Vysoká škola ekonomická v Praze

Bezpečnost informačních systémů: materiály ke cvičením 1

Luboš Pavlíček

Jiří Sedláček

2020

Autoři

Ing. Luboš Pavlíček

Vysoká škola ekonomická v Praze, Centrum informatiky

Ing. Jiří Sedláček, Ph.D.

Vysoká škola ekonomická v Praze, Katedra systémové analýzy

Oponenti

RNDr. Radomír Palovský, CSc.

Ing. Tomáš Zahradnický, Ph.D.

Poděkování

Tato publikace byla vytvořena za podpory projektu RP409030 IRS
Interní rozvojové soutěže Vysoké školy ekonomické v Praze.

ISBN 978-80-245-2403-0

Stručný obsah

Podrobný obsah

Seznam obrázků

Seznam tabulek

1 Hesla a heslová politika

V této kapitole se pokusíme odpovědět mimo jiné na následující otázky:

- Co je heslo, heslová fráze a PIN?
- Jak složité heslo byste měli používat?
- Co je to síla hesla a jak ji určit?
- Jaké jsou druhy útoků na hesla a modely hrozeb pro hesla?
- Jaký je rozdíl mezi lámáním a hádáním hesel?
- Proč se má heslo při ukládání solit? Kolik soli se má použít (velikost soli)?
- Proč se ve studijním systému požaduje heslo o minimální délce 10 znaků, zatímco na platební kartě se používá PIN o čtyřech číslicích?
- Proč by se neměla hesla pepřit?
- Jak se z hesla generují klíče pro symetrickou šifru?
- Které algoritmy použít pro ukládání hesel v souboru či v databázi?
- Proč nenavrhovat systémy zabezpečené pouze heslem?
- Jaká jsou současná doporučení týkající se hesel? Jaký je rozdíl mezi pojmy „pravidla pro tvorbu hesel“ a „heslová politika“?
- Co jsou programy pro odhad síly hesla, jaké jsou jejich přínosy a úskalí?
- Jaké existují druhy správců hesel, jejich výhody a nevýhody?
- Co je vícefaktorová autentizace a kdy se doporučuje její využití?

1.1 Co je heslo, heslová fráze či PIN?

Heslo (**password**) je utajovaný řetězec znaků, pomocí kterého se uživatel autentizuje vůči nějakému systému. Je použit princip *sdíleného tajemství* – heslo musí v nějaké podobě „znát“ obě strany.

Heslo je pravděpodobně *nejrozšířenější způsob autentizace uživatele* při práci s počítačem. Při přihlášení do systému uživatel obvykle zadává *uživatelské jméno* (login) a *heslo*. Uživatelské jméno je obvykle obecně známé, heslo by měl znát pouze uživatel sám. Heslo (předem domluvené slovo či slova) se používá i pro rozlišení osob ve tmě při strážní službě či v armádě pro rozlišení mezi vlastními a nepřátelskými vojáky.

Heslo je často *základem odvozování klíčů pro symetrické šifry* – pro zašifrování a dešifrování souborů s citlivými informacemi. Zde se častěji používá pojem **heslová fráze** (**passphrase**), aby se zdůraznil požadavek na větší délku hesla. Heslová fráze se obvykle skládá z více slov, často má více než 20 znaků.

PIN (**Personal Identification Number**) je variantou hesla, obvykle se jedná o čtyř-místné až šestimístné číslo, které se zadává při zapnutí/použití předmětu. PIN se používá u platební karty, mobilních telefonů či u vstupních systémů.[1]

V dalším textu bude *pojem heslo* zahrnovat také PINy a heslové fráze.

[1] Může být diskuse, zda se PIN používá k autentizaci či k autorizaci. Jeden z názorů je, že zadáním PINu k platební kartě se uživatel autorizuje k jejímu použití. A bankomat již nezkoumá, zda PIN zadal Petr či Pavel (neidentifikuje a neautentizuje uživatele). Ale při zadání uživatelského jména a hesla systém též nezkoumá, zda před monitorem sedí Petr či Pavel. Uživatel zadáním uživatelského jména sdělí určitou identitu, a tu zadáním hesla autentizuje. U problému PINu a autentizace převládá druhý názor – při zadání PINu k platební kartě se jedná o autentizaci oprávněného uživatele platební karty.

Tabulka 1.1: Definice pojmů heslo a heslová fráze dle standardu NIST-SP800-132

Heslo (password)	Posloupnost znaků, kterou zná pouze konkrétní entita (např. uživatel), a která se používá pro autentizaci (ověření) identity uživatele počítačového systému a/nebo pro autorizaci přístupu k systémovým zdrojům.
Heslová fráze (passphrase)	Množina slov (obvykle více než 20 znaků), která je použita pro autentizaci v počítačovém systému a/nebo pro autorizaci přístupu k systémovým zdrojům.

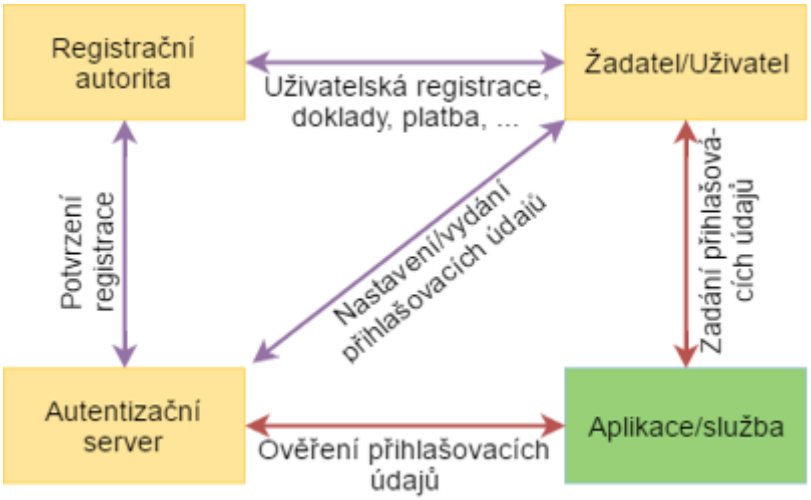
Zdroj: TURAN, Meltem, et al. NIST Special publication 800-132: *Recommendation for password-based key derivation*. Gaithersburg: NIST, Computer Security Division, Information Technology Laboratory, Technical Report, 2010.

1.2 Útoky na hesla

Před vlastní autentizací se uživatel musí zaregistrovat – uživatel požádá o založení účtu nebo je mu účet přidělen. Registrace může mít různou formu – žádost o založení účtu pomocí webového formuláře, založení pracovního poměru nebo vytvoření bankovního účtu.
Při registraci se mohou verifikovat zadávané údaje (např. e-mailová adresa) či provádět dodatečné kontroly (např. zda student studuje konkrétní předmět).

Účty spravuje autentizační server (služba), která může mít různé technické řešení. Mohou to být textové soubory (v UNIXu se používají soubory /etc/passwd a /etc/shadow), tabulka v databázi či adresářové služby (LDAP, Active Directory). Na autentizačním serveru se často ukládají i další údaje o uživateli z registrace – využívají se např. při obnově hesla, pokud ho uživatel zapomene.

Obrázek 1.1: Registrace a autentizace uživatelů



Zdroj: vlastní zpracování

Pro **ověřování přihlašovacích údajů** (**credentials**, obvykle jméno a heslo) se používají různé *autentizační protokoly*. Protokol může být jednoduché vyplnění přihlašovacích údajů do webového formuláře a odeslání aplikaci pomocí HTTP požadavku PUT. Existují specializované protokoly jako MSCHAP či EAP pro přihlašování do bezdrátových sítí i komplexní protokoly jako Kerberos, OAuth2 či SAML.

1.2.1 Skupiny útoků na hesla dle NIST 800-118

V dokumentu „Guide to Enterprise Password Management“ se popisují následující *skupiny útoků* na hesla. Platnost tohoto dokumentu z roku 2009 skončila v roce 2016, ale zde uvedené členění hrozeb je stále užitečné:[2]

- **Hrozby přímého získání hesla**. Hlavní charakteristikou je, že útočník získá heslo v *čitelné podobě*, útoky jsou *nezávislé na složitosti hesla*. Patří sem sledování osoby při psaní hesla (přímo či přes kameru), instalace hardwarových či softwarových keyloggerů, získání papíru/souborů se zaznamenaným heslem, sociální inženýrství (např. phishing). K přímému získání hesla může dojít i při přenosu pomocí nezabezpečeného kanálu – např. pokud se uživatel přihlašuje na webovou stránku přes nezabezpečený protokol http.
- **Hrozby zneužívající slabá hesla či slabé uložení hesel**. Útočník může simulovat přihlášení uživatele a zkoušet různá hesla (hádání hesel, *password guessing*). Lámání hesel (*password cracking*) popisuje situaci, kdy útočník získá přenášené či uložené haše[3] hesel a poté pomocí různých metod zkouší k těmto hašům najít odpovídající hesla.
- **Hrozby při změně či obnově hesla**. Útočníkovi se podaří změnit heslo na nové bez znalosti původního hesla. Tato hrozba je spojena s funkcemi jako je poslání zapomenutého hesla (*password recovery*), nastavení nového hesla správcem (*password reset*) či změna hesla (*password change*). Útočník i v těchto případech může používat techniky sociálního inženýrství (např. vydávat se za jinou osobu).
- **Hrozby zneužití kompromitovaného hesla.** Útočník může zkompromitované heslo použít jen do té doby, než si uživatel heslo změní – cílem je tuto dobu minimalizovat. Hrozbou je i situace, kdy uživatel používá stejné heslo ve více systémech – o kompromitaci hesla z externího systému se uživatel či správce nemusí dozvědět a nemůže na ni zareagovat. Nejjednodušším opatřením je nastavení pravidelné změny hesel. Ale krátký interval pro změnu může přinést více škody než užitku.

1.2.2 Model hrozeb pro hesla u webové aplikace

Při přihlašování do běžné webové aplikace si uživatel na své stanici spustí webový prohlížeč a otevře stránku webové aplikace. Poté do prohlížeče zadá uživatelské jméno a heslo. Toto heslo se otevřené přenese skrz zašifrovaný TLS kanál na aplikační server. Zde se heslo zahašuje a porovná s hašem uloženým v autentizační databázi.

John Steven z organizace OWASP vytvořil *model útoků na hesla*[4] pro jednoduché *webové aplikace*. V modelu se rozlišují dvě základní etapy útoku. V první útočník získá databázi haší hesel. V druhé se na základě toho snaží odvodit původní hesla. Model dále rozlišuje *šest základních kategorií hrozeb* (H1 až H6), v každé kategorii jsou pak uvedeny konkrétní metody, které útočníci využívají (tzv. *vektory hrozeb*). Mírně upravená verze je zobrazena na následujícím obrázku:

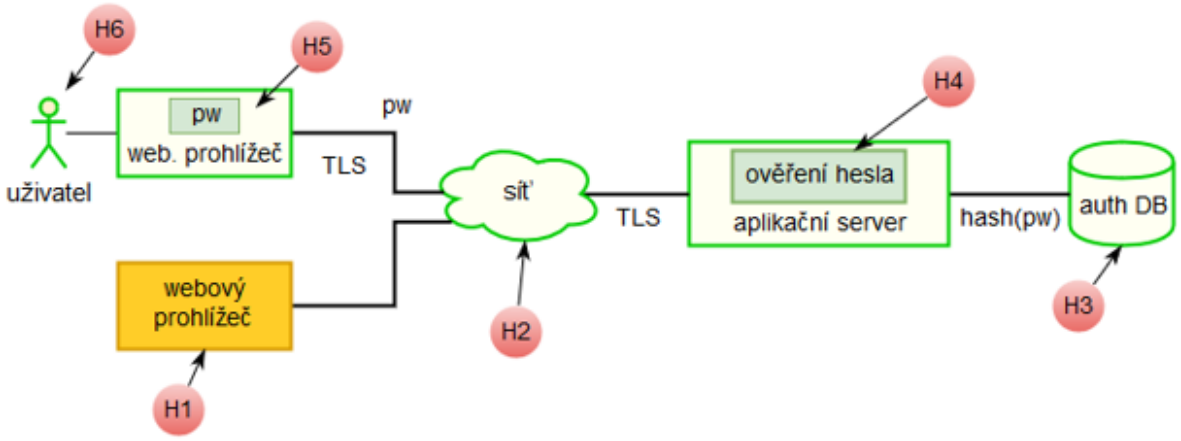
---

[2] SCARFONE, Karen, SOUPPAYA, Murugiah. Special Publication 800-118: Guide to Enterprise Password Management (Draft). Gaithersburg: NIST, 2009. [online] [cit. 2020-08-08]. Dostupné z WWW: < https://csrc.nist.gov/csrc/media/publications/sp/800-118/archive/2009-04-21/documents/draft-sp800-118.pdf >

[3] Hesla se obvykle *nepřenášejí* a neukládají v *otevřeném tvaru*: bližší rozbor je v dalších podkapitolách.

[4] STEVEN, John. *OWASP Threat Model for Secure Password Storage*. In: OWASP Foundation [online]. [cit. 2020-08-09]. Dostupné z WWW: <https://goo.gl/Spvzs>

Obrázek 1.2: Model útoků na hesla u běžné webové aplikace



Zdroj: vlastní zpracování (podle zdroje v poznámce pod čarou č. 4, upraveno)

Hrozby H1 – útočník k útoku využívá přístup k aplikaci z Internetu. Nejčastější vektory útoku jsou:

- Používá jakékoliv veřejně dostupné autentizační rozhraní pro hádání hesel. Může používat webový prohlížeč, specializovanou aplikaci či si napsat vlastní skript.
- Zkouší, zda je aplikace zranitelná na SQL injection. Pokud ano, snaží se získat databázi hesel nebo celou databázi, příp. získanou databázi použít k pokračování útoku.
- Zkouší další typy útoků jako XSS či CSRF pro získání či obejití autentizace.

Hrozby H2 – útoky typu MitM (Man in the Middle, muž uprostřed) na síti. Nejčastější vektory útoku v této skupině:

- Útočník pasivně odchytává HTTP provoz na síti a snaží se získat autentizační cookies.
- Útočník se snaží provoz směřovat přes svůj proxy server a přitom dešifrovat obsah zašifrovaného (HTTPS) provozu.

Hrozby H3 – útoky na databázi s autentizačními údaji. Nejčastější vektory útoky:

- Útočník získá síťový či lokální přístup k autentizační databázi a může z ní zcizit údaje (získá oprávnění administrátora databáze).
- Útočník získá přístup a data na úrovni operačního systému (zkopíruje soubory, upraví aplikace).
- Zcizení zálohy či neautorizovaná obnova dat ze zálohy.
- Odchytává síťový provoz mezi aplikačním serverem a autentizační databází.

Hrozby H4 – útoky na aplikační server. Vektory útoku:

- Úpravy nastavení aplikace – logování komunikace včetně hesel.
- Úpravy kódu aplikace a následné zachytávání otevřených hesel.

Hrozby H5 – útoky na počítač/prohlížeč oprávněného uživatele aplikace. Nejčastější vektory útoku jsou:

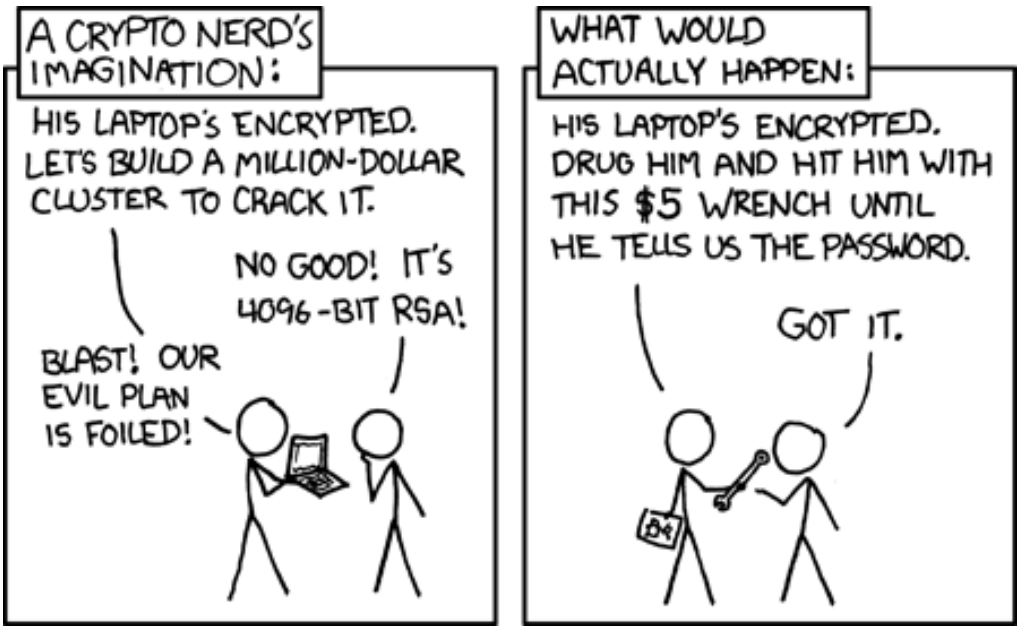
- Instalace softwarového keyloggeru (aplikace pro zaznamenávání stisků kláves, popř. i obsahu obrazovek).
- Získání hesla pomocí hardwarového keyloggeru.
- Úprava prostředí prohlížeče např. pomocí pluginů.
- Úprava operačního systému a odchytávání síťového provozu, získávání údajů z paměti prohlížeče či přesměrování prohlížeče na útočníkův proxy server.

Hrozby H6 – útoky na počítač oprávněného uživatele aplikace. Vektory útoky:

- Snímání činnosti uživatele pomocí kamery.
- Phishing – žádosti o zadání hesla na falešné stránky.
- Útoky na jiné servery, na kterých má oprávněný uživatel účet – zda nepoužívá stejné heslo.
- Osobní ovlivňování konkrétního uživatele – např. „pomohu Vám s problémem v aplikaci, ale potřebuji se přihlásit jako Vy“.
- Do této skupiny útoků patří i použití montážní páky.

Uvedený *model hrozeb je obecný*, pro konkrétní systém je vhodné vytvořit konkrétní model s přesně definovanými hrozbami. Následovat by měl podrobnější odhad rizik a návrh opatření pro omezení zranitelností, ale to je již mimo záběr tohoto textu.

Obrázek 1.3: „Praktický přístup“ k luštění hesel



Zdroj: Převzato z <https://xkcd.com/538/> (licence Creative Commons)

1.2.3 Náklady na získání cizího hesla

Má-li útočník přístup do prostředí, ve kterém zadává heslo oběť, tak se náklady na získání hesla *konkrétního* uživatele budou pohybovat v řádu tisíců korun. Kvalitní *skryté kamery* stojí od 5 tisíc Kč výše.[5] Běžný uživatel si této kamery nevšimne a útočník může sledovat jeho zadávání hesla na klávesnici.

Další možností je *hardwarový keylogger* – ceny začínají u 2 tisíc Kč. Hardwarového keyloggeru si obvykle nevšimne ani profesionál. V obou případech musí být útočník schopen umístit zařízení v prostorách, kde používá počítač potencionální oběť. Někdy je to velká překážka, v mnoha případech je to ale jednoduché a vyžaduje to po útočníkovi jenom trochu drzosti. Zvláštní kapitolou jsou veřejné prostory, za které lze považovat i učebny na školách.

[5] Podívejte se na nabídku nějakého obchodu se špionážní technikou, např. <https://www.spionazni-technika.cz/spionazni-a-skryte-kamery>.

Na straně útočníků též fungují *ekonomická pravidla* – případný zisk ze získaného hesla či hesel by měl být vyšší než náklady na jeho získání včetně započtení nákladů neúspěšných pokusů. Představte si, že hacker odněkud z Jižní Ameriky získá heslo do studijního systému jednoho studenta Vysoké školy ekonomické. Jak získané heslo zpeněží? Pravděpodobně nejziskovější pro něho bude použít školní účet pro rozesílání spamu. Příjmy ve výši jednotek korun, neboť podobné e-mailové účty pro rozesílání SPAMu se na černém trhu v roce 2013 prodávali za 0,01 USD.[6] Útočník může zkusit prodat přístup ke studijnímu systému na černém trhu za trochu vyšší částku – otázkou je, zda najde kupce.

Ceny hesel na černém trhu jsou nízké. V roce 2012 byly ze serveru LinkedIn zcizeny přihlašovací údaje, postupně se ukázalo, že až pro 167 miliónů účtů (údaje v literatuře se trochu liší, možná např. proto že část účtů již byla neaktivních). V roce 2016 byly k prodeji za přibližně jen 2200 USD, což znamená přibližně 0.001 centu za jeden účet.[7] Cena dále klesla, když LinkedIn rozeslal postiženým uživatelům žádost o změnu hesla. V únoru 2016 se prodávaly zcizené přístupové údaje ke službě NetFlix po 0,25 USD za kus a včetně 7denní garance funkčnosti.[8] V nabídce bylo přes 300 000 přístupových údajů (není jasné, zda všechny funkční).

Dle firmy Symantec byla většina přístupových údajů zcizena za pomocí phishingových kampaní. V červnu 2016 zveřejnila firma Kaspersky analýzu nabídek ze serveru xDedic.[9] Různí útočníci na něm nabízeli administrátorský přístup k 70 000 serverů po celém světě, cena se pohybovala mezi 6 a 8 USD za jeden (na trhu byl i jeden server z VŠE). Odhaduje se, že útočníci získali přístup díky chybám software (obvykle správce neprovádí bezpečnostní aktualizace) či účty na serverech měly slabá hesla a útočníci je uhodli.

Je zřejmé, že útočníci jsou schopni získávat přístupové údaje s nízkými náklady. Vynaložit několik tisíc korun na získání jednoho hesla se vyplatí pouze ve výjimečných případech – obvykle lidem, kteří dobře znají oběť, tj. spolupracovníkům, členům rodiny, pracovníkům konkurenčních firem apod. Tito útočníci znají prostředí, jsou schopni lépe ohodnotit informace dostupné v informačním systému. Mohou mít i jiné než finanční motivy pro zneužití přístupu na cizí účet – získání kompromitujících údajů, diskreditace konkrétní osoby, psychický nátlak[10] apod.

---

[6] *Buying Battles in the War on Twitter Spam* [online]. 2013-08-14 [cit. 2020-08-08]. Dostupné z WWW: <https://krebsonsecurity.com/2013/08/buying-battles-in-the-war-on-twitter-spam/>.

[7] HUNT, Troy: *Observations and thoughts on the LinkedIn data breach* [online]. 2016-05-24 [cit. 2020-10-25]. Dostupné z WWW: <https://www.troyhunt.com/observations-and-thoughts-on-the-linkedin-data-breach/>.

[8] MOGG, Trevor: *Netflix has a black market for passwords, and they sell for just 25 cents* [online]. 2016-02-12 [cit. 2020-10-25]. Dostupné z WWW: <https://www.digitaltrends.com/home-theater/netflix-black-market/>.

[9] Kaspersky Lab: *xDedic – the shady world of hacked servers for sale* [online]. 2016-06-15 [cit. 2020-10-25]. Dostupné z WWW: <https://securelist.com/xdedic-the-shady-world-of-hacked-servers-for-sale/75027/>.

[10] Příklad z VŠE – jedna studentka měnila jiné studentce termíny zkoušek či mazala odevzdané práce s cílem dotyčnou psychicky rozhodit.

Obrázek 1.4: Nabídka účtů služby NetFlix na černém trhu

Zdroj: obrázek z článku v poznámce pod čarou č. 8.

1.2.4 Proč uživatelé používají slabá hesla?

Proč uživatelé používají jednoduchá hesla či stejné heslo na více serverech? Obecně, proč *nedodržují bezpečnostní doporučení* a nechovají se bezpečně v kyberprostoru? *Cormac Herley* z Microsoft Research představil v roce 2009 hypotézu, že uživatelé se ve věcech počítačové bezpečnosti chovají *vlastně racionálně*, neboť se snaží minimalizovat náklady a maximalizovat užitek. Herley rozebírá běžná bezpečnostní doporučení s ohledem na ekonomické přínosy a náklady. Svoji hypotézu demonstruje na podvodných transakcích v případě internetového platebního systému *Paypal*.[11]

CISO (Chief Information Security Officer) Paypalu za rok 2008 uvádí, že podvodné transakce byly maximálně v objemu 0,49 % z obratu společnosti, tj. dosáhli maximálně částky 290 miliónů USD. Při 70 miliónech uživatelů je to 4,14 USD na uživatele. Znamená to necelých 14 minut času, pokud částku přepočteme pomocí průměrné hodinové mzdy v USA v roce 2008 (18,11 USD). 14 minut času je poměrně hodně na vymyšlení a zapamatování složitého hesla. Tento čas je možné snížit na polovinu, protože mnoho podvodů je nezávislých na kvalitě hesla – např. phishing (ten Herley analyzuje v samostatné části), keyloggery nebo úmyslné pokusy o podvod vlastníků účtů.

Do úvahy je ale třeba vzít *rozdělení ztráty mezi uživatele a Paypal* – většinu hradí firma, protože vyplacení pár tisíc dolarů je v konkrétním případě menší zlo než ztráta důvěry veřejnosti ve služby firmy. Nejde jen o ztrátu důvěry – náklady na prokázání viny na straně uživatele by mnohdy překračovaly ztrátu z konkrétního podvodu. Odhaduje se, že Paypal hradí přes 90 % ztrát z podvodů, na uživatele zbývá maximálně 10 % ztráty. Na vytváření hesla a jeho zabezpečení by měl věnovat maximálně jednu minutu.

To platí *jen v průměru:* osoba, které byl účet vybrán, by jistě zpětně ráda investovala do zabezpečení více než minutu času (zvláště když v realitě typicky neutrpí jen finanční

[11] HERLEY, Cormac. So long, and no thanks for the externalities: the rational rejection of security advice by users. In: *Proceedings of the 2009 workshop on New security paradigms workshop (NSPW '09)*. New York, ACM, 2009. s. 133–144.

ztrátu, ale též významnou časovou ztrátu a další újmy). Je zřejmé, že pro Paypal by bylo výhodné, kdyby uživatelé investovali více času do ochrany hesla a účtu, ale oni k tomu mají minimální motivaci.[12]

Ve studii „Is Everything We Know About Password-Stealing Wrong“[13] z roku 2012 Herley tyto myšlenky dále rozvíjí a na konkrétních případech ukazuje, že při krádežích peněz z účtu neztrácí peníze vlastník účtu se slabým heslem a ani banka, ale „bílý kůň“, přes jehož účet útočník pere peníze.

Tabulka 1.2: Rozdělení zisků a ztrát při krádeži peněz z účtu (zcizené přístupové údaje)

Kdo	**Role dané osoby**	**USD před odhalením**	**USD po odhalení**
Oběť	osoba, které byly zcizeny peníze z účtu	–9000	0
Banka	banka, u které měla oběť účet	0	0
Bílý kůň	osoba, přes jejíž účet byly peníze vyprány	+900	–8100
Útočník	osoba, která zcizila přístupové údaje a získala peníze	+8100	+8100

Zdroj: vlastní zpracování (údaje převzaty ze studie, viz poznámka 13)

Dodržování bezpečnostních doporučení běžnými uživateli *zlepšuje bezpečnost* (otázka je o kolik, protože pokrývají jen malou část případných vektorů útoku), ale mnohdy je drahé *na čas a úsilí* vzhledem k reálně hrozícím škodám a pravděpodobnosti jejich výskytu. Proto uživatelé *ignorují* základní bezpečnostní návyky – *intuitivně* došli ke správnému závěru, že se jim spíš vyplatí smířit se s občasnou ztrátou, než věnovat čas a úsilí na vytváření a zapamatování složitých hesel, což je pro ně jistá ztráta, byť v jiné formě.

K tématu se vrátíme ještě v kapitole 1.6 (která se zabývá pravidly pro tvorbu hesel a heslovou politikou) a speciálně pak v kapitole 1.6.6, která analyzuje schopnost uživatelů zapamatovat si (složitá) hesla. V následující podkapitole se zaměříme na jeden z nejčastějších *útoků na hesla* – na *hádání hesel*.

1.2.5 Základní způsoby hádání hesel

Při **hádání hesla (guessing)** se útočník *zkouší* přihlásit jako oprávněný uživatel systému: zadá uživatelské jméno a heslo. Základním předpokladem pro hádání hesel je přístup k autentizačnímu rozhraní příslušné služby.

Útočník může zkoušet *výchozí heslo* nastavené výrobcem – přihlášení se poměrně často podaří u hardwarových zařízení (domácí směrovač, tiskárna, kamera ap.). Další možností je *slovníkový útok* – postupně se zkouší hesla z předpřipraveného seznamu. Útočník může zkoušená hesla přizpůsobit pravidlům pro heslo (pokud je požadováno minimálně jedno velké písmeno a jedna číslice, tak ze slova `password` vytvoří `Password1`) či vytváří specifický slovník pro konkrétní službu (např. vytvoří slovník ze slov na webech firmy).[14]

Útok *hrubou silou*, tj. vyzkoušení všech možných kombinací je praktické pouze v případě malého počtu možných kombinací – např. PIN domovnímu systému či mobilu. V těchto případech by útok byl neúspěšný, neboť domovní systémy či mobily se většinou brání – po větším počtu neúspěšných pokusů se spustí alarm či zablokuje přístup.

[12] S nástupem ransomware se situace trochu změnila.
[13] FLORENCIO, Dinei, HERLEY, Cormac. Is everything we know about password stealing wrong? *IEEE Security & Privacy*, 2012, 10.6: 63–69.
[14] Viz např. <https://bitbucket.org/mattinfosec/wordhound>.

Je *zásadní rozdíl*, zda útočník útočí na jeden konkrétní účet či na větší skupiny účtů. Pokud uživatelé používají náhodně vygenerovaný čtyřmístný PIN a systém po třech neúspěšných pokusech zablokuje přístup do dalšího dne (obvyklé chování v případě platebních karet), tak lze správný PIN zjistit za 4,6 roku. Pokud může útočník vyzkoušet 20 000 různých účtů chráněných PINem se stejným mechanismem obrany, tak v průměru získá přístup k šesti účtům každý den!

Pokud si heslo *vybírá sám uživatel*, tak je situace pro útočníka mnohem výhodnější. Ve studii "A birthday present every eleven wallets? The security of customer-chosen banking PINs"[15] autoři analyzují PINy z iPhonů a na základě toho odhadují bezpečnost uživateli vytvářených PINů. U příkladu z předchozího odstavce by útočník za první den získal přístup ke 288 účtům (1,44 %).[16] Pokud by znal datum narození jednotlivých uživatelů, tak by za dva dny (6 pokusů) získal přístup k 1626 účtům (8,23 %).[17]

Z druhého údaje vychází i závěrečný odhad autorů: když útočník ukradne platební kartu 11 osobám a zjistí také datum narození (např. ukradne peněženku včetně osobních dokladů), tak při šesti pokusech o ověření (zkouší opět dva dny) může *v průměru* vybrat peníze jedné z jedenácti ukradených karet.

Dwayne Charrington se přihlásil na tři z pěti iPhonů (iPhone má omezení na tři chybné pokusy), když zkoušel následující PINy:[18]

1. datum narození,
2. rok narození,
3. nejčastější čísla jako 1234, 000, 1111 či čísla odvozené z číselné klávesnice (např. 1379, 2468, 1357 apod.),
4. datum narození děti.

K problematice hesel zvolených uživateli se vrátíme v kapitole 1.4.3.

1.2.6 Obrana před hádáním hesel

Omezení možností hádání hesel je *klíčové pro bezpečnost* informačního systému. Základní *způsoby obrany* jsou následující:

- Použití *dostatečně složitých hesel*, které odolají většímu počtu pokusu o hádání, a to i tehdy, kdy útočník má doplňující informace o osobě.
- Omezit *počet pokusů* o přihlášení.
- Omezení *míst*, ze kterých se uživatel může přihlašovat (např. jen z firemní sítě či přes VPNku). V některých situacích lze vymezit *i čas*, kdy se uživatel může přihlašovat.
- Nasazení vícefaktorové autentizace.

---

[15] BONNEAU, Joseph, PREIBUSCH, Sören; ANDERSON, Ross. A birthday present every eleven wallets? The security of customer-chosen banking PINs. In: *Financial Cryptography and Data Security*. Berlin, Heidelberg: Springer, 2012. s. 25–40.

[16] Porovnejte se 6 účty, které útočník získá za stejných podmínek v případě, že PINy jsou náhodně generovány. Lidé při vytváření PINů některé kombinace preferují – čtyři stejné číslice (např. 0000), posloupnosti (např. 1234), roky a data či linie z numerické klávesnice (např. 8520).

[17] Pokud útočník získá kartu osoby narozené 3. června 1983, tak dle autorů studie je optimální strategie postupně zkoušet následující PINy: 1983, 6383, 0306, 0603, 1234 a 0683. U nás bude pravděpodobně častější 3683 než 6383 a 0306 bude častější než 0603, neboť jsme zvyklí psát den následovaný měsícem. Do seznamu bych též doplnil poslední čtyři číslice z rodného čísla.

[18] CHARRINGTON, Dwayne. *The Most Common iPhone Passcodes (and how to guess them)* [online]. 2014-09-22 [cit. 2020-10-25]. Dostupné z WWW: <https://ilikekillnerds.com/2014/09/the-most-common-iphone-passcodes-and-how-to-guess-them/>.

Omezení počtu pokusů může být jednoduché – při zadání chybného hesla systém odpoví s prodlevou 30 až 60 vteřin. Prodlevu lze zvětšovat exponenciálně – po prvním neúspěšném pokusu bude vteřinová prodleva, po druhém dvě vteřiny, po třetím čtyři vteřiny, poté postupně 8, 16, 32, 64, 128 atd. vteřin. Některé systémy po několika neúspěšných pokusech zablokují přístup na několik hodin – např. po 6 neúspěšných pokusech z konkrétní IP adresy se zablokuje přístup z této IP adresy na 4 hodiny.

Takovéto omezení však může vést *k zablokování přístupu pro oprávněné osoby*. Když dle schématu výše útočník udělá 12 pokusů, prodleva se zvýší na 4096 vteřin (asi 68 minut). Pokud se v této situaci zkusí přihlásit oprávněný uživatel a nezadá-li heslo správně napoprvé, může se znovu přihlašovat až za více než 2 hodiny. Vhodnější je nastavit nízký strop prodlevy (např. na 32 vteřin), pokud se uživatel zkouší přihlásit z IP adresy, ze které se v minulosti již úspěšně přihlašoval. Jinou variantou je po třech neúspěšných pokusech požadovat po uživateli tzv. captchu, která obvykle omezí automatizované hádání hesel. Při přihlašování lze zkoumat prodlevy mezi stisky kláves či pohyby myši a dle toho vyhodnocovat, zda se přihlašuje člověk či nějaký automat (úspěšně využívá např. *Seznam.cz* pro omezení automatizovaného vytváření mailových účtů, které jsou zneužívány pro spam).[19]

1.3 Zcizení a lámání hašů hesel

Na serveru jsou hesla obvykle uložena v zahašovaném tvaru – pomocí *kryptografické* hašovací funkce převádějí do řetězce znaků, ze kterého nelze zpět odvodit heslo.

Hašovací funkce je *jednosměrný algoritmus*, který převede libovolný vstupní řetězec či soubor na otisk *pevné délky* (pro danou funkci). Při přihlašování se uživatelem zadané heslo zahašuje stejnou funkcí a nová haš se porovná s jeho uloženým hašem.

Tabulka 1.3: Hašovací funkce: ukázky pro různé vstupy

Funkce	Vstup	Haš (otisk, výstup hašovací funkce)
SHA-256	heslo hesla password	56B1DB8133D9EB398AABD376F07BF8AB5FC584EA0B8BD6A1770200CB613CA005 F75BB5A0503CE49B7D314C8F0F0BF0C5116CF28E1D4D107791DDE11CDDE60850 5E884898DA28047151D0E56F8DC6292773603D0D6AABBDD62A11EF721D1542D8
SHA-1	heslo hesla password	6E017B5464F820A6C1BB5E9F6D711A667A80D8EA 74C65DB635A129AD1268CBD4258948955DDA4D39 5BAA61E4C9B93F3F0682250B6CF8331B7EE68FD8

Heslo převede na „nesrozumitelnou" a nepředvídatelnou posloupnost bytů pevné délky
Zdroj: vlastní zpracování

1.3.1 Zcizení uložených či přenášených hašů hesel

Útočník může získat haše **uložené na disku** (soubor, databáze, registry. Při návrhu aplikace je vhodné počítat s tím, že se v ní nebo v zabezpečení serveru může objevit chyba (např. SQL Injection v aplikaci či útočník zjistí přihlašovací údaje administrátora) a útočník získá hesla v zahašovaném tvaru. Jak hesla uložit, aby byla dokonce *i po zcizení v bezpečí*, popisuje podrobněji kapitola 1.5.

[19] Slížek, David. *Seznam.cz při registraci sleduje stisky kláves, aby odhalil roboty* [online]. 2016-07-17 [cit. 2020-10-25]. Dostupné z WWW: <https://www.lupa.cz/clanky/seznam-cz-pri-registraci-sleduje-stisky-klaves-aby-odhalil-roboty/>. V článku z roku 2016 se zvažuje využití rozšířit, ale to by v současnosti mohlo být velmi problematické kvůli nové právní regulaci (GDPR).

Hesla lze získat i **při přenosu** pomocí odposlouchávání síťového provozu např. na otevřené bezdrátové síti. Možností obrany je více: v současné době se nejčastěji používá vytvoření bezpečného kanálu pro přenos informací. Například HTTP komunikace se umístí do TLS kanálu (protokol HTTPS).

V minulosti se často chránila *jen hesla* – po síti se posílal jejich haš, ostatní data nebyla šifrována. Při používání poštovních protokolů (IMAP, POP3 či SMTP) se můžete setkat s autentizačním protokolem **CRAM-MD5** (Challenge-Response Authentication Mechanism), který používá algoritmus HMAC-MD5.[20]

V současnosti se naopak *preferuje šifrovat veškerý provoz*, a to i v případě, že samotná data neobsahují nikterak citlivé údaje (většina „běžných" WWW stránek). Výhodou je též to, že není sledován obsah, který si prohlížíte, a hlavně nemůže být „po cestě" změněn. Již v říjnu 2019 přesáhl podíl šifrovaného provozu webových stránek (HTTPS) 90 procent.[21]

V další části budou popsány postupy, jak haše hesel převést na původní heslo, tj. jak se lámou haše hesla (v angličtině password cracking, též offline attack).

1.3.2 Lámání hašů hesel

Při **lámání hašů hesel** (**password cracking**, česky též **louskání hesel** či **prolamování hesel**) je postup velmi podobný jako při přihlašování uživatele – útočník vezme nějaký řetězec, z něho vypočítá haš a výsledek porovná se získaným hašem. Lámání je výrazně rychlejší než hádání hesel – není omezováno síťovou komunikací, lze využít více procesorů a počítačů, není omezen počet pokusů, nezpomalují se záměrně odpovědi, lze optimalizovat algoritmus.

1.3.3 Útok hrubou silou a slovníkový útok

Při **útoku hrubou silou** útočník postupně zkouší všechny možné kombinace znaků hesla. Tento útok je reálný u krátkých hesel (do 7 či 8 znaků, závisí na způsobu uložení).
Při **slovníkovém útoku** si útočník připraví slovníky s možnými hesly. Ze slov ve slovníku postupně vytváří haše a porovnává je se získaným hašem. Většinou je slovníkový útok doplněn o **pravidla**, které ze slova ve slovníku vytváří další kombinace.

Příkladem může být pravidlo, které první písmeno převede na velké a na konec slova postupně doplňuje jednotlivé číslice. Na základě slova *heslo* poté útočník vyzkouší *heslo0, heslo1, heslo2, …, heslo8, heslo9,* poté *Heslo0, Heslo1, Heslo2, …, Heslo8* a *Heslo9*. Dále *heslo00, heslo01*, …, *heslo99, Heslo00, Heslo01*, … *Heslo99* atd.

Používané slovníky a pravidla lze různě *optimalizovat* pro lámání konkrétních hesel:

- Vytváření slovníku z *cílového serveru* – slovník ze jmen či adres nalezených v databázi, slova a slovní spojení z webu serveru.
- Výběr slovníku *dle oboru webu* (při lámání hesel z webu věnovaných filmům si útočník stáhne názvy filmů a jména herců z IMDB nebo obdobných českých webů).[22]
- Výběr dalších pravidel na základě již odhalených hesel (*adaptivní algoritmy*).

[20] Útočník musí odchytit výzvu (Challenge) i odpověď (Response).
[21] HTTPS encryption traffic on the Internet has exceeded 90% [online]. [cit. 2020-10-10]. Dostupné z WWW: <https://meterpreter.org/https-encryption-traffic/>
[22] The Internet Movie Database: <https://www.imdb.com>, Česko-Slovenská filmová databáze: <https://www.csfd.cz>, Filmová databáze: <https://www.fdb.cz>

- Úpravy na základě dodatečných *informací o konkrétní osobě*.[23] Příklad – útočník na Facebooku uživatele zjistí, že je to velký příznivec seriálu *Hra o trůny*, a tak z opensubtitles.org stáhne české a anglické titulky k tomuto seriálu a vytvoří z nich pomocný slovník.

Tabulka 1.4: Srovnání principu útoku hrubou silou a slovníkového útoku

Útok hrubou silou		*Slovníkový útok*	
Trying aaaaaaaa	: failed	Trying jablko	: failed
Trying aaaaaaab	: failed	Trying jablko1	: failed
Trying aaaaaaac	: failed	…	
Trying aaaaaaad	: failed	Trying jablko9	: failed
Trying aaaaaaae	: failed	…	
Trying aaaaaaaf	: failed	Trying jablko99	: failed
Trying aaaaaaag	: failed	Trying jahoda	: failed
Trying aaaaaaah	: failed	…	
Trying aaaaaaai	: failed	Trying jahoda99	: failed
Trying aaaaaaaj	: failed	Trying justinbeiber	: failed
...		...	
Trying aaaaacda	: failed	Trying Barcelona	: failed
Trying aaaaacdb	: failed	Trying BarcelonA	: failed
Trying aaaaacdc	: failed	…	
Trying aaaaacdd	: success!	Trying henry123	: success!

Zdroj: vlastní zpracování

1.3.4 Předpřipravené tabulky: duhové tabulky

Útočník si může *spočítat haše dopředu* a uložit je do databáze. Vyhledávání hesel k hašům je poté rychlejší. Ukládání je náročné na diskový prostor. V praxi se většinou používají tzv. **duhové tabulky** (**rainbow tables**), které se snaží *o kompromis* mezi výpočetní náročností útoku hrubou silou a prostorovou náročností předem vypočtených tabulek.[24] Pro uložení všech NT hašů pro 9 znakové heslo (malá velká písmena a číslice) byste potřebovali přibližně 500 TB, při použití formátu duhových tabulek postačuje přibližně 1 TB.

Duhové tabulky si nemusíte generovat sami, na https://www.freerainbowtables.com/, jsou mnohé ke stažení (např. pro LM haše, MD5 haše, SHA-1 včetně varianty používané v rámci databázového systému MySQL, pro NTLM ad.). Můžete najít i placené nabídky různých duhových tabulek. Předpřipravené tabulky fungují dobře v případě, že se ze stejného hesla u různých uživatelů vygeneruje stejný haš.

Princip hledání hašů hesel v předpřipravených duhových tabulkách ilustruje následující Tabulka 1.5 na straně 23. Základní obranou proti duhovým tabulkám je *solení hesel*, které bude popsáno v kapitole 1.5.2.

[23] BONNEAU, Joseph. The science of guessing: analyzing an anonymized corpus of 70 million passwords. In: *The Security and Privacy (SP), 2012 IEEE Symposium on*, 2012, p. 538–552.
[24] Popis duhových tabulek je např. v článku *Rainbow tables tajemství zbavené* [online]. [cit. 2020-08-08] Dostupné z WWW: <https://www.soom.cz/clanky/1165--Rainbow-tables-tajemstvi-zbavene>.

Tabulka 1.5: Princip hledání hašů hesel v předpřipravených tabulkách

```
Searching: fb3ff3a2621621f09190eea4cddb67fd: FOUND: 123456ahoj
Searching: 6cbe615c106f422d23669b610b564800:  not in database
Searching: ee1c595fe2fdc9ba40a0aeb46bbaedbd: FOUND: Hermiona
Searching: 5d5638c09d4e1ce6a43a614bbc194082: FOUND: JamesBond007
Searching: 8e605ece1b6cbcfbae2673252ef9efd9: FOUND: linkinpark1
```

Zdroj: vlastní zpracování

1.4 Síla hesla

Síla hesla označuje potřebný počet pokusů, které musí *v průměru* útočník provést pro získání hesla z jeho haše. Potřebný počet pokusů ovlivňuje *délka hesla*, velikost *množiny znaků* použitých pro sestavení hesla, a zda je heslo vytvořeno *náhodně* či je použit nějaký *předvídatelný postup*.

Entropie se používá pro vyjádření variability, v informatice se obvykle vyjadřuje pomocí logaritmu o základu 2 ($\log_2$). Používá se též pro vyjádření síly hesel. PIN o čtyřech číslicích lze sestavit 10 000 způsoby – na každém místě je 10 možností, vzájemně jsou nezávislé, takže možnosti vynásobíme mezi sebou: 10 * 10 * 10 * 10. tj. 10^4. Entropie je $\log_2$ z počtu možností, tj.

$$\log_2 10^4 = \ 13.29 \text{ bit}$$

V Excelu vzorec vypadá takto:

=LOGZ(10^4;2)

Pokud má heslo sílu (entropii) 42 bitů, znamená to, že útočník při útoku hrubou silou musí spočítat 2^{42} hašů, aby ověřil všechny možnosti.[25] V mnoha výpočtech spojených s hesly lze místo entropie používat počet možných hesel. Pro entropii hovoří dva důvody:

- lidem se lépe pracuje s malými čísly,
- entropie se používá také při odhadech síly hesel vytvářených uživateli.

Používání entropie pro vyjádření síly hesla přidává další úroveň složitosti – je nutné si uvědomit, že přidání bitu síly hesla znamená, že útočník musí *zdvojnásobit* počet pokusů.

1.4.1 Náhodná hesla

V případě *náhodně generovaných hesel* závisí síla hesla na velikosti slovníku použitého pro tvorbu hesla, popř. na pravidlech omezujících tvorbu hesla (password policy). Slovník obvykle obsahuje symboly, ze kterých jsou hesla složena – nejčastěji to jsou písmena a číslice, ale mohou to být slova (vytváření heslové fráze) či např. obrázky (vybrat obrázky a sestavit z nich řadu pro ověření přístupu). Pokud heslo vznikne náhodným výběrem L symbolů z množiny o velikosti N, lze entropii H spočítat pomocí:

$$\mathrm{H} = \log_2 \mathrm{N}^{\mathrm{L}} = \mathrm{L} \log_2 \mathrm{N} = \mathrm{L} \frac{\log \mathrm{N}}{\log 2}$$

Pokud vytvoříme heslo ze šesti náhodně vybraných malých písmen anglické abecedy (26 znaků), bude mít entropii:

$$6 * \log_2 26 = 6 * 4.70 = 28.20 \text{ bit}$$

[25] Konkrétní heslo může útočník odhalit po pár pokusech na začátku lámání, ale i téměř na konci po vyzkoušení většiny možností. V průměru na získání hesla stačí polovina pokusů, tedy 2^{41} pokusů.

Minimální délku hesla pro požadovanou entropii spočteme dle vzorce:

$$L = \frac{H}{\log_2 N}$$

výsledek se zaokrouhlí nahoru na celé číslo. Následující tabulka obsahuje minimální délky hesel pro různé množiny znaků použité při náhodném generování hesel. Z tabulky je zřejmé, že se zvětšující se velikostí použité množiny znaků stačí na heslo o stejné entropii menší počet znaků. A samozřejmě čím delší heslo, tím je silnější.

Tabulka 1.6: Počet symbolů z dané abecedy pro získání hesla s požadovanou entropií

požadovaná entropie H v bitech	číslice 0–9	hex. číslice 0–9, A-F	latinská písmena bez rozlišení velikosti [a–z]	číslice, malá a velká písmena, [a–z, A–Z, 0–9]	tisknutelné znaky ASCII tabulky	diceware[26]
	10	***16***	***26***	***62***	***95***	***7776***
16	5	4	4	3	3	2
32	10	8	7	6	5	3
48	15	12	11	9	8	4
64	20	16	14	11	10	5
80	25	20	18	14	13	7
96	29	24	21	17	15	8
112	34	28	24	19	18	9
128	39	32	28	22	20	10
160	49	40	35	27	25	13
192	58	48	41	33	30	15
224	68	56	48	38	35	18
256	78	64	55	43	39	20

Zdroj: vlastní zpracování

1.4.2 Příklady výpočtů pro náhodně generovaná hesla

Zadání 1

InSIS před několika lety náhodně generoval počáteční hesla dle schématu nnxxxnxxX *(dvě číslice, tři malá písmena, jedna číslice, dvě malá písmena, velké písmeno). Jaká je entropie těchto hesel? Kolikrát se zkrátí čas lámání těchto hašů hesel proti testování všech 9znakových kombinací hesel ze 62 znaků ([a–zA–Z0–9])?*

Řešení

Schéma *nnxxxnxxX* – na prvním místě může být jedna z 10 číslic, na druhém opět jedna z deseti číslic, na třetím místě jedno z 26 malých písmen atd. Počet kombinací je následující:

$$10 * 10 * 26 * 26 * 26 * 10 * 26 * 26 * 26 = 10^3 * 26^6$$

[26] Diceware – vytváření hesla (heslové fráze) náhodným výběrem z množiny 7776 slov. Popis viz <https://world.std.com/~reinhold/diceware.html>.

Hesla se generují náhodně, tj. všechny kombinace jsou stejně pravděpodobné, proto lze pro výpočet entropie použít opět dvojkový logaritmus:

$$e = \log_2(10^3 * 26^6) \approx 38.2$$

Entropie stejně dlouhého náhodně generovaného hesla z 62 znaků je:

$$e = \log_2 62^9 \approx 53.6$$

Rozdíl je 15 bitů, tj. čas se zkrátí přibližně $2^{15} \approx 32\,000$ krát. Tj. pokud by heslo náhodně generované dle zadaného schématu „padlo“ za jednu hodinu, tak by stejně dlouhé heslo náhodně sestavené z malých, velkých písmen a číslic „padlo“ za přibližně 4 roky.

Zadání 2

Máte 8 znaková hesla složená z malých a velkých písmen a z číslic a chcete zlepšit odolnost vůči lámání hrubou silou. Zvažujete dvě varianty: zvětšení minimálního počtu znaků na 9 či rozšíření abecedy o netisknutelné znaky. Která varianta bude mít větší entropii?

Řešení

Hrubý výsledek se získá z porovnání hodnot 62^9 (rozšíření na 9 znaků hesla) a 95^8 (všechny netisknutelné znaky). V první variantě musí být aspoň jedna číslice, aspoň jedno malé písmeno a aspoň jedno velké písmeno. O zbývajících 6 znacích nic nevíme, tj. může tam být libovolný z 62 znaků z množiny.

$$10 * 26 * 26 * 62^6$$

Podobně je potřeba upřesnit výpočet pro druhou variantu:

$$10 * 26 * 26 * 33 * 95^4$$

Z výsledků obou variant již snadno zjistíte odpověď na otázku. Pokud si heslo vybírá sám uživatel, tak se množina 33 nepísmenných a nečíselných znaků zmenší na 6 až 10 speciálních znaků snadno dostupných na klávesnici.

1.4.3 Uživatelem vytvářená hesla

Uživatelé obvykle *nepoužívají náhodně generovaná hesla*, ale heslo vytvářejí, což by mělo usnadnit jeho zapamatování. Většina uživatelů vychází z jim známých slov. Jednotlivá písmena jsou na sobě závislá – pokud je první písmeno „a“, tak v češtině bude na druhém místě nejčastěji „n“ (22,4 %), nebo „r“ (10,5 %) či „l“ (10,2 %). Písmeno „e“ bude na druhém místě za a jen s pravděpodobností 0,6 % – písmeno „e“ je přitom v češtině nejčastější. Tyto závislosti výrazně snižují entropii hesel a usnadňují útočníkům jejich odhalování.

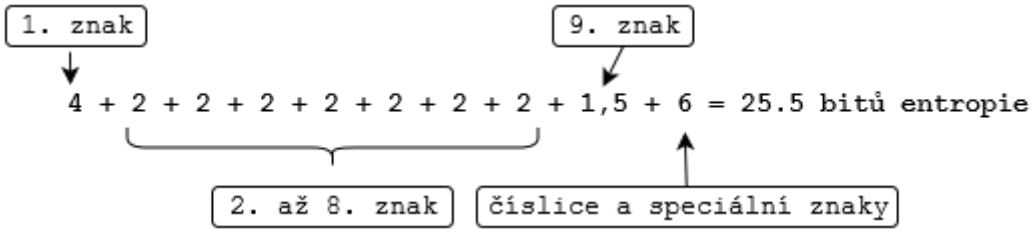
Počítat entropii pro závislé jevy je složité. Asi je lepší mluvit o *odhadu entropie*. Dnes *již neplatná*, ale řadu let používaná, verze publikace NIST (NIST 800-63 *Electronic authentication guideline*,[27] která byla nahrazena verzí NIST 800-63-3) doporučovala na základě Shannonovi analýzy anglického jazyka[28] následující *schéma pro hrubý odhad* entropie hesel vytvářených uživateli:

- za první znak se započítají 4 bity,
- za druhý až osmý znak se počítají 2 bity (za každý),
- za devátý až 20 znak v hesle se počítá 1,5 bitu,

[27] BURR, William E.; DODSON, Donna F.; POLK, William T. *Electronic authentication guideline*. US Department of Commerce, Technology Administration. Gaithersburg: National Institute of Standards and Technology, 2004.
[28] SHANNON, Claude E. Prediction and entropy of printed English. *Bell system technical journal*, 1951, 30.1: 50–64.

- za 21 a další znaky se počítá jeden bit,
- jako bonus se připočte 6 bitů entropie, pokud je zkontrolováno, že heslo obsahuje aspoň jedno malé písmeno, jedno velké písmeno a aspoň jeden nepísmenný znak,
- jako bonus se připočte 6 bitů, pokud má heslo méně než 20 znaků a provádí se kontrola hesla vůči slovníku o velikosti minimálně 50 tisíc slov. Před kontrolou se z hesla odeberou všechny nepísmenné znaky a řetězec se převede na malá písmena.

Obrázek 1.5: Výpočet entropie uživatelem vytvářených hesel dle NIST 800-63-2



Zdroj: vlastní zpracování (dle NIST 800-63-2)

1.4.4 Výpočty entropie uživatelských hesel dle NIST 800-63-2

Zadání

Spočítejte odhad entropii hesel vytvářených uživateli podle pravidel (z již neplatného) standardu NIST SP 800-63-2.

Řešení

Předpokládejme, že v daném informační systému jsou následující požadavky na heslo:

- Požadovaná minimální délka hesla voleného uživatelem je 9 znaků.
- Heslo musí obsahovat alespoň jedno malé písmeno.
- Heslo musí obsahovat alespoň jedno velké písmeno.
- Heslo musí obsahovat alespoň jednu číslici.

Aplikace výše uvedených (již neplatných pravidel NIST 800-63-2) je jednoduchá:

- 4 body za první písmeno,
- po dvou bodech za 2 až 8 písmeno, tj. 14 bodů,
- 1,5 bodu za 9 písmeno,
- prémie 6 bodů za požadavek na malé písmeno, velké písmeno a nepísmenný znak (číslici).

Tedy celkem 25,5 bodu entropie. Odpovídá náhodně vygenerovanému heslu s entropií 25,5, tj. jako by bylo $2^{25,5}$ kombinací náhodných hesel.

1.4.5 Kritika entropie dle NIST 800-63-2

Vypočet entropie dle NIST 800-63-2 byl značně *kritizován* ještě v době, kdy tato verze publikace platila. Např. heslo „*password*“ má dle těchto pravidel odhadovanou entropii 18 bitů, tj. pro odhalení by mělo být potřeba v průměru 2^{17} pokusů. V reálném světě je slovo

password jedním z prvních hesel, které útočník zkusí, neboť patří mezi nejpoužívanější hesla (ve většině analýz uniklých hesel je to dokonce na prvním až třetím místě).[29]

Podrobný rozbor je uveden např. v „Testing Metrics for Password Creation Policies by Attacking Large Sets of Revealed Passwords",[30] na závěr autoři doporučují kontrolovat uživateli zadaná hesla vůči slovníku častých hesel či uživatelem zadaná hesla doplňovat o náhodný prvek. Ve studii „Guess Again (and Again and Again): Measuring Password Strength by Simulating Password-Cracking Algorithms"[31] autoři ukazují odlišnou úspěšnost útoků u dvou pravidel se stejnou entropií dle NIST:

Tabulka 1.7: Popis pravidel vytváření hesel ze studie „Guess Again..."

Pravidlo	**Omezení pro vytváření hesel**	**Entropie dle NIST 800-63-2**
basic16	minimálně 16 znaků	30
comprehensive8	minimálně 8 znaků, aspoň jedno velké písmeno, aspoň jedna číslice či symbol, kontrola vůči slovníku)	30

Zdroj: vlastní zpracování, dle NIST 800-63-2 a studie „Guess Again…" (viz poznámka 31)

Pokud je málo pokusů (tisíc až milion, typicky při hádání hesel), tak útočník je úspěšnější v případě pravidla basic16. Pokud může útočník provést hodně pokusů (10^{12}, co je typické při lámání hašů), tak je výrazně úspěšnější v případě pravidla comprehensive8. Při zvětšujícím se počtu pokusů se rozdíl v úspěšnosti lámání zvyšuje. Předpokládá se, že útočník zná pravidla pro vytváření hesel.

Výsledkem studie není jen kritika entropie dle NIST 800-63-2, ale ještě jeden poznatek: pokud se omezí možnosti hádání, tak je bezpečnější požadovat delší hesla než mít složitá pravidla pro kratší hesla. Tato zjištění nakonec vedla k vydání dále popsané publikace *NIST 800-63-3*, která obsahuje *zcela přepracovaný soubor doporučení pro heslovou politiku*.

Výše uvedená pravidla lze též kritizovat na základě reálného průběhu lámání hesel vytvořených uživateli. V bezdrátové síti eduroam se hesla ukládají ve formátu NT-hash, viz kapitola 0. Jako vzorek pro pokus posloužilo tisíc hašů hesel uživatelů, kteří již více než rok nejsou studenty/zaměstnanci školy. Použit byl programu John-the-Ripper[32] a veřejně dostupný slovník hashkiller.dic.[33] Za první hodinu bylo odhaleno 369 hesel, z toho prvních 200 již za pouhé první tři minuty.

Z grafu (Obrázek 1.6) je zřejmé, že lámání *neprobíhá rovnoměrně* jako u náhodně generovaných hesel. Část hesel se obvykle nepodaří v rozumném čase odhalit. Obdobný průběh má většina lámání uživateli zadávaných hesel.

---

[29] DOČEKAL, Daniel: *TIP#175: Jaká jsou nejpoužívanější hesla a jak vůbec zacházet s hesly na Internetu?* 2015-06-24 [cit. 2020-10-25]. Dostupné z WWW: <https://365tipu.wordpress.com/2015/06/24/tip175-jaka-jsou-nejpouzivanejsi-hesla-a-jak-vubec-zachazet-s-hesly-na-internetu/>.

[30] WEIR, Matt, et al. Testing metrics for password creation policies by attacking large sets of revealed passwords. In: *Proceedings of the 17th ACM conference on Computer and communications security*. ACM, 2010. p. 162–175.

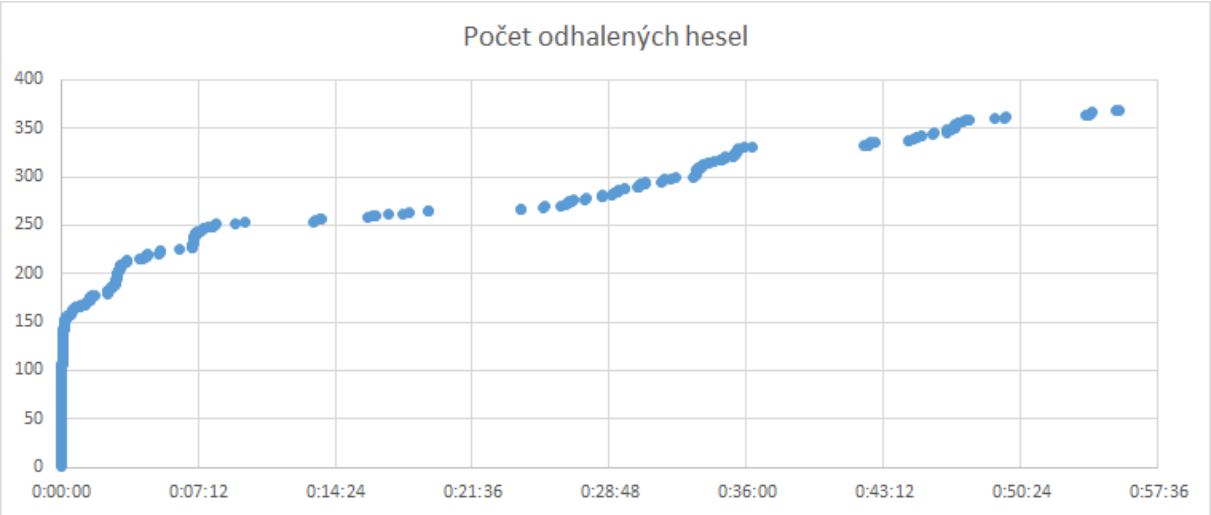
[31] KELLEY, Patrick Gage, et al. Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms. In: *Security and Privacy (SP), 2012 IEEE Symposium on*. IEEE, 2012. p. 523–537.

[32] John-the-Ripper <https://www.openwall.com/john/>.

[33] Passcape wordlist collection 9.2014 <https://weakpass.com/list/906>.

Naše závěry potvrzuje např. průběh lámání hašů hesel, která unikla ze seznamovací služby Ashley Madison.[34]

Obrázek 1.6: Časový průběh lámání hesel uživatelů eduroamu



Zdroj: vlastní zpracování

1.5 Ukládání hesel v systému

Hesla mohou být uložena v textovém souboru, v databázi, v registrech ap. Uložená hesla by měla být maximálně chráněna proti neoprávněnému zcizení pomocí přístupových práv. Praxe nás učí, že samotné nastavení přístupových práv *není dostatečnou ochranou*: chyby v aplikaci, malware, nepozornost správce při nastavování či kompromitace účtu správce vedou často ke zcizení hesel. Proto je vhodné při návrhu uložení hesel vycházet z předpokladu, že uložená hesla *budou dříve či později zcizena*.

V této kapitole postupně projdeme základní techniky používané při ukládání hesel: hašování hesel, solení hesel, pepření hesel a natahování hesel. Následovat budou popisy uložení hesel v UNIXu a ve Windows i doporučení pro jejich ukládání.

1.5.1 Hašování hesel

Hesla by se nikdy neměla ukládat v otevřeném tvaru – při případném odcizení souboru by je útočník mohl ihned zneužívat. Navíc otevřený tvar hesel svádí k obcházení pravidel – např. pokud se někdo potřebuje dostat k účtu spolupracovníka na dovolené, tak zavolá správci a požádá ho o heslo dotyčného.

Zašifrování hesla pomocí symetrické či asymetrické šifry obvykle není vhodným řešením. Pro ověření správnosti zadávaného hesla musí mít aplikace přístup ke klíči pro dešifrování hesel. Pokud se útočníkovi podaří překonat zabezpečení systému a získat soubor s hesly, tak je obvykle schopen získat i klíč potřebný pro jejich dešifrování.

[34] Stockley, Mark. *What Ashley Madison got right* [online]. 2015-08-31 [cit. 2020-08-08]. Dostupné z WWW: <https://nakedsecurity.sophos.com/2015/08/31/what-ashley-madison-got-right/>

Pro uložení hesel se používá nějaká **kryptografická hašovací funkce**, jež převede heslo do řetězce bitů pevné délky (např. 128, 160, 256 nebo 512 bitů a další). Ukázky pro několik jednoduchých hesel a dvě hašovací funkce (včetně změny výstupu při změně vstupu jednoho písmena) viz Tabulka 1.3.

Kryptografická hašovací funkce při změně i jediného bitu na vstupu vrátí úplně odlišný výsledek (při změně jednoho bitu na vstupu by se mělo změnit přibližně 50 procent bitů na výstupu). Na kryptografické hašovací funkce jsou kladeny následující požadavky:

1. *Odolnost vůči získání předlohy (pre-image resistance)* – pro daný haš $\boldsymbol{h}$ je prakticky nemožné spočítat $\boldsymbol{x}$ takové, že $hash(\boldsymbol{x}) = \boldsymbol{h}$. Tento požadavek definuje jednosměrnost hašovací funkce.
2. *Odolnost vůči získání jiné předlohy* ***(****second pre-image resistance****)*** – pro daný vstup $\boldsymbol{x}$ je prakticky nemožné spočítat/získat libovolné $\boldsymbol{y}$ takové, že $\boldsymbol{x} \neq \boldsymbol{y}$ a současně $hash(\boldsymbol{x}) = hash(\boldsymbol{y})$
3. *Odolnost vůči kolizím* ***(****collision resistence****)*** – je prakticky nemožné najít *dvojici libovolných* byť i nesmyslných vstupů $(\boldsymbol{x}, \boldsymbol{y})$, pro které platí $\boldsymbol{x} \neq \boldsymbol{y}$ a současně $hash(\boldsymbol{x}) = hash(\boldsymbol{y})$

Z hlediska uložení hesel není druhá a třetí vlastnost důležitá.[35] Např. hašovací funkce SHA-1 se považuje za zranitelnou z důvodu možného vyhledání kolizí (proto v roce 2017 hlavní webové prohlížeče přestaly akceptovat SSL certifikáty takto podepsané),[36] ale pro ukládání hesel lze tuto funkci stále používat. Kterou konkrétní hašovací funkci použít? K tomu se dostaneme v kapitole 1.5.12.

1.5.2 Solení hesel

Samotné hašování hesel má problémy:

- Pokud mají dva uživatelé *stejné heslo*, tak mají *i stejný haš*. Při získání hašů bude útočník lámat menší množství hesel. Pokud by uživatelé Alice a Bob měli stejné heslo a útočník to z hašů zjistil, tak při snaze o získání přístupu k účtu Alice může zkoušet techniky sociálního inženýrství na Boba.
- Pokud útočník získá více hašů, tak může spočtený haš *porovnat se všemi* získanými haši.
- Útočník si může předem připravit soubory s haši hesel – získání vlastního hesla z hašů je poté mnohem rychlejší a efektivnější. Předpřipravené tabulky (*rainbow tables*) byly popsány v kapitole 1.3.4.

Ochranou je tzv. **solení** – pro každý účet se vygeneruje nezávislý náhodný řetězec (*sůl*), který se připojí k heslu před hašováním. Sůl se neutajuje, obvykle je uložena vedle uživatelského jména a výsledného haše hesla, neboť je potřeba při každém ověření správnosti hesla zadaného uživatelem.

---

[35] Pokud by útočník znal otevřené heslo, tak pro něho nemá smysl hledat další heslo, ze kterého se spočítá stejný haš. Obdobně nemá smysl pro útočníka zkoušet hledat dvě jakákoliv hesla, která se zahašují stejně. Někdo může namítat, že útočník nepotřebuje najít přesně stejné heslo, ze kterého byl vytvořen haš. To je sice pravda, ale je potřeba si uvědomit, že entropie hesel je obvykle o mnoho řádů nižší než délka výsledného haše v bitech.

[36] JONES, J. C. *The end of SHA-1 on the Public Web*. Mozilla Security Blog [online]. 2017-02-23 [cit. 2020-10-25]. Dostupné z WWW: <https://blog.mozilla.org/security/2017/02/23/the-end-of-sha-1-on-the-public-web/>

Sůl se připojuje před heslo či za heslo – pořadí obvykle nemá vliv na bezpečnost.[37] Každý uživatel by měl mít svoji *jedinečnou sůl*. Občas někoho napadne použít jako sůl uživatelské jméno. V rámci jednoho systému je taková sůl jedinečná, ale ne mezi různými počítači. Téměř ve všech instalacích UNIXu má správce uživatelské jméno root. Útočník si připraví jednu tabulka se solí „root" pro lámání hesel všech správců UNIXu na světě. Obdobně je problematické použití id záznamu v tabulce s uživateli v databázi – např. ve většině instalací redakčního systému Wordpress je administrátor první založený uživatel, tj. jeho záznam má id rovné jedna.[38]

Sůl se nejčastěji vygeneruje pomocí generátoru náhodných čísel. Sůl by měla být tak dlouhá, aby bylo málo pravděpodobné, že se dvěma uživatelům vygeneruje stejná sůl, tedy vznikne kolize.[39] O kolizích a jak spočítat velikost soli pojednává následující kapitola.

Tabulka 1.8: Různé haše MD5 pro stejné heslo s různou solí

Hash("heslo")	= 1ff957bcca0b12c686d3b25bf46ea3b2
Hash("heslo" + "JcmlxWnH")	= 91e2ca509f6566984c4a91fc7ea2d6e5
Hash("heslo" + "GZXR0l6G")	= ce2f3a0dbe26f6104f349ac8a106dd57

Zdroj: vlastní zpracování

1.5.3 Kolize a výpočet délky soli

Kolize obecně znamená srážku či střetnutí. V kryptografii se pojem kolize nejčastěji používá pro popis situace, kdy hašovací funkce přiřazuje stejný výstup různým vstupním datům. Pojem kolize lze použít i pro situaci, kdy se pro dva uživatele vygeneruje stejná sůl.

S kolizí souvisí *narozeninový paradox*, který začíná zdánlivě jednoduchou otázkou – jak musí být velká skupina náhodně vybraných osob, aby v ní existovala dvojice osob s narozeninami ve stejný den s pravděpodobností minimálně 50 procent? Nejprve spočítáme, jaká je pravděpodobnost, že všech n narozenin je rozdílných. Pro $n > 365$ je tato pravděpodobnost rovna nule.

Pro $n \leq 365$ platí, že druhá osoba nemůže mít stejné narozeniny jako první, tj. má narozeniny v jednom z 364 dnů z 365 (pravděpodobnost je $364/365$). Třetí osoba nemůže mít narozeniny ve stejné dny jako první dvě osoby, tj. musí je mít v jednom ze zbývajících 363 dnů (pravděpodobnost $363/365$), atd. Výsledný vzorec poté vypadá takto:

$$\bar{p} = 1 * \left(1 - \frac{1}{365}\right) * \left(1 - \frac{2}{365}\right) * \ldots * \left(1 - \frac{n-1}{365}\right)$$

Skutečnost, že nejméně dva z n lidí mají narozeniny ve stejný den je komplementární jevu, že všechny data narozenin jsou různé. Proto pravděpodobnost p je

$$p = 1 - \bar{p}$$

[37] Zda je pořadí důležité závisí na konkrétně použité hašovací funkci – u obvykle používaných je to jedno. Občas se preferuje pořadí `heslo + sůl`, neboť na hašovací funkce MD5, SHA-1 či SHA-256 lze provést *length extension attack*.

[38] Sůl složená z uživatelského jména a nějakého jedinečného identifikátoru (např. jméno domény) se z hlediska bezpečnosti blíží náhodně generované soli. Sice lze stále namítat, že při změně hesla se používá stejná sůl, ale to snižuje bezpečnost hesel málo.

[39] V praxi se často používá jednoduché pravidlo – používejte sůl tak velkou, jako výsledek hašovací funkce. Tedy když použijete pro hašovací funkci SHA-1 s délkou výstupu 160bitů, tak byste měli používat sůl o délce minimálně 160 bitů. Při takto dlouhé soli je pravděpodobnost kolize zcela minimální. Cenou za tuto bezpečnost je větší množství času procesoru potřebného pro získání soli.

Tato pravděpodobnost překračuje 50 % pro $n = 23$ (pravděpodobnost kolize je $\approx 50{,}7\%$). Pro 57 a více lidí ve skupině je pravděpodobnost existence dvojice s narozeninami ve stejný den (pravděpodobnost kolize) více než 99 procent!

Pokud ze sady o velikost H vybereme n prvků, tak pro výpočet pravděpodobnosti kolize lze použít následující přibližný vzorec:

$$\mathrm{p} \approx 1 - \mathrm{e}^{-\frac{n^2}{2H}}$$

Tento vzorec použijeme ve dvou příkladech. V prvním budeme zjišťovat pravděpodobnost kolizí MAC adres při jejich generování pro virtuální servery. Ve druhém příkladu budeme počítat minimální velikost soli pro ukládání hesel.

Zadání 1

Při vytváření virtuálních serverů ve VMware se generuje ethernetová adresa (6 byte). První tři jsou pevně dány (00:50:56), další tři se náhodně generují. Jaká je pravděpodobnost kolize MAC adres v subnetu s maskou /22 (1022 počítačů)?

Řešení

Do vzorce pro výpočet pravděpodobnosti kolize za H doplníme 2^{24} (poslední tři byty MAC adresy mají 24 bitů), za n doplníme 1022. Pravděpodobnost kolize je 3,06 %. Tato pravděpodobnost je poměrně vysoká – VMware detekuje vznik případné kolize a pokud by k ní došlo, tak vygeneruje novou MAC adresu.

Zadání 2

Budete ukládat 20 000 hesel, pro každé uložené heslo vygenerujete náhodnou sůl. Kolik bitů má mít sůl, aby pravděpodobnost stejné soli (tj. kolize) byla menší než 0,1 %?

Řešení

Potřebné množství n vybraných prvků z množiny H k dosažení pravděpodobnosti kolize p spočítáme dle následujícího přibližného vzorce, který vznikl úpravou původního vzorce na výpočet pravděpodobnosti kolize:

$$\mathrm{n} \approx \sqrt{2\mathrm{H} * \ln\frac{1}{1-\mathrm{p}}}$$

Pro sůl kratší než 15 bitů (2^{15}) je *kolize jistá*, protože počet možných kombinací je menší než počet uložených hesel (obdobně jako když se sejde 366 lidí, tak *alespoň* dva *musí* mít narozeniny ve stejný den: opět neuvažujeme přestupný rok). Aplikací narozeninového paradoxu a uvedeného vzorečku snadno zjistíme, že pravděpodobnost 100 nebo *skoro 100 procent* platí i pro mnohem delší sůl.

V našem případně bychom museli použít sůl o délce minimálně 38 bitů (Tabulka 1.9), aby pravděpodobnost kolize byla menší než 0,1 %. Pokud uvažujeme účet pro každého člověka na zemi (téměř 8 mld.), tak potřebujeme generovat náhodnou sůl o délce minimálně 75 bitů. Bezpečná je délka náhodné soli s 80 či 128 bity.

Následující tabulka s pravděpodobnostmi kolize pro 20 000 ukládaných hesel vznikla v Excelu s tímto zápisem vzorce:

=1-EXP(-(20000^2)/(2*počet_kombinací))

Tabulka 1.9: Počet kombinací náhodně generované soli a pravděpodobnost kolize pro 20 000 ukládaných hesel

bitů soli	kombinací soli	pravděpodobnost kolize [%]
15	32768	100,00000
…		
23	8388608	100,00000
24	16777216	99,99934
25	33554432	99,74213
26	67108864	94,92190
27	134217728	77,46535
28	268435456	52,52933
29	536870912	31,10103
30	1073741824	16,99460
31	2147483648	8,89270
32	4294967296	4,54986
33	8589934592	2,30141
34	17179869184	1,15740
35	34359738368	0,58039
36	68719476736	0,29062
37	1,37439E+11	0,14541
38	2,74878E+11	0,07273
39	5,49756E+11	0,03637
40	1,09951E+12	0,01819
41	2,19902E+12	0,00909
42	4,39805E+12	0,00455
43	8,79609E+12	0,00227
44	1,75922E+13	0,00114
45	3,51844E+13	0,00057
46	7,03687E+13	0,00028
47	1,40737E+14	0,00014

Zdroj: vlastní zpracování

1.5.4 Pepření hesel, šifrování hašů hesel

Při lámání zcizených hašů hesel útočník většinou použije *slovníkový útok s pravidly*. Velmi brzy zjistí jednoduchá hesla vytvořená ze slov. Obranou proti tomu by mohlo být tzv. **pepření hesel**, kdy se k heslu a soli přidá ještě pepř:

```
$hashed_password = hash( $pepper . $salt . $password )
```

Pepř je náhodný řetězec, který je společný pro všechny uživatele a který *je utajován*, tj. není ukládán společně s haši hesel.[40] Délka by měla být minimálně 128 bitů. Nelze vynechat solení, bez solení by dva uživatelé se stejným heslem měli stejnou uloženou hodnotu.

[40] Pepření používá stejný princip jako HMAC (Hash-based Message Authentication Code, kód pro ověření zprávy s využitím hašovací funkce). Občas se proto doporučuje následující varianta pepření:
`$hashed_password = hmac ( hash ( $salt . $password ), $pepper )`

Pokud se podaří pepř utajit, tak pepření efektivně zabrání slovníkovému útoku. Pepření se *ale nedoporučuje*, neboť pokud útočník ovládne systém tak, že je schopen exportovat databázi s haši hesel, tak mu většinou nečiní problém získat z aplikace i hodnotu pepře. Pepření též není součástí standardních knihoven pro ukládání a verifikaci hesel. Lepší alternativou k pepření je šifrování výsledného haše symetrickou šifrou pomocí společného klíče pro všechny uživatele:

```
$encrypted_hash = encrypt ( hash ( $salt . $password ), $encrypt_key )
```

Bezpečnostní přínos je stejný jako u pepření, a stejně jako u pepření je Achillovou patou utajení šifrovacího klíče před útočníkem. Šifrování má ale proti pepření jednu výhodu. Je-li podezření na průnik do systému, můžete klíč pro symetrickou šifru změnit, tedy uložené haše dešifrovat původním klíčem a znovu zašifrovat pomocí nového klíče. V případě pepření musíte vyčkat, až si uživatel změní heslo sám.

Pro šifrování hesel si můžete zakoupit *hardwarový šifrovací modul (HSM, Hardware Security Module)*, který bude šifrovat hesla a ze kterého nelze používané heslo získat.[41] Existují i HSM moduly s podporou pepření.

Použití HSM je doporučené u systémů se 100 000 účty a v podstatně nutné u systémů s miliony účtů. Šifrování či pepření s kvalitními HSM se považuje za velmi bezpečný způsob uložení hesel.

1.5.5 Rychlost lámání, „natahování“ hesel

Kryptografické hašovací funkce se vybírají i s ohledem na rychlost provádění. V následující tabulce je rychlost hašování pěti často používaných hašovacích funkcí pro různé délky vstupního řetězce:

Tabulka 1.10: Počet zahašovaných bloků různé délky za jednu vteřinu

Hašovací funkce	16 bytů	64 bytů	256 bytů	1024 bytů	8192 bytů
MD4	2 979 315	2 375 360	1 310 619	501 649	73 707
MD5	2 270 182	1 685 217	923 290	335 447	48 219
SHA-1	2 665 515	1 846 157	1 003 605	345 949	49 930
SHA-256	1 672 252	891 874	369 282	111 768	14 850
SHA-512	1 225 479	1 209 175	454 850	158 329	22 662

Pro různé hašovací funkce, na jednom jádru procesoru
Intel(R) Xeon(R) CPU E5-2420 0 @ 1.90 GHz pomocí programu openssl
Zdroj: vlastní zpracování (MD4 a MD5 jsou zcela zastaralé a neměly by se vůbec používat, SHA-1 se přestala používat pro SSL certifikáty, viz poznámka 36)

Heslo včetně soli se většinou vejde do 64 B, takže na uvedeném počítači lze za vteřinu ověřit téměř 2 mil hesel uložených pomocí SHA-1. Specializované programy pro lámání hesel dosahují větších rychlostí, viz následující tabulka pro známý program *John the Ripper*.
U rychlých hašovacích funkcí spojování hesla se solí výrazně zpomalí rychlost lámání.

[41] Také u HSM modulů potřebujete mít možnost náhrady v případě poruchy či provozovat druhý záložní HSM modul. Musí tedy existovat nějaká možnost získání klíčů pro symetrickou šifru, ale je výrazně omezena (např. zálohu lze vytvořit jen při inicializaci modulu a pouze přes speciální rozhraní).

Tabulka 1.11: Otestované kombinace hesel v tisících za jednu vteřinu

Hašovací funkce	$heslo	$heslo . $sul
MD4	28 143K	13 215K
MD5	19 324K	9 703K
SHA-1	9 698K	6 541K

John the Ripper: jedno jádro procesoru Intel(R) Xeon(R) CPU E5-2420 0 @ 1.90GHz
Zdroj: vlastní zpracování

Natahování hesel (password stretching)[42] se snaží *zpomalit* rychlost lámání. Princip je jednoduchý – mnohokrát zopakovat operaci hašování:[43]

```
$hashed_password = hash ( $password . $salt )
for $i = 1; $i < $repeats; $i++ {
    $hashed_password = hash ( $hashed_password )
}
```

Pokud se operace hašování zopakuje 10 000krát, tak se odpovídajícím způsobem prodlouží i doba potřebná pro lámání hesel. Počet opakování se ale nesmí přehnat, jinak by se mohlo přihlásit k systému jen pár uživatelů za minutu.

Opakování jsou zabudována buď přímo v konkrétním algoritmu pro ukládání hesel (např. již zastaralý a nepoužívaný *DEScrypt* nebo současný *bcrypt*) nebo se používá obecná funkce PBKDF2, která bude popsána v následující kapitole.

1.5.6 Generování klíčů z hesel (Key Derivation Function), PBKDF2

Funkce pro odvození klíče (**KDF**, **Key Derivation Function**) z nějakého tajemství (heslo, heslová fráze či hlavní klíč) vygeneruje jeden i více klíčů pro symetrickou šifru. KDF obvykle používá hašovací funkce.

Pokud se klíč odvozuje z hesla či z heslové fráze, tak útočník může zkoušet útoky hrubou silou, může mít předpřipravené tabulky či používat slovníkový útok. Na obranu se používá sůl a velký počet iterací.

Nejznámějším příkladem KDF pro odvezení klíče z hesla je funkce **PBKDF2** (*Password-Based Key Derivation Function version 2.0*)**,** která je popsána v RFC 2898.[44] Funkce má 5 vstupních parametrů:

- PRF – pseudonáhodná funkce (Pseudorandom Function), která má dva vstupní parametry a výstup pevné délky. Existuje více vhodných pseudonáhodných funkcí, US standard NIST SP800-132 povoluje použít funkci HMAC[45] se schválenými kryptografickými hašovacími funkcemi.

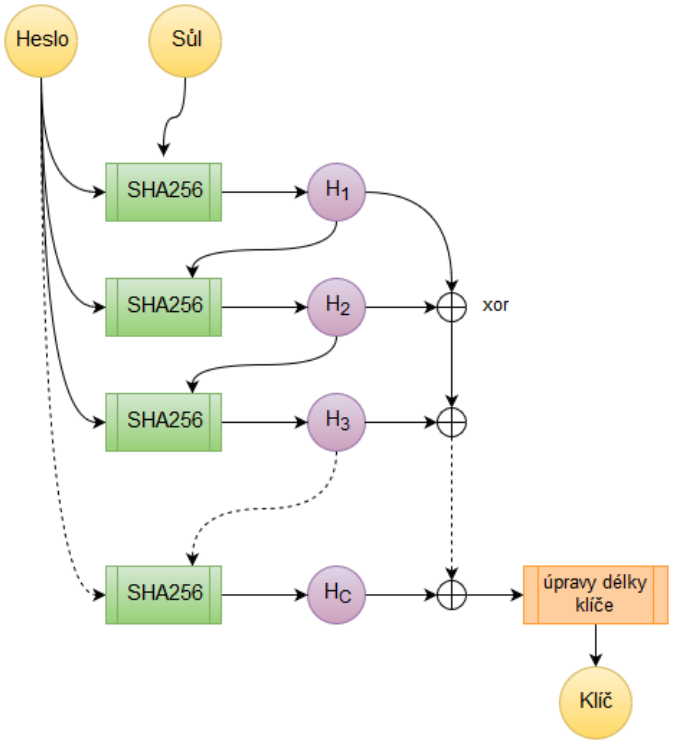
---

[42] Existuje též **password strengthening** (**posilování hesel**), kdy se (vedle dříve popsané soli o větší délce) při uložení použije ještě poměrně *krátká náhodná sůl* (např. o velikosti 2^16), která se ihned zapomene (nikam se neukládá). Při ověřování hesla se musí vyzkoušet možné kombinace tajné soli, aby se zjistilo, zda uživatel zadal správné heslo. To samé musí udělat i útočník při lámání.
MANBER, Udi. *A simple scheme to make passwords based on one-way functions much harder to crack*. Computers & Security, 1996, 15.2: 171–176.
[43] Viz KELSEY, John, et al. Secure applications of low-entropy keys. In: *Information Security*. Berlin, Heidelberg: Springer, 1997. p. 121–134.
[44] KALISKI, Burt. RFC 2898; PKCS# 5: Password-Based Cryptography Specification Version 2.0. 2000.
[45] HMAC (keyed-Hash Message Authentication Code) použije hašovací funkci dvakrát, zjednodušený vzorec pro první krok `hash(`*`heslo`*`+hash(`*`heslo`*`+`*`sůl`*`))`.

- Heslo – vstupní heslo či heslová fráze.
- Sůl – sůl.
- C – počet iterací.
- Délka_klíče – požadovaná délka výsledného klíče.

```
Klíč = PBKDF2(PRF, Heslo, Sůl, C, Délka_klíče)
```

Obrázek 1.7: Průběh PBKDF2 s hašovací funkcí HMAC-SHA256



Zdroj: vlastní zpracování

1.5.7 Příklady použití KDF

Bezdrátové sítě v domácnosti jsou obvykle chráněny pomocí sdíleného hesla. Vlastní komunikace s přístupovým bodem (Access Point) je zašifrovaná pomocí RC4 (starší protokol *WPA-PSK*) či AES-128 (novější *WPA2-PSK*). Klíč pro tyto symetrické šifry se vygeneruje ze sdíleného hesla pomocí PBKDF2 s využitím hašovací funkce SHA-1. Jako sůl se použije SSID a počet iterací je nastaven na 4096.

Správce hesel *1Password* šifruje soubor s uloženými hesly pomocí symetrické šifry AES-128. Z uživatelovy heslové fráze a z náhodně vygenerované soli je pomocí PBKDF2 vygenerován hlavní klíč, používá se hašovací funkce SHA-1, původně s 1000 iterací. Od roku 2010 je počet iterací proměnlivý dle výkonu zařízení,[46] minimum je 10 000 iterací. Pomocí hlavního klíče je zašifrován pomocný klíč – náhodný řetězec o délce 1024 B. Teprve z tohoto pomocného klíče je vytvořen klíč pro AES-128. Tato hierarchie klíčů má několik výhod:

[46] *OPVault Design* [online]. [cit. 2020-08-08]. Dostupné z WWW: <https://support.1password.com/opvault-design/>

- při změně heslové fráze není nutné dešifrovat a zašifrovat hlavní soubor s hesly,
- na různých zařízeních může být použita odlišná heslová fráze,

Uživatele *MS Office* si mohou vytvořené soubory chránit heslem.[47] Od verze 2007 se dokument zašifruje symetrickou šifrou AES-128. Klíč se generuje ze zadaného hesla pomocí upravené verze PBKDF2 – v každé iteraci je vstupem do hašovací funkce i pořadové číslo iterace. Sůl má velikost 16 B, v Office 2007 se používalo 50 000 iterací, od verze 2010 je to 100 000 iterací. V Office 2013 byla funkce SHA-1 nahrazena funkcí SHA-512.

Od verze Office 2010 lze používat i tzv. *Agile formát* – podobně jako u *1Password* se pro zašifrování souboru použije náhodně vygenerovaný klíč. Ten lze dešifrovat buď klíčem odvozeným z hesla, nebo heslem/soukromým klíčem administrátora – cílem je zajistit přístup k souboru v situaci, kdy uživatel zapomene heslo či odejde z firmy.

KeePass – správce hesel pro Windows – šifruje soubor s hesly pomocí symetrické šifry AES s klíčem o délce 256.[48] Používá se CBC mód šifrování,[49] při každém uložení se generuje nový inicializační vektor o délce 128 bitů (velikost bloku).

KeePass používá vlastní funkci pro odvození klíče z heslové fráze. Nejdříve pomocí SHA-256 zahašuje heslovou frázi a náhodnou sůl. Pak vygeneruje náhodný pomocný klíč 256 bitů, který uloží do databáze (při každém zápisu se generuje nový). S využitím tohoto pomocného klíče N-krát zašifruje výstup hašovací funkce. Výsledek šifrování opět zahašuje pomocí SHA-256. Výchozí počet opakování šifrování je 6000. Při inicializaci souboru s hesly či při změně heslové fráze lze zadat jinou hodnotu nebo si nechat spočítat počet opakování pro vteřinovou prodlevu – na mém počítači je to přes 5 milionů opakování.

V *OpenSSH* (staré verze) se soukromý klíč[50] ukládá do textového souboru:

```
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: AES-128-CBC,inicializační vektor

Mxx78SYtNCecbqIYi.... klíč převedený do base64
```

Soukromý klíč je v souboru zašifrován pomocí AES-128 v módu CBC s uvedeným inicializačním vektorem. Tato symetrická šifra se považuje za bezpečnou. Problémem ale je, že klíč pro symetrickou šifru se vytváří pomocí jednoho průchodu rychlé hašovací funkce MD5.[51] Útočník může za vteřinu vyzkoušet miliony možných hesel.

Od verze 6.5 programu OpenSSH je proto možné (*a doporučené*) používat *nový formát* souboru, ve kterém se klíč pro symetrickou šifru AES-256 generuje pomocí PBKDF2 a pomalé hašovací funkce *bcrypt* se 64 iteracemi, PBKDF2 má ve výchozím stavu 16 iterací,

---

[47] *Office Document Cryptography Structure* [online]. [cit. 2020-06-07]. Dostupné z WWW: <https://msdn.microsoft.com/en-us/library/cc313105.aspx>

[48] *KeePass Security* [online]. [cit. 2020-08-08]. Dostupné z WWW: <https://keepass.info/help/base/security.html>.

[49] Módy symetrických blokových šifer jsou probírány na přednáškách, základní popis najdete též na Wikipedii <https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation>.

[50] Soubor obsahuje i veřejný klíč. V případě RSA soubor obsahuje trojici hodnot ***n*** (modul), ***e*** (veřejný exponent) a ***d*** (soukromý exponent). Veřejným klíčem je dvojice (***n, e***), soukromým klíčem je dvojice (***n, d***). Soubor obsahuje ještě další odvozené proměnné pro rychlejší kontrolu podpisu podle principu čínského teorému.

[51] TOPONCE, Aaron. *Super Size The Strength Of Your OpenSSH Private Keys* [online]. 2014-12-08 [cit. 2020-08-08]. Dostupné z WWW: <https://pthree.org/2014/12/08/super-size-the-strength-of-your-openssh-private-keys/>.

lze zvýšit až na 1000. Při maximální hodnotě trvá vygenerování klíče z hesla od několika vteřin až po přibližně půl minuty, podle rychlosti vašeho notebooku.

PuTTY (SSH aplikace pro Windows) ukládá soukromý a veřejný klíč společně do souboru s příponou *.ppk*. Uživatelem zadané heslo se hašuje dvakrát pomocí SHA-1, pokaždé s jinou solí. Kombinací výsledků se vytvoří 256 bitů dlouhý klíč pro symetrickou šifru AES, pomocí které se zašifruje obsah souboru (soukromý a veřejný klíč uživatele). Útočník s grafickou kartou je schopen za vteřinu testovat stovky tisíc až miliony kombinací hesla. Problémem není ani tak samotné použití SHA-1 (i když použiti např. pomalé funkce *bcrypt* jako u nového formátu OpenSSH by bylo vhodné, navíc jde o obdobný účel), ale především to, že se nevyužívají iterace (nejlépe s volitelným počtem iterací).

1.5.8 Rychlost lámání CPU x GPU x ASIC

Při lámání hesel je pro útočníka důležité, kolik potencionálních hesel ověří za jednu vteřinu. Může si na lámání pořídit rychlejší procesor, počítač s více procesory či zapojit do lámání více počítačů. Dále může využít výpočetní kapacity grafické karty (**GPU**) – u některých šifer a některých grafických karet se zvýší rychlost lámání i více než 50x vůči hlavnímu procesoru.

Tabulka 1.12: Srovnání rychlosti lámání některých typů hašů hesel pomocí CPU (všechna jádra) a pomocí grafické karty

algoritmus	**Intel i7 2600 (4 jádra), program John the Ripper**	**graf. karta AMD Radeon HD 7970, program oclhashcat**	**kolikrát je GPU rychlejší**
DEScrypt	18 284K/s	65 594K/s	3,6x
MD5 crypt	66,9K/s	3 592K/s	53,7x
bcrypt	4,8K/s	4,1K/s	0,85x
LM hash	88 834K/s	2 384M/s	26,9x

Zdroj: vlastní zpracování

Pro lámání hesel lze používat i *specializované hardwarové obvody* (**ASIC**, **FPGA**). O jejich možnostech byly dlouhou dobu jenom drobné mlhavé zmínky, neboť jejich majitelé tyto údaje pochopitelně utajovali. Situace se změnila s nástupem *kryptoměn*, u kterých je těžba obvykle závislá na rychlosti hašovacích funkcí. Při těžbě Bitcoinů se používá hašovací funkce SHA-256, u Litecoinů se používá funkce *scrypt*.

Následující tabulka srovnává CPU, GPU a specializované hardwarové obvody. Uvedená konkrétní zařízení typu ASIC nebo FPGA se již nevyrábí, ale pro základní *srovnání rozdílů* postačuje. V současnosti bychom za obdobnou cenu a při obdobné spotřebě získaly pravděpodobně ještě několikrát výkonnější zařízení (ale vzrostly i výkony CPU a GPU).

Poznámka: **ASIC** (zkratka z *Application Specific Integrated Circuit*), je *zákaznický integrovaný obvod*, navržený a vyráběný pro určité konkrétní využití. **FPGA** (zkratka z *Field Programmable Gate Array*, česky *programovatelná hradlová pole*, je typ logického integrovaného obvodu, který je vyroben tak, aby mohl být naprogramován až u zákazníka. Tím se odlišuje od ASIC, jejichž funkce je dána již při výrobě.

Tabulka 1.13: Rychlost operací SHA-256 na různém hardware pro těžbu Bitcoinů

zařízení	typ	cena	spotřeba	SHA-256 v MHash/s
Core i7-2600K	CPU	317 USD	95 W	7
AMD Radeon 7970	GPU	550 USD	214 W	825
AntMiner S5+	ASIC[52]	2 300 USD	3 436 W	7 722 000
BitForce SHA256 Single	FPGA[53]	600 USD	80 W	832

Zdroj: vlastní zpracování, dle *Mining hardware comparison* [online]. [cit. 2020-08-08] Dostupné z WWW: <https://en.bitcoin.it/wiki/Mining_hardware_comparison>

Lze očekávat, že podobných rychlostí by bylo možné dosáhnout i u ASIC modulů pro lámání hesel. Rozdíl v rychlosti hašovacích operací mezi CPU a ASIC moduly je *tak velký*, že *ohrožuje použitelnost systémů založených na heslech* – pokud na serveru nastavíme počet opakování v PBKDF2 tak, že bude možné přihlášení pouze jednoho uživatele za vteřinu, tak útočník s ASIC modulem může za stejnou dobu ověřit milión možných kombinací hesla.

Řešení tohoto problému naznačuje předchozí tabulka (Tabulka 1.12) konkrétně řádek s algoritmem *bcrypt*, který je na grafické kartě *pomalejší* než na CPU. Důvodem jsou nároky algoritmu *bcrypt* na paměť – při vytváření klíče se zaplní 4 KiB paměti, která se poté v „náhodném“ pořadí čte a modifikuje.

Grafické karty mají stovky až tisíce specializovaných procesorů (*shader* či *stream procesor*) rozdělených do skupin – např. AMD Radeon 7970 má 128 skupin a celkem 2048 stream procesorů. Každý stream procesor má registry, každá skupina má menší množství sdílené paměti. Ta ale pro tabulku algoritmu *bcrypt* nepostačuje, a tak musí každý stream procesor přistupovat k hlavní paměti grafické karty. K ní ale může souběžně přistupovat jen malý počet procesorů, a tak při lámání *bcrypt* hesel stream procesory většinu času čekají na uvolnění sběrnice pro přístup k paměti. Pro algoritmy SHA-1 a SHA-256 postačují registry na stream procesorech, a tak je lze plně paralelizovat.

Další algoritmy pro ukládání hesel odolné vůči lámání pomocí GPU či zákaznických procesorů si popíšeme v kapitole 1.5.12, kde budou i doporučení pro ukládání hesel v systému. Předtím si ukážeme způsoby ukládání hesel v UNIXu a ve Windows.

1.5.9 Ukládání hesel v UNIXu

Ač se to zdá z dnešního pohledu neuvěřitelné, tak v počátcích UNIXu se hesla ukládala *nešifrovaně* do souboru /etc/passwd:

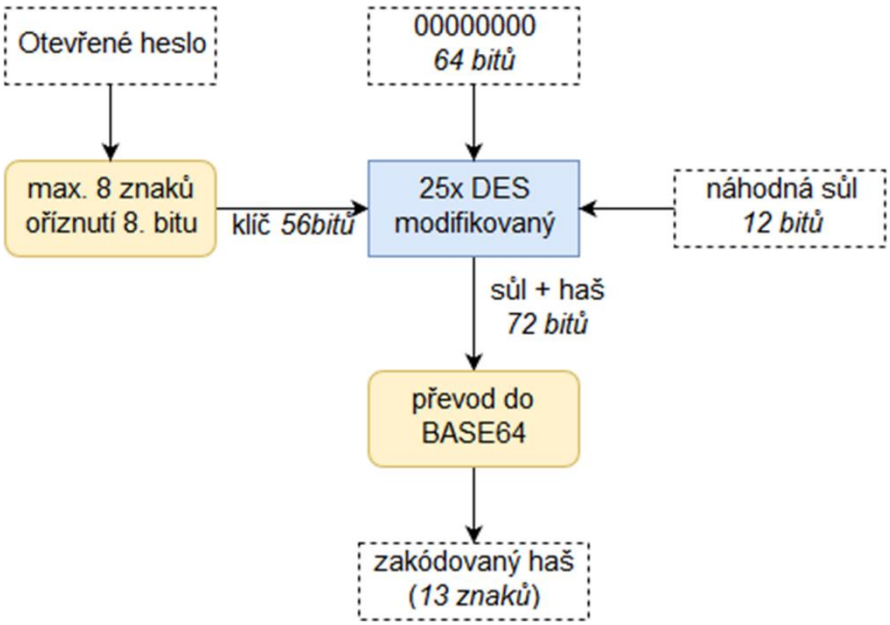
```
root:superpsecret:0:1:Super-User:/root:/bin/sh
raekwon:icecream:6:1:User:/home/raekwon:/bin/sh
inspectadeck:who?:8:1:User:/var/inspect:/bin/sh
```

[52] Jak vyplývá z definice zařízení typu ASIC, tato zařízení určená pro těžbu Bitcoinů *nelze* předělat na lámání hesel. Ale pokud je někdo schopen navrhnout a vyrobit ASIC pro těžbu Bitcoinů, tak zcela jistě bude schopen navrhnout i ASIC moduly pro lámání hesel.

[53] Programovatelná hradlová pole (FPGA) lze naprogramovat pomocí firmware. FPGA pro těžbu Bitcoinů by se po úpravě firmware možná *mohly* použít pro lámání hesel.

V UNIXu V6 z roku 1975 se hesla začala hašovat pomocí upravené symetrické šifry DES[54] s uplatněním kryptografické soli a většího počtu iterací.[55] Algoritmus se obvykle označuje zkratkou ***DEScrypt***, někdy jenom ***crypt*** dle názvu funkce v unixových knihovnách. V tomto algoritmu se nejdříve heslo v otevřeném textu ořízne na 8 znaků, dále je z každého znaku odstraněn nejvyšší bit. Tím vznikne 56bitový klíč vhodný pro šifru DES. Vygeneruje se náhodná 12bitová sůl, která modifikuje algoritmus DES (záměrem bylo, aby se pro lámání hesel nemohly použít tehdejší hardwarové šifrátory pro algoritmus DES). Pomocí klíče se 25x za sebou zašifruje 64bitový blok na začátku plný nul. Výsledek se včetně soli převede do prostého textu pomocí upraveného Base64. Hašování a kódování hesla pomocí DEScrypt.

Obrázek 1.8: Hašování a kódování hesla pomocí DEScrypt



Zdroj: vlastní zpracování

V *době svého návrhu* byl *DEScrypt* odolný vůči známým útokům, neboť byly velmi nákladné. Na většině počítačů z té doby trvalo ověření haše jednoho hesla přibližně vteřinu (bylo záměrně pomalé). Nejdříve se haše ukládaly místo původního hesla do souboru `/etc/passwd`, který je čitelný pro všechny přihlášené uživatele v systému. To se brzy začalo pokládat za nebezpečné a haše hesel putovaly ze souboru `/etc/passwd` do souboru `/etc/shadow`, který může číst pouze privilegovaný uživatel root.

Tabulka 1.14: DEScrypt: příklady hašování hesel. Sůl jsou první dva uložené znaky

Otevřené heslo	**Sůl (base64)**	**Uložené znaky**
heslo	Qi	QiRdV511osUbQ
heslo	ZC	ZCFk57ntsGFz.
tajne	ZC	ZCDtGsWTWphdU

Zdroj: vlastní zpracování

[54] Standard DES (Data Encryption Standard) byl publikován na začátku roku 1975.

[55] MORRIS, Robert; THOMPSON, Ken. Password security: A case history. *Communications of the ACM*, 1979, 22.11: 594–597.

S postupem času se rychlost počítačů zvyšovala a s tím se zvyšoval počet hesel, která bylo možné otestovat za vteřinu. Výrazně se zvyšovala kapacita disků, na které bylo možné ukládat předpřipravené tabulky pro *DEScrypt*. Tím se začaly projevovat principiální nedostatky algoritmu *DEScrypt* – ořezávání hesla na 56 bitů (které ale bylo dáno podstatou využitého algoritmu DES), krátká sůl a malý počet opakování.

Od roku 1994 se ve FreeBSD začal pro ukládání hesel používat algoritmus ***MD5crypt,***[56] který se rychle rozšířil do ostatních verzí UNIXu. MD5crypt podporuje hesla libovolné délky (maximum je 2^64 bitů), sůl může být v délce 12 až 48 bitů, pro ztížení lámání se operace MD5 opakuje 1000krát. Výsledek se převede na tisknutelný řetězec pomocí upraveného algoritmu base64.

```
digest = md5(password + salt)
rounds = 1000
while rounds > 0 {
  digest = md5(password + salt + digest)  // pořadí spojení řetězců
                                          // závisí na kroku cyklu
  rounds--
}
```

Nyní se algoritmus MD5crypt rovněž nepovažuje za vhodný pro ukládání hesel, neboť:

- malý počet opakování, na současných CPU a GPU lze lámat hesla velkou rychlostí,
- systémy s MD5crypt by mohly mít problémy s certifikací, neboť MD5 je v různých národních standardech zakázaný, byť z důvodu zranitelnosti při vyhledávání kolizí.

V roce 1999 bylo navrženo další schéma pro ukládání hesel v UNIXu – algoritmus ***bcrypt***,[57] který používá upravenou symetrickou šifru *Blowfish*: viz poznámka 69. Algoritmus *bcrypt* má tři vstupní parametry – nezašifrované heslo, sůl (128 bitů) a cena. Pomocí ceny se ovlivňuje rychlost hašování hesel a tím i rychlost lámání. Počet opakování se rovná 2^cena. V roce 1999 autoři doporučovali cenu 5 (tj. 32 opakování), nyní se doporučuje nastavit cenu na 12 (tj. 4096 opakování) nebo více.

```
bcrypt(cost, salt, password)
    state = EksBlowfishSetup(cost, salt, password) // inicializace klíče
    ctext = "OrpheanBeholderScryDoubt"             // tři 64-bit bloky
    repeat (64)
        ctext = EncryptECB(state, ctext)           //zašifrování pomocí
                                                   // BlowFish v ECB módu
    return Concatenate(cost, salt, ctext)
```

V roce 2007 byly navrženy algoritmy ***sha256crypt*** a ***sha512crypt***,[58] které používají standardizované hašovací funkce SHA-256 či SHA-512. Sůl má velikost 128 bitů, lze zadat počet opakování hašovací funkce (výchozí hodnota je 5000 opakování, minimum 1000, maximum 999 999 999). Průběh hašování hesla pro sha512crypt lze zjednodušeně popsat takto:[59]

---

56 NORRIS, Jeffrey S. Mission-Critical Development with Open Source Software: Lessons Learned. *IEEE Software*, 2004, 21.1: 42–49.

57 PROVOS, Niels; MAZIERES, David. A Future-Adaptable Password Scheme. In: *USENIX Annual Technical Conference, FREENIX Track*. 1999. p. 81–91.

58 DREPPER, Ulrich. *Unix crypt with SHA-256/512* [online]. 2007-09-19 [cit. 2020-08-08] Dostupné z WWW: <https://www.akkadia.org/drepper/sha-crypt.html>.

59 *Implementation of SHA512-crypt vs MD5-crypt* [online]. 2011-08-16 [cit. 2020-08-08] Dostupné z WWW: <https://www.vidarholen.net/contents/blog/?p=33>.

1. Spojí se heslo a sůl a pro výsledek se spočítá haš SHA-512.
2. V zadaném počtu opakování se počítá haš SHA-512 z předchozího haše střídavě spojeného s hašem hesla či s hašem soli.
3. Pomocí upraveného base64 se poslední haš převede na výsledný řetězec.

Tabulka 1.15: Příklady hašů hesel v UNIXu pro různé algoritmy

algoritmus	id	příklad
DEScrypt		`U.84ABidkUBEc`
MD5crypt	1	`$1$7kl1RSbt$LNjKn7YYO9nGVy6puBmBl/`
bcrypt	2a, 2y	`$2a$07$yjgMwqKcQ3HHPpdDE.iXB.FiPtQNasRsn7C5X5tdglI53A.WMjZhm`
sha256crypt	5	`$5$x4RU4h3stLpdbDbr$pMvwcN43hJ2JbWfmAIeYngGDsamwFAzG1wXWRqz8Vt1`
sha512crypt	6	`$6$CmUUXKBQbamzyGo1$SYO0.vfCL4Z2RuPeuE6GhXF8qs2Z/uDd0H.vUWGcDLFR4qgO5BlZY4PXQPbMzJ1dsMg4Cu2JUZyU9yAjrAzP00`

Zdroj: vlastní zpracování

1.5.10 Ukládání hesel ve Windows

Algoritmus **LM-hash** (též *LanMan hash*) vznikl v polovině 80. let při návrhu počítačové sítě LANMAN pro počítače s operačním systémem MS DOS. Používal se i pro ukládání hesel ve Windows, od Windows NT je nahrazen algoritmem *NT-hash*. Z důvodu zpětné kompatibility podporují *LM-hash* ještě Windows 7. Algoritmus má následující kroky.[60]

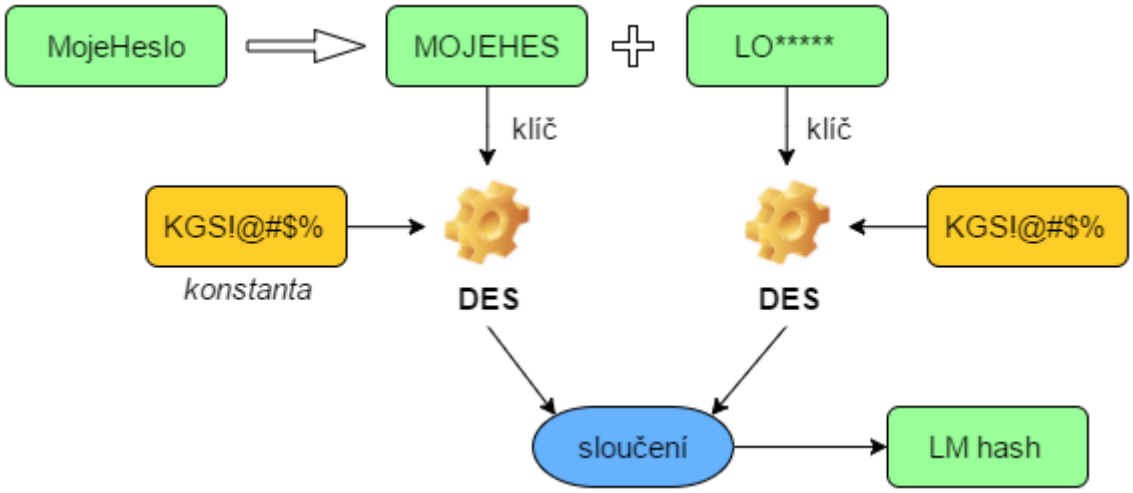
- Heslo se doplní znakem *ASCII nul* do délky 14, delší se naopak na 14 znaků ořízne.
- Písmena se převedou na velká písmena dle systémové znakové sady.
- Rozdělí se na dvě části po 7 bytech, tj. vzniknou dva klíče o délce 56 bitů pro šifru DES.
- Pomocí obou klíčů se zašifruje konstanta KGS!@#$% a výsledek se spojí do výsledného haše.

Z dnešního pohledu má *LM-hash* mnohé *zásadní slabiny*:

- Konverze na velká písmena omezuje znakovou sadu.
- Rozdělení na dvě poloviny znamená, že každou lze lámat nezávisle. Je to jako lámání dvou hesel o maximální délce 7 znaků.
- Použití systémové znakové sady vede k nekompatibilitám (stanice mohla být v české znakové sadě a server v západoevropské) a prakticky omezuje využití osmého bitu v hesle.
- Chybí sůl či větší (variabilní) počet opakování.

[60] *The NTLM Authentication Protocol and Security Support Provider* [online]. [cit. 2020-08-08] Dostupné z WWW: <http://davenport.sourceforge.net/ntlm.html#theLmResponse>
JOHANSSON, Jesper M. *Frequently Asked Questions About Passwords* [online]. 2008-05-20 [cit. 2020-08-08] Dostupné z WWW: <https://technet.microsoft.com/en-us/library/cc512606.aspx>
JOHANSSON, Jesper M. *Security Watch The Most Misunderstood Windows Security Setting of All Time* [online]. 2016-08-31 [cit. 2020-08-08] Dostupné z WWW: <https://technet.microsoft.com/en-us/magazine/2006.08.securitywatch.aspx>.

Obrázek 1.9: Algoritmus LM-hash pro ukládání hesel



Zdroj: vlastní zpracování

Od verze Windows NT 3.1 z roku 1993 využívá firma Microsoft pro uložení hesel algoritmus ***NT-hash*** s hašovací funkcí MD4:

```
NThash = MD4(UTF-16-LE(password))
```

Ale i v tomto algoritmu *chybí sůl* či větší množství *opakování* pro zpomalení lámání. Změna je velmi obtížná, neboť toto hašování hesel je zahrnuto do mnoha síťových protokolů.[61] Např. hesla pro bezdrátovou síť *eduroam* se stále ukládají ve formátu *NT-hash*, neboť všechny bezpečnější varianty uložení hesel narážejí na chybějící podporu na straně klientských operačních systémů.

Od Windows NT 4 SP 3 se Microsoft snaží zabezpečit haše hesel uložené na disku či v registrech pomocí další vrstvy šifrování s využitím symetrických šifer – problémem je, že klíč pro symetrickou šifru musí být někde uložen. U Windows 2000 přechází Microsoft Network na protokol *Kerberos*, který výrazně omezuje ukládání hesel na stanicích či přenos po síti.

Při nedostupnosti serveru na něm nelze ověřit heslo zadané uživatelem na stanici. Aby uživatel mohl pracovat aspoň v lokálním režimu, tak se na každé stanici připojené do Active Directory ukládají hesla posledních 10 přihlášených uživatelů. Nyní se používá formát DCC2, který používá jako sůl uživatelské jméno a dále funkci PBKDF2, která 10 240krát opakuje SHA-1:

```
DCC1 = MD4(MD4(Unicode($pass)).Unicode(strtolower($username)))
DCC2 = Truncate128bits(PBKDF2(HMAC-SHA1, 10240, DCC1, username))
```

1.5.11 Výpočty rychlosti lámání

Zadání 1

InSIS před několika lety generoval počáteční heslo dle schématu nnxxxnxxX *(dvě číslice, tři malá písmena, jedna číslice, dvě malá písmena, velké písmeno). Za jak dlouho by program* oclhashcat *otestoval všechny možné kombinace počátečních hesel na grafické kartě AMD*

[61] V době vzniku algoritmu bylo odchytávání hesel na síti mnohem palčivějším problémem než případný únik databáze s haši hesel.

Radeon HD 7970, pokud by hesla byla uložena pomocí LM hash? Grafická karta zvládne otestovat 2384 milionů (tedy skoro 2,4 miliardy) kombinací hesel uložených pomocí LM hash.

Řešení

Počet kombinací je spočítaný v kapitole 1.4.2. Doba lámání se spočítá jednoduše – počet kombinací se vydělí rychlostí, tj.

$$\frac{(10^3 * 26^6)}{2\ 384\ 000\ 000}$$

Výsledek je přibližně 130 vteřin.

Zadání 2

Spočtěte průměrnou dobu lámání hesel o entropii 16, 32, 40 a 48 (pro každou entropii vždy jednoho hesla) uložených pomocí MD5crypt, pokud útočník použije program oclhashcat, který na grafické kartě AMD Radeon HD 7970 zvládne otestovat 3592 tisíc (tedy skoro 3,6 miliónu) hesel MD5crypt za vteřinu. Budou se lišit výsledky v případě, že heslo je uloženo se solí o délce 48 bitů?

Řešení

Výpočet je jednoduchý. Entropie je mocnina 2, takže např. všechny kombinace hesel s entropii 32 se otestují za

$$\frac{(2^{32})}{3\ 592\ 000}$$

tj. za 1196 vteřin. V zadání se ale mluví o *průměrné* době lámání – výsledek je potřeba vydělit dvěma. Heslo se může najít hned či až po prozkoumání téměř všech kombinací. Níže je částečně vyplněná tabulka s výsledky.

Sůl zde nemá vliv – v zadání se mluví o lámání *jednoho náhodně* vygenerovaného hesla. Situace se ale změní, pokud by se mělo lámat např. 100 hesel o stejné entropii. Pokud se nepoužije sůl, tak všechna hesla se najdou nejvýše za 1196 vteřin, možná i za o něco kratší dobu, pokud žádné heslo nebude mezi posledními testovanými kombinacemi. Pokud by pro každé heslo byla jiná sůl, tak na lámání každého hesla potřebujete *v průměru* 598 (polovina z 1196) vteřin a všechna budete mít za 59 800 vteřin (téměř 17 hodin).

Tabulka 1.16: Částečně vyplněná tabulka s řešením druhého zadání

Entropie hesla	Doba lámání ve vteřinách	Doba lámání ve dnech
16		
32	598	
40		
48		453

Zdroj: vlastní zpracování

Zadání 3

Ve vytvářené aplikaci hesla kódujete pomocí PBKDF2 s funkcí SHA-256. V pravidlech pro hesla požadujete minimální délku 9 znaků, z toho minimálně jedno malé písmeno, jedno velké písmeno a jeden nepísmenný znak. V systému bude uloženo přibližně 20 000 účtů. Spočítejte počet iterací pro PBKDF2, aby se výrazně snížilo nebezpečí odhalení hesel při zcizení databáze. Předpokládáte, že útočník k lámání použije počítač s grafickou kartou AMD

Radeon HD7970, na které je schopen spočítat 1032 miliónů SHA-256 hašů za sekundu (tedy přes miliardu hašů za sekundu). Za jeden den by měl útočník získat v průměru jedno heslo, útočník používá útok hrubou silou. Předpokládejte náhodně generovaná hesla.
O kolik bude potřeba zvýšit počet iterací pro hesla vytvářená uživateli *(vycházejte z odhadu entropie dle již neplatné NIST 800-63-2)? Jak se změní výsledek, pokud se navíc budou vkládaná hesla kontrolovat vůči slovníku?*

Řešení
Při *náhodném generování* dle uvedených pravidel vznikne $26 * 26 * 43 * 95^6$ hesel o devíti znacích splňujících podmínky (je 43 nepísmenných znaků). To je přibližně $2{,}1 * 10^{16}$ kombinací. Útočník za den provede $1\,032\,000\,000 * 60 * 60 * 24 \approx 8{,}9 * 10^{13}$ operací SHA-256. Pokud by každý uživatel měl odlišnou sůl, tak odhalí jedno heslo za přibližně 4 měsíce při nastaveném jednom opakování v PBKDF2 (všechny kombinace pro jednu sůl otestuje za 240 dnů). Tedy jedno opakování v PBKDF2 u náhodně generovaných hesel vyhovuje zadání.[62]

Zcela jiná situace je v případě, že si heslo *volí uživatel* a *odhadujeme entropii* dle NIST 800-63-2. Odhad entropie je 25,5 (viz kapitola 1.4.4), tj. jako by bylo $2^{25{,}5} \approx 4{,}75 * 10^7$ náhodných hesel. Jednoduchým výpočtem $2 * (8{,}9 * 10^{13}/4{,}75 * 10^7)$, zjistíme, že za jeden den útočník odhalí přibližně 3 750 000 hesel při jednom opakování v PBKDF2. Pokud má útočník odhalit jedno heslo za den, je potřeba nastavit 3,75 milionu opakování do PBKDF2. Pokud se budou uživateli vytvářená hesla kontrolovat vůči slovníku, tak se entropie dle NIST 800-63-2 zvýší na 31,5 a výsledný potřebný počet opakování v PBKDF2 se tím sníží na přijatelných 60 000.

Zadání 4
Pokračování předchozího zadání. Kolik uživatelů se bude moci za vteřinu přihlásit na Vašem serveru se čtyřmi jádry Intel Xeon E3-1230? Tento procesor zvládá 36 milionů SHA-256 operací za vteřinu.

Řešení
Pokud v PBKDF2 nastavíte 3,75 miliónu iterací, tak náš server zvládne za vteřinu ověřit heslo *pouze* 9 *uživatelům*. Systém bude *obtížně použitelný až nepoužitelný*. Při počtu 20 000 účtu lze snadno odhadnout, že během běžné pracovní doby se minimálně několikrát za den (v tzv. špičkách, např. ráno, po obědě ap.) bude chtít přihlásit mnohem více uživatelů. Navíc bude snadné na něj dělat DoS útoky.[63] Pokud se použije slovník a počet opakování nastaví na 60 000, tak náš server zvládne za vteřinu ověřit hesla 600 uživatelů.

Zadání 5
Vytvářite aplikaci, která bude ukládat citlivé údaje do souboru zašifrovaného pomocí AES s klíčem o délce 256 bitů. Klíč budete odvozovat z uživatelem zadaného hesla pomocí PBKDF2 a hašovací funkce SHA-512. Kolik opakování nastavíte, když chcete na počítači s procesorem Intel(R) Xeon(R) CPU E5-2430L otevřít soubor za jednu vteřinu. Na tomto procesoru se provede 1 680 000 operací SHA-512 za vteřinu (pro vstup o délce 64 B).

[62] Sami si spočítejte, jak se změní počet iterací v případě, že by všechna hesla měla stejnou sůl.
[63] Denial of Service, česky odepření služby (nebo též přetížení služby). Útočník se snaží, aby cílový systém byl zahlcen a nebyl tak schopen fungovat pro své běžné uživatele.

Řešení
Zde postačí prostá úvaha – když procesor provede 1 680 000 operací SHA-512 za vteřinu, tak potřebujete nastavit 1 680 000 opakování v PBKDF2.

1.5.12 Algoritmy scrypt a Argon2

Na konci kapitoly 1.5.8 jsme se dostali k problému obrovského rozdílu rychlosti hašovacích funkcí mezi CPU, grafickou kartou a zákaznickými integrovanými obvody (ASIC a FPGA). Částečně odolný je algoritmus *bcrypt*, zde si popíšeme další dva.

Colin Percival v roce 2009 navrhoval zabezpečení ukládání záloh do on-line úložiště *Tarsnap*. Zálohy se šifrují pomocí AES-256,[64] pro odvození klíče z hesla navrhl nový algoritmus ***scrypt*** s cílem co nejvíce zvýšit náklady na výrobu ASIC modulů pro lámání hesel. Navrhl proto algoritmus náročný na paměť, který lze velmi obtížně paralelizovat.

V základním nastavení scrypt potřebuje 4 MB RAM. Při lámání lze použít menší množství paměti za cenu výrazného zvýšení nároků na procesor. V roce 2012 byl algoritmus *scrypt* popsán jako Internet Draft.

Algoritmus má následující vstupní parametry:

- heslo
- sůl
- $\boldsymbol{N}$ určuje počet interních iterací a tím náročnost na paměť a CPU,
- $\boldsymbol{r}$ určuje velikost tabulky použité v základních blocích algoritmu, vyšší hodnota parametru znamená větší náročnost na paměť bez zvýšení nároků na CPU,
- $\boldsymbol{p}$ určuje počet paralelních průchodů, tím lze zvýšit nároky na CPU bez zvýšení nároků na paměť,
- $\boldsymbol{dkLen}$ požadovaný počet bytů výsledného haše.

Při hašování scrypt potřebuje $\mathbf{128} * \boldsymbol{r} * \boldsymbol{N}$ bytů paměti. Nyní doporučované parametry[65] jsou $\boldsymbol{p} = \mathbf{1}, \boldsymbol{r} = \mathbf{8}, \boldsymbol{N} = \mathbf{2^{14}}$**.** Při tomto nastavení se zahašuje na procesoru Intel i5 heslo asi za 35ms při využití 16 MB RAM. Při odvozování klíče pro zašifrování souboru na uživatelské stanici je vhodné hodnoty zvýšit, aby ověřování trvalo aspoň 100 ms a použilo se více paměti.

Pro ověřování hesel na zatíženém serveru se naopak musí snížit výpočetní náročnost i spotřeba paměti. Při snížení $\boldsymbol{N}$ na $\mathbf{2^{10}}$ se heslo ověří za 2 ms při využití 1 MB RAM. Server s 8 jádry by měl zvládnout DoS útok, při kterém se útočník bude snažit ověřit 1000 uživatelů za vteřinu. Samotné ověřování hesel při útoku obsadí jeden až dva GB RAM. Při úniku hašů hesel bude útočník schopen lámat přibližně 1000 hesel za vteřinu na jednom počítači.

V roce 2013 vyhlásila nezávislá skupina kryptologů otevřenou soutěž „**Password Hashing Competition**“ na získání vhodných algoritmů pro ukládání hesel,[66] V roce 2015 vyhlásila vítěze – algoritmus **Argon2,** který má varianty *Argon2i* (menší pravděpodobnost úniku informací bočními kanály), *Argon2d* (větší odolnost vůči lámání pomocí GPU) a hybridní *Argon2id*.[67] Doporučuje se používat hybridní verzi, pokud nemáme zvláštní důvody proč preferovat jinou variantu. Algoritmus má následující parametry:

---

[64] *Tarsnap cryptography* [online]. [cit. 2020-08-08] Dostupné z WWW: <https://www.tarsnap.com/crypto.html>.
[65] FERRARA, Anthony. *Why I Don't Recommend Scrypt* [online]. 2014-03-12. [cit. 2020-08-08] Dostupné z WWW: <https://blog.ircmaxell.com/2014/03/why-i-dont-recommend-scrypt.html>.
[66] *PHC call for submissions* [online]. [cit. 2020-08-08] Dostupné z WWW: <https://password-hashing.net/cfh.html>
[67] *PHC winner argon2* [online]. [cit. 2020-08-08] Dostupné z WWW: <https://github.com/p-h-c/phc-winner-argon2>

- otevřené heslo,
- sůl,
- počet iterací (výchozí hodnota je 3),
- potřebnou paměť, 2^N KiB (výchozí hodnota 12, tj. 4096 KiB),
- počet vláken (výchozí hodnota 1).

1.5.13 Současná doporučení pro ukládání hesel

Pokud provozujete systém, který *vytvořil někdo jiný*, tak obvykle příliš možností ke změně algoritmu nemáte. Někdy lze úpravou konfigurace bezpečnost zlepšit:

- soubor se soukromým klíčem pro *openssh* lze převést do nového bezpečnějšího formátu,
- můžete zvýšit počet iterací algoritmu *sha512crypt* pro hesla z výchozích 5 000, většinou je bezproblémové zvýšení na desetinásobek,
- u správců hesel lze často zvýšit počet iterací pro generování klíče z hlavního hesla.

V některých případech musíte dále používat z dnešního pohledu slabé uložení hesel. Příkladem může být program *PuTTY* (viz strana 37) či ukládání hesel v bezdrátové síti *eduroam*, kde se musí používat NT-hash (z již popsaných důvodů).

Při *vytváření nové aplikace* máte více možností:

- Nejlepší je *neukládat hesla* a uživatele autentizovat pomocí některého single sign-on protokolu vůči existující adresářové službě. Příkladem může být ověřování prostřednictvím účtu na Facebooku, na Google či přes *Open-ID*. Ve firemních aplikacích lze využít systémy s protokoly *Kerberos, SAML* či *OAuth*.
- Pořídit *hardwarový kryptografický modul* či lépe dva a pomocí nich pepřit či šifrovat hesla, viz kapitola 0. Je to drahé řešení, ale v případě stovek tisíc uživatelů nejvhodnější. Zaplaťte i profesionálního kryptologa, aby Vám udělal návrh či aspoň audit ukládání a ověřování hesel.
- *Bezpečně hašovat hesla* a ukládat do databáze či adresářové služby. O této variantě bude zbytek kapitoly.

V následující tabulce jsou algoritmy pro ukládání hesel seřazeny *dle odolnosti vůči lámání*. Výběr vhodného algoritmu však není jednoduchý, důvody jsou v tabulce a dalším textu.

Tabulka 1.17: Přehled algoritmů doporučovaných pro bezpečné ukládání hesel seřazený dle odolnosti vůči lámání

#	Algoritmus	Poznámky
1.	Argon2	Nový algoritmus z otevřené soutěže, ale jeho bezpečnost není ověřena praxí. *Argon2* ani použitá hašovací funkce *BLAKE2* nejsou národními standardy, je navržen Internet Draft. Málo implementací, chybí podpora v knihovnách pro ověřování hesel.
2.	scrypt	Navržen pro generování klíčů z hesel, ne primárně pro ukládání hesel. Jsou pochybnosti, zda je dostatečně bezpečný při konfiguraci s menší pamětí.[68] Algoritmus *scrypt* ani použitá šifra *Salsa20/8* nejsou národními standardy, ale publikován jako RFC 7914. Podporován pouze v některých knihovnách pro ověřování hesel.

[68] Viz FERRARA, Anthony. *Why I Don't Recommend Scrypt* [online]. 2014-03-12. [cit. 2020-08-08] Dostupné z WWW: <https://blog.ircmaxell.com/2014/03/why-i-dont-recommend-scrypt.html> či existence ASIC modulů pro těžbu kryptoměny Litecoin, která používá scrypt.

#	Algoritmus	Poznámky
3.	bcrypt	Léty prověřený algoritmus pro ukládání hesel. Ale *bcrypt* není národní standard, šifra *Blowfish*, z které vychází (algoritmus však upraven) byla označena jako „dosluhující“.[69] Implementace v mnoha jazycích, součást mnoho knihoven pro ověřování hesel. Malá odolnost vůči lámání pomocí FPGA.
4.	PBKDF2 + SHA-2 /SHA-3	PBKDF2 i použité hašovací funkce jsou součástí národních standardů. Implementace v různých jazycích, součástí knihoven pro ukládání hesel. Menší odolnost vůči lámání pomocí GPU či pomocí zákaznických obvodů.

Zdroj: vlastní zpracování s využitím zdrojů uvedených v poznámkách 64 až 72

Pokud se nemusíte přizpůsobit národním standardům, tak použijte algoritmus *Argon2* či *bcrypt* – ale viz poznámka 69 (pro generování klíče z hesla je vhodnější použít *scrypt*). PBKDF2 se SHA-256 či SHA-512 je o trochu horší volbou, ale s dostatečným počtem iterací stále velmi dobrý způsob uložení hesla. Umožněte přizpůsobit parametry algoritmů konkrétnímu nasazení či změně podmínek v průběhu času.

Neprogramujte ukládání a ověřování hesel sami, ale použijte vhodnou knihovnu s otevřeným kódem, která byla otestována zkušenými kryptology. V PHP můžete použít knihovnu *phpass*,[70] další knihovnu *defuse/password-hashing*[71] lze volat z jazyků PHP, C#, Ruby či Java. Jinou známou knihovnou nejen pro hašování a ověřování hesel je *libsodium*,[72] kterou lze používat z mnoha programovacích jazyků.

U síťových aplikací můžete zkusit přenést část výpočetní zátěže na klienta. Snaží se o to skupina protokolů s označením *Password-authenticated key agreement*, v TLS je navržen protokol *Secure Remote Password* (SRP). Proti situaci ještě před pár lety se zlepšila dostupnost připravených *frameworků* pro nasazení (např. Botan, TLS-SRP, srp-client, node-srp), přesto není vyloučeno, že k bezpečnému využití těchto protokolů budete potřebovat zkušeného kryptologa. U webových aplikací navíc můžete narážet na chybějící binární implementaci kryptografických primitiv v Javascriptu a především na velmi rozdílnou rychlost zařízení (mobil, tablet, notebook).[73]

Pokud ale budou uživatelé používat hesla typu „Password123“, „12345678“, „ILoveYou1“, „HesloHeslo“ či „EduroamHeslo2“, tak Vám ani nejdůmyslnější způsob uložení hesel nepomůže.

Částečným řešením je využití *kvalitního* programu pro odhad síly hesla již v okamžiku, kdy uživatelé poprvé zadávají nové heslo (viz kap. 1.6.7). Nejprve ale v následující podkapitole souhrnně probereme *pravidla pro tvorbu hesel* a *politiky hesel*).

---

[69] „Dosluhující“ znamená do roku 2023, viz NÚKIB. *Minimální požadavky na kryptografické algoritmy: doporučení v oblasti kryptografických prostředků* [online]. 2018-11-28 [cit. 2020-10-18] Dostupné z WWW: <https://nukib.cz/download/uredni_deska/Kryptograficke_prostredky_doporuceni_v1.0.pdf> Je třeba ale brát v úvahu, že *bcrypt* používá upravený algoritmus a *nejedná se* o symetrické šifrování velkého objemu dat.

[70] *Portable PHP password hashing framework* [online]. [cit. 2020-10-18] Dostupné z WWW: <https://www.openwall.com/phpass/>

[71] *Password hashing* [online]. [cit. 2020-10-18] Dostupné z WWW: <https://github.com/defuse/password-hashing>

[72] *Password hashing* [online]. [cit. 2020-10-25] Dostupné z WWW: <https://jedisct1.gitbooks.io/libsodium/content/password_hashing/index.html>

[73] Nativní implementace kryptografických funkcí může být 10x, 100x i vícekrát rychlejší než implementace stejného algoritmu v čistém Javascriptu.

1.6 Vytváření hesel, politiky hesel

Jak si mají uživatelé vytvářet bezpečná hesla? Jaká by měla být firemní politika v oblasti hesel? Na tyto otázky se pokusí odpovědět tato kapitola.

1.6.1 Požadavky na hesla (politika hesel)

Při zadávání hesla jsou na uživatele kladeny *požadavky*, které musí *nové heslo splňovat*. Může to být minimální délka, požadavky na malá a velká písmena či jiné znaky v hesle či kontrola hesla vůči slovníku. Ne všechny požadavky, s kterými se v praxi (často) setkáte, jsou v souladu s aktualizovanými doporučeními (viz srovnání na s. 61: Tabulka 1.24).

Politika hesel je širší pojem, protože obsahuje nejen pravidla pro vytvářená hesla, ale také další požadavky na uživatele, např. požadavky na ochranu hesel, postup při zapomenutí hesla ad. Každá firma by měla mít vlastní dokument politiky hesel. Někdy bývá rozdělen do dvou dokumentů: *pravidla pro tvorbu hesel* a rozsáhlejší *politika hesel*.

Pravidla by měla odpovídat soudobým doporučením a poznatkům (např. by neměla obsahovat již překonané požadavky, které reálně bezpečnost nezvyšují) a také by měla být srozumitelná a co nejméně komplikovaná.

Z více důvodů jsou *zajímavá pravidla* pro hesla z *University of Pennsylvania* (ale i zde lze mít výhrady). Mimo jiné nenutí uživatele k pravidelným změnám hesla: viz též nová verze publikace NIST: NIST SP 800-63-3, s. 60). V těchto pravidlech jsou nároky na složitost hesla *variabilní* v závislosti na jeho *délce* (čím delší, tím méně požadavků).

Dále je součástí pravidel *kontrola hesla* vůči slovníkům zakázaných hesel a seznamu slov, která se nesmějí objevit uvnitř hesla (např. uživatelské jméno).[74] Na univerzitě též vytvářejí *modely přihlašování* jednotlivých uživatelů – kdo se kdy a odkud přihlašuje, s jakou frekvencí, z jakých zařízení a ke kterým službám. A řeší nalezené anomálie.

Tabulka 1.18: Příklad pravidel s proměnlivými nároky v závislosti na délce hesla

<table>
<tr><td colspan="3">The easiest option is a password where length alone makes it resistant to being compromised.
Long passwords have fewer requirements.
Tip: Use four or more unrelated words to form a strong password that's easy to remember, such as: boldaugustpretzelcloud. You can also create a shorter password that introduces more variety and unpredictability into your character combination. A password this long must contain:</td></tr>
<tr><td>20+ characters</td><td>any character type</td><td>žádné speciální požadavky</td></tr>
<tr><td>16–19 characters</td><td>A a</td><td>heslo musí obsahovat velká/malá písmena</td></tr>
<tr><td>12–15 characters</td><td>9 A a</td><td>musí obsahovat velká/malá písmena a číslice</td></tr>
<tr><td>8–11 characters</td><td>? 9 A a</td><td>musí navíc obsahovat ještě speciální znaky</td></tr>
<tr><td colspan="3">PennKey passwords are also screened for easily guessed combinations, such as:
Portions of, or simple variations on, your name or PennKey username
Single dictionary words (English and non-English): Diversifications, Alternativamente
Common 2–3 word combinations: PhiladelphiaEagles
Predictable strings: 123123, abcdef, qwerty, johnpaulgeorgeringo, pa$$w0rd, drowssap
Compromised passwords on published lists
If you have trouble remembering a longer password, write it down and handle it with the same caution you would a credit card.</td></tr>
</table>

Zdroj: Stránka s pravidly pro uživatele z *University of Pennsylvania* (poznámka č. 74)

[74] *PennKey Password Rules* [online]. [cit. 2020-10-25] Dostupné z WWW: <https://pennkeysupport.upenn.edu/password-guidelines>
Některé firmy *používané slovníky zakázaných slov zveřejňují* – příkladem je firma DropBox, která jej umístila na <https://github.com/dropbox/zxcvbn/tree/master/data>.

1.6.2 Pravidelná změna hesla: ano či ne?

Některá z dosud často používaných pravidel jsou *kontroverzní* až *kontraproduktivní*, jak ukázaly různé výzkumy i praxe. To posléze vedlo ke změnám některých oficiálních dokumentů zabývající se heslovou politikou, např. již zmíněný dokument NIST SP 800-63-3 explicitně *nedoporučuje* nutit uživatele měnit hesla: viz srovnání NIST SP 800-63-3 a předchozího NIST SP 800-63-2 (Tabulka 1.24, strana 61).

Nejčastější jsou námitky *proti pravidelné změně hesel*, která má zabránit dlouhodobému zneužití hesla v případě jejího úniku. Jedna z odborných studií pravidelných změn hesel vznikla v roce 2010 na Univerzitě v Severní Karolině,[75] kde používali informační systém s povinnou změnou hesla každé tři měsíce. Nové heslo muselo být odlišné od předchozích hesel použitých za poslední rok, dále požadovali minimálně jedno písmeno, jednu číslice, jeden speciální znak a minimální délku 8 znaků. Hesla ukládali jako MD5 haše. Cílem bylo zjistit, jak snadno mohou při znalosti předchozího hesla rozlomit nové heslo. Ve 41 % případů ho získali během tří vteřin. U 17 % případů pro získání hesla stačilo méně než 5 pokusů.

Ve studii „Quantifying the Security Advantage of Password Expiration Policies“[76] se autoři snaží kvantifikovat vliv změny hesla na úspěšnost získání hesla útočníkem. Pokud se útočníkovi podaří získat haše hesel, tak díky rychlost lámání získá velké množství hesel za dobu výrazně kratší, než je interval pro pravidelnou změnu hesla. I u hádání útočník nejdříve zkouší nejpravděpodobnější hesla a díky tomu s relativně malým počtem pokusů získá hodně hesel[77]. Autoři dochází k závěru, že pravidelná změna hesla má *malý či zanedbatelný* přínos k zabezpečení systému.

Problémy s pravidelnou změnou hesel jsou shrnuty např. ve článku „The problems with forcing regular password expiry“[78] od britské vládní zpravodajské a špionážní organizace GCHQ (nyní NCSC):

- V praktických průzkumech se zjistilo, že častá výměna hesel vede k jednodušším heslům, která se skládají z nějaké pevné části a části, která se pravidelně mění (pořadové číslo či rok a měsíc).
- Pokud má někdo silné heslo, tak není důvod ho měnit. Změna silného hesla znamená riziko, že nové heslo bude slabší.
- Pokud útočník získá heslo, tak obvykle nečeká a zneužije ho okamžitě. Přitom může nainstalovat zadní vrátka, takže změna hesla neomezí budoucí přístup útočníka do systému.
- Pokud uživatel používá nové heslo i jinde a útočník ho zjistí, tak pravidelná změna nepomůže.
- Je pravděpodobnější, že si uživatel pravidelně měněné heslo zapíše na papír, aby si ho zapamatoval.

---

[75] ZHANG, Yinqian; MONROSE, Fabian; REITER, Michael K. The security of modern password expiration: An algorithmic framework and empirical analysis. In: *Proceedings of the 17th ACM conference on Computer and communications security*. ACM, 2010. p. 176–186.

[76] CHIASSON, Sonia; VAN OORSCHOT, Paul C. Quantifying the security advantage of password expiration policies. *Designs, Codes and Cryptography*, 2015, 77.2-3: 401–408.

[77] Pro modelování průběhu lámání použili metriku Partian guessing metric z analýzy BONNEAU, Joseph. The science of guessing: analyzing an anonymized corpus of 70 million passwords. In: *2012 IEEE Symposium on Security and Privacy*. IEEE, 2012. p. 538–552.

[78] *The problems with forcing regular password expiry* [online]. 2016-10-05 [cit. 2020-10-25] Dostupné z WWW: <https://www.ncsc.gov.uk/blog-post/problems-forcing-regular-password-expiry>.

1.6.3 Jedinečnost hesla

Pokud útočníci získají heslo uživatele na nějakém serveru, tak v dalším kroku zkoumají, zda dotyčný nepoužívá stejné heslo i na jiných serverech. Protože mnoho serverů používá jako přihlašovací jméno e-mailovou adresu, tak lze toto zkoumání snadno zautomatizovat. Toto vede ke známému požadavku na hesla – *pro každou službu používat odlišné heslo*.

Odborné studie ukazují, že uživatelé stejné heslo používají velmi často:

- 61 % uživatelů uvedlo, že hesla používají na více serverech – studie CSID.[79]
- 55 % uživatelů zatrhlo volbu „Používám stejné heslo na většině či všech webech“: studie Ofcom z roku 2013.[80]
- 59 % uživatelů používá stejné heslo u Yahoo!Voices a SonyPictures.com – zjištěno na základě porovnání údajů z úniků účtu z těchto dvou služeb v roce 2012.[81]

V roce 2016 na Michiganské univerzitě zjišťovali mezi studenty vícenásobné používání stejných hesel,[82] konkrétně se snažili odpovědět na dvě otázky:

- Sdílí na více webech uživatelé častěji slabší či silnější hesla?
- Sdílí na více webech uživatelé častěji hesla, která často používají či hesla používaná zřídka?

Odpovědi jsou možná pro někoho překvapivé – častěji se sdílí silnější hesla a je vysoká korelace mezi sdílením hesla a častým používáním. Čím častěji uživatel heslo zadává, tím pravděpodobněji ho sdílí na více serverech.

Jak často hesla unikají? Rozsáhlé úniky (často s milióny a více účty) mají velkou publicitu, ale nejsou tak časté. Web *haveibeenpwned.com* ke konci října 2020 eviduje téměř 500 úniků, z toho 47 za rok 2020 (tedy vlastně za 10 měsíců).[83] Malých úniků je ale obrovské množství. A to mluvíme pouze o zveřejněných únicích, velká část případů zůstává utajena a útočníci se snaží získaná hesla zpeněžit či jinak využít.

Pokud uniknou nějaká hesla, tak není obtížné otestovat, zda uživatel nepoužívá stejné heslo i na dalších službách. Pro automatizaci lze použít například některý z následujících nástrojů:

- *Credmap*[84] – Credmap je open source nástroj, který případné znovupoužití hesla otestuje na několika často využívaných webech.
- *Shard*[85] – je nástroj určený pro příkazovou řádku, schopný testovat znovupoužití hesla.

Firma *Akamai*, provozovatel jedné z největších sítí pro doručování obsahu (Content Delivery Network) každoročně publikuje (různě zaměřené) zprávy o internetové bezpečnosti.

---

[79] *Consumer Survey: Password Habits* [online]. 2012-09 [cit. 2020-10-25] Dostupné z WWW: <https://www.csid.com/wp-content/uploads/2012/09/CS_PasswordSurvey_FullReport_FINAL.pdf>.

[80] <http://stakeholders.ofcom.org.uk/binaries/research/media-literacy/adult-media-lit-13/2013_Adult_ML_Tracker.pdf> (v současnosti studie již není dostupná).

[81] HUNT, Troy. *What do Sony and Yahoo! have in common? Passwords!* [online]. 2012-07-12 [cit. 2020-10-25] Dostupné z WWW: <https://www.troyhunt.com/2012/07/what-do-sony-and-yahoo-have-in-common.html>.

[82] WASH, Rick, et al. Understanding Password Choices: How Frequently Entered Passwords are Re-used Across Websites. In: *Symposium on Usable Privacy and Security (SOUPS)*. Denver, 2016. ISBN 978-1-931971-31-7

[83] *Pwned websites* [online]. [cit. 2020-10-31] Dostupné z WWW: <https://haveibeenpwned.com/PwnedWebsites>.

[84] SALGADO, Robert. *credmap* [online]. [cit. 2020-10-25] Dostupné z WWW: <https://github.com/lightos/credmap>.

[85] OKEEFE, Philip. *shard* [online]. [cit. 2020-10-25] Dostupné z WWW: <https://github.com/philwantsfish/shard>.

Ve své zprávě o bezpečnosti z roku 2016 Akamai mimo jiné uvádí:[86]

- Za první čtvrtletí 2016 zaznamenali 388 miliónů pokusů o přihlášení na weby zákazníků z 817 390 různých IP adres. Pro přihlašování použili 65 miliónů různých e-mailů.
- Na základě analýzy odhadují, že 70 % pokusů je prostřednictvím *automatizovaných nástrojů* či *botnetů*.
- Špičky pokusů jsou po únicích velkého množství hesel.

Ve zprávě z roku 2019 zaměřené *na phishing* (který je nejvíce využíván právě pro získání přihlašovacích údajů, případně ke krádežím identity) Akamai mimo jiné uvádí:[87]

- Phishing se adaptuje dle vývoje technologií, např. vedle e-mailů nyní široce využívá mobilní zařízení a sociální sítě. Běžně se nabízí též „phishing jako služba".
- Akamai monitoroval po dobu 60 dní přes 2 miliardy závadných domén. Z nich 89 % bylo funkčních méně než 24 hodin a 94 % méně než 3 dny.

Problém je sdílení i podobných hesel mezi různými službami. Autoři studie „Targeted Online Password Guessing"[88] vytvářejí personalizované slovníky na základě informací o osobě z úniků hesel a dalších údajů z jiných serverů, dle informací z cílového serveru, dle politiky hesel na cílovém serveru apod. S personalizovaným slovníkem na tisíc pokusů uhodli heslo ve 32 % až 73 % případech v závislosti na zaměření cílového serveru (u technických serverů byla nižší úspěšnost). Pokud si vzpomínáte na hádání PINů z kapitoly 1.2.5, tak by Vás výsledky studie neměly příliš překvapovat.

Požadavek *pro každou službu používat odlišné heslo* má své opodstatnění a měl by být součástí politiky hesel. Proti tomu je ale jeden podstatný argument – člověk je schopen si zapamatovat jenom malé množství hesel. Existuje jedno řešení – *generovat náhodná hesla* a ukládat je ve *správcích hesel*.

1.6.4 Správci hesel

Správce hesel lze rozdělit do následujících skupin:

- Správci hesel *ve webovém prohlížeči,* např. v Chrome či ve Firefoxu. Zabezpečení hesel závisí na zabezpečení vlastního počítače. Problematické zálohování a synchronizace.
- *Samostatné aplikace*, které ukládají hesla do souboru na počítači, mobilu. Synchronizace a zálohování obvykle závisí na uživateli. Příkladem může být Keepass,[89] Enpass[90] nebo 1Password.[91]
- *On-line správci hesel*, kteří ukládají hesla do cloudu. Díky tomu je máte k dispozici z různých zařízení. Při použití z různých platforem obvykle musíte platit roční poplatky.

---

[86] *Akamai's state of the Internet security. Q1 2016 report* [online]. [cit. 2020-08-08] Dostupné z WWW: <https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/akamai-q1-2016-state-of-the-internet-security-report.pdf>.

[87] *Akamai's 2019 State of the Internet / Security: Phishing – Baiting the Hook* [online]. [cit. 2020-10-31] Dostupné z WWW: <https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/soti-security-phishing-baiting-the-hook-report-2019.pdf>

[88] WANG, Ding, et al. Targeted Online Password Guessing: An Underestimated Threat. In: *CCS '16: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security.* ACM, October 2016, p. 1242–1254.

[89] *KeePass Password Safe* [online]. [cit. 2020-08-08] Dostupné z WWW: <https://keepass.info/>

[90] *Enpass: Password Manager for iOS, Android, Linux, Windows, Mac* [online]. [cit. 2020-08-08] Dostupné z WWW: <https://www.enpass.io/>

[91] *1Password: Password Manager for Families, Businesses, Teams* [online]. [cit. 2020-08-08] Dostupné z WWW: <https://1password.com/>

Při výpadku spojení na Internet nemusí být hesla dostupná. Příkladem je LastPass,[92] Dashlane,[93] Sticky Password[94] či RoboForm.[95]

- Správci hesel *s čipovou kartou či bezpečnostním tokenem*. Obvykle nejbezpečnější varianta. Nevýhodou je pořizovací cena, obvykle nelze používat na mobilu/tabletu. Problematické je zálohování pro případ ztráty čipové karty či bezpečnostního tokenu.
- *Bezstavoví správci hesel* nikam nic neukládají. Konkrétní heslo se dynamicky počítá pomocí hašovací funkce z hlavního hesla a značky pro konkrétní web (např. doménové jméno). Příkladem může být Password Hasher.[96]

Obrázek 1.10: Password Hasher – odvozování hesla pro konkrétní web z hlavního hesla a značky (tag)

Zdroj: vlastní zpracování

Každý uživatel má trochu odlišné požadavky na správce hesel, nejčastějších jsou:

- *Bezpečnost* ukládání a sdílení. Autoři by měli zveřejnit podrobný popis zabezpečení ukládaných hesel – použité kryptografické funkce, které položky se šifrují, ochrana dešifrovaných hesel v paměti, bezpečnost synchronizace.
- *Důvěryhodnost* programu a tvůrce/provozovatele aplikace. Přečíst si dostupná srovnání správců hesel i diskuse k vybranému správci.
- *Podporované platformy* – tj. zda je funkční ve Vámi používaných operačních systémech.
- Synchronizace mezi počítači či mezi počítačem a mobilem. I samostatné aplikace většinou mají nějaký způsob synchronizace, často přes cloudové služby pro sdílení souborů.

---

[92] *LastPass: #1 Password Manager & Vault App, Enterprise SSO & MFA* [online]. [cit. 2020-08-08] Dostupné z WWW: <https://lastpass.com/>

[93] *Dashlane: Password Manager App for Home, Mobile, Business* [online]. [cit. 2020-08-08] Dostupné z WWW: <https://www.dashlane.com/>

[94] *Sticky Password: Best password manager and free password safe* [online]. [cit. 2020-08-08] Dostupné z WWW: <https://www.stickypassword.com/>

[95] *RoboForm: Manage your passwords with ease and security* [online]. [cit. 2020-08-08] Dostupné z WWW: <https://www.roboform.com/>

[96] MACKIE, Simon. *Use a Password Hasher to Generate More Secure Passwords* [online]. 2010-08-13 [cit. 2020-08-08] Dostupné z WWW: <https://gigaom.com/2010/08/13/use-a-password-hasher-to-generate-more-secure-passwords/>

- *Automatické doplňování údajů do přihlašovacích formulářů*. Obvykle se instaluje rozšíření do webového prohlížeče či aplikace simuluje přes schránku vstup z klávesnice.
- *Generování hesel* – dle různých kritérií jako je délka, použité znaky či zapamatovatelnost.
- *Podpora přihlašování v jiných aplikacích* – někteří správci ukládají hesla jen z webových stránek. Většina správců hesel umí automaticky doplnit přihlašovací údaje jen na webu.
- Možnost *ukládání jiných citlivých informací* jako údaje z bankovní karty, údaje z občanského průkazu nebo licenční čísla k software. Vítaná je i možnost vložit soubory, např. certifikáty k elektronické poště či licenční soubory.
- *Zálohování a obnova* je důležité hlavně u správců bez automatického ukládání do cloudu. Ale i u cloudových služeb je vhodné počítat s útoky typu ransomware.
- *Sdílení* některých přihlašovacích údajů s dalšími uživateli, např. v rámci skupiny spolupracovníků či v rámci rodiny. U některých správců lze řešit pomocí více databází hesel a zajištěním synchronizace mezi uživateli.
- *Cena*.

Používání *dobrých* správců hesel s náhodně generovanými hesly se považuje za bezpečnější než používání stejného hesla na mnoha serverech. V tomto kontextu *je zákaz ukládání/ zapisování hesel nevhodný*, v politice hesel by měl být požadavek na bezpečné uložení používaných hesel. Neexistuje ale konsensus, co konkrétně znamená požadavek na bezpečné uložení hesel.

Ve správcích hesel si uživatel může generovat a ukládat *většinu hesel*. Stále ale bude zbývat několik silných hesel, které si musíte *pamatovat*.

1.6.5 Jak silné heslo potřebujete?

Jak silné heslo si potřebujete zapamatovat? Pokud útočník hádá hesla, tak závisí na tom, po kolika neúspěšných pokusech se systém zablokuje.

U hardwarových zařízení, ze kterých nelze získat kryptografické informace, se obvykle používá PIN se 4 číslicemi a zařízení/karta se zablokuje po 3 až 12 chybných pokusech o zadání PINu. Pravděpodobnost uhodnutí je menší než 0,12 %. Pokud si PIN volí uživatel sám, měl by aplikovat personalizovaný seznam blokovaných hodnot (viz kapitola 1.2.5).

Omezení počtu pokusů o hádání hesla v internetových aplikacích chybí nebo je výrazně vyšší. V již zmiňované studii „Targeted Online Password Guessing“ (viz kapitola 1.6.3) autoři počítají s možností 1000 pokusů o uhodnutí hesla u webových aplikací. Pokud bychom chtěli mít pravděpodobnost 0,1 % uhodnutí na 1000 pokusů, tak by heslo mělo mít entropii alespoň 20.

Autoři knihovny *zxcvbn* (bude popsána v kapitole 1.6.7) předpokládají webovou aplikaci, která se nijak nebrání pokusům o zadání hesla. Útočník použije 100 počítačů (botnet), na každém z nich poběží skripty, které otestují 100 hesel za vteřinu. Dobrá jsou hesla neuhodnutelná při této rychlosti za 10^6 vteřin (entropie 33) a silná, pokud se neodhalí za 10^8 vteřin (entropie 40).

Ve studii „Towards Reliable Storage of 56-bit Secrets in Human Memory“[97] autoři odhadují, že NSA je schopna za 1 milion USD sestavit zařízení, které spočítá 2^{70} hašů za rok

[97] BONNEAU, Joseph; SCHECHTER, Stuart. Towards reliable storage of 56-bit secrets in human memory. In: *23rd USENIX Security Symposium (USENIX Security 14)*. 2014. p. 607–623.

(předpokládají se operace SHA, SHA-2 či SHA-3). Při vynaložení miliardy USD se výkon zvýší na 2^{80} hašů za rok.

Ne každý je ale cílem NSA. Firma[98] či zločinecká skupina je schopna si sestavit či zakoupit speciální server pro lámání. Firma Sagita HPC prodává za necelých 20 tis USD server Brutallis, který je schopen za vteřinu spočítat 11 231 miliónů hašů algoritmu SHA-256. Za půl roku je to 2^58 hašů.

Pokud by Vaše heslo lámal jednotlivec pomocí grafické karty „AMD Radeon R9 290 Series“, tak je schopen za měsíc spočítat 2^{52} hašů SHA-256. Na procesoru Core i7-2600K za týden spočítá 2^{42} hašů.

Počet vyzkoušených kombinací závisí i na způsobu uložení hesla – na počtu iterací hašovací funkce. Např. 1000 iterací snižuje požadovanou entropii o 10, použijeme-li 5000 iterací, snížení je o 11,5. Aby pravděpodobnost uhodnutí byla menší než 50 % za uvedenou dobu, tak přičteme k entropii hodnotu 1.

Tabulka 1.19: Požadovaná entropie hesla pro různé útočníky a způsoby uložení hesla

Schopnosti útočníka	**Uložení hesla**		
	5000 iterací	**1000 iterací**	**1 iterace**
2^70 (NSA za rok)	58,5	61	71
2^58 (firma za půl roku)	46,5	49	59
2^52 (jednotlivec s GPU za měsíc)	41,5	43	53
2^42 (jednotlivec s CPU za týden)	31,5	33	43

Zdroj: vlastní zpracování

Pokud se v systému ukládá otevřený text hesla, tak ho útočník zjistí bez ohledu na jeho kvalitu. Ale to neznamená, že u těchto systémů máte používat heslo „123456“, neboť se potřebujete bránit i proti hádání hesel.

1.6.6 Zapamatovatelnost hesel – pravidlo sedm plus/mínus dva

Psycholog Georg A. Miller již v roce 1956 při testech krátkodobé paměti dospěl k závěru, že člověk je schopen si zapamatovat maximálně sedm samostatných elementů plus mínus dva.[99] Element může být písmeno, vyslovitelná slabika či smysluplné slovo.[100]

Tomuto omezení lidské paměti je potřeba přizpůsobit vytváření hesel. Vhodnější je vytvářet delší hesla ze slabik či ze slov než nutit uživatele si zapamatovat kratší heslo z náhodných znaků. Množina slabik a slov je v jazycích používajících pro zápis nějakou abecedu výrazně větší než množina dostupných znaků.

---

[98] Některé firmy preventivně lámají hesla svých zaměstnanců. Pokud se to podaří, tak si zaměstnanec musí změnit heslo.

[99] MILLER, George A. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological review*, 1956, 63.2: 81.

[100] Pokud posloupnost náhodných slov odpovídá struktuře jazyka, tj. podstatná jména, přídavná jména, slovesa jsou ve správném pořadí, tak si je člověk schopen zapamatovat o několik slov více. Viz TULVING, Endel; PATKAU, Jeannette E. Concurrent effects of contextual constraint and word frequency on immediate recall and learning of verbal material. *Canadian Journal of Psychology/Revue canadienne de psychologie*, 1962, 16.2: 83.
Experiment prováděli na angličtině, která neohýbá slova. Výsledky podobných testů v češtině budou záviset na tom, zda bude povoleno skloňování a časování náhodných slov.

Ukazuje to i následující tabulka s více způsoby generování náhodného hesla s entropií 48. Náhodný výběr slov (diceware) používá krátký (1296 slov o maximální délce 5 znaků) či dlouhý slovník (7776 slov o průměrné délce 7 znaků) vytvořený v roce 2016 po patronací organizace EFF (Electronic Frontier Foundation)[101]. Jako poslední jsem doplnil generování náhodných slabik – posloupností samohláska-souhláska-samohláska.

Tabulka 1.20: Počet znaků a elementů pro různé způsoby generování náhodného hesla s entropií 48

způsob generování hesla	**ukázka hesla**	**znaků**	**bitů/ znak**	**ele-mentů**	**bitů na elem.**
malá písmena	ifbblqrlzkf	11	4,4	11	4,4
číslice, malá a velká písmena,	TM0pPyRcx	9	5,3	9	5,3
tisknutelné znaky	H<~+LX4q	8	6,0	8	6,0
diceware 1294	ramp land virus uncut steep	27	1,8	5	9,6
diceware 7776	washboard astonish gusto mangle	31	1,5	4	12,0
slabiky	lux-fid-gal-cud-t	17	2,8	5	9,6

Zdroj: vlastní zpracování

Při požadavku na vyšší entropii se doporučuje používat zapamatovatelné heslové fráze. V článku „How to Memorize a Random 60-bit String“[102] autoři popisují několik způsobů generování takových frází (jsou schopni vygenerovat 2^{60} různých heslových frází pro daný typ) a měří jejich zapamatovatelnost. Vedle náhodného výběru ze slovníku (diceware) používají tři způsoby generování gramaticky správných anglických vět. V poslední variantě „Poetry“ se z náhodných slov vytváří rýmující se čtyřstopé jambické dvojverší.

V praktických experimentech autoři zjišťovali úspěšnost zapamatování vygenerovaných hesel i osobní preference konkrétních typů heslových frází. Zde vynechali hesla z varianty „*First Letter Mnemonic“,* protože s nimi měli špatné výsledky v předchozích analýzách.

Tabulka 1.21: Délka a zapamatovatelnost heslových frází s entropii 60 při různých způsobech jejich generování

metoda vytváření heslové fráze	**průměrný počet slov**	**průměrný počet znaků**	**úspěšnost zapa-matování [%]**	**uživatelská preference [%]**
Diceware (32768 slov)	4,0	31,2	58,3	5
First Letter Mnemonic	15,0	87,7		
All Letter Method	11,8	70,8	33,3	39
Frequency method	9,7	55,5	40,0	37
Poetry	7,2	52,7	61,5	19

Zdroj: vlastní zpracování

[101] BONNEAU, Joseph. *Deep Dive: EFF's New Wordlists for Random Passphrases* [online]. 2016-07-19 [cit. 2020-08-08] Dostupné z WWW: <https://www.eff.org/deeplinks/2016/07/new-wordlists-random-passphrases>.

[102] GHAZVININEJAD, Marjan; KNIGHT, Kevin. How to Memorize a Random 60-Bit String. *Parking*, 11.70.8: 58.83.

Zapamatování hesla je velmi výrazně ovlivněno „magickým číslem sedm plus mínus dva“. Pravidelný rytmus u dvojverší pomáhá k zapamatování hesel. Je zajímavé, že při výběru hesla uživatelé preferují gramaticky správné, ale delší věty. Pamatují si je ale jen přibližně, takže je u nich menší úspěšnost opětovného správného zadání.

Generovaná hesla či heslové fráze se používají málo. Mnohé aplikace nechávají vytvoření hesla na uživatelích s tím, že se snaží měřit *kvalitu nových hesel* při zadání prvního hesla nebo změně stávajícího hesla.

1.6.7 Programy odhadující sílu hesla, zxcvbn

PSE = Pasword Strength Estimator(s) nebo též **Password Strength Meter(s)** jsou programy **odhadující sílu hesla** již při zadávání (volbě) nového hesla nebo změně stávajícího. Cíl je zřejmý: působit **preventivně**, tedy pokud možno uživateli vůbec *nedovolit,* aby si zvolil *vysloveně slabé heslo* (např. password, 123456789, qwert123, pro české uživatele též např. slunicko, broucek, kocicka, hesloheslo, Janicka, Jarousek atd. atd., včetně hesel, která jsou jen triviální modifikací výše uvedených hesel: Janicka1, Janicka123, Janicka750 atd. atd.).

V angličtině se asi častěji používá termín *Password Strength Meter* (PSM)*,* což bychom mohli přeložit jako „měřiče síly hesla, ale termín *Pasword Strength Estimator* (PSE) je výstižnější, protože entropie hesla lze spolehlivě spočítat jen u hesel, která jsou *náhodně generovaná*. U hesel, která si *volí uživatelé* (nejsou náhodně generovaná) lze sílu hesla *jen odhadovat* (jak už bylo vysvětleno dříve).

Programy typu PSE mohou pro odhad síly hesla používat *velmi různorodé algoritmy,* od jednoduchých až po matematicky velmi sofistikované včetně kombinace více algoritmů. Mnohé používají též slovníky (viz stručný popis programu ZXCVBN dále), ale některé pracují bez slovníků.

Tyto programy *z principu* nikdy *nemohou být stoprocentně účinné*, ale dobrý PSE může velmi výrazně zvýšit bezpečnost. Různé výzkumy a též zkušenosti z praxe přinesly dvě důležitá zjištění: jedno dobré a jedno špatné. Začneme tím dobrým – pokud se uživateli zobrazuje odhad síly hesla, většina uživatelů se zobrazovanými údaji řídí a *snaží si zvolit silnější heslo*.

Při testu na Univerzitě v Britské Columbii[103] si účastníci experimentu měnili heslo do svého informačního systému. Předtím se program odhadující sílu hesla nepoužíval, při experimentu se některým graficky zobrazovala síla hesla, kontrolní skupině ale ne. Výsledek je dle očekávání – účastnící se zobrazeným odhadem síly hesla vytvářeli složitější hesla. Autoři studie dále testovali různé grafické zobrazení výsledků měření síly hesla – nezjistili výrazné rozdíly.

Tabulka 1.22: Změna entropie nových hesel v závislosti na zobrazení odhadu síly hesla

Odhadovaná entropie	S využitím PSE	Kontrolní skupina
před změnou hesla	46,53	46,53
po změně hesla	60,10	51,70

Zdroj: vlastní zpracování (dle studie Univerzity v Britské Columbii, poznámka 103)

[103] EGELMAN, Serge, et al. Does my password go up to eleven?: the impact of password meters on password selection. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2013. p. 2379–2388.

K podobnému výsledku došel v roce 2012 B. Ur se svými spolupracovníky.[104] Jejich dalším poznatkem bylo, že pokud nastavená pravidla vedou k heslům z velkého množství elementů, tak výrazně klesá ochota dosáhnout nejlepšího hodnocení.

A nyní to *špatné zjištění*. Různé analýzy bohužel ukázaly, že většina PSE dává přinejlepším *nekonzistentní*, v horším případě *velmi špatné výsledky*, např. u šesti v té době velmi často používaných programů (využívaných na tak navštěvovaných webech jako je např. Twitter nebo Yahoo). *Jediný ZXCVBN* všechna uvedená hesla označil jako *velmi slabá*, relativně slušný výsledek měl ještě program č. 3 (Mato Ilic's PWStrength), který 3 hesla označil též jako „very weak".

Uvedený test byl jednoduchý (což ale umožňuje snadné srovnání výsledků). V tomto testu bylo použito *jen několik mimořádně slabých hesel* (viz tabulka níže), všechny byly vybrány ze seznamu „10 000 nejhorších hesel", např. heslo „abc123" je v tomto seznamu 14. nejčastější a první, které používá kombinaci písmen a číslic (kromě toho délka 6 znaků je dnes sama o sobě zcela nedostačující). Heslo „primetime21" je v seznamu na pozici 8280 a nejdelší, které kombinuje písmena a číslice.

Autor článku dále kritizuje, že některé programy používají matoucí nebo dvojznačné termíny: co znamená „medium" či „mediocre"? Lze namítnout, že pouhých 6 hesel není reprezentativní studie, ale šlo o tak slabá hesla, že výsledky jsou přesto zarážející. Navíc podobné výsledky zaznamenaly i jiné studie, které zkoušely tisíce různých hesel.[105]

Tabulka 1.23: Tristní výsledky 5 programů typu PSE, uspěl jediný ZXCVBN

Heslo	**Testované programy**					
	1	**2**	**3**	**4**	**5**	**zxcvbn**
abc123	weak	weak	very weak	weak	weak	very weak
trustno1	normal	weak	very weak	good	make it stronger	very weak
ncc1701	medium	weak	very weak	weak	make it stronger	very weak
iloveyou!	medium	weak	mediocre	good	weak	very weak
primetime21	medium	medium	weak	good	make it stronger	very weak

Zdroj: podle článku na webu NakedSecurity (viz poznámka 105)

Jestliže *dobrý PSE* je významný bezpečnostní nástroj, *špatný PSE* je možná horší než žádný, protože uživateli dá **falešný pocit bezpečí:** bude se domnívat, že si zvolil silné heslo. Ve skutečnosti však půjde o velmi slabé heslo: je velmi pravděpodobné, že takovéto heslo útočník vyzkouší i při online útoku, a prakticky jisté při off-line útoku.

Autorem programu ZXCVBN je *D. L. Wheeler* ze společnosti *Dropbox*, na jejím webu byl proto nasazen nejdříve. Protože prokázal svoji kvalitu v reálných podmínkách, dnes ho využívá celá řada nejnavštěvovanějších amerických webů, rovněž je součástí standardní instalace např. programu *WordPress*.

Program *kombinuje více metod* pro odhad síly hesla. Používá několik menších, ale pečlivě sestavených slovníků (počet a velikost slovníků lze samozřejmě upravit): typicky

[104] UR, Blase, et al. How does your password measure up? the effect of strength meters on password creation. In: *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*. 2012. p. 65–80.
[105] STOCKLEY, Mark. *Why you STILL can't trust password strength meters* [online]. 2016-08-17 [cit. 2020-10-25] Dostupné z WWW: <https://nakedsecurity.sophos.com/2016/08/17/why-you-still-cant-trust-password-strength-meters/>. Hesla pro test byla vybrána z následující „seznamu nejhorších hesel": BURNETT, Mark. *10,000 Top Passwords* [online]. 2011-06-21 [cit. 2020-10-25] Dostupné z WWW: <https://xato.net/10-000-top-passwords-6d6380716fe0>

obsahuje slovník nejčastějších slov obecné slovní zásoby, nejčastějších příjmení v USA, nejčastějších mužských a ženských křestních jmen v USA a slova vybraná z názvů nejoblíbenějších filmů, televizních seriálů a písniček. Jde tedy o slova, která uživatelé při tvorbě hesel (bohužel) velmi často využívají. Kromě toho obsahuje ještě jeden slovník, v kterém je přímo několik tisíc nejslabších hesel (sestaveno na základě podrobné analýzy miliónů uniklých hesel) a vybrané posloupnosti kláves (viz dále).

Kromě toho obsahuje několik nepříliš složitých, ale promyšlených algoritmů, které kombinují různé postupy a data ze slovníků. Právě proto velmi dobře odhaduje sílu, resp. slabost pro velkou většinu hesel, která pocházejí z *anglického jazykového a kulturního* prostředí. Jako jeden z mála programů dobře hodnotí *také heslové fráze* složené z více slov, hesla kombinující slova a číslice (viz tabulka testů 6 programů výše), kombinace obsahují např. roky, dny či měsíce a v neposlední řadě také *posloupnosti kláves na klávesnici* (např. hesla typu „*qwer1234“*, „a*sdfghjk“* apod. – odsud i *původ názvu programu*).

Mezi další přednosti patří: program je open source software, je multiplatformní, snadno použitelný, nezávisí na dalších knihovnách (který by bylo nutné instalovat), funguje ve všech hlavních WWW prohlížečích a je nenáročný na zdroje. Poslední dvě vlastnosti jsou velmi důležité: v současnosti velká část přihlašování do různých aplikací (viz např. na VŠE přihlašování do InSIS, do pošty nebo do MS Teams pro účast na online výuce) probíhá skrze webový prohlížeč, a to nejen z počítačů, ale také z mobilů, tabletů apod.

Pochopitelně ani ZXCVBN není a nemůže být „dokonalý“. Donedávna ale jeho největší slabina pro nasazení v českém (nebo slovenském) prostředí bylo jeho *zaměření na anglicky* mluvící uživatele. Např. heslo *mojeheslo* ohodnotí 3. stupněm na pětistupňové škále (0 až 4), kde nula znamená mimořádně slabé heslo, 1 velmi slabé, 2 slabé, 3 dobré, 4 silné. Přitom je jasné, že při hádání/lámání hesel v českém prostředí bude toto heslo mezi prvními, která útočník vyzkouší.

Toto se ale týká *všech PSE*, které nějakým způsobem *pracují se slovníky*, nejen programu ZXCVBN. *Dan Wheeler* si tohoto byl dobře vědom, a již v závěru svého článku, který popisuje první verzi programu, zdůrazňuje, že program je zatím zaměřen na angličtinu (a to ještě hlavně na americkou angličtinu).[106]

V současnosti je upravený ZXCVBN již reálně využíván v *českém prostředí*, např. na VŠE při volbě nebo změně hesla pro *InSIS* (integrovaný studijní a informační systém školy) a také pro síť *Eduroam*. Důležitost kvalitního hesla pro *Eduroam* je dána tím, že uživatel jedné sítě se pak stejným účtem přihlašuje do libovolné zapojené sítě (většina vysokých škol v Evropě + např. některá turistická informační centra v ČR).

Autorem úprav je *Luboš Pavlíček* z Centra informatiky na VŠE a zahrnují drobnou úpravu algoritmů, a především české a slovenské slovníky. Přehled, rozsah a struktura slovníků (a také postup jejich vytvoření) jsou uvedeny dále. Také tato verze je volně ke stažení pro všechny.[107]

---

[106] WHEELER, Dan: *zxcvbn: realistic password strength estimation* [online]. 2012-04-10 [cit. 2020-10-25] Dostupné z WWW: <https://dropbox.tech/security/zxcvbn-realistic-password-strength-estimation> Doporučujeme přečíst alespoň Úvod (kde mimo jiné najdete další *malé srovnání PSE*, použitých v té době na webech jako Citibank, Twitter, PayPal, eBay, Facebook, Yahoo, Gmail) a kap. 8. Conclusion (kde mj. uvádí možná vylepšení programu: část z nich se objevila v novější verzi programu, další by však bylo velmi obtížné implementovat).

[107] PAVLÍČEK, Luboš: *ZXCVBN Password Strength Estimation with Czech or Slovak Dictionaries* [online]. [cit. 2020-10-25] Dostupné z WWW: <https://github.com/lpavlicek/zxcvbn-czech>

České slovníky zahrnují přibližně:

- 23 000 nejčastějších českých příjmení a jmen,
- 25 000 hesel vycházejících z českých slov (sestaveno na základě rozsáhlé analýzy mnoha úniků hesel, které prokazatelně patřily českým uživatelům),
- 4000 položek sestavených na základě českých názvů filmů, seriálů apod.,
- 8000 nejběžnějších českých slov obecné slovní zásoby (analýzou české Wikipedie).

Sestavit české (slovenské) slovníky nebylo vždy bez komplikací. Pro výběr nejběžnějších slov obecné slovní zásoby nejen pro češtinu, ale prakticky pro všechny jazyky je nyní výborným zdrojem Wikipedie, pro české názvy filmů a seriálů lze využít CSFD.cz nebo FDb.cz. Nejčastější česká příjmení a jména šlo získat na webu Ministerstva vnitra (které u nás vede evidenci obyvatel a jediné má tato data k dispozici). Po mnoha letech je však najednou přestalo zveřejňovat s podivným zdůvodněním, že není statistický úřad, a tudíž mu nepřísluší tyto statistiky zveřejňovat. Za několik let to může být problém, protože četnosti jmen se v delším období mohou významně změnit. Domácké podoby nejčetnějších jmen (Janička, Jarda, Jaroušek, atd.), která jsou z hlediska tvorby hesel velmi důležitá, zase nebyla nikde k dispozici v elektronické podobě a bylo nutno je opsat z tištěné knihy.[108]

Také česká verze obsahuje i anglické slovníky (mj. proto, že řada českých uživatelů si volí heslo založené na anglických slovech), ale jejich velikost byla redukována pro zachování přiměřené velikosti programu včetně dat. To je důležité zejména pro zařízení s omezenou velikostí pamětí a méně výkonným procesorem a také s ohledem na ceny mobilních dat, které v ČR stále patří mezi nejdražší v EU. Původní program jen s anglickými slovníky má přibližně 0,8 MB, tato verze méně než 0,9 MB.

Není-li celková velikost tak důležitá (např. uživatelé se do aplikace budou přihlašovat pouze z počítačů a na rychlé vnitřní síti firmy), lze samozřejmě k českým slovníkům přidat původní anglické slovníky neredukované velikosti a/nebo využít ještě větší české (slovenské) slovníky. Empirické zkušenosti však ukazují, že od jisté velikosti slovníků jejich zvětšování již není tak zásadní. Např. pro angličtinu poměrně velký původní slovník příjmení pokrývá 80 % americké populace, zatímco výrazně menší slovník jmen pokrývá 90 % populace. Což není překvapivé, protože variabilita jmen je výrazně menší než příjmení.

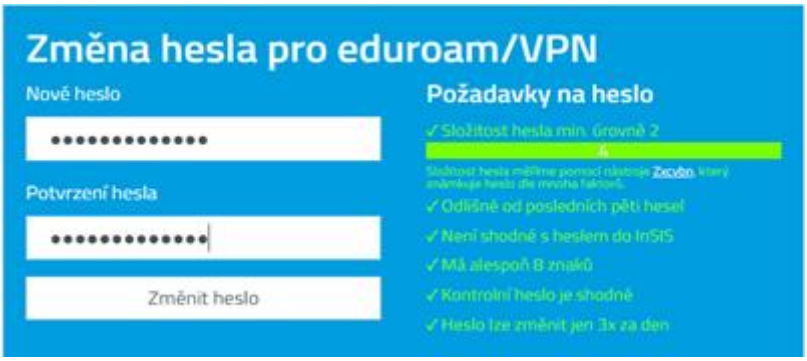
Aktuální velikosti slovníků též *umožňují zkombinovat více jazyků*, pokud je to potřeba. Pokud bychom např. chtěli program ZXCVBN nasadit v prostředí mezinárodní firmy, kde pracuje více národností, lze vytvořit verzi, která bude obsahovat dejme tomu anglické, české, slovenské, německé příp. např. francouzské nebo polské slovníky. Hlavní komplikací je, že (pokud je nám známo), slovníky pro další evropské jazyky zatím nejsou. S využití námi popsané metodologie však není tak těžké (byť poměrně pracné) potřebné slovníky (a také layouty národních klávesnic) vytvořit.

Jak již bylo uvedeno, s programem se nyní setká *každý* zaměstnanec nebo student VŠE při volbě *nového hesla* nebo jeho *změně* pro InSIS nebo pro síť Eduroam. Kromě toho, speciálně pro studenty kursu 4SA313, jsou na stránkách serveru *https://bis.vse.cz*, volba Síla hesla, k dispozici 3 různé verze: jen s původními anglickými slovníky, českými slovníky, slovenskými slovníky. Student kurzu BIS si tak může sám porovnat, jak program vyhodnotí např. heslo *mojeheslo* s původními a upravenými slovníky. Druhou odlišností oproti produkční verzi je, že program vypisuje mnohem podrobnější informace. Produkční verze uživateli zobrazuje jen celkové hodnocení na škále 0 až 4 (číslem i graficky).

---

[108] Knappová, Miloslava. *Jak se bude vaše dítě jmenovat?* Praha: Academia, 2010. 783 s. ISBN 978-80-200-1888-5

Obrázek 1.11: Ukázka produkční verze ZXCVBN na VŠE



Zdroj: vlastní zpracování

Jak vypadá testovací verze programu na serveru *https://bis.vse.cz* a její podrobné výpisy si již vyzkoušejte samostatně. Mnohem podrobnější informace o heslech obecně, programech typu PSE a úpravě ZXCVBN pro české a slovenské jazykové a kulturní prostředí najdete v článku „Adaptation of password strength estimators to a non-English environment — the Czech experience“.[109]

1.6.8 Moderní pravidla pro hesla

V roce 2016 začal americký National Institute of Standards and Technology (NIST) připravovat novou verzi standardu NIST SP 800-63 *Digital Authentication Guidelines*, který obsahuje i *doporučenou politiku hesel*. Nový návrh byl po připomínkách schválen v roce 2018 jako NIST SP 800-63-3 a nahradil předchozí verzi NIST SP 800-63-2 z roku 2013. Nová verze reflektuje poznatky z hádání a lámání hesel a vychází z následujících principů:[110]

- Důraz na *uživatelskou přijatelnost* a přívětivost, jinak budou uživatelé hledat snazší a méně bezpečná řešení.
- *Realistická* bezpečnostní očekávání, v mnoha případech je potřeba dvoufaktorová autentizace. Ve většině případů jedním z faktorů bude heslo.
- Nároky je potřeba klást *na provozovatele* systémů, ne na koncové uživatele.
- Nenutit uživatelům pravidla, která prokazatelně *nezvyšují bezpečnost*.

Změny jsou shrnuty v následující tabulce. Striktně vzato, dokumenty Národního úřadu standardů a technologie mají platnost jen pro USA. V realitě jsou však používány jako „*de facto“ standardy* a „*current best practices*“ po celém světě. Vyplatí se proto změny shrnuté v tabulce pečlivě prostudovat.

Některé úpravy jsou jen zpřesněním a/nebo doplněním starších doporučení a pravidel, jiné představují významnou změnu oproti starším doporučením a také proti mnohdy stále ještě přetrvávající praxi v různých informačních systémech.

---

[109] DOUCEK, Petr; PAVLÍČEK, Luboš; SEDLÁČEK, Jiří; NEDOMOVÁ, Lea. Adaptation of password strength estimators to a non-English environment — the Czech experience. *Computers & Security*. Aug 2020, Vol. 95, No. 8, p. 1–11. ISSN 0167-4048 DOI:10.1016/j.cose.2020.101757

[110] GRASSI, Paul. A., GARCIA, Michael E., FENTON, James L. *Digital identity guidelines: Authentication and lifecycle management*. NIST Special Publication 800-63-3. Gaithersburg: NIST, 2018. DOI: 10.6028/NIST.SP.800-63-3
Kdo chce sám podrobněji porovnat se starší, již *neplatnou* verzí 800-63-2, je stále k dispozici:
BURR, William. E. et al. *Electronic Authentication Guideline*. NIST Special Publication 800-63-2. Gaithersburg: NIST, 2013. DOI: 10.6028/NIST.SP.800-63-2

Tabulka 1.24: Změny požadavků na hesla: srovnání dvou verzí NIST SP 800-63

požadavek na hesla	**již neplatná verze standardu (NIST SP 800-63-2)**	**nová verze standardu (NIST SP 800-63-3)**
minimální délka hesla	6 znaků/4 číslice pro systémy úrovně 1, 8 znaků/6 číslic pro úroveň 2 a vyšší	pro všechny úrovně minimálně 8 znaků či 6 číslic PINu
maximální délka hesla	nebylo specifikováno	maximální délka musí být alespoň 64 znaků (podpora heslových frází)
mezera v hesle	nebylo specifikováno	musí být povolena
znaková sada	90 či více znaků	všechny tisknutelné znaky ASCII tabulky. Doporučeno povolit všechny znaky z Unicode včetně emotikonů
nápovědy k heslu	nebylo specifikováno	nápověda nesmí být přístupná neautorizované osobě, nezobrazovat nápovědu při vytváření/změně hesel
omezení pro hesla (např. jedno malé a jedno velké písmeno)	kontrolovat heslo proti omezením či proti slovníku	nedoporučují omezení na skupiny znaků apod, doporučují kontrolu vůči slovníku (velikost přibližně 100 000 hesel).
ukládání	nesmí se ukládat v plaintextu	musí být použita sůl s entropii aspoň 32bitů, musí být použita PBKDF2 se schválenou hašovací funkci (SHA-1, SHA-2 či SHA-3). Doporučuje se minimálně 10 000 iterací, doporučen HW kryptografický modul.
zobrazení hesla při zadávání	nebylo specifikováno	uživatel má mít možnost zobrazit si po určitou dobu znaky v hesle
pravidelná změna hesla	nebyla specifikována	neměla by se vyžadovat pravidelná změna hesla
obnova hesla	může být pomocí sdílené znalosti (např. jaká je jméno matky za svobodna?) či odpovědi na uživatelem zvolené otázky	nesmí se používat

Zdroj: vlastní zpracování s využitím uvedených publikací NIST (poznámka 110)

Např. nyní se vysloveně *nedoporučuje povinně* vyžadovat po uživateli určité *skupiny znaků* (a naopak doporučuje se kontrola volených hesel dle slovníku, tedy využití *programu typu PSE*), *nedoporučuje* se vynucovat *pravidelnou změnu* hesla a *obnova hesla na základě* odpovědi na otázku typu jméno matky za svobodna je *striktně zakázána*.

Poslední požadavek je zcela jasný (odpovědi na obdobné otázky mohl útočník relativně snadno zjistit), k předchozím je asi vhodný doplňující komentář. Je pochopitelně vhodné, aby heslo obsahovalo co nejvíce různých znaků a pokud si uživatel po určité době heslo sám promyšleným způsobem mění, je to určitě také dobré.

Formální vynucování těchto pravidel však bezpečnost nezvyšuje, uživatele jen obtěžuje, činí hesla obtížně zapamatovatelná a někdy tato pravidla byla tak komplikovaná, že byl problém je vůbec pochopit. Např. heslo „*Janicka123*“ formálně splňuje požadavek aspoň

jedno velké písmeno a jedna číslice, ale pokud nevyužijete vhodný program pro odhad síly hesla, vůbec nezjistíte, že jde o velmi slabé heslo (zvláště pokud si útočník naopak zjistí, že manželka uživatele se jmenuje Jana).

Na závěr této kapitoly doplníme standard NIST ještě o *výběr z doporučení pro správu hesel*, které na konci roku 2015 publikovala Velká Británie, konkrétně její instituce National Technical Authority for Information Assurance, resp. v rámci ní Communications-Electronic Security Group (CESG). Již z názvu dokumentu je zřejmé, že hlavním cílem těchto doporučení je zjednodušit správu hesel z hlediska uživatelů i administrátorů.[111]

- Změňte všechna výchozí hesla.
- Snižte počet hesel uživatelů – omezte množství hesel, které uživatelé musí používat; povolte použití správců hesel; nepožadujte pravidelnou změnu hesla.
- Technická omezení proti hádání hesel jsou mnohem efektivnější, než vynucování složitých (a obtížně zapamatovatelných) hesel po uživatelích.
- Při generování hesel používejte algoritmy, které vytváří zapamatovatelná hesla (heslové fráze, několik náhodných slov ze slovníku, heslo ze slabik apod.). Vygenerujte více návrhů hesla a nechte uživatele heslo zvolit.
- Na administrátory a na osoby pracující ze vzdálených míst by měly být vyšší nároky.
- Monitorujte autentizaci a pokusy o ní – upozorněte uživatele na problematické přístupy, zablokujte účet při podezření ze zneužití apod.

1.7 Vícefaktorová autentizace

Obecně rozlišujeme tzv. vícefaktorovou a dvoufaktorovou autentizaci.

Vícefaktorová autentizace (též vícefázové ověření, anglicky *multi-factor authentication*, běžně je používána i zkratka MFA) je proces, kdy uživatel při autentizaci poskytne několik na sobě nezávislých důkazů (faktorů) potvrzujících jeho identitu.

Dvoufaktorová autentizace (též dvoufázové ověření, anglicky *two-factor authentication*, běžně je používána i zkratka 2FA) je podmnožinou vícefázového ověření. Vyžaduje od uživatele prokázání dvou faktorů. Vedle obecných *bezpečnostních důvodů* může být další příčinou pro (nutné) zavedení dvoufaktorové nebo vícefaktorové autentizace *platná legislativa*.

Např. v zemích EU, a tedy také v České republice vstoupila 14. září 2019 v platnost poslední část druhé revidované směrnice Evropské Unie o platebních službách, tzv. **PSD2** (*Revised Payment Services Directive* neboli Directive (EU) 2015/2366), která nahradila předchozí směrnici PSD z roku 2007. Směrnice byla přijata dne 25. listopadu 2015 a implementována měla být postupně od konce roku 2018 do 14. září 2019. Vedle dalších změn mezi hlavní patří zvýšené povinné požadavky na autentizaci, resp. na kontrolu identity zákazníků při platbách online.[112]

---

[111] *Password Guidance: Simplifying Your Approach* [online]. 2015 [cit. 2020-10-25] Dostupné z WWW: <https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/458857/Password_guidance_-_simplifying_your_approach.pdf>.

[112] Směrnice Evropského parlamentu a Rady (EU) 2015/2366 ze dne 25. listopadu 2015 o platebních službách na vnitřním trhu, o změně směrnice 2002/65/ES, 2009/110/ES, 2013/36/EU a nařízení (EU) č. 1093/2010 a o zrušení směrnice 2007/64/ES [online]. 2015-12-23 [cit. 2020-10-25] Dostupné z WWW: <https://eur-lex.europa.eu/legal-content/CS/TXT/?uri=CELEX:02015L2366-20151223>

Co je to *faktor*? Metody *autentizace uživatelů* (autentizací mezi počítači se zde nebudeme zabývat) lze obecně rozdělit do tří, resp. čtyř skupin. České označení prvních tří skupin je víceméně doslovným překladem „tradičního“ označení v angličtině:

1. *Něco, co víte* (anglicky *something you know*, příp. označováno jako *knowledge factor*): znalost něčeho, co ví, resp. měl by vědět pouze daný uživatel. Typickými příklady jsou právě heslo nebo PIN či jednorázové heslo.
2. *Něco, co máte* (anglicky *something you have*, příp. *ownership factor*): vlastnictví něčeho, co má pouze daný uživatel. Typickými příklady jsou platební karta, čipová karta, tzv. hardwarový (příp. softwarový) token atd.
3. *Něco, čím jste* (anglicky *something you are*, příp. *inherence factor*): nějaká průkazná (neoddělitelná) charakteristika daného uživatele. Souhrnně označováno jako *biometriky,* např. otisk prstu (prstů), snímek sítnice oka, rozpoznávání tváře, hlas, profil způsobu psaní na klávesnici a mnoho dalších.
4. *Kde jste* (anglicky *somewhere you are*, příp. *location-based factor*): ke třem „tradičním“ druhům autentizace uvedeným výše (něco, co…) se dnes zpravidla přidává ještě čtvrtá skupina, dostatečně spolehlivý průkaz toho, že daný uživatel se vyskytuje na určitém místě (např. GPS pozice).

Vedle pojmů dvoufaktorová (vícefaktorová) autentizace se můžeme setkat ještě s příbuzným, ale nikoli stejným pojmem *silná autentizace* (anglicky *strong authentication*). Podle Evropské centrální banky (viz zmíněná směrnice PSD2) jde o proces založený na dvou nebo více autentizačních faktorech. Použité faktory musí být vzájemně nezávislé a aspoň jeden z faktorů nesmí být znovupoužitelný nebo replikovatelný (výjimka z tohoto pravidla platí pro biometriky). V tomto pojetí je velmi blízký pojmům vícefaktorová, resp. dvoufaktorová autentizace, ale má striktněji a precizněji definované požadavky na kombinaci faktorů a na požadované vlastnosti jednotlivých faktorů. Existují však i velmi odlišné definice silné autentizace.

Dvoufaktorová (vícefaktorová) autentizace se rozšiřuje zejména v posledním letech v důsledku zvýšených bezpečnostních požadavků a také jako reakce na platnou legislativu. Lze však uvést známé příklady, kde se dvoufaktorová autentizace používá již mnoho let.

Typickým příkladem jsou *výběry z bankomatů* nebo *internetové bankovnictví*. První on-line bankomat (ATM) na území Česka (resp. Československa) byl instalován v roce 1992 (v zahraničí se pochopitelně využívají ještě mnohem déle). Uživatel musí do bankomatu vložit svou platební kartu (*něco co má*) a zadat PIN (*něco co ví*). V případě on-line bankomatů a výběru hotovosti bankomat navíc ověřuje, že uživatel má na účtu dostatečný zůstatek.

První internetovou bankou u nás (a jednou z prvních v celé Evropě) se stala v roce 1998 tehdejší *Expandia Banka* (od roku 2001 *eBanka*, zakoupena Raiffeisenbankou v roce 2006, po dokončení fúze definitivně zanikla v roce 2009). Tato banka používala dále popsané bezpečnostní tokeny.

S rozšířením mobilů se však v případě internetového bankovnictví nejčastější formou dvoufaktorové autentizace na řadu let stala kombinace uživatelského jména a hesla doplněná alespoň pro tzv. aktivní operace (např. bankovní převod) o potvrzování *jednorázovými hesly* zasílanými pomocí SMS. Jen některé české banky tehdy nabízely i jiné autentizační metody, příp. si uživatel mohl vybrat z více možností.

Bankomaty i internetové bankovnictví jsou současně příkladem toho, že ani dvoufaktorová autentizace *není všespasitelná*, je nutné volit dle potřeb, vývoje techniky i metod útočníků a nezanedbávat ani další složky bezpečnosti. Navíc bankomaty a internetové bankovnictví jsou pochopitelně velmi atraktivním cílem útočníků.

Organizace ENISA (European Union Agency for Cybersecurity) vydala již v roce 2009 varování týkající se bankomatů. Odhadované škody v Evropě tehdy činily kolem 500 milionů EUR (tedy cca 13 mld. CZK) se značným meziročním nárůstem, v současnosti proto budou pravděpodobně ještě mnohem vyšší.[113] V této částce jsou zřejmě i krádeže celých bankomatů a přepadení při/po výběru v automatu.

Jen počet tzv. *skimming incidentů* (z angl. data skimming: sbírání dat) za rok 2008 dosáhl 10 302 případů. Pachatelé používají nebo používali řadu metod (některé jednoduché, další sofistikované): skimmovací zařízení ve vstupu pro kartu, falešné klávesnice nebo speciální fólie na klávesnici, mikrokamery sledující zadávání PIN, infrakamery (termální otisk použitých kláves), dočasné znepřístupnění výdeje bankovek, bezkontaktní čtení RFID čipu karty ad. V roce 2018 ENISA vydala další zprávu, kde popisuje nové techniky, např. jackpotting (bankomat vydá celou zbylou hotovost) s využitím malware, manipulaci s diskem, útok „Man-in-the-Middle" ad.[114]

Některé z výše popsaných metod nyní již nejsou možné nebo alespoň byly velmi ztíženy (např. konstrukčními úpravami bankomatů, vylepšením software). Banky samozřejmě bankomaty pravidelně kontrolují a většinou je nepřetržitě monitorují kamerami, ale právě bankomaty jsou učebnicově názornou ukázkou toho, že nelze stoprocentně zabezpečit zařízení, které nemáte pod *výlučnou* fyzickou kontrolou (nedílnou součástí bezpečnosti je i *fyzická bezpečnost*).

Obdobně v případě internetového bankovnictví se SMSky s jednorázovými *autorizačními kódy* (*jednorázovými hesly*) po řadu let považovaly za vysoký standard zabezpečení (přestože samotné SMS nejsou nijak zvlášť chráněny). A mělo se za to, že útočník by musel napadnout/ovládnout *dvě nezávislá zařízení* (PC s WWW prohlížečem a „klasický" tlačítkový telefon) a komunikace využívá *dva nezávislé komunikační kanály* (šifrované spojení přes Internet a GSM síť telefonů pro zasílání SMS).

Ojedinělé případy podvodných transakcí byly zaznamenány i v ČR, ale téměř vždy šlo o situaci, kdy útočník dokázal neoprávněně změnit číslo telefonu pro zasílání autorizačních SMS.[115] Banky někdy (tzv. pro pohodlí klientů) měly málo průkazný způsob změny tohoto telefonního čísla. Nešlo tedy o *prolomení, ale o obejití* tohoto autentizačního mechanismu. Opět jde o učebnicový příkladu *dilematu pohodlí uživatele* x *bezpečnost*.

V zahraničí však byly zaznamenány i rozsáhlejší a závažnější případy útoků přímo na bankovní transakce. V květnu 2017 společnost O2 Telefónica potvrdila, že v Německu útočníci zkombinovali infekci počítačů klientů malwarem a využití zranitelnosti v protokolu SS7 (který používá telefonní síť).[116]

Navíc, v současnosti velká část uživatelů pro internetové bankovnictví používá tzv. chytré mobily. Ty jsou obecně mnohem snadněji napadnutelné než „klasické" tlačítkové telefony (existuje mnoho doložených výskytů *malware* dokonce i *na oficiálních platformách*

---

[113] *ENISA warns: increase in ATM crime* [online]. 2009-09-07 [cit. 2020-10-25] Dostupné z WWW: <https://www.enisa.europa.eu/news/enisa-news/enisa-warns-increase-in-atm-crime>.

[114] *ATM cash-out attacks* [online]. 2018-09-05 [cit. 2020-10-25] Dostupné z WWW: <https://www.enisa.europa.eu/publications/info-notes/atm-cash-out-attacks>.

[115] Případně se útočníkovi pro změnu podařilo přimět uživatele, aby mu autorizační kód sdělil. V tomto případě tedy šlo o formu *phishingu*.

[116] KHANDELWAL, Swati. *Real-World SS7 Attack — Hackers Are Stealing Money From Bank Accounts* [online]. 2017-05-04 [cit. 2020-10-25] Dostupné z WWW: <https://thehackernews.com/2017/05/ss7-vulnerability-bank-hacking.html>

Google Play pro Android nebo AppStore pro výrobky fy Apple)[117] a současně přestalo platit, že uživatel používá dvě nezávislá zařízení pro komunikaci s bankou. Banky proto již před několika lety začaly postupně zavádět nové autentizační mechanismy a od roku 2019 je k tomu nutí i již zmíněná směrnice PSD2.

1.7.1 Bezpečnostní token

Bezpečnostní token (anglicky *security token*), někdy též *autentizační token*, často zkráceně jen *token* je zpravidla fyzické (hardwarové) zařízení (existuje i tzv. softwarový token), které usnadňuje uživatelům autentizaci, tam kde je vyžadována. Token může zcela nahradit hesla nebo být využit v rámci dvou/vícefaktorové autentizace pro celou řadu různých aplikací. Může na něm být uložen např. soukromý (kryptografický) klíč uživatele nebo vybraná biometrická data (konstrukce čipu a způsob uložení je navržen tak, aby tato data byla maximálně chráněna).

Z hlediska fyzické konstrukce, nabízených funkcí a způsobu připojení lze rozlišit velké množství *variant tokenů*, zde uváděný přehled není vyčerpávající. Token obsahuje čip, který nabízí jednoduché i složitější funkce, některé poskytují i více autentizačních metod. Token dále může obsahovat malý displej, příp. též malou klávesnici např. pro vložení PINu, ev. (jen) tlačítko pro spuštění určitého algoritmu. Některé obsahují též audio funkce (tyto jsou určeny pro uživatele se zrakovým postižením). Jedno ze základních členění je na *nepřipojitelné* a *připojitelné tokeny*, ty se dále dělí na kontaktní a bezkontaktní připojitelné tokeny.

Nepřipojitelný token nelze žádným způsobem připojit k počítači. Musí tedy obsahovat LCD displej, na kterém se zobrazí generovaný autentizační kód (který se mění např. každou minutu) a který se musí ručně přepsat na klávesnici počítače např. při přihlašování do internetového bankovnictví. Zpravidla jde o velmi malá zařízení, které lze připnout na klíčenku, aby ho uživatel měl stále po ruce (viz obrázek níže). Mohou ale mít i podobu připomínající malou kalkulačku. Tento typ se někdy označuje pojmem *autentizační kalkulátor*, na malé klávesnici je nejprve nutno zadat PIN a teprve poté se na displeji zobrazují generované kódy k opsání. Nepřipojitelné tokeny jsou zpravidla velmi levné a patří k nejpoužívanějším prostředkům ve vícefaktorové autentizaci.

Připojitelné tokeny se k počítači připojují buď *kontaktně* (v současnosti zpravidla přes USB rozhraní, ale může to být též např. 3,5 mm jack) nebo *bezkontaktně* (bezdrátově) pomocí Bluetooth, RFID nebo Wi-Fi. Tokeny s rozhraním USB (zkráceně *USB token*) vzhledem i velikostí připomínají všem známé USB flash disky (USB flash paměti).

Použití *audio rozhraní* (nejčastěji 3,5 mm jack) pro připojení bezpečnostního tokenu se může zdát na první pohled zvláštní, ale je poměrně často využíváno u mobilů, aby nebyl blokován zpravidla jediný USB port (micro-USB, nyní USB-C) resp. Lightning port (iPhone), který se kromě připojení různých zařízení používá i k napájení. Avšak iPhone od verze 7 (2016) a řada modelů s Androidem od cca 2019/2020 již audio jack nemá.

---

[117] Varování o nebezpečných aplikacích se objevující víceméně pravidelně. Za všechny jeden článek týkající se *GooglePlay* (tedy mobily s Androidem) a jeden *AppStore* (iPhone), oba z října 2019:
GELINAS, James. *Check your phone for these dangerous apps with 335 million installs* [online]. 2019-10-03 [cit. 2020-10-25] Dostupné z WWW: <https://www.komando.com/technology/infected-apps-google-play-malware/601727/>
BARRETT, Brian. *How 18 Malware Apps Snuck Into Apple's App Store* [online]. 2019-10-25 [cit. 2020-10-25] Dostupné z WWW: <https://www.wired.com/story/apple-app-store-malware-click-fraud/>

Další častou variantou připojitelného tokenu jsou tzv. *čipové karty* (malé plastové karty s čipem, dále mohou obsahovat hologram a jiné prvky znesnadňující padělání). Pro připojení k PC je třeba *čtečka čipových karet* (ta se do PC připojuje opět přes USB rozhraní). Čipové karty jsou zpravidla *multifunkční*: vedle autentizace v informačních systémech mohou sloužit jako identifikační karta zaměstnanců, pro přístup do různých místností, odemykání bran nebo závor pro vjezd, platby za různé služby a mnoho dalších funkcí.

Připojitelné tokeny se používají též k ochraně vybraného software (proti nelegálnímu kopírování, resp. užití). Součástí zakoupeného software je tzv. hardwarový klíč, který obsahuje zašifrovanou informaci o licenci. Při spuštění programu musí být připojený, aby mohl zareagovat na dotaz programu a ověřit vlastníka licence. Těmito tokeny se ale nebudeme dále zabývat.

Jako hlavní *výhodu* tokenů lze uvést, že jde o jednoúčelová zařízení, speciálně navržená pro tyto účely. Tato výhoda je současně i jejich hlavní *nevýhodou*: uživatel nesmí zapomenout nosit token s sebou (byť jde o malá zařízení). Mezi další nevýhody patří možnost ztráty či krádeže, nezanedbatelné není ani ekonomické hledisko. Zařízení jsou sice poměrně levná, ale často jsou jich potřeba tisíce. Navíc náklady na jejich vydávání, vracení a správu obecně mohou často několikanásobně převýšit samotné pořizovací náklady. Proto se v posledních letech místo tokenů často používají *mobilní telefony*, kterými se (z hlediska autentizace) zabývá následující podkapitola.

Obrázek 1.12: Ukázky různých druhů bezpečnostních tokenů

Zdroj: REINHOLD, Arnold. *Several security tokens with a U.S. penny coin for comparison.* Wikimedia Commons [online]. 2009-03-11 [cit. 2020-10-25] Dostupné z WWW: <https://commons.wikimedia.org/wiki/File:SecurityTokens.CryptoCard.agr.jpg>

1.7.2 Užití mobilů k autentizaci

Využití mobilních telefonů k autentizaci se „nabízí“ z několika důvodů. Především, většina lidí nyní nosí mobil s sebou víceméně neustále, odpadá tedy potřeba pořizovat a spravovat speciální zařízení (např. výše popsané bezpečnostní tokeny). Smartphony umožňují využít pro autentizaci nejen jednorázové kódy (více způsoby), ale i další metody.

Jednorázové kódy: lze je zasílat pomocí již dříve popsaných SMS, u smartphonů se nyní často používá tzv. push notifikace nebo lze využít speciální aplikaci (program), který bude jednorázová kódy generovat zvoleným algoritmem.

Jiné metody autentizace využívají toho, že smartphony jsou nyní vybaveny celou řadou specializovaných senzorů a současně lze instalovat různé specializované aplikace. Mezi

nejčastější metody v současnosti patří čtečka otisků nebo rozpoznávání obličeje, lze očekávat rozšíření jiných metod (využití lokalizace, dalších senzorů apod.).

Otisky prstů nebo *rozpoznávání obličeje* se používají k ochraně přístupu do samotného mobilu (k odemčení zařízení) a lze je použít též k autentizaci pro různé weby, resp. aplikace či služby. Z hlediska zařazení těchto metod jde o tzv. *biometriky* (kategorie *něco čím jste*, resp. *inherence factor*).

V případě *mobilů iPhone* od fy Apple, byly senzorem fungujícím jako čtečka prstu vybaveny modely počínaje iPhone 5S (rok 2013), od modelu iPhoneX (2017) ho nahradil senzor rozpoznání obličeje a systém *Face ID* místo předchozího *Touch ID*. Systém Face ID se používá pro odemčení zařízení, pro přístup k chráněným souborům/datům v mobilu a v neposlední řadě pro online platby (App Store, iTunes Store, Apple Books Store) a platby všude tam, kde je podporován mobilní platební systém *Apple Pay*.[118]

V případě *mobilů s Androidem* se vybavenost těmito senzory pochopitelně liší dle výrobce a modelu, ale lze říci, že naprostá většina v současnosti prodávaných mobilů s operačním systémem Android nabízí rozpoznávání obličejů, příp. i čtečku otisků prstu (čtečku měla již Motorola Atrix 4G z roku 2011).[119]

Přestože mobily jako nástroj pro autentizaci nabízí řadu výhod a více metod autentizace, nesmíme zapomínat ani na možné nevýhody. Část je stejná jako u bezpečnostních tokenů: možnost ztráty nebo krádeže, navíc může být vybitá baterie či jiná porucha. Další nevýhody jsou spojeny právě s *víceúčelovostí* smartphonů: riziko instalace malware, phishing, současné využití mobilů pro SMS, e-maily a WWW ad.

1.7.3 Vícefaktorová autentizace v InSIS

Integrovaný studijní a informační systém (*InSIS*) používaný na VŠE obsahuje velké množství důležitých informací vztahujících se ke studiu a používá ho přibližně 20 000 uživatelů (studenti, učitelé a další zaměstnanci). Proto uživatele s tzv. superprávy a uživatelé se zvýšenými právy (např. fakultní správcové InSISu, studijní referentky a řada dalších) musí nyní vícefaktorovou autentizaci využívat *povinně*, ostatním (tj. především učitelé a studenti) je *doporučeno*. Pro autentizaci bylo zvoleno právě využití mobilů.

Jak již bylo v těchto skriptech několikrát uvedeno, zavedení určitého bezpečnostního opatření *je v praxi* vždy *určitým kompromisem* mezi co možná největší bezpečností, pohodlím (nebo alespoň „snesitelností“) pro uživatele a také dostupností pro co nejvíce uživatelů. Právě s ohledem na tato kritéria byly zvoleny i *parametry vícefaktorové autentizace pro InSIS*.

1. Pro autentizaci bylo zvoleno využití *mobilní aplikace*, která poté generuje jednorázová hesla. V *omezené míře* mohou vícefaktorovou autentizaci používat i uživatelé bez chytrého mobilního telefonu, pak je však přihlašování omezeno na konkrétní počítač.
2. Uživatel si může vybrat z několika *doporučených bezplatných aplikací*. Prvním a rozhodujícím faktorem pochopitelně bude dostupnost aplikace pro operační systém mobilu (ev. jiného zařízení), který uživatel vlastní. Všechny doporučené aplikace jsou pro *Android* i *iOS*, Microsoft Authenticator je dostupný i pro Windows Phone, některé aplikace podporují i Apple Watch. Dále budou rozhodovat preference a zvyklosti

[118] *Use Face ID on your iPhone or iPad Pro* [online]. [cit. 2020-10-25] Dostupné z WWW: <https://support.apple.com/en-us/HT208109>

[119] TRIGGS, Robert. *Facial recognition technology explained*. Android Authority [online]. 2019-01-14 [cit. 2020-10-25] Dostupné z WWW: <https://www.androidauthority.com/facial-recognition-technology-explained-800421/>

uživatele a/nebo co která aplikace nabízí, např. někdo již využívá Google Authenticator pro služby Google (nebo alespoň používá tyto služby), další bude preferovat stejnou aplikaci pro InSIS a pro Office 365, aplikace se dále liší možnostmi zálohování atd.

3. Jak rozumný kompromis mezi pohodlím uživatele a co největší bezpečností je též to, že uživatel může určitá zařízení (PC doma, notebook) označit jako tzv. *bezpečná zařízení* (zde se jednorázové heslo vyžaduje jen „občas“). Záleží samozřejmě na zodpovědnosti každého uživatele, která zařízení takto označí, jinak by smysl vícefaktorové autentizace byl značně negován.[120]
4. V případě InSIS se jednorázová hesla používají jen při *přihlašování*, v případě jiných informačních systémů mohou být vícefaktorovou autentizací chráněny i (vybrané) jednotlivé operace.

Základní postup nastavení vícefaktorové autentizace pro InSIS je následující. Uživatel si nainstaluje některou doporučenou mobilní aplikaci. Poté se přihlásí do InSIS (zatím pochopitelně jen uživatelským jménem a heslem). Zde v hlavním menu InSIS, v sekci Nastavení informačního systému si zvolí „Nastavení autentizace pomocí jednorázových hesel (OTP)“. Poté je mu vygenerován QR. Je velmi doporučeno si tento QR kód vytisknout a uložit na bezpečné místo (záloha pro případ ztráty mobilu apod.). Po naskenování QR kódu kamerou mobilu z nainstalované aplikace se provede tzv párování InSIS a mobilu, po jejím dokončení se aktivuje využívání jednorázových hesel. Podrobnější návod a řešení možných problémů je na stránkách Centra informatiky VŠE.[121]

1.8 Zadání a otázky

1.8.1 Zadání výpočtů

1. Generujete počáteční hesla ve tvaru `kvk-kvk-kvk-kvk`, kde *k* zastupuje souhlásky (konsonanty) a *v* samohlásky (vokály). Kolik různých kombinací hesel může vzniknout? Jaká je entropie takto generovaných hesel?
2. Doplnění k předchozímu zadání – kolik slabik musíte vygenerovat, pokud chcete u hesla entropii 56?
3. Systém Brutalis od fy Sagita je schopen otestovat za vteřinu 98 285 hesel[122] ve formátu sha512crypt (5000 iterací). Generujete hesla pomocí slabik (viz zadání 1), kolik slabik musíte použít, aby útočník uhádl v průměru jedno heslo za týden?
4. Pro 10 000 uživatelů generujete náhodné heslo ze 7 malých písmen anglické abecedy. Jaká je pravděpodobnost, že dva uživatelé budou mít stejné heslo?
5. Jaká je pravděpodobnost, že v systému se 120 uživateli budou mít dva z nich stejnou sůl, pokud sůl bude mít délku 12 bitů (viz algoritmus DEScrypt v kapitole 1.5.9).

---

[120] Součástí dále zmíněného návodu jsou i názorné příklady, která zařízení lze a které naopak nelze považovat za bezpečná zařízení (např. PC na učebnách, mobil půjčovaný dětem atd.).

[121] *Vícefaktorová autentizace [v InSIS]* [online]. [cit. 2020-10-25] Dostupné z WWW: <https://ci.vse.cz/sluzby/dalsi/insis/vicefaktorova-autentizace/>

[122] GOSNEY, Jeremi M. *World's First 8x R9 290X oclHashcat Benchmark* [online]. [cit. 2020-10-25] Dostupné z WWW: <https://gist.github.com/epixoip/8171031>

1.8.2 Otázky

1. V kapitole 1.2.4 je uvedeno, že pro uživatele není efektivní dodržovat bezpečnostní pravidla, neboť jejich náklady budou vyšší než potencionální přínosy. Jaké vidíte cesty pro nápravu?
2. Náklady v řádu tisíců korun na získání hesla konkrétního uživatele pomocí keyloggeru jsou poměrně nízké. Proč se tedy téměř každý den nevyskytují v novinách titulky, že někomu bylo zcizeno heslo?
3. Velmi častou námitkou proti používání správců hesel je strach z krádeže souboru s hesly. Předpokládejte, že uživatel je schopen si zapamatovat 5 složitých hesel. První čtyři jsou jedinečná (heslo k účtu v práci, heslo do banky, heslo k poště u Google a heslo k účtu na osobním počítači). Dále přistupuje k 25 různým systémům. Porovnejte bezpečnost následujících dvou situací:
   1. varianta – páté heslo používá ve všech 25 systémech.
   2. varianta – každý z 25 systémů má náhodně vygenerované heslo uložené ve správci hesel. Uživatel si pamatuje heslo do správce hesel.

   Jaké jsou hrozby u jednotlivých variant? Jak jsou pravděpodobné jednotlivé hrozby v případě následujících třech útočníků:
   - anonymní hacker z Internetu, jehož motivací je zisk
   - soused, který uživatele nemá rád,
   - někdo, s kým uživatel sdílí stejné prostory (spolupracovník z kanceláře, žárlivý partner/partnerka, spolubydlící na koleji, …).
4. Článek „Suggesting you shouldn't digitise your sexual exploits isn't "victim blaming", it's common sense"[123] popisuje případ, kdy si nějaký americký učitel natočil video se sexuálními radovánkami v posteli a poté ho umístil na svůj Dropbox. Někdo se k jeho účtu dostal (velmi slabé heslo), video odeslal jeho žákům a následně vedení školy. Reakce vedení školy byla rychlá – učitele okamžitě propustili za narušování mravní výchovy. Učitel je v tomto případě jednoznačně obětí nějakého zloděje/hackera/zlomyslného člověka/ … (dosaďte si vlastní vhodné označení). A nyní přichází morální problém: pokud budeme upozorňovat na to, že si zvolil slabé heslo, neděláme z něho spoluviníka? Neříkáme tvrzením „kdyby si zvolil správně heslo, tak …" či „kdyby video nedával na Dropbox, tak …" že si za své problémy může on sám?

[123] HUNT, Troy. *Suggesting you shouldn't digitise your sexual exploits isn't "victim blaming", it's common sense* [online]. 2016-02-19 [cit. 2020-10-25] Dostupné z WWW: <https://www.troyhunt.com/suggesting-you-shouldnt-digitise-your/>.

2 Protokol SSH a sada programů PuTTY

2.1 Historie vzdáleného přístupu

Vývoj síťových služeb, ve spojení s internetem umožnily vznik serverů, které nerušeně běží ve vzdálené serverovně. Pro *vzdálenou správu serverů* se zpočátku používali klienti na bázi **protokolu Telnet**. Protokol byl prvně využíván kolem roku 1969 pro komunikaci mezi počítači *v uzavřené síti*.[124] Protokol *nebyl zabezpečen*, bylo možné odposlechnout heslo, které zadával uživatel při přihlašování k serveru.[125]

V reakci na odchytávání hesel na počítačové síti univerzity v Helsinkách *Tatu Ylönen* navrhl a naprogramoval v roce 1995 první verzi *protokolu a programu* **SSH** jako *šifrované náhrady* za protokoly *telnet, rlogin* a *rsh* (remote-shell).

Protokol se vyvíjel dále, a tak v roce 1999 vznikla verze **SSH-2**. Roku 2006 byla revidovaná verze protokolu SSH-2 navržena jako skupina *internetových standardů*:

- **RFC 4250** The Secure Shell (SSH) Protocol Assigned Numbers
- **RFC 4251** The secure shell (SSH) protocol architecture
- **RFC 4252** The secure shell (SSH) authentication protocol.
- **RFC 4253** The secure shell (SSH) transport layer protocol.
- **RFC 4254** The secure shell (SSH) connection protocol.
- **RFC 4255** Using DNS to securely publish secure shell (SSH) key fingerprints.
- **RFC 4256** Generic Message Exchange Authentication for the Secure Shell Protocol (SSH).

První verze SSH klienta i serveru byly nejdříve uveřejněny včetně zdrojových kódů. Jen několik měsíců po první verzi, na konci roku 1995, se používal již v 50 zemích. Pro velký zájem o program a pro zajištění technické podpory, *Ylönen* v prosinci 1995 založil společnost SSH Communications Security a vyšší verze programu byly následně distribuovány jako *komerční produkt*. Proto později vznikly jiné volně dostupné implementace.

OpenSSH je nejrozšířenější bezplatná implementace protokolu SSH, která vznikla v roce 1999 a je součástí operačního systému OpenBSD. Existují porty do Linux, Solarisu, Windows a mnoha dalších operačních systémů, čímž se OpenSSH zpřístupňuje širokým vrstvám uživatelů včetně zdrojových kódů.

SSH je zkratka ze spojení *Secure Shell*, které evokuje existenci příkazového řádku. SSH ale neposkytuje příkazový řádek, jedná se o *síťový protokol*, který často zpřístupňuje příkazový řádek na vzdáleném serveru. Používá se ale i pro přenos souborů, přesměrování portů a další.

Protokol SSH není vázán na konkrétní šifry a další kryptografické algoritmy. Postupně se rozšiřuje o novější a díky tomu nebylo nutné jej příliš měnit přes to, že je z roku 1999 (verze SSH-2), tedy více než 20 let starý.

Varování:
SSH-1 obsahuje bezpečnostní chyby již v návrhu. NEPOUŽÍVEJTE.

[124] KABELOVÁ, Alena a Libor DOSTÁLEK. *Velký průvodce protokoly TCP/IP a systémem DNS*. 5. aktualiz. vyd. Brno: Computer Press, 2008. ISBN 978-80-251-2236-5

[125] A uvažujme také, že v té době bylo primárním cílem zajistit funkci a provoz systému. Bezpečnost byla v té době víceméně opomíjena. Navíc ani neexistovaly vhodné kryptografické nástroje.

2.2 Použité kryptografické principy

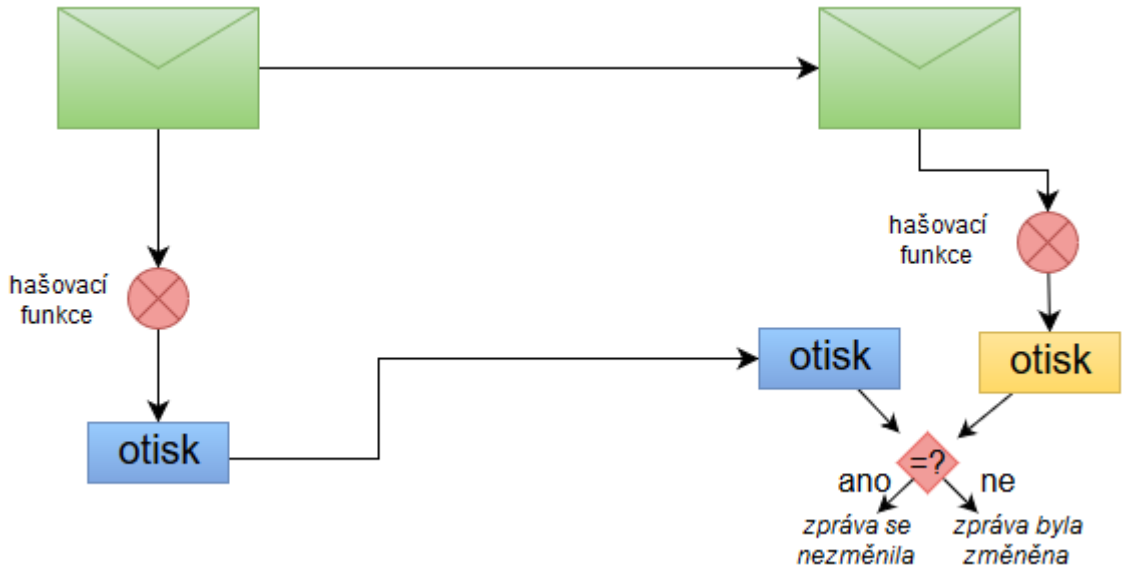
Nejdříve si zopakujeme některé *kryptografické principy* použité v protokolu SSH, konkrétně autentizaci pomocí soukromého a veřejného klíče, zajištění integrity a autentizace přenášených zpráv a protokol *Diffie-Hellman* pro domlouvání klíčů přes nezabezpečený kanál.

2.2.1 Kód pro ověření zprávy (Message authentication code)

Při posílání zpráv je potřeba zajistit, že nějaký útočník cestou nezmění jejich obsah (zajištění integrity) či místo zprávy nepošle jinou zprávu (zajištění autenticity).

Příjemce zprávy potřebuje poznat, zda ji cestou někdo nezměnil. První, co Vás asi napadne, je vytvoření otisku pomocí *kryptografické hašovací funkce*. Odesílatel ke zprávě spočítá otisk a zprávu včetně otisku pošle příjemci. Příjemce spočítá z přijaté zprávy druhý otisk a poté oba otisky porovná. Pokud se liší, tak byla zpráva změněna.

Obrázek 2.1: Princip využití hašovací funkce pro kontrolu integrity zprávy

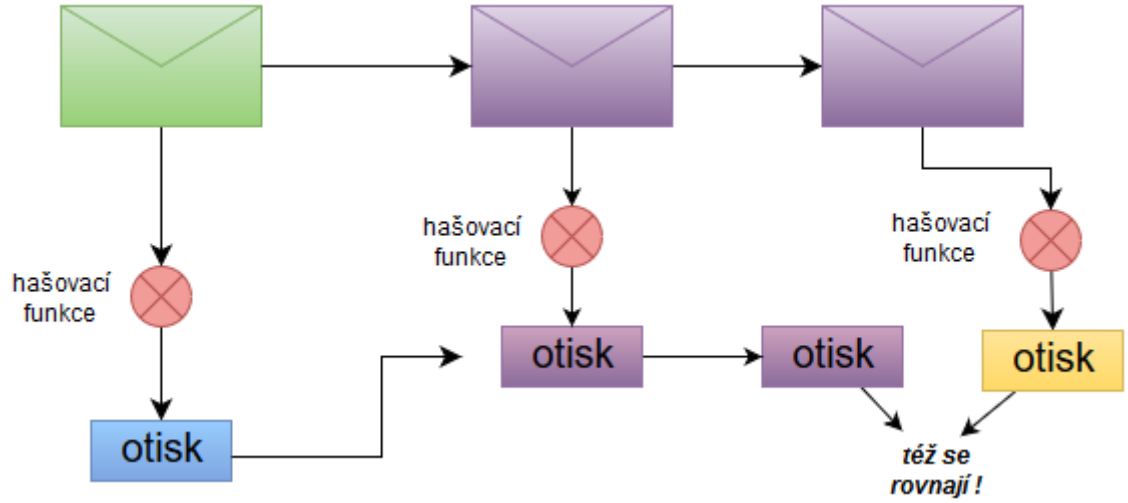


Zdroj: vlastní zpracování

Toto jednoduché použití hašovací funkce má však *vážný bezpečnostní problém*. Pokud útočník odchytí zprávu, tak ji může změnit, spočítat svůj otisk a poté upravenou zprávu se svým otiskem poslat příjemci. Tomu budou při kontrole otisky odpovídat přesto, že *zpráva je podvržena!* Tento bezpečnostní problém ilustruje Obrázek 2.2 na následující stránce.

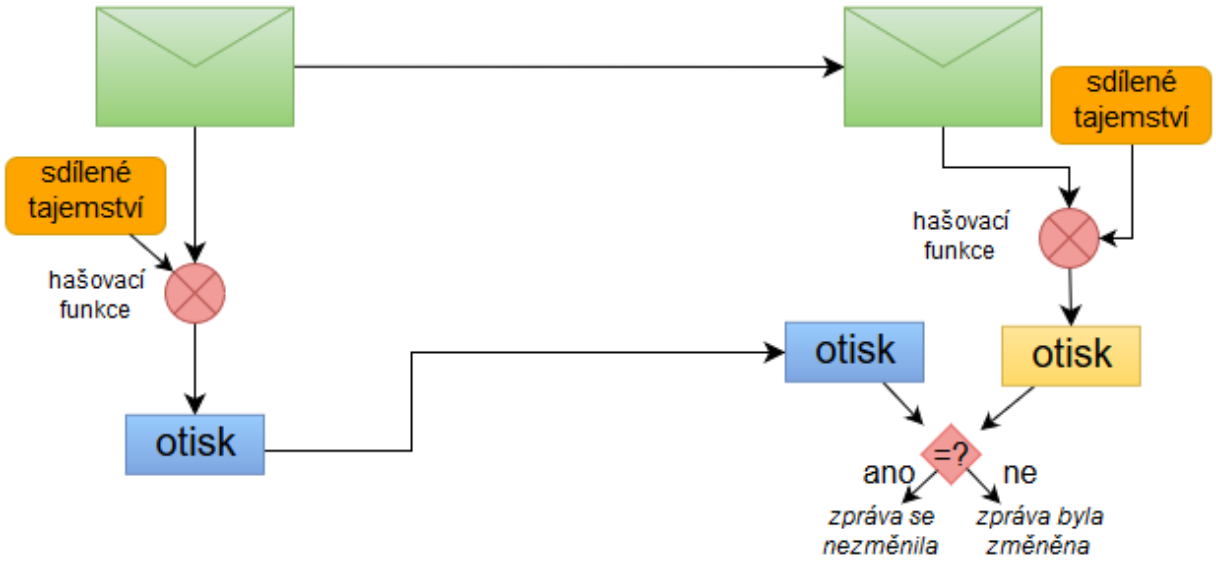
Existuje několik řešení. Jedním z nich je použití sdíleného tajemství (sdílený klíč, sdílené heslo) pro výpočet **kódu pro ověření zprávy** (**MAC**, **Message Authentication Code**). Jeho princip ilustruje Obrázek 2.3, rovněž na následující stránce.

Obrázek 2.2: Problém hašovací funkce: otisk se rovná také u podvržené zprávy



Zdroj: vlastní zpracování

Obrázek 2.3: Princip využití MAC pro kontrolu integrity zprávy



Zdroj: vlastní zpracování

Často se používá algoritmus **HMAC** (**keyed-Hash Message Authentication Code**), který počítá kód pomocí následujícího *zjednodušeného vzorce* (chybí zarovnání délky klíče i operace xor s klíčem a předdefinovanými konstantami):

```
hash(key+hash(key+message))
```

Poznámka:
Používá se výše uvedený *zdvojený výpočet*, neboť u varianty jednoduchého výpočtu

hash(key+message)

je hrozbou tzv. *length extension* attack[126] pro následující hašovací algoritmy:[127] MD5, SHA-1 a SHA-2. Druhou variantu jednoduchého výpočtu:

hash(message+key)

by zase mohl *zneužít odesílatel*, pokud by našel dvě kolidující zprávy (dvě zprávy se stejným výsledným hašem).

Dokud útočník nezjistí sdílené tajemství, tak jakoukoliv útočníkovu úpravu příjemce detekuje. **MAC** vedle *integrity* zajišťuje *i autenticitu* – zprávu mohl odeslat pouze ten, kdo zná sdílené tajemství. Uvedené schéma *není* odolné vůči **útoku opakováním** (*replay attack*) – útočník může zprávu zachytit a poslat ji ve vhodný okamžik znovu. Jak si příjemce s odesílatelem domluví sdílené tajemství? Jedna z variant bude popsána v následující části.

2.2.2 Domlouvání klíče přes nezabezpečený kanál (Key exchange)

Sdílené tajemství (sdílený klíč, tajný klíč, sdílené heslo) se používá pro *ověření integrity* a *autenticity zpráv* a pro symetrické šifrování přenášených zpráv (*zajištění důvěrnosti*).
Na nezabezpečeném přenosovém kanále si lze domluvit sdílený klíč např. pomocí **algoritmu Diffie-Hellman** z roku 1976:

1. Jedna strana (obvykle server) vygeneruje či vybere z připraveného seznamu[128] velké prvočíslo (***P***) a základ logaritmu (***g***) a oboje pošle druhé straně.
2. Každá strana si nezávisle určí prvočíslo (server k_1, klient k_2), které zůstává jejím tajemstvím. Toto číslo je využito jako „soukromý klíč“ pro další výpočty (neplést se soukromým klíčem pro autentizaci).
3. Každá strana si spočítá „veřejný klíč“ dle vzorce $K = g^k \bmod P$, tím vzniknou klíče (K_1; K_2).
4. Obě strany si vymění klíče (K_1; K_2).
5. Za použití svého „soukromého“ klíče a „veřejného“ klíče si protistrany spočítají sdílený soukromý klíč (***j***), který ačkoliv je vypočítán nezávisle každou stranou (serverem a klientem) z opačných soukromých a veřejných klíčů, je pro obě strany totožný.

$$K_2^{k1} \bmod P = j = K_1^{k2} \bmod P$$

[126] Podrobný popis útoku je v řadě publikací. Základní informace viz článek na anglické Wikipedii: *Wikipedie: Length extension attack* [online]. 2020 [cit. 2020-10-25]. Dostupné z WWW: <https://en.wikipedia.org/w/index.php?title=Length_extension_attack>

[127] Útok se týká hašovacích funkcí vytvořených pomocí *Merkleovy-Damgårdovy konstrukce*. U SHA-3 a také u některých dalších hašovacích funkcích není zdvojený výpočet nutný.

[128] Vygenerování a otestování velkého prvočísla (1024 a více bitů) trvá dlouho (vteřiny až minuty). Proto jsou prvočísla často součástí distribuce serveru, popř. se generují při instalaci programu. Několik konkrétních prvočísel bylo definováno i ve standardech – „Diffie Hellman group 1“ (prvočíslo 768 bitů), „Diffie Hellman group 2“ (1024 bitů) až „Diffie Hellman group 14“ (2048 bitů). Poté stačí poslat protistraně číslo skupiny.

Evropská organizace ENISA (*The European Union Agency for Cybersecurity*) doporučuje klíče o délce **3072 bitů** a větší (stejně velké musí být i prvočíslo).[129]

Používá se též algoritmus **Diffie-Hellman nad eliptickými křivkami**, kde jsou klíče o řád kratší. V protokolu SSL/TLS se pro domlouvání klíčů používá také *šifra RSA*.

Zde popsaná verze algoritmu Diffie-Hellman není odolná vůči útoku *„muž uprostřed" (Man in the Middle).* V SSH je algoritmus zkombinován s autentizací serveru.

2.2.3 Šifrování s veřejným a soukromým klíčem

Asymetrická šifra pracuje se dvěma klíči ***S*** a ***V*** – první klíč je **soukromý** (někdy se označuje též jako **privátní**), druhý je **veřejný**. Názvy klíčů odkazují na obvyklé použití – soukromý klíč by měl majitel bezpečně uschovat pro sebe, veřejný klíč může jeho vlastník sdělit komukoliv (i případnému útočníkovi).

Klíče jsou na sobě *závislé – vytváří dvojici*. Z veřejného klíče nesmí být možné spočítat soukromý klíč v nějakém reálném čase. Dvojici klíčů lze použít dvěma rozdílnými způsoby: pro šifrování zpráv a pro digitální podepisování.

Při **šifrování** se použije veřejný klíč ***V*** *příjemce zprávy*. Zprávu může dešifrovat *pouze vlastník soukromého klíče* ***S***, tedy jeden *konkrétní příjemce zprávy*. Pokud chceme stejnou zprávu poslat více příjemcům, je nutné ji pro každého příjemce šifrovat zvlášť, vždy veřejným klíčem každého jednotlivého příjemce zprávy.[130]

Při **digitálním podepisování** podepisující (autor) spočítá haš ze zprávy a ten následně zašifruje *pomocí svého soukromého klíče* ***S***. Každý (libovolný) příjemce zprávy poté může ověřit podpis – spočítá haš ze zprávy a ten následně porovná s hašem, který získá po dešifrování podpisu *pomocí veřejného klíče* ***V*** *podepisujícího*. Pokud jsou stejné, tak ověří, že se zpráva cestou *nezměnila* a současně ověří (*autentizuje*) autora podpisu (vlastníka odpovídajícího soukromého klíče).

Poznámka:
U algoritmu RSA *lze stejnou dvojici* klíčů použít pro oba účely, ale *nedoporučuje se to*. U ostatních algoritmů se pro každý účel generuje *samostatná dvojice klíčů s odlišnými charakteristikami*. Dvojici klíčů pro šifrování nelze zaměnit za dvojici klíčů pro digitální podpis.

[129] *ENISA: Algorithms, key size and parameters report 2014* [online]. 2014-11-21 [cit. 2020-10-25]. Dostupné z WWW: <https://www.enisa.europa.eu/publications/algorithms-key-size-and-parameters-report-2014>.

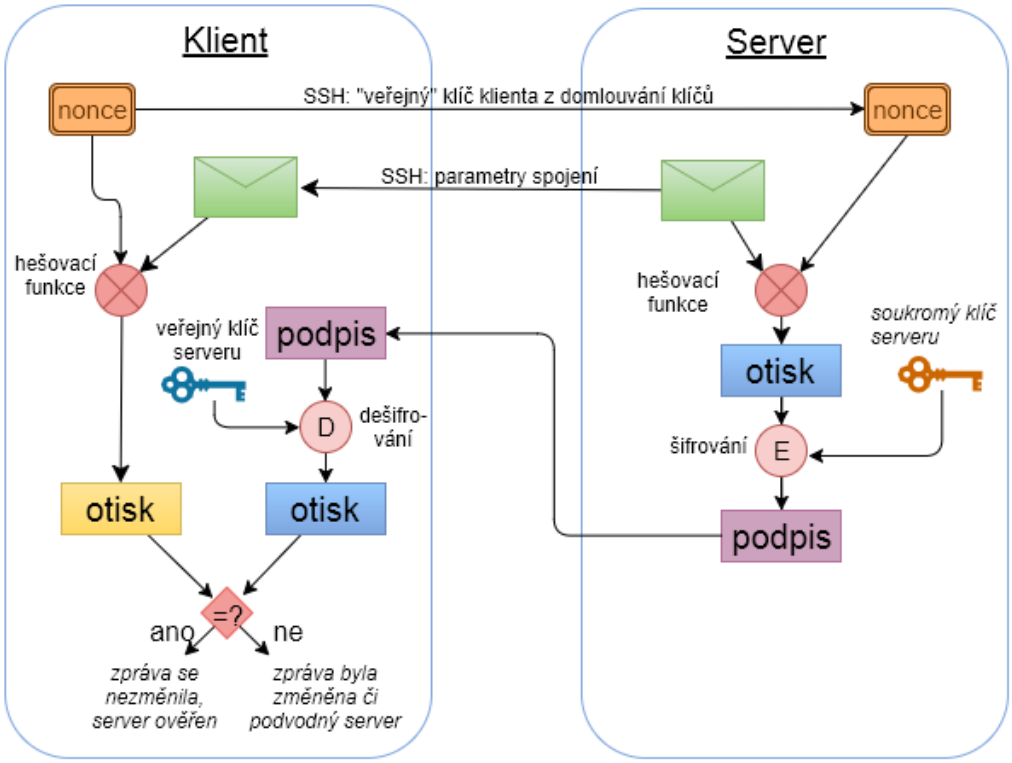
[130] V reálných systémech se při šifrování stejné zprávy pro více příjemců postupuje trošku jinak: z více ryze praktických důvodů se kombinují *symetrické* a *asymetrické* (s veřejným a soukromým klíčem) šifrovací algoritmy. Proto se tyto systémy někdy označují jako *hybridní*. Náhodně se vygeneruje symetrický klíč, který se použije pro zašifrování zprávy. Tento symetrický klíč je poté chráněn tak, že je zašifrován (pro každého příjemce zvlášť, tedy vícekrát) s využitím asymetrické kryptografie. Na základním výše uvedeném principu (nutnost použít tolik veřejných klíčů, kolik je různých příjemců), se však pochopitelně nic nemění. Konkrétní příklad implementace je uveden dále v kapitole o *OpenPGP* a programu *GnuPG*.

2.2.4 Autentizace serveru

Před zahájením autentizace serveru musí mít server vygenerovánu svoji dvojici veřejný a soukromý klíč a *klient musí znát* veřejný klíč serveru. Autentizace má poté následující kroky:

1. Klient vygeneruje **náhodný řetězec** (**nonce**) a pošle ho serveru. Bez tohoto náhodného řetězce by útočník mohl provést útok opakováním. V případě SSH se jako náhodný řetězec použije „veřejný“ klíč klienta K_2 z protokolu Diffie-Hellman (viz předchozí část).
2. Server pomocí hašovací funkce vypočítá *otisk z nonce* a ze zprávy s dohodnutými parametry komunikace (jaké se budou používat algoritmy).
3. Server *otisk zašifruje* pomocí svého soukromého klíče a tím vytvoří *digitální podpis*.
4. Server zprávu a digitální podpis pošle klientovi.
5. Klient pomocí hašovací funkce vypočítá *otisk z nonce* a *ze zprávy*.
6. Klient pomocí veřejného klíče serveru *dešifruje digitální podpis* a získá z něho otisk vypočítaný serverem.
7. *Oba otisky porovná* – pokud se rovnají, tak se ověřila integrita zprávy, autenticita zprávy a autenticita serveru.

Obrázek 2.4: Základní schéma autentizace serveru



Zdroj: vlastní zpracování

Poznámka:
Uvedený průběh autentizace předpokládá, že klient má předem veřejný klíč serveru.
Jak klient ověří, že veřejný klíč patří správnému serveru?

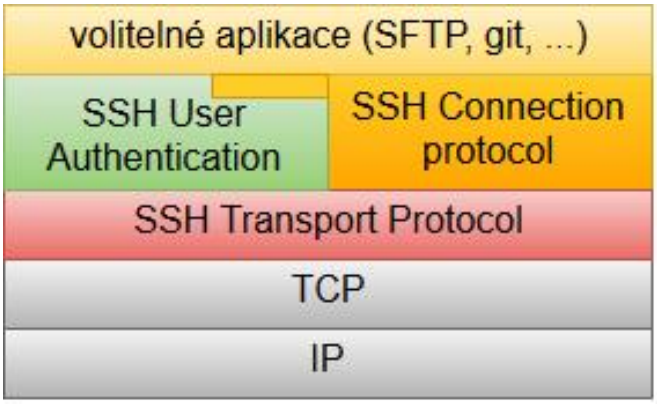
Odpověď najdete v následující kapitole.

2.3 Protokol SSH

2.3.1 Architektura protokolu SSH

SSH používá **klient-server architekturu** – uživatel se připojuje prostřednictvím klienta obvykle ze svého osobního počítače na server. Server je proces, který poslouchá obvykle na **TCP portu 22**. Uživatel musí v klientovi zadat **adresu serveru** (IP adresu či doménové jméno) a volitelně číslo portu.

Obrázek 2.5: Schéma architektury protokolu SSH



Zdroj: vlastní zpracování

Protokol SSH se skládá *ze tří základních protokolů*, nad kterými mohou být *další aplikace*.

- transportní protokol/vrstva (SSH Transport Layer Protocol)
- autentizace uživatele (SSH Authentication Protocol)
- protokol spojení (SSH Connection protocol)

2.3.2 Transportní vrstva – vytvoření spojení

Cílem transportního protokolu je *vytvořit zašifrované (důvěrné) spojení* se zajištěním *integrity* a *autenticity zpráv*. Při vytváření SSH spojení se domlouvají používané algoritmy a klíče sezení, ověřuje se autenticita serveru. Poté se vytvoří transportní vrstva, která přenáší zprávy z vyšších vrstev.

Při vytváření spojení se domlouvají následující algoritmy:

Tabulka 2.1: Algoritmy domlouvané při vytváření SSH spojení

Druh algoritmu	**Charakteristika algoritmu**
kex_algorithms	algoritmy pro domlouvání klíčů
server_host_key_algorithms	algoritmy soukromého/veřejného klíče (server může mít více klíčů pro svoji autentizaci)
encryption_algorithms client_to_server	symetrické šifry pro přenos od klienta k serveru
encryption_algorithms server_to_client	symetrické šifry pro přenos od serveru ke klientovi
mac_algorithms client_to_server	algoritmy pro výpočet MAC při přenosu klient server
mac_algorithms server_to_client	algoritmy pro výpočet MAC při přenosu serveru klient

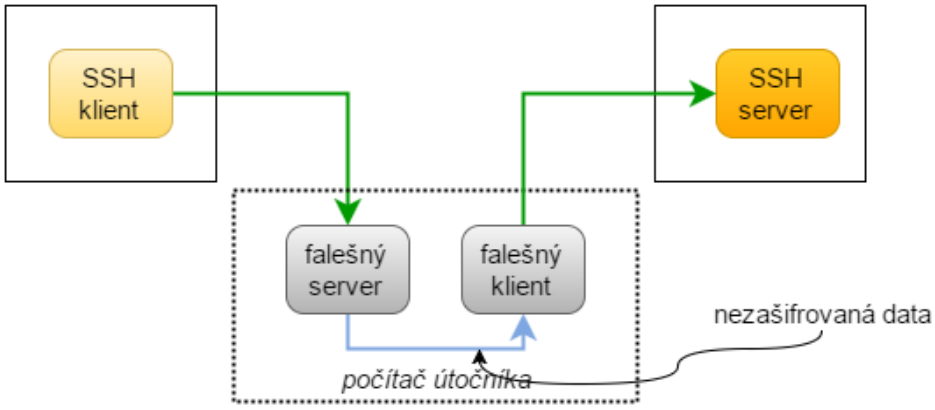
Druh algoritmu	Charakteristika algoritmu
compression_algorithms client_to_server	kompresní algoritmy od klienta k serveru
compression_algorithms server_to_client	kompresní algoritmy od serveru ke klientovi

Zdroj: vlastní zpracování

Z tabulky je zřejmé, že v každém směru přenosu může být použito jiné šifrování. Doporučuje se však používat stejné algoritmy v obou směrech. Podporované algoritmy se mohou lišit v jednotlivých verzích programů. Navíc uživatel (správce serveru) může omezit algoritmy v konkrétní instalaci. Občas se stane, že se starý klient nepřipojí na nový server.

Při vytváření SSH spojení se o*věřuje autenticita serveru* – ověřuje se jeho veřejný klíč, a že k němu má server soukromý klíč. Postup je popsán v předchozí kapitole, kterou jsme ukončili konstatováním slabiny – útočník může podvrhnout svoje klíče pomocí útoku Man-in-the-Middle.

Obrázek 2.6: Schéma útoku Man-in-the-Middle



Zdroj: vlastní zpracování

SSH *částečně spoléhá* na uživatelovo ověření klíče serveru. Při prvním připojení na konkrétní server se uživateli zobrazí *otisk veřejného klíče serveru* (**fingerprint**) a uživatel by ho měl ověřit u správce serveru. Je na klientovi, zda bude se serverem dále komunikovat. Pokud uživatel potvrdí, že se jedná o důvěřuje zobrazenému otisku, tak si klient uloží jméno a klíč serveru do souboru (v Unixu soubor `~/.ssh/known_hosts`) či ve Windows do *registrů*. Při druhém a dalším připojení ke stejnému serveru se kontroluje, zda vrací stejný klíč, který má již klient pro jméno serveru uložen.

Při odlišném klíči unixový klient spojení obvykle odmítne. Pokud chce uživatel používat nový klíč (např. po přegenerování klíče na serveru), tak musí nejdříve smazat starý klíč v `~/.ssh/known_hosts`. Dále popsaný program `putty.exe` zobrazí upozornění a nechá na rozhodnutí uživatele, zda akceptuje nový klíč.

Veřejný klíč SSH serveru může být *uložen v DNS* (předpokladem je, že DNS záznamy budou zabezpečeny pomocí DNSSEC). – klienti poté ověří klíč vůči tomuto záznamu a uživatele se neptají. PuTTY či WinSCP (zatím) ověření vůči DNS nepodporují. RFC 6187 popisuje *ověření pomocí certifikátů* – tato možnost je v implementacích protokolu málo rozšířena.

2.3.3 Transportní vrstva – vlastní přenos

Po dohodnutí algoritmů, klíčů a po autentizaci serveru se všechny přenášené údaje šifrují pomocí symetrické šifry a zabezpečnují pomocí **kódu pro ověření zprávy (MAC, Message Authentication Code).**

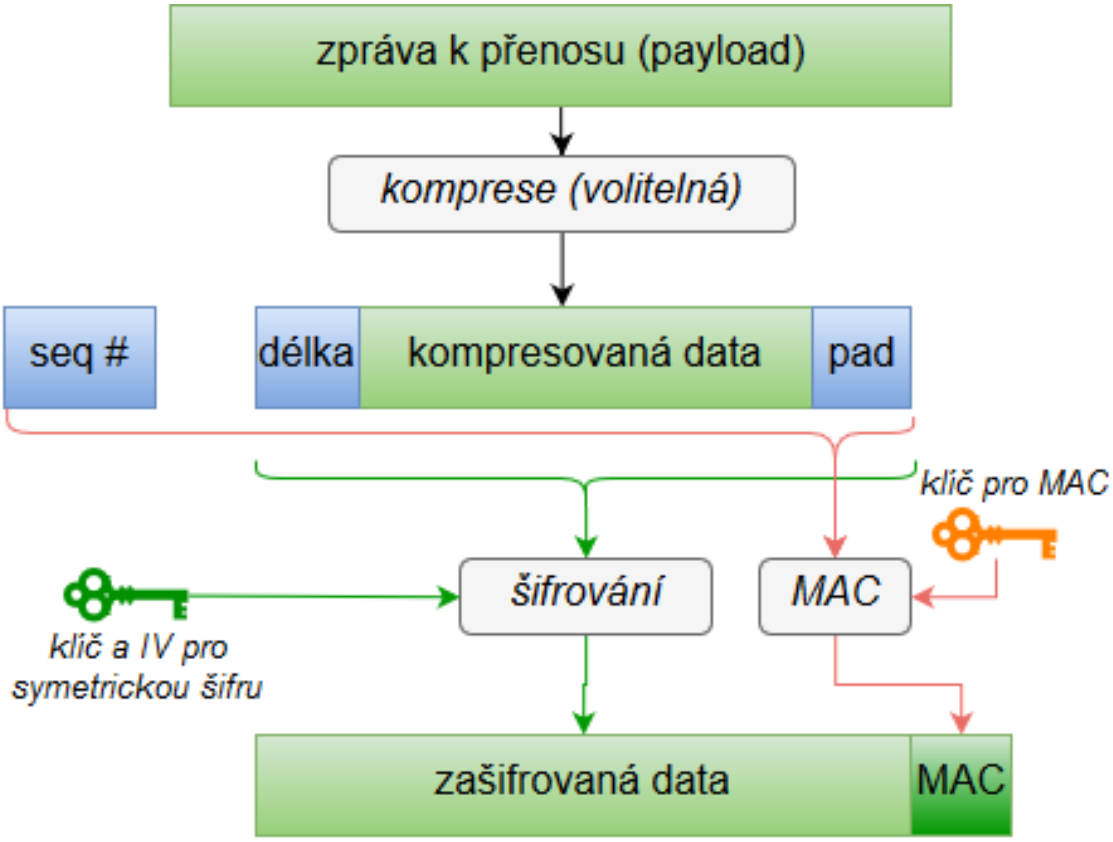
Pro každý směr přenosu (od klienta k serveru a od serveru ke klientovi) se nezávisle **počítají odeslané zprávy** – je to ochrana *proti útoku opakováním*. Pořadové číslo zprávy se na druhou stranu nepřenáší.

Zpráva (používá se pojem **payload** – doslova *užitečný náklad*) se nejdříve volitelně zkompresuje a poté se k ní připojí náhodná vycpávka (*padding*) o minimální délce 4 B.

Následně se počítá **MAC** – do něho se zahrne i pořadové číslo odesílané zprávy. Následně se data zašifrují. Na druhou stranu se pošlou zašifrovaná data a MAC. Příjemce dešifruje data a následně spočítá vlastní MAC. Po větším množství přenesených dat nebo po určité době se obě strany mohou domluvit na nových klíčích.

Po vytvoření bezpečného spojení následuje *autentizace uživatele*, která je popsána v následující podkapitole.

Obrázek 2.7: Schéma přenosu na transportní vrstvě



Zdroj: vlastní zpracování (pad = padding, tedy náhodná vycpávka)
IV = inicializační vektor, zda a jak je použit záleží na módu blokové šifry, viz pozn. 49.

2.3.4 Autentizace uživatele

Po navázání šifrované komunikace a vytvoření bezpečného spojení začne autentizace uživatele. Nejdříve server pošle klientovi podporované typy autentizace. Klient se poté postupně zkouší autentizovat pomocí uvedených typů (nemusí použít všechny možnosti). Následují standardizované typy autentizace v obvyklém pořadí ověřování.

1. **gssapi** – ověření pomocí Kerberos tiketů.
2. **hostbased** – server důvěřuje autentizaci uživatele na počítači, ze kterého se hlásí. Například u uživatele *bob* na serveru B může být napsáno, že pokud se bude hlásit uživatelka *alice* ze serveru A, tak se nemá vyžadovat heslo. Principiálně nebezpečné, neboť nelze zajistit, aby se uživatel *malory* při přihlašování ze serveru A nevydával za uživatel alice. *Obvykle zakázáno.*
3. **publickey** – autentizace pomocí veřejného a soukromého klíče uživatele, viz dále.
4. **keyboard-interactive (challenge-response)** – server postupně posílá dotazy klientovi a uživatel na ně odpovídá, dokud server nakonec nerozhodne, zda se uživatel autentizoval či ne. Používá se např. u jednorázových hesel či při vícefaktorové autentizaci. Lze použít i pro přihlašování pomocí běžných hesel (nedoporučuje se).
5. **password** – autentizace pomocí hesla. Klient zjistí od uživatele heslo a poté ho nezašifrované pošle ve vytvořeném bezpečném spojení na server, kde se ověří

V RFC 4252 je ještě definována metoda *none* bez jakéhokoliv ověření. Zásadně se nedoporučuje používat.

Autentizace publickey (autentizace uživatele pomocí soukromého/veřejného klíče) je v principu podobná autentizaci serveru:

1. Uživatel nejdříve vygeneruje svoji dvojici soukromý/veřejný klíč. Soukromý klíč je obvykle chráněn pomocí hesla.
2. Uživatel svůj veřejný klíč umístí na server. Způsob není určen – může sám po přihlášení pomocí hesla či prostřednictvím správce či webové aplikace, ke které se přihlásí (to používá např. *github*).
3. Klient vytvoří autentizační zprávu (obsahuje uživatelské jméno, veřejný klíč a několik dalších údajů) a tu digitálně podepíše pomocí uživatelova soukromého klíče.[131]
4. Na uživatelské jméno se klient zeptá uživatele, zjistí ho z konfiguračního souboru či použije jméno, pod kterým je uživatel přihlášen na počítači.
5. Server vezme odpovídající uložený veřejný klíč uživatele a ověří obdržený digitální podpis. Pokud je v pořádku, tak se uživatel správně autentizoval. Pokud ne, tak vyzve klienta k dalšímu pokusu o autentizaci – např. pomocí jiné dvojice klíčů či pomocí hesla.

Proti autentizaci serveru zde *není potřeba* posílat ze serveru náhodná data (nonce) – je již vytvořen bezpečný kanál a nehrozí útok opakováním.

[131] Klient může zvolit i delší výměnu – nejdříve se serveru zeptá, zda může použít konkrétní veřejný klíče. Pokud server odpoví ano, tak teprve poté odešle podepsanou zprávu. Tato delší cesta ušetří na straně klienta čas procesoru nutný pro vytvoření digitálního podpisu v případě odmítnutého klíče. Též umožňuje, aby se klient ptal na heslo k soukromému klíči pouze v případě, kdy odpovídající veřejný klíč server schválí.

Veřejný klíč může být též ve *formě certifikátu* – podepsaný osobní certifikační autoritou uživatele. Uživatel poté na server neumístí všechny své veřejné klíče, ale klíč své certifikační autority. Podpora certifikátů není v implementacích příliš rozšířená.

Na cvičeních budeme používat autentizaci pomocí hesla (počáteční přihlášení k serveru, později zakázáno nebo alespoň omezeno na školní síť) a pomocí klíče (mnohem bezpečnější, přitom současně komfortnější – viz popis PuTTY dále).

2.3.5 Protokol spojení – kanály, základní aplikace

Protokol SSH umožňuje zabezpečit *téměř jakoukoliv obousměrnou komunikaci* mezi dvěma body v síti. Lze například bezpečně kopírovat soubory (*scp* a *sftp*), synchronizovat adresáře (*rsync+ssh*), tunelovat celou komunikaci po vybraný port, nebo zabezpečit nešifrovaný protokol jako je přístup k MySQL databázi nebo X-Window protokol. SSH se často používá v systémech na správu verzí jako je *Git* či *Subversion*.

Variabilitu SSH umožňují **nezávislé kanály**, které se vytvářejí po autentizaci uživatele. Kanálů může být více, kanál může vytvořit libovolná strana komunikace, lze je v průběhu spojení přidávat a odebírat. Při ukončení všech kanálů končí i SSH spojení.

Tabulka 2.2: Typy kanálů v SSH a jejich užití

Typ kanálu	**Užití kanálu**
session	Interaktivní kanál se spuštěním programu na vzdálené straně. Může se použít pro *spuštění programu na serveru* a zobrazení výsledků. Tento kanál používá i pro *přenos souborů* – na serveru se spustí modul *sftp* a lokální klient si s ním vyměňuje soubory a informace ze souborového systému (výpis adresáře, …). Nejčastěji se používá *pro emulaci terminálu* a spuštění shellu na vzdáleném počítači. Uživatel poté na serveru spouští programy s textovým uživatelským rozhraním
x11	Kanálu *pro grafické aplikace* spouštěné na serveru. Grafické okno se zobrazí na počítači klienta, uživatel může používat klávesnici, myš…
forwarded-tcpip	Tunel, který přesměruje TCP provoz z portu na serveru na klienta (*vzdálený tunel*).
direct-tcpip	Tunel, který přesměruje TCP provoz z portu klienta na server (*lokální a dynamický tunel*).

Zdroj: vlastní zpracování

2.4 Používání sady programů PuTTY

Na cvičení věnovaném SSH mají studenti následující úkoly:

- Přihlásit se na server *bis.vse.cz* pomocí SSH a hesla.
- Vytvořit osobní dvojici soukromý/veřejný klíč, veřejný klíč nakopírovat na server a přihlásit se *pomocí klíče*.
- Načíst soukromý klíč do paměti (`pageant` či `ssh-agent`) a přihlásit se pomocí klíče bez zadávání hesla. Zkopírovat soubory pomocí `WinSCP`.
- Vyzkoušet *tunely* (lokální, vzdálený a dynamický).

Studenti, kteří mají již se SSH předchozí zkušenosti, mají následující *doplňkové úkoly*:

- Spuštění příkazu na serveru a uložení výsledku do souboru na lokálním počítači.
- Vytvořit Jump host – přihlásit se přes jeden server na další server.

V první části bude popsán postup řešení těchto úkolů za použití sady programů `PuTTY` pro operační systém Windows. Stručné řešení stejných úkolů z UNIXu (*Linuxu*) či *macOS*, tj. používání programů `ssh` či `ssh-keygen`, bude popsáno v následující kapitole 2.4.2.

V příkladech budeme popisovat přístup na server *bis.vse.cz* s linuxovým operačním systémem *Debian* v aktuální verzi, na kterém je spuštěn *OpenSSH server*. Principy a postupy zde uvedené jsou však stejné i při přihlašování na jiné servery. SSH je k dispozici téměř pro každý unixový (linuxový) systém. SSH servery existují i pro další operační systémy včetně operačního systému Windows.

Přehled programů v sadě PuTTY

PuTTY je skupina (sada) bezplatných aplikací pro emulaci terminálu, komunikaci přes síť (protokol SSH a nezabezpečené protokoly telnet či rlogin) či přes sériovou linku a pro přenos souborů. Autorem programu je *Simon Tatham*, britský softwarový inženýr. Aktuální verzi si můžete stáhnout z <https://www.chiark.greenend.org.uk/~sgtatham/putty/> (ke stažení využívejte tuto oficiální stránku).

Pro Windows je k dispozici 32bitová i 64bitová verze a dále si můžete vybrat variantu MSI Windows Installer nebo prostý ZIP soubor, který stačí rozbalit do libovolného adresáře (fakticky tedy jde o přenositelnou/portable verzi). Doporučujeme ZIP, velikost aktuální verze je méně než 3 MB, takže není problém nahrát např. na USB flash a všude s sebou nosit aktuální verzi.

V současnosti existuje též oficiální verze *pro Linux*, instalujte pokud možno z repozitářů vaší linuxové distribuce. Případně nejnovější verze ve formě zdrojového kódu (tar.gz) je opět na výše uvedené oficiální stránce. Existují též neoficiální portace na jiné platformy.

Distribuce obsahuje šest programů, některé používají grafické uživatelské rozhraní, další jsou řádkoví klienti, které ovládáte příkazy. Autor (*Simon Tatham*) používá název PuTTY jak pro celou sadu programů, tak pro hlavní program. Zde se budeme snažit odlišit a pro označení programu používat název `putty` či `putty.exe`.

- **PuTTY** (`putty.exe`) – hlavní program, *SSH*, *Telnet* a *rlogin* klient s emulací terminálu.
- **PuTTYgen** (`puttygen.exe`) – generování a správa SSH klíčů včetně převodů mezi různými formáty uložení klíčů.
- **Pageant** (`pageant.exe`) – uchovává SSH klíče v paměti, ostatní programy poté využijí pro autentizaci. Tento klíč umí využít také již zmíněný program `WinSCP` (není součást PuTTY, bude krátce popsán dále).
- **PSCP** (`pscp.exe`) – přenos souborů pomocí příkazové řádky.
- **PSFTP** (`psftp.exe`) – přenos souborů pomocí příkazové řádky s ovládáním podobným jako mají FTP klienti určení pro terminály.
- **Plink** (`plink.exe`) – řádkový SSH klient, obvykle pro provedení jednoho příkazu na vzdáleném serveru (možnost automatizace opakovaných rutinních činností).

2.4.1 Konfigurační okno programu putty.exe

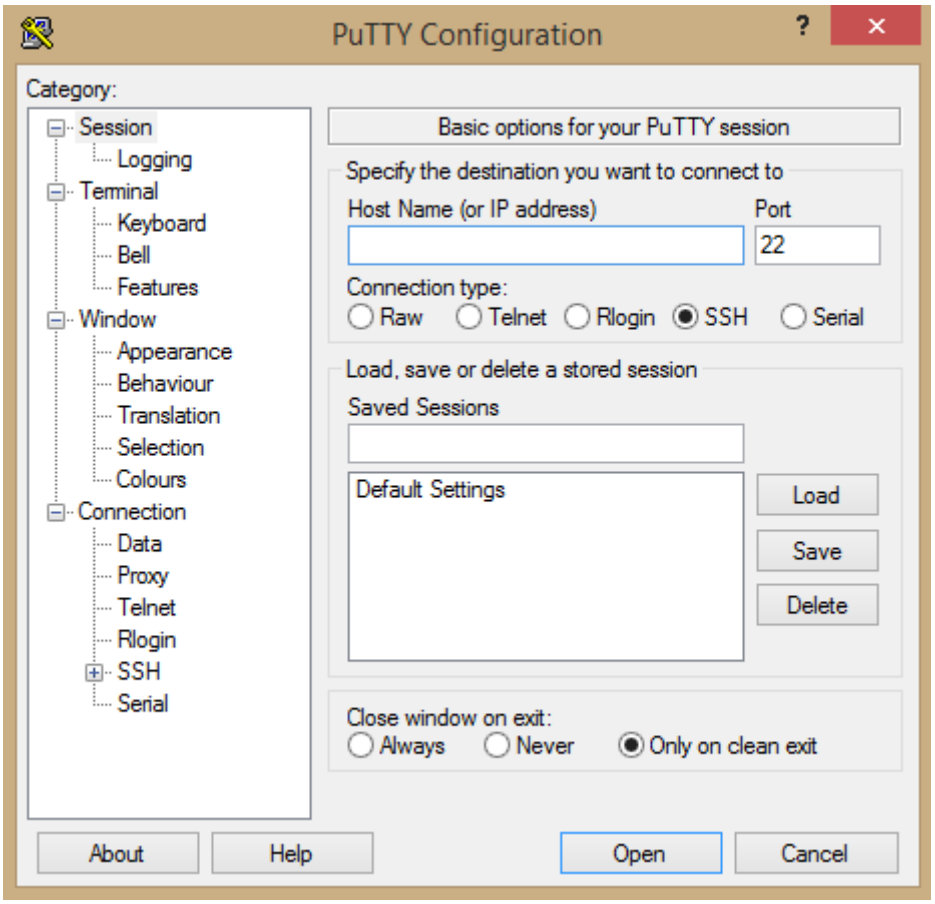
Nejčastěji se používá *terminálová emulace přes SSH protokol*. GUI rozhraní programu `putty.exe` obsahuje velké množství nastavení, nejdůležitější jsou popsány dále. Po přihlášení k serveru se *pro emulaci terminálu* zobrazí typické okno s černým pozadím a bílým písmem (od studentů jsme zaslechli slangové označení „černé okénečko“). Je to nejčastěji používaný SSH klient pro Windows, již jsme zmínili i existenci unixové/linuxové verze.

Po spuštění programu `putty.exe` se objeví konfigurační okno, ve kterém lze zadat parametry pro připojení k serveru. Okno se skládá z několika částí (viz obrázek níže). Vlevo je panel „*Category*“ se stromově uspořádaným velkým počtem voleb pro konfiguraci parametrů budoucího připojení.

V pravé části nahoře nadepsané „*Basic options…*“ (Základní volby) se do prvního bílého vstupního pole zadává cíl budoucího spojení (Specify the destination you want to connect to). Pod ní je správa sezení/relací (sessions) – parametry spojení si můžete uložit (tlačítko „Save“) a příště je rychle použít pomocí dvojkliku na jméno relace. Do parametrů spojení se ukládá nejen kam se chcete připojit, ale také všechna nastavení z panelu *Category*. Čím více nastavení provedete, tím výhodnější je nastavení uložit (např. typ emulovaného terminálu, velikost jeho okna, písmo a spousta dalších možností – viz dále).

Ve specifikaci cíle se do okénka „*Host Name*“ typicky zadává doménové jméno serveru ke kterému se chcete připojit (ale lze zadat i IP adresu). V okénku „Port“ je již předpřipraveno číslo 22, to je výchozí port SSH. Nicméně lze se setkat i se servery, které využívají jiný port. Nabídka „Connection Type“ umožňuje vybrat způsob připojení, použijeme předvolené SSH. Pro jistotu připomínáme, že připojení typu *Telnet* nebo *rlogin* jsou *nezabezpečená* spojení. Veškeré informace včetně hesel se v těchto případech přenášejí *nešifrovaně*!

Obrázek 2.8: Hlavní konfigurační a přihlašovací okno programu putty.exe

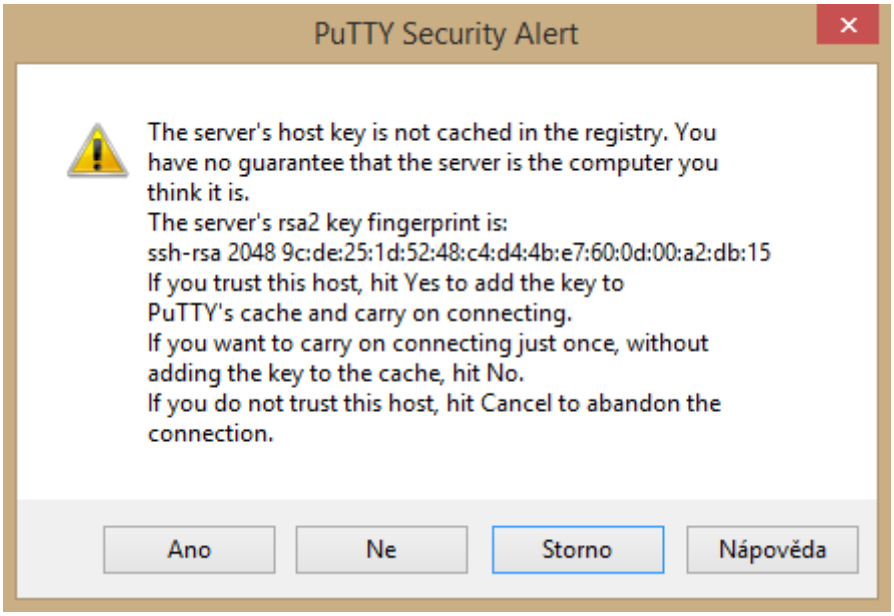


Zdroj: vlastní zpracování

2.4.2 První přihlášení na server bis.vse.cz (autentizace heslem)

Do okénka „*Host Name*“ vyplníme jméno serveru *bis.vse.cz* a tlačítkem „Open“ zahájíme komunikaci. Po zahájení komunikace se *autentizuje server* – pokud jste se serverem ještě nekomunikovali, zobrazí se Vám *otisk (fingerprint)* veřejného klíče serveru a výzva na jeho ověření.

Obrázek 2.9: Otisk veřejného klíče serveru při prvním přihlašování



Zdroj: vlastní zpracování

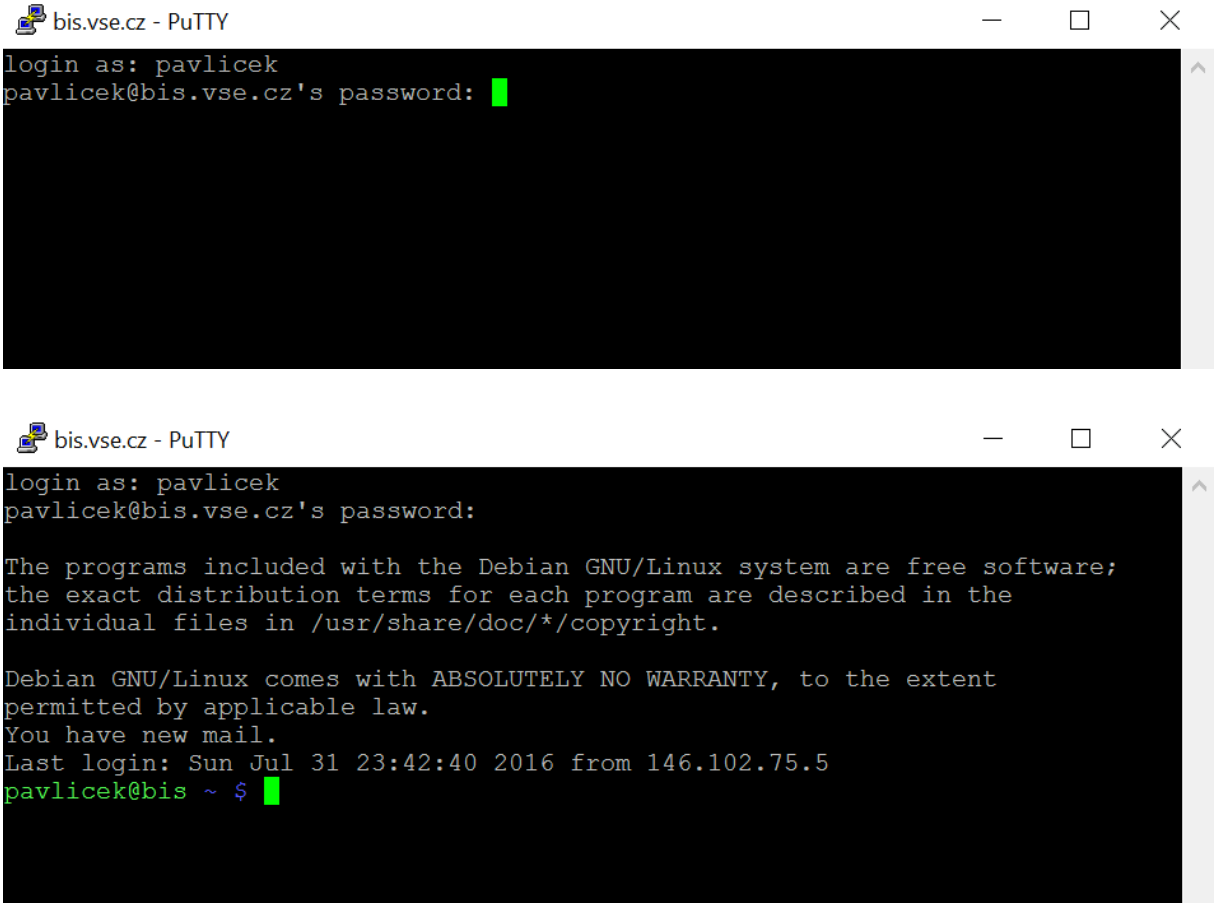
Pokud klíči (otisku klíče) *věříte*, stiskněte tlačítko „*Ano*“ (či „Yes“) a `putty.exe` si klíč uloží do registru Windows mezi důvěryhodné klíče. Při dalších připojeních `putty.exe` ověřuje, zda se klíč serveru nezměnil – pokud ne, tak se připojí bez dodatečných dotazů.

Kliknete-li na tlačítko „*Ne*“ („No“), připojování k serveru bude dále pokračovat, ale při příštím přihlašování se tento dialogový box s otiskem klíče zobrazí znovu. Tlačítkem „*Storno*“ pochopitelně přihlašování k serveru přerušíte.

Nepodceňujte ověření otisku klíče serveru při prvním přihlašování!
Zde na cvičení klíč potvrdíme bez dodatečného ověřování.
V reálném životě byste však ověření otisku klíče měli věnovat velkou pozornost. Ideálně otisk klíče obdržíte předem také *druhým a zabezpečeným* komunikačním kanálem, např. mailem, který je šifrovaný a digitálně podepsaný a mail porovnáte s tím, co se vám zobrazí v tomto dialogovém okně. Zaslání otisku klíče SMSkou již není ideální (běžné SMS zprávy nejsou zabezpečeny).

Následně můžete získat dodatečné informace od serveru a jste vyzváni *k zadání jména a hesla*. Heslo můžete zadat třikrát. Pokud se Vám ani jednou nepodaří zadat správné přihlašovací údaje, tak se SSH spojení ukončí. (Také po poměrně krátké době, pokud nezadáte nic.) Pokud se úspěšně přihlásíte na server *bis.vse.cz*, uvítá vás příkazový řádek (shell) *bash*. Často se zobrazí též úvodní informace a varování vztahující se k serveru. Přihlašování heslem je omezeno na školní síť, z domova nutno použít VPN.

Obrázek 2.10: Zadávání hesla a úspěšné přihlášení



Zdroj: vlastní zpracování

2.4.3 Základy používání programu PuTTY, doporučená nastavení

Používání schránky pro kopírování textu z/do okna putty.

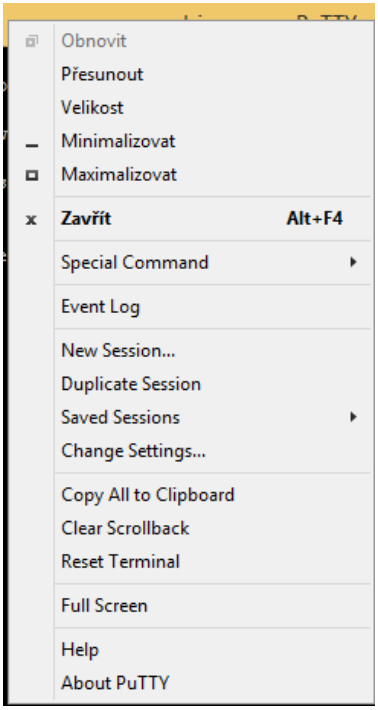
Program `putty.exe` podporuje v terminálovém okně kopírování do schránky a vkládání ze schránky. Ovládání se ale *liší* od běžných programů ve Windows. Kombinaci kláves `Ctrl+C` **nelze použít**, neboť tato kombinace se v UNIXu/Linuxu používá pro ukončení procesu (a tato zkratka vznikla v době, kdy Windows ještě vůbec neexistovaly). Ovládání z unixové grafické nadstavby *X-Windows* nebylo možné dlouhou dobu používat, neboť vyžaduje tři tlačítka myši. Ve výchozím nastavení proto používá putty.exe v **terminálovém okně** následující *kompromis* pro dvoutlačítkovou myš.

- Textu označený pomocí myši se *automaticky* vloží (zkopíruje) do schránky (nepoužívá se žádná klávesová zkratka).
- Obsah schránky se vloží do terminálového okna po stisknutí druhého (obvykle pravého) tlačítka myši.

Kontextové menu z horní lišty

Po zobrazení terminálového okna klikněte druhým (obvykle pravým) tlačítkem myši na horní lištu a zobrazí se Vám následující kontextové menu.

Obrázek 2.11: Kontextové menu programu putty.exe



Zdroj: vlastní zpracování

Z mnoha nabízených možností nyní stručně probereme některé důležité:

- **Event Log** – log z navazování a změny parametrů SSH spojení. Log soubor Vám pomůže při hledání různých problémů. Obrázek ukazuje log při úspěšném připojení pomocí klíče uloženého v paměti s pomocí programu `pageant` (viz kapitola 2.4.5).
- **Change Settings…** – můžete měnit parametry již vytvořeného spojení, např. volby pro *terminálovou emulaci* (popsané vzápětí) či přidávat/ubírat *SSH kanály* (viz kap. 2.4.7).

Nastavení emulace terminálu, klávesnice a písma

UNIX historicky podporuje různé terminály, které používají různé sekvence pro nastavení pozice kurzoru, pro volbu barvy textu či posílají různé sekvence na server při stisknutí funkčních kláves. Při vytvoření kanálu typu session s emulací terminálu `putty` posílá na server typ terminálu, a server dle tohoto typu interpretuje obdržené sekvence znaků a odesílá sekvence pro zobrazení v terminálovém okně programu *putty.exe*.

Typ emulovaného terminálu. Program standardně posílá typ *xterm* s podporou 16 barev. Pro připojení na linuxové servery doporučujeme změni na typ **putty-256color** či **xterm-256color**. Nastavuje se v *Connection -> Data -> Terminal-type string*. Některé aplikace budou automaticky používat 256 barev, u jiných musíte upravit nastavení.

Změna barvy. Výchozí je dobře čitelné bílé písmo na černém pozadí. Při použití některých linuxových příkazů (např. u příkazu `ls`) se však (obvykle) některé položky automaticky zobrazí v jiných barvách (podle toho, zda je o adresář, soubor s určitými parametry atd.) Text v modré barvě občas nebývá v černém okně dobře čitelný. Pak je vhodné ve *Window -> Colours* změnit odstín pro barvu *ANSI Blue*.

Klávesnice. Pokud Vám v aplikaci nefungují správně některé funkční klávesy, zkuste v *Terminal -> Keyboard* změnit emulaci funkčních kláves na *Linux*.

Panel s číslicemi a šipkami vpravo na klávesnici se používal jako aplikační panel: aplikace ke klávesám může přiřadit speciální význam a poté nelze vkládat číslice. To je v současnosti většinou matoucí, a proto doporučuji v *Terminal -> Features* zatrhnout volbu **„***Disable application keypad mode***“.**

Ve *Window -> Appearance* si můžete změnit font a velikost písma. Pokud se Vám chybně zobrazují české znaky či čáry, zkontrolujte ve *Window -> Translation*, že máte zvolenu znakovou sadu ***UTF-8*** a zatrženu volbu „*Use Unicode lie drawing code points”*.

Sezení (sessions) – uložení parametrů spojení.
Nastavené parametry spojení můžete uložit jako sezení (relaci, session) a příště dvojklikem na název snadno vytvořit spojení se všemi nastavenými parametry. Před stisknutí tlačítka ***Save*** pro uložení zadejte jméno sezení do políčka *Saved Sessions*.

V sezení si můžete přednastavit i uživatelské jméno – vložte ho do adresy v poli *Host Name* ve formě: **login@bis.vse.cz.**

2.4.4 Vygenerování klíčů v programu PuTTYgen

Přihlašování pomocí hesla umožňuje jejich hádání útočníky. V logovacích souborech serveru se často setkáte s velkým počtem neúspěšných přihlášení z různých částí světa. Nejčastěji se snaží uhádnout heslo pro uživatele `root` (administrátorský účet v UNIXu/Linuxu), z toho důvodu bývá přihlašování přes SSH na účet `root` zakázáno nebo omezeno na přihlášení pomocí klíče.

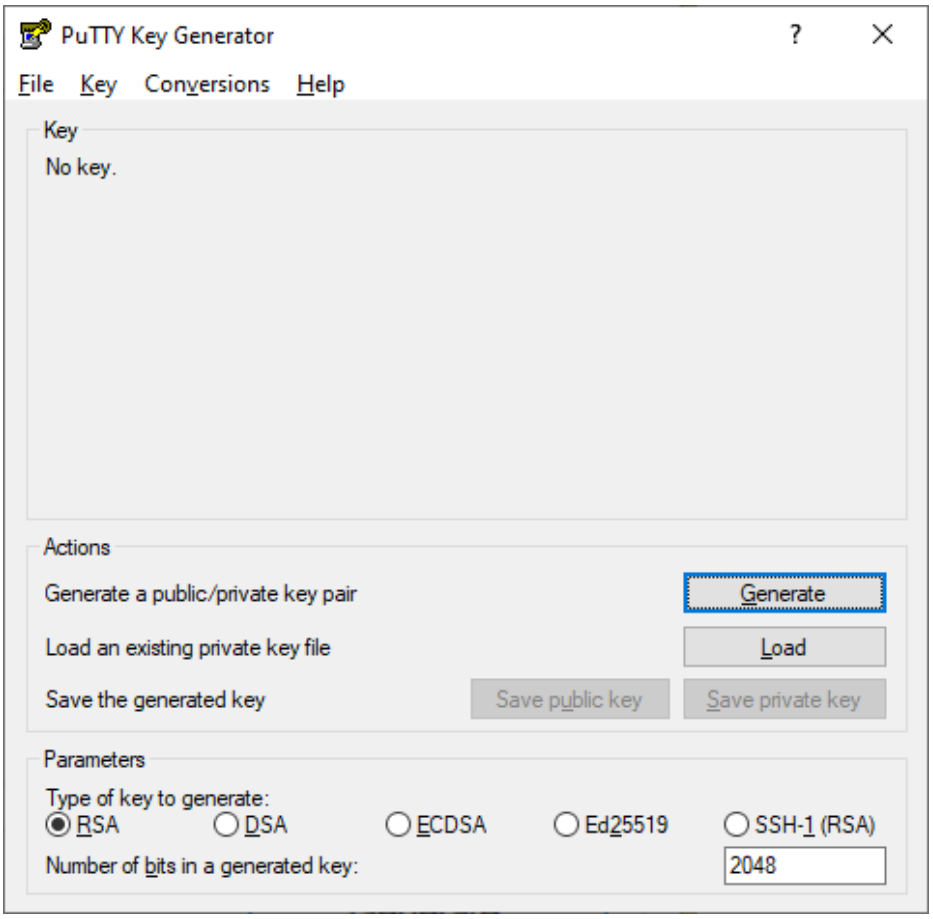
Přihlašování pomocí klíče je mnohem **bezpečnější**, neboť:

- *Entropie* hesel se pohybuje mezi 30 a 50, u klíčů se entropie odhaduje na přibližně 120 a výše. Útočníci nemají reálnou šanci uhodnout klíč.
- Klíče jsou bezpečné *vůči některým útokům* obvyklým u hesel. Uživatel si klíče neopisuje na papír, není schopen je někomu sdělit na potkání, samotné sledování klávesnice kamerou nevede ke kompromitaci klíčů.
- Na serveru je uložen *pouze veřejný klíč*. Kompromitace serveru nevede k prozrazení soukromého klíče. Soukromý klíč neopouští klientský počítač – nehrozí zcizení při přenosu po síti.

Přihlašování pomocí klíče podporuje většina SSH aplikací a SSH serverů.[132] SSH klíč se obvykle využívá *pouze* pro přihlašování (autentizaci).[133] Uživatel si musí nejdříve vygenerovat dvojici soukromý/veřejný klíč. Pro generování a správu klíčů je součástí sady programů PuTTY program `PuTTYgen`, který je popsán na následujících stránkách.

[132] Přihlašování pomocí *klíče* je dle RFC 4252 *povinná* autentizační metoda. Ověření uživatele pomocí *hesla nemusí* být implementováno. Ale v případě některých specializovaných zařízení, jako síťové prvky, autentizace pomocí klíče často chybí.
[133] Vygenerovanou dvojici klíčů lze použít i pro další účely, např. jako *osobní certifikát* pro S/MIME. Musíte ale překonat praktickou překážku v obtížné dostupnosti vhodného nástroje pro konverzi mezi formáty. Je to i proti bezpečnostním doporučením – pro každý účel by se měly používat odlišné klíče. Neplatí v případě vygenerování soukromého klíče na čipové kartě – zde není neobvyklé použití stejné dvojice klíčů pro různé účely.

Obrázek 2.12: Hlavní okno programu PuTTYgen



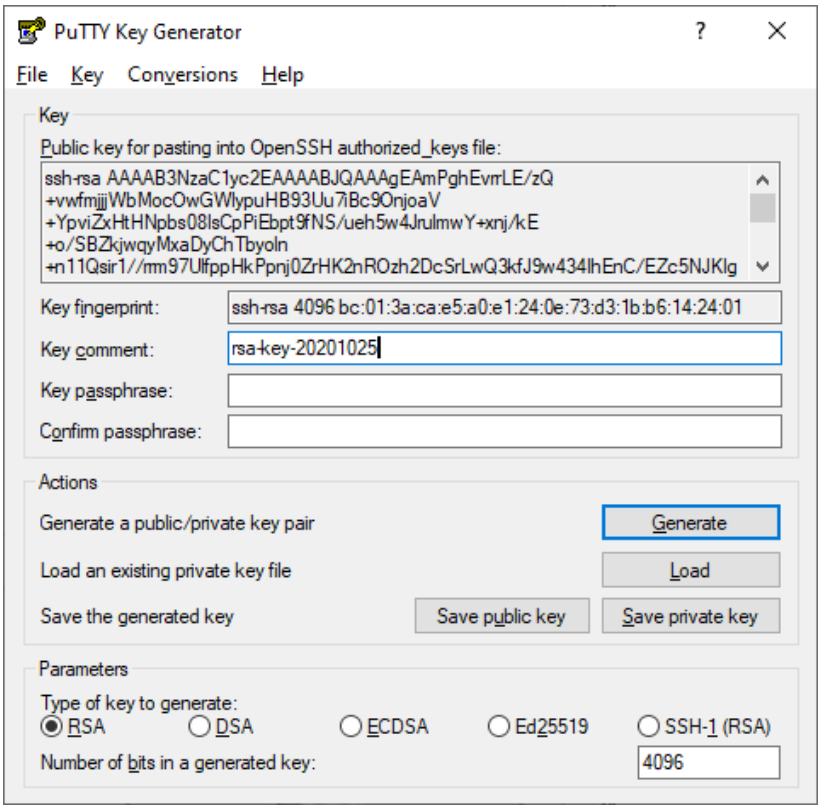
Zdroj: vlastní zpracování

Po spuštění programu PuTTYgen si lze *zvolit typ klíče a jeho délku*. Můžete použít předvolenou možnost – generovat RSA klíče pro verzi 2 protokolu SSH o délce 2048 bitů. Následují pravidla pro výběr klíčů.

- **RSA / SSH-2 RSA** – nejvhodnější z hlediska *kompatibility*. Délku klíče volte 2048, ale raději 3072 či 4096 bitů. Kratší či delší nemusí být na některých serverech podporovány.
- **ECDSA** – některé starší servery nemusí podporovat.
- **Ed25519** – nový moderní algoritmus, ale některé starší servery nemusí podporovat.
- **DSA** NEPOUŽÍVAT – od verze 7 programu *OpenSSH* přestaly být podporovány (ke konci roku 2020 je aktuální verze *OpenSSH* 8.4).
- **SHA-1 (RSA)** NEPOUŽÍVAT – první verze protokolu SSH není bezpečná a prakticky se s ní již nesetkáte.

Po stisknutí tlačítka „*Generate*“ se vygeneruje pár klíčů. Pro generování jsou potřeba náhodné hodnoty – jedním ze vstupu je pohyb kurzoru myši nad aplikací `PuttyGEN`. Po vygenerování je vhodné upravit komentář (*Key comment*), neboť při používání více klíčů Vám datum vygenerování příliš nepomůže. Doporučuji uvést vlastní e-mailovou adresu a účel klíče, např. `pavlicek@vse.cz github`.

Obrázek 2.13: Generování klíče v PuTTYgen



Zdroj: vlastní zpracování

Soukromý klíč je silně doporučeno chránit *dlouhou* heslovou frází,[134] neboť jinak může nezvaný návštěvník Vašeho počítače snadno zneužít Váš soukromý klíč. Heslová fráze se zadává do polí „*Key passphrase*“ a „*Confirm passphrase*“.

Pomocí tlačítka „*Save private key*“ uložíte soukromý i veřejný klíč společně do souboru s koncovkou `.ppk` (interní formát používaný celou sadou programů *PuTTY*). Samotné uložení veřejného klíče ve formátu *PuTTY* je zbytečné a spíše matoucí, neboť vytvořený soubor nemá smysluplné využití.

V programu PuTTYgen můžete též otevřít existující soubor se soukromým klíčem a poté např. změnit heslovou frázi. Lze též importovat soukromý klíč vygenerovaný v UNIXu (Linuxu) či exportovat soukromý klíč z PuTTY do formátu používaném v UNIXu.

Důležité upozornění – rozlišujte formáty ukládání klíčů:
PuTTY používá odlišný formát ukládání klíčů než program *OpenSSH*. Na svém počítači s Windows budete mít vždy *oba klíče* (soukromý a veřejný) uloženy *společně v jednom souboru* s příponou `.ppk`. Zájemci si mohou přečíst poměrně podrobné zdůvodnění volby tohoto formátu autorem programu na oficiálních stránkách programu.

Při připojování k serveru v UNIXu (Linuxu), resp. k serveru používající *OpenSSH* musíte *pouze veřejný klíč* uložit do formátu „*OpenSSH* `authorized_keys` file“ a nahrát na „správné místo“ na serveru (podrobnosti v následující kapitole).

[134] Zabezpečení souboru `.ppk` se soukromým klíčem je bohužel zastaralé, používá hašovací funkci SHA1 *bez iterací*: podrobněji viz s. 35. Pokud útočník získá soubor se soukromým klíčem, tak je schopen zkoušet stovky tisíc hesel za vteřinu. Velmi dlouhá heslová fráze je proto opravdu důležitá.

Uložení *pouze* veřejného klíče ve formátu *OpenSSH* do samostatného textového souboru je vhodné provést ihned *při generování klíče*. Z okénka nahoře (nadepsaného „*Public key for pasting into OpenSSH…*“) ho zkopírujte do schránky Windows a odsud uložte to textového souboru.

Lze to pochopitelně provést i dodatečně: v tomto případě klikněte na tlačítko „*Load*“, vyberte příslušný soubor s příponou `.ppk` a zadejte heslovou frázi chránící přístup k souboru. Poté opět přes schránku zkopírujte a uložte do textového souboru.

2.4.5 Přihlašování pomocí klíčů

Nyní předpokládáme, že máte vygenerovaný klíč ve formátu `.ppk` (tento soubor obsahuje soukromý i veřejný klíč, jak již bylo vysvětlenou) uložený ve vhodném adresáři na svém počítači s Windows a současně máte v samostatném souboru připraven *pouze* veřejný klíč ve formátu *OpenSSH*. Ten je třeba „správně“ nahrát na server.

Nejprve se pochopitelně musíte (zatím ještě se zadáním uživatelského jména a hesla) programem `PuTTY` přihlásit na server. K nahraní klíče lze použít program `pscp.exe` nebo `psftp.exe` (oba jsou součástí sady programů PuTTY, jak již bylo dříve popsáno), ale pro běžné uživatele bez zkušeností s Linuxem je asi mnohem jednodušší použít program `WinSCP`, který je rovněž stručně charakterizován na konci následující podkapitoly 2.4.6).

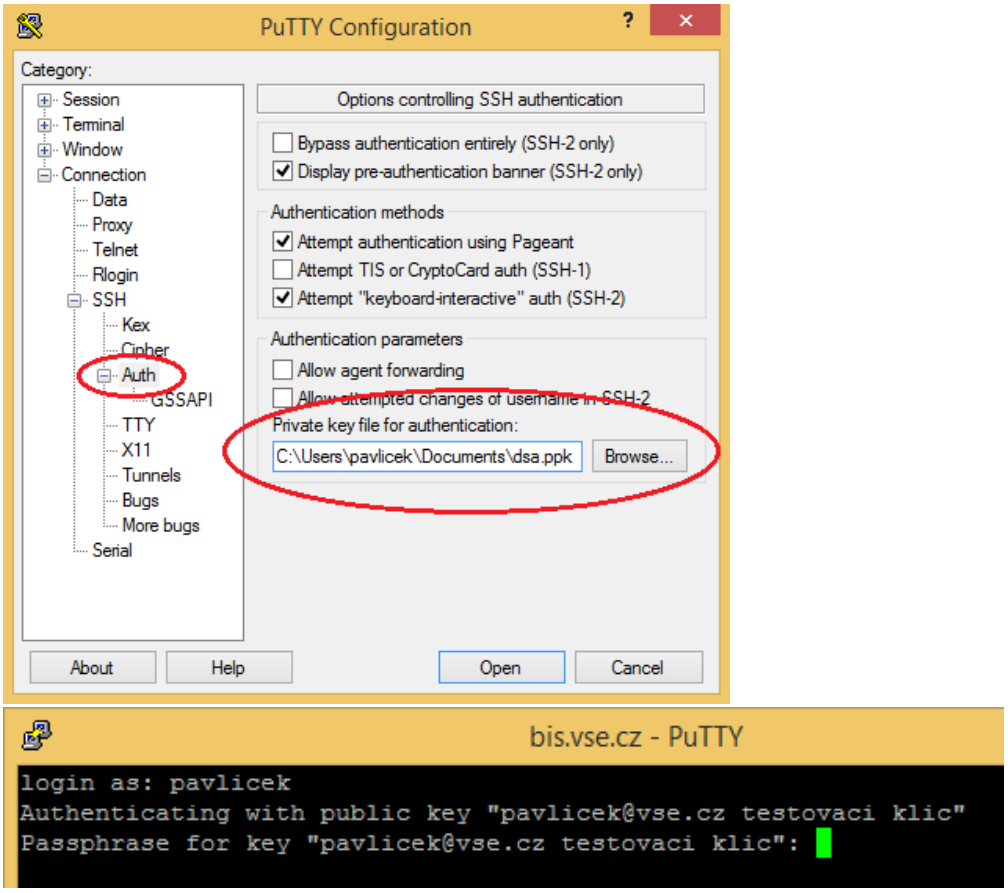
V Linuxu (UNIXu) se veřejný klíč ve formátu *OpenSSH* obvykle zapisuje do souboru `authorized_keys` v podadresáři `.ssh` domovského adresáře uživatele: cesta k souboru se zkráceně zapisuje jako `~/.ssh/authorized_keys` (znak tilda znamená domovský adresář). Adresář `.ssh` nemusí ještě existovat, dle potřeby ho nejprve vytvořte. Připomínáme, že adresáře a soubory začínající tečkou jsou v Linuxu *skryté* (např. se vám nezobrazí příkazem `ls`, pokud k němu nepřidáte příslušný přepínač, doporučujeme `ls -la`, což současně vypíše také nastavení přístupových práv, která jsou pro správné fungování důležitá).

Před přihlášením musíte programu `putty.exe` *zpřístupnit soukromý klíč*. Existují dvě základní možnosti:

- *Do konfigurace spojení* (Category: Connection, SSH, Auth – viz Obrázek 2.14) zadáte jméno souboru se soukromým klíčem (soubor s příponou `.ppk`: jak víte, ten v `PuTTY` obsahuje soukromý i veřejný klíč). Při *každém* připojení musíte zadat heslovou frázi, která chrání přístup k soukromému klíči.
- *Soukromý klíč nahrajete do paměti programem* `pageant` (viz Obrázek 2.15). Pokud je klíč (klíče) v paměti, program `putty.exe` poté *automaticky* vyzkouší přihlášení pomocí *všech* soukromých klíčů v paměti (mimo jiné se tak lze snadno střídavě přihlašovat k různým serverům s různým klíčem). Heslovou frázi k soukromému klíči zadáváte *pouze jednou* při jeho nahrání do paměti.

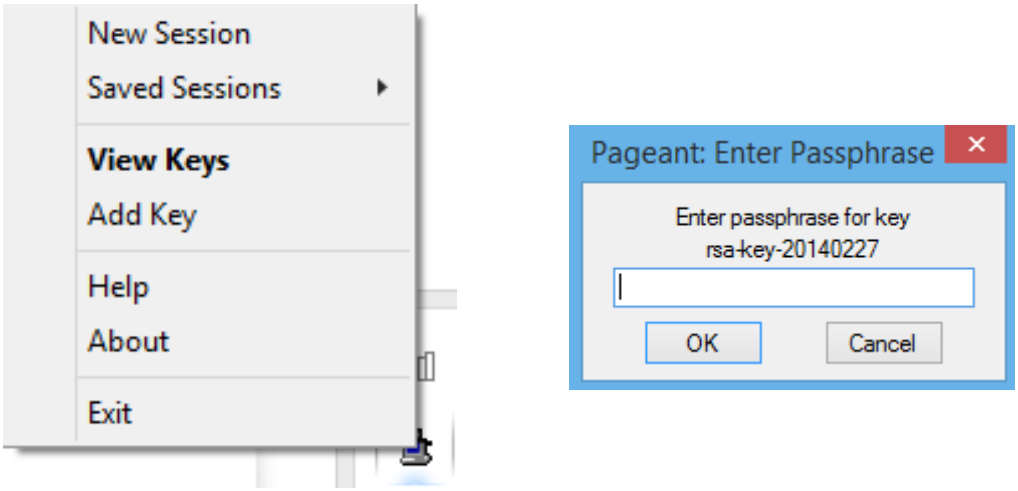
Použití programu `pageant.exe` je jednoduché. Nejprve tento program (součást instalace PuTTY) spustíte. Zdánlivě se nic neděje, pouze se vytvoří ikona v tzv. *oznamovací oblasti* Windows. Je vcelku nenápadná, takže někteří studenti ji přehlédnou (standardně je vpravo dole, vlevo od zobrazení data/času). Ihned nebo kdykoli později můžete přes *kontextové menu* prohlížet, přidávat a odebírat klíče z paměti. Pozor: obvykle již po několika minutách Windows ikonku „uklidí“, takže pak je nutné nejprve kliknout v oznamovací oblasti na „Zobrazit skryté ikony“ a až zde kliknout pravým tlačítkem na ikonku programu `pageant`. Samozřejmě při přidávání klíče je nutné zadat jeho heslovou frázi.

Obrázek 2.14: Přidání klíče do konfigurace spojení a následné zadání heslové fráze



Zdroj: vlastní zpracování

Obrázek 2.15: Použití programu Pageant



Zdroj: vlastní zpracování

Pokud chcete jen přidat klíč (a program jste instalovali, takže přípona `.ppk` je s ním asociována – nebo při rozbalení ZIP portable verze jste asociaci přidali) stačí v Průzkumníkovi (jiném správci souborů) na soubor s koncovkou `.ppk` kliknout.

V kontextovém menu programu `pageant` jsou také volby pro spuštění `putty.exe` s novou konfiguraci (*New Session*) či s existujícím uloženým sezením/relací (*Saved Sessions*). V uložených sezeních by mělo být uživatelské jméno – jinak ho musíte při přihlašování zadat.

Nahrání klíčů do programu `pageant` je *preferovaná a pohodlnější cesta*, ale s určitým *bezpečnostním rizikem* – když opustíte přihlášený počítač, tak si kdokoliv může sednout za Vaši klávesnici a přihlásit se na vzdálený server jako Vy. Opatření je jednoduché (a měli byste ho používat vždy, i když nepracujete se servery) – při *každém odchodu* od klávesnice *počítač uzamknout*. Ve Windows pomocí kombinace `Win+L` (případně pomocí kombinace `Ctrl+Alt+Del` a následně vybrat z nabídky volbu *Uzamknout*).

Časté chyby začátečníků při nahrávání veřejného klíče na server

„Vynalézavost" studentů a obecně začátečníků v Linuxu „při vytváření chyb" je dost vysoká, ale podle našich zkušeností ze cvičení by následující tipy měly vyřešit *naprostou většinu případů*, kdy vám přihlašování klíčem *nefunguje* (stále je nutné přihlašovat se uživatelským jménem a heslem).

- *Chybné umístění* veřejného klíče na serveru nebo chyba v *názvu adresáře* (většinou chybějící tečka) nebo *souboru* („s" místo „z" ve slově *authorized*, „key" místo „*keys*", pomlčka nebo mezera místo podtržítka „_" v názvu). Velké písmeno (písmena) někde v názvu adresáře nebo souboru.
- Nevhodné nastavení *přístupových práv* k adresáři a/nebo souboru. „Vadí" nejen nedostatečná, ale i nadměrná práva. Právo číst i zapisovat do souboru by měl mít jen vlastník souboru, skupina i ostatní maximálně číst. Obdobně i práva k adresáři, kde vlastních pochopitelně má navíc i právo „execute").[135]
- Veřejný klíč byl nahrán ve *špatném formátu* (místo formátu *OpenSSH* ve formátu *PuTTY*) nebo jste při generování klíče a jeho uložení přes schránku do formátu *OpenSSH* nezkopírovali celý klíč. Vypište si příkazem `ls -la` soubory v adresáři *.ssh* a zkontrolujte jak nastavení práv (viz výše), tak délku souboru., není-li nula (časté!) nebo „divná". V případě RSA 2048 bitů by délka měla být cca 395–410 bytů, v případě RSA 4096 bitů cca 730–750 bytů (záleží mj. na délce komentáře). Vymažte příkazem `clear` obrazovku terminálu a soubor vypište příkazem `cat authorized_keys`. Na začátku výpisu musí být „ssh-rsa" a na konci *celý* váš komentář. Pokud by klíč byl ve formátu *PuTTY*, tak to z výpisu také poznáte.
- Pokud jste z nějakého důvodu generovali klíč dvakrát, je možné, že veřejný klíč uložený na serveru *neodpovídá* souboru `.ppk` na vašem PC ve Windows resp. načtenému do paměti.
- Poslední tip je spíše do budoucna. V souboru `authorized_keys` může být (a bývá) *více veřejných klíčů* (proto je také v názvu „keys" nikoli „key": viz dříve popsané chyby v názvu) s různou délkou, každý klíč na jednom dlouhém řádku (délky viz výše). Naproti tomu v souboru `.ppk` je vždy *jenom jeden* soukromý a současně korespondující veřejný klíč a jsou zapsány na větším počtu krátkých řádků.

[135] Důsledky *nedostatečných* práv (vlastník nemá právo *write*, příp. dokonce ani *read*) jsou zřejmé. *Nadměrná* přístupová práva jsou nejen zbytečným *bezpečnostním rizikem*, ale minimálně tehdy, pokud se uživatel připojuje k serveru z počítače s Linuxem nebo macOS a použije zde standardního řádkového klienta *ssh* (jeho popis viz kap. 2.5), tak ten oznámí, že soubor je nedostatečně zabezpečen a *odmítne* v přihlašování k serveru pokračovat. Program `PuTTY` při připojování z Windows byl v tomto směru tolerantnější, ale zdá se, že poslední verze jsou nyní také přísnější. Pro soubor se doporučují práva „644", příp. „640".

2.4.6 Kopírování souborů přes SSH

V SSH jsou dva protokoly pro kopírování souborů – protokol **scp** a protokol **sftp**. V PuTTY jsou dva odpovídající *řádkové programy*: pscp.exe a psftp.exe.

Program `pscp.exe` má podobnou syntaxi jako příkaz `cp` (copy) pro běžné kopírování souborů v Linuxu (UNIXu) resp. jako linuxový program `scp` pro kopírování na/ze serveru: při spuštění musíte zadat zdroj i cíl kopírování, poté se soubor či soubory zkopírují a program skončí. V následujících ukázkách se předpokládá klíč v paměti (`pageant.exe`).

```
# zkopírování soubor seznam ze serveru bis.vse.cz do lokálního adresáře
pscp pavlicek@bis.vse.cz:seznam .
# zkopírování adresáře obrazky na bis.vse.cz do domovského adresáře
pscp -r obrazky pavlicek@bis.vse.cz:
```

Program `pstfp.exe` emuluje chování programu *ftp* – vytvoříte spojení se serverem a poté se Vám zobrazí prompt `psftp>` pro zadávání jednotlivých příkazů. Příkaz `help` zobrazí přehled dostupných příkazů (viz ukázka).

```
psftp pavlicek@bis.vse.cz
Using username "pavlicek".
Remote working directory is /home/pavlicek
psftp> help
! run a local command
bye finish your SFTP session
cd change your remote working directory
chmod change file permissions and modes
close finish your SFTP session but do not quit PSFTP
del delete files on the remote server
dir list remote files
exit finish your SFTP session
get download a file from the server to your local machine
help give help
lcd change local working directory
lpwd print local working directory
ls list remote files
mget download multiple files at once
mkdir create directories on the remote server
mput upload multiple files at once
mv move or rename file(s) on the remote server
open connect to a host
put upload a file from your local machine to the server
pwd print your remote working directory
quit finish your SFTP session
reget continue downloading files
ren move or rename file(s) on the remote server
reput continue uploading files
rm delete files on the remote server
rmdir remove directories on the remote server
psftp>
```

Pohodlnější je pro kopírování souborů používat **WinSCP**: Windows program s grafickým uživatelským rozhraním (Martin Přikryl, domovská stránka https://winscp.net/). I když název vznikl ze slov *Windows Secure Copy*, program podporuje nejen protokol ***SCP***, ale též ***SFTP*** (nyní výchozí), ***FTP*** (původní FTP protokol s nešifrovaným přenosem), ***WebDAV*** a rovněž ***Amazon S3*** (přístup ke službám *AWS: Amazon Web Services*).

Jde o program s otevřeným zdrojovým kódem (licence GNU GPL), implementace *protokolu SSH* vychází právě ze sady programů *PuTTY* a program rovněž umí *automaticky* použít klíče z paměti, které tam načetl program `pageant`. Standardně se vám ke stažení nabídne instalační program pro Windows, ale lze zvolit i *přenositelnou verzi*. Další výhodou přenositelné verze je též to, že funguje rovněž v *Linuxu* pod *Wine*.

Standardně program používá dvou panelové ovládání obdobně jako *Total Commander* (*Double Commander*, *Midnight Commander*), ale lze přepnout též do zobrazení jednoho panelu ve stylu Průzkumníka ve Windows. Doporučujeme však dvou panelové ovládání (typicky v jednom panelu máte adresáře a soubory na serveru, v druhém panelu adresáře a soubory na vašem lokálním počítači). Kdo někdy používal *Total Commander* (*Double Commander*) neměl by mít s ovládáním žádné problémy, dokonce i většina *základních zkratek* je stejná (*F5* kopírování, resp. upload/download, *F7* nový adresář, *F8* smazání, dále např. *Alt+F1* a *Alt+F2*, *Alt+F7*, *Ctrl+A*, *Num+*, *Num–*, Num*, *Ins* atd.).

Upozornění: program respektuje *konvence Linuxu* a proto ve výchozí konfiguraci *nezobrazuje skryté* adresáře/soubory (začínající tečkou), tedy ani adresář `.ssh`. Pochopitelně lze snadno upravit v nastavení.

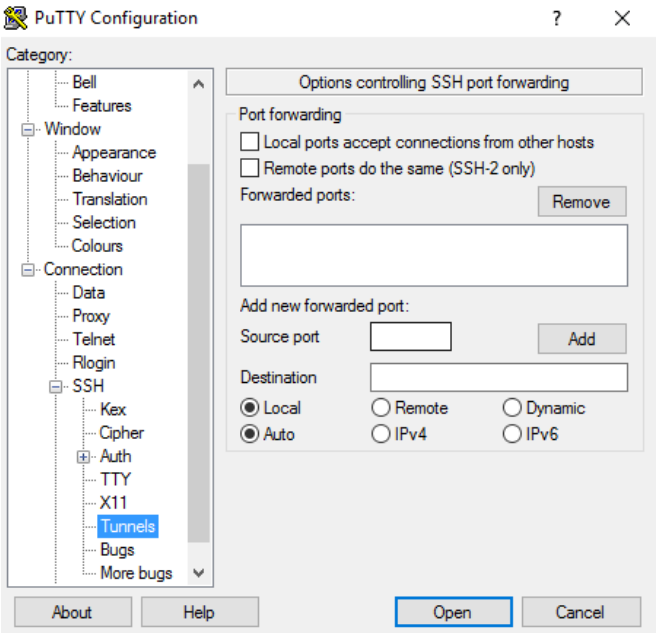
2.4.7 Tvorba tunelů a přesměrování portů

Protokol SSH *není omezen na* sezení s terminálovým oknem či na přenos souborů. Lze jej využít ke komunikaci programů prostřednictvím vytvořeného bezpečného kanálu. To se v praxi používá k:

- obejití firewallu prostřednictvím bezpečného tunelu,
- zašifrování komunikace, která jinak šifrování nepodporuje.

Oba případy se *vzájemně doplňují*, oba používají schopnost **přesměrování portů (port forwarding)** SSH. Přesměrování portů lze nastavit při připojení, i v průběhu již navázaného spojení. V `putty.exe` se přesměrování nastavuje v záložce *Connection -> SSH -> Tunnels*

Obrázek 2.16: Nastavení vytvářeného tunelu v PuTTY



Zdroj: vlastní zpracování

Existují **čtyři základní druhy přesměrování:**

- Lokální (Local, Forward) tunel
- Vzdálený (Remote, Reverse) tunel
- Dynamický (Dynamic, Proxy)
- X11 (X11 forwarding)

Lokální (Local, Forward) tunel zpřístupní jinak nepřístupný vzdálený TCP server pro lokální klienty. Např. MySQL databáze není dostupná z Vašeho domácího počítače, tak vytvoříte tunel se vstupem na lokální stanici (např. port 7000) a cílem v MySQL databázi (port 3306) na serveru. A poté na stanici spustíte MySQL klienta, který připojíte na lokální port 7000. SSH tunel přenese TCP stream na port 3306 MySQL databáze a zpět klientovi doručí TCP odpovědi.

Tunel může končit na jiném interním serveru – spojení ze SSH serveru k internímu serveru *není šifrováno*.

Vzdálený (Remote, Reverse) tunel zpřístupní připojení pro vzdálený server. Vytvoříte SSH spojení na server a v něm tunel, jehož vstupní port bude na serveru a cíl bude na lokální stanici či v síti lokální stanice.

Server *bis.vse.cz* má blokován přístup na Internet a Vy se z něho chcete přihlásit na server *telehack.com* na port 23. Ze své stanice vytvoříte tunel, který otevře vstupní port 6000 na serveru *bis.vse.cz* a cíl bude na serveru *telehack.com* na portu 23. Poté na serveru *bis.vse.cz* spustíte program telnet s cílem *localhost:6000*.

Dynamický (Dynamic, Proxy) zobecňuje lokální tunely – aplikace si sama může určit cílový server a port. Pro zadání cíle *musí* aplikace podporovat *protokol SOCKS 5.* Ve webových prohlížečích podpora většinou je, v mnoha jiných aplikacích chybí.

X11 (X11 forwarding) – na serveru spustíte grafickou aplikaci a vlastní grafické okno se zobrazí na lokálním počítači. Stisky kláves a pohyby myši se přenášejí z lokálního počítače do grafické aplikace na serveru. Na lokálním počítači musíte mít spuštěn tzv. `X-Server`: aplikaci, která umí zobrazovat grafiku z grafických unixových aplikací. Příkladem takové aplikace může být *VcXsrv Windows X Server* která využívá X11 tunely vytvořené pomoci PuTTY (https://sourceforge.net/projects/vcxsrv/). Jinou možností je aplikace *MobaXterm,* která v sobě integruje X-Server se SSH klientem (https://mobaxterm.mobatek.net/).

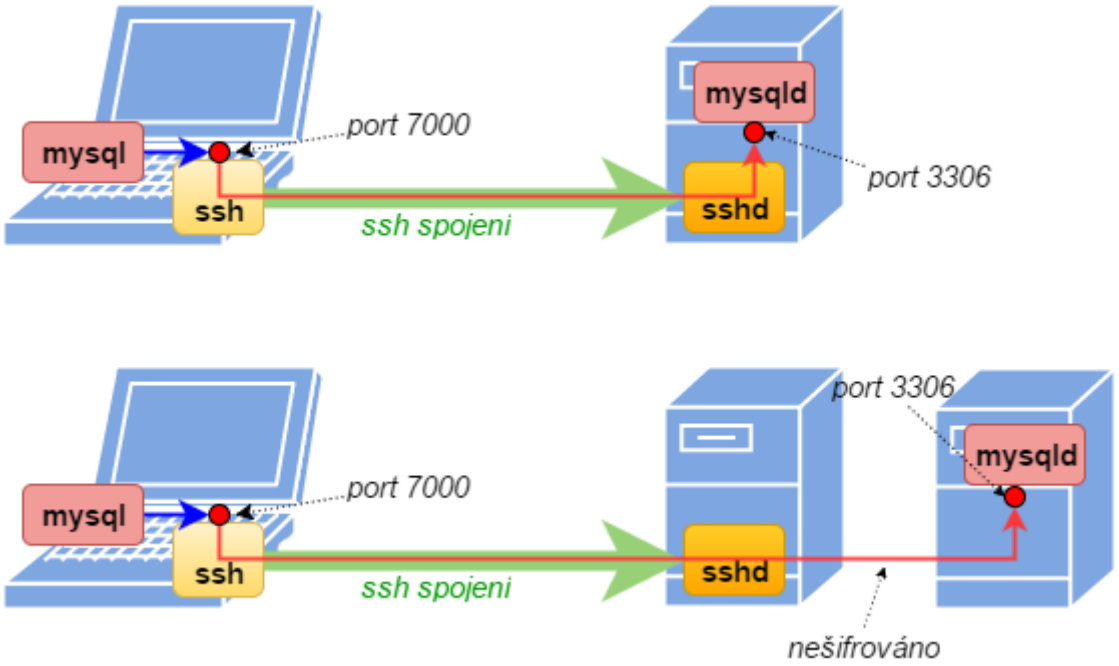
2.4.7.1 Lokální tunel

V obou příkladech na lokální tunel budeme přistupovat na *databázi* MySQL resp. MariaDB (TCP port 3306). V prvním případě databáze poběží na serveru *bis.vse.cz*, ve druhém případě na serveru *kitscm.vse.cz*.

Jaké volit číslo lokálního portu?. Číslo portu je 16bitové číslo bez znaménka, možný rozsah tedy je 0–65535. Obvykle se volí čísla nad 1024, protože porty 0–1023 jsou tzv. *privilegované porty* a *většina* čísel v tomto rozsahu je obsazena základními službami Internetu (např. 20 a 21 je FTP, 22 je SSH, 25 a 110 využívají protokoly SMTP a POP3, 53 používá systém DNS, 67–68 DHCP, 80 a 443 se používají pro HTTP resp. HTTPS atd.

I mnohé porty nad 1024 jsou přiřazeny ke známým službám, např. výše již bylo zmíněno, že databáze MySQL (MariaDB) používá port 3306. Jako „*vhodné*“ číslo lze uvést např. 7000 nebo 8000 a další čísla v tomto rozsahu (ale třeba 8008 a 8080 jsou *oficiálně* přiřazeny jako alternativní k portu 80 pro *http* – i když dnes již pravděpodobně nevýznamné, vzhledem k převažujícímu použití HTTPS). Ve Windows obsazené porty zjistíte příkazem `netstat -a -p TCP`, v Linuxu `netstat -ltn`.

Obrázek 2.17: Schéma dvou základních variant lokálního tunelu

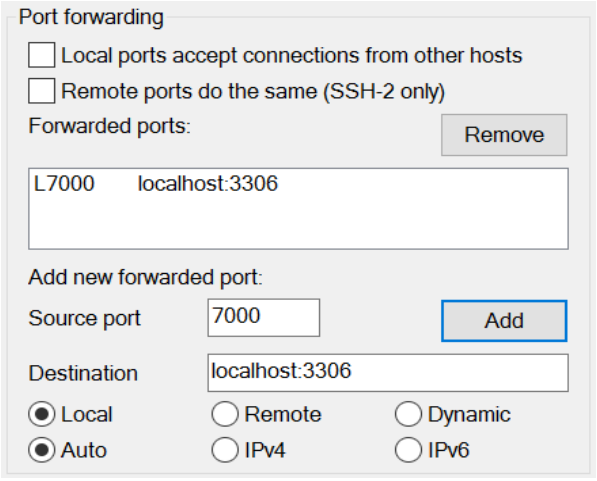


Zdroj: vlastní zpracování

Jméno cíle se vyhodnocuje na vzdáleném konci tunelu. Většinou to nepředstavuje komplikace, výjimkou je jméno *localhost*, které se převádí na IPv4 adresu `127.0.0.1` (v případě IPv6 na adresu `::1`). Tato adresa existuje na každém počítači s podporou protokolu IP, *localhost* na notebooku je jiný než *localhost* na *serveru bis.vse.cz*.

V `PuTTY` se tunel nastavuje v záložce *Connection -> SSH -> Tunnels:* zdrojový port (*Source port*) v našem příkladu (MySQL na serveru *bis.vse.cz* – viz Obrázek 2.18 níže) bude 7000, cíl (*Destination*) bude localhost:3306, necháte předvolený lokální (*Local*) tunel a automatickou volbu protokolu. Poté přidáte (*Add*) tunel do seznamu (*Forwarded ports*). Pro databázi na serveru *kitscm.vse.cz* je nastavení obdobné, ale cíl *kitscm.vse.cz:3306*.

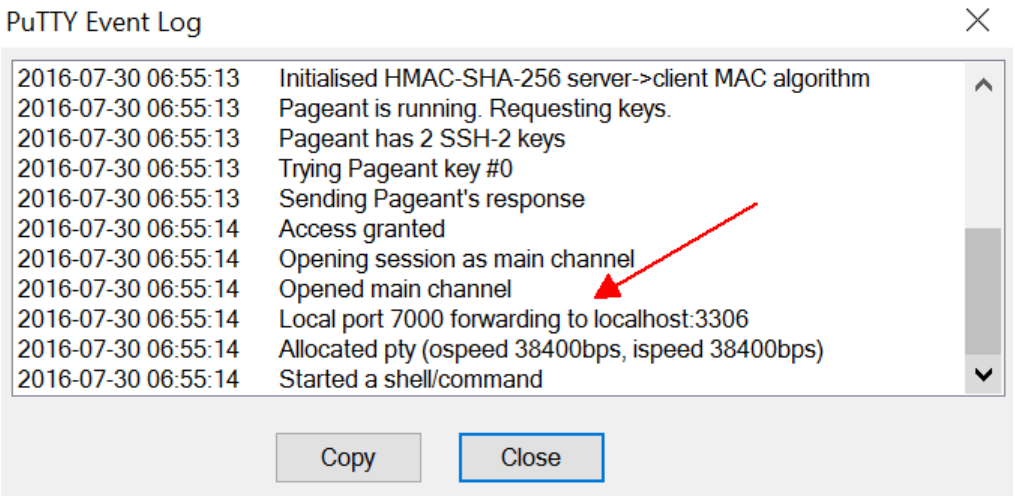
Obrázek 2.18: Nastavení lokálního tunelu (databáze MySQL na portu 3306)



Zdroj: vlastní zpracování

Nejčastější chybou je nestisknutí tlačítka **Add** pro přidání návrhu do seznamu tunelů. V logu (*Event Log*) programu `PuTTY` si můžete zkontrolovat, že se tunel vytvořil, viz následující obrázek, opět pro případ databáze na *bis.vse.cz*.

Obrázek 2.19: Kontrola vytvoření tunelu v logu



Zdroj: vlastní zpracování

Pokud máte na stanici nainstalovaného řádkového klienta, tak se na MySQL server připojíte následujícím příkazem (v grafických klientech je to podobné).

```
mysql --host localhost --port 7000 --user abcd01 -password
```

Na tunelované komunikaci s MySQL (MariaDB) serverem si ukážeme, jak se v posílané zprávě mění adresy a porty. MySQL (MariaDB) klient spuštěný na notebooku posílá *TCP paket* s následujícími parametry:

```
Zdrojová IP adresa: 127.0.0.1
Cílová IP adresa: 127.0.0.1
Zdrojový TCP port: náhodný, např. 37256
Cílový TCP port: 7000
```

Paket dostane SSH klient, který ve vytvořeném kanálu SSH spojení na druhou stranu pošle data z TCP paketu. Na druhém konci vytvoří následující TCP paket:

```
Zdrojová IP adresa: 146.102.18.87 (bis.vse.cz)
Cílová IP adresa: 146.102.18.2 (kitscm.vse.cz)
Zdrojový TCP port: náhodný, např. 56102
Cílový TCP port: 3306
```

Důsledkem je, že server *kitscm.vse.cz* si myslí, že spojení na MySQL databázi vzniklo na *bis.vse.cz* a o SSH tunelování a notebooku nic neví.

2.4.7.2 Vzdálený (remote) tunel

Vzdálené (remote, reverse) přesměrování portů si ukážeme na přístupu z *bis.vse.cz* ke službě telnet (port 23) na serveru `telehack.com`. Server *bis.vse.cz* má blokovaný přístup mimo školu a z toho důvodu příkaz `telnet telehack.com 23` nebude úspěšný.

Poznámka: *telehack.com* provozuje tzv. open telnet server a přes protokol telnet zpřístupňuje řadu menších služeb a aplikací (dnes již převážně jako zajímavost, a také jako ukázku, jak vypadal Internet v druhé polovině 80. let a na začátku 90. let, tedy předtím, než se rozšířila služba WWW).

Obrázek 2.20: Schéma vzdáleného tunelu (na server telehack.com)

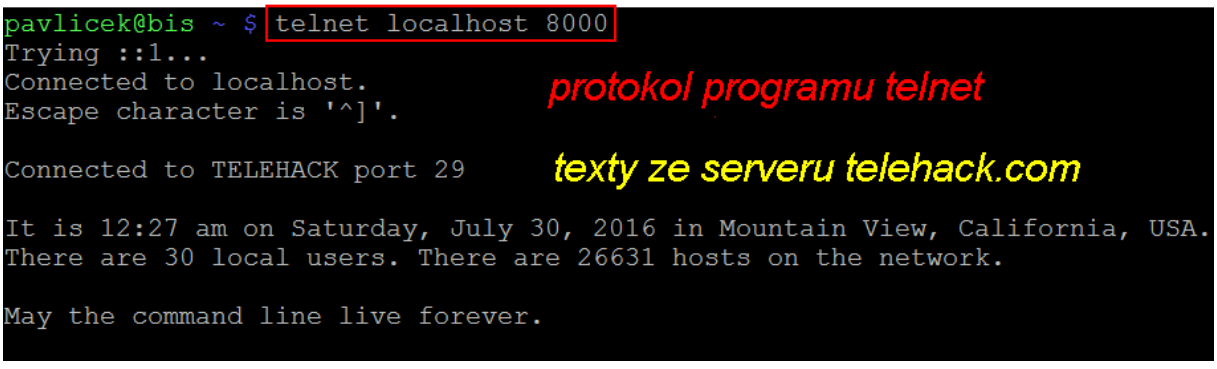
Zdroj: vlastní zpracování

Notebook má přístup na server *telehack.com* a tak pouze nastavíme vzdálený tunel v `putty`. V konfiguraci nastavíme zdrojový port `8000` (na ilustrativním schématu výše je uveden port 6000, zatímco ve skutečném připojení na obrázku níže je opravdu port 8000). Cílem bude `telehack.com:23` a pod tím vybereme typ *Remote*. Opět nezapomeneme na tlačítko **Add**.

Po vytvoření tunelu se k portu 8000 mohou připojit *všichni* uživatelé přihlášení na *bis.vse.cz* pomocí níže uvedeného příkazu telnet, výsledek viz Obrázek 2.21 níže). Všimněte si, že v příkazu `telnet` je číslo portu odděleno *mezerou*, zatímco v `PuTTY` nebo třeba do adresního řádku webového prohlížeče se před číslo portu píše *dvojtečka*!

```
telnet localhost 8000
```

Obrázek 2.21: Připojení se k serveru Telehack příkazem telnet (skrz vzdáleny tunel)



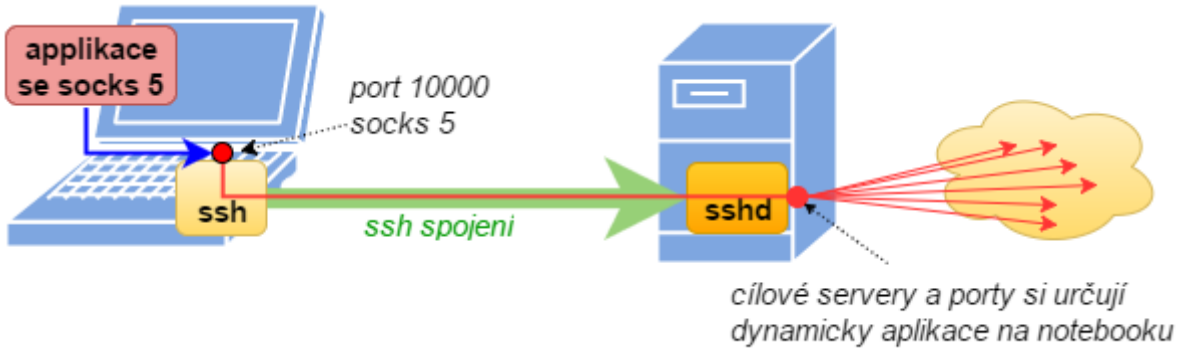
Zdroj: vlastní zpracování

Komunikace mezi notebookem a serverem telehack.com *šifrována není*. Dále nutno zdůraznit, že v případě vzdáleného přesměrování musíte zvolit zdrojový port, který není již na serveru použit. Pokud dva uživatelé použijí stejný zdrojový port, tak vznikne kolize. Zda se tunel vytvořil, můžete opět zjistit z logu `putty.exe` (*Event Log*).

2.4.7.3 Dynamický tunel

V posledním příkladu vytvoříme dynamický tunel na portu 10 000. Již jsme uvedli, že dynamický tunel zobecňuje lokální tunely – aplikace si sama může určit cílový server a port. Nutnou podmínkou pro využití dynamických tunelů je *podpora protokolu* SOCKS 5 na straně aplikace. Základní schéma dynamického tunelů je na obrázku níže.

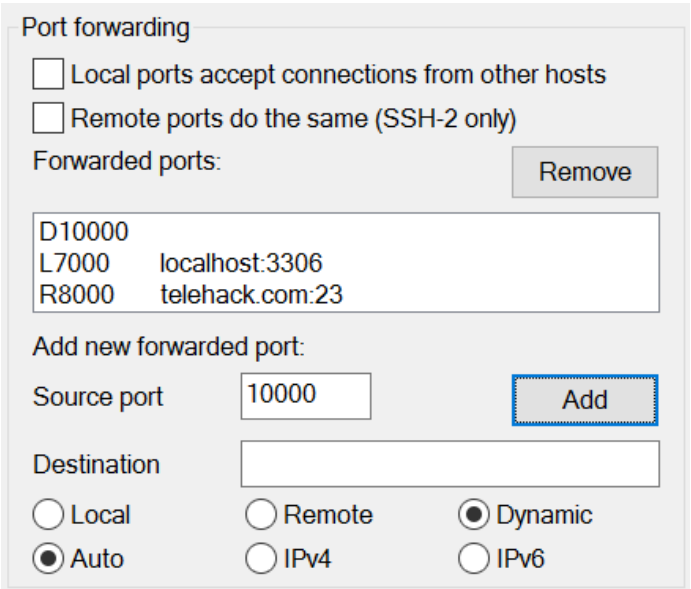
Obrázek 2.22: Základní schéma dynamického tunelu



Zdroj: vlastní zpracování

Vytvoření dynamického tunelu v `PuTTY` je jednoduché, Stačí vyplnit zdrojový port a zvolit typ *Dynamic*. Políčko *Destination* zůstává nevyplněné. Opět nezapomeňte na tlačítko **Add**.

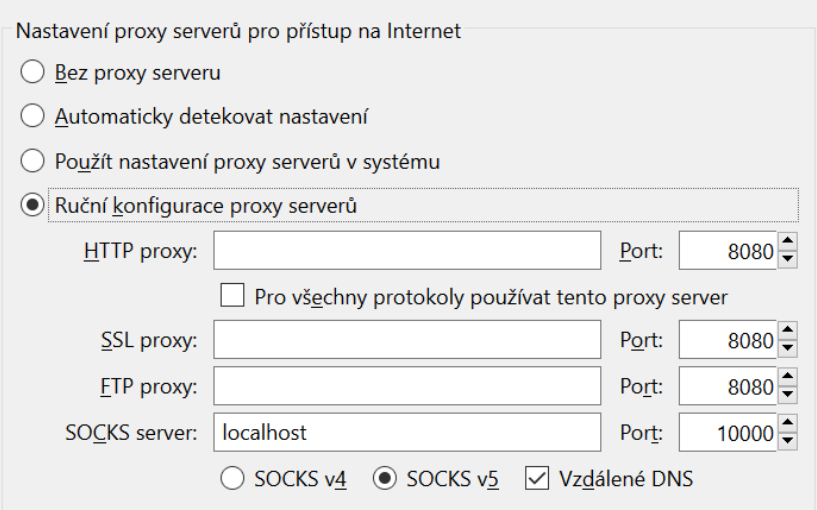
Obrázek 2.23: Vytvoření dynamického tunelu v PuTTY



Zdroj: vlastní zpracování

Jen některé aplikace podporují SOCKS proxy verze 5, příkladem mohou být webové prohlížeče. Pokud v aplikaci podpora pro SOCKS 5 proxy chybí, můžete zkusit proxy programy typu *proxifier* (https://en.wikipedia.org/wiki/Comparison_of_proxifiers). Obrázek ukazuje nastavení ve Firefoxu.

Obrázek 2.24: Nastavení SOCKS proxy verze 5 ve Firefoxu.



Zdroj: vlastní zpracování

2.4.8 Plink – připojení z příkazové řádky

Součástí sady programů *PuTTY* je též program `plink.exe`, který umožňuje z příkazové řádky spustit příkaz na vzdáleném serveru. Všechny uvedené příklady opět předpokládají klíč načtený v paměti. V příkladu spustíme na serveru příkaz `ls -l` (výpis souborů v tzv. dlouhém formátu) a výstup uložíme do lokálního souboru pro další zpracování:

```
plink.exe pavlicek@bis.vse.cz "ls -l" > vypis.txt
```

Program `plink.exe` též umí *vytvořit tunel* bez nutnosti otevírat terminálové spojení. Následují příkazy pro tunely z předchozí kapitoly. Přepínače `-L`, `-R`, `-D` u jednotlivých příkazů (před číslem portu) pochopitelně znamenají typ tunelu (Lokální, vzdálený/Remote resp. Dynamický). Přepínač `-N` znamená nespouštět shell (interpret příkazů).

```
plink -ssh -L 7000:localhost:3306 -N pavlicek@bis.vse.cz
plink -ssh -L 7001:kitscm.vse.cz:3306 -N pavlicek@bis.vse.cz
plink -ssh -R 8000:telehack.com:23 -N pavlicek@bis.vse.cz
plink -ssh -D 10000 -N pavlicek@bis.vse.cz
```

Hlavní využití programu `plink.exe` je v dávkových souborech či při spuštění z jiných programů. Např. *HeidiSQL* (https://www.heidisql.com/, volně stažitelný Windows program s GUI rozhraním pro správu MySQL, MariaDB, Microsoft SQL, PostgreSQL a SQLite databází) vytváří *ssh tunely* právě pomocí programu plink.exe.

2.4.9 Přihlašování přes bastion

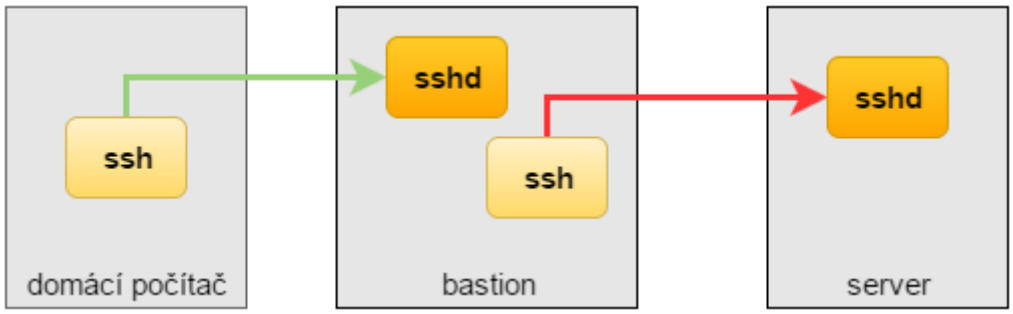
Často není vhodné všechny servery vystavovat na Internet. Můžete vystavit *pouze jeden server* (tzv. **bastion**) a z něho se hlásit na ostatní servery. Existují dva základní přístupy: **řetězení** a **tunelování spojení**. V každém přístupu je *více variant* realizace. Řetězení je *méně bezpečné*.

V logovacích souborech na interním serveru je zapsáno, že se uživatel přihlásil z bastionu. Pro zjištění, z jaké IP adresy se uživatel skutečně přihlásil je proto potřeba projít i logy z bastionu.

2.4.9.1 Řetězení serverů (SSH chaining)

Při řetězení serverů se z domova pomocí klíče *přihlásíte na bastion* a získáte *shell* (příkazovou řádku). Na bastionu spustíte program/příkaz `ssh` (základní použití příkazu *ssh* jako takového je probráno v kapitole 2.5) a připojíte se na interní server.

Obrázek 2.25: Základní schéma využití bastionu

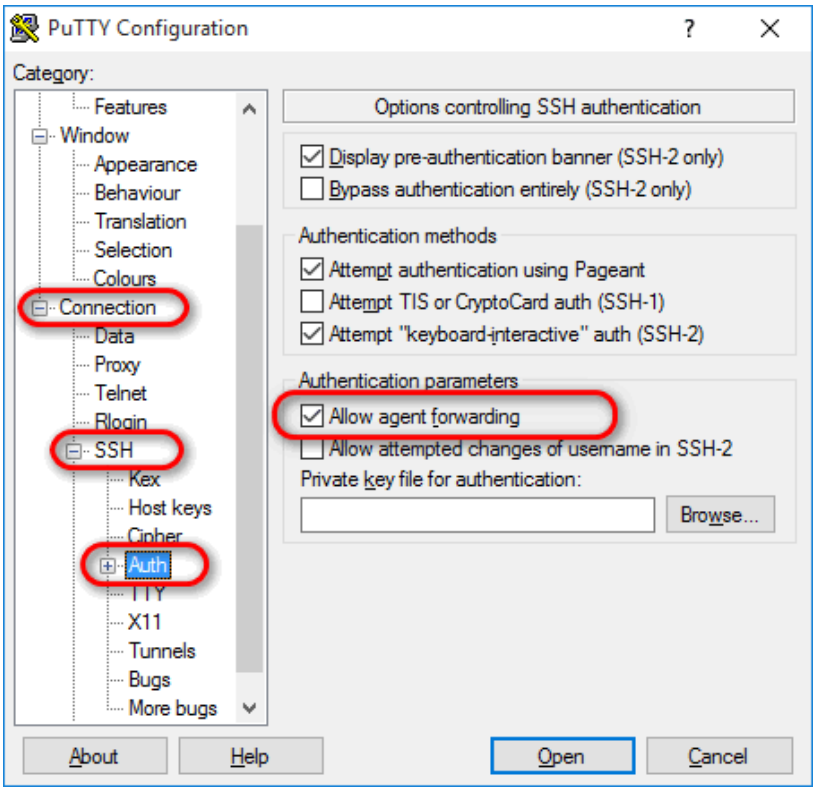


Zdroj: vlastní zpracování

Při řetězení serverů musíte řešit *autentizaci* pro spojení z *bastionu na interní server.* Máte několik možností:

- **Autentizace heslem**. Musí být povoleno. Při častějším připojování to začne uživatelům vadit. Je zde nebezpečí hádání hesel, byť jen od uživatelů s přístupem dovnitř sítě.
- **Druhá dvojice klíčů**. Na bastionu vygenerujete další dvojici soukromý/veřejný klíč a veřejný klíč umístíte na server. Při vytváření spojení z bastionu na interní server zadáte heslo k soukromému klíči či nahrajete klíč do `ssh-agent` (obdoba programu `pageant` v případě *OpenSSH*) spuštěného na bastionu.
- **Soukromý klíč nahrajete na bastion**. Na bastion zkopírujete svůj soukromý klíč z domácího počítače a na server umístíte svůj veřejný klíč. Podobně jako v předchozím případě musíte na bastionu zadat heslo k soukromému klíči. Toto řešení porušuje bezpečnostní pravidlo – jedna dvojice klíčů pro jeden účel.
- **Agent-forwarding**. Na domácím počítači máte soukromý klíč nahraný v agentovi (program `pageant` v případě PuTTY, `ssh-agent` u OpenSSH), veřejný klíč máte na bastionu. Stejný klíč zkopírujete i na interní server. Před vytvořením spojení na bastion zapnete v klientovi na domácím počítači *agent-forwarding* (postup v případě `PuTTY` viz následující obrázek).

Obrázek 2.26: Nastavení agent forwarding v PuTTY



Zdroj: vlastní zpracování

Součástí spojení na bastion poté bude vytvoření *virtuálního ssh agenta* na bastionu, který všechny požadavky na klíč přepošle agentovi na domácím počítači. Při vytvoření spojení z bastionu na interní server se díky forwardingu uživatel ověří na domácím počítači.

Výhodou je, že soukromý klíč nikdy neopustí domácí počítač. *Nevýhodou* naopak je, že správce bastionu (či útočník, který kompromituje bastion) může zneužít agent-forwarding konkrétního uživatele a prostřednictvím jeho klíčů na domácím počítači se připojit na další server. *Obrana* je jednoduchá v případě využití programu `ssh-agent` na domácím počítači: při každém použití soukromého klíče si vyžádá od uživatele potvrzení. `Pageant` z *PuTTY* bohužel tuto funkcionalitu (zatím) nemá.

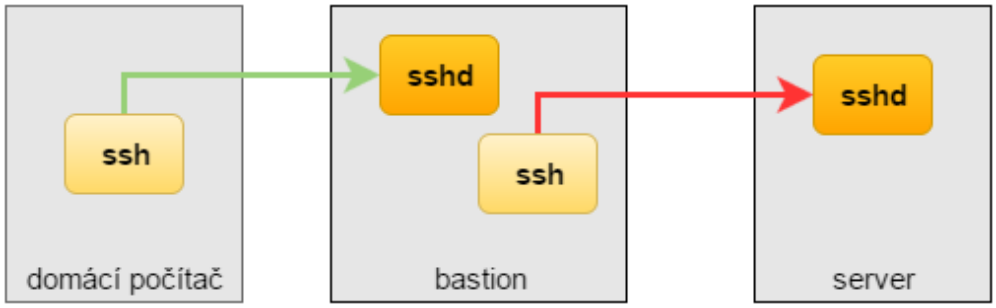
Všechny varianty řetězení mají několik *bezpečnostních nedostatků*. Pokud útočník kompromituje bastion, tak se může hlásit na interní servery pod cizí identitou. Útočník též může sledovat celou komunikaci, neboť při řetězení se na bastionu údaje předávají z jednoho zašifrovaného spojení do druhého. Postup útočníka je poměrně jednoduchý: upraví ssh klienta či ssh server a přenášené údaje loguje (stačí zapnout ladící volby v konfiguraci).

Dalším problémem je, že uživatel má plnohodnotný *přístup k shellu bastionu*. Díky tomu může na bastionu spouštět různé programy, může si na něm uschovávat soubory – tím roste nebezpečí úspěšného útoku na bastion i zvyšují nároky na správu bastionu.

2.4.9.2 Tunelování SSH spojení

Alternativou k řetězení je **tunelování**, které umožňuje bezpečné připojení na server i v případě kompromitovaného bastionu. Základní schéma tohoto způsobu znázorňuje následující Obrázek 2.27

Obrázek 2.27: Základní schéma tunelování SSH spojení

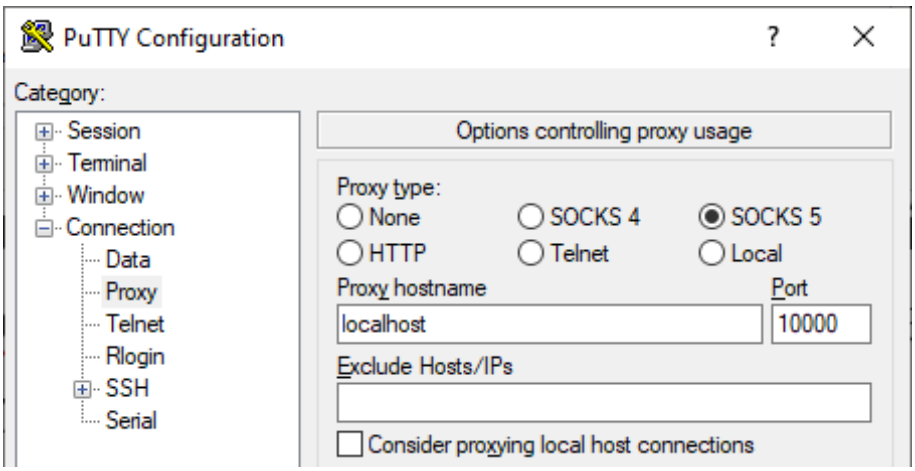


Zdroj: vlastní zpracování

Při tunelování vytvoříte *SSH spojení na bastion* a *v něm tunel*. *Druhé SSH* spojení bude ze stanice do tunelu a z bastionu bude pokračovat dál na server. U tohoto typu spojení se bastion označuje **Jump host** či **SSH proxy server**. Na *bastionu není potřeba* vytvářet spojení s emulací terminálu, uživatel nemusí mít možnost spouštění programů. (Omezení účtu jen na tunelování viz `man sshd`, sekce `authorized_keys`.) Na bastionu i na serveru musí mít uživatel veřejný klíč. Existuje *opět několik variant* realizace principu.

- **Klasický lokální tunel**. Připojíte se na bastion a vytvoříte klasický lokální tunel. Ve druhém ssh spojení se připojíte na lokální konec tunelu. V případě většího počtu interních serverů je správa tunelů obtížná.
- **Dynamická proxy**. Opět máte dvě spojení. V prvním se připojíte na bastion a vytvoříte dynamický tunel (SOCKS proxy) např. na portu 10000. Ve konfiguraci spojení na server nastavíte v *Connection -> Proxy* typ proxy na Socks 5, hostname na localhost a port na 10000. Nastavení pro PuTTY viz následující Obrázek 2.28.
- **Proxy command**. V obou předchozích způsobech uživatel musí spustit dva programy (dvakrát Putty). Vytvoření tunelu a navázání spojení lze zapsat *do jedné konfigurace* (sezení), viz následující příklad, popsaný v samostatné následující kapitole.

Obrázek 2.28: Nastavení dynamické proxy v PuTTY



Zdroj: vlastní zpracování

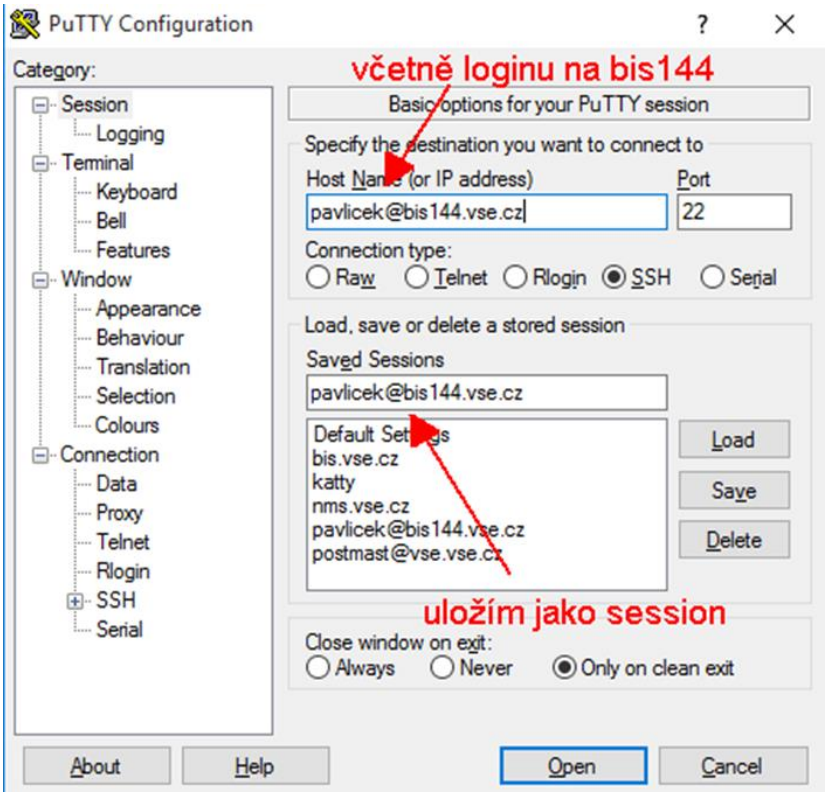
2.4.9.3 Proxy command v Putty

Ze svého počítače se chci připojit na server *bis144.vse.cz*, který je dostupný pouze přes server *bis.vse.cz* (tj. bastion). Na obou serverech mám již veřejný klíč, je uložen standardně v `.ssh/authorized_keys`. Na stanici používám program `pageant` a mám v něm nahrán příslušný klíč (či dva klíče, pokud se liší).

Nejprve v programu `putty` do *Host Name* zadám jméno cílového serveru včetně přihlašovacího jména na tomto serveru, tj. v našem příkladě zadám *pavlicek@bis144.vse.cz*. Viz následující Obrázek 2.29.

Poté v *Connection -> Proxy* nastavím typ proxy na *local* a dále do políčka *local proxy command* doplním příkaz `plink pavlicek@bis.vse.cz -agent -nc %host:%port\n`. Ukázka druhé části nastavení viz Obrázek 2.30.

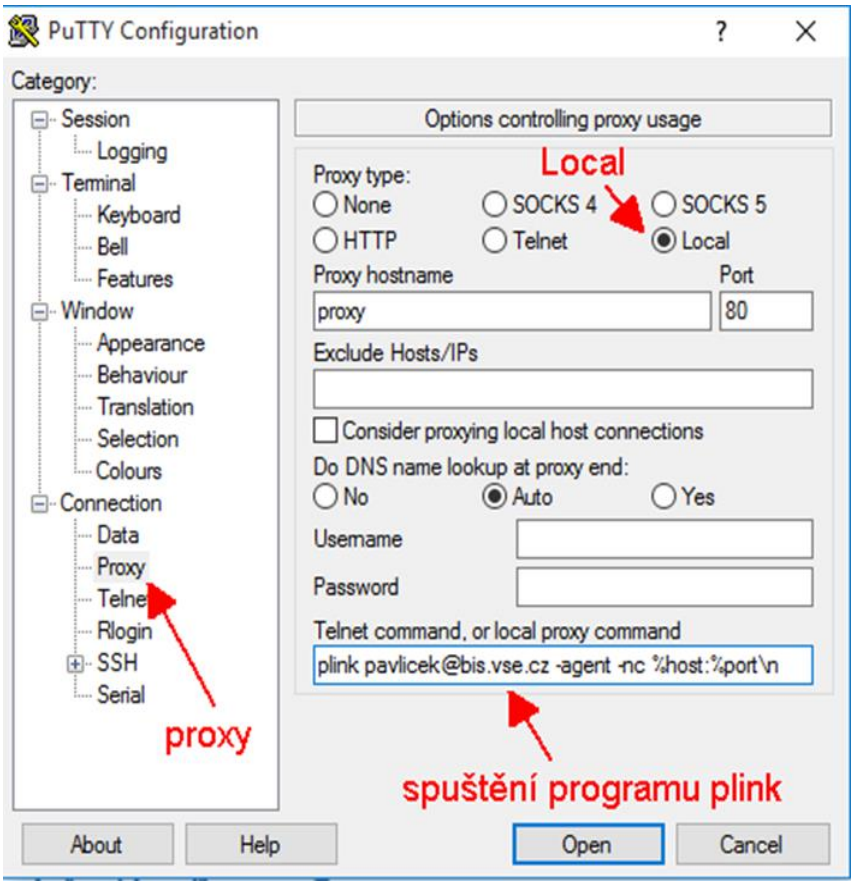
Obrázek 2.29: Proxy command v PuTTY (1. část nastavení)



Zdroj: vlastní zpracování

Před navázáním spojení na server *bis144.vse.cz* se spustí program `plink`, který vytvoří spojení na *bis.vse.cz* s uživatelským jménem *pavlicek* a vytvoří potřebný *proxy tunel* (parametr `-nc`) na *bis144.vse.cz* a port 22 (zadáno odkazem na hodnoty vyplněné v položce *Host Name* a port na úvodní stránce konfigurace). Parametr `-agent` u programu plink určuje *způsob autentizace* – bude číst klíč nahraný do paměti programem `pageant`. V novějších verzích *PuTTY* již není nutné tento parametr zadávat. Další informace o proxy najdete v dokumentaci.

Obrázek 2.30: Proxy command v PuTTY (2. část nastavení)



Zdroj: vlastní zpracování

2.5 Použití SSH z Linuxu či macOS na straně klienta

Pokud na svém PC (notebooku) používáte *macOS* nebo *Linux* (nebo o instalaci *Linuxu* uvažujete), je tato podkapitola určena právě vám. V *Linuxu* (nebo UNIXu) i na *macOS* je obvykle součástí základní instalace také klientská část *OpenSSH* – řádkové příkazy pro přihlašování na vzdálený server pomocí protokolu SSH. Příkazy se používají v *terminálovém okně* (příkazovém řádku), příslušná *aplikace pro virtuální terminál* se může jmenovat např. `Terminal, Konsole, rxvt, Xterm, LXTerminal, ETerm, Guake` – závisí na konkrétním operačním systému a také na použité grafické nadstavbě (např. v KDE zpravidla `Konsole`, v MATE to je `mate-terminal` atd.).

Příkazy můžete zadávat i v programu *PuTTY* – přihlásíte se např. na server *bis.vse.cz* a z něho se budete hlásit na další server. V současnosti existuje též oficiální verze celé sady programů *PuTTY pro Linux*, port pro *macOS* zatím nebyl dokončen, ale pro *macOS* existují jiné GUI programy, pokud by vám řádkový program `ssh` na straně klienta nevyhovoval.

Popis použití `ssh` na straně klienta bude *méně podrobný*, je to doplněk k předchozí kapitole (kde jsme se věnovali PuTTY ve Windows), zaměříme se na základní operace v programu `ssh`, generování klíčů (`ssh-keygen`), správa klíčů v paměti (již v předchozí kapitole zmíněný `ssh-agent` a `ssh-add` pro přidávání klíčů) a kopírování souborů v rámci SSH (`scp` a `sftp`).

2.5.1 Základy používání programu ssh

Nejprve pochopitelně spusťte příslušnou *aplikaci pro virtuální terminál*, její jméno se liší dle operačního systému či distribuce (viz výše), ale spuštění z GUI rozhraní by neměl být žádný problém. Ve většině distribucí také najdete v oficiálních repositářích hned několik alternativ, pokud by vám standardně nainstalovaná z jakéhokoliv důvodu nevyhovovala.

Práce s *historií* příkazů a *dokončování příkazů* je stejná jako na serveru *bis.vse.cz*. Začneme (stejně jako v *PuTTY*) přihlášením pomocí uživatelského jména a hesla. Uživatelské jméno se zadává před jménem serveru oddělené znakem @. Pro jistotu připomínáme, že příkazy nutno zadat malými písmeny, resp. na rozdíl od Windows se obecně (v příkazech i jménech souborů) rozlišují velká a malá písmena.

```
ssh pavlicek@bis.vse.cz
```

Při prvním přihlášení se objeví dotaz na *ověření platnosti otisku* veřejného klíče serveru (stejně jako v PuTTY, kde je podrobně vysvětleno, *proč* a *jak* byste otisk měli ověřit, viz kapitola 2.4.2 výše).

```
The authenticity of host 'bis.vse.cz (146.102.18.87)' can't be established.
RSA key fingerprint is 30:14:af:4b:64:89:4f:42:86:aa:92:63:98:22:09:ad.
Are you sure you want to continue connecting (yes/no)?
```

Po ověření a zadání *yes* se veřejný klíč uloží do souboru `.ssh/known_hosts` (což se vám zobrazí jako upozornění) a poté se objeví dotaz na heslo:

```
Warning: Permanently added 'bis.vse.cz,146.102.18.87' (RSA)
to the list of known hosts.
pavlicek@bis.vse.cz's password:
```

Při příštím přihlášení se dotaz na potvrzení klíče *již neobjeví*. Pokud server změní svůj veřejný klíč, tak se nepřihlásíte – musíte v editoru otevřít soubor `.ssh/known_hosts`, smazat údaj o klíči pro server a až poté se můžete přihlásit s ověřením nového veřejného klíče.

Jméno uživatele lze zadat také pomocí parametru (přepínače) `-l` (malé „el“). Pokud se jméno nezadá, tak se použije uživatelské jméno používané v lokálním operačním systému.

```
# alternativní způsob zadání jména uživatele
ssh -l pavlicek bis.vse.cz

# bez zadání jména uživatele, použije se jméno v lokálním OS
ssh bis.vse.cz
```

Při problémech s připojením zadejte parametr `-v` (verbose) a budou se zobrazovat podrobné informace o připojení, z kterých by měla být zřejmá příčina. Přepínač lze zapsat i dvakrát nebo třikrát za sebou, pak jsou vypisované informace ještě podrobnější.

```
# vypisování podrobných informací o připojení (pro řešení problémů)
ssh -v pavlicek@bis.vse.cz

# nejpodrobnější informace o připojení
ssh -vvv pavlicek@bis.vse.cz
```

2.5.2 Konfigurace klienta .ssh/config

Parametry konkrétních spojení lze zapsat do souboru `.ssh/config`, tj. v domovském adresáři na *Vašem notebooku* v podadresáři `.ssh` vytvoříte soubor `config`. V ukázce definujeme pro *bis.vse.cz* implicitní uživatelské jméno (tj. poté ho nemusím zadávat v každém příkazu):

```
Host bis.vse.cz
   User pavlicek
```

Obdobně si lze zkrátit i *bis.vse.cz* na *bis:* do konfiguračního souboru doplníme Hostname:

```
Host bis
   Hostname bis.vse.cz
   User pavlicek
```

V následující ukázce se při připojení ke všem serverům se jménem začínajícím řetězcem *raspberry* doplní uživatelské jméno *pi* a nastaví parametry pro udržování spojení.

```
Host raspberry*
   User pi
   ServerAliveInterval 60
   ServerAliveCountMax 10
   TCPKeepAlive no
```

Podrobný popis možností souboru .ssh/config získáte v manuálové stránce po zadání příkazu:

```
man ssh_config
```

2.5.3 Vygenerování a zkopírování klíče

Uživatel si vygeneruje svoji *dvojici soukromý/veřejný klíč* pomocí programu `ssh-keygen`. Bez zadání dalších parametrů se generují RSA klíče o délce 2048 bitů (v některých operačních systémech může být odlišné nastavení). Doporučuji ale generovat klíče typu *ed25519*, které jsou kratší a přitom bezpečnější, ale na některých starších systémech nemusí být podporovány. Pokud vygenerujete klíč RSA o délce 2048, tak to není chyba.

Program se zeptá na jméno souboru pro uložení soukromého klíče (doporučuji neměnit) a na heslovou frázi k soukromému klíči.

```
ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/pavlicek/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/pavlicek/.ssh/id_ed25519.
Your public key has been saved in /home/pavlicek/.ssh/id_ed25519.pub.
The key fingerprint is:
SHA256:SoJ8f8yuyUu9Hzghpops8tdvFoe/mBmhPlemh4pos0A pavlicek@localhost
```

Soubor `.ssh/id_ed25519` obsahuje soukromý klíč i veřejnou část klíče a je chráněný heslovou frází. Vznikne i druhý soubor `.ssh/id_ed25519.pub`, který obsahuje jen veřejný klíč ve formátu vhodném pro vložení do `authorized_keys` (tedy celý klíč na jednom dlouhém řádku, jak již bylo vysvětleno při popisu generování klíčů v *PuTTY*).

Program `ssh-keygen` umí vygenerovat i další typy klíčů, lze zadat délky klíčů, změnit heslo klíčů. Přehled všech možností získáte v manuálových stránkách: `man ssh-keygen`.

Před přihlášením pomocí klíče *musíte* zkopírovat veřejný klíč ze souboru `.ssh/id_ed25519.pub` (nebo ze souboru `.ssh/id_rsa.pub` pokud jste zvolili RSA) do souboru `.ssh/authorized_keys` na cílovém serveru. Do cílového souboru můžete klíč vložit více způsoby, lze použít pomocný skript `ssh-copy-id`, který zkopíruje veřejný klíč na cílový server (tento skript se snaží zabránit vzniku některých typických chyb uživatelů).

```
ssh-copy-id pavlicek@bis.vse.cz
```

2.5.4 Přihlašování pomocí klíče

Pokud existuje soubor `.ssh/id_ed25519` (nebo `.ssh/id_rsa` nebo `.ssh/id_ecdsa`: podle toho, jaký algoritmus jste si při generování klíče zvolili),[136] tak program `ssh` nejdříve zjišťuje, zda se lze pomocí tohoto klíče přihlásit na vzdálený server. Pokud ano, tak se následně zeptá na heslovou frázi k soukromému klíči:

```
ssh pavlicek@bis.vse.cz
Enter passphrase for key '/home/pavlicek/.ssh/id_ed25519':
```

Podobně jako u *PuTTY* můžete klíč (více klíčů) nahrát do paměti a následně při každém přihlášení nemusíte zadávat heslo. V textovém rozhraní se pro tyto účely používá program `ssh-agent` (případně `gpg-agent` se stejným API).[137]

Program `ssh-agent` se obvykle spouští následujícím způsobem:

```
# na vlastním notebooku, ne na bis.vse.cz!!!
eval "$(ssh-agent -s)"
```

SSH klíč poté přidáte pomocí programu `ssh-add`:

```
# na vlastním notebooku, ne na bis.vse.cz!!!
ssh-add
Enter passphrase for /home/pavlicek/.ssh/id_rsa:
Identity added: /home/pavlicek/.ssh/id_rsa (/home/pavlicek/.ssh/id_rsa)
```

Nyní se již můžete přihlásit bez zadávání hesla (`ssh pavlicek@bis.vse.cz`).

Můžete též *jednorázově spouštět programy* na vzdáleném serveru a výstup si *ukládat lokálně*, (v případě *PuTTY* jsme prakticky stejným způsobem použili řádkový program `plink`, viz kapitola 2.4.8). V příkladu vypíšeme soubory v tzv. dlouhém formátu a výstup uložíme do lokálního souboru.

```
ssh pavlicek@bis.vse.cz 'ls -l' > bis_soubory.txt
```

[136] Doporučujeme „standardní“ jména, ale obecně soukromý klíč může být v jiném souboru, při spuštění `ssh` zadáte cestu pomocí parametru `-i`, např. `ssh -i .ssh/druhy_klic pavlicek@bis.vse.cz`

[137] Rozhraní programu `ssh-agent` se též používá při *agent forwarding*. Grafické verze programu pro správu klíčů a hesel mají obvykle v názvu *keychain*.

2.5.5 Kopírování souborů: programy scp a sftp

Součástí *OpenSSH* jsou též programy `scp` a `sftp` pro kopírování souborů, které mají podobné ovládání jako dříve popsané programy `pscp` a `psftp` z sady programů *PuTTY*. Následující syntaxe obsahuje nejčastěji používané parametry programu `scp` (úplný přehled viz manuálové stránky pro tento program, tedy `man scp`):

```
scp [-3pr] [[user@]host1:]file1 [[user@]host2:]file2
```

`-3` kopíruje se mezi dvěma vzdálenými servery,
`-p` snaží se u kopírovaných souborů nastavit čas a další atributy dle zdroje,
`-r` rekurzivní kopírování, kopíruje se obsah podadresářů,
`[[user@]host1:]file1` odkud se kopíruje, lokální či vzdálený soubor/adresář,
`[[user@]host2:]file2` kam se kopíruje, lokální či vzdálený soubor/adresář.

Následují dva jednoduché příklady použití programu `scp`:

```
# zkopírování souboru seznam ze serveru bis.vse.cz do lokálního adresáře
scp pavlicek@bis.vse.cz:seznam .
# zkopírování lokálního adresáře obrazky
#   ==> na server bis.vse.cz do domovského adresáře
scp -r obrazky pavlicek@bis.vse.cz:
```

Program `sftp` používá podobné ovládání jako řádkový klient pro *protokol ftp*. Dostupné povely a jejich popis zjistíte z manuálových stránek (základní nápovědu samozřejmě získáte přímo v programu):

```
man sftp
```

Uživatelé s grafickým rozhraním na lokálním počítači mají většinou k dispozici nějakou *GUI aplikaci* pro zobrazení adresářů a kopírování souborů přes SSH. V případě Linuxu se výchozí aplikace tohoto typu může lišit dle distribuce a též dle použitého desktopového prostředí (např. KDE x MATE x Xfce). S pomocí *Wine* lze použít též *WinSCP* v portable verzi (stručný popis tohoto programu je na konci kapitoly 2.4.6).

2.5.6 Tunely

SSH klient z OpenSSH podporuje *všechny typy tunelů*: lokální (Local, Forward), vzdálené (Remote, Reverse), dynamické (Dynamic, Proxy) i X11 tunely pro spouštění grafických aplikací na vzdáleném počítači. Následuje syntaxe programu `ssh` související s tunely:

```
ssh [-fNXY] [-D port] [-L port:host:hostport] [-R port:host:hostport]
[user@]hostname [command]
```

`-f` vytvoření tunelu na pozadí, používá se např. pro spuštění grafické aplikace,
`-N` nespustí se vzdálený příkaz, tj. pouze tunelování,
`-X` vytvoření tunelu pro grafické (X11) aplikace,
`-Y` bezpečná varianta k `-X`,
`-D port` vytvoření dynamického tunelu,
`-L port:host:hostport` vytvoření lokálního tunelu,
`-R port:host:hostport` vytvoření vzdáleného tunelu.

Následují *dva příklady* na vytvoření *lokálního tunelu* do databáze MySQL (princip obou variant je podrobně v kap. 2.4.7.1, zejména viz Obrázek 2.17). V *prvním příkladu* vznikne tunel, který bude poslouchat na portu 7000 na lokálním počítači a provoz bude tunelovat do databáze MySQL na *bis.vse.cz* (což je localhost na druhé straně tunelu). *Druhý* lokální tunel přesměrovává provoz z lokálního portu 7100 na MySQL server na počítači *kitscm.vse.cz* (pokud chceme použít *oba tunely současně*, musíme použít dvě různá čísla portů).

```
ssh -L 7000:localhost:3306 -N pavlicek@bis.vse.cz
ssh -L 7100:kitscm.vse.cz:3306 -N pavlicek@bis.vse.cz
```

Vzdálený (Remote, Reverse) tunel poslouchá na portu 8000 na serveru *bis.vse.cz* (druhý konec SSH spojení) a přesměrovává provoz přes lokální stanici na server *telehack.com* na port 23 (vysvětlení tohoto typu tunelů viz kapitola 2.4.7.2).

```
ssh -R 8000:telehack.com:23 -N pavlicek@bis.vse.cz
```

V tomto případě je vhodné *vynechat* parametr `-N`, abyste se přihlásili na *bis.vse.cz* a *současně* se vytvořil tunel. Po přihlášení se můžete připojit na vstupní bod tunelu na portu 8000 vytvořený na *bis.vse.cz* příkazem `telnet localhost 8000` (raději znova připomínáme, že v případě programu/příkazu `telnet` je před číslem portu *mezera*, nikoli dvojtečka).

```
ssh -R 8000:telehack.com:23 pavlicek@bis.vse.cz
Last login: Sat Oct 17 11:37:30 2020 from 2001:718:1e02:42::38
telnet localhost 8000
Connected to localhost.
...
```

Dynamický tunel na bis.vse.cz bude přijímat SOCKS 5 připojení na lokální stanici na portu 10000 (vysvětlení dynamických tunelů viz kapitola 2.4.7.3):

```
ssh -D 10000 -N pavlicek@bis.vse.cz
```

2.5.7 Přihlášení skrz bastion – jump host

V kapitole 2.4.9 se popisuje průchod skrz **bastion** pomocí *řetězení* (čtyři varianty) a *tunelování* (tři varianty). *OpenSSH klient* používá stejné principy, zde popíšeme jen proxy-command tunelování (*Jump Host*) v této a *agent forwarding* v následující podkapitole.

V *OpenSSH* od verze 7.3 je přepínač `-J` pro zadání průchozího serveru. Lze zadat i uživatelské jméno (a případně také čísla portů). Následující příkaz přihlásí uživatele na server *bis144.vse.cz* skrz *bis.vse.cz*:

```
ssh -J pavlicek@bis.vse.cz pavlicek@bis144.vse.cz
```

Z jaké IP adresy jste přihlášeni můžete zjistit pomocí příkazu `who` na cílovém serveru. Parametr lze zapsat i do konfiguračního souboru `.ssh/config`

```
Host bis144.vse.cz
  User pavlicek
  ProxyJump pavlicek@bis.vse.cz
```

a poté se přihlásit pomocí `ssh bis144.vse.cz`.

`ProxyJump` je zjednodušeným zápisem mnohem staršího a obecnějšího `ProxyCommand`, který tuneluje *stdin* (standardní vstup) a *stdout* (standardní výstup) skrz proxy servery a bastiony.

V následujícím zápisu do konfiguračního souboru `.ssh/config` na notebooku je pro cílový server *bis144.vse.cz* nadefinován přístup skrz bastion *bis.vse.cz*:

```
Host bis144.vse.cz
  User pavlicek
  ProxyCommand ssh pavlicek@bis.vse.cz -W %h:%p
```

Za `%h` uvedeného u parametru `-W` se doplní *hostname* z definice Host, popř. z parametru Hostname. Za `%p` se doplní číslo portu (výchozí je port 22). V ukázce jsou též uživatelská jména: parametr `User` je pro cílový server, v `ProxyCommand` je uvedeno uživatelské jméno pro *bis.vse.cz*. Z notebooku se na *bis144.vse.cz* přihlásím příkazem (předpokládám, že klíče jsou v `ssh-agent` na notebooku):

```
ssh bis144.vse.cz
```

2.5.8 Přihlášení skrz bastion – agent forward

Agent forwarding je jednodušší na nastavení, ale méně bezpečný. Pokud se na bastion dostane hacker či ho spravuje zlomyslný administrátor, tak mohou zneužít Váš klíč k přihlášení na další servery.

Agent forwarding se nastavuje na notebooku v souboru `.ssh/config` parametrem ForwardAgent:

```
Host bastion.vse.cz
   ForwardAgent yes
```

Konkrétní příklad včetně nastavení uživatelského jména:

```
Host bis.vse.cz
   User pavlicek
   ForwardAgent yes
```

Po nastavení se nejdříve přihlásíte na bastion (na *bis.vse.cz*). A zde na příkazové řádce se přihlásíte na cílový server (např. na *bis144.vse.cz*). Dotaz na klíč bude z bastionu (z *bis.vse.cz*) přesměrován na `ssh-agent` běžící na notebooku.

2.6 Základy konfigurace SSH serveru (openssh)

Sice existuje více implementací protokolu SSH, v Unixu včetně Linuxu má v současné době dominantní postavení balík *OpenSSH* distribuovaný pod licencí typu BSD. Hlavní klientské programy (`ssh`, `ssh-agent`, `ssh-keygen`) byly stručně popsány v předchozí kapitole. Zde stručně popíšeme `sshd` – **SSH server daemon** – včetně zabezpečení konfigurace.

Podrobnou dokumentaci získáte z manuálových stránek pomocí příkazů:

```
man sshd
man sshd_config
```

2.6.1 Daemon sshd

Daemon `sshd` standardně poslouchá na portu 22, což lze ověřit příkazem `netstat -atn`, který vypíše otevřená TCP spojení (pomocí `grep` omezíme výpis na port 22):

```
netstat -atn | grep ':22'
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 146.102.18.87:22        146.102.93.146:50387    ESTABLISHED
tcp        0      0 146.102.18.87:22        217.112.172.43:58602    ESTABLISHED
tcp        0      0 146.102.18.87:22        146.102.93.145:60033    ESTABLISHED
tcp        0      0 146.102.18.87:22        217.112.172.43:58604    ESTABLISHED
```

Ve výpisu je vidět, že naslouchá na portu 22, a dále že jsou otevřená spojení z 3 různých IP adres. Z IP adresy 217.112.172.43 jsou otevřena dvě souběžná spojení.

Program `ps -ef` vypisuje spuštěné procesy, pomocí `grep` omezím výpis na daemon sshd:

```
ps -ef | grep sshd
root      6111     1  0 Sep21 ?        00:00:07 /usr/sbin/sshd -D
root     30035  6111  0 12:26 ?        00:00:00 sshd: pavlicek [priv]
pavlicek 30046 30035  0 12:26 ?        00:00:00 sshd: pavlicek@pts/1
root     14569  6111  0 14:30 ?        00:00:00 sshd: test99 [priv]
test99   14575 14569  0 14:30 ?        00:00:00 sshd: test99@pts/14
root     14697  6111  0 14:35 ?        00:00:00 sshd: test99 [priv]
test99   14703 14697  0 14:35 ?        00:00:00 sshd: test99@pts/15
root     21220  6111  0 Sep21 ?        00:00:00 sshd: xabcd01 [priv]
xabcd01  21230 21220  0 Sep21 ?        00:00:01 sshd: xabcd01@pts/0,pts/5
```

Při nastartování serveru se spustí program **/usr/sbin/sshd**, a tím se vytvoří proces (zde proces č. `6111`). Tento proces poslouchá na portu **22** a čeká na navázání spojení od nějakého klienta. Pro procesy bez vlastního uživatelského rozhraní se používá označení **daemon**.

Když se připojí uživatel, tak se pro toto spojení vytvoří dva procesy. První proces vytvoří daemon a tento proces běží též pod uživatelem **root** (u spojení uživatele *pavlicek* je to proces `30035`). Tento proces vytváří a spravuje *transportní vrstva* protokolu SSH. Po vytvoření transportní vrstvy a autentizaci uživatele spustí další proces, nyní **s právy přihlášeného uživatele** (proces `30046` pro spojení uživatele *pavlicek*). Tento proces obsluhuje jednotlivá spojení (jednotlivé kanály), např. spouští shell (příkazovou řádku) konkrétního uživatele. Samostatné procesy umožňují minimalizovat škody v případě úspěšného útoku na implementaci `sshd`.

2.6.2 Jak upravit konfiguraci pro sshd

Konfigurace daemonu sshd je v souboru `/etc/ssh/sshd_config`. Úpravy může provádět pouze uživatel *root*. Na svém serveru můžete získat toto oprávnění příkazem `sudo -i`.

U následujících příkazů předpokládám, že jste tento příkaz úspěšně spustili, tj. že máte oprávnění uživatele *root*. Před změnami si udělejte *zálohu*, např. příkazem

```
cp /etc/ssh/sshd_config /etc/ssh/sshd_config_zaloha
```

Pro editaci použijte editor *nano* (jednodušší pro začátečníky) či editor *vim* (mnohem více možností, barevná syntaxe).

```
nano /etc/ssh/sshd_config
# nebo
vi /etc/ssh/sshd_config
```

Po úpravě a uložení souboru zkontrolujte syntaxi úprav pomocí příkazu `sshd -t`.
V následující ukázce se vypíší chyby:

```
sshd -t
/etc/ssh/sshd_config: line 105: Bad configuration option: llowUsers
/etc/ssh/sshd_config line 105: Directive 'llowUsers' is not allowed
within a Match block
```

Pokud nejsou *syntaktické chyby*, tak restartujte `sshd` pomocí příkazu

```
systemctl restart sshd
```

Nezavírejte otevřené ssh spojení! I když příkaz `sshd -t` nevypsal žádné chyby syntaxe, tak **stále mohou být v** `sshd_config` **chyby**, které způsobí, že se *nebudete moci přihlásit*. Např. omylem zakážete přihlašování heslem i klíčem.

Otevřete nové ssh spojení a ověřte, že se můžete úspěšně přihlásit. Pokud se nemůžete přihlásit, tak zkontrolujte, zda je spuštěn proces `sshd` pomocí příkazu:

```
ps -ef | grep sshd
root      3680     1  0 17:12 ?        00:00:00 sshd: root@pts/0
root      3824     1  0 17:35 ?        00:00:00 /usr/sbin/sshd -D
root      3876  3689  0 17:41 pts/0    00:00:00 grep --color=auto sshd
```

Ve výpisu musíte najít `/usr/sbin/sshd -D`. Pokud tento řádek nenajdete, tak se při restartu nespustil proces `sshd`. Důvod najdete ke konci logu `/var/log/daemon.log`.

```
less /var/log/daemon.log
```

Pokud proces `/usr/sbin/sshd` běží, tak důvod, proč se nemůžete přihlásit najdete ke konci souboru `/var/log/auth.log`.

```
less /var/log/auth.log
```

2.6.3 Struktura konfiguračního souboru, podmíněné sekce

Parametry v konfiguračním souboru se skládají z klíčového slova a jedné či více hodnot, na řádku může být pouze jedno klíčové slovo. *Komentáře* začínají znakem `#`. V následující ukázce je začátek konfiguračního souboru:

```
# Package generated configuration file
# See the sshd_config(5) manpage for details
Port 22
AddressFamily inet
```

Na začátku konfiguračního souboru `/etc/ssh/sshd_config` se uvádějí **parametry s globální platností**. Za nimi následují **podmíněné (Match) bloky** – zadává se podmínka a na dalších řádcích parametry pro spojení odpovídají podmínce. Blok je ukončen buď následujícím podmíněným blokem či koncem souboru.

```
...
PermitRootLogin no
...
Match address 146.102.0.0/16,127.0.0.1/32
   PermitRootLogin without-password
```

V *globální části konfiguračního souboru* je zakázáno přihlášení uživatele *root*. Podmínkou v sekci je spojení ze školních IP adres (146.102.0.0/16) či z přímo z vlastního serveru (127.0.0.1). Pokud je podmínka splněna, tak se povolí přihlášení uživatele **root** pomocí klíče.

Vedle IP adres klienta se v podmínkách často používá uživatelské jméno (*user*) či skupina (*group*) anebo obojí, např. pokud část uživatelů nemá mít možnost vytvářet tunely či pokud některý uživatel může pouze kopírovat soubory.

2.6.4 Zabezpečení SSH serveru

Následují doporučení pro zabezpečení SSH serveru, rozdělená do několika skupin:

- zákaz přihlášení pomocí hesla,
- zákaz přihlášení uživatele root,
- omezení tunelů,
- omezení uživatelů,
- naslouchání na jiném portu (kontroverzní),
- omezení počtu pokusů o přihlášení.

2.6.4.1 Zákaz přihlášení pomocí hesla

Pokud mají uživatelé ověřeno přihlašování klíčem, tak je vhodné zakázat přihlašování pomocí hesla. Začíná-li řádek `#`, tedy `#PasswordAuthentication no`, znamená to implicitně *yes*! Častý omyl. Příslušný řádek proto změňte na `PasswordAuthentication no`. Následující pokusy o přihlášení pomocí hesla poté skončí chybou, viz následující ukázka:

```
ssh bis145.vse.cz
Permission denied (publickey).
```

Je vhodné též nastavit `ChallengeResponseAuthentication no`. Tato metoda se používá např. při vícefaktorové autentizaci pro zadání hodnoty druhého faktoru. V závislosti na konfiguraci knihovny PAM může totiž umožnit přihlašování pomocí hesla.

Na *bis.vse.cz* je povoleno přihlašování heslem pouze ze školní sítě. V konfiguraci je využita *podmíněná sekce* (a jak si určitě pamatuje z předchozí části, tak podmíněné sekce musí být až na konci konfiguračního souboru):

```
# ... část globální konfigurace
PasswordAuthentication no
ChallengeResponseAuthentication no
# ... další část globální konfigurace
#
Match address 146.102.0.0/16
    PasswordAuthentication yes
```

2.6.4.2 Zákaz přihlášení uživatele root

Pokud se správci mohou stát privilegovanými uživateli pomocí příkazu `sudo`, tak ve většině případů neexistují důvody pro povolení přihlašování pod uživatelem *root*. Výjimkou mohou být nástroje po vzdálenou administraci serveru jako je *Ansible*. Ale i v těchto případech byste měli omezit přihlášení uživatele *root* pouze z vybraných IP adres (příklad výše).

Parametr **PermitRootLogin** určuje možnosti přihlášení uživatele root. Možné hodnoty:

`yes`	může se přihlašovat bez omezení,
`no`	nemůže se přihlásit,
`without-password`	nemůže se přihlásit heslem,
`forced-commands-only`	lze spustit pouze vybrané příkazy, např. zálohování.

2.6.4.3 Omezení tunelů

Na serveru obvykle nepotřebujete spouštět Xkové (grafické) aplikace, a proto zakažte X11Forwarding: `X11Forwarding no`. Na serveru často nepotřebujete ani ostatní typy tunelů. Ty zakážete nastavením `PermitTunnel no`.

Vhodné je to na serverech, kam se přihlašuje větší množství uživatelů. Zákazem tunelů omezíte možnosti zneužití serveru jako proxy např. k anonymnímu surfování či ke stahování torrentů. Pro vybrané uživatele může povolit tunely v podmíněném (Match) bloku.

2.6.4.4 Omezení uživatelů (AllowUsers, AllowGroups)

Pomocí parametrů **AllowUsers**, **AllowGroups**, **DenyUsers** a **DenyGroups** můžete omezit přihlášení na konkrétní uživatele či uživatelské účty. **Doporučuji použít jen jeden z těchto parametrů:** před použitím dvou či více si předem pečlivě prostudujte v dokumentaci, jak se vzájemně ovlivňují (logická operace AND).

Osobně preferuji použití *AllowGroups* a to včetně vytvoření speciální skupiny uživatelů, kteří se mohou hlásit. Na *bis.vse.cz* je následující řádek:

```
AllowGroups root sudo users
```

To znamená, že se mohou hlásit členové skupin `root` (obsahuje uživatele *root*), `sudo` (uživatelé s možností stát se privilegovaní pomocí *příkazu* `sudo`) a členové skupiny `users`.

2.6.4.5 Naslouchání na jiném portu

Tuto radu považujeme za *kontroverzní*. Jde o tzv. security by obscurity. Je třeba *zvážit výhody a nevýhody* pro konkrétní případ. Je-li server otevřen do Internetu, v logu je mnoho pokusů o hádání hesla. Za den to jsou tisíce pokusů, zvlášť pokud máte povoleno přihlašování pomocí hesla. Změnou čísla portu se počet pokusů výrazně sníží, obvykle na jednotky pokusů. Tím snížíte zátěž serverů i pravděpodobnost náhodného úspěchu útočníka.

Ale změna portu není účinná vůči útočníkovi, který se *zaměří* na Váš server. Jednoduchým skenem najde port, na kterém SSH daemon poslouchá. Změna portu je též *nepříjemná pro uživatele* – pokud mají přistupovat k většímu počtu serverů a na každém je jiné číslo SSH portu, tak budou zmateni a často nebudou schopni se přihlásit. V podstatě jim neúmyslně omezíte přístup ke službám.

Změna portu v `sshd_config` je jednoduchá:

```
Port 2244
```

2.6.4.6 Omezení počtu pokusů o přihlášení

Útočníci se pomocí skriptů neustále snaží o přihlášení na dostupné servery – pro různé účty a s různými hesly. I když je vypnuté přihlašování pomocí hesla, tak to zbytečně zatěžuje server. Používají se různé metody a jejich kombinace. Omezení počtu pokusů může vést k odmítnutí služby, kdy se *nepřihlásí ani legální* uživatel.

Pomocí parametru **MaxAuthTries** v `sshd_config` lze omezit počet pokusů o přihlášení v průběhu jednoho připojení. Výchozí hodnota je 6, takže uživatel může zkusit šest chybných hesel, než server ukončí spojení. Přestože najdete mnohá doporučení na snížení této hodnoty na 3 či méně, tak to příliš nedoporučuji, neboť to může zablokovat přihlášení pomocí klíčů. Když má uživatel v paměti načteny např. 4 klíče, tak program (`pageant` nebo `ssh-agent`) postupně pro přihlášení vyzkouší všechny. A pokud by bylo omezení na tři pokusy, tak se čtvrtý klíč nikdy nezkusí.

Další metody stručně:

- *omezení na firewallu*: lze omezit počet souběžných spojení z IP adresy i počet navázaných spojení z IP adresy za určitou dobu.
- *specializované aplikace* jako *fail2ban*: tyto aplikace průběžně čtou logy a pokud je z nějaké IP adresy více neúspěšných pokusů o přihlášení, tak tuto IP adresu na určitou dobu (např. 12 hodin) zablokují.
- PAM moduly: pro PAM existuje více modulů, které ovlivňují přihlašování. Lze omezit počet souběžných přihlášení od jednoho uživatele. Modul *Pam_Tally2* umožňuje blokovat na určitou dobu (např. 3 hodiny) IP adresy, ze kterých byl větší počet neúspěšných přihlášení (např. více než 5 za posledních 30 minut).

2.7 SSH otázky

1. Lze uložit soukromý klíč k SSH bez hesla? Pokud ano, popište smysluplné využití, které je lze považovat i za bezpečné?
2. Proč je přihlašování pomocí klíče bezpečnější než přihlašování pomocí hesla?
3. Lze se se stejným soukromým klíčem přihlásit na dva různé účty na stejném serveru? Pokud ano, jak musí vypadat konfigurace na serveru?
4. Lze vytvořit z počítače SSH spojení na server bez toho, že by se otevřelo terminálové spojení? Pokud ano, uveďte nějaký smysluplný či používaný příklad.
5. Lokální tunely na HTTP servery obvykle neposkytují správné výsledky. Proč? Vyzkoušejte tunel např. na http://bis.vse.cz či http://fis.vse.cz. Příčinu hledejte v přesměrování na HTTPS a v obsahu HTTP hlavičky Host.
6. Předchozí otázka je na HTTP servery v kombinaci s lokálními tunely. Proč lze tunelovat HTTP/HTTPS přes dynamické tunely?
7. Je možné vytvořit SSH tunel a do něho umístit další SSH spojení? Pokud ano, uveďte nějaký trochu smysluplný příklad.
8. Programy `pageant` či `ssh-agent` po zadání hesla načtou soukromé klíče do paměti. Následně se lze připojit na SSH servery bez zadání hesla ke klíči. Jaká jsou bezpečnostní rizika používání těchto programů?
9. Lze vytvořit SSH tunel, který by směřoval na konkrétní webovou stránku? Např. aby byla dostupná stránka http://osi.vse.cz/vpnka/ a nebyla dostupná stránka http://osi.vse.cz/eduroam/? Pokud ano, jak byste tento tunel nastavili?

10. Při vytvoření SSH spojení se kontroluje veřejný klíč serveru. Je to obrana proti útoku Man-in-the-middle. Popište stručně, jak by takový útok mohl probíhat.
11. Jaký je rozdíl mezi `scp` a `sftp`?
12. Lze tunelovat protokol FTP přes lokální tunel? Pokud ano, tak v jakém režimu přenosu dat (aktivní/pasivní)? Pokud ne, tak zdůvodněte.
13. Lze tunelovat protokol FTP přes dynamický tunel? Pokud ano, tak v jakém režimu přenosu dat (aktivní/pasivní)? Pokud ne, tak zdůvodněte.
14. SSH používá protokol TCP či UDP? Jaké je výchozí číslo portu SSH serveru při terminálovém přístupu? Jaké je výchozí číslo portu SSH serveru při kopírování souborů pomocí protokolu SCP?
15. Co je fingerprint SSH serveru? K čemu se používá?
16. Co obsahuje záznam SSHFP v doménovém prostoru DNS? K čemu se používá? Může se použít bez DNSSEC (bez podepisování DNS záznamů)? Umí ho využít program PuTTY?
17. Co je SSH Jump Host? Uveďte konkrétní příklad využití.
18. Co je SSH Agent Forwarding? Jaká má bezpečnostní rizika?

3 OpenPGP a program GnuPG

3.1 Úvod do OpenPGP

OpenPGP označuje *kryptografický protokol* v počátcích určený hlavně pro zabezpečení e-mailových zpráv.[138] Příjemce může ověřit, zda zpráva nebyla změněna během cesty a zda podpisujícím je konkrétní držitel soukromého klíče. Zprávy lze zašifrovat pro určeného příjemce (držitele konkrétního soukromého klíče).

OpenPGP podporuje též šifrování a podepisování datových souborů, tj. je možné šifrovat a ověřovat integritu jakýchkoliv dat. Příkladem může být *podepisování balíčků* v unixových distribucích či *podepisování zápisů* (commits) v gitu. Existují i různá pokročilá využití – např. transparentní šifrování konfiguračních souborů s hesly před uložením do veřejného úložiště (git-crypt: viz https://www.agwa.name/projects/git-crypt/).

OpenPGP používá **asymetrickou kryptografii** (kryptografie *s veřejným klíčem*) s dvojicemi klíčů – *veřejný klíč*, který může znát kdokoliv, a *soukromý klíč* (též privátní), který by měl znát pouze vlastník. Z veřejného klíče nelze reálně odvodit soukromý klíč.

3.1.1 Začátky PGP

První verzi šifrovacího programu **Pretty Good Privacy (PGP)** napsal *Phil Zimmermann* roku 1991 pro bezpečnou výměnu zpráv a souborů mezi protijadernými aktivisty přes veřejné sítě, v té době nejčastěji přes BBS (Bulletin board system).

O program, který byl pro nekomerční využití byl zdarma včetně zdrojových kódů, byl mezi uživateli zájem. Brzy se rozšířil i na Internet a stal se *de-facto standardem* v oblasti bezplatných programů pro šifrování a podepisování souborů a elektronické pošty.

V roce 1993 byl autor obžalován *za nepovolený vývoz zbraní* z USA, neboť v té době pro vývoz kryptografických systémů s klíčem delším než 40 bitů byla nutná licence na export vojenského materiálu. Program PGP používal od začátku klíče s minimální délkou 128 bitů. Autor v roce 1996 spor vyhrál – zdrojový kód nechal vydat jako knihu a poté se odkazoval na první dodatek ústavy USA (svoboda vyznání, tisku, projevu, shromažďování). Tento spor je součástí tzv. **první války o kryptografii**, která skončila v roce 2000 uvolněním pravidel pro vývoz kryptografického software z USA.

Po ukončení soudního sporu autor s dalšími založil firmu PGP Inc., která začala připravovat komerční verzi. Díky akvizicím a obchodním transakcím se komerční práva k programu a značce PGP přesouvala mezi různými firmami.

3.1.2 PGP, OpenPGP, GnuPG a GPG

V roce 1996 vyšlo RFC 1991, které popisuje formát zpráv a souborů pro PGP. V roce 1997 firma PGP Inc. předložila IETF návrh standardu na **protokol OpenPGP**, který v roce 1998 vyšel jako RFC 2440. Protokol prošel aktivním vývoje, v současné době je *platné RFC 4880* z roku 2007.

---

[138] V RFC 4880 je napsáno, že pojem OpenPGP označuje kryptografický software založený na PGP a na daném standardu. Dle převládajícího názoru pojem **OpenPGP** označuje *kryptografický standard* a *neměl* by se používat pro označení programů, viz např. https://help.ubuntu.com/community/GnuPrivacyGuardHowto.

Existují *dvě hlavní implementace* protokolu OpenPGP – první je komerční s názvem **PGP**. Druhá implementace s názvem **GNU Privacy Guard** (zkratka **GnuPG**, neoficiálně také **GPG**) je pod svobodnou licencí GNU a její vývoj zaštituje organizace GNU Project. Základní řádkový program z této implementace se jmenuje **gpg**.

GnuPG je součástí většiny unixových distribucí, existují *porty pro MacOS* (**GPGtools:** https://gpgtools.org/) či do *Windows* (Gpg4win: https://www.gpg4win.org/). Součástí **Gpg4win** je *též rozšíření* do poštovního klienta *Outlook* od firmy Microsoft. Pomocí rozšíření **Enigmail:** https://www.enigmail.net/index.php/en/, se doplní podpora OpenPGP do poštovního klienta *Thunderbird* (Mozilla Thunderbird), který je dostupný pro Windows, Linux, macOS i FreeBSD.

Dále pro *linuxové systémy* existují *grafická rozhraní* podle toho, jaké desktopové prostředí daná linuxová distribuce využívá. **KGpg** je grafické rozhraní pro KDE, kromě toho byla podpora integrována přímo do programu *Kontac*t resp. **KMail** (standardní poštovní program v tomto prostředí).

Obdobně pro desktopové prostředí GNOME existuje program **Seahorse**, který je přímo využit např. v programech **Evolution** (poštovní program a Personal Information Managaer) nebo ve správci souborů *Nautilus*.

Historicky vznikly tři hlavní vývojové větve GnuPG:

- Verze 1.4 (*Classic*) s monolitickou aplikací *gpg*.
- Verze 2.0 (*Stable*) byla první s oddělenou kryptografickou knihovnu, což umožnilo snazší používání OpenPGP z jiných aplikací. Součástí je textová a grafická aplikace pro dotazování na hesla (tj. lze snadněji používat v grafickém uživatelském prostředí). Další novinkou se stala podporu standardu S/MIME. Vývoj verze 2.0 již byl ukončen
- Verze 2.1/2.2 (*Modern*) navazuje na verzi 2.0 a přidává podporu kryptografie založené na *eliptických křivkách*.

Z hlediska ovládání jsou mezi verzemi minimální rozdíly. Verze 1.4 může být nainstalována společně s verzí 2.1/2.2. Pokud chceme explicitně volat určitou verzi (jsou-li instalovány obě), pak se pro starší verzi se použije příkaz `gpg1`, pro novější `gpg2`. Na kterou verzi odkazuje příkaz `gpg` lze zjistit pomocí přepínače `--version`:

```
gpg --version
gpg (GnuPG) 2.1.17
libgcrypt 1.7.5
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later
<https://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Home: /home/pavlicek/.gnupg
Supported algorithms:
Pubkey: RSA, ELG, DSA, ECDH, ECDSA, EDDSA
Cipher: IDEA, 3DES, CAST5, BLOWFISH, AES, AES192, AES256, TWOFISH,
        CAMELLIA128, CAMELLIA192, CAMELLIA256
        Hash: SHA1, RIPEMD160, SHA256, SHA384, SHA512, SHA224
        Compression: Uncompressed, ZIP, ZLIB, BZIP2
```

Ve skriptech bude primárně popsána verze 2.1/2.2. Se starší verzí se již příliš nesetkáte, ale „pro jistotu“ v příslušných kapitolách stručně zmíníme odchylky tam, kde jsou významné, např. při generování klíčů. Podrobné informace o programu *gpg* lze vyčíst z manuálových stránek `man gpg`. *Základní syntaxe* je následující:

```
gpg [přepínače] [parametry] jméno_souboru
```

Přepínače začínají dvěma pomlčkami, několik (nejčastější) jich má i krátkou jednopísmennou variantu s jednou pomlčkou (obvyklá syntax Linux příkazů). Sémanticky lze z přepínačů vyčlenit *příkazy*, které označují *základní akci*, která se má po spuštění gpg provést. Příkladem je `--help, --version, --sign, --encrypt, --verify` či `--edit-key`.

Pokud se nezadá žádný přepínač, ale *pouze jméno souboru*, tak se akce (kde je to dostatečně jednoznačné) odvodí od typu souboru. Pokud je soubor zašifrovaný, logicky ho asi chceme dešifrovat, zatímco pokud je soubor jen podepsaný, zkontroluje se jeho podpis.

Parametry dále *upřesňují* zadaný typ akce. Např. v příkladu níže se v první verzi příkazu zadá jako typ akce *podepisování*, v druhé verzi je přidán ještě upřesňující parameter `--armor` (podrobnější popis těchto příkazu včetně přesného významu parametru `--armor` je dále). Několik málo příkazů (zejména generování klíčů) nepotřebuje jméno souboru, většina ano a pokud ze vstupního souboru současně vzniká výstupní soubor, lze *volitelně* uvést ještě odlišné jméno *výstupního* souboru (`--output jméno_výst._souboru`).

```
# Podepisování souboru (podrobný výklad později)
gpg --sign soubor.txt

# Podepisování souboru s upřesňujícím parametrem --armor
gpg --sign --armor soubor2.txt
```

3.2 Vytvoření a úpravy klíčů, výměna veřejných klíčů

Asymetrická kryptografie s veřejnými klíči je základem používání OpenPGP. Před podepsáním dokumentu/dopisu si musíte vytvořit svoji dvojici soukromého a veřejného klíče. Před zašifrováním souboru pro někoho musíte získat nejdříve veřejný klíč příjemce. Obdobně před odesláním zašifrovaného dopisu pro Vás musí odesílatel získat Váš veřejný klíč.

3.2.1 Vygenerování soukromých a veřejných klíčů

Své klíče vygenerujete v terminálovém okně po zadání příkazu:

```
gpg --gen-key
```

Ve verzi 2.1/2.2 bylo generování klíčů *výrazně zjednodušeno*: standardně se zadávají jen dva nezbytné údaje: jméno a příjmení (Real name) + e-mailová adresa (Email address). Pro účely předmětu 4SA313 prosím zadejte e-mailovou adresu `username@bis.vse.cz`. Následně potvrdíte správnost údajů, poté se objeví dotaz na heslovou frázi (Passphrase). Po získání dostatečného množství entropie pro kryptografický generátor náhodných čísel se vygenerují klíče a současně se vytvoří *revokační certifikát* pro revokování klíče (odvolání jeho platnosti) v případě ztráty či zcizení soukromých klíčů. Dle verze se standardně

vygenerují RSA klíče 2048 nebo 3072 bitů (na *bis.vse.cz*). Postup, jak vyvolat generování klíčů se všemi možnými parametry (a tedy např. možnost změny délky klíče) je uveden dále.

```
# Zjednodušené generování klíčů v GnuPG verze 2.1x/2.2x
# Implicitně se generují klíče RSA 2048 nebo 3072 bitů (dle přesné verze)
gpg --gen-key
gpg (GnuPG) 2.1.17; Copyright (C) 2016 Free Software Foundation, Inc.

Note: Use "gpg --full-gen-key" for a full featured key generation dialog.

GnuPG needs to construct a user ID to identify your key.

Real name: Luboš Pavlíček
Email address: pavlicek@bis.vse.cz
You are using the 'utf-8' character set.
You selected this USER-ID: "Luboš Pavlíček <pavlicek@bis.vse.cz>"

Change (N)ame, (E)mail, or (O)kay/(Q)uit? O

… dotaz na heslovou frázi …

gpg: key 531F3C9784EDED03 marked as ultimately trusted
gpg: revocation certificate stored as '/home/pavlicek/.gnupg/openpgp-
revocs.d/5F6531D215510ED6C980D072D8B962F5C16C4D45.rev'
public and secret key created and signed.

pub   rsa2048 2016-12-03 [SC] [expires: 2018-12-03]
      A169316E65B5F8F393AC9886531F3C9784EDED03
uid                      Luboš Pavlíček <pavlicek@bis.vse.cz>
sub   rsa2048 2016-12-03 [E] [expires: 2018-12-03]
```

Vygenerovaný klíč identifikuje buď **otisk** (*fingerprint*) nebo *dlouhé* či *krátké* **keyid**. Pro výše vygenerovaný klíč mají následující hodnoty:

```
fingerprint (otisk): A169316E65B5F8F393AC9886531F3C9784EDED03
long keyid (dlouhé keyid): 531F3C9784EDED03
short keyid (krátké keyid): 84EDED03
```

Otisk se počítá z klíče obvykle jako SHA-1 haš o délce *160 bitů* (*40 hexadecimálních číslic*), *dlouhé keyid* je *posledních 16 hex znaků* z otisku, *krátké keyid* je posledních *8 hex znaků* z otisku. **Nedoporučuje** se používat *krátké keyid*, neboť existuje poměrně velká pravděpodobnost kolize již v případě několika tisíc uživatelů (viz narozeninový paradox). Verze 2.1 a 2.2 preferuje používání otisků.

Je vhodné nastavit vygenerovaný klíč jako *preferovaný* – v souboru `~/.bashrc` či v souboru `~/.profile` nastavíte proměnnou GPGKEY, tj. doplníte následující řádek s otiskem případně dlouhým keyid Vašeho klíče. Nastavení se uplatní při příštích přihlášení. Váš klíč můžete též doplnit do konfiguračního souboru `~/.gnupg/gpg.conf`.

```
export GPGKEY=A169316E65B5F8F393AC9886531F3C9784EDED03
```

V mnoha případech se lze na klíč odkazovat i pomocí e-mailové adresy. Vygenerovaný klíč expiruje za 2 roky, v průběhu doby uživatel může platnost prodloužit.

3.2.2 Pokročilejší generování klíče, generování ve starších verzích

Pokud byste ve verzi 2.1/2.2 chtěli klíče dle jiného algoritmu, chtěli delší klíč RSA či jinou dobu platnosti klíčů, tak pro generování použijte přepínač `--full-gen-key`. Naopak ve verzi 1.4 a 2.0 použijete výše uvedený příkaz `--gen-key`, který se v tomto případě vždy ptá na všechny parametry.

```
gpg --full-gen-key # verze 2.1/2.2
odpovídá přepínači --gen-key ve verzích 1.4 a 2.0 programu gpg
gpg --gen-key # verze 2.0 či 1.4
```

Nejdříve se vybírá *algoritmus generovaného klíče*. Obvykle se vybírá volba (1) RSA, při volbě (2) se algoritmy kombinují: Diffie-Hellman se použije pro výměnu klíčů. DSA pro podepisování, Elgamal pro šifrování. Ve třetí a čtvrté volbě se vygenerují *pouze* klíče pro podepisování. Klíče pro šifrování lze vygenerovat dodatečně.

Následně volíte délku klíče. U RSA je přípustná délka od 1024 do 4096, ale důrazně se nedoporučuje se používat klíče kratší než 2048 bitů (navíc v některých aplikacích jsou kratší klíče odmítány).

Následně můžete určit *dobu platnosti* vygenerovaných klíčů. Tuto dobu si můžete sami kdykoliv prodloužit, a proto se doporučuje nastavit platnost na jeden rok nebo nejvýše dva. Pokud byste klíče ztratili či přestali používat, tak uplynutím času se zneplatní.

Pozor na ***způsob zadávání délky***! Pokud zadáte jen číslo, zadáváte *dobu ve dnech*! Dříve (když ve starších verzí GPG byl tento dialog standardní), si řada studentů nastavila omylem platnost klíče jen na jeden nebo dva dny!

Další průběh generování klíče je stejný, jako *ve zjednodušené variantě*. Existuje ještě **expertní úroveň** generování klíčů. Ve verzi 2.1/2.2 lze poté zvolit i klíče založené na *eliptických křivkách*. Uživatele starších verzí s nimi ale nebudou schopni pracovat, takže zatím se příliš nedoporučuje používat je jako hlavní klíč.

```
gpg --full-gen-key --expert # verze 2.1/2.2 (umožňuje eliptické křivky)
      Please select what kind of key you want:
        (1) RSA and RSA (default)
        (2) DSA and Elgamal
        (3) DSA (sign only)
        (4) RSA (sign only)
        (7) DSA (set your own capabilities)
        (8) RSA (set your own capabilities)
        (9) ECC and ECC
        (10) ECC (sign only)
        (11) ECC (set your own capabilities)
      Your selection?
nebo
gpg --gen-key --expert # verze 2.0 či 1.4
```

3.2.3 Klíčenka – úložiště klíčů a užití klíčů

Při vygenerování klíčů se vytvoří v domovském adresáři podadresář `.gnupg`, a v něm *klíčenka* (anglicky *keyring:* česky tedy doslova „kroužek na klíče) – *soukromé úložiště* pro *vlastní* klíče i pro *veřejné klíče ostatních*. Pokud klíč není ve Vaší klíčence, tak ho nemůžete použít pro šifrování či ověřování podpisů. Ve verzi 2.1/2.2 je klíčenka v jednom souboru `pubring.kbx`,

ve starších verzích se klíče ukládají do dvou souborů – soukromé do `secring.gpg` a veřejné do souboru `pubring.gpg`.

V GnuPG se rozlišují **čtyři různá užití klíčů** (anglicky *capabilities*):

S (sign)	podepisování dopisů a souborů,
C (certify)	podepisování vlastních podklíčů a cizích veřejných klíčů,
E (encrypt)	šifrování dopisů/souborů,
A (authenticate)	autentizace uživatel, GnuPG může obsahovat klíče pro přihlašování pomocí *SSH* či *TLS*.

Hlavní klíč musí mít vždy užití C pro podepisování jiných klíčů. Pro ostatní užití mohou být vygenerovány *podklíče (subkey)*: samostatné dvojice soukromý a veřejný klíč. Toto rozdělení umožňuje nastavovat různou dobu platnosti jednotlivých podklíčů, různé algoritmy a délky klíčů. *Hlavní klíč* má obvykle nastaveno užití *Certify* a *Sign*. Pro *šifrování (Encryption)* se vygeneruje *podklíč*.[139]

Obrázek 3.1: Schéma využití hlavního klíče a podklíče

Zdroj: vlastní zpracování

Uživatel obvykle nemusí nad existencí podklíčů přemýšlet – program vybírá správné klíče dle požadované operace. Seznam známých klíčů (vygenerovaných nebo importovaných) lze vypsat následujícími příkazy:

```
gpg --list-keys # seznam veřejných klíčů
pub   rsa2048 2016-12-03 [SC] [expires: 2018-12-03]
      A169316E65B5F8F393AC9886531F3C9784EDED03
uid           [ultimate] Luboš Pavlíček <pavlicek@bis.vse.cz>
sub   rsa2048 2016-12-03 [E] [expires: 2018-12-03]

gpg --list-secret-keys # seznam soukromých klíčů
sec   rsa2048 2016-12-03 [SC] [expires: 2018-12-03]
      A169316E65B5F8F393AC9886531F3C9784EDED03
uid           [ultimate] Luboš Pavlíček <pavlicek@bis.vse.cz>
ssb   rsa2048 2016-12-03 [E] [expires: 2018-12-03]
```

[139] *Oddělení klíče pro šifrování* vychází též *z vlastností algoritmů* nad Diffie-Hellman, kde dvojici klíčů určenou pro digitální podepisování (DSA) *nelze* použít pro šifrování pomocí Elgamal. Stejné omezení je také u algoritmů nad eliptickými křivkami.

Ve výpisech jsou u klíčů následující údaje:

- *typ klíče* – používají se zkratky **pub** (veřejný klíč), **sec** (soukromý klíč), **sub** (veřejná část podklíče), **ssb** (soukromá část podklíče),
- *algoritmus a velikost klíče* – rsa2048 (někdy 2048R) označuje RSA klíče o délce 2048 bitů,
- *datum vytvoření klíče* a případně i datum expirace klíče,
- *užití klíče* (viz výklad výše),
- *datum expirace*, pokud je nastaveno.

Druhý řádek výpisu obsahuje *otisk klíče* (ve starších verzích GnuPG obvykle jen *keyid* hned vedle typu klíče). Řádek *uid* obsahuje identifikace vlastníka klíče (jméno, příjmení a e-mail). Před jménem je v hranatých závorkách platnost (validity). Pokud znáte soukromý klíč, tak platnost je vždy ultimate (nejvyšší). Podrobněji o platnosti klíčů viz kap. 3.4.

3.2.4 Editace klíčů

Řadu údajů u svých klíčů i cizích veřejných klíčů můžete měnit pomocí příkazu:

```
gpg --edit-key ID_klíče
```

Této příkaz uživatele přepne do zvláštního interaktivního režimu, ve kterém můžete pomocí povelů *modifikovat vybraný klíč*. V tomto režimu *nelze* zadávat příkazy popsané v jiných kapitolách a platí zde *speciální sada příkazů*. Tento režim poznáte podle *zvláštního promptu* na příkazové řádce (`gpg>`). Mezi základními povely zmiňme:

- *help* – vypsání přehledu dostupných povelů pro editaci klíčů (základní nápověda),
- *quit* – ukončení editace, zeptá se, zda uložit změny,
- *save* – uloží provedené úpravy a ukončí program,
- *adduid* – přidání další identifikace uživatele (uid), obvykle přidání další e-mailové adresy,
- *deluid* – mazání některé identifikace uživatele,
- *addkey* – přidání podklíče,
- *delkey* – mazání podklíče,
- *revkey* – zneplatnění (revokování) klíče,
- *passwd* – změna heslové fráze k soukromému klíči,
- *expire* – úprava doby vypršení platnosti klíče či podklíče,
- *trust* – nastavení důvěry pro cizí veřejný klíč (viz dále),
- *sign* – podepsání cizího veřejného klíče (viz dále),
- *revsig* – zneplatnění (revokování) podpisu cizího veřejného klíče,

Při editaci klíče se nejdříve zobrazí informace o klíči včetně neplatných podklíčů. Vedle *platnosti (validity)* se zobrazuje *důvěryhodnost (trust)*. Obojí podrobněji v kap. 3.4.

```
# Editace klíče - přidání druhé e-mailové adresy.
gpg --edit-key A169316E65B5F8F393AC9886531F3C9784EDED03
Secret key is available.

sec  rsa2048/531F3C9784EDED03
     created: 2016-12-03  expires: 2018-12-03  usage: SC
     trust: ultimate      validity: ultimate
ssb  rsa2048/EE058757E90BF9F5
     created: 2016-12-03  expires: 2018-12-03  usage: E
     [ultimate] (1). Luboš Pavlíček <pavlicek@bis.vse.cz>
```

```
gpg> adduid
Real name: Luboš Pavlíček
Email address: lubos.pavlicek@bis.vse.cz
Comment:
You are using the 'utf-8' character set.
You selected this USER-ID: "Luboš Pavlíček <lubos.pavlicek@bis.vse.cz>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? o

sec  rsa2048/531F3C9784EDED03
     created: 2016-12-03  expires: 2018-12-03  usage: SC
     trust: ultimate      validity: ultimate
ssb  rsa2048/EE058757E90BF9F5
     created: 2016-12-03  expires: 2018-12-03  usage: E
     [ultimate] (1)  Luboš Pavlíček <pavlicek@bis.vse.cz>
     [ unknown] (2). Luboš Pavlíček <lubos.pavlicek@bis.vse.cz>

gpg> save
```

U všech klíčů se *zobrazuje dlouhé keyid* (ve starších verzích se zobrazuje krátké keyid). *Podklíč* má vlastní *odlišné keyid*. Vždy zadávejte *identifikaci hlavního klíče* – výjimkou je prodlužování platnosti či revokování jednoho konkrétního podklíče.

3.2.5 Exportování a importování veřejných klíčů

Aby bylo možné někomu poslat zašifrovaný soubor nebo zprávu, je nutné získat *jeho veřejný klíč*. Existuje několik možností, jak si klíče vyměnit, každá má své výhody a nevýhody. Zde popíšeme exportování do souboru a importování ze souboru. Bezpečný přenos zajistíte sami. Exportování veřejných klíčů zajistí přepínač `--export`.

```
gpg --export [--output jméno_souboru] [--armor] [id_klíče]
```

Lze zadat jméno výstupního souboru, jinak na standardní výstupní zařízení. Běžně se vytváří binární výstup, přepínač `--armor` zajistí konverzi do ASCII formátu. Jako parametr zadejte identifikaci jednoho či více klíčů. Pokud klíče nezadáte, vyexportují se všechny klíče.

```
# Příklady různých variant exportu klíče/klíčů
gpg --export pavlicek@bis.vse.cz > pavlicek.gpg
gpg --export --output pavlicek.txt --armor 531F3C9784EDED03
gpg --export --output all_public_keys.gpg
```

Na prvním řádku se binárně vyexportuje veřejný klíč uživatele *pavlicek@bis.vse.cz* a pomocí přesměrování STDOUT se uloží do souboru. Na druhém řádku se vyexportuje klíč s keyid 531F3C9784EDED03 do textového souboru. Ve třetím příkladu se vyexportují všechny veřejné klíče.

Exportuje se vždy veřejná část hlavního klíče a veřejné části všech podklíčů. Součástí výstupu jsou i mnohé atributy spojené s klíčem – identifikace uživatele, platnost, podpisy klíčů či preferované kryptografické algoritmy uživatele.

Klíče uložené do souboru se do klíčenky nahrají pomocí přepínače `--import`.

```
gpg --import jméno_souboru
```

Import je inteligentní – pokud již klíč ve Vaší klíčence existuje, tak pouze zaktualizuje jeho atributy. Následující výpis veřejných klíčů pomocí `gpg --list-keys` zobrazuje mimo jiné importovaný klíč (druhý klíč od uživatele `robot@bis.vse.cz`).

```
gpg --list-keys
pub   rsa2048 2016-12-03 [SC] [expires: 2018-12-03]
      A169316E65B5F8F393AC9886531F3C9784EDED03
uid           [ultimate] Luboš Pavlíček <pavlicek@bis.vse.cz>
sub   rsa2048 2016-12-03 [E] [expires: 2018-12-03]

pub   rsa2048 2014-03-20 [SC] [expires: 2017-03-08]
      5A4CEE4D6FEABAD3916B73EC5082CC38651A108C
uid           [ unknown] Robot <robot@bis.vse.cz>
sub   rsa2048 2016-03-08 [E] [expires: 2017-03-08]
```

Přepínač `--export-secret-keys` exportuje *soukromé klíče*. To použijete v případě, že chcete přenášet své klíče na *další počítač*. Pro import soukromých klíčů se stejně jako pro veřejné klíče používá přepínač `--import`.

3.2.6 Výměna veřejných klíčů přes klíčový server (keyserver)

Exportovaný klíč si mohou účastníci vyměňovat jako přílohu e-mailu, přes přenosný USB disk apod. Pokud je větší počet účastníků (více než 10), tak je tento způsob výměny klíčů problematický – někdo pošle změnu klíče jen některým účastníkům, někdo si zapomene naimportovat některý klíč apod.

Řešením je použit **klíčový server (keyserver)** – když někdo změní nějaký klíč, tak změnu nahraje na klíčový server. Všichni si pravidelně aktualizují své klíčenky dle údajů na klíčovém serveru. Klíčové servery jsou *veřejné a soukromé*, mohou používat protokoly HKP (OpenPGP HTTP Keyserver Protocol) či LDAP. Z veřejných serverů jsou nejznámější `keys.gnupg.net` či `pgp.mit.edu`.

Na cvičeních Bezpečnost informačních systémů budeme používat *soukromý* klíčový server na `bis.vse.cz`, který je dostupný pouze ze školní sítě. Server má též jednoduché webové rozhraní na `https://bis.vse.cz/sks/`.

Údaje na klíčovém serveru jsou *dostupné komukoliv*, kdo se na server připojí. To *zjednodušuje výměnu* veřejných klíčů, ale současně *omezuje soukromí*, neboť kdokoliv se může dozvědět e-mailové adresy a případně i vazby mezi uživateli. Veřejné klíčové servery usnadňují MitM útoky, neboť kdokoliv může vytvořit klíč s cizím jménem a s cizí e-mailovou adresou a tento klíč publikovat.

Pro výměnu klíčů s klíčovým serverem se používají následující příkazy:

```
gpg --keyserver klíčový_server --send-keys ID_klíče […]
gpg --keyserver klíčový_server --search-keys parametr_hledání
gpg --keyserver klíčový_server --recv-keys ID_klíče […]
gpg --keyserver klíčový_server --refresh-keys
```

Přepínač `--search-keys` se používá pro vyhledání či aktualizaci klíče, stačí zadat část jména či část e-mailové adresy a vyhledají se všechny klíče, které zadanému řetězci přibližně odpovídají. Uživatel si poté vybere klíče, které si chce nahrát do své klíčenky. U přepínačů `--send-keys` a `--recv-keys` je potřeba zadat *identifikaci klíče*. Přepínač `--refresh-keys` zaktualizuje *všechny veřejné klíče*, které máte ve své klíčence.

```
# Vyhledání a stažení konkrétního klíče (robot@bis.vse.cz)
gpg --keyserver bis.vse.cz --search-keys robot@bis.vse.cz
gpg: data source: https://bis.vse.cz:11371
(1) Robot Primus (R.U.R.) <primus@bis.vse.cz>
    4096 bit RSA key 4CEB66E448C0187B, created: 2015-10-23, expires: 2018-04-01
(2) Robot <robot@bis.vse.cz>
    2048 bit RSA key 5082CC38651A108C, created: 2014-03-20, expires: 2017-03-08
Keys 1-2 of 2 for "robot@bis.vse.cz".  Enter number(s), N)ext, or Q)uit > 2
gpg: key 5082CC38651A108C: public key "Robot <robot@bis.vse.cz>" imported
```

```
# Stažení veřejného klíče, nahrání jiného veřejného klíče
gpg --keyserver bis.vse.cz --recv-keys 92001940314FDED5
gpg: key 92001940314FDED5: public key "Dr. Gall (R.U.R.)
<gall@bis.vse.cz>" imported
gpg --keyserver bis.vse.cz --send-keys 5082CC38651A108C
gpg: sending key 5082CC38651A108C to hkp://bis.vse.cz
```

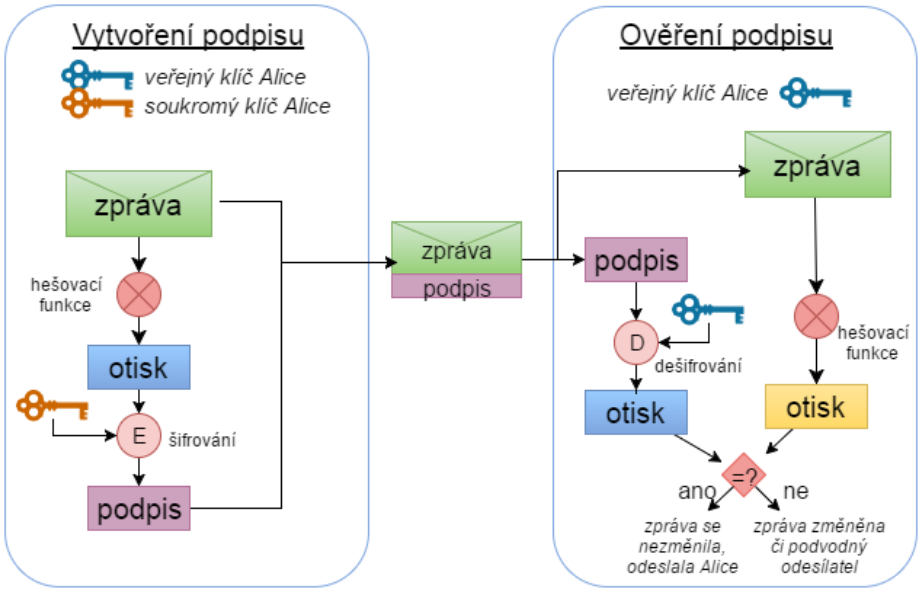
3.3 Šifrování a podepisování souborů

V této kapitole budeme probírat dva stěžejní způsoby využití kryptografie s veřejným klíčem: šifrování a podepisování souborů.

3.3.1 Podepisování souborů/zpráv

Digitální podpis zajišťuje *integritu* a *autenticitu* posílaných dat. Příjemce si může ověřit, zda se data cestou nezměnila a kdo data odeslal, tj. čí soukromý klíč byl použit pro podepsání (samotné podepisování *bez* zašifrování celé zprávy pochopitelně *nezajišťuje důvěrnost*). Před ověřením podpisu musí příjemce získat veřejný klíč podepisujícího a měl by ověřit jeho platnost (viz následující kapitola 3.4).

Obrázek 3.2: Schéma podepsání zprávy a ověření podpisu



Zdroj: vlastní zpracování

Podepsání souboru je jednoduché: zadáte přepínač `--sign` a jméno vstupního souboru:

```
gpg --sign [--armor] [--output výstupní_soubor] soubor_k_podpisu
```

Při podepisování standardně vznikne nový *binární* soubor, který obsahuje zkomprimovaná původní data a digitální podpis. Pokud nezadáte jméno výstupního souboru, použije se jméno původního souboru doplněné o příponu `.gpg`. Přepínač `--armor` zajistí výstupu *do ASCII souboru* ve formátu *Radix-64* (varianta *Base64*, běžně označováno též jako *ASCII armor*) a soubor bude mít koncovku `.asc`.

```
# První nově vzniklý soubor má příponu .gpg a je binární
gpg --sign soubor.txt
… dotaz na heslo k soukromému klíči …
# Druhý nově vzniklý souborů je ASCII (zajistí to přepínač -armor)
gpg --sign --armor soubor2.txt
ls -l
-rw-r--r-- 1 pavlicek pavlicek 145390 Dec  7 21:00 soubor.txt
-rw-r--r-- 1 pavlicek pavlicek  47009 Dec  7 21:01 soubor.txt.gpg
-rw-r--r-- 1 pavlicek pavlicek  29043 Dec  7 21:09 soubor2.txt
-rw-r--r-- 1 pavlicek pavlicek  14695 Dec  7 21:09 soubor2.txt.asc
```

Při podepisování se program gpg zeptá na heslo k soukromému klíči. Ve verzi 2.1/2.2 programu se automaticky na pozadí spustí **gpg-agent**, který si soukromý klíč pamatuje v paměti (obdoba programu `pageant` z PuTTY v případě SSH). Po dobu 10 minut nemusíte znovu zadávat heslo k soukromému klíči.

Podepsaný soubor se „*rozbalí*“ pomocí následující příkazu (obnoví se původní soubor a současně se zkontroluje podpis):

```
gpg podepsaný_soubor
```

Není tedy třeba zadávat žádný zvláštní přepínač. Při obnově původního souboru a současné kontrole podpisu mohou nastat následující situace:

- Máme platný veřejný klíč podepisujícího a podpis je v pořádku (*Good signature*).
- Podpis je v pořádku, máme veřejný klíč podepisujícího, ale nevíme, zda patří správné osobě (*There is no indication that the signature belongs to the owner*).
- Nemáme veřejný klíč podepisujícího (*No public key*).
- Máme veřejný klíč podepisujícího, ale podpis není platný (*BAD signature*).
- Chyba při přenosu, na kterou se přišlo díky kontrolním součtům (*CRC error*) či při dekompresi obsahu (*uncompressing failed*). Pouze v těchto případech se neobnoví původní soubor.

Následující výpis ilustruje dvě z výše uvedených situací. V prvním případě (`soubor3.txt.gpg`) nemáme veřejný klíč podepisujícího, ve druhém (`soubor4.txt.gpg`) veřejný klíč máme, ale (zatím) není platný.

```
ls -l
-rw-r--r-- 1 pavlicek pavlicek   9945 Dec  9 19:57 soubor3.txt.gpg
-rw-r--r-- 1 pavlicek pavlicek   1119 Dec  9 19:59 soubor4.txt.gpg
gpg soubor3.txt.gpg
gpg: Signature made Fri 09 Dec 2016 07:57:21 PM CET
gpg:                using RSA key 8F43F7DD03E72450
gpg: Can't check signature: No public key
```

```
gpg soubor4.txt.gpg
gpg: Signature made Fri 09 Dec 2016 07:58:56 PM CET
gpg:                using RSA key 92001940314FDED5
gpg: Good signature from "Dr. Gall (R.U.R.) <gall@bis.vse.cz>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:        There is no indication that the signature belongs to the
owner.
Primary key fingerprint: B697 1090 4644 0776 812A  804A 9200 1940 314F
DED5
ls -l
-rw-r--r-- 1 pavlicek pavlicek  25510 Dec  9 19:59 soubor3.txt
-rw-r--r-- 1 pavlicek pavlicek   9945 Dec  9 19:57 soubor3.txt.gpg
-rw-r--r-- 1 pavlicek pavlicek   1074 Dec  9 19:59 soubor4.txt
-rw-r--r-- 1 pavlicek pavlicek   1119 Dec  9 19:59 soubor4.txt.gpg
```

Při zadání přepínače `--verify` se *pouze* zkontroluje podpis bez obnovy („rozbalení“) původního souboru.

```
gpg --verify podepsaný_soubor
```

3.3.2 Oddělený podpis

Digitální podpis může být v *odděleném (samostatném)* souboru. Místo `--sign` zadejte přepínač `--detach-sign`. Další volitelné přepínače jsou stejné jako v předchozí podkapitole. Nezadáte-li výstupní soubor přepínačem `--output`, bude jméno stejné jako vstupní soubor doplněný v tomto případě o koncovku `.sig`. Přepínač `--armor` zajistí opět vytvoření ASCII souboru ve formátu *Radix-64* (varianta *Base64*), koncovka bude `.asc`.

```
gpg --detach-sign [--armor] [--output výstupní_soubor] soubor_k_podpisu
```

Oddělený podpis *trochu komplikuje* distribuci k příjemci, neboť zajištění přenosu dvou souborů je o trochu složitější než jednoho souboru. Při kontrole podpisu je potřeba *zadat oba soubory* – nejdříve soubor s podpisem a poté vlastní podepisovaný soubor.

```
 gpg -verify  soubor_s_podpisem  podepisovaný_soubor
```

Výhodou oddělených podpisů ale je možnost podepsání souboru *od více osob:* každý vytvoří svůj oddělený podpis a ty se poté spojí dohromady (standardním linuxovým příkazem `cat`). Nebo *podepisování ISO obrazů* či *balíčků* (jiných souborů, které nemají být modifikovány). Parametr `-u` specifikuje použitý soukromý klíč, pokud jich uživatel má více. Níže je příklad podepsání dvěma klíči, spojení podpisů a následná kontrola podpisů. Oba klíče jsou platné.

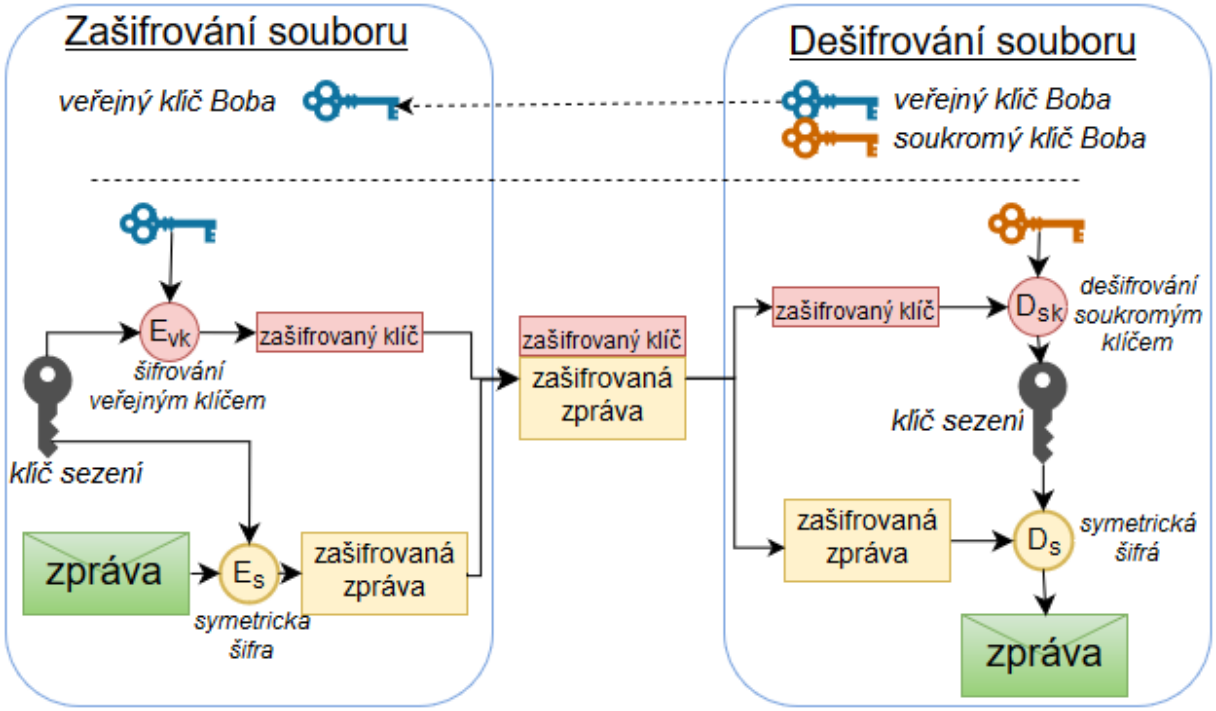
```
gpg --detach-sign --output soubor5.sig1 -u 92001940314FDED5 soubor5.txt
gpg --detach-sign --output soubor5.sig2 -u 4CEB66E448C0187B soubor5.txt
# Spojení dvou souborů s podpisem standardním příkazem cat
cat soubor5.sig1 soubor5.sig2 > soubor5.sig
# Kontrola podpisu
gpg --verify soubor5.sig soubor5.txt
gpg: Signature made Fri 09 Dec 2016 08:31:27 PM CET
gpg:                using RSA key 92001940314FDED5
gpg: Good signature from "Dr. Gall <gall@bis.vse.cz>" [full]
gpg: Signature made Fri 09 Dec 2016 08:31:55 PM CET
gpg:                using RSA key 4CEB66E448C0187B
gpg: Good signature from "Robot Primus <primus@bis.vse.cz>" [full]
```

3.3.3 Šifrování souborů

OpenPGP při šifrování **kombinuje** *symetrické a asymetrické* šifrovací algoritmy. Někdy se označuje jako *hybridní kryptografie*. Nejprve se vlastní *zpráva zašifruje symetrickou* šifrou (např. AES-128) jedinečným náhodným *klíčem sezení* (*session key*). Tento klíč sezení se poté zašifruje pomocí *veřejného klíče příjemce*, který musí být v klíčence. Velmi se doporučuje mezi příjemce vždy přidat též sám sebe (důvod viz dále). Obě části se spojí a odešlou.

Příjemce nejdříve dešifruje zašifrovaný klíč sezení pomocí svého *soukromého klíče*. A poté dešifruje vlastní zprávu algoritmem symetrického šifrování pomocí klíče sezení, který právě získal. Je-li příjemců více, každý pochopitelně má svůj soukromý klíč, ale samotnou zprávu poté všichni dešifrují stejným klíčem sezení.

Obrázek 3.3: Zašifrování a dešifrování souboru: kombinace symetrického šifrování a šifrování pomocí veřejného klíče



Zdroj: vlastní zpracování

Kombinování symetrické a asymetrické kryptografie má *dvě zásadní výhody*:

- Šifrování symetrickou šifrou je výrazně (mnohonásobně) rychlejší.
- Mnohem snadněji a efektivněji se šifruje stejná zpráva pro více příjemců.

Výhody kombinace symetrických a asymetrických algoritmů si ukážeme na ilustrativním příkladu. Předpokládejme soubor o velikosti 10 MB, který budeme posílat deseti příjemcům. Pokud bychom použili *pouze* asymetrickou kryptografii, museli bychom šifrovat stejných 10 MB dat celkem desetkrát (pokaždé jiným veřejným klíčem), tedy celkem 100 MB dat.

V OpenPGP budeme 10 MB šifrovat *pouze jednou* symetrickým algoritmem + skutečně *desetkrát* asymetrickým algoritmem šifrovat klíč sezení: ten má ale jen 128 nebo 256 bitů (tedy 16 nebo 32 bytů). Celkově tedy navíc šifrujeme jen přibližně 160 až 320 bytů, což vzhledem k 10 MB nebo dokonce 100 MB dat je zanedbatelné.

A protože symetrické algoritmy jsou samy o sobě mnohonásobně rychlejší a současně se šifruje jen zlomek objemu dat, celkový výsledný rozdíl v *rychlosti šifrování* snadno může

být *několik řádů*! Rozdíl v *objemu přenášených dat* je v našem ilustrativním případě „jen“ 1:10 (velikost vícenásobně zašifrovaného stejného klíče sezení můžeme zanedbat), ale obecně jak rozdíl v rychlosti šifrování, tak v objemu dat je tím větší, čím větší je šifrovaný soubor a čím více je příjemců daného souboru.

GnuPG zašifruje soubor po zadání přepínače `--encrypt`, volitelný přepínač `--armor` má stejný význam jako v příkladech výše. Je potřeba zadat aspoň jednoho příjemce, ten může být určen pomocí e-mailové adresy či pomocí *keyid*. Přepínač pro příjemce `--recipient` má i krátkou variantu `-r`. Lze zadat více příjemců, poté se (pouze) *klíč sezení* zašifruje vícekrát, pro každého příjemce samostatně (jak už bylo vysvětleno výše).

```
gpg --encrypt [--sign] [--armor] [--output výstupní_soubor]
   --recipient id_příjemce [--recipient id_přijemce2…] vstupní_soubor
```

Je dobrým zvykem *zašifrovaný* soubor *také podepsat*, tj. doplnit přepínač `--sign`. V tomto případě musíte zadat i heslo ke svému soukromému klíči (není-li klíč nahraný v `gpg-agent`). Toto pravidlo lze formulovat i „přísněji“: zatímco někdy má smysl soubor jen digitálně podepsat (viz předchozí podkapitola) a nechcete ho zašifrovat, jakmile použijete šifrování (přepínač `--encrypt`), vždy byste současně měli použít i přepínač `--sign`. Jen tak zajistíte *důvěrnost*, *integritu* a *autenticitu* přenášené zprávy (souboru) současně.

```
# Zašifrování souboru pro příjemce primus@bis.vse.cz a současně
# podepsání souboru soukromým klíče odesílatele
gpg --encrypt --sign --recipient primus@bis.vse.cz soubor2.txt
… dotaz na heslo k (mému) soukromému klíči …
```

Pokud *klíč příjemce není platný*, tak je třeba *potvrdit* zadání správného příjemce. Následující příklad to ilustruje: šifruji soubor pro uživatele robot@bis.vse.cz *s nepotvrzeným klíčem* a současně *též pro sebe* (pavlicek@bis.vse.cz).

To je další *doporučená praxe*: přidat mezi příjemce vždy *také odesílatele*. Pokud někdy v budoucnu z jakéhokoliv důvodu nebudu mít přístup k mému *původnímu nezašifrovanému* souboru, mohu odeslaný zašifrovány soubor také dešifrovat. A naopak, pokud nejsem mezi příjemci zprávy, nemohu ji pochopitelně dešifrovat, i když jsem ji vytvořil!

```
gpg --encrypt --sign -r robot@bis.vse.cz -r pavlicek@bis.vse.cz soubor3.txt
gpg: 53E9E0382F22902F: There is no assurance this key belongs to the
named user
sub  rsa2048/53E9E0382F22902F 2016-03-08 Robot <robot@bis.vse.cz>
 Primary key fingerprint: 5A4C EE4D 6FEA BAD3 916B  73EC 5082 CC38 651A
108C
      Subkey fingerprint: 5C78 FED7 FFAB 3ECE C4FA  1AE5 53E9 E038 2F22
902F

It is NOT certain that the key belongs to the person named
in the user ID.  If you *really* know what you are doing,
you may answer the next question with yes.

Use this key anyway? (y/N) y
```

Při zadání přepínače `--armor` se binární výstup opět převede do ASCII souboru ve formátu *Radix-64* (varianta *Base64*). Přepínač `--output` umožňuje zadat odlišné jméno výstupního souboru.

Základní příkaz *pro dešifrování* je stejný jako *pro rozbalení* (jen) podepsaného souboru. Pro dešifrování pochopitelně musíte ještě zadat heslo k soukromému klíči – pokud klíč již není načten v paměti programem `gpg-agent`.

```
gpg zašifrovaný_soubor
```

Výpis níže ukazuje dešifrování souboru `soubor3.txt.gpg`, který byl zašifrován výše. Všimněte si, že *keyid klíče* pro šifrování *neodpovídá* otisku klíče u podpisu – je to tím, že pro šifrování se používá *podklíč*, kdežto pro podepisování *hlavní klíč*. Existence dvou dvojic veřejného a soukromého klíče (hlavní a podklíč) a důvody proto již byly vysvětleny při generování klíčů (viz kap. 3.2.3) Obě id jsou vidět při editaci klíče (viz kap. 3.2.4).

```
# Dešifrování souboru, který byl zašifrován pro dva příjemce
 gpg soubor3.txt.gpg
… dotaz na heslo k soukromému klíči …
gpg: encrypted with 2048-bit RSA key, ID 53E9E0382F22902F, created 2016-03-08
     "Robot <robot@bis.vse.cz>"
gpg: encrypted with 2048-bit RSA key, ID EE058757E90BF9F5, created 2016-12-03
     "Luboš Pavlíček <lubos.pavlicek@bis.vse.cz>"
gpg: Signature made Sat 10 Dec 2016 06:23:49 PM CET
gpg:                using RSA key A169316E65B5F8F393AC9886531F3C9784EDED03
gpg:                 aka "Luboš Pavlíček <pavlicek@bis.vse.cz>" [ultimate]
```

Pokud nemáme v klíčence odpovídající soukromý klíč, pokus o dešifrování souboru pochopitelně selže, což ilustruje následující ukázka:

```
gpg soubor2.txt.gpg
gpg: encrypted with 2048-bit RSA key, ID F03A3750367220A6, created 2016-10-08
     "Robot Primus (R.U.R.) <primus@bis.vse.cz>"
gpg: decryption failed: No secret key
```

3.3.4 Šifrování pomocí sdíleného hesla

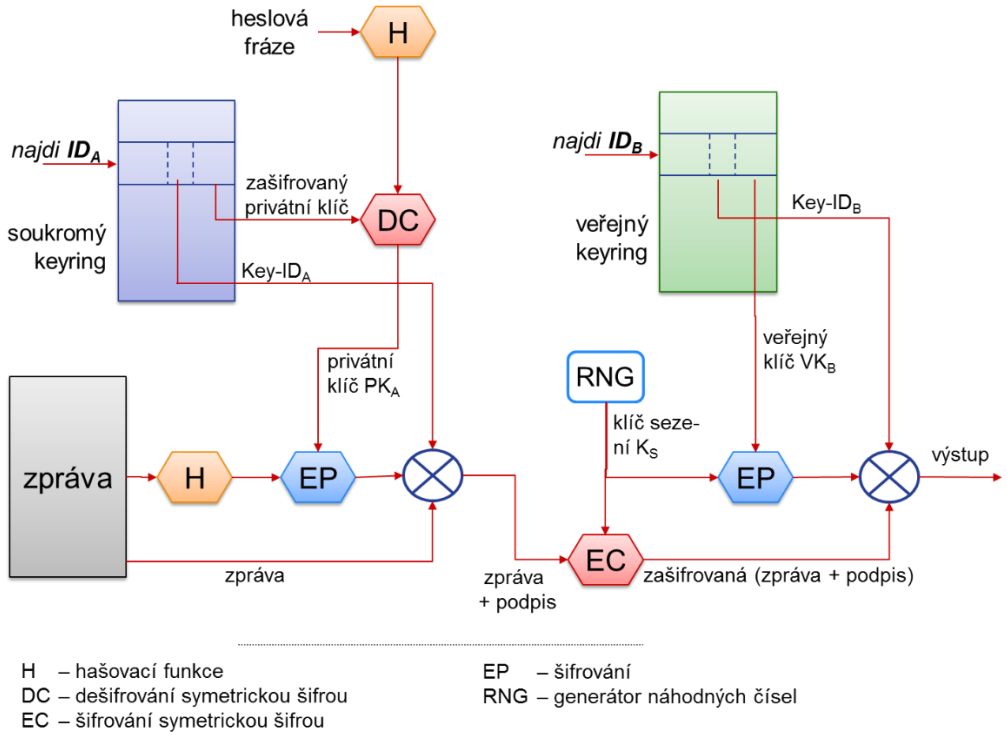
OpenPGP též podporuje šifrování pomocí *sdíleného hesla* (použije se *pouze* symetrický šifrovací algoritmus, nevyužívají se žádné klíče v klíčence). V tomto případě se použije přepínač `--symetric`. Nepovinné přepínače `--armor` a `--output` mají stejný význam jako v předchozích příkladech.

```
gpg --symmetric [--armor] [--output výstupní soubor] vstupní soubor
```

Program se *zeptá na heslo*, které se zadává interaktivně z klávesnice. I toto heslo se na *omezenou dobu* ukládá do paměti programu `gpg-agent`. Ze zadaného hesla GnuPG vygeneruje klíč pro symetrickou šifru. Při generování klíče používá funkci podobnou PBKDF2 s náhodnou solí a vysokým počtem iterací odvozeným z rychlosti procesoru.

Soubor mohou *dešifrovat všichni, kdo znají heslo*. Všem příjemcům ho tedy musíme sdělit nějakým „rozumně bezpečným způsobem“. Útočník se může snažit *odchytit heslo* při jeho výměně mezi účastníky. Heslo lze také hádat pomocí slovníků (*slovníkový útok*) i pomocí *hrubé síly*. Je tedy třeba pečlivě zvážit, kdy je vhodné přepínač `--symmetric` použít.

Obrázek 3.4: Schéma současného podepsání a zašifrování zprávy v OpenPGP



Zdroj: vlastní zpracování (obrázek se vztahuje ke kap. 3.3.3 výše)

3.4 Platnost klíče, důvěryhodnost

Zajištění *bezpečné distribuce klíčů* je jedna z největších *výzev* pro kryptology. Asymetrická kryptografie *část* problémů vyřešila, ale mnohé zůstaly. Ukážeme si to na poslání zašifrované zprávy.

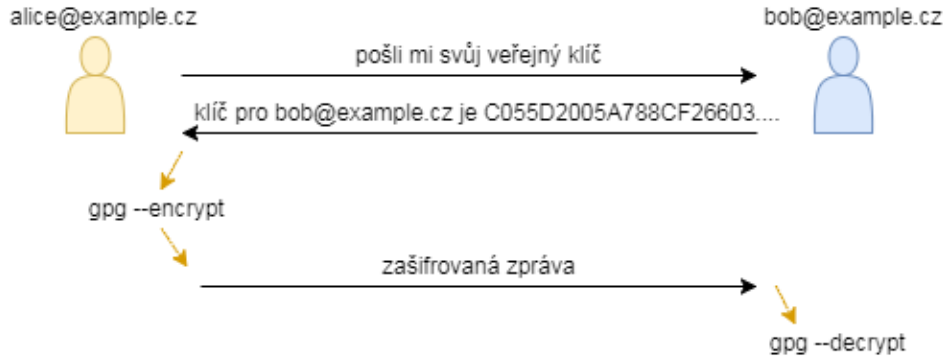
3.4.1 Útok MitM na distribuci veřejného klíče

Scénář odeslání zašifrované zprávy je jednoduchý. Alice chce poslat Bobovi zašifrovanou zprávu. Proto požádá Boba o poslání jeho veřejného klíče. Bob klíč pošle, Alice pomocí něho zašifruje zprávu a pošle ji Bobovi. Protože pouze Bob má soukromý klíč, tak pouze on může zprávu dešifrovat.

Pro útočníka je nejjednodušší zaútočit na začátku komunikace mezi Alicí a Bobem. Mallory odchytí požadavek Alice, kterým žádá Boba o zaslání jeho veřejného klíče. Poté Mallory vygeneruje novou dvojici klíčů pro adresu bob@example.cz. Falešný veřejný klíč pošle Alici, ta si myslí, že klíč patří Bobovi, a proto pomocí něho zašifruje zprávu. Mallory zašifrovanou zprávu odchytí, a protože má odpovídající soukromý klíč, tak může zprávu dešifrovat.

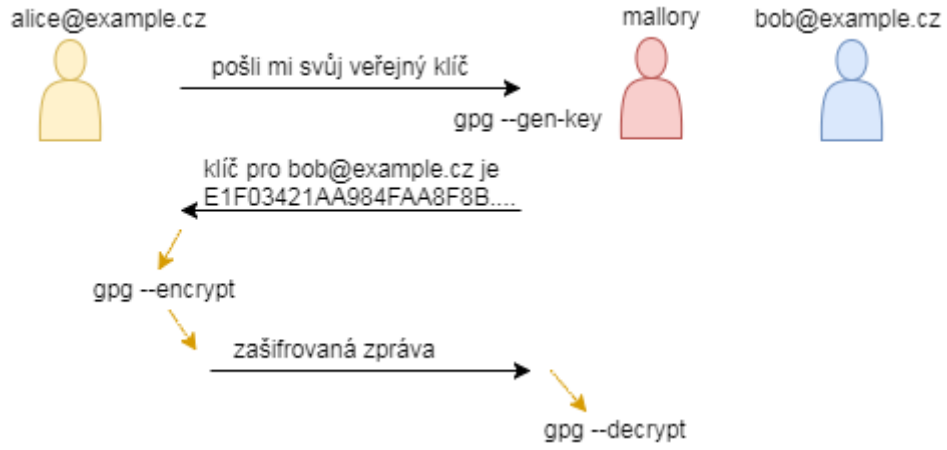
Jak Alice bezpečně ověří, že získaný veřejný klíč patří Bobovi, že ho nikdo nepodvrhl?

Obrázek 3.5: Alice požádá o klíč a poté s ním zašifruje zprávu



Zdroj: vlastní zpracování

Obrázek 3.6: Útočník může odchytit počáteční dotaz, vrátit podvržený klíč a následně přečíst zašifrovanou zprávu



Zdroj: vlastní zpracování

3.4.2 Platnost klíče (key validity)

Ověření identity majitele klíče je v případě OpenPGP *poměrně komplikovaná záležitost*. Nikdo není omezen, jaké reálné/smyšlené jméno či e-mailovou adresu do klíče zadá, takže kdokoliv se může vydávat třeba za prezidenta USA. Proto OpenPGP implicitně při použití cizího veřejného klíče upozorňuje *na nejistotu přiřazení klíče ke konkrétní identitě* a vyžaduje potvrzení akce od uživatele, viz následující výpis.

```
# Upozornění na použití veřejného klíče s NEZNÁMOU platností
# (zde při šifrování).

It is NOT certain that the key belongs to the person named
in the user ID.  If you *really* know what you are doing,
you may answer the next question with yes.

Use this key anyway? (y/N)
```

Veřejný klíč je **pro Vás *platný (valid)***, pokud **Vy *věříte***, že patří v něm *uvedené identitě* a současně pokud *není expirovaný* (neskončila mu platnost) či *není revokovaný* (odvolaný). Platnost *není univerzální* – pro někoho může být klíč platný a pro někoho jiného ne.

Expirace se ověří jednoduše – u klíče je datum od kdy platí a *může* být datum, do kdy platí.[140] Informace o odvolání klíče (revokace) je obvykle publikována na klíčovém serveru: musíte svoji klíčenku pravidelně aktualizovat vůči němu, abyste získali informaci o odvolání klíče. Jak revokovat svůj klíč je popsáno v kapitole 3.4.9.

Správnost identifikačních údajů (jméno a příjmení, přiřazené e-mailové adresy a případně vložené fotografie) a přiřazení klíče k této identitě můžete ověřit sami či to můžete delegovat na někoho jiného – *síť důvěry* si popíšeme v kapitole 3.4.8.

3.4.3 Osobní ověření identifikačních údajů klíče

Jednotlivé kroky kontroly si ukážeme na případu, kdy si Alice ověřuje veřejný klíč Boba:

1. Bob si vytiskne identifikační údaje a otisk klíče (fingerprint) *na štítek*.
2. Alice si domluví osobní schůzku s Bobem. Na ní převezme vytištěný štítek a nechá si od něho potvrdit, že to jsou jeho údaje.
3. Ještě na schůzce si Alice nechá předložit úřední doklad (např. občanský průkaz) a zkontroluje, zda Bob vzhledem odpovídá fotce v dokladu, ověří jméno a příjmení.
4. Alice dorazí domů, z klíčového serveru si stáhne klíč dle otisku uvedeného na štítku. Porovná údaje uvedené u klíče (jméno, e-mailová adresa, datum vytvoření) s údaji na štítku.
5. Pokud údaje odpovídají, tak svým klíčem podepíše e-mailovou adresu uvedenou u klíče. Podepsaný klíč vyexportuje, zašifruje klíčem Boba a pošle na uvedenou e-mailovou adresu. Pokud u klíče je více e-mailových adres, tak postup opakuje pro každou z nich.
6. Bob si vyzvedne zašifrovaný podepsaný klíč ze své poštovní schránky – tím se ověří, že je schopen číst dopisy pro uvedenou e-mailovou adresu. Poté dešifruje soubor s klíčem – tím se ověří, že k veřejnému klíči má odpovídající soukromý klíč. Nakonec podepsaný klíč nahraje na klíčový server.
7. Alice si stáhne z klíčového serveru aktualizovaný klíč Boba. Pokud je u klíče její podpis, tak se ověřili všechny údaje a klíč je pro Alici platný. Nyní ho může použít pro zašifrování zprávy, kterou chce poslat Bobovi.

Nemusí se používat *klíčový server*, Bob může klíč ukládat i na jiný sdílený prostor (např. LDAP server) či ho posílat elektronickou poštou. Uvedený postup se používá při ověřování *klíčů neznámých* či *málo známých osob* např. na tzv. **setkáních na podepisování klíčů** (***key signing party***). Další popis viz https://carouth.com/blog/2014/05/25/signing-pgp-keys/ nebo https://wiki.apache.org/apachecon/PgpKeySigning.

Pokud podepisující ověřil některé údaje *již v minulosti*, tak se průběh dost zjednoduší. Při podepisování klíče dlouholetého kolegy v kanceláři nemusíte kontrolovat úřední doklad nebo ověřovat jeho pracovní e-mailovou adresu přeposláním klíče do jeho poštovní schránky. Pomůže, pokud máte důvěryhodnou e-mailovou komunikaci, kterou dotyčný podepisoval svým klíčem.

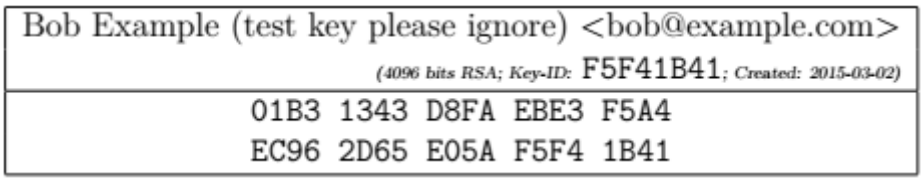
---

[140] Připomínám, že OpenPGP klíč se skládá obvykle ze *dvou dvojic klíčů* – jedna pro *podepisování* a druhá pro *šifrování*. Datum expirace může být uvedeno jen u veřejného klíče pro šifrování. Expirace i revokace se posuzuje u toho veřejného klíče, který chcete použít.

Někdy je osobní setkání *komplikované či nemožné*. Pokud ověřovanou osobu aspoň trochu znáte, tak můžete klíč ověřit s využitím *dalšího kanálu*. Bob nejdříve pošle podepsaný e-mail Alici, poté Alice např. zavolá Bobovi a nechá si od něho nadiktovat otisk jeho klíče[141] nebo požádá Boba o zaslání otisku jeho klíče pomocí SMS zprávy.

Ověření na dálku *je méně bezpečné* než osobní schůzka. Pokud vzdálenou osobu znáte málo či jste ji nikdy neviděli, tak je nejvhodnější *delegovat ověření* na někoho, komu důvěřujete, že kontrolu provede pečlivě.

Obrázek 3.7: Ukázka štítku s identifikačními údaji a otiskem klíče (fingerprint)



Zdroj: PDF soubor obsahující následující štítky byl vygenerován na serveru <https://openpgp.quelltextlich.at/slip.html>. Klíče hledá na veřejném klíčovém serveru.

3.4.4 Podepsání klíče

Ověření identifikačních údajů vyjádříte *digitálním podepsáním příslušného klíče* za použití přepínače `--sign-key`, popř. pomocí povelu `sign` při *editaci* podepisovaného klíče:

```
gpg --sign-key id_klíče
```

Po zadání přepínače `--sign-key` pro zvoleného uživatele se nejdříve z klíčenky zobrazí informace o podepisovaném klíči. Uživatel `gall@bis.vse.cz` má hlavní klíč pro podepisování zpráv certifikaci klíčů (`usage: SC`) a tři podklíče pro šifrování (`usage: E`). Z klíčů pro šifrování již dva expirovaly, třetí měl platnost do 1. dubna 2018 (výpis byl pořízen před tímto dnem). Hlavní klíč má neznámou platnost (`validity: unknown`). Také před e-mailovou adresou je uvedena neznámá (unknown) platnost této identity.

```
# Ukázka podepsání klíče uživatele gall@bis.vse.cz
gpg --sign-key gall@bis.vse.cz

pub rsa4096/92001940314FDED5
    created: 2015-10-23 expires: 2018-04-01 usage: SC
    trust: unknown validity: unknown
sub rsa2048/416B647304AFC009
    created: 2016-03-12 expired: 2016-05-11 usage: E
sub rsa2048/06D962E87CE05B18
    created: 2016-10-08 expires: 2018-04-01 usage: E
sub rsa4096/25AA068C768822F3
    created: 2015-10-23 expired: 2015-11-06 usage: E
[ unknown] (1). Dr. Gall (R.U.R.) <gall@bis.vse.cz>
```

[141] Při diktování a poslouchání posloupnosti čtyřiceti hexadecimálních číslic lze snadno *udělat chybu*. Již v roce 1995 proto vznikl pro PGP *slovník*, který převádí dvojici číslic na anglická slova, která se při vyslovování od sebe výrazněji liší – viz https://en.wikipedia.org/wiki/PGP_word_list.

```
This key is due to expire on 2018-04-01.
Are you sure that you want to sign this key with your
key "Luboš Pavlíček <lubos.pavlicek@bis.vse.cz>" (531F3C9784EDED03)
Really sign? (y/N) y
... dotaz na heslo k soukromému klíči ...
```

Podepisujete klíč pro podepisování klíčů (`usage: C`) a *identifikaci uživatele* (jméno, příjmení a e-mailovou adresu). Musíte zadat heslo pro svůj klíč, pokud aktuálně není v paměti programu `gpg-agent`. Podpis platí i pro podklíče, a to včetně všech budoucích. Vlastník může též prodloužit dobu platnosti klíče a Váš podpis bude stále platný. Pokud si ale vlastník klíče přidá další e-mailovou adresu, tak tato nová adresa nebude Vámi podepsaná.

Po podpisu byste měli podepsaný klíč exportovat a zašifrovat pro vlastníka klíče:

```
gpg --export --armor --output gall_podepsany_klic gall@bis.vse.cz
gpg --encrypt --recipient gall@bis.vse.cz --armor gall_podepsany_klic
```

Vznikne soubor `gall_podepsany_klic.asc` a ten pošlete na e-mailovou adresu vlastníka. Příjemce soubor dešifruje a nahraje do své klíčenky:

```
gpg gall_podepsany_klic.asc
gpg --import gall_podepsany_klic
```

Svůj klíč včetně informace o novém podpisu odešle na *klíčový server* (*musí* se uvést *keyid*, nestačí e-mailová adresa):

```
gpg --send-keys --keyserver bis.vse.cz 92001940314FDED5
```

Občas se podepisující a podepisovaný domluví, že *již podepisující* odešle podpis na klíčový server. Z něho si informaci o podpisu může stáhnout vlastník klíče i všichni ostatní.

Informace o podpisech klíčů lze vypsat pomocí přepínače `--list-sigs`:

```
gpg --list-sigs [id_klíče]
```

Následující ukázka vypíše informace o podpisech klíče `gall@bis.vse.cz` po jeho podepsání. Podepsaný klíč je pro mne již plně (**full**) platný – přesněji je platná e-mailová adresa `gall@bis.vse.cz` (řetězec „**uid**“ na začátku řádku) u klíče s otiskem B697109046440776812A804A92001940314FDED5 (řetězec „**pub**“ na začátku řádky).

Klíč podepsaly *již dvě osoby* (keyid DE6A7B75CFB69AAE a CA22AE326B9AD9A0), jejichž klíče se nenašli v klíčence. Můj podpis klíče je vidět na dalším řádku – podepsaní pomocí klíče s keyid 531F3C9784EDED03. Na dalších třech řádcích (viz „**sig 3**“ na začátku řádku) je vidět, že uživatel `gall@bis.vse.cz` podepsal tři své podklíče. Součástí výpisu je i informace o platných podklíčích (řetězec „**sub**“ na začátku řádky) – konkrétně je platný jeden podklíč pro šifrování, který podepsal vlastník hlavního klíče, tj. `gall@bis.vse.cz`.

```
# Výpis informací o podpisech klíče gall@bis.vse.cz
gpg --list-sigs gall@bis.vse.cz

pub   rsa4096 2015-10-23 [SC] [expires: 2018-04-01]
      B697109046440776812A804A92001940314FDED5
uid           [ full ] Dr. Gall (R.U.R.) <gall@bis.vse.cz>
sig           DE6A7B75CFB69AAE 2015-10-23 [User ID not found]
```

```
sig      CA22AE326B9AD9A0 2016-11-11 [User ID not found]
sig      531F3C9784EDED03 2017-01-14 Luboš Pavlíček
<lubos.pavlicek@bis.vse.cz>
sig 3    92001940314FDED5 2016-03-12 Dr. Gall (R.U.R.) <gall@bis.vse.cz>
sig 3    92001940314FDED5 2016-10-08 Dr. Gall (R.U.R.) <gall@bis.vse.cz>
sig 3    92001940314FDED5 2015-10-23 Dr. Gall (R.U.R.) <gall@bis.vse.cz>
sub rsa2048 2016-10-08 [E] [expires: 2018-04-01]
sig      92001940314FDED5 2016-10-08 Dr. Gall (R.U.R.) <gall@bis.vse.cz>
```

OpenPGP podporuje též **lokální podpis** při použití přepínače `--lsign-key` (popř. pomocí povelu `lsign` při *editaci* klíče). Lokální podpis *zůstává ve Vaší klíčence*, nikam se neexportuje a neposílá. Klíč bude *platný pouze pro Vás*. Používá se v případě jenom částečného ověření identity či v klíčenkách pomocných procesů.

```
gpg --lsign-key id_klíče
```

3.4.5 Útok na podpisy na veřejných keyserverech

V červnu 2019 neznámá osoba zaútočila na podpisy klíčů na veřejných keyserverech `keys.gnupg.net` – k některým zveřejněným klíčům doplnili statisíce podpisů. A s takto velkým počtem podpisů u jednoho klíče si aplikace nedokážou v rozumném čase poradit. Z keyserverů používajících software SKS nelze uvedené klíče a podpisy odstranit, neboť jejich návrh používá princip append-only databáze. Lze přidávat, nelze odebírat.

Přijatá řešení:

- v aplikaci `gpg` se změnil default pro stahování klíčů – nestahují se a neaktualizují podpisy,
- doporučuje se přejít na keyserver `keys.openpgp.org`, na kterém běží odlišný software s jinými principy.

3.4.6 Kvalita ověření identity

OpenPGP umožňuje při podepisování zadat i kvalitu ověření. Existují čtyři úrovně:

0 – nezadáno/nepovím, výchozí úroveň,
1 – nijak jsem nekontroloval,
2 – částečně jsem ověřil,
3 – pečlivě jsem ověřil.

Úroveň ověření větší než nula vypisuje např. přepínač `--list-sigs`. Výpis pro klíč gall@bis.vse.cz (kap. 3.4.4 výše) kvalitu ověření 3 u podpisu vlastních podklíčů (trojka vedle řetězce sig). V Unixu/Linuxu se program `gpg` obvykle na kvalitu ověření neptá a automaticky doplní nulu. Kvalita ověření se exportuje s klíčem, odesílá se na klíčový server. Při kvalitě ověření 1 (nijak jsem nekontroloval) se klíč ignoruje při výpočtu platnosti dle důvěry.

3.4.7 Důvěryhodnost (trust) osoby

Důvěra v klíč (*trust*) definuje, jakým způsobem věříte danému *člověku v ověřování* jiných lidí. Tedy jak věříte *jeho podpisům*. Existují možnosti:

- nevím, neznámá důvěra (*trust unknown*),
- nedůvěřuji (*never trust*),
- částečně důvěřuji (*marginal trust*),

- plně důvěřuji (*full trust*),
- nekonečně (*ultimate*) – používá se pro vlastní veřejný klíč, tj. když k veřejnému klíči máte i soukromý klíč.

Pokud důvěřujete *nekonečně cizímu* klíči, tak to znamená, že věříte všem podpisům klíčů dotyčného i v případě, že vlastní klíč dotyčného pro Vás není platný (např. expiroval). *Důrazně se nedoporučuje* nastavovat nekonečnou důvěru v cizí klíče.

Nastavená důvěra se *zobrazuje* např. *při podepisování klíče* – má důvěra (trust) v uživatele `gall@bis.vse.cz` při ověřování klíčů je neznámá (*unknown*), viz strana 137. Důvěru lze *nastavit* při editaci klíče (přepínač `--edit-key`) pomocí povelu *trust*.

```
# Nastavení důvěry pro uživatele gall@bis.vse.cz
gpg --edit-key gall@bis.vse.cz
... část výpisu vynechána ...
gpg> trust
pub  rsa4096/92001940314FDED5
     created: 2015-10-23  expires: 2018-04-01  usage: SC
     trust: unknown       validity: full
sub  rsa2048/06D962E87CE05B18
     created: 2016-10-08  expires: 2018-04-01  usage: E
 [  full  ] (1). Dr. Gall (R.U.R.) <gall@bis.vse.cz>

Please decide how far you trust this user to correctly verify other
users' keys
 (by looking at passports, checking fingerprints from different sources,
etc.)

  1 = I don't know or won't say
  2 = I do NOT trust
  3 = I trust marginally
  4 = I trust fully
  5 = I trust ultimately
  m = back to the main menu

Your decision? 4

pub  rsa4096/92001940314FDED5
     created: 2015-10-23  expires: 2018-04-01  usage: SC
     trust: full          validity: full
sub  rsa2048/06D962E87CE05B18
     created: 2016-10-08  expires: 2018-04-01  usage: E
 [  full  ] (1). Dr. Gall (R.U.R.) <gall@bis.vse.cz>

gpg> save
```

Důvěra je individuální – informace o důvěře se *neexportují* s klíčem, *neposílají* se na klíčový server. Informace o důvěře se ukládají do samostatného souboru `trustdb.gpg`, při přenášení klíčenky na jiný počítač musíte samostatně přenést i údaje o důvěře, které exportujete pomocí:

```
gpg2 --export-ownertrust > otrust.txt
```

A na druhém počítači importujete pomocí:

```
cd ~/.gnupg
rm trustdb.gpg
gpg2 --import-ownertrust < otrust.txt
```

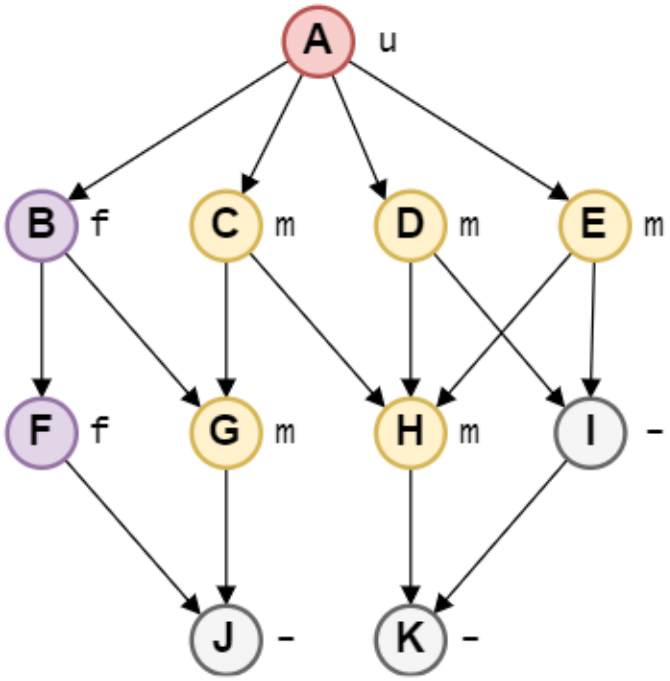
3.4.8 Vyhodnocení platnosti klíče, pavučina důvěry (Web of trust)

Při vyhodnocování platnosti klíčů pro Vás se postupuje následovně:

- Platné jsou *Vaše klíče*, tj. veřejné klíče, ke kterým máte soukromý klíč.
- Platné jsou klíče, které jste sami *podepsali*.
- Platné jsou klíče, které podepsal někdo, komu plně (*full*) důvěřujete a jehož klíč je pro Vás platný.
- Platné jsou klíče, které podepsaly aspoň *tři osoby*, kterým částečně (*marginal*) důvěřujete a jejichž klíče jsou pro Vás platné.

Důvěra se nastavuje klíčům, které jsou pro Vás platné – u neplatných klíčů nemá význam. Následující obrázek ukazuje podpisy klíčů a vztahy důvěry mezi jedenácti uživateli – používá se **pojem síť důvěry (Web of Trust)**. Které klíče jsou platné pro uživatele A?

Obrázek 3.8: Síť důvěry – podpisy klíčů a vztahy důvěry mezi 11 uživateli



Zdroj: vlastní zpracování

Řešení:

Platný je vlastní klíč A.
Platné jsou klíče B, C, D a E, neboť je uživatel A sám podepsal.
Platné jsou klíče F a G, neboť jsou podepsány uživatelem B, kterému A plně důvěřuje.
Platný je klíč H, neboť je podepsán třemi uživateli C, D a E, kterým A částečně důvěřuje.
Klíč I není platný, neboť má pouze dva podpisy od osob, kterým A důvěřuje jen částečně.
Klíč J je platný, neboť je podepsán uživatelem F, kterému plně důvěřuje uživatel A.
Klíč K je neplatný, neboť uživatel H má pouze částečnou důvěru a uživatel I není platný.
Souhrnná informace o síti důvěry se vypíše přepínačem `--check-trustdb`. Následující výpis ukazuje platnost klíčů na jednotlivých úrovních pro uživatele s minimálně 44 klíči v klíčence.

- Na úrovni 0 má platný jeden klíč, kterému důvěřuje na nekonečně (ultimate). Je to vlastní klíč uživatele. Uživatel podepsal 5 dalších klíčů.

- Na úrovní 1 má uživatel platných dalších 5 klíčů (ty které podepsal), čtyřem důvěřuje částečně (marginal) a jednomu plně (full). Pomocí těchto 5 klíčů bylo podepsáno 38 jiných klíčů.
- Na úrovni 2 je platných dalších 37 klíčů (z 38 podepsaných), u všech je důvěryhodnost neznámá.

```
# Souhrnné info o síti důvěry po zadání přepínače --check-trustdb
gpg --check-trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:   1  signed:   5  trust:  0-, 0q, 0n, 0m, 0f, 1u
gpg: depth: 1  valid:   5  signed:  38  trust:  0-, 0q, 0n, 4m, 1f, 0u
gpg: depth: 2  valid:  37  signed:   0  trust: 37-, 0q, 0n, 0m, 0f, 0u
```

3.4.9 Zneplatnění (revokování) klíče, revokování podpisu

Při *kompromitaci vlastního soukromého klíče* je důležité nejen si vytvořit nový, ale dát i ostatním vědět, že Váš předchozí klíč již není platný a důvěryhodný. Klíč revokujete při editaci klíče (přepínač `--edit-key`) pomocí povelu *revkey*, klíč je následně považován za neplatný. Veřejný klíč je nadále distribuován s příznakem zneplatnění. Revokovat může vlastník soukromého klíče.

Revokovat lze i pomocí tzv. *revokačního certifikátu* – Vašim soukromým klíčem podepsaný speciální soubor, který se obvykle vytváří při generování klíče (kap 3.2.1) či pomocí přepínače `--gen-revoke`. Certifikát použijete k zneplatnění klíče v situaci, kdy *ztratíte* svůj soukromý klíč.

```
# Příklad revokování testovacího klíče
# včetně odeslání informace na klíčový server.

gpg --edit-key C055D2005A788CF26603E1F03421AA984FAA8F8B
Secret key is available.

sec  ed25519/3421AA984FAA8F8B
     created: 2016-12-10  expires: never       usage: SC
     trust: ultimate      validity: ultimate
ssb  cv25519/CC6E03AF7C66F34B
     created: 2016-12-10  expires: never       usage: E
[ultimate] (1). test ecc <testecc@bis.vse.cz>

gpg> revkey
Do you really want to revoke the entire key? (y/N) y
Please select the reason for the revocation:
  0 = No reason specified
  1 = Key has been compromised
  2 = Key is superseded
  3 = Key is no longer used
  Q = Cancel
Your decision? 3
Enter an optional description; end it with an empty line:
> test ukončen
>
Reason for revocation: Key is no longer used
test ukončen
Is this okay? (y/N) y

The following key was revoked on 2017-01-15 by ? key 3421AA984FAA8F8B
test ecc <testecc@bis.vse.cz>
```

```
sec  ed25519/3421AA984FAA8F8B
     created: 2016-12-10  revoked: 2017-01-15  usage: SC
     trust: ultimate      validity: revoked
The following key was revoked on 2017-01-15 by ? key 3421AA984FAA8F8B
test ecc <testecc@bis.vse.cz>
ssb  cv25519/CC6E03AF7C66F34B
     created: 2016-12-10  revoked: 2017-01-15  usage: E
[ revoked] (1). test ecc <testecc@bis.vse.cz>
gpg> save
gpg --send-keys --keyserver bis.vse.cz 3421AA984FAA8F8B
```

Uživatel může *revokovat i svůj podpis pod cizím* klíčem. Při editaci podepsaného klíče zadáte povel *revsig*. Nezapomeňte poté klíč s revokovaným podpisem odeslat na keyserver, aby se tato informace rozšířila.

3.5 Šifrování a podepisování pošty

E-mail komunikace při využití standardních poštovních protokolů *nedává* příliš mnoho *bezpečnostních záruk*: není zajištěno, že zpráva dorazí, že ji nikdo jiný nebude číst, že ji nikdo nebude měnit, a dokonce ani nezaručuje *identitu odesílající osoby* (mj. lze poměrně snadno zfalšovat tzv. hlavičky poštovní zprávy). Důvod je vcelku prostý a obdobný jako v případě *telnet* versus *SSH* (první verze SSH až 1995). E-mail patří mezi nejstarší služby Internetu (začátek 70. let) a zpočátku se bezpečnost neřešila, důležitá byla funkčnost. Dokonce nebyly ani vhodné nástroje a algoritmy: první články o asymetrické kryptografii byly publikovány v druhé polovině 70. let a reálné využití nastalo ještě o dost později.

Protokol OpenPGP může bezpečnost výrazně zvýšit: šifrování zajistí, že si zprávu přečte jen daná osoba (vlastník příslušného soukromého klíče). Díky podepisování příjemce pozná, zda zprávu někdo modifikoval a může do určité míry ověřit identitu odesílajícího.

Při podepisování a šifrování e-mailových zpráv *nutně nepotřebujete* podporu *přímo v poštovním klientovi*. Konec konců zprávu můžete napsat do souboru, soubor podepsat a zašifrovat v programu GnuPG (podrobně probráno v předchozích kapitolách), a poté ho poslat jako běžnou *přílohu*.

Ale zabudování podpory *protokolu OpenPGP* do *poštovních klientů* uživatelům šifrování a podepisování výrazně *zjednodušuje*.[142] V Unixu/Linuxu je obecně podpora větší, nicméně i v Microsoft Outlook lze zprávy opatřit OpenPGP šifrováním či podpisy díky již dříve zmíněnému programu `Gpg4win`. Ve cvičeních použijeme linuxového poštovního klienta „`mutt`“, který GnuPG podporuje, a navíc dobře funguje v terminálu.

[142] Zásadní otázkou je, zda je *zjednodušení dostatečné*. Většina uživatelů poštovní zprávy nepodepisuje a nešifruje. *Bruce Schneier*, známý odborník na kryptografii (mimo jiné autor šifrovacích algoritmů *Blowfish* a *Twofish*, napsal také několik knih o aplikované kryptografii) na svém blogu zveřejnil krátkou úvahu (odvolávající se též na článek dalšího autora) o preferenci šifrovaných služeb pro posílání zpráv jako je *Signal* či *WhatsApp*. V těchto službách je kryptografie pro uživatele většinou *transparentní*, při ověřování identity se spoléhají na mobilní telefonní čísla. Úvaha vyvolala rozsáhlé komentáře čtenářů blogu o rozdílech mezi posíláním krátkých zpráv (instant messaging) a e-mailů, a také diskusi o výhodách a nevýhodách PGP a příčinách malého rozšíření podepsaných a šifrovaných e-mailů.
SCHNEIER, Bruce: Giving Up on PGP. In: *Schneier on Security* [online]. 2016-12-16 [cit. 2020-10-29]. Dostupné z: <https://www.schneier.com/blog/archives/2016/12/giving_up_on_pg.html>

3.5.1 Internetový standard MIME

MIME (*Multipurpose Internet Mail Extension*) je internetový standard, který *rozšiřuje* možnosti přenášených zpráv v e-mailu o podporu textu psaného v jiných *znakových sadách*, podporu *různých příloh* a *vícedílné zprávy*. Současní klienti tento standard využívají pro přenos zpráv zašifrovaných a podepsaných pomocí protokolu OpenPGP.

```
# Ukázka dopisu zašifrovaného pomocí OpenPGP.
# Některé části jsou vynechány.

Return-Path: <pavlicek@bis vse.cz>
Delivered-To: root@bis.vse.cz
Received: by bis.vse.cz (Postfix, from userid 1000)
        id 9ED406073B; Sun, 15 Jan 2017 16:23:21 +0100 (CET)
Date: Sun, 15 Jan 2017 16:23:21 +0100
From: Lubos Pavlicek <pavlicek@bis.vse.cz>
To: root <root@bis.vse.cz>
Subject: Test - zasifrovany dopis
Message-ID: <20170115152321.ffe4xma7dpxclglj@bis.vse.cz>
MIME-Version: 1.0
Content-Type: multipart/encrypted; protocol="application/pgp-encrypted";
        boundary="c7se3r5l2izv3kct"

--c7se3r5l2izv3kct
Content-Type: application/pgp-encrypted
Content-Disposition: attachment; filename="msg.asc"

-----BEGIN PGP MESSAGE-----

hF4DzG4Dr3xm80sSAQdAVmqmsC12P+PI0sIYFefqLlRcfbYaPTHBRi2grREgVhgw
D6fRjrsbQuzgi8MQuPoo8vRPvXXl5dPw5WWFfwCsLYj72pPoo3qIvh0iHlg03g65
    ... část vynechána ...
kMXP1iQ8JkpZn5//YN4fGbNSCPIG/3DokXKR0YEHq0oL31+aGe2052zsDkhcCPYJ
L3f4Lcb/qIxZYj36z+py90TkDkAL5cdDg70TmocLY2uJ3c7TL9Rr0NzDxoLO5ZRA
sctKEz+T9cVeSR9ZmeJ544ef6dZOyDjz5hZYQ+LM
=+D/K
-----END PGP MESSAGE-----

--c7se3r5l2izv3kct--
```

Zašifrovaný dopis má *content-type multipart/encrypted* s *protokolem application/pgp-encrypted*. Část typu „application/octet=stream“ obsahuje zašifrovaný vlastní text převedený do *armor formátu*, který jste již dobře poznali v předchozích podkapitolách. Z ukázky je vidět, že se *nešifruje hlavička* – kdokoliv na cestě si může přečíst *metadata* jako je čas odeslání, e-mailové adresa odesílatele a příjemce či předmět zprávy (Subject).

Pokud je dopis *pouze podepsaný*, tak má dopis *dvě části* – v první je nezašifrovaný vlastní text dopisu, ve druhé je připojen digitální podpis – opět ve formátu armor.

```
# Ukázka dopisu podepsaného pomocí OpenPGP.
# Některé hlavičky jsou vynechány.

Return-Path: <pavlicek@bis.vse.cz>
Delivered-To: root@bis.vse.cz
Received: by bis.vse.cz (Postfix, from userid 1000)
        id 77F4B6073B; Sun, 15 Jan 2017 16:14:14 +0100 (CET)
Date: Sun, 15 Jan 2017 16:14:14 +0100
From: Lubos Pavlicek <pavlicek@bis.vse.cz>
To: root <root@bis.vse.cz>
Subject: Test - podepsany dopis
Message-ID: <20170115151414.7iagxrkriq2wc3aa@bis145.vse.cz>
MIME-Version: 1.0
Content-Type: multipart/signed; micalg=pgp-sha512;
        protocol="application/pgp-signature"; boundary="jeplxw2qtrrni4ce"

--jeplxw2qtrrni4ce
Content-Type: text/plain; charset=utf-8
Content-Transfer-Encoding: quoted-printable

Uk=C3=A1zka podepsan=C3=A9ho dopisu.

--jeplxw2qtrrni4ce
Content-Type: application/pgp-signature; name="signature.asc"

-----BEGIN PGP SIGNATURE-----

iQEzBAABCgAdFiEEoWkxbmW1+POTrJiGUx88l4Tt7QMFAlh7kcMACgkQUx88l4Tt
7QMR1gf+K267CE2kTL4f8gYF8gsc/t2ejq7qY+FkGeS0PaXK+pJ0c9TFNZ14PZhJ
8scc1F4IJmeEyfk/peaZO/xzMg5V5ehEPdUjbvuLozBgMIuK8YPfBhAKCYb2Tvgk
709XQ9tpKE6Ew9YXo9y6/oZaRqBM8+k/76/wXoaa0YqS7mIFSo5O76jQA9OtJSOG
beLoebG5VnALbc5AxRytftmueCj9WgeivYdyY02JuZh+r1reLX0SzhGPQPYYLy4q
voTFPAONB8n955PVEQ8/xoZ6ytrKtPvz/aT2vQw4G+zUAhgwm24Eulul0uXbb1zp
1QZD+WTktWrhYnn3AVVkxIvT5v4yEg==
=hXC9
-----END PGP SIGNATURE-----

--jeplxw2qtrrni4ce--
```

3.5.2 OpenPGP a poštovní klient Mutt

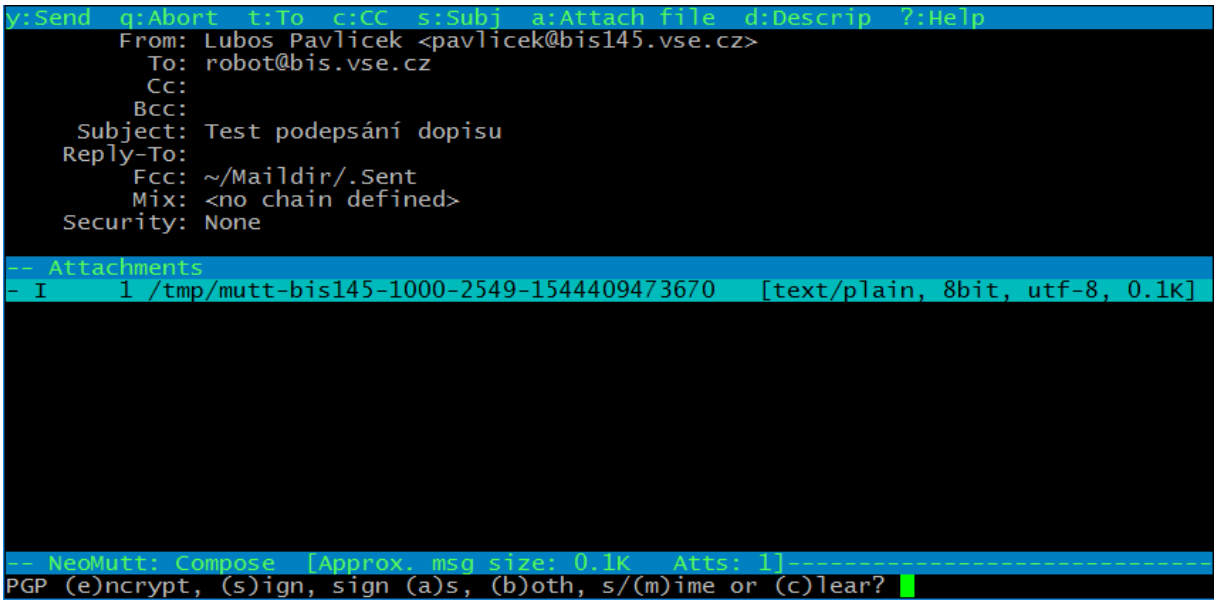
Poštovní klient Mutt je terminálová aplikace (aplikace pracující v textovém režimu). Základní ovládání klienta probíhá pomocí klávesnice – stisknutí konkrétního písmene vyvolá příslušnou akci. Nápovědu nejčastějších operací v daném kontextu najdete na prvním řádku okna (viz obrázek níže).

Po spuštění programu uživatel vidí přijaté zprávy (*Inbox*). Pomocí *šipek* se lze pohybovat mezi zprávami, pomocí klávesy *Enter* se zobrazí zpráva. U přijatých zpráv klient *automaticky* kontroluje podpisy, zašifrované zprávy dešifruje. Zobrazení přijaté zprávy se ukončí pomocí klávesy „i“.

Vytváření nové zprávy začíná povelem „**m**“ (*message*). Nejdříve je třeba na dolní řádce vyplní adresu příjemce (*To:*) a předmět (*Subject:*). Poté se zobrazí okno editoru a můžete psát vlastní text zprávy.

Po ukončení editoru se uživatel dostane do okna nové zprávy, viz následující obrázek. Po stisknutí klávesy „**p**“ můžete zvolit šifrování či podepsání zprávy. Na dolním řádku se zobrazí nabídka – zašifrování (*encrypt*), podepsání (*sign*), podepsat jako (*sign as*), obojí (šifrovat i podepsat, *both*) či pro zašifrování/podepsání použít standard S/MIME (OpenPGP podporuje od verze 2.0). Zprávu odešlete po zadání písmene „y“ (= *Send*).

Obrázek 3.9: Klient mutt: nastavení šifrování/podepisování před odesláním dopisu



Zdroj: vlastní zpracování

Podrobnou dokumentaci programu *Mutt* najdete na <https://www.mutt.org/doc/manual/>. Wiki stránky projektu pak na <https://gitlab.com/muttmua/mutt/-/wikis/home>. V rámci této Wiki najdete mimo jiné často kladené otázky (FAQ) a „Příručku pro začátečníky“ (Newbie guide: <https://gitlab.com/muttmua/mutt/-/wikis/MuttGuide>).

3.6 Struktura vytvářených zpráv

Z popisu struktury vytvářených souborů např. zjistíte, zda součástí podepsaného souboru je i veřejný klíč podepisujícího. V druhé části bude poté popsán ASCII formát, který vzniká po přepínači `--armor` či který se používá v elektronické poště.

3.6.1 Struktura zpráv (souborů)

Ukážeme si strukturu tří souborů – soubor s odděleným odpisem, zašifrovaný soubor a podepsaný zašifrovaný soubor. Další typy jako soubor zašifrovaný symetrickou šifrou, exportovaný veřejný klíč, podepsaný klíč či exportovaný soukromý klíč chráněný heslem jsou podrobně popsány ve standardu RFC 4880 <https://tools.ietf.org/html/rfc4880> – *OpenPGP Message Format*.

Výstupní soubory jsou buď *binární*, či se binární výstup převádí do „*ASCII“ formátu* pomocí přepínače `--armor`. ASCII formát bude popsán v následující kapitole. Výstupní soubory se skládají z *balíčků (packets)*. Balíčky se mohou skládat za sebe či různě do sebe vnořovat. Strukturu souboru si můžete vypsat pomocí přepínače `--list-packets`.

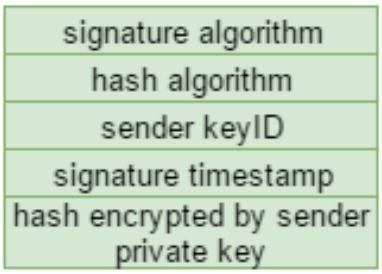
```
gpg --list-packets jméno_souboru
```

Začneme popisem **odděleného podpisu** (viz Obrázek 3.10), který obsahuje následující položky:

- Identifikaci algoritmu digitálního podpisu – např. RSA.
- Identifikaci použité hašovací funkce – např. SHA512.
- Dlouhé *keyid* klíče použitého při šifrování haše.
- Čas vytvoření podpisu, tento čas může být snadno zfalšován.
- První dva byty haše – pro kontrolu správnosti dešifrování haše.
- Haš ze souboru zašifrovaný pomocí soukromého klíče podepisujícího, tj. digitální podpis v nejužším významu. V případě RSA je to výsledek operace m^d **mod** n, kde ***m*** je vypočtený haš, ***d*** je soukromý exponent (klíč) a ***n*** je modul pro daný klíčový pár délky 2048 bitů a více.

Každý podpis v souboru s odděleným podpisem představuje samostatný balíček.

Obrázek 3.10: Soubor s odděleným podpisem



Zdroj: vlastní zpracování

```
# Zobrazení struktury souboru s odděleným podpisem: --list-packets
gpg --list-packets soubor5.sig
# off=0 ctb=89 tag=2 hlen=3 plen=540
:signature packet: algo 1, keyid 92001940314FDED5
        version 4, created 1481311887, md5len 0, sigclass 0x00
        digest algo 10, begin of digest f1 46
        hashed subpkt 2 len 4 (sig created 2016-12-09)
        subpkt 16 len 8 (issuer key ID 92001940314FDED5)
        data: [4093 bits]
```

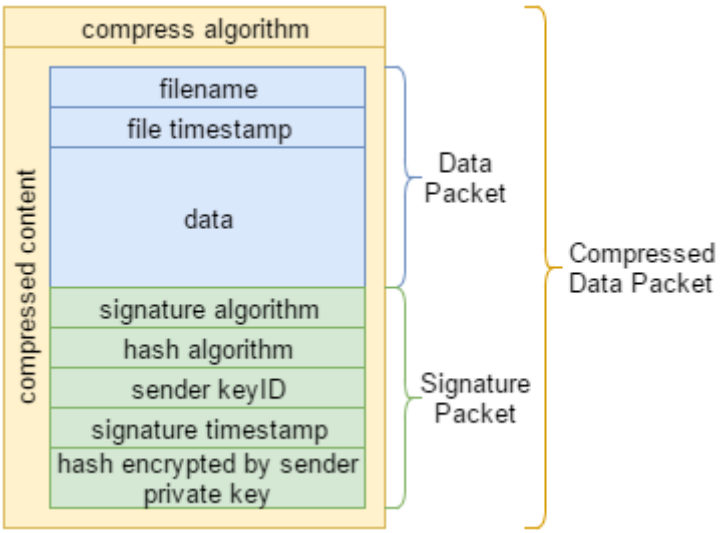
Podepsaný soubor (Obrázek 3.11) obsahuje *tří balíčky* – vedle balíčku podpisu též balíček s vlastními daty, původním jménem souboru a časem poslední změny souboru. Tyto dva balíčky jsou vloženy do balíčku pro kompresi dat. Součástí je i informace, který kompresní algoritmus byl použit (ZIP, ZLIB či BZIP2).

Ještě si ukážeme strukturu **podepsaného a zašifrovaného souboru** (Obrázek 3.12). Na začátku jsou balíčky se zašifrovaným *klíčem sezení* – pro každého příjemce jeden balíček. V každém z nich se klíč sezení šifruje veřejným klíčem příjemce pomocí odpovídajícího algoritmu veřejného klíče (např. RSA). Součástí zašifrovaného klíče sezení je i identifikace použitého algoritmu symetrické šifry (např. AES128, AES192, AES256, Blowfish, Twofish, IDEA atd.). Aktuální přehled podporovaných algoritmů pro danou verzi programu získáte příkazem `gpg --version`.

Na začátku balíčku **zašifrovaného symetrickou šifrou** je inicializační vektor pro symetrickou šifru, která se používá v CFB módu. Zašifrovaným obsahem je obvykle

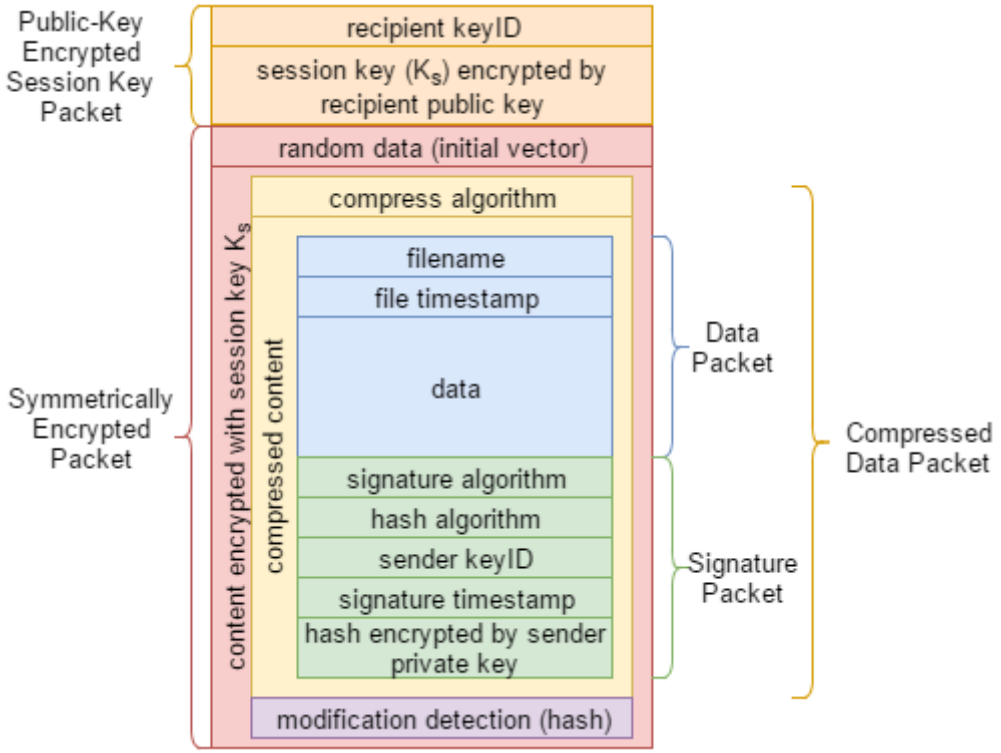
kompresovaný balíček obsahující datový balíček a volitelně digitální podpis. Zašifrovaný obsah je doplněn o balíček s kontrolním hašem, který se počítá pomocí SHA1 z vlastního obsahu před zašifrováním. Tento kontrolní haš se používá pro kontrolu dešifrování – zjistí se případné chyby při přenosu či úmyslné záměny obsahu nebo klíče sezení.

Obrázek 3.11: Struktura podepsaného souboru



Zdroj: vlastní zpracování

Obrázek 3.12: Struktura podepsaného a zašifrovaného souboru s jedním příjemcem



Zdroj: vlastní zpracování

.

3.6.2 ASCII formát – armor, Base64, Radix-64

Při zadání přepínače `--armor` program `gpg` nevytvoří binární výstupní soubor, ale textový výstupní soubor, kdy jsou data převedena do ASCII formátu (obvykle s koncovkou `.asc`). Anglické slovo **armor** má význam *obrnit* či ***vyztužit***. Tento formát vznikl před vytvořením a rozšířením standardu MIME pro posílání binárních souborů v elektronické poště. Cílem bylo vytvořit takový tvar, který bezpečně projde tehdejší 7bitovou elektronickou poštou.

```
# Podepsaný soubor v ASCII formátu, tj. po zadání přepínače --armor

-----BEGIN PGP MESSAGE-----

owEBXAGj/pANAwAKAVMfPJeE7e0DAawVYgF4WIEYNHRvdG8gamUgdGVzdHMKiQEz
BAABCgAdFiEEoWkxbmW1+POTrJiGUx88l4Tt7QMFAliBGDQACgkQUx88l4Tt7QMf
UQf/XaiIq/JlOoqCDDwG243feivpnKlq2ei+IRHxgtw6LdlD5Tu75aU/lHjuhdsY
RIbcL5CZ18QZgjkpxaRMy9l0irv7MEtoLCdzDsgjSknfsYG0Fm/LFmWTqNuynsQ0
ad2yKrk3kLkrT8I8LE/YyfEHOc6kGBcYkIwbnr6auy2/xIy1ccV9eWTJtsHnlemx
ZMWNNhXxkpRTpaRaNQ4ulAtLgB2bUbcwvu8K7lcYj1ideX/r46oUSZR55aiM65+0
4I/OlD89Ut7cz5hv6qaFtSfBO8kozoehLH6mBGthU9M3Ga216S2nxOwe3Uk/Q6Kg
CSj4TcfOfKhK1MQrSIkhrTwwiw==
=B68t
-----END PGP MESSAGE-----
```

ASCII formát obsahuje *vždy* úvodní a závěrečný řádek označující začátek a konec dat.
To umožňuje vložit textový soubor do textu dopisu a při přijetí z něho vybrat. V textu dopisu (obecně souboru) může být vloženo více OpenPGP „souborů“, např. více podepsaných klíčů. Mezi úvodní a závěrečnou hlavičkou je binární soubor převedený do *Radix-64*, což je formát založený na kódování *Base64*.

```
# Exportovaný veřejný klíč v ASCII formátu (přepínač --armor)

-----BEGIN PGP PUBLIC KEY BLOCK-----

mDMEWEwpORYJKwYBBAHaRw8BAQdAac/Gv5pouTyP4bnARoPkkRxc3FngXO8Sb9oJ
IuTnVGe0HXRlc3QgZWNjIDx0ZXN0ZWNjQGJpcy52c2UuY3o+iJAEExYKADgWIQTA
VdIAWniM8mYD4fA0IaqYT6qPiwUCWEwpOQIbAwULCQgHAwUVCgkICwUWAwIBAAIe
AQIXgAAKCRA0IaqYT6qPi1sgAQCEWZ/RBi9/Mds4aTl/jylbOtcjXSZ8OtC+yj/D
Wp+6AAD/bd4dciYvGXMR2P+ocEuwEzbfPfxMV/Iz57Pw3l7XeQu4OARYTCk5Egor
BgEEAZdVAQUBAQdAMmJZ6Q6qiYbXc4VroDcPF0X9QXz1jZDFfByHmIOU6E0DAQgH
iHgEGBYKACAWIQTAVdIAWniM8mYD4fA0IaqYT6qPiwUCWEwpOQIbDAAKCRA0IaqY
T6qPi1zmAPwPjVMdaoUAjD+upJWLNfLmPADv6u/ZY6P7EwOKZ/Ie3QEA35RqdMR/
Ar5BKcU2W/ITaY8+3UjlEBu0tIXkfi+y1AI=
=eP9M
-----END PGP PUBLIC KEY BLOCK-----
```

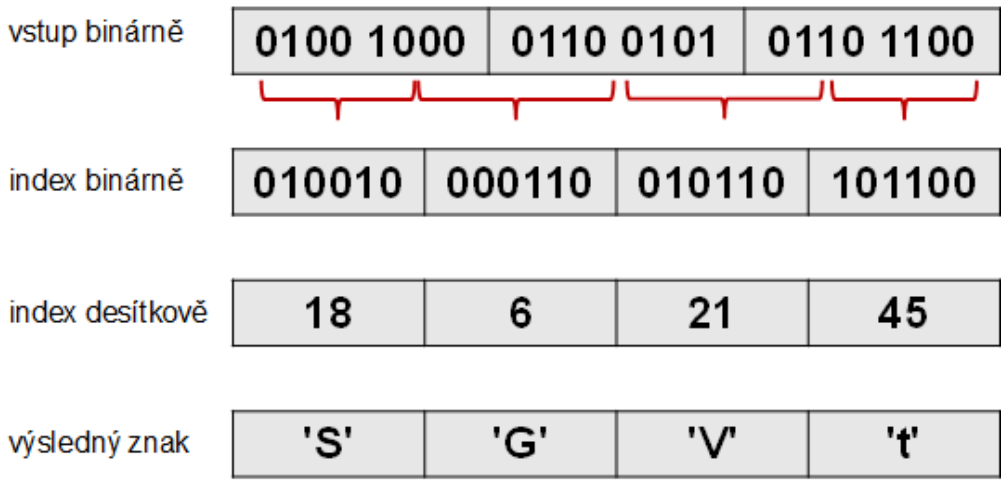
Kódování **Base64** je *skupina kódovacích schémat*, která převádí binární data na posloupnost 64 tisknutelných znaků z ASCII tabulky. Nejčastěji se používá v elektronické poště pro přenos binárních souborů nebo též při přenosu zašifrovaného obsahu či digitálního podpisu. Sadu 64 znaků tvoří písmena malé a velké anglické abecedy, číslice (26 + 26 +10 = 62 znaků) a dva další znaky. Podle RFC 2045 (použito i v OpenPGP, RFC 4880) tyto dva zbylé znaky jsou „+“ a „/“ (jak můžete vidět i v ukázkách výše), ale existuje více variant.

V RFC 4648 (sekce 5), je definována varianta označovaná jako *base64url*. Jak naznačuje již název, tato varianta se používá pro bezpečné překódování „nevhodných“ znaků v URL a zde se místo „+“ a „/“ používá pomlčka „-“ a podtržítko „_“.

Při převodu do Base64 se nejdříve vytvoří tabulka, která k indexům 0 až 63 přiřadí konkrétní znaky (0–25: A–Z, 26–51: a–z, 52–61: 0 až 9, 62: „+“, 63: „/“. Binární data se rozdělí pro třech oktetech (bytech), každá trojice se poté převede na čtyři tisknutelné znaky (viz Obrázek 3.13). Z toho logicky vyplývá, že binární data se při převodu do Base64 zvětší asi o 33 procent.[143] Pokud binární vstup nevychází přesně na trojice, zakóduje se poslední jeden/dva znaky a výstup se doplní o dvě či jedno rovnítko „=“, tzv. *padding*.

Varianta **Base64** specifikovaná v RFC 4800 a označovaná jako **Radix-64** (případně *Radix64*) a používaná v OpenPGP omezuje maximální *délku řádku na 76 znaků*, protože poštovní klienti často automaticky zalamovaly řádky po 80 znacích. Na konec souboru se doplňuje *kontrolní součet* CRC-16, který ve zde použité verzi má 24 bitů a je opět převedený do Base64. Navíc je uvozen jedním znakem „=“ (v ukázce výše to je řádek `=eP9M`).[144]

Obrázek 3.13: Zakódování tři binárních oktetů na 4 znaky Base64 tabulky



[143] Vedle *Base64* existuje i *Base32* se 32 tisknutelnými znaky a *Base16*, která označuje zakódování binárních dat pomocí šestnáctkové (hexadecimální) číselné soustavy. Při převodu do Base32 se zpráva zvětší přibližně o 60 %, v případě Base16 je výsledná velikost dvojnásobná.

[144] CRC-16 (z anglického Cyclic Redundancy Check, cyklický redundantní součet) je jedna z variant (dalšími jsou např. CRC-32, CRC-64) speciální hašovací funkce, která je určená pro detekci chyb při přenosu či ukládání dat. Někdy lze na základě CRC chybu nejen detekovat, ale i opravit. CRC však *nejsou kryptografické* hašovací funkce!

3.7 Různé

Tato část obsahuje *rozšiřující témata*. Začneme *modelem důvěry* nazvaným TOFU, ukážeme jak lze pomocí `gpg` šifrovat hesla pro jiné aplikace, dále podrobněji popíšeme fungování programu `gpg-agent`. A též si ukážeme použití `gpg` klíčů pro autentizaci pomocí `ssh`.

3.7.1 Důvěřuj při prvním styku (TOFU)

V kapitole 3.4.8 byl popsán *klasický model důvěry* obsažený v OpenPGP – platné jsou klíče podepsané Vámi nebo důvěryhodnými osobami (*pavučina důvěry*). V roce 2015 byl program `gpg` rozšířen o model *Trust On First Use (TOFU)* – důvěřuj při prvním užití.[145]

Princip je jednoduchý – když přijde první podepsaná zpráva z nějaké e-mailové adresy, tak se z veřejného klíčového serveru stáhne klíč a označí jako částečně platný – je přidána nová úroveň platnosti. U několika následných použití uvedeného klíče (ověření podpisu či zašifrování pro dotyčného) se zobrazuje upozornění *There is limited assurance this key belongs to the named user*. Poté se stane plně platným.

U všech dalších zpráv se kontroluje, zda se pro danou e-mailovou adresu klíč nezměnil: v tomto případě se zobrazí upozornění na možné podvržení klíče. Je též možný MitM útok při prvním použití klíče. Útočník ale potom musí podvrhovat klíč i při následující komunikaci, jinak je velmi pravděpodobné odhalení útoku.

Model TOFU se používá při ověřování důvěryhodnost serveru v SSH. U OpenPGP chybí dotaz na ověření otisku serveru při prvním výskytu.[146] TOFU je méně bezpečný než klasický model – ověřuje se pouze přiřazení stejného klíče k e-mailové adrese, neověřuje se identita držitele soukromého klíče.

Model TOFU vyžaduje minimální aktivitu uživatelů, což paradoxně může vést k lepšímu zabezpečení než klasický model, ve kterém uživatelé neověřují uživatele a podepisují cizí klíče automaticky. Bezpečnost TOFU se zvyšuje se zvětšujícím se počtem vyměněných podepsaných zpráv. Výhodou je i prodlužování platnosti klíčů vlastníky – pokud někdo stejným klíčem podepisuje většinu zpráv více let, tak zpráva podepsaná jiným klíčem pro stejnou identitu je automaticky podezřelá a `gpg` na ni upozorní.

3.7.2 Šifrování hesel pro další aplikace

`Gpg` umožňuje bezpečné uložení hesel pro další aplikace a jejich zpřístupnění pomocí hesla k soukromému klíči. Příkladem může být poštovní program `mutt`, který umí číst zprávy na IMAP serveru a odesílat je přes autentizované spojení vůči SMTP serveru. Hesla jsou uložena v souboru `.mutt/passwords`:

```
set imap_pass="heslo_k_imap_serveru"
set smtp_pass="heslo_k_smtp_serveru"
```

[145] WALFIELD, Neal H.; KOCH, Werner. TOFU for OpenPGP. In: Proceedings of the 9th European Workshop on System Security. *ACM*, 2016. p. 2.

[146] Pomocí přepínače či parametru konfiguračního souboru lze nastavit, že se program `gpg` bude při prvním výskytu ptát podobně jako `ssh`.

Dále v konfiguračním souboru `.mutt/muttrc` bude následující řádek:

```
source ~/.mutt/passwords
```

Toto není bezpečné uložení hesel. Může si je přečíst administrátor, budou uloženy na zálohách systému. Pokud není zašifrován disk, tak si je může přečíst i případný zloděj počítače. Řešením je soubor s hesly zašifrovat pro sebe pomocí `gpg` a původní soubor následně smazat:

```
gpg --recipient vaše_keyid --encrypt ~/.mutt/passwords
rm ~/.mutt/passwords
```

V souboru `.mutt/muttrc` se konfigurace upraví následovně:

```
source "gpg --no-tty -qd ~/.mutt/passwords.gpg \|"
```

Při spuštění programu `mutt` se program `gpg` zeptá na heslo k soukromému klíči (pokud není v `gpg-agent`), dešifruje soubor s hesly a přes rouru předá obsah programu `mutt`. Z `gpg` se tak stává správce ostatních hesel.

Podrobnější popis včetně dalších bezpečnostních doporučení najdete v článku https://wiki.xmission.com/Hosted_Email:Mutt. Ne všechny programy popisované řešení umožňují.

3.7.3 gpg-agent – správa soukromých klíčů a hesel pro aplikace

Program `gpg-agent` běží na pozadí (daemon) a spravuje soukromé klíče a hesla pro aplikace. Od verze 2.0 se tento uživatelský daemon spouští automaticky.

Pokud program `gpg` potřebuje soukromý klíč, tak kontaktuje proces `gpg-agent`. Pokud je klíč již v paměti, tak ho poskytne žádající aplikaci. Pokud v paměti není, tak se ho pokusí načíst z klíčenky. Přitom se zeptá na heslo k soukromému klíči. Pro dotazy na heslo se používá některá z aplikací *pinentry*. Existují verze pro textové i pro grafické rozhraní.

`Gpg-agent` při spuštění čte konfigurační souboru `~/.gnupg/gpg-agent.conf`. Můžete nastavit např. následující parametry (další možnosti jsou popsány v manuálových stránkách):

```
default-cache-ttl 1800
max-cache-ttl 43200
```

Parametr default-cache-ttl udává čas neaktivity ve vteřinách. Přednastaveno je 600 vteřin: pokud se 600 vteřin konkrétní klíč či konkrétní heslo nepoužije, tak se vymaže z paměti. Druhý parametr max-cache-ttl má přednastaveno 7200 vteřin (2 hodiny) – klíč se smaže z paměti po dvou hodinách od nahrání i když se průběžně používá. Uživatel poté musí znovu zadat heslo ke klíči.

3.7.4 Klíče pro autentizaci

Napadla Vás někdy otázka, zda by nebylo možné použít RSA klíče z *OpenSSH* též v *OpenPGP* či obráceně? Nejdříve je potřeba si uvědomit, že v SSH je jedna dvojice klíčů, kdežto v OpenPGP je hierarchie klíčů – hlavní dvojice je pro podepisování klíčů a případně

dokumentů. Pro další užití jsou podklíče. Je definováno i užití *Authentication* – určené pro autentizaci uživatele. A SSH klíče se používají pro autentizaci.

`gpg-agent` umí simulovat chování programu `ssh-agent`. Tedy nahraje autentizační klíče do paměti a programu `ssh` je poskytuje stejným způsobem, jako to dělá `ssh-agent`. V tomto případě ale nesmí být spuštěn program `ssh-agent`. `gpg-agent` nabídne k autentizaci podklíče s užitím pro autentizaci, je schopen nabídnou i ssh klíče vygenerované mimo gpg.

Klíče pro autentizaci můžete vytvořit při expertní úrovní editace klíče:

```
gpg --expert --edit-key keyid
```

Další klíč přidáte pomocí povelu `addkey` – na expertní úrovni se Vám nabídnou volby s možností nastavení užití vytvářeného klíče (*set your own capabilities*). V dalším kroku nastavíte vytvářené dvojici klíčů užití na autentizaci.

Do konfiguračního souboru ~/.gnupg/gpg-agent.conf je potřeba doplnit řádek:

```
enable-ssh-support
```

Otisk (tzv. keygrip) konkrétního klíče pro SSH autentizaci je potřeba zapsat do souboru `~/.gnupg/sshcontrol`:

```
# Zjištění „keygrip“ autentizačního klíče
# a zapsání do souboru .gnupg/sshcontrol.
gpg -k --with-keygrip
/home/pavlicek/.gnupg/pubring.kbx
---------------------------------
pub   rsa2048 2018-03-14 [SC] [expires: 2020-03-13]
      AB96794BC38AC89538F5DB0DDEAF94B63E7B49C5
      Keygrip = 7C9D19B7815151B91E7DAD85BABC003CBFA8E581
uid           [ultimate] Lubos Pavlicek <pavlicek@bis.vse.cz>
sub   rsa2048 2018-03-14 [E] [expires: 2020-03-13]
      Keygrip = A8BE3E60570B41FE19E3101AB2EE0A5174F99CAB
sub   ed25519 2018-03-14 [A]
      Keygrip = 541E8E50D76897CB55227B66CF237194498661E0

echo 541E8E50D76897CB55227B66CF237194498661E0 > ~/.gnupg/sshcontrol
```

Dále je potřeba

```
export SSH_AUTH_SOCK=$(gpgconf --list-dirs agent-ssh-socket)
```

Poslední krok je zjistit veřejný klíč ve formátu použitelném do souboru `.ssh/authorized_keys` pomocí programu `ssh-add -l`. V následující ukázce se vypíše klíče vygenerovaný algoritmem *ed25519* (Curver 25519).

```
ssh-add -l
256 SHA256:fHPpI2b21jsBlV14X7i6zP4/3fPntday3laXiLO9OsY (none) (ED25519)
```

Podrobný popis najdete v článku „Using a GPG key for SSH Authentication“.[147] Ve Windows může program `gpg-agent` nahradit aplikaci `pagent` z PuTTY. Výhodou je řádově bezpečnější uložení klíčů i možnost nastavení dotazu na heslo při delší době nepoužití klíče.

[147] LUE, Ryan. *Using a GPG key for SSH Authentication* [online]. 2017-06-29 [cit. 2020-10-25] Dostupné z WWW: <https://ryanlue.com/posts/2017-06-29-gpg-for-ssh-auth>.

V článku „How to use a GPG key for SSH authentication“[148] je popsáno využití hardwarového tokenu pro uložení soukromého klíče. Neexistuje snadný způsob, jak do klíčenky naimportovat dvojici klíčů vytvořenou pro SSH. Problematický je i export soukromého klíče pro autentizaci z klíčenky do formátu, který používá SSH.

3.8 Otázky a neřešené úkoly

1. Jak se v *OpenPGP* zašifruje soubor, pokud zadáte dva příjemce? Bude mít zašifrovaný soubor dvojnásobnou velikost ve srovnání se situací, když vstupní soubor zašifrujete pouze pro jednoho příjemce?
2. Jaké problémy spojené s ochranou soukromí vznikají při používání veřejného klíčového serveru (*keyserver*)?
3. Jaké problémy mohou nastat, pokud delší dobu neaktualizujete klíče ve své klíčence (*keyring*) dle stavu na klíčovém serveru (*keyserver*)?
4. Pomocný program *caff* pro podepisování klíčů po podepsání podepíše každou e-mailovou adresu v klíči samostatně (tj. pokud jsou tři e-mailové adresy, podepíše klíč třikrát nezávisle). Každý podpis klíče poté odešle na podepsanou e-mailovou adresu. Podepsaný klíč se posílá podepsaný a zašifrovaný veřejným klíčem příjemce. Jaké má tento postup výhody z hlediska bezpečnosti proti prostému nahrání podepsaného klíče na keyserver?
5. Popište, jak byste poslali e-mail s adresou odesílatele *pavlicek@vse.cz* (tj. cizí e-mailová adresa), který by byl podepsaný pomocí *OpenPGP* veřejným klíčem s adresou *pavlicek@vse.cz* a tento klíč byl dohledatelný na veřejných keyserverech.
6. Jaké symetrické šifry podporuje *gpg* na *bis.vse.cz*? Která z nich je defaultní při šifrování souboru pomocí sdíleného hesla a jak dlouhý klíč se použije? Která symetrická šifra se použije při hybridním šifrování, tj. při šifrování pro příjemce s veřejným klíčem?
7. Které hašovací funkce podporuje *gpg* na *bis.vse.cz*? Která z nich se standardně použije při podepisování souboru?
8. Jak by se pro Alici zjednodušila kontrola klíče před podepsáním v případě, že by Alice s Bobem delší dobu předem komunikovali e-mailem a Bob své dopisy podepisoval?
9. Oddělené podpisy se často používají při distribuci software. Např. si můžete stáhnout archív *.tar.gz* poslední verze webového serveru Apache z jeho stránek. K dispozici je též SHA1 haš z tohoto archivu a oddělený OpenPGP podpis. Proti jakým problémům pomůže kontrola SHA1 haše? Proti jakým útokům pomůže kontrola digitálního podpisu z PGP souboru? Jakým útokům brání distribuce přes HTTPS (oproti HTTP)?
10. V kapitole 3.4.1 je popsán MitM útok na distribuci klíčů. Znemožnil by se útok MitM, pokud by se Alice s Bobem nejdříve domluvili na sdíleném klíči pomocí Diffie Hellman Key Exchange?

[148] *How to use a GPG key for SSH authentication* [online]. 2020-10-07 [cit. 2020-10-25] Dostupné z WWW: <https://www.linode.com/docs/guides/gpg-key-for-ssh-authentication/>

Seznam literatury

1Password: Password Manager for Families, Businesses, Teams [online]. [cit. 2020-08-08] Dostupné z WWW: <https://1password.com/>

Akamai's 2019 State of the Internet / Security: Phishing – Baiting the Hook [online]. [cit. 2020-10-31] Dostupné z WWW: <https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/soti-security-phishing-baiting-the-hook-report-2019.pdf>

Akamai's state of the Internet security. Q1 2016 report [online]. [cit. 2020-08-08] Dostupné z WWW: <https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/akamai-q1-2016-state-of-the-internet-security-report.pdf>

Apache: HTTP Server Project [online]. [cit. 2020-10-31] Dostupné z WWW: <https://httpd.apache.org/>

ATM cash-out attacks [online]. 2018-09-05 [cit. 2020-10-25] Dostupné z WWW: <https://www.enisa.europa.eu/publications/info-notes/atm-cash-out-attacks>

AYER, Andrew. *git-crypt – transparent file encryption in git* [online]. [cit. 2020-10-25] Dostupné z WWW: <https://www.agwa.name/projects/git-crypt/>

BARRETT, Brian. *How 18 Malware Apps Snuck Into Apple's App Store* [online]. 2019-10-25 [cit. 2020-10-25] Dostupné z WWW: <https://www.wired.com/story/apple-app-store-malware-click-fraud/>

BONNEAU, Joseph, PREIBUSCH, Sören; ANDERSON, Ross. A birthday present every eleven wallets? The security of customer-chosen banking PINs. In: *Financial Cryptography and Data Security*. Berlin, Heidelberg: Springer, 2012. s. 25–40.

BONNEAU, Joseph. *Deep Dive: EFF's New Wordlists for Random Passphrases* [online]. 2016-07-19 [cit. 2020-08-08] Dostupné z WWW: <https://www.eff.org/deeplinks/2016/07/new-wordlists-random-passphrases>

BONNEAU, Joseph. The science of guessing: analyzing an anonymized corpus of 70 million passwords. In: *The Security and Privacy (SP), 2012 IEEE Symposium on*, 2012, p. 538–552.

BONNEAU, Joseph; Schechter, Stuart. Towards reliable storage of 56-bit secrets in human memory. In: *23rd USENIX Security Symposium (USENIX Security 14)*. 2014. p. 607–623.

BURNETT, Mark. *10,000 Top Passwords* [online]. 2011-06-21 [cit. 2020-10-25] Dostupné z WWW: <https://xato.net/10-000-top-passwords-6d6380716fe0>

BURR, William E.; DODSON, Donna F.; POLK, William T. *Electronic authentication guideline*. US Department of Commerce, Technology Administration. Gaithersburg: National Institute of Standards and Technology, 2004.

BURR, William. E. et al. *Electronic Authentication Guideline*. NIST Special Publication 800-63-2. Gaithersburg: NIST, 2013. DOI: 10.6028/NIST.SP.800-63-2

Buying Battles in the War on Twitter Spam [online]. 2013-08-14 [cit. 2020-08-08]. Dostupné z WWW: <https://krebsonsecurity.com/2013/08/buying-battles-in-the-war-on-twitter-spam/>

Consumer Survey: Password Habits [online]. 2012-09 [cit. 2020-10-25] Dostupné z WWW: <https://www.csid.com/wp-content/uploads/2012/09/CS_PasswordSurvey_FullReport_FINAL.pdf>

Dashlane: Password Manager App for Home, Mobile, Business [online]. [cit. 2020-08-08] Dostupné z WWW: <https://www.dashlane.com/>

DOČEKAL, Daniel: *TIP#175: Jaká jsou nejpoužívanější hesla a jak vůbec zacházet s hesly na Internetu?* 2015-06-24 [cit. 2020-10-25]. Dostupné z WWW: <https://365tipu.wordpress.com/2015/06/24/tip175-jaka-jsou-nejpouzivanejsi-hesla-a-jak-vubec-zachazet-s-hesly-na-internetu/>

DOUCEK, Petr; PAVLÍČEK, Luboš; SEDLÁČEK, Jiří; NEDOMOVÁ, Lea. Adaptation of password strength estimators to a non-English environment — the Czech experience. *Computers & Security*. Aug 2020, Vol. 95, No. 8, p. 1–11. ISSN 0167-4048 DOI:10.1016/j.cose.2020.101757

DREPPER, Ulrich. *Unix crypt with SHA-256/512* [online]. 2007-09-19 [cit. 2020-08-08] Dostupné z WWW: <https://www.akkadia.org/drepper/sha-crypt.html>

EGELMAN, Serge, et al. Does my password go up to eleven?: the impact of password meters on password selection. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2013. p. 2379–2388

ENISA warns: increase in ATM crime [online]. 2009-09-07 [cit. 2020-10-25] Dostupné z WWW: <https://www.enisa.europa.eu/news/enisa-news/enisa-warns-increase-in-atm-crime>

ENISA: *Algorithms, key size and parameters report 2014* [online]. 2014-11-21 [cit. 2020-10-25]. Dostupné z WWW: <https://www.enisa.europa.eu/publications/algorithms-key-size-and-parameters-report-2014>

Enpass: Password Manager for iOS, Android, Linux, Windows, Mac [online]. [cit. 2020-08-08] Dostupné z WWW: <https://www.enpass.io/>

FERRARA, Anthony. *Why I Don't Recommend Scrypt* [online]. 2014-03-12. [cit. 2020-08-08] Dostupné z WWW: <https://blog.ircmaxell.com/2014/03/why-i-dont-recommend-scrypt.html>

FLORENCIO, Dinei, HERLEY, Cormac. Is everything we know about password stealing wrong? *IEEE Security & Privacy*, 2012, 10.6: 63–69.

Free Rainbow Tables: Distributed Rainbow Table Project [online]. [cit. 2020-08-08] Dostupné z WWW: <https://www.freerainbowtables.com/>

GELINAS, James. *Check your phone for these dangerous apps with 335 million installs* [online]. 2019-10-03 [cit. 2020-10-25] Dostupné z WWW: <https://www.komando.com/technology/infected-apps-google-play-malware/601727/>

GHAZVININEJAD, Marjan; KNIGHT, Kevin. How to Memorize a Random 60-Bit String. *Parking*, 11.70.8: 58.83.

GOSNEY, Jeremi M. *World's First 8x R9 290X oclHashcat Benchmark* [online]. [cit. 2020-10-25] Dostupné z WWW: <https://gist.github.com/epixoip/8171031>

GRASSI, Paul. A., GARCIA, Michael E., FENTON, James L. *Digital identity guidelines: Authentication and lifecycle management*. NIST Special Publication 800-63-3. Gaithersburg: NIST, 2018. DOI: 10.6028/NIST.SP.800-63-3

HERLEY, Cormac. So long, and no thanks for the externalities: the rational rejection of security advice by users. In: *Proceedings of the 2009 workshop on New security paradigms workshop (NSPW '09)*. New York, ACM, 2009. s. 133–144.

How to use a GPG key for SSH authentication [online]. 2020-10-07 [cit. 2020-10-25] Dostupné z WWW: <https://www.linode.com/docs/guides/gpg-key-for-ssh-authentication/>

HTTPS encryption traffic on the Internet has exceeded 90% [online]. [cit. 2020-10-10]. Dostupné z WWW: <https://meterpreter.org/https-encryption-traffic/>

HUNT, Troy. *Suggesting you shouldn't digitise your sexual exploits isn't "victim blaming", it's common sense* [online]. 2016-02-19 [cit. 2020-10-25] Dostupné z WWW: <https://www.troyhunt.com/suggesting-you-shouldnt-digitise-your/>

HUNT, Troy. *What do Sony and Yahoo! have in common? Passwords!* [online]. 2012-07-12 [cit. 2020-10-25] Dostupné z WWW: <https://www.troyhunt.com/2012/07/what-do-sony-and-yahoo-have-in-common.html>

HUNT, Troy: *Observations and thoughts on the LinkedIn data breach* [online]. 2016-05-24 [cit. 2020-10-25]. Dostupné z WWW: <https://www.troyhunt.com/observations-and-thoughts-on-the-linkedin-data-breach/>

CHARRINGTON, Dwayne. *The Most Common iPhone Passcodes (and how to guess them)* [online]. 2014-09-22 [cit. 2020-10-25]. Dostupné z WWW: <https://ilikekillnerds.com/2014/09/the-most-common-iphone-passcodes-and-how-to-guess-them/>

CHIASSON, Sonia; VAN OORSCHOT, Paul C. Quantifying the security advantage of password expiration policies. *Designs, Codes and Cryptography*, 2015, 77.2-3: 401–408.

Implementation of SHA512-crypt vs MD5-crypt [online]. 2011-08-16 [cit. 2020-08-08] Dostupné z WWW: <https://www.vidarholen.net/contents/blog/?p=33>

JOHANSSON, Jesper M. *Frequently Asked Questions About Passwords* [online]. 2008-05-20 [cit. 2020-08-08] Dostupné z WWW: <https://technet.microsoft.com/en-us/library/cc512606.aspx>

JOHANSSON, Jesper M. *Security Watch The Most Misunderstood Windows Security Setting of All Time* [online]. 2016-08-31 [cit. 2020-08-08] Dostupné z WWW: <https://technet.microsoft.com/en-us/magazine/2006.08.securitywatch.aspx>

John-the-Ripper [online]. [cit. 2020-08-08] Dostupné z WWW: <https://www.openwall.com/john/>

JONES, J. C. *The end of SHA-1 on the Public Web*. Mozilla Security Blog [online]. 2017-02-23 [cit. 2020-10-25]. Dostupné z WWW: <https://blog.mozilla.org/security/2017/02/23/the-end-of-sha-1-on-the-public-web/>

KABELOVÁ, Alena a Libor DOSTÁLEK. *Velký průvodce protokoly TCP/IP a systémem DNS*. 5. aktualizované vyd. Brno: Computer Press, 2008. ISBN 978-80-251-2236-5.

Kaspersky Lab: *xDedic – the shady world of hacked servers for sale* [online]. 2016-06-15 [cit. 2020-10-25]. Dostupné z WWW: <https://securelist.com/xdedic-the-shady-world-of-hacked-servers-for-sale/75027/>

KeePass Password Safe [online]. [cit. 2020-08-08] Dostupné z WWW: <https://keepass.info/>

KeePass Security [online]. [cit. 2020-08-08]. Dostupné z WWW: <https://keepass.info/help/base/security.html>

KELLEY, Patrick Gage, et al. Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms. In: *Security and Privacy (SP), 2012 IEEE Symposium on*. IEEE, 2012. p. 523–537.

KHANDELWAL, Swati. *Real-World SS7 Attack — Hackers Are Stealing Money From Bank Accounts* [online]. 2017-05-04 [cit. 2020-10-25] Dostupné z WWW: <https://thehackernews.com/2017/05/ss7-vulnerability-bank-hacking.html>

Knappová, Miloslava. *Jak se bude vaše dítě jmenovat?* Praha: Academia, 2010. 783 s. ISBN 978-80-200-1888-5

LastPass: #1 Password Manager & Vault App, Enterprise SSO & MFA [online]. [cit. 2020-08-08] Dostupné z WWW: <https://lastpass.com/>

LUE, Ryan. *Using a GPG key for SSH Authentication* [online]. 2017-06-29 [cit. 2020-10-25] Dostupné z WWW: <https://ryanlue.com/posts/2017-06-29-gpg-for-ssh-auth>

MACKIE, Simon. *Use a Password Hasher to Generate More Secure Passwords* [online]. 2010-08-13 [cit. 2020-08-08] Dostupné z WWW: <https://gigaom.com/2010/08/13/use-a-password-hasher-to-generate-more-secure-passwords/>

MILLER, George A. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological review*, 1956, 63.2: 81

Mining hardware comparison [online]. [cit. 2020-08-08] Dostupné z WWW: <https://en.bitcoin.it/wiki/Mining_hardware_comparison>

MOGG, Trevor: *Netflix has a black market for passwords, and they sell for just 25 cents* [online]. 2016-02-12 [cit. 2020-10-25]. Dostupné z WWW: <https://www.digitaltrends.com/home-theater/netflix-black-market/>

MORRIS, Robert; THOMPSON, Ken. Password security: A case history. *Communications of the ACM*, 1979, 22.11: 594–597.

Mutt Project: Wiki [online]. [cit. 2020-08-08]. Dostupné z WWW: <https://gitlab.com/muttmua/mutt/-/wikis/home>

NORRIS, Jeffrey S. Mission-Critical Development with Open Source Software: Lessons Learned. *IEEE Software*, 2004, 21.1: 42–49.

NÚKIB. *Minimální požadavky na kryptografické algoritmy:* d*oporučení v oblasti kryptografických prostředků* [online]. 2018-11-28 [cit. 2020-10-18] Dostupné z WWW: <https://nukib.cz/download/uredni_deska/Kryptograficke_prostredky_doporuceni_v1.0.pdf>

Office Document Cryptography Structure [online]. [cit. 2020-06-07]. Dostupné z WWW: <https://msdn.microsoft.com/en-us/library/cc313105.aspx>

OKEEFE, Philip. *shard* [online]. [cit. 2020-10-25] Dostupné z WWW: <https://github.com/philwantsfish/shard>

OpenPGP key paper slip generator [online]. [cit. 2020-08-08] Dostupné z WWW: <https://openpgp.quelltextlich.at/slip.html>

OpenPGP Message Format (RFC 4880) [online]. [cit. 2020-08-08] Dostupné z WWW: <https://tools.ietf.org/html/rfc4880>

OPVault Design [online]. [cit. 2020-08-08]. Dostupné z WWW: <https://support.1password.com/opvault-design/>

Passcape wordlist collection 9.2014 [online]. [cit. 2020-08-08] Dostupné z WWW: <https://weakpass.com/list/906>

Password Guidance: Simplifying Your Approach [online]. 2015 [cit. 2020-10-25] Dostupné z WWW: <https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/458857/Password_guidance_-_simplifying_your_approach.pdf>

Password hashing [online]. [cit. 2020-10-18] Dostupné z WWW: <https://github.com/defuse/password-hashing>

Password hashing [online]. [cit. 2020-10-25] Dostupné z WWW: <https://jedisct1.gitbooks.io/libsodium/content/password_hashing/index.html>

PAVLÍČEK, Luboš: *ZXCVBN Password Strength Estimation with Czech or Slovak Dictionaries* [online]. [cit. 2020-10-25] Dostupné z WWW: <https://github.com/lpavlicek/zxcvbn-czech>

PennKey Password Rules [online]. [cit. 2020-10-25] Dostupné z WWW: <https://pennkeysupport.upenn.edu/password-guidelines>

PHC call for submissions [online]. [cit. 2020-08-08] Dostupné z WWW: <https://password-hashing.net/cfh.html>

PHC winner argon2 [online]. [cit. 2020-08-08] Dostupné z WWW: <https://github.com/p-h-c/phc-winner-argon2>

Portable PHP password hashing framework [online]. [cit. 2020-10-18] Dostupné z WWW: <https://www.openwall.com/phpass/>

PROVOS, Niels; MAZIERES, David. A Future-Adaptable Password Scheme. In: *USENIX Annual Technical Conference, FREENIX Track*. 1999. p. 81–91.

Pwned websites [online]. [cit. 2020-10-31] Dostupné z WWW: <https://haveibeenpwned.com/PwnedWebsites>

Rainbow tables tajemství zbavené [online]. [cit. 2020-08-08] Dostupné z WWW: <https://www.soom.cz/clanky/1165--Rainbow-tables-tajemstvi-zbavene>

REINHOLD, Arnold. *Several security tokens with a U.S. penny coin for comparison*. Wikimedia Commons [online]. 2009-03-11 [cit. 2020-10-25] Dostupné z WWW: <https://commons.wikimedia.org/wiki/File:SecurityTokens.CryptoCard.agr.jpg>

RoboForm: Manage your passwords with ease and security [online]. [cit. 2020-08-08] Dostupné z WWW: <https://www.roboform.com/>

SALGADO, Robert. *credmap* [online]. [cit. 2020-10-25] Dostupné z WWW: <https://github.com/lightos/credmap>

SCARFONE, Karen, SOUPPAYA, Murugiah. Special Publication 800-118: *Guide to Enterprise Password Management* (Draft). Gaithersburg: NIST, 2009. [online] [cit. 2020-08-08]. Dostupné z WWW: < https://csrc.nist.gov/csrc/media/publications/sp/800-118/archive/2009-04-21/documents/draft-sp800-118.pdf >

SHANNON, Claude E. Prediction and entropy of printed English. *Bell system technical journal*, 1951, 30.1: 50–64

Slížek, David. *Seznam.cz při registraci sleduje stisky kláves, aby odhalil roboty* [online]. 2016-07-17 [cit. 2020-10-25]. Dostupné z WWW: <https://www.lupa.cz/clanky/seznam-cz-pri-registraci-sleduje-stisky-klaves-aby-odhalil-roboty/>

Směrnice Evropského parlamentu a Rady (EU) 2015/2366 ze dne 25. listopadu 2015 o platebních službách na vnitřním trhu, o změně směrnice 2002/65/ES, 2009/110/ES, 2013/36/EU a nařízení (EU) č. 1093/2010 a o zrušení směrnice 2007/64/ES [online]. 2015-12-23 [cit. 2020-10-25] Dostupné z WWW: <https://eur-lex.europa.eu/legal-content/CS/TXT/?uri=CELEX:02015L2366-20151223>

STEVEN, John. *OWASP Threat Model for Secure Password Storage*. In: OWASP Foundation [online]. [cit. 2020-08-09]. Dostupné z WWW: <https://goo.gl/Spvzs>

Sticky Password: Best password manager and free password safe [online]. [cit. 2020-08-08] Dostupné z WWW: <https://www.stickypassword.com/>

STOCKLEY, Mark. *What Ashley Madison got right* [online]. 2015-08-31 [cit. 2020-08-08]. Dostupné z WWW: <https://nakedsecurity.sophos.com/2015/08/31/what-ashley-madison-got-right/>

STOCKLEY, Mark. *Why you STILL can't trust password strength meters* [online]. 2016-08-17 [cit. 2020-10-25] Dostupné z WWW: <https://nakedsecurity.sophos.com/2016/08/17/why-you-still-cant-trust-password-strength-meters/>.

Tarsnap cryptography [online]. [cit. 2020-08-08] Dostupné z WWW: <https://www.tarsnap.com/crypto.html>

TATHAM, Simon. *PuTTY: a free SSH and Telnet client* [online]. [cit. 2020-08-08] Dostupné z WWW: <https://www.chiark.greenend.org.uk/~sgtatham/putty/>

The GNU Privacy Guard [online]. [cit. 2020-08-08] Dostupné z WWW: <https://gnupg.org/>

The NTLM Authentication Protocol and Security Support Provider [online]. [cit. 2020-08-08] Dostupné z WWW: <http://davenport.sourceforge.net/ntlm.html#theLmResponse>

The problems with forcing regular password expiry [online]. 2016-10-05 [cit. 2020-10-25] Dostupné z WWW: <https://www.ncsc.gov.uk/blog-post/problems-forcing-regular-password-expiry>

TOPONCE, Aaron. *Super Size The Strength Of Your OpenSSH Private Keys* [online]. 2014-12-08 [cit. 2020-08-08]. Dostupné z WWW: <https://pthree.org/2014/12/08/super-size-the-strength-of-your-openssh-private-keys/>

TRIGGS, Robert. *Facial recognition technology explained*. Android Authority [online]. 2019-01-14 [cit. 2020-10-25] Dostupné z WWW: <https://www.androidauthority.com/facial-recognition-technology-explained-800421/>

TULVING, Endel; PATKAU, Jeannette E. Concurrent effects of contextual constraint and word frequency on immediate recall and learning of verbal material. *Canadian Journal of Psychology/Revue canadienne de psychologie*, 1962, 16.2: 83

TURAN, Meltem, et al. NIST Special publication 800-132: *Recommendation for password-based key derivation*. Gaithersburg: NIST, Computer Security Division, Information Technology Laboratory, Technical Report, 2010.

UR, Blase, et al. How does your password measure up? the effect of strength meters on password creation. In: *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*. 2012. p. 65–80.

Use Face ID on your iPhone or iPad Pro [online]. [cit. 2020-10-25] Dostupné z WWW: <https://support.apple.com/en-us/HT208109>

Vícefaktorová autentizace [v InSIS] [online]. [cit. 2020-10-25] Dostupné z WWW: <https://ci.vse.cz/sluzby/dalsi/insis/vicefaktorova-autentizace/>

WALFIELD, Neal H.; KOCH, Werner. TOFU for OpenPGP. In: Proceedings of the 9th European Workshop on System Security. *ACM*, 2016. p. 2.

WANG, Ding, et al. Targeted Online Password Guessing: An Underestimated Threat. In: *CCS '16: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, October 2016, p. 1242–1254.

WASH, Rick, et al. Understanding Password Choices: How Frequently Entered Passwords are Re-used Across Websites. In: *Symposium on Usable Privacy and Security (SOUPS)*. Denver, 2016. ISBN 978-1-931971-31-7

WEIR, Matt, et al. Testing metrics for password creation policies by attacking large sets of revealed passwords. In: *Proceedings of the 17th ACM conference on Computer and communications security*. ACM, 2010. p. 162–175.

WHEELER, Dan: *zxcvbn: realistic password strength estimation* [online]. 2012-04-10 [cit. 2020-10-25] Dostupné z WWW: <https://dropbox.tech/security/zxcvbn-realistic-password-strength-estimation>

Wikipedie: Length extension attack [online]. 2020 [cit. 2020-10-25]. Dostupné z WWW: <https://en.wikipedia.org/w/index.php?title=Length_extension_attack>

ZHANG, Yinqian; MONROSE, Fabian; REITER, Michael K. The security of modern password expiration: An algorithmic framework and empirical analysis. In: *Proceedings of the 17th ACM conference on Computer and communications security*. ACM, 2010. p. 176–186.

Stránky našeho nakladatelství
https://oeconomica.vse.cz/

Název	**Bezpečnost informačních systémů: materiály ke cvičením 1**
Autoři	Ing. Luboš Pavlíček Ing. Jiří Sedláček, Ph.D.
Vydavatel	Vysoká škola ekonomická v Praze Nakladatelství Oeconomica
Doporučeno	pro bakalářské a magisterské studium na VŠE v Praze
Vydání	první
Návrh obálky	Daniel Hamerník, DiS.
Počet stran	162
DTP	Vysoká škola ekonomická v Praze Nakladatelství Oeconomica
Sazba	autoři

Tato publikace neprošla redakční úpravou

ISBN 978-80-245-2403-0

Zdarma ke stažení