodnutiu kľúča s entropiou 1 bajt [2], alebo Selfie útok na vzájomnú autentizáciu klienta a servera na základe spoločného tajomstva v TLS 1.3 [7].

Napriek uvedeným príkladom, podobne ako pri iných kryptografických konštrukciách, je vhodné používať štandardné protokoly s pravidelne udržiavanou implementáciou. Návrh vlastného kryptogra-

[23] teda bez ochrany pred timing útokom

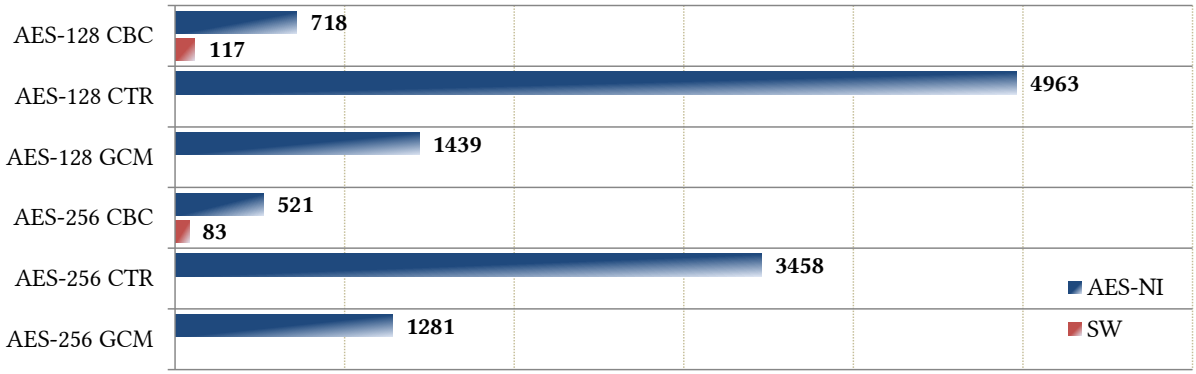
fického protokolu a jeho implementácia dopadne s vysokou pravdepodobnosťou z bezpečnostného hľadiska zle.

6 Použitie kryptografických konštrukcií

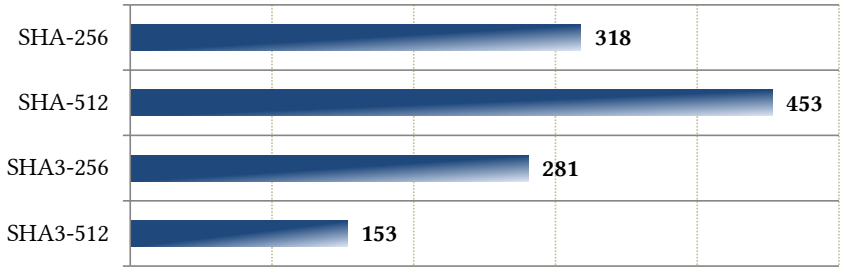
6.1 Výkonové porovnanie

V tejto časti uvedieme výkonové porovnanie vybraných kryptografických algoritmov. Konkrétny výkon sa môže dramaticky líšiť pri rôznych platformách a implementáciách algoritmov. Častokrát sa výkon líši aj v závislosti od verzie použitých knižníc, prípadne volieb pri ich kompilácii. Preto uvedené hodnoty skôr ilustrujú relatívne výkonové rozdiely medzi jednotlivými algoritmami. Hodnoty boli získané v nasledujúcom prostredí: procesor i7-2600 (3,40 GHz), implementácia OpenSSL.

Obrázky 7 a 8 zobrazujú výkonové charakteristiky pre šifrovanie a hašovanie. V oboch prípadoch sa spracúvali bloky dĺžky 8KB blokov, pričom kryptografické operácie vykonávalo jedno aplikačné vlákno (thread). Obrázok 7 ukazuje výrazný rozdiel medzi softvérovou implementáciou a využitím hardvérovej podpory pre AES (ilustrované na CBC móde). Zároveň je vidieť rozdiel medzi jednotlivými módmi a medzi šifrovaním s rôznou dĺžkou kľúča (teda s rôznym počtom kôl).

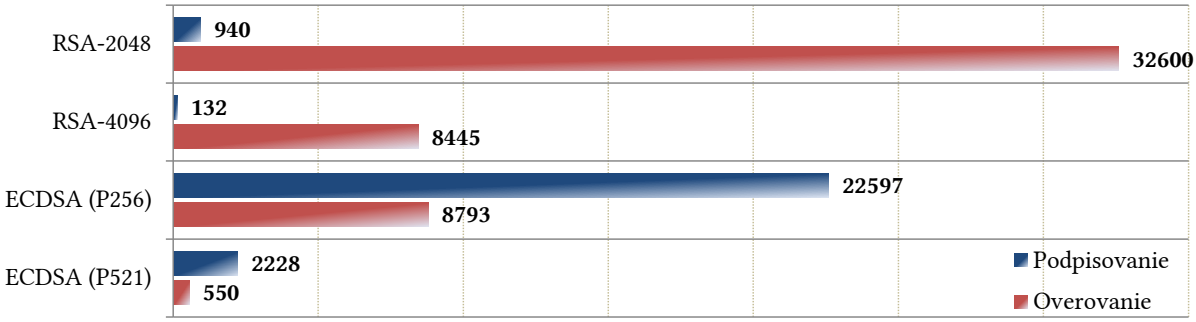


Obr. 7: Rýchlosť šifrovania 8KB blokov [MB/s]



Obr. 8: Rýchlosť hašovania 8KB blokov [MB/s]

Graf na obrázku 9 porovnáva výkon podpisových schém RSA a ECDSA pri rôznych dĺžkach kľúčov. V prípade ECDSA sú zvolené dve z eliptických kriviek štandardizovaných NIST. V prípade RSA schémy je to zároveň indikácia výkonu šifrovacej a dešifrovacej transformácie schém s rovnako dlhými kľúčmi. Pri interpretácii výsledkov je užitočné uvedomiť si, aké sú ekvivalentné dĺžky kľúčov medzi oboma schémami (pozri časť 5).



Obr. 9: Rýchlosť podpisovania a overovania podpisov [počet/s]

6.2 S/MIME a OpenPGP

S/MIME (Secure/Multipurpose Internet Mail Extensions) je štandard pre šifrovanie a podpisovanie elektronickej pošty, podporovaný väčšinou mailových klientov (napr. Outlook, Apple Mail, Thuderbird). V prípade webových poštových služieb je zvyčajne potrebné podporu S/MIME riešiť doplnkami prehliadačov. Verejné kľúče používateľov sú distribuované vo forme X.509 certifikátov. Formát správ je definovaný ako CMS (Cryptographic Message Syntax). V S/MIME sa používajú štandardné kryptografické konštrukcie. Prehľad povinne implementovaných konštrukcií v ostatných troch verziách S/MIME a v navrhovanej verzii 4.0 je uvedený v tabuľke 3. Podotknime, že implementácie v mailových klientoch zahŕňajú širšiu sadu algoritmov kvôli vzájomnej interoperabilite ako aj spätnej kompatibilite.

Povinné v CMS („MUST“)	3.0 (1999) RFC 2633	3.1 (2004) RFC 3851	3.2 (2010) RFC 5751	4.0 (2019) RFC 8551 *návrh*
Hašovacia funkcia	SHA-1	SHA-1	SHA-256	SHA-256 SHA-512
Podpisová schéma	DSA	RSA DSA	RSA	RSA ECDSA EdDSA
Asymetrické šifrovanie, resp. dohodnutie kľúča	DH	RSA	RSA	RSA ECDH
Symetrické šifrovanie	3DES CBC	3DES CBC	AES-128 CBC	AES-128 GCM AES-256 GCM AES-128 CBC

Tabuľka 3: Povinne implementované algoritmy v rôznych verziách S/MIME

Iné riešenie pre zabezpečenie dôvernosti a autentickosti elektronickej pošty je štandard OpenPGP (RFC 4880), s voľne dostupnou implementáciou GnuPG. Hlavný rozdiel oproti S/MIME je jednoduchší spôsob správy a distribúcie kľúčov – bez použitia certifikátov, väzieb na certifikačné autority a pod. Inak poskytuje OpenPGP podobné kryptografické riešenie ako S/MIME, teda kombinuje symetrické a asymetrické šifrovanie s vhodnou podpisovou schémou. Použitie v mailových klientoch vyžaduje obvykle inštaláciu doplnku. OpenPGP formát sa často používa aj pri podpisovaní súborov, napr. pri

distribúcii softvérových balíkov.

Elektronická pošta je oblasť, kde sa kryptografické konštrukcie používajú dlhodobo. Práve preto prekvapil v roku 2018 výskum v oblasti zraniteľností implementácií S/MIME a OpenPGP v mailových klientoch a ich rozšíreniach, publikovaný pod názvom EFAIL [17]. Identifikované zraniteľnosti sa týkali 23 z 35 testovaných S/MIME klientov a 10 z 28 testovaných OpenPGP klientov, zároveň však poukázali aj na nedostatky v samotných štandardoch. Bezpečná implementácia a integrácia kryptografických mechanizmov v aplikáciách nie je jednoduchá úloha.

7 Rady na záver

Na záver len dve rady, ktorých naplnenie nie je jednoduché, ale pomôže zvýšiť kryptografickú bezpečnosť IT prostredia.

- ✓ Inšpirujte sa existujúcimi doporučeniami renomovaných inštitúcií a organizácií. Samozrejme, takých doporučení a štandardov existuje veľa, najznámejšie vydáva NIST. Ak si odmyslíme legislatívne a sektorové požiadavky (napr. PCI DSS pre oblasť platobných kariet[24]), zaujímavé môžu byť napríklad aj:
  - v oblasti kryptogafických mechanizmov, vrátane doporučení pre konfiguráciu protokolov TLS, IPsec a SSH, technické usmernenia nemeckého BSI TR-02102[25];
  - v oblasti aplikačnej bezpečnosti, vrátane požiadaviek súvisiacich s kryptografickými opatreniami, OWASP Application Security Verification Standard [16].
- ✓ Sledujte zraniteľnosti aj v tejto oblasti. Hoci zvyčajne je potrebné počkať na záplaty a aktualizácie výrobcov softvéru, v niektorých prípadoch je potrebná konfiguračná zmena používaných kryptografických mechanizmov.

8 Otázky a úlohy

Riešenie rôznorodých úloh a zamyslenie sa nad vybranými otázkami súvisiacimi s používaním kryptografických techník má pomôcť k lepšiemu pochopeniu a snáď aj prehĺbeniu prebraných tém.

1. Zistite, koľko RSA podpisov pre dĺžku modulu 2048 a 4096 bitov zvládne za sekundu vykonať váš notebook.
2. Overte podpis softvérového balíka stiahnutého z internetu (napr. OpenSSL[26], KeePass[27]). Zmeňte v súbore 1 bajt a skúste podpis opätovne overiť.
3. Zistite, v koľkých certifikačných logoch (denníkoch transparentnosti) sa nachádza certifikát webového servera vašej organizácie, resp. iného servera. Aké certifikáty sú registrované v certifikačných logoch pre vašu doménu? Využite dostupné webové rozhrania[28].

[24] https://www.pcisecuritystandards.org/ (september 2019)
[25] https://www.bsi.bund.de/EN/Publications/TechnicalGuidelines/tr02102/tr02102_node.html (november 2020)
[26] https://www.openssl.org/source/ (november 2020)
[27] https://keepass.info/integrity.html (november 2020)
[28] Napr. https://crt.sh/ alebo https://transparencyreport.google.com/https/certificates (november 2020)

4. Pri konfigurácii IPsec tunela máte možnosť vybrať štruktúru, v ktorej prebehne DH protokol v rámci IKEv2. Na výber máte tieto možnosti: group18 a group19. Zdôvodnite, ktorú zvoliť, ak je vašou prioritou bezpečnosť.

5. Vo vybratej distribúcii Linuxu zistite, aká funkcia je použitá na ukladanie hesiel. Kde je ukladaná soľ a ako viete ovplyvniť počet iterácií?

6. Zamyslite sa, či je pri zmene hesla vhodné vygenerovať aj novú soľ a prečo.

7. DNSSEC je sada rozšírení DNS s cieľom zabezpečiť autentickosť dát (teda klient dokáže overiť, že získané DNS informácie sú autentické). Zistite aká schéma a aký dlhý kľúč je použitý na podpisovanie záznamov v koreňovej (root) zóne.

8. Symetrický 128 bitový kľúč pre šifrovanie súboru bol vygenerovaný deterministickým generátorom, ktorý bol inicializovaný aktuálnym časom (s presnosťou na 1 ms). Útočník vie, v ktorý deň bol súbor zašifrovaný. Koľko kľúčov potrebuje útočník prezrieť a teda akej dĺžke symetrického kľúča zodpovedá skutočný priestor kľúčov?

9. Stiahnite aktuálny CRL vybratej certifikačnej autority. Aká je štruktúra CRL a koľko certifikátov obsahuje? Aký má význam atribút „Next Update“?

10. Používatelia využívajú rôzne služby na internete. Na jednostrannú autentizáciu používateľa a prenos ním zvoleného kľúča spojenia je použitý nasledujúci postup. Pri nadviazaní spojenia pošle používateľ na server služby B nasledujúcu správu: $A, E_B K, \text{sig}_A$, kde A je identita používateľa (certifikát jeho verejného kľúča), $E_B(K)$ je verejným kľúčom služby B zašifrovaný kľúč spojenia K (zvolený používateľom) a sig_A je podpis pre $E_B(K)$, ktorý používateľ vytvorí s pomocou svojho súkromného kľúča. Server po prijatí správy overí platnosť certifikátu, pomocou verejného kľúča používateľa overí podpis sig_A pre $E_B(K)$. Ak je podpis korektný, tak dešifruje kľúč spojenia K. Identifikujte bezpečnostný problém.

Literatúra

[1] *Algorithms, Key Size and Protocols Report (2018)*. ECRYPT – Coordination & Support Action, 2018. URL: https://www.ecrypt.eu.org/csa/documents/D5.4-FinalAlgKeySizeProt.pdf (cit. 08/2019) (cit. na str. 16).

[2] D. Antonioli, N. O. Tippenhauer a K. B. Rasmussen. „The KNOB is Broken: Exploiting Low Entropy in the Encryption Key Negotiation Of Bluetooth BR/EDR“. In: *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, 2019, s. 1047–1061. URL: https://www.usenix.org/conference/usenixsecurity19/presentation/antonioli (cit. 11/2020) (cit. na str. 19).

[3] H. Böck, J. Somorovsky a C. Young. „Return Of Bleichenbacher's Oracle Threat (ROBOT)“. In: *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, 2018, s. 817–849. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/bock (cit. 11/2020) (cit. na str. 19).

[4] H. Böck, A. Zauner, S. Devlin, J. Somorovsky a P. Jovanovic. „Nonce-Disrespecting Adversaries: Practical Forgery Attacks on GCM in TLS“. In: *10th USENIX Workshop on Offensive Technologies (WOOT 16)*. USENIX Association, 2016. URL: https://www.usenix.org/conference/woot16/workshop-program/presentation/bock (cit. na str. 5).

[5] *Digital Signature Standard (DSS)*. FIPS PUB 186-4. National Institute of Standards and Technology, 2013. DOI: 10.6028/NIST.FIPS.186-4 (cit. na str. 10).

[6] *Digital Signature Standard (DSS)*. FIPS PUB 186-5 (Draft). National Institute of Standards and Technology, 2019. DOI: 10.6028/NIST.FIPS.186-5-draft (cit. na str. 10).

[7] N. Drucker a S. Gueron. *Selfie: reflections on TLS 1.3 with PSK*. Cryptology ePrint Archive, Report 2019/347. 2019. URL: https://eprint.iacr.org/2019/347 (cit. 11/2020) (cit. na str. 19).

[8] T. Duong a J. Rizzo. *Here Come The ⊕ Ninjas*. Unpublished manuscript. 2011 (cit. na str. 4).

[9] T. Halevi a N. Saxena. „Keyboard acoustic side channel attacks: exploring realistic and security-sensitive scenarios“. In: *International Journal of Information Security* 14.5 (2015), s. 443–456. DOI: 10.1007/s10207-014-0264-7 (cit. na str. 19).

[10] N. Heninger, Z. Durumeric, E. Wustrow a J. A. Halderman. „Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices“. In: *Proceedings of the 21st USENIX Security Symposium*. 2012. URL: https://factorable.net/weakkeys12.extended.pdf (cit. 11/2020) (cit. na str. 19).

[11] T. Jager, S. A. Kakvi a A. May. „On the Security of the PKCS#1 V1.5 Signature Scheme“. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. CCS '18. ACM, 2018, s. 1195–1208. DOI: 10.1145/3243734.3243798 (cit. na str. 10).

[12] A. Kwong, D. Genkin, D. Gruss a Y. Yarom. „RAMBleed: Reading Bits in Memory Without Accessing Them“. In: *41st IEEE Symposium on Security and Privacy (S&P)*. 2020. URL: https://rambleed.com/docs/20190603-rambleed-web.pdf (cit. 11/2020) (cit. na str. 19).

[13] G. Leurent a T. Peyrin. „From Collisions to Chosen-Prefix Collisions Application to Full SHA-1“. In: *Advances in Cryptology – EUROCRYPT 2019*. Springer, 2019, s. 527–555 (cit. na str. 6).

[14] C. Meyer a J. Schwenk. *Lessons Learned From Previous SSL/TLS Attacks - A Brief Chronology Of Attacks And Weaknesses*. Cryptology ePrint Archive, Report 2013/049. 2013. URL: https://eprint.iacr.org/2013/049 (cit. 11/2020) (cit. na str. 19).

[15] M. Nemec, M. Sys, P. Svenda, D. Klinec a V. Matyas. „The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli“. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS '17. ACM, 2017, s. 1631–1648. DOI: 10.1145/3133956.3133969 (cit. na str. 19).

[16] *OWASP Application Security Verification Standard 4.0.2*. 2020. URL: https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project (cit. 11/2020) (cit. na str. 22).

[17] D. Poddebniak, C. Dresen, J. Müller, F. Ising, S. Schinzel, S. Friedberger, J. Somorovsky a J. Schwenk. „Efail: Breaking S/MIME and OpenPGP Email Encryption using Exfiltration Channels“. In: *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, 2018, s. 549–566. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/poddebniak (cit. 11/2020) (cit. na str. 22).

[18] G. d. S. Faria a H. Y. Kim. „Identification of Pressed Keys by Acoustic Transfer Function“. In: *2015 IEEE International Conference on Systems, Man, and Cybernetics*. 2015, s. 240–245. DOI: 10.1109/SMC.2015.54 (cit. na str. 19).

[19] *Secure Hash Standard (SHS)*. FIPS PUB 180-4. National Institute of Standards and Technology, 2015. DOI: 10.6028/NIST.FIPS.180-4 (cit. na str. 6).

[20] *SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash, and ParallelHash*. NIST Special Publication 800-185. National Institute of Standards and Technology, 2016. DOI: 10.6028/NIST.SP.800-185 (cit. na str. 6).

[21] *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. FIPS PUB 202. National Institute of Standards and Technology, 2015. DOI: 10.6028/NIST.FIPS.202 (cit. na str. 6).

[22] I. Shumailov, L. Simon, J. Yan a R. Anderson. „Hearing your touch: A new acoustic side channel on smartphones“. In: *ArXiv* abs/1903.11137 (2019). URL: https://arxiv.org/abs/1903.11137 (cit. na str. 19).

[23] T. Tervoort. *Zerologon: Unauthenticated domain controller compromise by subverting Netlogon cryptography (CVE-2020-1472)*. Whitepaper, Secura Cryptology ePrint Archive, Report 2019/347. 2020. URL: https://www.secura.com/uploads/whitepapers/Zerologon.pdf (cit. 11/2020) (cit. na str. 4).

[24] C. Thompson, M. Shelton, E. Stark, M. Walker, E. Schechter a A. P. Felt. „The Web's Identity Crisis: Understanding the Effectiveness of Website Identity Indicators“. In: *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, 2019, s. 1715–1732. URL: https://www.usenix.org/conference/usenixsecurity19/presentation/thompson (cit. 11/2020) (cit. na str. 13).

[25] *Transitioning the Use of Cryptographic Algorithms and Key Lengths*. NIST Special Publication 800-131A Rev. 2. National Institute of Standards and Technology, 2019. DOI: 10.6028/NIST.SP.800-131Ar2 (cit. na str. 2).

[26] M. Vanhoef a E. Ronen. „Dragonblood: Analyzing the Dragonfly Handshake of WPA3 and EAP-pwd“. In: *IEEE Symposium on Security & Privacy (SP)*. IEEE, 2020. URL: https://wpa3.mathyvanhoef.com/ (cit. 11/2020) (cit. na str. 19).

Príloha: ilustračné príklady

Ilustračné príklady využívajú program OpenSSL vo verzii 1.1.1.

RSA – generovanie inštancie

Generovanie inštancie RSA schémy (nerozlišujeme, či je určená na šifrovanie alebo pre podpisovú schému) s dĺžkou kľúča 2048 bitov:

```
$ openssl genrsa -out myrsa.pem 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
.....................................................+++++
.........................+++++
e is 65537 (0x010001)
```

Poznamenajme, že v našom príklade je kľúč uložený nešifrovane, hoci openssl umožňuje kľúč aj šifrovať s použitím hesla. V súbore myrsa.pem sú uložené jednotlivé parametre RSA inštancie (samotný súbor obsahuje dáta vo formáte PEM kódované v base64, ktoré sú v nasledujúcom výstupe zobrazené v časti „writing RSA key“). Vo výstupe sú pre skrátenie výstupu vynechané niektoré riadky. K významu

jednotlivých hodnôt (pozri aj časť 2.1, pričom hodnoty exponent1, exponent2 a coefficient sú použité na urýchľovanie súkromnej RSA transformácie):

modulus	hodnota n
publicExponent	hodnota e
privateExponent	hodnota d
prime1, prime2	prvočísla p, q (bez ujmy na všeobecnosti v tomto poradí)
exponent1, exponent2	hodnoty $d \bmod (p-1)$ a $d \bmod (q-1)$
coefficient	hodnota $q^{-1} \bmod p$

```
$ openssl rsa -in myrsa.pem -text
RSA Private-Key: (2048 bit, 2 primes)
modulus:
    00:d0:71:30:bf:c0:be:64:25:00:9f:d6:3a:e4:e5:
    ... vynechaných 16 riadkov ...
    2c:03
publicExponent: 65537 (0x10001)
privateExponent:
    00:c8:9c:98:01:85:5c:f8:7f:50:69:85:42:eb:77:
    ... vynechaných 16 riadkov ...
    61:a9
prime1:
    00:f9:1e:03:4e:95:95:a4:5a:8e:91:c2:9e:cc:bd:
    ... vynechaných 7 riadkov ...
    26:ac:bb:7f:15:3f:d5:2d:15
prime2:
    00:d6:33:7c:59:43:d5:29:72:03:6f:8d:3b:e8:3a:
    ... vynechaných 7 riadkov ...
    a2:31:ca:e4:d8:39:a9:aa:b7
exponent1:
    00:be:2b:30:21:14:45:98:a2:5c:85:5e:c9:74:c7:
    ... vynechaných 7 riadkov ...
    7e:33:8c:2a:16:21:95:6d:85
exponent2:
    00:b5:93:71:6e:ae:1c:cd:84:53:bb:45:4b:2a:41:
    ... vynechaných 7 riadkov ...
    9a:d2:80:be:db:38:8e:46:23
coefficient:
    00:d3:4f:f4:76:27:8d:13:56:44:70:55:76:38:7a:
    ... vynechaných 7 riadkov ...
    2b:e5:49:95:1a:91:44:ee:f2
writing RSA key
-----BEGIN RSA PRIVATE KEY-----
MIIEpgIBAAKCAQEA0HEwv8C+ZCUAn9Y65OVOOdSLBeTqZxHWC7rISlJYP4r6Ldbu
UKLpF2XOsCbyqSKnANqInBspAfzswZhdQJ2HKdz8v//KpYJlQ0OcfOQvjAEU7YXp
... vynechaných 22 riadkov ...
Yz+O7ZjgC5fFTYge/lrEAnNwUbCX3lnpCEBFsp53haMKK+VJlRqRRO7y
```

```
-----END RSA PRIVATE KEY-----
```

Extrakcia verejného kľúča a jeho súčasti:

```
$ openssl rsa -in myrsa.pem -pubout -out myrsa-pub.pem
writing RSA key
$ openssl rsa -pubin -in myrsa-pub.pem -text
RSA Public-Key: (2048 bit)
Modulus:
    00:d0:71:30:bf:c0:be:64:25:00:9f:d6:3a:e4:e5:
    ... vynechaných 16 riadkov ...
    2c:03
Exponent: 65537 (0x10001)
writing RSA key
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA0HEwv8C+ZCUAn9Y65OVO
OdSLBeTqZxHWC7rISlJYP4r6LdbuUKLpF2XOsCbyqSKnANqInBspAfzswZhdQJ2H
... vynechané 4 riadky ...
AwIDAQAB
-----END PUBLIC KEY-----
```

RSA – šifrovanie a podpisovanie

Šifrovanie a dešifrovanie krátkeho textu pomocou RSA a výplne PKCS #1 v1.5. Výplňová schéma je explicitne zadaná pri šifrovaní (nie je to nutné, táto schéma je implicitná). Zašifrovaný text je v binárnom súbore cipher.bin. Za povšimnutie stojí fakt, že pri šifrovaní je jedným zo vstupov súbor s verejným kľúčom (myrsa-pub.pem) a pri dešifrovaní súbor so súkromným kľúčom (myrsa.pem).

```
$ echo 'Kryptologia II - pokusny text' | openssl pkeyutl -encrypt
  -pkeyopt rsa_padding_mode:pkcs1 -pubin -inkey myrsa-pub.pem
  -out cipher.bin
$ openssl pkeyutl -decrypt -inkey myrsa.pem -in cipher.bin
Kryptologia II - pokusny text
```

Podpísanie a následné overenie podpisu súboru text.txt pomocou RSA a výplne PKCS #1 v1.5 (implicitná voľba), pričom použijeme hašovaciu funkciu SHA-256. Podpis je uložený v binárnom súbore sig.bin. Pri podpisovaní sa použije súkromný kľúč a pri overovaní verejný kľúč RSA schémy.

```
$ cat text.txt | openssl dgst -sha256 -binary | openssl pkeyutl -sign
  -inkey myrsa.pem -out sig.bin -pkeyopt digest:sha256
$ cat text.txt | openssl dgst -sha256 -binary | openssl pkeyutl -verify
  -sigfile sig.bin -pubin -inkey myrsa-pub.pem -pkeyopt digest:sha256
Signature Verified Successfully
```

CSR a samopodpísaný certifikát

Vygenerovanie novej inštancie RSA schémy s dĺžkou kľúča 2048 bitov. Súkromný a verejný kľúč sú (nešifrované) uložené v súbore myrsa.pem. V súbore mycsr.csr je uložený CSR pre potenciálnu žiadosť o vydanie certifikátu certifikačnou autoritou.

```
$ openssl req -new -newkey rsa:2048 -keyout myrsa.pem -nodes -subj
  "/C=SK/L=Bratislava/O=Testovacia spolocnost/CN=www.test.xx" -out mycsr.csr
Generating a RSA private key
.....................................................+++++
.....................................................+++++
writing new private key to 'myrsa.pem'
-----
```

Pozrime sa na štruktúru informácií v CSR, pričom samotný súbor mycsr.csr obsahuje dáta vo formáte PEM kódované v base64, ktoré sú v nasledujúcom výstupe zobrazené v časti ohraničenej BEGIN/END CERTIFICATE REQUEST.

```
$ openssl req -in mycsr.csr -text
Certificate Request:
  Data:
    Version: 1 (0x0)
      Subject: C = SK, L = Bratislava, O = Testovacia spolocnost,
               CN = www.test.xx
      Subject Public Key Info:
        Public Key Algorithm: rsaEncryption
          RSA Public-Key: (2048 bit)
          Modulus:
            00:d5:3c:4e:43:ff:96:3b:81:2a:bd:90:b7:9c:4d:
            ... vynechaných 16 riadkov ...
            18:97
          Exponent: 65537 (0x10001)
      Attributes:
        a0:00
  Signature Algorithm: sha256WithRSAEncryption
    83:77:ab:5c:ba:af:4d:85:4a:59:b1:25:40:e6:c7:12:e3:6b:
    ... vynechaných 13 riadkov ...
    16:c2:f2:28
-----BEGIN CERTIFICATE REQUEST-----
MIICnTCCAYUCAQAwWDELMAkGA1UEBhMCU0sxEzARBgNVBAcMCkJyYXRpc2xhdmEx
... vynechaných 13 riadkov ...
KA==
-----END CERTIFICATE REQUEST-----
```

Samopodpísaný certifikát získame (vrátane novej RSA inštancie) napríklad takto, pričom súkromný kľúč je uložený v súbore myrsa2.pem a certifikát v súbore mycer2.cer:

```
$ openssl req -new -newkey rsa:2048 -keyout myrsa2.pem -nodes -subj
  "/C=SK/L=Bratislava/O=Testovacia spolocnost/CN=www.test.xx" -x509
  -days 1000 -out mycer2.cer
```

Overenie certifikátu prostredníctvom OCSP

Overme platnosť certifikátu web servera Európskej komisie pomocou OCSP. Certifikát si uložíme do súboru ec.pem a následne z neho získame url, kam možno posielať OCSP požiadavky:

```
$ openssl s_client -connect ec.europa.eu:443 </dev/null 2>/dev/null |
  openssl x509 -outform PEM >ec.pem
$ openssl x509 -in ec.pem -ocsp_uri -noout
http://ocsp2.globalsign.com/gsorganizationvalsha2g2
```

Získame certifikát CA, ktorá vydala certifikát pre web server, vrátane konverzie z DER do PEM formátu. V tomto prípade je CA „GlobalSign Organization Validation CA - SHA256 - G2“. Výstup je v súbore gs.pem.

```
$ openssl x509 -in ec.pem -text -noout | grep "CA Issuer"
      CA Issuers - URI:http://secure.globalsign.com/cacert/
                   gsorganizationvalsha2g2r1.crt
$ wget -q  http://secure.globalsign.com/cacert/gsorganizationvalsha2g2r1.crt
$ openssl x509 -in gsorganizationvalsha2g2r1.crt -inform DER -out gs.pem
```

Informácie posielané a získané v odpovedi pri overovaní certifikátu ec.pem pomocou OCSP vidieť na nasledujúcom príklade. Sumár bez detailov je v posledných 4 riadkoch výpisu.

```
$ openssl ocsp -nonce -issuer gs.pem -cert ec.pem -url
  http://ocsp2.globalsign.com/gsorganizationvalsha2g2 -text
OCSP Request Data:
    Version: 1 (0x0)
    Requestor List:
        Certificate ID:
          Hash Algorithm: sha1
          Issuer Name Hash: 0C9E4D9C3DEDEF84D891E972C7CF8406BC197B07
          Issuer Key Hash: 96DE61F1BD1C1629531CC0CC7D3B830040E61A7C
          Serial Number: 34D7E3DA71A37D6F9627FA6D
    Request Extensions:
        OCSP Nonce:
           0410BA1B29B27389E35972466F9068E47D94
OCSP Response Data:
    OCSP Response Status: successful (0x0)
    Response Type: Basic OCSP Response
    Version: 1 (0x0)
    Responder Id: 9C4D0099000E8BB0018175A1BAF0D025D7A01C47
```

```
    Produced At: Jan 17 21:28:35 2021 GMT
    Responses:
    Certificate ID:
      Hash Algorithm: sha1
      Issuer Name Hash: 0C9E4D9C3DEDEF84D891E972C7CF8406BC197B07
      Issuer Key Hash: 96DE61F1BD1C1629531CC0CC7D3B830040E61A7C
      Serial Number: 34D7E3DA71A37D6F9627FA6D
    Cert Status: good
    This Update: Jan 17 21:28:35 2021 GMT
    Next Update: Jan 21 21:28:35 2021 GMT

    Response Extensions:
        OCSP Nonce:
            0410BA1B29B27389E35972466F9068E47D94
    Signature Algorithm: sha256WithRSAEncryption
         57:8d:3a:6f:f5:10:cd:9b:1b:20:6f:59:1d:cc:05:33:fb:2f:
         ... vynechaných 13 riadkov ...
         53:ca:72:e2

Certificate:
    ... vynechané riadky popisujúce certifikát OCSP Respondera ...

Response verify OK
ec.pem: good
        This Update: Jan 17 21:28:35 2021 GMT
        Next Update: Jan 21 21:28:35 2021 GMT
```