1 Příhody Alice a Boba

V této kapitole si představíme čtyři základní *kryptografická primitiva* – „kostičky“, ze kterých je vystavěna většina kryptografických protokolů: symetrické a asymetrické šifry, hešovací funkce a náhodné generátory. Zatím je budeme popisovat neformálně (detaily ještě pár kapitol počkají), ale vyzkoušíme si, jak se o nich přemýšlí.

1.1 Symetrické šifry

Začneme jednoduchou úlohou. Alice chce svého kamaráda Boba pozvat do čajovny. Chce mu proto po síti poslat *zprávu* – nějaký řetězec znaků, pro jednoduchost předpokládejme, že jedniček a nul. Jenže na síti nejsou sami: vždycky se najde někdo, kdo zvědavě poslouchá, co se šustne, a pak šíří drby. Říkejme mu třeba Eva⟨1⟩ nebo CIA.⟨2⟩

Alice s Bobem se proto dohodli, že zprávy budou *šifrovat.* Alice zprávu zakóduje nějakým algoritmem, který je známý jenom jí a Bobovi. Eva stále může zprávu odposlechnout, jen už jí nebude dávat žádný smysl.

Utajovat (potenciálně složitý) algoritmus je ovšem docela nepraktické. Místo toho algoritmus parametrizujeme *klíčem* – nějakým dalším řetězcem, který ovlivňuje šifrování a který utajíme místo algoritmu. Alice a Bob se dopředu bezpečnou cestou shodnou na klíči. Cokoliv Alice pomocí klíče zašifruje, Bob pomocí téhož klíče dešifruje. Ale Eva klíč nezná, takže jí zpráva nebude srozumitelná. Tomu říkáme *symetrická šifra,* protože Alice a Bob používají stejný klíč.

Obecně bychom se na symetrickou šifru mohli dívat jako na dvojici „krabiček“ E a D. *Šifrovací krabička* E (*encryption box*) dostane *otevřenou zprávu (plain text)* a klíč a spočítá z nich *zašifrovanou zprávu (cipher text). Dešifrovací krabička* D (*decryption box*) dostane zašifrovanou zprávu a klíč a spočítá původní otevřenou zprávu. Většinou chceme, aby šifrování zachovávalo délku zprávy.

Příklad (Caesarova šifra): Podívejme se na triviální příklad symetrické šifry pocházející prý od Julia Caesara. Alice chce poslat zprávu složenou z písmen abecedy `ABC...Z`. Zvolí si klíč $k \in \{0, \ldots, 25\}$ a její šifrovací krabička místo každého písmena pošle takové, které je v abecedě o k pozic dál (cyklicky). Takže zprávu `AHOY` při $k = 3$ zašifruje na `DKRB`. Bobova dešifrovací krabička posouvá o k písmen v opačném směru.

⟨1⟩ V angličtině to funguje lépe: je to Eve podle slova *eavesdrop* s významem *potají naslouchat.*

⟨2⟩ Autor zajisté ví, že americká třípísmenková agentura starající se o odposlech zpráv není CIA, ale NSA. Jenže copak se tady dá odolat zkratce začínající na C?

Pravidlu, že utajujeme klíč a nikoliv algoritmus, se říká *Kerckhoffsův princip.*[3] Proč je tak důležitý?

- Dobrých šifer je známých málo a je těžké je vytvořit.
- U prakticky žádné šifry neumíme dokázat, že je bezpečná. Je tedy lepší použít veřejně známou šifru, kterou už hodně kryptografů studovalo a nenašli v ní žádnou slabinu.
- Pokud se protivník dozví část našeho tajemství, je mnohem snazší vyměnit zkompromitovaný klíč než algoritmus.

Poznámka: Ještě dodejme, že Alice a Bob nemusí být konkrétní osoby. Jsou to *role* v protokolu. Chceme-li na zprávu odpovědět, postavy si role prohodí. Pokud chceme bezpečně zálohovat data, Bob bude Alice někdy v budoucnosti. A tak dále.

1.2 Asymetrické šifry

Představte si, že bychom místo dvojice Alice a Bob měli N lidí, kteří si chtějí navzájem vyměňovat šifrované zprávy. Každé zprávě přitom smí rozumět jenom odesílatel a adresát. Použití symetrické šifry by vyžadovalo, aby se každá dvojice lidí předem shodla na svém klíči. To znamená vytvořit a bezpečně distribuovat řádově N^2 klíčů, což je moc.

Pořídíme si *asymetrickou šifru.* Ta má dva klíče: *šifrovací* a *dešifrovací.* Ty lze vyrobit jen jako pár (máme-li jen jeden, nepomůže nám to k získání druhého). Co zašifrujeme šifrovacím klíčem, dá se dešifrovat jen příslušným dešifrovacím klíčem.

Pak stačí, aby každý z N účastníků protokolu zveřejnil svůj šifrovací klíč (třeba v nějakém veřejném adresáři) a ponechal si ten dešifrovací. Tím pádem Alici stačí najít v adresáři Bobův šifrovací klíč, zašifrovat pomocí něj zprávu a tu pak umí dešifrovat jen Bob.

Pozor na to, že klíče mají dva druhy vlastností: *veřejný* vs. *soukromý* a *šifrovací* vs. *dešifrovací*. V našem protokolu je šifrovací klíč veřejný a dešifrovací soukromý. Zajímavá je i opačná kombinace:

Příklad (podpis): Zveřejníme-li dešifrovací klíč a ponecháme si šifrovací, získáme něco, co se chová jako *podpis.* Každý si může zprávu přečíst, ale jenom majitel příslušného soukromého klíče ji umí vytvořit. Všimněte si rozdílu: šifra zajišťuje *utajení* obsahu zprávy (nepovolané osoby ji neumí přečíst), zatímco podpis zajišťuje *integritu* (pokud nepovolaná osoba zprávu změní, přijde se na to). Často budeme chtít kombinovat obojí.

[3] Přišel s ním koncem 19. století nizozemský kryptograf Auguste Kerckhoffs.

Časem se ukáže, že naše triviální použití veřejných klíčů má zásadní slabinu: Nic nebrání útočníkovi podstrčit do veřejného adresáře svůj klíč místo klíče oběti. Později tento problém vyřeší certifikační autority.

1.3 Hešovací funkce

Dalším důležitým primitivem je *kryptografická hešovací funkce*. Ta na vstupu dostane libovolnou zprávu a spočítá z ní *otisk (fingerprint)* fixní délky (třeba 256 bitů). Funkci budeme považovat za „dostatečně náhodnou“ a oproti hešovacím funkcím známým z datových struktur po ní budeme navíc chtít, aby byla:

- bez inverzí – pro zadaný otisk se neumí efektivně najít vstup, který se zahešuje na tento otisk;
- bez kolizí – ačkoliv *kolize* (dvojice zpráv se stejným otiskem) evidentně nastávají, je těžké nějakou najít.

Příklad (vylepšení podpisů): Podepisování zpráv pomocí asymetrických šifer není zrovna šikovné. Pokud zprávu postupně podepíše více osob (představte si petici), je k přečtení zprávy potřeba ji mnohokrát dešifrovat. To je jednak pomalé, ale také to selže, pokud nám chybí některý z veřejných klíčů. Raději proto pošleme zprávu otevřeně a kdykoliv ji někdo bude chtít podepsat, asymetricky zašifruje otisk zprávy. Jednotlivé podpisy budou krátké a na sobě nezávislé. Hešovací funkce nám pak zajistí, že podpis skutečně patří k obsahu zprávy. Všimněte si, že bezkoliznost heše je zásadní: kdyby někdo uměl vytvořit dvě zprávy se stejným otiskem, mohl by oběti předložit k podepsání jednu z nich a pak tvrdit, že podepsala tu druhou.

Příklad (symetrické podpisy): Pomocí hešovacích funkcí také můžeme sestrojit symetrickou obdobu podpisů (té se říká *message authentication code*), v níž se tentýž klíč používá jak na podepisování, tak na ověřování podpisu. To se hodí, pokud chceme zajistit integritu dat mezi dvěma stranami, které si navzájem věří. Konstrukce je snadná: spočítáme otisk řetězce vzniklého slepením podepisovacího klíče se zprávou.

1.4 Náhodné generátory

Kryptografické protokoly také často potřebují generátor náhodných bitů *(RNG – random number generator)*. Ten má být nejen statisticky rovnoměrný, ale pro vnějšího pozorovatele kompletně nepředvídatelný – i pokud by pozorovatel znal celou historii vygenerovaných bitů, nesmí mu to pomoci zjistit cokoliv o následujícím bitu.

RNG se používá zejména pro generování klíčů, ale má i jiné zajímavé aplikace.

Příklad (hybridní šifra): Zatímco symetrické šifry obvykle mají lineární časovou složitost, všechny známé asymetrické šifry jsou asymptoticky pomalejší. Proto se používá konstrukce, která kombinuje oba druhy šifer. Říká se jí *hybridní šifra* a funguje následovně:

- Značení:
  - Nechť E_A a D_A je šifrovací a dešifrovací funkce nějaké asymetrické šifry.
  - K_E a K_D je šifrovací a dešifrovací klíč pro tuto šifru vygenerovaný Bobem. Klíč K_E je veřejný a zná ho i Alice. Klíč K_D je soukromý Bobův.
  - E_S a D_S je šifrovací a dešifrovací funkce nějaké symetrické šifry.
- Alice chce poslat nějakou zprávu x Bobovi.
- Alice vygeneruje náhodný klíč N pro symetrickou šifru.
- Alice zašifruje zprávu symetricky: $y = E_S(N, x)$.
- Alice zašifruje klíč N asymetricky: $z = E_A(K_E, y)$.
- Alice pošle Bobovi y a z.
- Bob použije asymetrickou šifru, aby získal klíč $N = D_A(K_D, z)$.
- Bob použije symetrickou šifru, aby získal zprávu $x = D_S(N, y)$.

Všimněte si, že pomalou asymetrickou šifru používáme jen na krátký řetězec N fixní délky, zatímco celou dlouhou zprávu x šifrujeme symetricky.

Útočník má následující možnosti:

- zaútočit na symetrickou šifru
- zaútočit na asymetrickou šifru
- zaútočit na náhodný generátor: pokusit se uhodnout klíč N

Příklad (výzva a odpověď): Alice je uživatel a Bob server. Alice chce Boba přesvědčit, že zná heslo, ale nechce mu ho posílat. Udělají to následovně: Bob vygeneruje náhodný řetězec – *výzvu (challenge)*. Alice výzvu podepíše pomocí svého klíče, například zahešováním výzvy spojené s heslem, a pošle to jako *odpověď (response)*. Bob také zná heslo, takže může provést stejný výpočet a ověřit, že mu vyšlo totéž.

Příklad (přehrávací útok a nonce): Zamysleme se nad vlastnostmi výzvy v předchozím příkladu. Především se nesmí opakovat. Jinak by si totiž útočník mohl zaznamenávat všechny Aliciny odpovědi a když by později od Boba dostal tutéž výzvu, mohl by odpovědět zaznamenanou odpovědí. Tomu se říká *přehrávací útok (replay attack)* – útočník použil legitimní zprávu, ale v jiném kontextu.

To je v kryptografii velmi typická situace: potřebujeme generovat nějaké řetězce „na jedno použití“. Těm se říká *nonce* z anglického „number used once“. My si toto slovo

počeštíme a budeme ho skloňovat podle vzoru „růže“. Kde takovou nonci sehnat? Buďto si můžeme pořídit počítadlo a postupně ho zvyšovat; jak uvidíme v kapitole TODO, udržování stavu je v praxi dost problematické. Proto bývá jednodušší nonci generovat náhodně. Teoreticky se sice může zopakovat, ale jak uvidíme v oddílu TODO, stane se to s velmi malou pravděpodobností.

1.5 Alice a Bob na dražbě

Na závěr této kapitoly zkusíme z kryptografických primitiv poskládat jeden trochu složitější protokol. Budeme (trochu naivně) předpokládat, že primitiva jsou dokonalá, ale i tak se objeví nečekaně mnoho problémů. Pohodlně se usaďte, příběh začíná…

Alice se chce zúčastnit dražby, na níž se prodává originál plyšového tučňáka Tuxe. Dražba se ovšem koná kdesi v Antarktidě, tak se chce nechat zastupovat svým věrným přítelem Bobem. Bude sledovat průběh dražby v přímém televizním přenosu a když přijde správná chvíle, pošle Bobovi zprávu typu „přihoď x tolarů“, případně „skonči“. Tyto zprávy potřebuje šifrovat, protože jinak by ji konkurence mohla předběhnout; zároveň si ale uvědomíme, že zprávy stačí utajit na velmi krátkou dobu – nejpozději po několika minutách Bob požadovanou akci provede a bude zveřejněna v televizním přenosu.

Začneme triviálním protokolem, postupně budeme odhalovat jeho slabiny a opravovat je.

První pokus: potřebujeme padding

Použijeme symetrickou šifru. Než Bob odjede, domluví se s Alicí na nějakém klíči, ideálně náhodně vygenerovaném. Pokaždé když bude chtít Alice poslat zprávu, vytvoří řetězec tvaru `PRIHOD␣12345` nebo `KONEC`, zašifruje ho a odešle.

Co je na tomto protokolu špatně? Především z něj „prosakuje“ spousta informací. Jelikož symetrické šifry zachovávají délku zpráv, triviálně ze zašifrovaného textu poznáme, zda se jedná o `KONEC` – to je jediná pětiznaková zpráva. A pokud se jedná o `PRIHOD`, z počtu znaků zjistíme počet cifer částky, tedy její desítkový logaritmus.

Oprava je jednoduchá: všechny zprávy doplníme mezerami na nějaký pevný počet znaků. Tomu se říká *padding* neboli *vycpávka*. Důležité je, aby padding byl *reverzibilní* – když přijmeme opadovanou zprávu, musíme být schopni padding odstranit a získat zprávu původní. Jelikož naše zprávy jsou textové a nekončí na mezeru, stačí na jejich konec přidat mezery. Aliciny peněžní rezervy sotva tvoří víc než 10^{13} tolarů,⟨4⟩ takže stačí na částku rezervovat 13 číslic a na celou zprávu 20 znaků.

⟨4⟩ Jak jsme k číslu přišli: Ekonomové odhadují, že peněz v hotovosti a na snadno dostupných účtech je cca $8 \cdot 10^{12}$ USD. Viz *https://www.rankred.com/how-much-money-is-there-in-the-world/*.

Porovnávání zpráv: přidáme nonci

Překvapivě i po této úpravě z protokolu prosakují informace: Šifrování je deterministické, takže pokud Alice pošle podruhé tentýž příkaz, vznikne stejná zašifrovaná zpráva. Útočník přitom vidí v televizním přenosu, jak Bob zareagoval na kterou zašifrovanou zprávu, takže si může vytvářet slovník známých zpráv s jejich významy.

Pomoc je snadná: Před šifrováním ke zprávě přidáme nějakou nonci, třeba náhodný 64-bitový řetězec. Pak už bude velmi nepravděpodobné, že by útočník potkal tutéž zašifrovanou zprávu vícekrát.

Přehrávací útoky: hodí se sériová čísla

Co kdyby útočník zkusil nějaké zprávy vyrábět? Pokud si nějakou úplně vymyslí, nejspíš po dešifrování nebude dávat smysl. Daleko snazší je zopakovat nějakou autentickou zprávu a tím přimět Boba, aby zopakoval jeden z předchozích příkazů.

Nonce už tomu teoreticky brání: Bob si může pamatovat množinu všech noncí, které už potkal, a když se nějaká zopakuje, ví, že se jedná o duplikát zprávy, takže ho bude ignorovat. Nepraktické je, že na to potřebuje spoustu paměti. Zprávy proto doplníme *sériovým číslem* – počáteční zpráva má sériové číslo 0, každá další pak o 1 větší. Bob si pamatuje, jaké poslední sériové číslo viděl, a pokud nová zpráva nemá ostře větší, zahodí ji. (Všimněte si, že nevyžaduje přesně o 1 vyšší, takže ho nezmate vynechaná zpráva.)

Modifikace zpráv: podepisujeme

Další ošklivost, kterou by nám mohl útočník provádět, je obsah zpráv upravovat. Může se zdát, že pokud zašifrovanou zprávu jakkoliv změníme, způsobí to po dešifrování naprosté rozbití zprávy. To překvapivě často není pravda a z obecných požadavků na bezpečnost šifer to neplyne. Například pro všechny proudové šifry (TODO) platí, že překlopíme-li jeden bit v zašifrovaném textu, změní se bit na stejné pozici v dešifrované zprávě.

S tím se dá tropit nejrůznější neplecha. Především můžeme libovolně měnit sériové číslo: jeho původní hodnotu snadno známe (to je počet zpráv od počátku komunikace), takže víme, které bity změnit, abychom dostali o 1 vyšší číslo. Tím pádem sériová čísla vůbec nechrání proti přehrávání zpráv. Nebo můžeme sériové číslo nahradit nějakým hodně velkým (třeba překlopit jeho nejvyšší bit), takže Bob napříště zahodí všechny autentické zprávy od Alice.

Také můžeme upravovat částky. Pokud uhodneme, kolikamístnou částku Alice poslala (to se dá podle průběhu dražby tipovat), můžeme následující mezeru změnit na nulu a tím částku vynásobit deseti. Navíc první číslice bude často jednička,[5] a pokud si to útočník správně tipne, může ji změnit na jakoukoliv jinou číslici.

[5] Pravděpodobně se i zde uplatňuje tzv. Benfordův zákon.

Konečně můžeme zkusit změnit `PRIHOD` na `KONEC␣`, a tím donutit Boba odejít z dražby. Na to bychom ale potřebovali, aby Bob ignoroval přebytečné číslice za příkazem `KONEC`. Nebo pokud bychom uhodli, že Alice chce skončit, mohli bychom naopak z konce udělat jakékoliv přihození.

Všechny tyto problémy jsou důsledkem toho, že šifra nezaručuje nic o integritě zpráv. Přidáme proto ke každé zprávě ještě podpis. Jelikož obě strany už sdílí společné tajemství, můžeme podepisovat symetricky, tedy MACem. Podepisovat budeme až zašifrovanou zprávu.

Hotový protokol

Pojďme shrnout celý protokol. Zprávy budou vypadat takto:

- zašifrovaná část:
  - nonce (64 bitů)
  - sekvenční číslo (64 bitů)
  - text příkazu dopadovaný mezerami na 20 znaků
- podpis zašifrované části

Alice si bude udržovat počítadlo zpráv a podle něj nastavovat sekvenční číslo. Nonce bude generovat náhodně.

Bob si bude pamatovat poslední přijaté sekvenční číslo. Když mu přijde zpráva, nejdříve ověří podpis, a pokud nesouhlasí, zprávu zahodí. Jinak zprávu dešifruje, porovná její sekvenční číslo s posledním přijatým, a pokud nové není větší, zprávu zahodí. Pokud zpráva prošla i tímto textem, Bob ji považuje za autentickou a provede příkaz.

Překvapení na závěr

Alice je spokojená – vydražila svého milovaného Tuxe navzdory všem nepřátelům. Tak se při příští dražbě pokusí osvědčený protokol použít znovu. Ale ouha, nepřátelům se najednou daří podstrkovat příkazy, které Alice neposílala. Co je špatně? Inu, sériová čísla sice brání přehrávání zpráv v rámci jedné instance protokolu, ale už ne kopírování zpráv z jedné instance protokolu do druhé.

Potřebujeme tedy instance nějak odlišit. Jedna možnost je vygenerovat pro každou instanci nový klíč. Pokud se Alice s Bobem před každou instancí setká, mohou ho vygenerovat náhodně. Nesetkají-li se, mohou klíč generovat hešovací funkcí z nějakého hlavního klíče a identifikátoru instance (to může být třeba datum a čas začátku dražby).

Nebo můžeme identifikátory instancí přidávat do zpráv a nechat Boba zahazovat všechny zprávy, které nepatří k aktuální instanci. Vlastně je ani nemusíme se zprávou posílat –

stačí, když je započítáme do podpisu. Jakmile se pak objeví zpráva z jiné instance, prostě jí nebude souhlasit podpis.