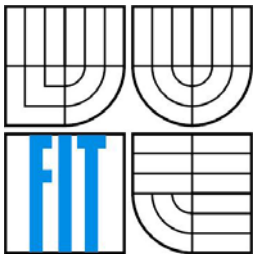
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

BEZPEČNÉ KRYPTOGRAFICKÉ ALGORITMY

SAFE CRYPTOGRAPHY ALGORITHMS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. Lukáš Zbránek
AUTHOR

VEDOUCÍ PRÁCE — Ing. Petr Chmelař
SUPERVISOR

BRNO 2008

Abstrakt

V této práci jsou popsány současné kryptografické algoritmy, porovnání jejich vlastností, silné a slabé stránky a vhodné případy použití jednotlivých algoritmů. Hlavními tématy jsou bezpečnost algoritmů, jejich chyby, vylepšení a odolnost proti průnikům. Jako doplněk k šifrám je věnována pozornost i hašovacím funkcím. Také jsou ukázány nejběžnější metody kryptoanalýzy. Jako použití popsaných algoritmů v praxi uvádím systémy pro zabezpečený přenost dat, kterými jsou SSH a SSL/TLS a je proveden také praktický útok na SSL spojení. V závěru se nachází shrnutí a doporučení vybraných bezpečných algoritmů pro další použití a bezpečné parametry spojení pro SSH a SSL/TLS.

Klíčová slova

bezpečnost, symetrická a asymetrická kryptografie, kryptoanalýza, bloková a proudová šifra, haš, SSH, SSL/TLS, útok, Man-In-The-Middle

Abstract

In this thesis there is description of cryptographic algorithms. Their properties are being compared, weak and strong points and right usage of particular algorithms. The main topics are safeness of algorithms, their bugs and improvements and difficulty of breaching. As a complement to ciphers there are also hash functions taken in consideration. There are also showed the most common methods of cryptanalysis. As a practical application of described algorithms I analyze systems for secure data transfer SSH and SSL/TLS and demonstrate an attack on SSL connection. In conclusion there recomendation of safe algorithms for further usage and safe parameters of SSH and SSL/TLS connections.

Keywords

security, symetric and asymetric cryptography, cryptanalysis, block and stream cipher, hash, SSH, SSL/TLS, attack, Man-In-The-Middle

Citace

Zbránek Lukáš: Bezpečné kryptografické algoritmy. Brno, 2008, diplomová práce, FIT VUT v Brně.

Bezpečné kryptografické algoritmy

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Petra Chmelaře

Další informace mi poskytli Ing. Michal Drozd.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Lukáš Zbránek
4. 5. 2008

Obsah

1 Úvod

Informačním technologiím se podařilo velmi rychle proniknout do všech sfér lidské společnosti. V souvislosti s tím také vyvstala potřeba zachování důvěrnosti dat. Díky tomu a novým technologickým možnostem se začaly rozšiřovat metody šifrování komunikace dříve známé spíše z diplomatických a vojenských kruhů.

Procesu šifrování zpráv se věnuje věda zvaná kryptografie a její počátky se datují až do starověku. Samozřejmě aby byl důvod skrývat obsah zpráv, musel zde existovat někdo, komu se tyto informace neměly donést. Což samozřejmě podnítilo snahy o rozluštění šifer - tím byly položeny základy krytpoanalýze. Celé dějiny kryptografie pak vypadají jako tahová hra, kdy kryptologové vytvoří šifru a vzápětí se kryptoanalytikové snaží najít způsob, jak zjistit obsah takto nově zašifrované zprávy bez znalosti klíče k jejímu dešifrování. Otázkou v tomto případě není jestli se to podaří, ale kdy se to podaří. Toto schéma již bylo prověřeno několika staletími a nic nenasvědčuje tomu, že by se tato situace měla v blízké době nějak rapidně změnit.

S využitím počítačů pro člověka odpadá nutnost šifrovat zprávy znak po znaku, takže běžný uživatel již nemusí být odborník v kryptografii a přesto je mu dána jistá záruka toho, že jeho aktivity nebudou moci býti sledovány či se za něj někdo bude vydávat. Podmínkou pro dosažení tohoto stavu je buď existence takových šifrovacích nástrojů, jejichž prolomení je se stávajícími technickými prostředky nemožné anebo by alespoň prolomení bylo dosažitelné za pomocí neúměrně vysokých prostředků a času, tudíž by se takové pokusy útočníkovi nevyplatily.

Vedlejším efektem takto prudkého rozšíření kryptografie je množství šifrovacích algoritmů, které se najednou začaly používat. Ovšem ne každá šifra, která poskytuje velké množství klíčů, musí být nutně bezpečná. Proto je cílem této diplomové práce vytvořit přehled aktuálně používaných algoritmů, které odolávají známým způsobům prolomení, a jejichž náročnost na prolomení je závislá pouze na délce klíče. Jinými slovy – pokud jediná šance, jak rozluštit zašifrovanou zprávu, je otestování každého z možných klíčů, pak je takovýto algoritmus bezpečný. U takovýchto algoritmů je pak za použití dostatečně dlouhého klíče šance, že pro příštích několik let nebudou moci být lehce prolomeny.

Během této práce jsem nejdříve probral základy modulární aritmetiky, která je využita v asymetrické kryptografii. Tím se dostávám k hlavnímu dělení šifrovacích algoritmů. Nejdříve se v práci zabývám symetrickými šiframi a jejich specifickými vlastnostmi, jako jsou módy blokových šifer. Po symetrických šifrách následuje rozbor vybraných asymetrických šifer, známých také jako šifry s veřejným klíčem. Závěr části o algoritmech je věnován hašovacím funkcím, které nacházejí využití jako samostatný bezpečnostní prvek (pro kontrolu integrity dat), jako součást

kryptografických algoritmů nebo také např jako jeden ze základních prvků generátorů pseudonáhodných čísel.

Další část práce je věnována systémům pro zabezpečení přenosu, které pro svou činnost kombinují několik různých druhů kryptografických algoritmů. Podrobně probírám protokol SSH, jeho různé verze a algoritmy, které používá. Na základě informací získaných při zpracovávání algoritmů jsem byl schopen určit algoritmy, které již do budoucna neposkytují požadovanou bezpečnost. Podobným způsobem jsem následně zpracoval protokol SSL/TLS.

Kromě útoků na kryptografické algoritmy jsem do své práce také zahrnul útoky na systémy, které tyto algoritmy využívají. Praktická ukázka útoku Man-In-The-Middle vedla k prozrazení přihlašovacích informací k českému emailovému serveru centrum.cz.

V závěru práce se nachází shrnutí získaných informací a doporučení postavená na jejich základě.

2 Základy modulární aritmetiky

Předem samotného přehledu různých druhů šifer a jich využívajících systémů bych rád ukázal některé matematické základy, které jsou kryptografii využívány. Informace pro tuto kapitolu jsem čerpal převážně z [46].

Nechť je okruh definován jako množina M s binárními opreacemi sčítání a odčítání, pak předpokládejme, že *R* je jedním z okruhů $\mathbb{Z}$ (celá čísla) nebo $\mathbb{F}_2$[x] (polynom druhého stupně s proměnnou x). Dále pak *m* je prvkem *R*. (*m*) resp. *mR* definujeme jako množinu {*mx : x* ∈ *R*}, kterou nazýváme ideálem *R*. Číslo *m* nazýváme modulem (z ang. modulus) *R/mR*. S těmito informacemi můžeme začít popisovat *podílové okruhy* nebo také *okruhy zbytkových tříd R/mR*.

2.1 Podílové okruhy

Okruh zbytkových tříd *R/mR* je komutativní okruh s prvky nazývanými třídy faktorové množiny *mR* a má následující formu:

$$\bar{a} = a + mR = \{a + mx : x \in R\}, \tag{2.1}$$

pravidla pro sčítání a násobení jsou odvozena od těch, které platí v *R*:

$$\bar{a} + \bar{b} = (a + mR) + (b + mR) = (a + b) + mR = \overline{(a + b)}, \tag{2.2}$$

$$\bar{a} * \bar{b} = (a + mR) * (b + mR) = (a * b) + mR = \overline{(a * b)}. \tag{2.3}$$

Fakt, že *R/mR* je okruh, znamená, že výše zmíněné operace (+) a (*) jsou na třídách faktorových množin definované a splňují zákony asociativity a distributivity.

Příklad. Uvažujme okruh R/mR = $\mathbb{Z}$/m $\mathbb{Z}$ s modulem *m* = 21. Uvažujeme operaci sčítání (+) a násobení (*) pro dva páry prvků z tohoto okruhu: $\overline{2} = \overline{23}$ a $\overline{-2} = \overline{19}$, na kterých si ukážeme, že nezáleží který prvek z každé dvojice použijeme pro výpočet součtu a součinu. Součet bude vypadat následovně:

$$\overline{2} + \overline{-2} = \overline{2 + (-2)} = 0, \tag{2.4}$$

nebo také můžeme počítat:

$$\overline{23} + \overline{19} = \overline{23 + 19} = \overline{42}, \tag{2.5}$$

což je rovno $\overline{0}$, protože 42 je v 21$\mathbb{Z}$. Násobení je stejně jako sčítání nezávislé na prvku, který z vyberem z páru ekvivalentních prvků:

$$\overline{2} * \overline{-2} = \overline{2 + (-2)} = \overline{0}, \tag{2.6}$$

což platí, protože -4 = 17 + (-1) * 21, nebo

$$\overline{23} * \overline{19} = \overline{437} = \overline{17}, \tag{2.7}$$

zde je totožnost výpočtů určena výpočtem 437 = 17 + 420 = 17 + 20 * 21.

2.2 Operátor mod

V obou okruzích $R = \mathbb{Z}$ a $R = \mathbb{F}_2[x]$ máme operátor mod m pro tvorbu nejmenší kanonické reprezentace prvku podílu kruhu R/mR. To znamená, že můžeme pracovat s nejmenším nebo redukovaným reprezentantem ve výpočtech na R/mR. V praxi to znamená, že všechny prvky, které mají po dělení modulem m zbytek z jsou reprezentovány pouze tímto z. Operace sčítání (+) a násobení (*) jsou definovány následovně:

$$\big((a \bmod m) + (b \bmod m)\big) \bmod m = (a + b) \bmod m \tag{2.8}$$

$$\big((a \bmod m) * (b \bmod m)\big) \bmod m = (a * b) \bmod m \tag{2.9}$$

a dále platí, že $a \bmod m = b \bmod m$ tehdy a právě jen tehdy, když $\bar{a} = \bar{b}$. Příležitostně lze také použít podobný binární booleovský operátor $\equiv \bmod m$. Výsledek výrazu $a \equiv b \bmod m$ je pravdivý právě když $\bar{a} = \bar{b}$ nebo také pokud $(a - b) \bmod m$ je 0.

Hodnota $a \bmod m$ může být vypočítána dlouhým dělením (z ang. long division), tj. postupným odčítáním násobků m, dokud není konečný výsledek menší, než m. Definice x je menší než y je $x < y$ pro kladná $x, y \in \mathbb{Z}$ a $deg(x) < deg(y)$ pro polynomy $x, y \in \mathbb{F}_2[x]$.

Příklad. Použijeme operátor mod k nalezení kanonické reprezentace pro x^7 v $\mathbb{F}2[x]/(x2 + x + 1)$. Nejdříve x^7 vyjádříme jako $(x^3)^2 * x$ a vypočteme:

$$\begin{aligned} x^3 \bmod (x^2 + x + 1) &= \big(x^3 + x * (x^2 + x + 1)\big) \bmod (x^2 + x + 1) \\ &= (x^2 + x) \bmod (x^2 + x + 1) \\ &= \big((x^2 + x) + (x^2 + x + 1)\big) \bmod (x^2 + x + 1) = 1. \end{aligned} \tag{2.10}$$

Výsledek první rovnice dosadíme do původního vyjádření x^7 a následně zjistíme, že

$$x^7 \bmod x^2 + x + 1 = (1^2 * x) \bmod (x^2 + x + 1) = x. \tag{2.11}$$

Postupným dělením bychom dokázali, že

$$x^7 = (x^5 + x^4 + x^2 + x) * (x^2 + x + 1) + x, \tag{2.12}$$

čímž potvrzujeme výsledek

$$x^7 \bmod (x^2 + x + 1) = x. \tag{2.13}$$

2.3 Prvočísla a nerozložitelné polynomy

Nenulový ideál (p) v R (=$\mathbb{Z}$ nebo $\mathbb{F}_2[x]$) je prvoideál, pokud je p prvočíslem nebo nerozložitelným polynomem. Následující teorém je zobecněním Malé Fermatovy věty.

Teorém 1. Nechť je (*p*) prvoideálem *R* a nechť *N* je rovno *R/(p) – 1*. Pak platí, že $\overline{a}^N = 1$ pro každé nenulové $a \in R/(p)$. Opačně pokud existuje prvek $a \in R/(p)$ daného řádu *N*, pak je *(p)* prvočíslem.

Dále tvrdíme, že polynom g(x) je primitivní tehdy a právě jen tehdy, když prvek *x* je daného řádu $N \in R/(g(x))$.

2.4 Nerozložitelné polynomy

Teď bychom rádi nalezli nerozložitelné polynomy nízkého řádu v $\mathbb{F}_2$[x], a spolu s tím vysvětlíme některé kroky pro efektivnější určení ne/rozložitelnosti polynomu.

Stupeň 1: lineární polynomy *x* a *x + 1* jsou nerozložitelné.

Stupeň 2: polynom $x^2 + x + 1$ je nerozložitelný na základě předchozího teorému a faktu, že $\overline{x}, \overline{x^2} = \overline{x} + 1$ a $\overline{x^3} = 1$. Na druhou stranu je jasně vidět, že jediní další kandidáti: $x^2, x^2 + x$ a $x^2 + 1 = (x + 1)^2$ jsou rozložitelní.

Lemma 2. Pokud je *f(x)* polynom, pak platí, že *f(x) mod (x – a) = f(a)*, konkrétně pak $f(x) = (\mathrm{x} - \mathrm{a})\mathrm{g}(\mathrm{x})$ tehdy a právě tehdy, když $f(a) = 0$.

Pro polynomy nad $\mathbb{F}_2$ je hodnota *f(0)* konstantou a *f(1)* je číslo nenulových koeficientů *mod 2*, což nám dává jednoduchý test dělitenosti lineárními polynomy.

Stupeň 3: Díky předchozímu testu je zřejmé, že jediní netriviální kandidáti připadající v úvahu jsou

$$x^3 + x + 1$$
$$x^3 + x^2 + 1$$

a tyto polynomy jsou nerozložitelné, protože nemají žádného lineárního činitele.

Stupeň 4: Nejdříve vyloučíme $(x^2 + x + 1) = x^4 + x^2 + 1$, tj. jediný polynom čtvrtého stupně, který je dělitelný nerozložitelným polynomem druhého stupně. Každý další rozložitelný polynom tedy musí mít dělitele prvního stupně a aplikujeme lemma abychom zúžili seznam nerozložitelných polynomů na:

$$x^4 + x^3 + 1, x^4 + x + 1, x^4 + x^3 + x^2 + x + 1.$$

Stupeň 5: Stejně jako pro polynomy čtvrtého stupně, i zde vyloučíme polynomy s dělitelem druhého stupně:

$$(x^2 + x + 1)(x^5 + x + 1) = x^5 + x^4 + 1 \quad (2.14)$$
$$(x^2 + x + 1)(x^3 + x^3 + 1) = x^5 + x^4 + 1 \quad (2.15)$$

po čemž dojdeme k závěru, že všechny ostatní polynomy stupně pět s konstantou 1 a lichým počtem koeficientů jsou nerozložitelné:

$$x^5 + x^3 + 1,\ x^5 + x^2 + 1,$$
$$x^5 + x^4 + x^3 + x^2 + 1,\ x^5 + x^4 + x^3 + x + 1,$$
$$^5 + ^4 + ^2 + \ + 1,\ ^5 + ^3 + ^2 + \ + 1.$$

2.5 Polynomy dělení kruhu (z ang. cyclotomic polynomials)

V předchozí části bylo ukázáno, že existuje šest nerozložitelných polynomů pátého řádu. Pro porozumění nerozložitelným a primitivním polynomům nejdříve představíme polynomy dělení kruhu.

Definice. Polynomy dělení kruhu $\Phi_N(x)$ jsou definovány jako: (2.16)

$$x^N - 1 = \prod_{m|N} \Phi_{\mathrm{m}}\,(x).$$

Příklad. Pro ukázku, jak nám tato informace slouží pro definování polynomů dělení kruhu si ukážeme několik příkladů:

$$\Phi_1(x) = x - 1 \quad (2.17)$$
$$\Phi_2(x) = x + 1 \quad (2.18)$$
$$\Phi_3(x) = x^2 + x + 1 \quad (2.19)$$
$$\Phi_4(x) = x^2 + 1 \quad (2.20)$$
$$\Phi_5(x) = x^4 + x^3 + x^3 + x^2 + x + 1 \quad (2.21)$$
$$\Phi_6(x) = x^2 - x + 1 \quad (2.22)$$

Navíc pokud je (p) prvočíslo, pak platí:

$$\Phi(x) = \frac{x^p - 1}{x - 1} = x^{p-1} + \cdots + x + 1. \quad (2.23)$$

Doposud definici polynomů dělení kruhu nevyužijeme pro polynomy definovanými nad $\mathbb{F}_2$, ale pro polynomy definovanými nad $\mathbb{Z}$ již lze definice využívat.

Teorém 3. Polynom dělení kruhu $\Phi_{\mathrm{N}}(x)$ stupně $\varphi(N)$ je nerozložitelný nad $\mathbb{Z}$.

Funkce $\varphi(N)$ je nazývána jako Eulerova -funkce a je definována následovně: (2.24)

$$\varphi(N) = \prod_{p^r || N} p^{r-1}\,(p - 1),$$

kde $p^r || N$ znamená, že p^r dělí N, ale p^{r_1} již N nedělí.

Analogické tvrzení o nerozdělitelnosti nad $\mathbb{F}_2$ by bylo chybné, ale můžeme formulovat tvrzení o způsobu faktorizace polynomu dělení kruhem nad $\mathbb{F}^2$.

Teorém 4. Nerozložitelný polynom $g(x) \in \mathbb{F}_2[x]$ stupně n dělí polynom $x^N + 1$ a žádný jiný polynom $x^m + 1$ kde $m < N$ tehdy a právě tehdy, když $g(x)$ dělí $\Phi_N(x)$. Celé číslo N je rovno $2^n - 1$ tehdy a právě tehdy, když je $g(x)$ primitivní.

Důsledek 5. Polynom dělení kruhem $\Phi_N(x) \in \mathbb{F}_2[x]$ pro $N = 2^n - 1$ je součin různých primitivních polynomů n-tého stupně.

Příklad. Výše jsme zjistili, že existuje šest nerozložitelných polynomů pátého stupně v $\mathbb{F}_2[x]$. Protože $N = 2^5 - 1 = 31$, což je prvočíslo, každý nerozložitelný polynom pátého stupně je prvočíslo. Protože stupeň $\Phi_{31}(x)$ je $(31) = 31 - 1 = 30$, mohli jsme rovnou určit, že existuje přesně šest primitivních a nerozložitelných polynomů pátého stupně.

2.6 Linear Feedback Shift Registr keystreamy

LFSR je posuvný registr, jehož vstupní bity jsou výstupem funkce jeho aktuálního stavu. S použitím primitivního polynomu, kdy se perioda výstupní sekvence s délkou $N = 2^n - 1$ a zvyšujícím se n zvětšuje exponenciálně, je možné dosáhnout sekvencí s velmi dlouhou periodou. Navíc může být LFSR výpočetně efektivnější s použitím primitivních polynomů s některými koeficienty nulovými (z ang. sparse primitive polynomial) jako např.:

$$14: x^{14} + x^7 + x^5 + x^3 + 1$$

$$15: x^{15} + x^5 + x^4 + x^2 + 1$$

$$16: x^{16} + x^5 + x^3 + x^2 + 1$$

$$17: x^{17} + x^3 + 1$$

Jednoduchý kryptosystém je možné postavit tak, že výstup LFSR je klíčem proudové šifry, který je následně XORován se zprávou. Naneštěstí díky Berlekamp-Masseyho [110] algoritmu stačí znát pouze 2n bitů z keystreamu ke zjištění kompletní sekvence. Takovýto systém by pak nesplňoval základní podmínku bezpečnosti.

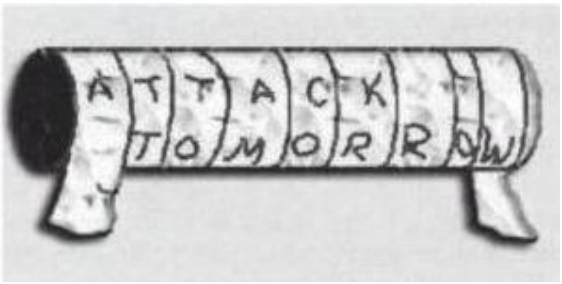
Existuje ale také kryptosystém zvaný Shrinking Generator, který používá dva LFSR, a který zatím úspěšně odolává pokusům o kryptoanalýzu [111].

3 Historie kryptografie

První pokusy o šifrování textu proběhly již starověku, kdy pomocí utajeného textu bylo prozrazeno datum útoku perské armády do Evropy a následkem toho byla perská armáda zastavena ve slavných bitvách u Thermopyl. Zde ještě nešlo o šifrování ve smyslu v jakém jej známe dnes. Podstata skrytí této zprávy zpočívala v tom, že Demaratus seškrábal vosk z dřevěných destiček, zprávu vyryl přímo do dřeva a poté destičky zalil voskem, takže se jevily jako nepoužité. Taková a jí podobné metody jsou dnes známy spíše pod názvem steganografie. Princip zpočívá v ukrytí zprávy, ne v zabránění její čitelnosti. Dnes se používá například zakódování textu do obrázků či hudebních souborů.

3.1 Starověk

Později se v Řecku objevily šifry již založené na mechanických zařízeních, algoritmech či kódových tabulkách. Mezi jednu z nejstarších mechanických šifrovacích pomůcek patří skytála – dřevěná tyč o přesně daném průměru (Obr. 1.1). Na něj se namotal proužek kůže, který byl následně popsán zprávou. Její následné zobraznení tak bylo možné pouze na tyči s přesně stejnou tloušťkou [2], čímž byly položeny základy transpoziční metody.



Obr. 1.1: Příklad skytály, převzato z [2]

Jako další alternativa se ve starém Římě objevila substituční metoda, první takovou byla Caesarova šifra. Zde bylo každé písmeno zprávy posunuto o 3 pozice v abecedě dál [1]. Počet možných klíčů v této šifře je velmi malý (konkrétně vždy o jeden menší než počet písmen použité abecedy), takže v dnešní době tato šifra rozhodně nenajde bezpečné uplatnění. Nicméně princip, na kterém staví se s různými obměnami a v kombinaci s transpozičními metodami používá i v dnešní kryptografii.

3.2 Frekvenční analýza

Od starověku po středověk kryptografie v západním světě stagnovala, používaly se převážně jednoduché substituční šifry. Pokud například ve středověku chtěl pán před svým služebnictvem něco skrýt, nemusel používat příliš sofistikované metody, už jen proto, že úroveň gramotnosti mezi prostým lidem nebyla příliš vysoká. Pokud tedy již dotyční uměli číst, jednoduché šifry, jako například Caesarova, často stačily na ochranu informace před nežádoucími čtenáři.

V arabském světě té doby se ovšem vytvořily velmi vhodné podmínky pro vznik celého vědního oboru kryptoanalýzy. Díky úrovni matematiky, statistiky a jazykovědy si začali místní vědci všímat jazykových chrakteristik. Ve zdejších teologických školách se hlavně do hloubky studoval a rozebíral Korán, kde začly být zřetelné statistické rozdíly v četnosti nejdříve celých slov, později byly tyto rozdíl vztaženy na jednotlivá písmena. Vědci zjistili, že 2 nejčastější písmena používaná v arabštině jsou **„a“** a **„l“**. Na proti tomu písmeno **„j“** se objevuje asi 10x méně často. V souvislosti s tímto nálezem byla zformulována jedna z průlomových technik kryptoanalýzy – frekvenční analýza.

S první ucelenou definicí frekvenční analýzy přišel arabský vědec Abū-Yūsuf Ya’qūb ibn Ishāq al-Kindī v 9. století [4]. Pro její úspěšné použití potřebujeme vědět v jakém jazyce je zpráva napsaná a mít k dispozici dostatečně dlouhý libovolný text ve stejném jazyce. Z tohoto libovolného textu nejdříve vytvoříme pořadí četnosti jednotlivých písmen daného jazyka. Poté jim začneme přiřazovat symboly s podobnou četností ze šifrované zprávy. Pokud se nám tak podaří určit význam alespoň některých symbolů ve zprávě, je velká šance, že takto odkryjeme celý šifrovaný text.

3.3 Vigenerova šifra

V případě Caesarovy šifry šlo o monoalfabetickou substituci, protože byla použita jen jedna abeceda. Navzdory její jednoduchosti byla ve středověku hojně využívána, jen kromě obyčejného posunu písmen bylo přidáno i jejich přeházení, než její slabiny poměrně drasticky demonstrovala poprava Marie Stuartovny. V té době už bylo kryptografům jasné, že je potřeba přijít s mnohem důmyslnějším systémem skrytí obsahu zpráv. Jedním ze způsobů bylo vylepšení stávajících substitučních šifer například přidáním dalších abeced, podle kterých bude zpráva šifrována.

Konečným produktem tohoto vývoje je Vigenerova šifra z konce 16. století, ačkoli její základ se objevil o více než sto let dříve. Jedním z jejich duchovních otců je Francouz Leon Battist Alberti. Ten navrhl přidání další jedné nebo více abeced podle které by se zpráva šifrovala a to tak, že by se použité abecedy střídaly. Při použití dvou abeced (každá z nich vytvořená permutací původní otevřené abecedy) by pak šifrování probíhalo následovně – první písmeno zprávy zašifrováno pomocí první abecedy, druhé písmeno pomocí druhé abecedy, třetí písmeno opět pomocí první a

tak dále. Užitečnou vlastností tohoto způsobu šifrování je to, že stejné písmena otevřeného textu mohou mít přiřazeny různé symboly v šifrované textu. Pravděpodobnost různé reprezentace stejných písmen se zvyšuje s počtem použitých abeced a podstatně znesnadňuje prolomení šifry pomocí frekvenční analýzy. Alberti vytvořil i užitečnou pomůcku zvanou Albertiho disk (Obr. 1.2), který se ovšem dal použít pouze na šifrovací abecedy tvořené pouze posunem písmen jako tomu je například u Caesarovy šifry. Tento nástroj se skládá ze dvou kruhů, na kterých je abeceda, posunem vnitřního kruhu o daný počet pozic pak lze rovnou vidět, jak bude probíhat nahrazování písmen při šifrování zprávy.

Obr. 1.2: Albertiho šifrovací disk [4]

Alberti sice objevil šifru obrovského významu, nicméně ji nedotáhl do konce. V jeho práci pokračovali Johannes Trithemius, Giovanni Porta a hlavně Blaise de Vigenere, který, po prostudování prací svých předchůdců, dovedl tento způsob šifrování do konečné podoby silné šifry, která pak nesla právě jeho jméno – Vigenerova šifra. Je postavena na použití 26 abeced, které jsou tvořeny posunem otevřené abecedy vždy o jednu pozici. Každé písmeno zprávy je pak nahrazeno symbolem z některé abecedy [3]. Jako pomůcka při šifrování zpráv byl vytvořen Vigenerův čtverec (Obr. 1.3) a uplatnění nalezl také Albertiho šifrovací disk, který proces šifrování zpřehlednil.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Obr. 1.3: Vigenerův čtverec

Aby byla zpráva zpětně rozluštitelná, je nutné řádek, ze kterého bude vybrán symbol pro nahrazení původního písmena, vybírat podle nějakého vzoru. Tím vzorem je klíčové slovo, na kterém se musí obě komunikující strany předem dohodnout. Pro názornost tuto techniku přiblížím na jednoduchém příkladu.

Otevřeným textem bude **„bezpecnekryptografickealgoritmy"** a jako klíčové slovo použiji **„fitvutbr"**.

Klíčové slovo: **fitvutbrfitvutbrfitvutbrfitvutb**

Otevřený text: **bezpecnekryptografickealgoritmy**

Výsledná šifra: **gmskyvovpzrmnhhifnbxexbclwkdnfz**

Klíčové slovo jsem opsal tolikrát, jaký je počet znaků otevřeného textu. Ke každému znaku otevřeného textu jsem pak vyhledal písmeno klíče, které udává použitý řádek čtverce. V daném řádku jsem pak už jen vyhledal náhradu za šifrovaný znak otevřeného textu.

Na tomto příkladu je dobře vidět, jak je řešena ochrana proti frekvenční analýze. Díky použití několika různých abeced (v našem příkladu sedm) může být jeden znak zakódován náhodně více různými způsoby. Stejně tak více výskytů jednoho znaku v šifrovaném textu může mít více významů. Tato mnohoznačnost velmi znesnadňuje kryptoanalýzu a prakticky vylučuje možnost úspěšného použití běžné frekvenční analýzy. Spolu s rostoucí délkou klíče roste i složitost šifry, také množství klíčů je obrovské, ať už by to jsou slova ze slovníku nebo vymyšlená. Na svou dobu to byla výborná

šifra a o to více je překvapující fakt, že se příliš nerozšířila. Velkou zásluhu na tom měla zřejmě její složitost a díky tomu i náchylnost na chyby lidského faktoru.

Tato šifra zůstala neprolomena po bezmála 300 let. Teprve až v druhé polovině 19. století navrhl pruský major Kasinski metodu na její prolomení. Její princip spočíval v nalezení délky klíčového slova pomocí opakujících se dvojic znaků. Jakmile útočník zjistí délku klíčového slova **n**, tak ví, že každé **n-tý** symbol je zašifrován jednou abecedou. Pak už jen rozdělí zprávu do skupin podle jednotlivých abeced a frekvenční analýzou zkusí přijít na to, o které abecedy se jedná. Aby ovšem byla tato metod úspěšná, musí se v šifrované zprávě nějaké opakující se dvojice objevit – pokud je zpráva příliš krátká jako ta ve výše uvedeném příkladu, zmíněný typ útoku selže.

3.4 Vernamova šifra

Na počátku 20. století, konkrétně roku 1917, navrhl Gilbert Vernam pro společnost American Telephone &Telegraph systém, který pracoval na principu sčítání náhodných čísel se znaky zprávy. Tato náhodná čísla byla do stroje vkládána pomocí děrného štítku a výsledek operace byl v zápětí odeslán dálnopisem. Tento systém se stal první dokonalou šifrou.

Bohužel byla dokonalá pouze z kryptografického hlediska. Její využití v praxi bylo ovšem limitováno několika faktory. Prvním z nich bylo použití klíče stejně dlouhého jako samotná zpráva. To stavělo odesílatele před problém, jak bezpečně přenést příjemci klíč, aby bylo možno zprávu opět dešifrovat. S dnešním objemem přenášených dat se tato vlastnost stává ještě hůře realizovatelnou. Aby opravdu nebylo možné takto šifrované zprávy rozluštit, každý klíč bylo možno použít maximálně jednou, čímž dále stoupá množství klíčů, které je potřeba bezpečně přenést. Poslední vlastnost klíče byla z hlediska realizce tehdy nejnáročnější a jednoduchou záležitostí to není ani dnes – klíč musí být absolutně náhodný. Proto se při použití generátorů pseudonáhodných čísel pro tvorbu klíčů nejedná o pravou Vernamovu šifru [6].

Výše popsaný systém byl v minulosti pouze několikrát uveden do praxe, jako jeden příklad za všechny jmenujme spojení tzv. horkou linkou mezi hlavami USA a SSSR. Přes velvyslanectví byly vyměňovány pásky s klíči, které byly ihned po použití zničeny. Ovšem s nárůstem objemu přenášené komunikace docházelo k porušování zásady o maximálně jednom použití klíče a místo náhodných dat, se začaly používat generátory pseudonáhodných čísel. Takový generátor byl zabudován ve vysílači i v příjmači zpráv. K jeho nastavení se používal tzv. inicializační vektor a klíč. Na kvalitě takového generátoru pak stojí i padá kryptografická síla celého komunikačního systému [6].

3.5 Enigma

S dalším vývojem se přišlo na to, že pro větší bezpečnost je dobré klíče často obměňovat. K tomuto účelu se ukázaly býti velmi vhodné stroje, které poskytovaly automatizaci některých časově náročných úkonů. Nové technologie díky tomu našly velké využití v období světových válek a během takzvané studené války. V souvislosti s automatizací šifrování na tomto místě nelze nezmínit německý šifrovací stroj Enigma.

Tento přístroj byl patentován v roce 1918 německým vědcem jménem Arthur Scherbius. Základní myšlenka spočívala v několikanásobné substituci znaků, aby se tak zvětšil počet možných abeced. Za tímto účelem používala Enigma 3 rotory, které měly uvnitř pevně spojené vstupní a výstupní kontakty a šifrované písmeno se tak s každý průchodem rotorem změnilo. Pořadí rotorů už ovšem pevně dané nebylo a dále tak zvětšovalo množinu klíčů. Jako další vylepšení pak byl zvětšen počet rotorů na pět, z nich pak byly tři vybrány k šifrování. Za účelem nasazení pro vojenské zpravodajství byla dále zvětšována kryptografická síla stroje – přidání reflektoru způsobilo změnu znaku příchozího z nejlevějšího rotoru a tento nový znak opět procházel všemi rotory, kde byl měněn. Po každém zašifrovaném písmenu se první rotor pootočil, když došel na zarážku (jejíž nastavení bylo součástí klíče), tak se pootočil i druhý rotor, taktéž třetí když došel na zarážku i druhý. Pak zde byla deska se spoji, kde bylo propojeno deset dvojic písmen, která byla prohazována.

Poté, co operátor podle kódové knihy vybral a nastavil rotory a provedl propojení dvojic písmen, samotné šifrování probíhalo takto:

- operátor zadal písmeno na klávesnici
- to se změnilo nejprve na propojovací desce
- pak prošlo postupně všemi rotory – v každém proběhla změna
- na konci prošlo reflektorem a bylo opět změněno
- následoval zpětný průchod rotory
- odtud do propojovací desky, kde bylo opět změněno
- z každého písmena na desce vedl drát k žárovce označující dané písmeno

Výsledné písmeno operátor opsal na papír. Když byla celá zpráva zašifrována, byla odeslána rádiem. [5].

Ačkoli by bylo pro útok typu hrubou silou vyzkoušet i na dnešní poměry enormní množství klíčů, tak některé chyby v návrhu enigmy (např. fakt, že žádné písmeno nemůže být zakódováno samo do sebe) umožnily nejdříve polským a později i britským kryptoanalytikům prolomit tuto šifru. Pro automatizaci luštění šifer sestrojili Poláci přístroj Bomba. V britském kryptoanalytickém ústředí

Bletchley Park pak vznikl stroj Bomb pracující jinak než polská Bomba, ovšem se stejným účinkem. Na jeho vzniku se podílel i vědec Alan Turing [4].

3.6 Současnost

Všechny zmíněné vynálezy a nápady přispěly k rozvoji kryptografie, ale technologický pokrok přenesl její další aplikaci převážně do softwarového a hardwarového vybavení počítačů, případně specializovaných zařízení. Proto je dnes hlavní úlohou lidského faktoru v procesu šifrování hlavně ověření kontrolních údajů jakými je např. platnost certifikátu, kterým se identifikuje odesílatel či příjemce zprávy. Otrocké šifrování zprávy znak po znaku již má na starosti elektronika, která tuto práci zvládá mnohem efektivněji než člověk, na člověku je pak vytvoření dostatečně bezpečných a efektivních algoritmů pro ochranu dat.

4 Šifrovací algoritmy

V této kapitole se dostávám k samotným kryptografickým algoritmům, před samotným popisem je ovšem vhodé ukázat jejich dělení. Základními dvě skupiny algoritmů jsou symetrické a asymetrické, také označované jako šifry s veřejným klíčem. Hlavní rozdíl mezi těmito druhy algoritmů je počet klíčů, se kterými pracují.

Symetrické algoritmy používají pro šifrování i dešifrování stejný klíč, takže z toho výplývá, že tento klíč je nejdůležitější pro uchování tajemství. Klíč symetrické šifry mívá délku 56 bitů a více, což nám dává exponenciálně rostoucí prostor možných klíčů od 2^{56} nahoru, což dává složitost prohledávání prostoru možných klíčů při útoku hrubou silou. Moderní šifry však pracují minimálně se 128 bitovým klíčem, protože aktuálně jsou k dispozici dostatečně výkonná jednoúčelová zařízení, která jsou schopny prohledat stavový prostor o velikosti 2^{56} v řádu dní, např. Copacobana [105] to zvládne v průměru za 7 dní, takže použití takto krátkého klíče již není bezpečné. Současně tvůrci Copacobany tvrdí, že tento přístroj je schopen efektivně prohledávat stavový prostor až do velikosti 2^{64}.

Naproti tomu asymetrické šifry používají dva různé klíče. Jakmile je zpráva zašifrována libovolným z této dvojice klíčů, lze ji dešifrovat pouze druhým klíčem. Tyto šifry jsou založeny na problému faktorizace velkých čísel (RSA) nebo nalezení diskrétního logaritmu (Diffie-Hellman), takže délky jejich klíčů bývají 1024 bitů a výše. Právě problém faktorizace je považován za NP problém, který neumíme řešit v polynomiálním čase.

V třetí části této kapitoly se zabývám kryptografickými hašovacími funkcemi. Nejsou to sice šifrovací algoritmy, nicméně jsou s nimi těsně spjaty a jejich funkce je podobná. Od šifrovacích algoritmů se nejvíce liší tím, že jejich proces zpracování dat má být nevratný.

4.1 Symetrické šifry

Symetrické šifry se dále dělí na blokové a proudové. Jejich rozdíl spočívá v různém zpracování vstupních dat. Blokové šifrovanou zprávu rozdělí do bloků, se které postupně šifrují. Proudové šifry příjmají vstupní data jako neomezený proud a šifrují data okamžitě. Později si ukážeme módy blolkových šifer, které umožní funkci proudových šifer emulovat. Symetrické šifrovací algoritmy také bývají často podstatně rychlejší než asymetrické.

4.1.1 Blokové symetrické šifry

Tyto algoritmy si vstupní text dělí do bloků pevně daných délek (z původních 64 bitů v počátcích je dnes standartních 128 – 256 bitů), se kterými následně pracuje. První šifry kódovaly bloky nezávisle na sobě, pouze podle klíče, vylepšením pozdějších bylo, že vždy následující blok byl šifrován s použitím hodnot klíče a předešlého zašifrovaného bloku, jen pro první blok musí být nastaven inicializační vektor, který plní funkci předchozího bloku. Pro samotné šifrování se pak používají funkce, které provádějí substituci znaků, tzv. S-boxy, a funkce, které provádějí jejich permutace, tzv. P-boxy. Pro kvalitu celého algoritmu je pak životně důležité, aby tyto funkce neobsahovaly žádné slabiny, to by podstatně snížilo bezpečnost daného algoritmu. Blokové šifry se používají ve čtyřech možných režimech, které jsou popsány níže, informace o nich jsem čerpal z [43].

Základní informace o jednotlivých algoritmech jsem čerpal z [12], případné podrobnější informace jsou ze zdrojů uvedených u jednotlivých algoritmů.

Blokové šifry lze provozovat v těchto módech:

- Electronic Code Book mód
- Cipher Block Chaining mód
- Cipher Feedback mód
- Output Feedback mód

Nyní bude následovat podrobný popis jednotlivých módů.

Electronic Code Book mode (ECB)

ECB je základní mód blokových šifer – plaintext je rozdělen do bloků, které jsou následně samostatně šifrovány. Díky níže popsaným vlastnostem se nejedná o vhodný způsob použití blokových šifer, pokud má zpráva délku větší než jeden blok.

ECB Šifrování:

Vstup:	k-bitový klíč	K
	n-bitové bloky plaintextu	$M = M_1 M_2 \dots M_t$
	Algoritmus:	$C_j = E_K (M_j)$
	Výstup: n-bitové bloky šifrovaného textu	$C = C_1 C_2 \dots C_t$

ECB Dešifrování:

Vstup:	k-bitový klíč	K
	n-bitové bloky šifrovaného textu	$C = C_1 C_2 \dots C_t$
	Algoritmus:	$M_j = D_k (C_j)$
	Výstup: n-bitové bloky plaintextu	$M = M_1 M_2 \dots M_t$

Vlastnosti:

1) Pro dva identické bloky plaintextu jsou vygenerovány identické bloky šifrovaného textu - tato vlastnost zabraňuje ve skrytí vzorů ve zprávě (např. obrázek).
2) Přeskládání bloků plaintextu vede pouze ke změně pořadí bloků šifry.
3) Chyba v plaintextu se projeví pouze v odpovídajícím bloku šifry.
4) Pokud útočník nahradí blok C_j blokem C_j', příjemce bude mít v plaintextu podvržený blok M_j'.

Cipher Block Chaining mode (CBC)

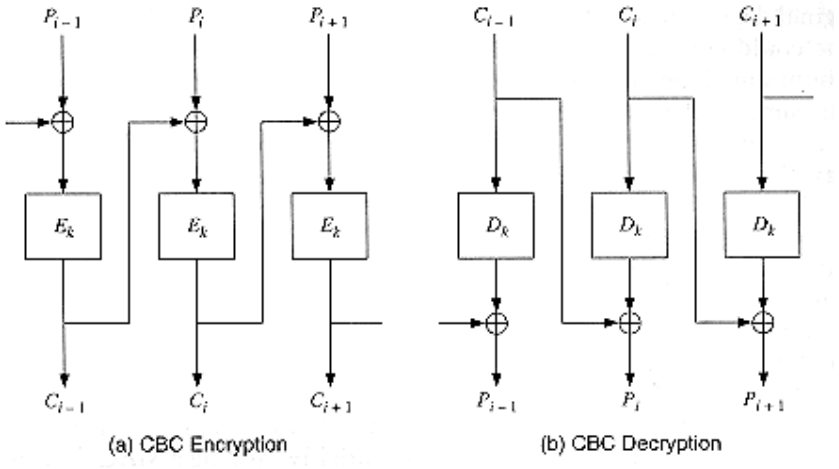
V tomto módu je před samotným procesem šifrování každý blok plaintextu XORován s předcházejícím blokem šifrovaného textu. Pro zašifrování prvního bloku zprávy se používá inicalizační vektor (IV), což může být číslo náhodné, pseudonáhodné nebo nějaká jiná hodnota, která je známá oběma komunikujícím stranám. Dá se posílat jako otevřená informace. Vzhledem ke způsobu šifrování lze zprávu šifrovat pouze sériovým způsobem, naproti tomu dešifrovat zprávu lze paralelně.

CBC Šifrování:

Vstup: k-bitový klíč	K
n-bitový inicializační vektor	C_0
n-bitové bloky plaintextu	$M = M_1 M_2 \dots M_t$
Algoritmus:	$C_j = E_K (C_{j-1} \oplus M_j)$
	kde $\oplus$ je logický operátor XOR
Výstup: n-bitové bloky šifrovaného textu	$C = C_0 C_1 ... C_t$

CBC Dešifrování:

Vstup: k-bitový klíč	K
n-bitové bloky šifrovaného textu	$C = C_0 C_1 ... C_t$
Algoritmus:	$M_j = C_{j-1} \oplus D_K (C_j)$
Výstup: n-bitové bloky plaintextu	$M = M_1 M_2 \dots M_t$



Obr. 3.3: Schéma šifrování v CBC módu, převzato z [44].

Vlastnosti:

1) Pro dva identické bloky plaintextu, jsou při použití stejného klíče a různého IV vytvořeny různé bloky šifrovaného textu.
2) Protože blok šifry C_j je závislý na C_{j-1}, je nutné zachování pořadí bloků.
3) Pokud před dešifrováním vznikne chyba v bloku C_j, tak tato chyba ovlivní dešifrování bloků C_j a C_{j+1}. Blok C_{j+2} již bude dešifrován korektně.

Cipher Feedback Mode (CFB)

Tento mód bývá označován jako proudová šifra s vlastní synchronizací, pro její realizaci se používají běžné algoritmy blokových šifer. V vznikl z potřeby šifrování bloků, které jsou menší než velikost bloku, se kterou pracuje vybraný alogritmus. Pracuje podobně jako předchozí CFC – výstup po šifrování bloku je použit pro šifrování bloku následujícího. Inicializační vektor naplní posuvný registr a je za pomocí primárního klíče zašifrován vybraným algoritmem. Výstup této operace je subklíč, jehož nejlevějších n bitů (nejvíce významných) následně XORujeme s prvním n-bitovým blokem plaintextu. Posuvný registr se po tomto kroku posune o n bitů vlevo a na uvolněné místo je zapsán blok zašifrovaného textu získaný z předchozího XORování. Postupně je takto zašifrován celý vstupní text. Stejně jako u CFC i zde lze šifrovat pouze sériově, dešifrování pak může probíhat paralelně.

CFB Šifrování:

Vstup: k-bitový klíč	K
n-bitový inicializační vektor	I_1
r nejlevějších bitů vektoru I	L_r
n – r nejpravějších bitů vektoru I	R_{n-r}

r-bitové bloky plaintextu $M = M_1 M_2 \ldots M_t$

Algoritmus: $C_j = M_j \oplus L_r (E_k (I_j))$

$I_{j+1} = R_{n-r} (I_j) \,||\, C_j$

kde || je operátor konkatenace

Výstup: r-bitové bloky šifrovaného textu $C = C_0 C_1 \ldots C_t$

CFB Dešifrování:

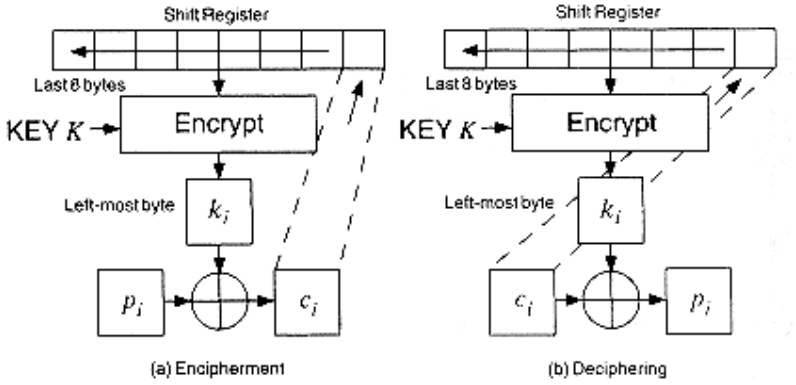
Vstup: k-bitový klíč K

n-bitový inicializační vektor I_1

r-bitové bloky šifrovaného textu $C = C_0 C_1 \ldots C_t$

Algoritmus: $M_j = C_j \oplus L_r (E_k (I_j))$

Výstup: r-bitové bloky plaintextu $M = M_1 M_2 \ldots M_t$



Obr. 3.4: Schéma šifrování v CFB módu, převzato z [44].

Vlastnosti:

1) Pro dva identické bloky plaintextu, jsou při použití stejného klíče a různého IV vytvořeny různé bloky šifrovaného textu.
2) Protože blok šifry C_j je závislý na předchozích blocích plaintextu M_1, ..., M_{j-1}, je nutné zachování pořadí bloků.
3) Chyba v bloku šifry C_j způsobí nekorektní dešifrování následujících n/r bloků. Tato chyba se v bloku M_j objeví na stejné pozici jako je v bloku C_j a v následujících blocích jako důsledek posunu r-bitového bloku v registru.

Output Feedback Mode (OFB)

V tomto módu je již bloková šifra použita k vytvoření synchronní proudové šifry. Stejně jako v předchozím případě je i zde posuvný registr, pro jehož naplnění se nepoužívají bloky plaintextu. Nejdříve je naplněn inicalizačním vektorem, který je poté zašifrován vybraným algoritmem.

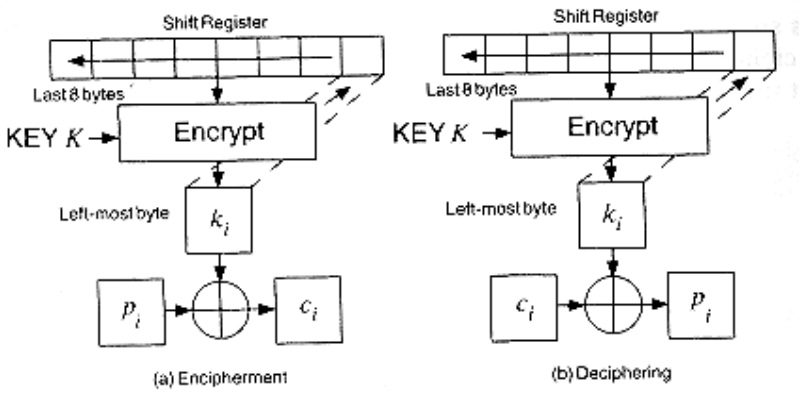
Výstupem je subklíč, jehož nejlevějších r bitů je nasunuto z pravé strany do registru a zároveň XORováno s r-bitovým blokem plaintextu. Tímto způsobem subklíče naplňují registr a zároveň tvoří klíč, se kterým je postupně celá zpráva XORována.

OFB Šifrování:

Vstup: k-bitový klíč	K
n-bitový inicializační vektor	I_0
r-bitové bloky plaintextu	$M = M_1 M_2 \dots M_t$
Algoritmus:	$I_j = E_K (I_{j-1})$
	$C_j = M_j \oplus L_r (I_j)$
Výstup: r-bitové bloky šifrovaného textu	$C = C_0 C_1 \dots C_t$

OFB Dešifrování:

Vstup: k-bitový klíč	K
n-bitový inicializační vektor	I_0
r-bitové bloky šifrovaného textu	$C = C_0 C_1 \dots C_t$
Algoritmus:	$M_j = C_j \oplus L_r (I_j)$
Výstup: r-bitové bloky plaintextu	$M = M_1 M_2 \dots M_t$



Obr 3.5: Schéma šifrování v OFB módu, převzato z [44].

Vlastnosti:

1) Stejně jako u CBC a CFB platí, že pro dva identické bloky plaintextu, jsou při použití stejného klíče a různého IV vytvořeny různé bloky šifrovaného textu.
2) Pro správné dešifrování je nutné zachovat pořadí bloků šifry.
3) Chyba v šifrovaném textu způsobí změnu ekvivalentního bitu plaintextu, ale neprojevuje se v dalších blocích.

4.1.1.1 Feistelova šifra

Jedním z průkopníků blokových šifer byl Horst Feistel, který je autorem Feistelovy šifry (známé také jako Feistelova síť), jejíž princip využívají také další algoritmy. Proto zde podrobněji vysvětlím jak funguje.

Nechť **F** je funkce, která provádí samotné šifrování, **n** je počet kol, tzn. kolikrát je každý blok šifrován, **i** značí konkrétní kolo, $\mathbf{K_0, K_1, ..., K_n}$ jsou subklíče použité v každém kole, $\mathbf{L_i}$ a $\mathbf{R_i}$ jsou levá a pravá polovina bloku v i-tém kole a znak $\oplus$ značí logickou operaci XOR.

Allgoritmus Feistelovy šifry v každé kroku provádí následující výpočty:

Pro **i = 0, 1, …, n**

$\mathbf{L_{i+1} = R_i}$

$\mathbf{R_{i+1} = L_i \oplus F(R_{i+1}, K_i)}$

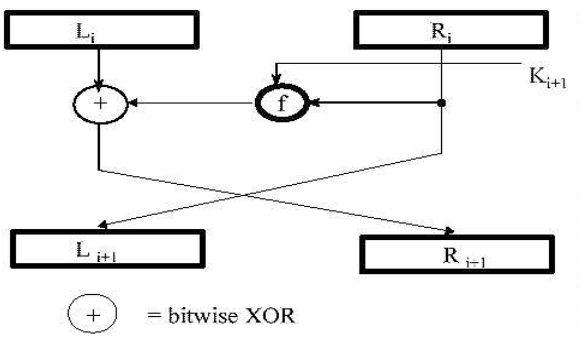
Při dešifrování se výpočet liší jen minimálně:

Pro **i = n, n-1, …, 0**

$\mathbf{R_i = L_{i+1}}$

$\mathbf{L_i = R_{i+1} \oplus F(L_{i+1}, K_i)}$

L_0R_0 nám pak značí otevřený text. Lépe je to pak vidět z obrázku 3.1:



Obr. 3.1: Schéma jednoho průchodu Feistelovy šifry, převzato ze [7].

Modifikací pak jsou nesouměrné Feistelovy šifry, které dělí blok na dvě různě velké části.

Z Feistelovy šifry použité Feistelem v algoritmu Lucifer se vyvinul DES (Data Encryption Standard), který byl v pozdějších letech nahrazen standartem AES (Advanced Encryption Standard). Pro standart AES byla vypsána soutěž, které se účastnily některé z níže zmíněných algoritmů, vítězem se stal algoritmus Rijandel.

4.1.1.2 3-Way

Šifra byla vytvořena v roce 1994 Joane Daemenem, který ji navhrl jako 11 kolovou substitučně-permutační síť. Blok má délku 96 bitů stejně jako klíč. V době svého vzniku poměrně bezpečná šifra, ale v dnes má již málokterá délku klíče menší než 128 bitů.

Během doby byl také objeven způsob jejího prolomení, jedná se o útok, který zkoumá vztahy mezi klíči (dále jako related-key útok) [106]. Tím pro nás tato šifra přestává být zajímavou.

4.1.1.3 Blowfish

Tento algoritmus byl navržen Brucem Schneierem v roce 1993. Byla navržena jako náhrada za DES, nicméně dnes tuto funkci přebírá AES. Délka jeho bloku je 64 bitů, ale délka klíče je proměnná a může dosahovat hodnot od 32 po 448 v 8 bitových krocích. Uvnitř pracuje 16 kolová Feistelova šifra doplněná o S-boxy závislými na klíči a neinvertibilní funkci F.

Byly publikovány útoky na Blowfish, který při šifrování provedl pouze 4 kola, ale ve stávající podobě je považován za bezpečný algoritmus. Pouze kvůli malé délce bloku je možné, že při souborech s více než 2^{32} datovými bloky by byl úspěšný útok pracující na principu narozeninového paradoxu. S tímto se ale v běžném použití setkáme málokdy [112].

4.1.1.4 CAST

Vznikl v roce 1996 a jeho autory jsou Carlisle Adams a Stafford Tavares. Je používán v některých verzích PGP a GPG a stavbou je podobný šifře Blowfish, také používá Feistelovu šifru, S-boxy závislé na klíči a neinvertibilní funkci F. První verzí byl CAST-128, který měl délku bloku 64 bitů a klíč mohl mít velikost od 40 do 128 bitů. Pokud byla délka klíče větší než 80 bitů, bylo prováděno 16 kol Feistelovy šifry, jinak jen 12. Jeho vylepšenou verzí je CAST-256, který má zvětšenou délku bloku na 128 bitů, klíče o velikosti od 128 po 256 bitů v 32 bitových krocích a 48 kol obecné Feistelovy šifry.

Útoky diferenciální kryptoanalýzou probíhaly na algoritmech s redukovaným počtěm kol, na prolomení standartního algoritmu CAST lze použí pouze útok hrubou silou, který ovšem může být úspěšný za použití velmi krátkého klíče. Za zmínku stojí také, že tento algoritmus byl jedním z kandidátů na AES, ale nedostal se do užšího výběru [106].

4.1.1.5 CMEA

Algoritmus vyvinutý v roce 1991 Asociací pro telekomunikační průmysl (Telecommunications Industry Association) pro zabezpečení komunikace mobilních telefonů v USA. Používá poměrně krátký klíč o délce 64 bitů a proměnnou délku bloku s hodnotami od 16 do 64 bitů. Tento algoritmus není bezpečný, útok s možností vybrání otevřeného textu a shlédnutí jeho zašifrované podoby (dále jen chosen-plaintext útok) je úspěšný, přičemž stačí mít jen 338 takových vzorků.

4.1.1.6 DES

Jeden z prvních algoritmů na principu blokových šifer, vyvinut firmou IBM v roce 1975 na zakázku americké vlády. Do vývoje zasáhla i americká Agentura pro národní bezpečnost (NSA), která přiměla IBM zkrátit klíč z původních 128 bitů na 56 a vynutila si i určité úpravy S-boxů, kvůli kterým byla dlouho podezírána z vytvoření zadních vrátek do tohoto algoritmu. Tyto představy byly přiživovány také utajováním designu S-boxů. Až po prolomení systému pomocí diferenciální kryptoanalýzy IBM uvolnila kriteria pro návrh S-boxů, což vedlo ke konci obviňování NSA.

V dnešních poměrech nám tato šifra nemůže zaručit bezpečnost, protože kvůli krátkému 56 bitovému klíči je prolomení útokem hrubou silou otázkou několika dnů, viz Copacobana [105]. Jeden blok má délku 64 bitů a data jsou kódována 16 průchody Feistelovy šifry.

Vylepšenou verzí byl Triple DES, také znám jako TDEA (Triple Data Encryption Algorithm), ve kterém probíhalo šifrování třikrát - **DES(K_3;DES(K_2;DES(K_1;M))).** Pokud jsou klíče K_1 a K_3 stejné, pak se jedná o variantu 2TDES. Oproti DES přinesl Triple DES podstatné zvýšení bezpečnosti, ale již se jedná o poměrně pomalý algoritmus, který nenabízí nic víc než některé mnohem rychlejší algoritmy. Z těchto důvodů se od něj upouští a bývá nahrazen jeho následníkem AES [12].

4.1.1.7 DEAL

Další z algoritmů, založených na DES. Oproti DESu přináší vylepšení v podobě 128 bitového bloku a tří možných velikostí klíče – 128, 192 a 256 bitů, podle toho se lze setkat s algoritmy pojmenovanými DEAL-128, DEAL-192 a DEAL-256. Pro první dvě verze je doporučení provést s každým blokem alespoň 6 kol šifrování, pro DEAL-256 je doporučených alespoň 8 kol.

Jeho nevýhodou je rychlost, která je u 6 kolového DEALu podobná jako u Triple DES. Na druhou stranu je zde možnost implementovat DEAL na stávajícím DES hardwaru či softwaru.

Útoky na tuto šifru jsou možné spíše v teoretické rovině, pro útok s použitím otevřeného textu na 6 kolový DEAL by bylo potřeba 2^{70} otevřených textů a 2^{121} DES-zašifrování[8].

4.1.1.8 FEAL

Šifra vyvinutá spoečností Nippon Telephone & Telegraph jak vylepšení DESu. Používá 64 bitový klíč a 64 bitový blok a je zde několik verzí lišící se počtem kol FEAL-4, FEAL-6 a FEAL-8.

Všechny tyto verze se ukázaly být velmi slabými šiframi, všechny lze poměrně snadno prolomit (pro FEAL-4 stačí dokonce 5 párů bloků otevřených textů, pro FEAL-6 je to 100 párů a pro FEAL-8 je to 2^{15}).

Dalšími verzemi byly FEAL-N kde N značí počet kol, takže pro N = 4 se jedná o stejnou šifru jako FEAL-4. Dalším vylepšením byla verze FEAL-NX, kde X znamenalo expansion, jedná se o šifru

FEAL-N se 128 bitovým klíčem. Pokud byly v pravé polovině klíče samé nuly, algoritmus by pracoval jako FEAL-N. To zaručovalo zpětnou kompatibilitu, která je charakteristická pro rodinu šifer FEAL [9].

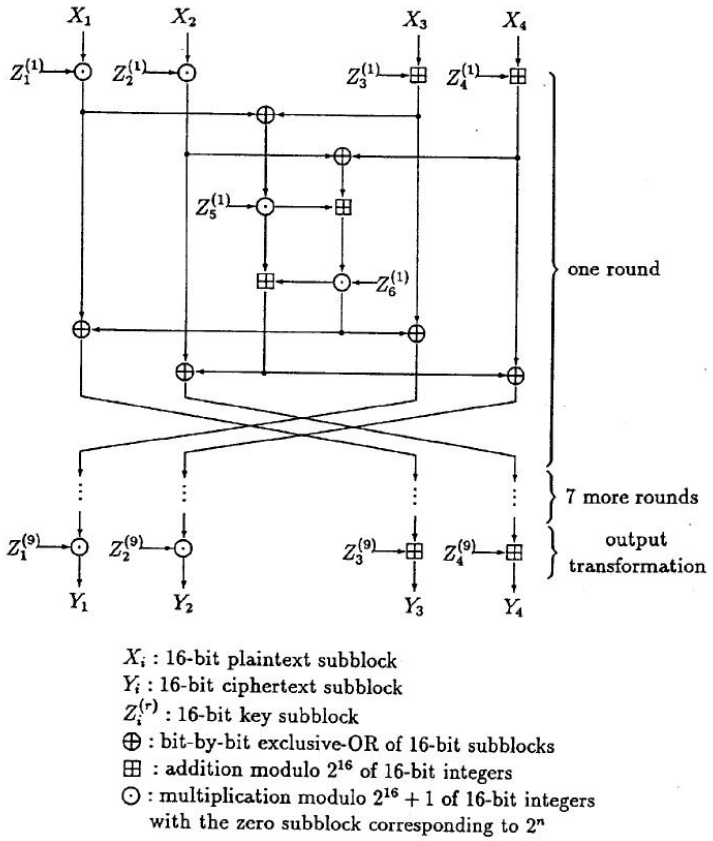
4.1.1.9 GOST

Je to ruská obdoba algoritmu DES, ovšem aby nic nebylo ponecháno náhodě, bylo použito 32 kol a 256 bitový klíč, bloky zůstaly 64 bitové. Při snížení počtu použitých kol na 24 existuje metoda, s jejíž pomocí lze tuto šifru prolomit. Je k tomu potřeba mít k dispozici otevřené texty a jejich zašifrované protějšky spolu s možností volby klíče, poté lze za pomocí diferenciální kryptoanalýzy prolomit redukovanou verzi této šifry [12].

4.1.1.10 IDEA

Pochází ze Švýcarska a poprvé byla popsána v roce 1991. Jejími autory jsou Xuejia Lai a James Massey. Původně byla navržena jako náhrada za dosluhující DES standart.

Tento systém pracuje se 64 bitovými bloky, 128 bitovým klíčem a 8 koly šifrování každého bloku. Pro šifrování kombinuje 3 různé algebraické skupiny a to: XOR, sčítání (modulo 2^{16}, přetečení se nebere v úvahu) a násobení (modulo $2^{16}+1$, opět se nebere v potaz přetečení), viz obr. 3.2:



Obr: 3.2: schéma šifrování algoritmu idea, převzato z [10].

Díky rozdělení každého bloku na 16 bitové subbloky se tato šifra dá dobře implementovat i na hardwarových 16 bitových zařízeních, ačkoli některé použité aritmetické operace nejsou zdaleka tak rychlé při softwarovém výpočtu. Výsledkem při softwarovém použití je rychlost podobná DESu.

IDEA je považována za velmi bezpečný alogritmus, navzdory tomu, že byla nalezena skupina slabých klíčů (její velikost je 2^{51}), která ovšem díky celkovému počtu možných klíčů (2^{128}) nemůže celou šifru znatelně ohrozit [11]. Také je známá její odolnost vůči diferenciální kryptoanalýze. Při úspěšných experimentech o prolomení této šifry bylo nutno zredukovat počet kol při šifrování bloků. Doteď nejsou známy použitelné útoky na běžnou verzi této šifry. Již ale existuje její nástupce s názvem IDEA NXT [107].

4.1.1.11 LOKI

Další z algoritmů, které měly nahradit DES. Pracuje s 64 bitovými bloky a 64 bitovým klíčem. První verze LOKI byla rychle prolomena (bylo ukázáno, že počet klíčů pro prohledání při útoku hrubou silou se dá redukovat na 256). Proto byla uvedena nová verze pod názvem LOKI91, která ovšem neodolá útoku, kdy útočníkovi stačí prozkoumat soubory, které šifroval s jím vybranými klíči. Poslední modifikace je LOKI97, která byla kandidát na AES, ale již byly navženy útoky diferenciální kryptoanalýzou i na tuto verzi.

4.1.1.12 Lucifer

Jedna z prvních moderních šifer, navrhl ji Horst Feistel v 60. letech a jako první implementuje Feistelův princip. Během let se objevilo několik verzí, ale dnes již žádná z nich není bezpečná.

4.1.1.13 MacGuffin

Autory této šifry jsou Matt Blaze a Bruce Scheier a původně vznikla jako experiment. Používá Feistelovu síť, ale s jednou modifikací – blok není dělen na stejné poloviny, ale jedna část má 16 bitů a druhá 48. Defacto se celý blok dělí na 4 16 bitové subbloky a pracuje se nejdříve s třemi pravými subbloky, na kterých je provedena série operací, a jejíž výsledek je vyXORován s levým subblokem. Na konci každého kola se provede rotace tak, že nejlevější subblok se stane nejpravějším. Během šifrování se s každým blokem provede 32 takovýchto kol. Podle autorů je možné používat skoro jakoukoli délku klíče a počet kol, ale jako standartní hodnoty doporučují 128 bitový klíč a výše zmíněných 32 kol [13].

Bylo ovšem zjištěno, že tato šifra neobstojí před diferenciální kryptoanalýzou. Podle [14] je MacGuffin s 32 koly šifrování pro každý blok méně odolný než standartní 16 kolový DES.

4.1.1.14 MARS

Tento algoritmus byl příspěvkem firmy IBM do soutěže o AES. Tato šifra těží ze stále rostoucího výpočetního výkonu počítačů, a tak používá snad všechny možné kryptografické techniky zakomponované do jediného algoritmu. Pracuje se sčítáním, odčítáním, násobením, S-boxy a pevnými a datově závislými rotacemi. Tyto metody aplikuje na bloky o velikosti 128 bitů a velikost klíče může být od 128 po 448 bitů [15].

V dnešní době není znám žádný úspěšný útok, který by vedl k prolomení celé šifry, v [16] jsou ovšem popsány útoky na strukturu MARSu, které mají za cíl zviditelnit potenciální slabiny algoritmu.

4.1.1.15 MISTY

Algoritmus vyvinutý firmou Mitsubishi Electric v roce 1995 a je navržen tak, aby odolal lienární i diferenciální kryptoanalýze. Současně také není znám případ prolomení této šifry, což může být dáno také tím, že zatím nebyla podrobena důkladnému průzkumu. Pracuje se 64 bitovými bloky, 128 bitovými klíči, s variabilním počtem kol v násobcích 4 (doporučená hodnota je 8).

4.1.1.16 MMB

Původně alternativa k algoritmu IDEA, ale používá 128 bitový blok. Ukázala se jako nedostatečně bezpečná a bylo na ni realizováno několik úspěšných útoků [12].

4.1.1.17 NewDES

Neúspěšný nástupce DESu, zjistilo se dokonce, že je ještě snáze prolomitelný, než původní DES. Pro jeho prolomení stačí 24 různých útočníkovi známých klíčů a k tomu 530 vybraných párů bloků otevřený text/šifra [106].

4.1.1.18 RC2, RC5, RC6

RC2 vznikla v roce 1989 se zaměřením primárně na 16 bitové procesory. Jejím autorem je Ron Rivest ze společnosti RSA Data Security, Inc. Šifra jako taková není patentována, ale je obchodním tajemstvím společnosi RSA Data Security, Inc., nicméně její kód byl zveřejněn na jednom z diskuzních fór. Byl publikován způsob útoků na tuto šifru pomocí vybraných párů bloků otevřený text/šifra, těchto párů ale musí být zhruba 2^{34} [106].

Další verzí byla RC5, která byla zajímavá proměnnou délkou bloku (dána platformou, na které algoritmus pracoval), dále měla proměnnou velikost klíče a počet kol. David Wagner, John Kelsey a Bruce Schneier objevili některé slabé klíče, ale pravděpodobnost jejich výběru je $2^{-10.r}$ kde **r** je počet kol. Při dostatečně vysokých r (tj. r > 10) nepředstavují tyto klíče velké ohrožení. V [18] je ovšen popsán útok diferenciální kryptoanalýzou.

Další evolucí a současně příspěvkem Rona Rivesta do soutěže o AES byl algoritmus RC6. Používá 128 bitové bloky, 20 kol šifrování bloku a klíče o délce 128, 192 a 256 bitů. V [19] je popsán postup pro získání klíče z šifry, která neprovádí více než 15 kol šifrování každého bloku a je rychlejší než útok hrubou silou.

4.1.1.19 REDOC

Tento algoritmus se vyskytuje ve dvou verzích – REDOC-II a REDOC-III. REDOC-II pracuje s bloky o délce 80 bitů a se 160 bitovým klíčem. Každý blok je pak šifrován v 10 kolech. Navzdory tomu, že se provedlo prolomit REDOC-II, který s každým blokem provedl jen jedno kolo šifrování, tak 10 kol šifrování každého bloku je zatím stále nepřekonaná ochrana a tento algoritmus je považován za bezpečný.

Jeho stinnou stránkou je ovšem rychlost, nebo spíše pomalost, se kterou pracuje. Proto byl vytvořen REDOC-III, který měl přinést bezpečnost REDOCu-II doplněnou o rychlost. To se bohužel nepodařilo a podle [20] k útoku pomocí kryptoanalýzy stačí 2^{20} dvojic bloků otevřený text/šifra a 2^{30} bytů paměti. Jen pro zajímavost ještě uvedu, že umožňuje použití klíče o délce až 20480 bitů [12].

4.1.1.20 Rijndael

Tento algoritmus vytvořený Joanem Daemenem a Vincentem Rijmenem se stal vítězem soutěže o standart AES. Jeho tvůrci byli inspirováni algoritmem Square, který je popsán níže. Mezi některé jeho důležité vlastnosti patří i to, že jej lze implementovat s různým důrazem na složitost nebo rychlost. Také se dá použít na Smart kartách, kde může zabírat malé množství paměti, zde je pak kompromis mezi velikostí a výkonem. Díky svému paralelnímu návrhu je dobře využitelný na víceprocesorových strojích, které se stále rozšiřují.

Délka bloku se může pohybovat mezi 128 a 256 bity stejně jako délka klíče. Počet kol není pevný, ale je dán funkcí, která je ovlivněna délkou bloku i délkou klíče. Tato funkce nabývá hodnot 10 (pokud je délka klíče i bloku 128 bitů), 12 (délka klíče nebo bloku 256 bitů) a 14 (délka klíče nebo bloku 256 bitů). Hodnota minimálně 10 kol byla zvolena podle nejlepšího dosaženého útoku, který zvládne prolomit maximálně 7 kolovou šifru Rijndael. Proto byl základní počet kol zvednut na 10, aby zde byla bezpečnostní rezerva [21].

O kvalitě této šifry hovoří i její status nového amerického standartu pro šifrování dat. Zajímavostí také je, že autoři zavrhli patentování algoritmu nebo jeho implementace.

4.1.1.21 Safer

Na žádost společnosti Cylink Corporation vytvořil Robert Massey algoritmus Safer (zkratka pro Secure And Fast Encryption Routine). Existuje ve více verzích s 40, 64 a 128 bitovým klíčem – podle

toho nesou názvy Safer K40, Safer K64 a Safer K128. Tyto verze ovšem mají slabinu v tvorbě subklíčů, opravené algoritmy mají místo pouhého K v názvu SK. Bloky této šifry mají 64 bitů.

Jsou známy úspěšné útoky na Safer s redukovaným počtem kol, ale jinak Safer bezpečný proti diferenciální i lineární kryptoanalýze[12].

4.1.1.22 Serpent

Taktéž účastník soutěže o AES. Autoři Ross Anderson, Eli Biham a Lars Knudsen kombinují principy DESu a paralelní zpracování bloku pro zvýšení rychlosti a efektivy (tato metoda se jmenuje bitslicing). Pracuje se 128 bitovými bloky, které šifruje ve 32 kolech, a 256 bitovými klíči. V první verzi Serpent 0 obsahoval S-boxy použité přímo v DESu, v další verzi Serpent 1 již používá S-boxy lehce změněné. Serpent 1 je odolný vůči diferenciální i lineární kryptoanalýze.

Podle [22] je dosud nejlepší provedený útok účinný na Serpent s maximálně 10 koly.

4.1.1.23 SQUARE

Tato šifra se 128 bitovým blokem i klíčem byla vytvořena s důrazem na odolnost vůči diferenciální a lineární kryptoanalýze. Bohužel se podařilo vytvořit útok přizpůsobený právě této šifře až do úrovně 6. kola šifrování každého bloku. Proto se doporučuje použití nejméně 8 kol, nicméně v [23] již používání tohoto algoritmu nedoporučují. Díky tomu, že tento algoritmus inspiroval i tvůrce Rijndaelu, je tento útok účinný i na Rijndael.

4.1.1.24 Skipjack

Algoritmus vyvinutý americkou Agenturou pro národní bezpečnost (NSA), který byl po několik let udržován v tajnosti. Jeho zvláštností je použití dvou druhů iterací šifrování – tzv. A-iterace a B-iterace. Na každém 64 bitovém bloku je provedeno nejdříve 8 A-iterací, pak 8 B-iterací následovaných 8 A-iteracemi a na závěr 8 B-iterací. Klíč má délku 80 bitů, blok má velikost 64 bitů.

Podle [24] pro útok na Skipjack, který využívá pouze 16 iterací, stačí 2^{17} vybraných dvojic bloků otevřený text/šifra, ale Skipjack v původní podobě s 32 koly zůstává neprolomen.

4.1.1.25 Twofish

Nástupce šifry Blowfish, také člen soutěže o AES. Jsou použity bloky o velikosti 128 bitů a klíč až do velikosti 256 bitů. Šifra Twofish je specifická svými S-boxy závislými na klíči, z klíče je totiž použita polovina bitů pro kódování a druhá polovina pro úpravu S-boxů [113].

Možnosti prolomení této šifry ještě nejsou plně prozkoumány, ale zatím se jeví jako velmi slibná.

4.1.2 Proudové symetrické šifry

Proudové šifry, jak už název napovídá, nerozdělují šifrovaný text na bloky, ale zpracovávají jej po jednotlivých znacích. Většinou tak, že jej podle dané funkce kombinují s klíčem. Již ze své podstaty se tak hodí nejlépe pro šifrování nějakého nepravidelného toku dat, jakými jsou např telefonní hovory. Tyto šifry mohou být zastoupeny také blokovými šiframi, které ovšem pracují v CFB módu – ten poskytuje vlastnosti proudového zpracování s vlastní synchronizací, takže z jakékoli bezpečné blokové šifry lze vytvořit i šifru proudovou.

4.1.2.1 ORYX

Šifra využívaná právě pro kódování telefonních hovorů v mobilních sítích. Tento systém se používá v severoamerické síti, ale již není považován za bezpečný. Ve [25] je popsán způsob útoku na tuto šifru, ke kterému stačí pouze 25-27 bytů plaintextu. Tímto způsobem se dá také získat plný 96 bitový vnitřní stav šifry.

4.1.2.2 RC4

Opět produkt společnosti RSA Data Security, Inc. Stejně jako blokové RC2, RC5 i RC6 je kód této šifry chráněn jako obchodní tajemství, ačkoli i tato šifra byla zveřejněna neznámým autorem v roce 1994, což dalo podnět k analýze této šifry. Tato šifra je použita i například v SSL, nicméně se nedoporučuje její nasazení v nových aplikacích. V [26] jsou popsány její slabiny spočívající ve špatném generování subklíčů.

4.1.2.3 SEAL

Vytvořen v roce 1992 Rogawayem Coppersmithem za účelem získání rychlé softwarové proudové šifry. I po 10 letech existence byla stále nejrychlejší proudovou šifrou – při kódování provádí pouze 5 instrukcí na byte. Díky tomu se hodí například pro šifrování obsahu disku nebo pro podobné aplikace, kde je potřeba přistupovat k datům uprostřed proudu. Ovšem ve [27] je popsán způsob útoku na poslední verzi 3.0.

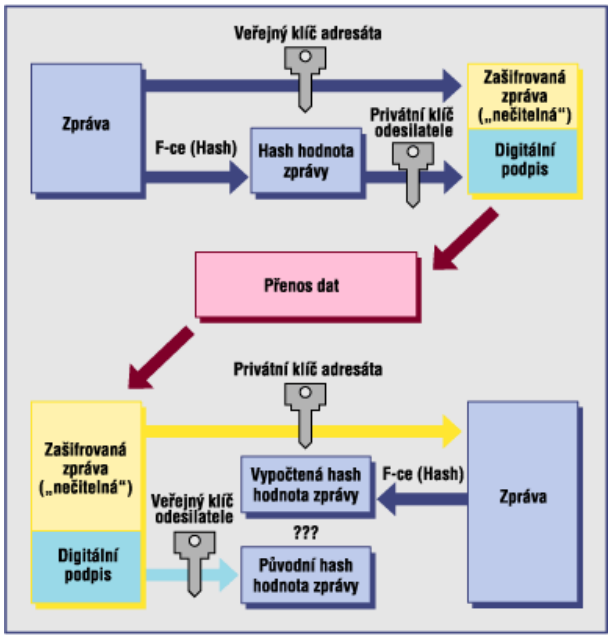
4.1.3 Shrnutí vybraných algoritmů symetrických šifer

V této části provádím sumarizaci vlastností jednotlivých algoritmů spolu s hodnocením vhodnosti jejich dalšího využití. Pro tento účel se mi jevilo jako nejvhodnější použití tabulky, kde budou veškerá data o vybraných algoritmech přehledně uspořádána. Tato tabulka s hodnocením bezpečnostních šifer je kompletně uvedena v příloze A.

4.2 Asymetrické šifry

Jak jsem již popsal v úvodu kapitoly, tyto šifry pracují se dvěma druhy klíčů. Každý účastník komunikace má jeden klíč soukromý, který je tajný a majitel ho musí chránit před kompromitací, a druhý veřejný, který může někde vystavit. Základem použití asymetrických šifer je fakt, že použití privántího klíče autentizuje jeho majitele. Zachování důvěrnosti zprávy je pak zajištěno použitím veřejného klíče, čímž se zpráva stává pro kohokoli jiného než přijemce nečitelná.

Na tomto principu je postavené použití digitálního podpisu, které je znázorněno na Obr. 5.3.



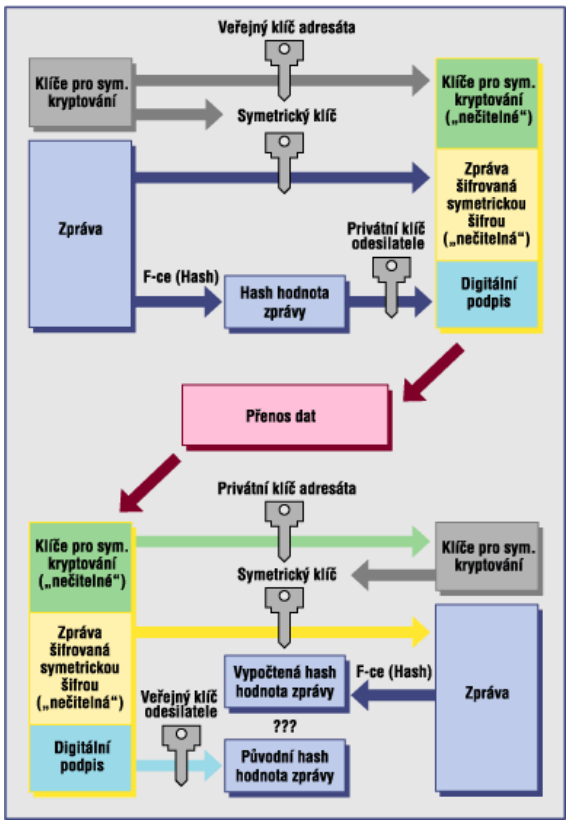
Obr 5.3: Schéma elektronického podpisu zpráv, převzato z [88].

Zpráva je nejdříve zpracována hašovací funkcí (ty budou probrány v další kapitole) a její haš hodnota je podepsána privátním klíčem odesílatele, čímž je zajištěna autenticita zprávy. Hašovací funkce je zde proto, aby nikdo nemohl zasílanou zprávu nahradit podvrhem, protože je nepravděpodobné, že by daná zpráva mohla mít stejnou haš. Bohužel to není naprosto nemožné a u některých hasovacích funkcí lze nalézt kolidující zprávy s různým obsahem ale stejnou haší.

Haš zprávy je pak přibalena k samotné zprávě, která je zašifrovaná veřejným klíčem příjemce. Ten nejdřív pomocí svého privátního klíče dešifruje zprávu, pak vypočte její haš a tu porovná s přijatou hodnotou dešifrovanou pomocí veřejného klíče odesílatele.

Nevýhodou asymetrických šifer je jejich nízká rychlost, takže není efektivní jejich použití na velké objemy dat. Ale právě v tomto místě se asymetrické šifry výborně doplňují se symetrickými, u kterých je problém bezpečné předání klíče druhé straně. Při zašifrování klíče a hodnot pro nastavení

symetrické šifry veřejným klíčem druhé strany mám jistotu, že nikdo jiný než příjemce tato data nezjistí. Veškerá další komunikace pak probíhá dohodnutou symetrickou šifrou, která je výrazně rychlejší. Tento proces je znázorněn na Obr. 5.4.



Obr. 5.4: Použití kombinace symetrické a asymetrické kryptografie, převzato z [88].

Bezpečnost asymetrických šifer může být založena na několika problémech, nejčastěji to pak bývá faktorizace velkých čísel a problém výpočtu diskrétního logaritmu. Z důvodu jiného principu šifrování mívají asymetrické šifry mnohem delší klíče než šifry symetrické, aktuálně je doporučována jako nejmenší bezpečná délka (s určitou rezervou do budoucna) 1024 bitový klíč.

4.2.1 RSA

Tento algoritmus je skoro synonymem pro elektronický podpis, nicméně je vhodný i pro šifrování souborů. Jeho tvůrci jsou Ron Rivest, Adi Shamir a Leonard Adleman, kteří s tímto modelem přišli již v roce 1977. Na základě informací z [28] tento algoritmus popíši.

Celý systém je postaven na tom, že je pro stávající počítače extrémně těžké rozložit dané číslo na součin prvočísel. Tomuto procesu se říká faktorizace. Výpočet samotných klíčů pak vypadá následovně:

- uživatel, např. Bob, potřebuje náhodně 2 velká prvočísla p_B a q_B

- jejich součin je pak hodnota $N_b = p_B \cdot q_B$
- dále pak vypočte hodnotu $\lambda(N_b) = NSN(p_B-1, q_B-1)$ (NSN = nejmenší společný násobek)
- dalším krokem je výpočet exponentu e_B, pro který musí platit $NSD(e_B, \lambda) = 1$
- v závěru vypočte Bob Euclidovským algoritmem inverzní hodnotu $e_B \bmod \lambda(N_b)$ – tímto číslem je číslo d_B takové, T_{dB} je inverzní k T_{eB} pro které platí:

$$T_{eB}: x \rightarrow x^{eB} \pmod{N_B}.$$

Bob může zveřejnit čísla N_B a e_B a on sám si uschová rozklad čísla N_B a číslo d_B. Každá zpráva, kterou pak Bob odešle nebo příjme, musí být přetransformována na číslo $x < N_B$, např je rozdělena na bloky délky k kde $2^k < N_B$ a každý blok je potom číslo velikosti od 0 do $2^k - 1$. Pokud Bob odesílá, pak je výsledkem šifra $s = T_{dB}(x)$. Kdokoli pak může na takovouto šifru provést inverzní operaci $T_{eB}(s)$, která mu dá výsledek x, tedy původní Bobovu zprávu.

Bezpečnost tohoto způsobu šifrování zpočívá v tom, že nedovedeme v rozumném čase spočítat faktor velkých čísel. Pro klíč dlouhý 256 bitů by to nebyl až takový problém, ale běžně používané klíče délky 1024 – 2048 již nejsou prakticky prolomitelné.

4.2.2 Diffie-Hellman

V roce 1976 vzikl protokol, který znamenal revoluci v moderní kryptografii. Jeho autoři, Whitfield Diffie a Martin Hellman, vytvořili základní koncepci systému s veřejným klíčem a digitálního podpisu. Jedná se o protokol, který umožní dvěma komunikujícím stranám vytvořit a předat si klíč bezpečnou cestou v nezabezpečeném prostředí. Podle [29] funguje šifra následovně.

Mějmě účastníky komunikace Alice a Boba. Spolu se dohodnou na základu **g**, se kterým budou dále pracovat. Každý z nich si zvolí své tajné číslo **SA** a **SB**. Následně si každý vypočtou hodnoty $\mathbf{a_A = g^{SA}}$ a $\mathbf{a_B = g^{SB}}$, které si následně vymění. Ve finále si Alice spočítá hodnotu $\mathbf{a_{AB} = a_b^{SA} = g^{SASB}}$ a Bob si vypočte hodnotu $\mathbf{a_{BA} = a_a^{SB} = g^{SASB}}$. Jak je vidět hodnoty $\mathbf{a_{AB}}$ a $\mathbf{a_{BA}}$ jsou stejné a tvoří klíč pro veškerou další komunikaci.

Bohužel tento algoritmus nezajišťuje autentizaci, proto by mohl být napaden útokem man-in-the-middle. Možným vyřešením tohoto problému by bylo podepsání subklíčů $\mathbf{g^{SA}}$ a $\mathbf{g^{SB}}$. Pro bezpečnost algoritmu je také důležitá správná volba základu **g**, ten by měl být prvočíslo nebo mít velkého prvočíselného dělitele, aby byla dostatečně ztížena možnost výpočtu klíče nepovolanými osobami [3].

4.2.3 Digital Signature Algorithm

Tento algoritmus je jedním z prvků Digital Signature Stadard. Patent na něj byl podán v roce 1991 a jako autorem je uveden David W. Kravitz. V témže roce byl navržen americkým Národním institutem

pro standarty a technologie na standart pro digitální podpis, přijat byl v roce 1994 pod označením FIPS 186 [81]). V roce 2000 prošel revizí, která rozšiřuje působnost jeho použití. Nyní je aktuální návrh na třetí revizi z března roku 2006, který mimo DSA obsahuje dále algoritmy RSA a za určitých podmínek i ECDSA – DSA s využitím eliptických křivek. Následující popis algoritmu DSA je založen na informacích uvedných v posledním návrhu DSS FIPS 186-3 [82]).

Pro vytvoření digitálního podpisu pomocí DSA je zapotřebí mít množinu doménových parametrů (hodnoty pro `p, q, g`, volitelně také `domain_parameter_seed` a `counter`, jejichž význam je vysvětlen níže), privátní klíč `x`, klíč zprávy `k` (náhodné číslo unikátní pro každou zprávu), hašovací funkci a data, která mají být podepsána. Tato podepsaná data jsou pak kontrolována pomocí stejných doménových parametrů, veřejným klíčem `y` odpovídajícím privátnímu klíči `x` a za použití stejného hašovacího algoritmu. Dále jsou definovány hodnoty L a N, které odpovídají délce modulu p a dělitele q v bitech. Tyto dvojice jsou následovné:

- L = 1024, N = 160
- L = 2048, N = 224
- L = 2048, N = 256
- L = 3072, N = 256

Teď si můžeme definovat jednotlivé prvky potřebné pro vytvoření digitálního podpisu:

- **p** – prvočíselný modul, pro jeho délku musí platit $2^{L-1} < p < 2^{L}$.
- **q** – prvočíselný dělitel hodnoty (p – 1), jeho délka v bitech musí splňovat $2^{N-1} < q < 2^{N}$.
- **g** – generátor podgrupy řádu q mod p, pro který platí $1 < g < q$.
- **x** – privátní klíč, náhodné nebo pseudonáhodné číslo takové, že $0 < x < q$, x musí zůstat tajné.
- **y** – veřejný klíč, vytvořený výpočtem $y = g^{x} \bmod p$.
- **k** – tajná hodnota unikátní pro každou zprávu, tzv. klíč zprávy (nebo také NONCE z ang. Number-used-ONCE), vytvořena náhodně nebo pseudonáhodně.

Prvky `p, q, g, y` tvoří veřejný klíč, `x` je pak klíč privátní.

Průběh podpisu zprávy M je následovný:

1. Je vygenerována hodnota k.
2. Výpočet hodnoty $r = (g^{k} \bmod p) \bmod q$.
3. z = nejlevějších N bitů haše zprávy M.
4. Výpočet hodnoty $s = (k^{-1} (z + xr)) \bmod q$.
5. Provedení kontroly, zda není r = 0 nebo s = 0, v takovém případě je doporučeno vygenerování nové hodnoty k a opakování výpočtu.

6. Hodnoty (s, r) tvoří digitální podpis zprávy M a jsou k ní přikládány.

Mějme přijaté hodnoty M‘, r‘,s‘ a y je veřejný klíč. Verifikace zprávy pak probíhá takto:

1. Kontrola zda platí 0 < r’ < q a 0 < s’ < q, pokud tento vztah neplatí, podpis je odmítnut.
2. $w = (s')^{-1} \bmod q$.
3. z = nejlevějších N bitů haše zprávy M’.
4. u1 = (zw) mod q.
5. u2 = (r’w) mod q.
6. $v = ((g^{u1}\, y^{u2}) \bmod p) \bmod q$.
7. Pokud v = r’, je podpis přijat jako platný.

V [83] probírají Dr. Klíma a Dr. Rosa možnosti napadení této šifry. Uvažovaný útok není postaven na mezeře v algoritmu, ale na možných problémech v jeho implementaci. Speciálně se zaměřuje na metodu generování klíče zprávy k, který musí zůstat utajený. Jak lze vidět z rovnice ve 4. kroku podpisu zprávy, znalost tohoto klíče útočníkovi umožňuje získat privátní klíč jednoduchým výpočtem.

Další bezpečnostní riziko popsané v [83] představuje i jakákoli chyba v generátoru pseudonáhodných čísel, který generuje k. Pokud by útočník mohl generátor ovlivnit, aby hodnota k nebyla použita jen jednou, ale vícekrát, pak by z dat získaných z odchycených zpráv mohl velice jednoduše sestavit soustavu dvou rovnic o jedné neznámé. Dvě různé zprávy používají při podpisu následující rovnice:

$$(z_1 + xr_1)\, s_1^{-1} = k_1 \pmod q \qquad (4.1)$$

$$(z_2 + xr_2)\, s_2^{-1} = k_2 \pmod q \qquad (4.2)$$

Při použití stejné hodnoty $k = k_1 = k_2$ platí:

$$x = (z_2 s_1 - z_1 s_2)(r_1 s_2 - r_2 s_1)^{-1} \bmod q \qquad (4.3)$$

jejíž řešení je již triviální.

Jako třetí riziko je zde zmíněna metoda, která staví na znalosti několika bitů klíče zprávy k. Tento způsob získání privátního klíče je již složitější a podrobně je popsán v [84].

Na základě zjištěných informací mohu určit, že algoritmus DSA je z matematického hlediska bezpečný, avšak vytváří další proměnnou, kterou je třeba zabezpečit stejně dobře, jako privátní klíč. Největší nebezpečí tohoto algoritmu je založeno na možnosti ovlinit generování klíče zprávy k nebo jeho úplné případně částečné prozrazení. Tato rizika je třeba mít na paměti při implementaci tohoto algoritmu a při jejich správném ošetření je jakékoli prolomení na hranici praktické proveditelnosti.

4.3 Kryptografické hašovací funkce

Hašovací funkce jsou jednosměrné funkce, jejichž použití umožní z jakkoli dlouhého řetězce dat *x* vytvořit řetězec přesně stanovené délky *y = f(x)*. Jejich jednosměrnost spočívá ve faktu, že pro dané *y* je výpočetně příliš náročné nalezení vstupní hodnoty *x*. Příkladem takovéto jednocestné funkce je vynásobení dvou velkých prvočísel – jejich opětovné rozdělení je natolik náročné, že s dnešním vybavením to není možné. Na tomto problému faktorizace je postavena šifra RSA. Hašovací funkce se tedy využívají např. pro kontrolu integrity dat, kdy libovolně dlouhé zprávy nebo soubory vytvoří "otisk" zprávy pevné délky. Tato délka se liší podle použitých šifrovacích algoritmů, v dnešní době to je nejčastěji 128 nebo 160 bitů, ačkoli jsou i algoritmy s haší větší, např 256 či 320 bitů.

Další podstatnou vlastností hašovacích funkcí je jejich odolnost vůči kolizím. Tímto pojmem je myšlen fakt, že je nepřiměřeně výpočetně náročné nalezení dvou zpráv *x* a *y*, pro které by platilo *f(x) = f(y)*. Podle [90] je dodržení této podmínky nazýváno bezkolizností prvního řádu. Kromě kolizí prvního řádu je v praxi důležitější dodržení podmínky o problému nalezení kolize druhého řádu. V této situaci neznáme původní zprávu *x*, pouze výsledek hašovací funkce *h(x)* a naším cílem je nalézt *y*, pro které platí opět kolizní vztah *f(x) = f(y)*. Dodržení těchto dvou vlastnostní umožňuje schématu elektronického podpisu používat pro podepsání pouze haš zprávy, která je mnohdy podstatně kratší než samotná zpráva, což vede k nezanedbatelné úspoře času při šifrování a dešifrování.

Je ovšem nutné si uvědomit, že bezkoliznost neznamená, že neexistuje další zpráva, která by měla stejnou haš, ale že je pro nás výpočetně příliš náročné ji nalézt. V. Klíma v [90] poukazuje na fakt, že při délce haše D může existovat pouze $2^{D+1} - 1$ zpráv, které mají různou haš. Ale množství prohledávaných zpráv je nekonečně mnoho, proto kolize musí existovat, ovšem jejich cílené nalezení podle definice nesmí být ve stávajících podmínkách efektivně proveditelné.

Bezpečnost hašovacích funkcí je podobně jako u symetrických šifer ovlivněna počtem možných stavů, jakých může haš nabývat. Mějme např. často využívanou funkci SHA-1, která má délku haše 160 bitů. V takovém případě existuje 2^{160} možných haší. O haš funkci, pro kterou můžeme nalézt kolizi se stejnou složitostí jako počet možných haší, tvrdíme, že je bezpečná. Zde ovšem vstupuje do hry narozeninový paradox popsaný v kapitole věnované útokům na kryptografické algoritmy. Ten nám říká, že s 50% pravděpodobností nalezneme kolizi už při $2^{D/2}$ pokusech, což by pro SHA-1 znamenalo 2^{80} pokusů. Tímto paradoxem je složítost hašovacích funkcí obecně snížena z 2^D na $2^{D/2}$. Pokud se ale podaří najít metodu, která vede k nalezení kolizí v podstatně menším počtu kroků, funkce je považována za prolomenou.

Nalezením metody pro efektivní hledání kolizí by měl daný algoritmus ztratit status korektního hašovacího algoritmu (neboť již nesplňuje dané podmínky) a jeho používání by mělo být přerušeno. Ne ve všech oblastech použití je však nalezení kolizí takovým problémem. Funkcím, pro které byl nalezen efektivní algoritmus nalezení kolizí, to přímo nebrání ve využití v různých generátorech pseudonáhodných čísel (momentálně nejsou známy útoky na generátory pseudonáhodných čísel využívající kolizí v hašovacích funkcí, ale pokud je to možné, je samozřejmě vhodné takovéto funkce v generátorech vyměnit za bezpečnější verze). Avšak jejich použití např. pro elektronický podpis je krajně nevhodné, neboť je tím dokázáno, že lze vytvořit dvě různé zprávy se stejným, platným podpisem.

Základem dnešních hašovacích funkcí je Damgard–Merklův princip iterativní hašovací funkce s využitím kompresní funkce. Princip této funkce (podle [90]) je následující. Kompresní funkce zpracuje blok zprávy m_i a výsledkem této operace je tzv. kontext (H_i). Tento kontext následně tvoří jeden ze vstupů pro kompresi následujícího bloku zprávy. Tato iterativní konstrukce je popsána vztahy:

$$H_0 = IV \tag{4.4}$$

$$H_i = f(H_{i-1}, m_i) \tag{4.5}$$

Protože délka kontextu H_i zůstává pořád stejná, ale na vstupu má funkce dva parametry o celkové délce větší než H_i, dochází ke ztrátě informace. To je důvod, proč se jedná o kompresní funkci.

Dále jsou v [90] uvedeny tyto vztahy:

1) Bezkoliznost hašovací funkce implikuje bezkoliznost kompresní funkce.
2) Bezkoliznost kompresní funkce nemusí nutně znamenat bezkoliznost celé hašovací funkce.
3) Ovšem pro Damgard–Merklovu konstrukci bylo dokázáno, že bezkoliznost kompresní funkce implikuje bezkoliznost hašovací funkce. Na tomto principu pracují moderní hašovací funkce.

V dalších částech kapitoly se již zaměřím na některé konkrétní funkce.

4.3.1 MD5

Tento algoritmus zde uvádím spíše kvůli jeho tradici a hojonosti nasazení ve své době, v aktuálně vytvářených systémech již není používán. Je to jeden z mnoha algoritmů vytvořených Ronem Rivestem a je poslední ze série algoritmů Message–Digest (odtud zkratka MD). Vznikl v roce 1992 jako náhrada za předchozí MD4, k jehož otázce bezpečnosti byly vzneseny jisté výhrady [91].

Délka jeho haše je, v porovnání s novějšími funkcemi pouze, 128 bitů, prakticky má tedy složitost 2^{64} (podle narozeninového paradoxu). V [91] je popsáno, jak MD5 upravuje délku zprávy tak, aby byla násobkem 512. Doplnění probíhá vždy (ikdyž zrovna platí, že délka L zprávy M L(M) mod 512 = 0) a zprávu lze doplnit o 1–512 bitů. Doplnění má tento formát – první bit je 1 a zbylé jsou 0. Použití doplňovacích bitů se řídí pravidlem ,že zpráva M', která je už doplněná má délku L(M') mod 512 = 448. Posledních 64 bitů je rezervováno pro hodnotu délky zprávy. Po tomto kroku máme jistotu, že zpráva je bezezbytku dělitelná na bloky o 16 slovech (slovo je zde 32 bitové).

Následně je inicializován zásobník pro 4 slova A, B, C, D (přesné inicializační hodnoty jsou v [91]) a celý systém je připraven pro provední čtyř kol, kdy v každém z nich provede 16 výpočtů vybrané funkce. V každém kole je v hlavní rovnici výpočtu použita jedna funkce z níže uvedených (v každém kole je 16x použita jedna rovnice):

- $f(B,C,D) = (B \wedge C) \vee (\neg B \wedge D)$ (4.6)
- $f(B,C,D) = (B \wedge D) \vee (C \wedge \neg D)$ (4.7)
- $f(B,C,D) = B \oplus C \oplus D$ (4.8)
- $f(B,C,D) = C \oplus (B \wedge \neg D)$ (4.9)

Matematický význam symbolů je: ∧ - AND, ∨ - OR, ¬ - NOT, ⊕ - XOR.

Rovnice samotného výpočtu vypadá takto:

$$A = B + (A + f(B,C,D) + X_k + T_i)^{\ll s} \quad (4.10)$$

kde $^{\ll s}$ znamená cyklický posun o *s* pozic. Takto je provedeno nad každým blokem celkem 64 výpočtů plus několik přídavných mezivýpočtů a výsledek haše je uložen ve slovech A, B, C, D, E.

V průběhu let se však MD5 stal terčem mnoha pokusů prolomení. Již v roce 1996 Hans Dobbertin v [92] ukázal, že lze nalézt kolize pro kompresní funkci algoritmu MD5. To je již první náznak potenciálně možných budoucích nálezů kolizí celé funkce. V březnu 2004 se rozjel projekt MD5CRK, který usiloval o nalezení kolizí hrubou silou pomocí distribuované výpočetní sítě, aby ukázal v praxi dosažené teoretické výsledky. Byl ovšem předčasně ukončen zprávou čínského týmu profesorky Wangové, který oznámil nalezení kolizí v [93] v srpnu 2004 na kryptografické konferenci Crypto 2004. Tady ukázali nalezení kolizí pro funkce MD4, MD5, HAVAL-128 a RIPEMD, jejichž nalezení dosáhli do 1 – 2 hodin na clusteru IBM p690. Dalším významným krokem v této oblasti je výzkum V. Klímy, který v roce 2006 vyvrcholil vytvořením programu, který je schopen nalézt kolize MD5 do jedné minuty na běžném PC pomocí tzv. metody tunelování. Na stránce tohoto projektu [94]) je celá metoda popsána a k dispozici jsou i zdrojové kódy programu.

Ve světle těchto událostí se z MD5 stává již historická záležitost, která by už dnes neměla najít uplatnění, hlavně díky existenci kvalitnějších algoritmů.

4.3.2 RIPEMD–160

Jedná se o vylepšenou verzí algoritmu RIPEMD, který se příliš nerozšířil. Jeho autory jsou Hans Dobbertin, Antoon Bosselaers a Bart Preneel a byl vytvořen v roce 1996. Délka haše je, jak už z názvu vyplývá, 160 bitů.

Kromě verze s délkou haše 160 bitů patří do této generace i RIPEMD-128, který byl uveden jako nahrazení prolomeného RIPEMD pro případy, kde je za potřebí délka haše 128 bitů. Rozšířením těchto dvou algoritmů pak dále vznikly RIPEMD-256 a RIPEMD-320, které podle [95] mají za úkol pouze snížit možnost náhodné kolize, jejich bezpečnost zůstává stejná jako bezpečnost jejich vzorů s poloviční délkou haše.

Oproti MD5 přidává RIPEMD-160 jedno 32 bitové slovo navíc (takže v zásobníku slov se nachází A, B, C, D, E), aby bylo ve finále dosaženo 160 bitové délky haše. V každém kroku jsou pak prováděny výpočty (podle [96]):

$$A = (A + f(B, C, D) + X + K)^{\ll s} + E \tag{4.11}$$

$$C = C^{\ll 10} \tag{4.12}$$

Funkce použité v jednotlivých kolech jsou následující:

- $f(B, C, D) = B \oplus C \oplus D$ (4.13)
- $f(B, C, D) = (B \wedge C) \vee (\neg B \wedge D)$ (4.14)
- $f(B, C, D) = (B \vee \neg C) \oplus D$ (4.15)
- $f(B, C, D) = (B \wedge D) \vee (C \wedge \neg D)$ (4.16)
- $f(B, C, D) = B \oplus (C \vee \neg D)$ (4.17)

Následně proběhne pět kol šifrování.

V současné době nevím o žádném úspěšném útoku na komplentí neredukovaný RIPEMD-160.

4.3.3 SHA

Jedná se o celou rodinu hašovacích funkcí, první SHA-0 byla vydána americkým Úřadem pro standarty a technolgie (NIST) pod označením FIPS PUB 180 [97]) v roce 1993. Krátce nato byl však nahrazen verzí SHA-1 (označení FIPS PUB 180-1), kterou vytvořil americký Úřad pro národní bezpečnost (NSA). Jediným rozdílem oproti původní SHA-0 je změna v bitové rotaci v kompresní funkci, ale NSA odmítla chybu blíže specifikovat. SHA-0 i SHA-1 mají délku haše 160 bitů.

Informace pro popis algoritmu jsem čerpal z [98]. Metoda doplňování délky zprávy je v tomto případě stejná jako u algoritmu MD5, tj. doplnění délky na násobek 512, kde posledních 64 bitů je vyhrazeno pro hodnotu délky zprávy. Zcela jiná je však metoda výpočtu. Pro výpočet jsou použity dva zásobníky, každý o pěti slovech o délce 32 bitů. Slova v prvním zásobníku jsou označena jako A,

B, C, D, E, slova v druhém jsou pak H_0, H_1, H_2, H_3, H_4. Dále výpočet pracuje se sekvencí 80 slov označených W_0, W_1, ... W_{79} a pomocnou proměnnou TEMP typu word. Posledním použitým prvkem je konstanta *K*, která má dané hodnoty pro každých 20 kroků výpočtu. Samotná zpráva je pak rozdělena na bloky o délce 16 slov M_1, M_2, .., M_n. Zpracování každého bloku zprávy probíhá v těchto krocích:

- M_i je rozděleno na jednotlivá slova W_0, .., W_{15}.
- Pro *t = 16, .., 79* se počítá:

$$W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16})^{\ll 1} \tag{4.18}$$

- Nechť A = H_0, B = H_1, C = H_2, D = H_3, E = H_4.
- Pro *t = 0, .., 79* se počítá:

$$TEMP = A^{\ll 5} + f(B, C, D) + E + W_t + K_t, \tag{4.19}$$

$$E = D; D = C; C = B^{\ll 30}; B = A; A = TEMP. \tag{4.20}$$

- Nechť $H_0 = H_0 + A$, $H_1 = H_1 + B$, $H_2 = H_2 + C$, $H_3 = H_3 + D$, $H_4 = H_4$.

Výsledná haš zprávy je pak tvořena konkatenací slov H_0, H_1, H_2, H_3, H_4.

Pro úplnost informací už jen potřebujeme znát použité funkce *f(B, C, D)* a konstanty *K*. Jednotlivé funkce f jsou stejně jako konstanty K postupně přiřazeny krokům *t = 0, .., 19; t = 20, .., 39; t = 40, ..,59; t = 60, ..,79*.

Použité rovnice jsou:

- $f(B, C, D) = (B \wedge C) \vee (\neg B \wedge D)$, (4.21)
- $f(B, C, D) = B \oplus C \oplus D$, (4.22)
- $f(B, C, D) = (B \wedge C) \vee (B \wedge D) \vee (B \wedge D)$, (4.23)
- $f(B, C, D) = B \oplus C \oplus D$. (4.24)

A hodnoty konstanty K jsou v hexadecimální soustavě tyto:

- *K = 5A827999,*
- *K = 6ED9EBA1,*
- *K = 8F1BBCDC,*
- *K = CA62C1D6.*

Aby byly informace o způsobu tvorby haše metodou SHA-1 kompletní, je třeba také říci, na jaké hodnoty jsou incializována slova H_0, H_1, H_2, H_3, H_4 (opět hexadecimálně):

- H0 = 67452301,
- H1 = EFCDAB89,
- H2 = 98BADCFE,
- H3 = 10325476,

- H4 = C3D2E1F0.

Verze SHA-0 již byla úspěšně několikrát prolomena, naposledy opět týmu profesorky Wangové, který kolize v SHA-0 nalezl spolu s kolizemi pro MD5, v roce 2005 však přišel s inovovanou metodou, která snížila složitost nalezení kolize na 2^{39} operací [99]).

Ani verze SHA-1 nezůstala neprolomena, mimořádně aktivní tým profesorky Wangové na konferenci Crypto 2005 ukázal zjednodušení teoretické složitosti útoku z 2^{80} na 2^{69} [100]). Aktuálně pracuje na dalším zjednodušení až na 2^{63} pomocí metody diferenční cesty (z ang. differential path, [101]).

Naštěstí již existuje několik nových verzí algoritmu SHA, souhrnně označovaných jako SHA-2. V této skupině algoritmů najedeme SHA-224, SHA-256, SHA-384 a SHA-512, kde čísla v názvu vyjadřují délku haše jednotlivých funkcí. Tyto funkce jsou vedeny pod standartizačním názvem FIPS PUB 180-2 [102]). Od svých předchůdců se liší hlavně zpracováváním více slov v jednom kole a větším počtem konstant, ale samotný princip funkce je skoro stejný, proto bude zajímavé sledovat, zda-li zneužití slabin starších verzí SHA-0 a SHA-1 budou aplikovatelná i na tyto nové funkce, zatím ovšem nejsou známy žádné částečné ani kompletní útoky. Díky délce haše zatím poskytují jistou rezervu do budoucna, ale pro jistotu již NIST vyhlásil soutěž na příští standart SHA-3. Uzavírka přihlášek je 31. října 2008 [103]), takže budeme moci sledovat, zda-li dojde k nějakým zásadním změnám v návrhu hašovacích funkcí. Prozatím nezbývá než doporučit nahrazení starých algoritmů jejich novějšími nebo bezpečnějšími nástupci či konkurenty (např. SHA-1 za SHA-256, SHA-512 nebo RIPEMD-160).

5 Typy útoků na kryptografické algoritmy

5.1 Ciphertext-only attack

Při tomto typu útoku zná útočník pouze zašifrovaný text a je pro útok tudíž nejobtížnější. Předpokladem úspěchu takového útoku je jistá předvídatelnost obsahu zprávy, přinejmenším by útočník měl alespoň vědět o jaký typ zprávy se jedná, zda-li text nebo např. obrázek. Při luštění dřívějších jednoduchých algoritmů právě zde našla uplatnění frekvenční analýza. Úspěchem je při tomto útoku získání dešifrované zprávy nebo použitého klíče.

5.2 Known-plaintext attack

Zde má útočník výhodu znalosti i nezašifrované podoby analyzované zprávy. Možnost přímého porovnání originálu a šifry vede ke snazšímu nalezení klíče než v případě znalosti pouze zašifrovaného textu.

Tato technika se uplatnila mimo jiné při dešifrování zpráv vytvořených strojem Enigma za druhé světové války zejména při zasílání zpráv o počasí, kde se určitá slova nacházela vždy na stejném místě a tak zjednodušila nalezení jejich zašifrovaných protějšků. Tento typ útoku je použitelný na zašifrované ZIP archivy, kde je při znalosti alespoň jednoho původního souboru z archivu získání hesla otázkou několika minut [30].

5.3 Chosen-plaintext attack

V této situaci si útočník může vybrat, jaké texty nechá zašifrovat, aby získal vzorky pro porovnání. Mezi tyto typy útoků se řadí také diferenciální kryptoanalýza. Použitím této metody se může také trochu usnadnit hledání slabin v daném algoritmu.

Tento útok se dá rozdělit na dva typy:

a) batch chosen-plaintext attack – útočník vybere sadu textů, které zašifruje najednou a následně s nimi pracuje

b) adaptive chosen-plaintext attack – útočník šifruje postupně různé texty, které si vybírá na základě zjištěných informací z předchozích šifrování.

Jiné využití tohoto útoku je vkládání vlastních částí do zprávy kterou útočník zachytí a pošle ji dál [35]. Touto technikou bylo také prolomeno zabezpečení WEP v [36].

5.4 Chosen-ciphertext attack

Tento útok je již proveditelný podstatně obtížněji než ostatní výše uvedené. V tomto případě si útočník může vybrat šifrovaný text, který bude dešifrován a následně k němu bude mít přístup. Toho by šlo teoreticky docílit tak, že by původnímu přijemci do zprávy podstrčil jím vytvořenou šifru. Příjemce by ji dešifroval a pokud by měl útočník přístup do jeho počítače (např. pomocí trojského koně), tak by se mohl dostat k dešifrovaným textům.

5.5 Related-key attack

Tímto útokem je pečlivě prozkoumán algoritmus a výpočty, které v něm probíhají. Útočník má k dispozici několik klíčů, které nemusí znát, ale ví, že je mezi nimi nějaký vztah, např. že mají stejný určitý počet bitů.

Zajímavým příkladem takto zranitelného systému je již jednou zmíněný WEP. V [37] je popsán related-key ciphertext-only útok na proudovou šifru RC4 použitou v systému zabezpečení bezdrátových sítí WEP. Šifra RC4 je v tomto systém inicializována 24 bitovým inicializačním vektorem, který je konkatenován s hlavním klíčem šifry. Autoři [37] ukázali, že tyto vektory mohou být zneužity pro získání přístupu do takto chráněné sítě. Tehdy bylo třeba zachytit 4 miliony různých datových rámců, ale od té doby se objevily další efektivnější a rychlejší způsoby, jako např. v [38], kde stejného cíle dosáhli do 60 sekund.

5.6 Birthday attack

Tento útok se týká hlavně hašovacích funkcí a jeho název je odvozen od matematického narozeninového paradoxu. Hašovací funkce pracují jako jednocestné funkce s předpokladem, že je výpočetně natolik náročné provést inverzní funkci, že to s dostupnými technickými prostředky není v praxi proveditelné. Proto přichází v úvahu pouze postupné zkoušení jednotlivých možných hodnot a porovnávání jejich haše s vzorkem, jehož původní hodnotu hledáme. Ačkoli se tento postup zdá být příliš zdlouhavý, tak díky narozeninovému paradoxu lze odhadnout kolik vzorků musíme vyzkoušet, abychom našli dvě hodnoty se stejnými haši.

Tento paradox je nejčastěji demonstrován na příkladu hledání dvou lidí z náhodně vybrané skupiny se stejným datem narození. Na tomto místě je třeba rozlišit, zda budeme hledat 2 lidi s libovolným stejným datem narození (kolize prvního řádu) nebo vybereme jednoho a k němu budeme hledat odpovídající osobu (kolize druhého řádu).

Pro vyhledávání kolizí prvního řádu v tomto příkladu pak platí následující vztah:

$$(50\%) \cong \sqrt{2 * 365 * \ln\left(\frac{1}{1-0,5}\right)} \cong \sqrt{506} \cong 22,49\ lidí \quad (5.1)$$

a pro výpočet kolizí druhého řádu v tomto příkladu platí následující vztah:

$$(50\%) \cong 1 - \left(\frac{365-1}{365}\right), \text{kde } n \text{ je počet lidí} => \cong 253\ lidí \quad (5.2)$$

Převzato z [39].

Narozeninový paradox je postaven na výše uvedených principech a ukazuje složitost výpočtu hašovacích funkcí. Při použití **n**-bitové hašovací funkce s 50% pravděpodobností nastane kolize při vypočtení $\mathbf{2^{n/2}}$ hodnot namísto předpokládaných $\mathbf{0,5*2^n}$. Proto se např. o 160 bitovém algoritmu SHA mluví jako o 80 bitovém, protože narozeninový paradox snižuje složitost hledání kolize na 2^{80}.

5.7 Útok hrubou silou

Jedná se o typ útoku, při kterém dochází k prohledávání stavového prostoru možných klíčů. Proti tomuto typu útoku jsou odolné pouze algoritmy typu Vernamova šifra, které pro šifrování používají opravdu náhodný klíč stejné délky jako je zpráva. Jinak je touto metodou teoreticky prolomitelný každý jiný algoritmus. V takovém případě pak záleží na velikosti prohledávané množiny. Obecně je považován za bezpečný takový algoritmus, který nelze rychleji prolomit jiným způsobem, než útokem hrubou silou, pak je pro bezpečnost takového algoritmu rozhodující délka klíče.

V dnešní době jsou za bezpečné považovány klíče o délce 128 a více bitů. Prolomení dříve hojně užívaneho algoritmu DES, který používá klíč o délce 56 bitů, se stalo několikrát cílem soutěže, kterou vyhlásila firma RSA. Tato soutěž se jmenovala zprvu Secret Key Challenge, později DES Challenge. Cílem těchto soutěží bylo nejdříve dokázat slabiny tak krátkého klíče, později ukázat, že jeho prolomení je otázkou několika hodin. Podle [31] se poprvé podařilo prolomit DES v roce 1997 a úkol trval 140 dní. V roce 1999 již specializovanému vítěznému stroji stejný úkol trval pouhých 22 hodin a 15 minut [32].

Při použití 128 bitového klíče dostáváme 2^{128} možných klíčů, což je rovno číslu $3,4*10^{38}$. V případě, že by pro nalezení klíče byl použit systém, který prolomil DES za 22 hodin, který otestuje 245 miliard klíčů za sekundu [33], pak by nalezení klíče trvalo zhruba $1,06*10^{21}$ let. Pro srovnání stáří celého vesmíru se odhaduje na $1,3*10^{10}$ let [34].

5.8 Diferenciální kryptoanalýza

Patří do skupiny chosen-plaintext útoků. Její základy položili E. Biham a A. Shamir v [40], kde popsali útok na 16 kolový DES, který byl efektivnější než brute force metoda. Základním nástrojem útočníka

jsou dvojice šifrovaných bloků textu, jejichž plaintextové vzory vykazují určité rozdíly. Tyto bloky plaintextu mohou být náhodné a útočník ani nemusí znát jejich obsah, důležité jsou pouze rozdíly mezi nimi a to jak se tyto rozdíly projeví v šifrovaném textu [40].

Hlavní náplní diferenciální analýzy není jen určení výstupního bitu pouze na základě bitu vstupního, ale spíše se zabývá otázkou, jak se projeví změny určitého vstupního bitu na výstupu. Útočník zkoumá operace probíhající uvnitř algoritmu tak, aby zjistil i hodnoty subklíčů v jednotlivých kolech šifrování [41].

Diferenciální kryptoanalýzu můžeme dělit na následující typy:

a) Podmíněný diferenciální útok (Conditional Differential Cryptanalys)
b) Diferenciální útok vyšších stavů (High order Differential Cryptanalysis)
c) Zkrácený diferenciální útok (Truncated Differential Cryptanalysis)
d) Nemožný diferenciální útok (Impossible Differential Cryptanalysis)
e) Bumerangový diferenciální útok (Boomerang Differential Cryptanalysis)

Jedním z nejpoužívanějších typů je Bumerangový útok, jehož princip dále přiblížím.

Bumerangový diferenciální útok

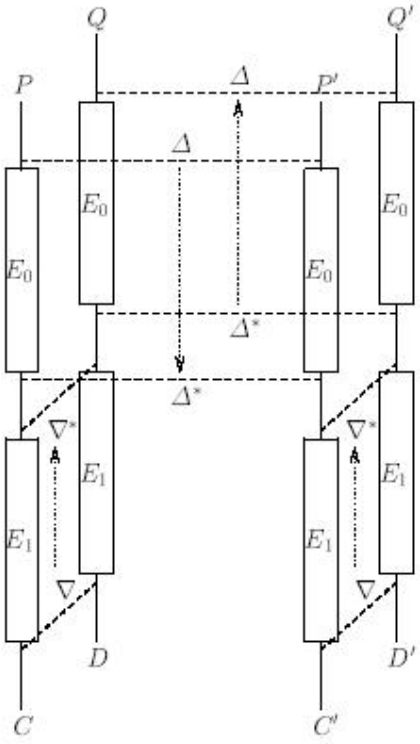
Tento typ útoku se snaží vytvořit čtveřici plaintextů v polovině šifrování bloku. K jeho provedení musí mít útočník možnost vybrání čtveřice plaintextů P, P', Q, Q' a jím odpovídajících šifrovaných textů C, C', D, D'. Nechť E (x) značí operaci šifrování bloku x. E roložíme na E_0 a E_1, kde E_0 tvoří první polovinu šifrování a E_1 tvoří polovinu druhou. Analogicky k tomuto značí E_1^{-1} první polovinu dešifrovacího procesu. Dále si vytvoříme rozdílové charakteristiky $\triangle \rightarrow \triangle^*$ pro E_0 a stejně tak $\triangledown \rightarrow \triangledown^*$ pro E_1^{-1}. Dvojice P a P' by měla pokrývat diferenci pro E_0, tj. $\triangle \rightarrow \triangle^*$. Rozdíly v párech mezi P a Q a mezi P' a Q' by měly odpovídat charakteristice E_1^{-1}, tj. $\triangledown \rightarrow \triangledown^*$. S takovýmto nastavením pak můžeme tvrdit, že pár Q, Q' bude mít charakteristiku E_0^{-1}, tj. $\triangle^* \rightarrow \triangle$.

Zaměřujeme se na hodnotu po provedení poloviny počtu kol šifrování, a pokud platí předchozí tři charakteristky, dostáváme tuto rovnici:

$$\begin{aligned} E_0(Q) \oplus E_0(Q') &= E_0(P) \oplus E_0(P') \oplus E_0(P) \oplus E_0(Q) \oplus E_0(P') \oplus E_0(Q') \\ &= E_0(P) \oplus E_0(P') \oplus E_1^{-1}(C) \oplus E_1^{-1}(D) \oplus E_1^{-1}(C') \oplus E_1^{-1}(D') \\ &= \triangle^* \oplus \triangledown^* \oplus \triangledown^* = \triangle^* \end{aligned} \qquad (5.3)$$

kde $\oplus$ je znak logické operace XOR.

Tímto máme splněnou podmínku pro charakteristiku $\triangle^* \rightarrow \triangle$ pro dešifrování první poloviny kol šifry. Pokud tedy tato podmínka platí, tak mezi plaintexty Q a Q' bude stejný rozdíl, jako mezi původními plaintexty P a P'. Průběh tohoto typu útoku je zřetelný na obr. 3.1.



Obr. 3.1: Schéma průběhu bumerangového útoku, převzato z [45].

Aby výše popsané vztahy platily, je třeba vhodně vybrat plaintexty i šifry a jejich rozdíly. Doporučeným postupem je vytvořit P' = P $\oplus$ $\triangle$. Zašifrováním bloků P a P' pak získáme C a C'. Následně vytvoříme D = C $\oplus$ $\triangledown$ a D' = C' $\oplus$ $\triangledown$. Dešifrováním D a D' získáme plaintexty Q a Q'. Popis funkce tohoto typu diferenciální kryptoanalýzy jsem čerpal z [45].

5.9 Lineární kryptoanalýza

Tato technika byla vytvořena M. Matsuim a od diferenciální analýzy se liší tím, že se nesoustředí na jednotlivé součásti (např. S-boxy), ale místo toho bere algoritmus jako celek a snaží se najít vztah mezi vstupními a výstupními bity. Jedná se o approximaci algoritmu aby jeho použití nalezeného vztahu dávalo pokud možno stejné výstupy jako původní šifrovací algoritmus.

Pokud je takto analyzováno velké množství párů otevřeného a šifrového textu a je nalezen klíč, který produkuje stejné výsledky jako původní algoritmus, je pravděpodobné, že většina bitů takto nalezeného klíč bude shodovat s klíčem, kterým byla původní zpráva zašifrována [41].

5.10 Integrální kryptoanalýza

Další metoda, která se soustředí na procesy uvnitř algoritmu a je vhodná hlavně na blokové symetrické šifry se subsitutčně-permutačními sítěmi. Za tímto účelem používá velké množství vybraných plaintextů.

Jako příklad (převzato z [42]) mějme 256 bloků o 128 bitech rozdělených do matic o velikosti 4x4 bytů. Všechny bloky jsou stejné až na jeden byte, který je pro každý blok různý. Operace XOR všech bloků pak vyjde 0. V každém kole šifrování se provede přidání subklíče, substituce jednotlivých bytů a jejich lineární transformace a funkce MixColumn změní 4 byty v daném sloupci matice. Po dvou kolech šifrování text zabírá všech 256 hodnot v každém ze 16 bytů a po třetím kole je součet všech 256 bytů na daných pozicích v matici roven 0. Tyto informace pak mohou být použity pro útok na takovýto šifrovací algoritmus.

Tento způsob útoku vznikl původně jako útok pouze na šifru Square, později byl pozměněn aby byl všestrannější. Je účinný především na algoritmy Rijndael, Square nebo Crypton, které nepoužívají více než čtyři kola šifrování.

5.11 Využití metod útoků

Ne všechny uvedené metody nacházejí širší uplantění. Příkladem za všechny nechť je útok hrubou silou. Ten je použitelný pouze na starší algortimy využívající krátkého klíče (do 64 bitů nebo šifry používající slabé klíče), jako např. DES s jeho 56 bitovým klíčem. Pro jeho prolomení hrubou silou již bylo vytvořeno několik zařízení, z posledních bych jmenoval projekt Copacobana [105]. Jedná se o speciální hardwarový lamač hesel, který se skládá ze 120 FPGA čipů a klíče nachází pomocí masivní paralelizace procesů. Možnost praktické realizace takovýchto strojů je také důvodem, proč se používají většinou šifry s klíči o délce 128 a více bitů.

Naproti tomu např. narozeninový paradox je uvažován již automaticky při řešení složitosti nalezení kolizí pro hašovací algoritmy, takže pro lagoritmy se 160 bitovou haší je uvažována složitost 2^{80}, analogicky algoritmy s např. 320 bitovou haší se bere složitost 2^{160}.

Zřejmě nejrozšířenější metodou analýzy algoritmů je diferenciální kryptoanalýza. Díky její všestrannosti (zčásti dané množstvím používaných modifikací) se stala jedním z hlavních měřítek bezpečnosti algoritmů, díky čemuž vývojáři šifrovacích algoritmů věnují při návrhu velkou pozornost zajištění odolnosti šifry proti takovým útokům. Z různých variant diferenciální kryptoanalýzy se lze nejčastěji setkat s nemožnou diferenciální kryptoanalýzou a s bumerangovým útokem.

6 Vybrané kryptografické systémy

V praxi se lze většinou setkat s kombinací výše uvedených algoritmů. Systémy, které uvedené algoritmy implementují, se tím snaží skloubit výhody jednotlivých druhů algoritmů, tj. rychlost symetrických šifer, jejichž problém předání klíče je vyřešen pomocí šifer asymetrických. Hašovací funkce dále kontroluje, zda-li nebyla data po cestě mezi komunikujícími stranami změněna. Takto je pro uživatele dosaženo maximální bezpečnosti se zachováním komfortu z hlediska rychlosti.

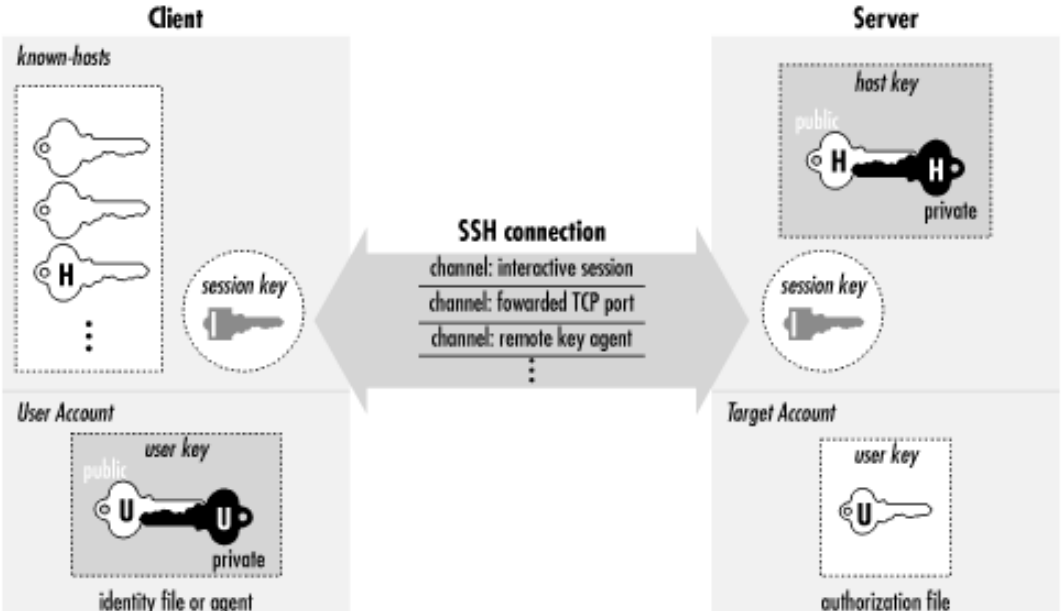
6.1 SSH - Secure Shell

SSH je protokol, který vytváří bezpečné spojení mezi dvěma stanicemi. Vznikl v roce 1995 na Technologické Univerzitě v Helsinskách a jeho autorem je Tatu Ylönen. Základními vlastnostmi tohoto protokolu je zajištění následujících podmínek pro komunikaci:

- *Zachování důvěrnosti dat* – pomocí šifrování dat
- *Integrita dat* – přenesená data nebyla nikým po cestě změněna
- *Autentizace* – ověření identity komunikujících stran
- *Autorizace* – kontrola uživatelových práv pro přístup např. k účtu
- *Tunelování* – možnost vytvoření zabezpečeného kanálu pro přesměrování TCP portů, poté lze výše zmíněných vlastností SSH využít pro další programy které pracují nad TCP

Název tohoto protokolu vznikl odvozením z názvu protokolu rsh (remote shell), který ovšem neposkytuje zabezpeční.

Tento protokol má architekturu klient/server a pracuje v aplikační vrstvě TCP/IP modelu. Obr. 6.1 ukazuje základní schéma funkce protokolu SSH. Po první verzi SSH-1 přišla hned v roce 1996 verze SSH-2, která přinesla opravu některých slabin. Obě verze jsou mezi sebou nekompatibilní, takže před začátkem komunikace se musí obě strany dohodnout, kterým protokolem budou komunikovat. Protože se obě verze v některých ohledech podstatně liší, každá bude popsána ve vlastní podkapitole.



Obr 6.1: Základní schéma protokolu SSH, převzato z [51].

Různé implementace SSH se krátce po svém vzniku rychle rozšířily a postupně nahradily programy využívající nezabezpečené spojení jako např. telnet, rlogin, rsh, rcp a rdist, a současně poskytly možnost bezpečného přenosu souborů s využitím SFTP nebo SCP případně také zabezpečený kanál pro přesměrování TCP portů a X11 spojení.

Na tomto místě bych rád uvedl některé základní pojmy a jejich význam, protože se objeví v následujících kapitolách:

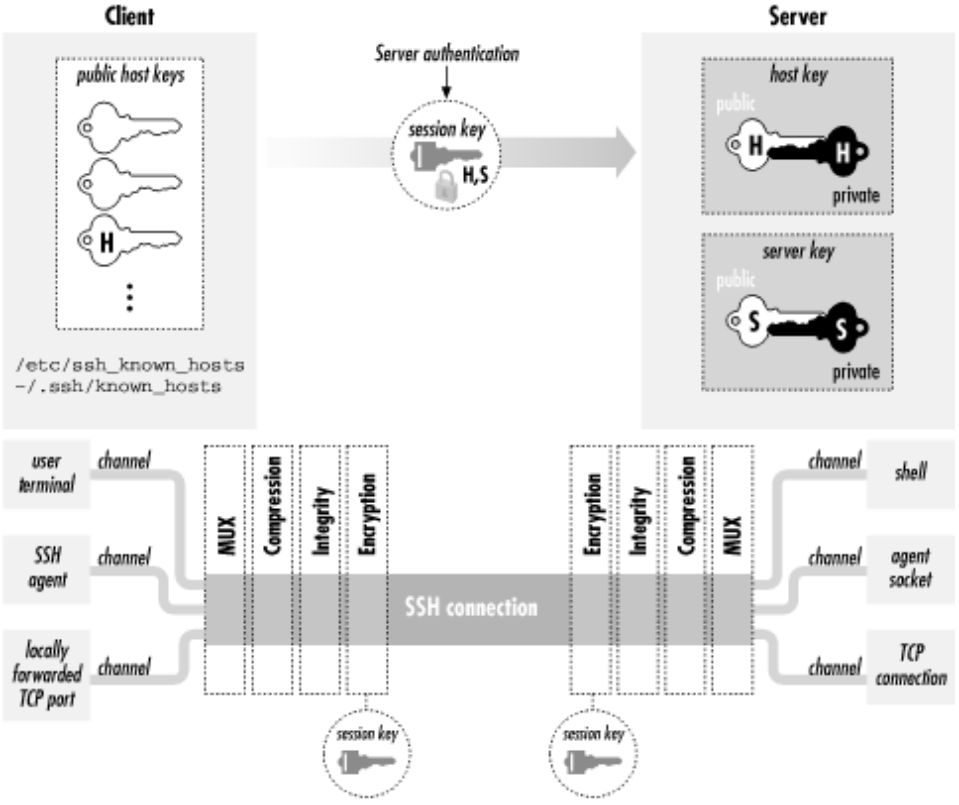
- *Server* – program na hostitelské stanici, který příjmá ssh připojení, provádí autentizaci a autorizaci a další potřebné úkony.
- *Klient* – přihlašuje se k serveru a zasílá mu požadavky.
- *Relace* – probíhající spojení mezi klientem a serverem, vzniká po autentizaci a zaniká s ukončením spojení.
- *Uživatelský klíč* – asymetrický klíč, kterým klient prokazuje svou identitu, je trvalý a veřejný.
- *Hostitelský klíč* – také asymetrický klíč, slouží k prokázání identity serveru, stejně jako uživatelský klíč je trvalý a veřejný.
- *Klíč serveru* – dočasný asymetrický klíč, který má uplatnění pouze v SSH-1, je generován serverem v pravidelných intervalech (obvykle co hodinu), jeho privátní část není nikdy nikam zasílána.

- *Klíč relace* – klíč unikátní pro každou relaci, vzniká během nastavování vlastností relace a po jejím ukončení klíč zaniká, je symetrický a používá se k zašifrování komunikace během relace.
- *Generátor klíčů* – program pro vytvoření uživatelských a hostitelských klíčů.
- *Databáze známých hostitelů* – databáze klíčů, pomocí které se klient i server snaží zjistit, zda-li spolu již dříve komunikovali.
- *Náhodné zrní* – náhodná data sloužící k inicalizaci generátoru pseudonáhodných čísel.

S těmito informacemi již můžeme přistoupit k bližšímu popisu jednotlivých verzí SSH, který jsem čerpal převážně z [47] a [51].

6.1.1 SSH-1

Na obr. 6.2 je stručný popis architektury SSH-1, který je podrobněji rozepsán na následujících řádcích.



Obr. 6.2: Architektura protokolu SSH-1, převzato z [51].

6.1.1.1 Vytvoření bezpečného spojení

Základem jakékoli komunikace je ustanovení spojení a následné zajištění jeho bezpečnosti. V SSH-1 je tento proces rozdělen do sedmi kroků:

1) *Klient kontaktuje server* – stadartní žádost o spojení na TCP port, nejčastěji na port 22.
2) *Klient a server si oznámí podporované verze SSH* – tato informace je reprezentována ASCII řetězcem, volitelně zde může být i verze implementace daného protokolu.
3) *Klient a server začnou komunikovat paketovým protokolem* – každý paket má 4 bytové pole s informací o délce paketu, 1 – 8 bytů nahodné výplně, 1 byte pro kód typu paketu a 4 bytové pole pro kontrolu integrity paketu.
4) *Identifikace serveru klientovi a ustanovení parametrů relace* – v této stále nezašifrované zprávě server zašl klientovi tyto informace:
    - svůj hostitelský klíč pro pozdější prokázání identity
    - klíč serveru použitý při vytáření bezpečného spojení
    - náhodnou sekvenci osmi kontrolních bytů, tyto byty musí klient vložit do následující zprávy zaslané serveru
    - seznam kompresních, šifrovacích a autentizačních metod podporovaných serverem

    Obě strany vypočtou 128 bitový identifikátor relace, který slouží některým operacím pro bezpečné určení relace, je to kontrolní součet hostiteského a serverového klíče a kontrolních bytů pomocí MD5. Jakmile klient obdrží hostitelský klíč, prohledá databázi známých klíčů pro zjištění, zda-li již s tímto serverem komunikoval. Pokud zde takový záznam není nebo je klíč jiný než klíč stejného serveru v databázi, klient se zeptá uživatele, jestli má klíč přijmout nebo odmítnout.
5) *Klient vytvoří a zašle klíč relace serveru* – pomocí generátoru pseudonáhodných čísel klient vytvoří klíč relace pro symetrickou šifru, kterou podporuje jak server, tak klient. Pro doručení tohoto klíče serveru klient zašifruje tento klíč relace nejdříve hostitelským klíčem a tuto šifru ještě veřejným klíčem serveru. Tímto je zajištěno, že pouze server bude schopen tento klíč relace dešifrovat. Spolu s takto zašifrovaným klíčem klient dále posílá kontrolní byty a určí šifrovací algoritmus vybraný z nabídky, kterou klient obdržel v předchozí zprávě.
6) *Zapnutí šifrování a závěr autentizace serveru* – v tomto kroku již obě strany začnou šifrovat veškerou komunikaci vybranou šifrou s dohodnutým klíčem. Nejdříve však klient čeká na potvrzení serveru, které již musí přijít v šifrované podobě. Timto server finálně prokáže svou identitu, protože pouze on dokáže dešifrovat klíč zaslaný v předchozím kroku, který byl zašifrován klíčem shodným s klíčem daného serveru nalezeným v databázi známých hostitelů. Tím, že byl tento klíč zašifrován i dočasným klíčem serveru je definitvně potvrzena identita serveru, je to vlastnost v literatuře

zmiňovaná jako *perfect forward secrecy*, což v tomto případě znamená, že ikdyby útočník získal hostitelský klíč serveru, tak by nebyl schopen dešifrovat klíč relace. Proto je velmi důležité, aby klient před jakoukoli další komunikací vyčkal do přijetí platné potvrzující zprávy. V případě, že by obsah této zprávy neodpovídal korektním hodnotám, klient ukončí spojení.

7) *Bezpečné spojení je navázáno* – v tuto chvíli již server i klient komunikují pomocí šifrovaných zpráv.

6.1.1.2 Autentizace klienta

Po ustanovení bezpečného spojení se klient pokusí autentizovat serveru, zkouší to tak dlouho, dokud neuspěje nebo nevyčerpá všechny možnosti autentizace, které protokol nabízí. Protokol SSH-1.5 nabízí šest možností autentizace:

1) Heslem
2) Veřejným klíčem
3) Pomocí Rhosts
4) Pomocí RhostsRSA
5) Systémem Kerberos
6) Systémem typu One Time Pad

Autentizace heslem

Základní možnost autentizace, po vytvoření bezpečného spojení uživatel zadá do svého SSH klienta heslo, které je následně v zašifrované podobě zasláno serveru. Ten zjistí, zda přijaté heslo odpovídá požadovanému účtu a na základě tohoto povolí či zamítne spojení. Tento způsob je vhodný pro uživatele, kteří často střídají počítače, na kterých nechtějí nebo nemohou používat svůj privátní klíč. Negativem tohoto způsobu autentizace jsou možná bezpečnostní rizika při zadávání hesla do neznámých počítačů (uživatel si nemůže být jist, zda na počítači neběží program pro logování stisknutých kláves), stejně tak uživatel nemá jistotu co se s heslem děje na serveru v případě, že by server byl úspěšně napaden. Tato negativa uspokojivě řeší až autentizace veřejným klíčem nebo systémy s jednorázovými hesly, tzv. One Time Password systémy (OTP).

Autentizace veřejným klíčem

Při využití tohoto typu autentizace klient dokazuje, že zná tajnou část klíče, který přísluší jeho účtu. Pro operace s veřejným klíčem je v SSH-1 implicitně použita šifra RSA. Samotný proces autentizace má takovýto průběh:

1. Klient zašle serveru žádost o autentizaci veřejným klíčem a do této žádosti rovnou vloží svůj veřejný klíč.
2. Server zjistí, zda má požadovaný účet ve svém autorizačním souboru záznam s klíčem shodným s tím, který obdržel. Pokud takovýto záznam nenalezne, autentizace veřejným klíčem selže.
3. Poku server nalezne tento klíč pro daný účet, zkontroluje poznámky u tohoto klíče, jestli zde jsou pro použití tohoto klíče nějaká omezení, která by v daném případě znemožnily autentizaci.
4. Pokud je vše v pořádku, server vygeneruje výzvu – to je náhodný 256 bitový řetězec, který je zašifrován veřejným klíčem klienta a je mu zaslán.
5. Klient výzvu po přijetí dešifruje svým soukromým klíčem, dešifrovanou výzvu sloučí s identifikátorem relace a na výsledek použije hašovací funkci MD5. Tuto hodnotu pak zašle zpátky serveru. Použití identifikátoru relace slouží jako ochrana proti replay útokům. Hašování je zde pro potlačení možnosti zneužití soukromého klíče klienta – v případě, že by klient posílal zpět jen výzvu dešifrovanou svým soukromým klíčem, mohla by nastat tato situace: napadený server zašle jako výzvu určitý text a použitím soukromého klíče by klient tento text v podstatě podepsal elektronickým podpisem.
6. Server vypočítá haš výzvy a identifikátoru relace a pokud se shoduje s hodnotou přijatou od klienta, pak je klient úspěšně autentizován.

Autentizace pomocí veřejného klíče se dá považovat, za jednu z nejbezpečnějších metod. Pro její úspěšné provedení musí klient mít soukromý klíč a heslo pro jeho dešifrování. Privátní klíč není nikam posílán a pokud má uživatel dostatečně silné heslo, jeho nalezení bruteforce metodou není příliš reálné. Pokud se útočníkovi povede získat jednu z těchto dvou položek, stále mu to nestačí pro přístup k uživatelovu účtu. Pokud je to potřeba, heslo je možné změnit bez nutnosti změny klíče.

Další vlastnotí, kterou se liší např. RhostsRSA je, že server nedůvěřuje informacím, které klient poskytuje – jediným kritériem, které musí splnit je důkaz o vlastnictví soukromého klíče. Samozřejmě pokud útočník úspěšně napadne uživatelovu stanici a získá klíče i hesla, pak má přístup k jeho účtům zajištěn, ovšem SSH ze své podstaty ani nemůže chránit proti takovýmto útokům.

Spolu s autentizací tato metoda umožňuje provádět některé kroky autorizace – v konfiguračních souborech lze definovat omezení pro jednotlivé klíče.

Ovšem ani tento způsob není dokonalý, jeho nevýhodou jsou vyšší technické nároky na uživatele. Ten musí generovat a spravovat klíče a autorizační soubory a s tím spojené možné chyby (špatná přístupová práva souborů či adresářů apod.). Při ztrátě privátního klíče je nutné vygenerovat novou dvojici privátního a veřejného klíče a tuto informaci je třeba zanést do všech účtů, které

využívaly starý klíč. SSH nemá žádný systém pro bezpečný transport klíčů, takže to je další záležitost, kterou musí uživatel vyřešit.

Navzdory výše uvedeným nedostatkům bych tento způsob autorizace spíše doporučil vzhledem k poměru bezpečnost/náročnost instalace a správy, kterou tento způsob nabízí.

Rhosts a RhostsRSA

Oproti dvěma výše uvedeným způsobům je zde využit trochu jiný princip. Při autentizaci heslem nebo veřeným klíčem se uživatel prokazuje znalostí nebo vlastnictvím nějakého tajemství a počítač, ze kterého se přihlašuje v tomto procesu nehraje roli. Naproti tomu systémy Rhosts a RhostsRSA vytváří důvěrné spojení mezi stanicemi. Proto se uživatel jen přihlásí do systému stanice v takto zabezpečné síti a pokud se chce přijpojit na jiný účet na jiné stanici a má na to oprávnění, pak nemusí zadávat žádná hesla ani klíče a Rhosts již autentizaci sám provede.

Při pokusu o spojení se také ověřuje, zda o spojení žádá důvěryhodný program, který potvrdí uživatelovu identitu. Tato kontrola se provádí pomocí tzv. privilegovaných portů, což je konvence pocházející ze starších Unixových systémů. Jde o to, že na takovýchto systémech jsou porty 1 až 1023 považovány za privilegované a ke komunikaci je mohou používat pouze programy, které mají administrátorská práva, z čehož vyplývá, že byly instalovány administrátorem. Na základě důvěryhodnosti takovýchto programů pak systém předpokládá, že takovýto privilegovaný program nemůže lhát o uživatelově identitě (jinými slovy – že běžný uživatel nemůže změnit takovýto program, aby se za někoho mohl vydávat).

V SSH-1 jsou podporovány jak Rhosts, RhostsRSA. V defaultním nastavení je ovšem použit pouze systém RhostsRSA, který poskytuje lepší bezpečnost než Rhosts. Základní kroky pro vytvoření bezpečného spojení jsou pak:

1. SSH klient zašle SSH serveru žádost o spojení.
2. SSH server vyhledá podle adresy jméno stanice, která žádá o spojení. Pro RhostsRSA dále platí, že každý SSH klient má svůj asymetrický klíč, který jej identifikuje, takže RhostsRSA se nespoléhá pouze na IP adresu stanice, ale pro autentizaci stanice využívá také asymetrické kryptografie se stejnou výměnou výzvy jaká je při samotné autentizaci veřejným klíčrm v předcházející podkapitole.
3. SSH server vyhledá záznamy o stanici ve své databázi a podle výsledku hledání určí, zda je stanice důvěryhodná či ne.
4. Dále server zkontroluje, zda-li je program žádající o spojení privilegovaný. Pro RhostsRSA je tato kontrola provedena již v kroku 2, protože soukromý klíč je na disku

chráněn tak, že pouze program s administrátorskými právy se s ním může prokázat. Tímto je v kroku 2 splněna i podmínka z kroku 4.

5. V tomto kroku je stanice úspěšně autentizována.

V dnešní době není velký problém překonat kontrolu spočívající v důvěryhodnosti adresy stanice. Stejně tak nalezneme mnoho operačních systémů, které nepodporují koncept privilegovaných portů nebo ve kterých uživatelé pracují pod účtem s administrátorskými právy. Z těchto důvodů rozhodně nelze doporučit používání systému Rhosts. V úvahu pak přichází pouze použití RhostsRSA, ale jen ve specifickém prostředí, které splňuje nároky na zabezpečení systému a kde běžný uživatel nemá administrátorská práva (typicky podniky se sítí unixových systémů). I zde je však nutné velmi pečlivé nastavení systému RhostsRSA, aby bylo minimalizováno riziko bezpečnostní mezery vzniklé špatnou konfigurací.

Výhodou tohoto řešení je pak jednoduchost pro uživatele, který nemusí znát žádná další hesla než to, které potřebuje k přihlášení se ke stanici. Tohle je výborná vlastnost i při použití naplánovaných úloh (např. cron), kde není nutné mít hesla či klíče v různých skriptech (což by z hlediska bezpečnosti bylo více než nevhodné řešení).

Kerberos

SSH lze také využít spolu s autentizačním a autorizačním systémem Kerberos. Jak jsem již uvedl, jako jeden z nejlepších způsobů autentizace považuji autentizaci s veřejným klíčem. Jeho zmíněné nevýhody týkající se zajištění správy a distribuce klíčů jsou systémem Kerberos eliminovány.

Základem celého systému je zabezpečený server pro distribuci klíčů (KDC server), který má funkci důvěryhodné třetí strany je složen z částí autentizační server (AS) a server pro generování lístků (z ang. Ticket Granting Server - TGS). Postup autentizace je ve stručnosti následovný (zdroj [54]):

1. Klient zašle AS žádost se svým ID o udělení Ticket Granting Ticketu – lístek, na jehož základě může klient obdržet lístky pro přístup ke vzdáleným službám.
2. AS podle ID zkontroluje, zda je klient v databázi uživatelů. Pokud ano, tak vytvoří TGT s časovým razítkem a dobou platností standartně 8 hodin (ochrana proti pokusů o nalezení klíče hrubou silou, je malá pravděpodobnost, že se to podaří do 8 hodin).
3. TGT spolu s klíčem relace jsou zaslány klientovi.
4. O lístek pro komunikaci s konkrétní službou již klient žádá TGS pomocí TGT.
5. Pokud je TGT stále platný, TGS zašle klientovi lístek pro využití dané služby.
6. Klient následně kontaktuje požadovanou službu a zašle jí lístek.

7. Služba zkontroluje platnost lístku a zažádá KDC server o lístek, který následně zašle klientovi.
8. Klient zkontroluje platnost lístku a komunikace mezi ním a službou může začít.

Pro podniky, kde je již zavedený systém Kerberos je výše popsaný způsob autentizace zřejmě ideálním řešením, nicméně raději ve spojení s protokolem SSH-2, protože SSH-1 již neposkytuje dostatečné zabezpečení.

One Time Password

Systému typu One Time Password neboli OTP kombinují jednoduchost a přenositelnost použití s ochranou proti ztrátě či odcizení hesla. Pokud uživatel často střídá stanice, ze kterých se přihlašuje, hrozí riziko, že na některém může být nainstalován program pro logování stisklých kláves. V takovém případě byl jeho účet okamžitě napadnutelný. Použití OTP zajistí, že při každém přihlášení bude požadováno jiné heslo. Zde je pár příkladů OTP systémů:

1) S/Key™ OTP systém nabízí možnost vytištění hesel na papír a uživatel pak při každém přihlášení zadá odpovídající heslo, jiná možnost je, že před každým přihlášením si uživatel vypočte heslo pomocí programu na svém PDA nebo notebooku.
2) Firma RSA Security, Inc. nabízí systém SecurID. Tento systém používá hardwarový token, který je synchronizován se SecurID serverem a na svém displeji zobrazuje aktuální platné heslo, které je obměňováno v pravidelných intervalech
3) Firma Trusted Information Systems, Inc. nabízí pro změnu systém, ve kterém server zobrazí výzvu, kterou uživatel zadá do svého tokenu (může být hardwarový či softwarový) a ten vytvoří korektní odpověď, se kterou se užviatel následně přihlásí.

6.1.1.3 Kontrola integrity

Zde se dostáváme k jednomu z důvodů, proč se SSH-1 dočkala nástupce tak brzy. SSH-1 totiž kontroluje integritu dat pouze pomocí jednoduchého kontrolního součtu CRC-32. Tato metoda je dostačující pro kontrolu náhodné změny některého bitu při transportu dat, ale proti cílenému útoku neodolá. Tímto tématem se budu konkrétněji zabývat v kapitole věnované útokům na SSH. Nicméně už v tuto chvíli mohu prohlásit, že slabý kontrolní součet je jednou z největších slabin SSH-1 a také důvod, proč bych jeho použití dále nedoporučoval.

6.1.1.4 Komprese dat

Komprese dat probíhá ještě před šifrováním a použitý kompresní algoritmus je GNU gzip. Díky linkám s vysokou propustností není již použití komprese natolik znatelné jako v době, kdy bylo běžné vytáčené spojení.

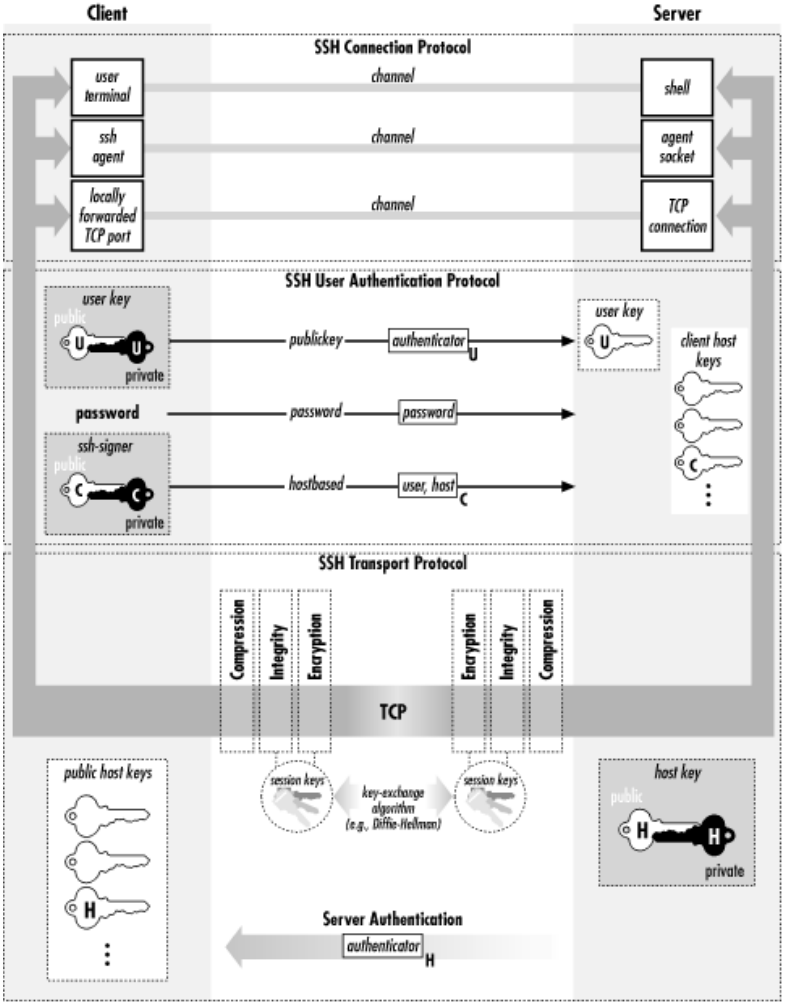
6.1.2 SSH-2

Vytvoření SSH-2 nespočívalo jen v doplnění SSH-1 o další užitečné vlastnosti a v opravě chyb, tato verze by se dala považovat za úplně nový protokol. SSH-1 bylo kompletně přepsáno a z jeho monolitické struktury vznikl modulární systém tvořený třemi protokoly [47]:

- *SSH Transport Layer Protocol* – základní protokol, který zajišťuje i funkci ostatních protokolů. Má na starosti zahájení spojení, autentizaci serveru, kontrolu integrity a základní šifrování.
- *SSH Authentication Protocol* – Tento modul slouží pro autentizaci klienta.
- *SSH Connection Protocol* – Tento protokol se stará o všechno kolem tunelování spojení, forwardování portů a přístupu ke vzdáleným službám.

Hierarchie uspořádání výše uvedených částí není striktně vertikální. Platí, že základním protoklem je SSH Trasnport Layer Protocol, ale zbylé dva protokoly již pracují v jedné rovině, dokonce jsou na sobě nezávislé – příkladem takového chování by byl systém nastavený na tunelování jakéhokoli spojení přes určitý port bez nutnosti autentizace klienta.

Výsledkem takto radikálních změn je kromě zvýšené bezpečnosti také vzájemná nekompatibilita mezi oběma verzemi SSH. V následujících podkapitolách se zaměřím na prvky SSH-2, které se od verze SSH-1 liší nejvíce.



Obr. 6.3: Schéma komunikace protokolem SSH-2, převzato z [51].

6.1.2.1 Výběr parametrů spojení

V SSH-1 si komunikující strany předložily podporované typy šifrování dat a jeden z typů následně používaly. Ovšem tím byly vyčerpány všechny možnosti volby, kterou strany měly. RFC 4251 [47]) poskytuje informace o dalších možnostech, které SSH-2 přináší – komunikující strany si mohou vybrat hostitelský klíč (pokud jich má server více), způsob výměny klíčů relace, způsob autentizace zprávy (MAC), hašovací funkci a metodu komprese dat.

Další inovací je možnost rozšíření počtu podporovaných alogritmů o vlastní, které musí mít přesný formát a v použité implementaci musí být jasně dané priority jednotlivých algoritmů. Pro tento případ existují dvě možnosti pojmenování protokolů:

- Jména přidělená IETF konsensem – nesmí obsahovat znak "@", "," a ani žádné ASCII znaky s kódem 32 a nižším nebo kódem 127, délka je maximálně 64 znaků. Mezi

takovéto názvy patří např. "3des-cbc", "sha-1", "hmac-sha1" a "zlib". Tato jména jsou platná pouze pouze pokud jsou již registrována organizací IANA.

- Kdokoli si však může nadefinovat další algoritmus, ovšem aby ho mohl použít v SSH-1, tak se pro jeho pojmenování v SSH-2 musí držet následujících pravidel – formát jména musí být ve tvaru jméno@doména, např. moje_sifra@fit.cz, v názvu musí být přesně jeden znak "@", musí se skládat ze znaků US-ASCII abedecedy, nesmí obsahovat znak "," a ani žádné ASCII znaky s kódem 32 a nižším nebo kódem 127, délku musí mít maximálně 64 znaků a jméno domény musí být platné.

6.1.2.2 Výměna klíčů

V této verzi je už výměna klíčů standartně prováděna pomocí algoritmu Diffie-Hellman (pokud se strany nedohodnou jinak). Je zde opět zaručena perfect forward secrecy, prozrazení klíče relace nebo dokonce privátního klíče jedné strany po skončení relace nemůže vést k prozrazení informací z dřívějších relací. V případě prozrazení soukromých parametrů D-H metody pro tvorbu klíče obou stran zároveň by samozřejmě bylo možné nalézt klíč relace, z toho důvodu je vhodné tyto parametry zrušit, jakmile je klíč relace vytvořen. Díky tomu, že D-H metoda umožňuje výměnu klíče relace s dodržením podmínky perfect forward secrecy, není nutné aby měl SSH server ještě klíč serveru, ze kterého se tímto stává přežitek.

6.1.2.3 Certifikáty

Jedním z nejdůležitějších aspektů asymetrické kryptografie je kontrola identity majitele veřejného klíče. Pokud chce uživatel Bob poslat zprávu Alici, musí mít jistotu, že zprávu zašifruje jejím veřejným klíčem. Pakliže by mu útočník podstrčil svůj veřejný klíč, Alice s Bobem by to rychle zjistili tak, že by zprávu Alice nebyla schopna dešifrovat, ale to by již útočník měl dávno zprávu přečtenou.

V tuto chvíli je potřeba zavést důvěryhodnou třetí stranu, která je schopna autoritativně potvrdit identitu majitele klíče. Autoritu této třetí strany je také potřeba ověřit čímž získáváme hierarchický strom certifikačních autorit. Ten tvoří základ infrastruktury veřejných klíčů. Obsahem těchto certifikátů nejčastěji bývají informace o majiteli certifikátu, jeho veřejný klíč, údaje o platnosti certifikátu a autorita, která certifikát vydala.

Můžeme mít dva modely certifikátů:

- Klient má databázi klíčů a k nim přiřazené odpovídající vlastníky. Při této volbě není třeba mít důvěryhodnou třetí stranu, nicméně se vzrůstajícím množstvím položek v databázi bude její správa stále náročnější.

- Přiřazení klíčů k jejich vlastníkům řeší certifikační autorita, klient má pouze veřejný klíč této autority a předpokládá správnost údajů, ke kterým se váže certifikát podepsaný touto autoritou.

SSH-2 umožňuje využití certifikátů ve formátech X.509 [114] a OpenPGP [115].

6.1.2.4 Autentizace

I tato část doznala určitých změn. V první řadě server nemusí oznámit podporované metody autentizace pouze před samotným začátkem autentizace, ale i kdykoli během ní a hlavně může seznam podporovaných způsobů měnit podle přednastavených pravidel. Např. po několika neúspěšných pokusech o přihlášení pomocí veřejného klíče může server tuto metodu zakázat a nadále povolit pouze přihlášení heslem.

Ve [48] je také popsána možnost částečné autentizace – server může ohlásit, že autentizace danou metodou byla úspěšná, ale že je nutné autentizovat se ještě jednou metodou. Dá se tak kombinovat např již zmíněná autentizace veřejným klíčem s autentizací heslem.

Vylepšena byla i Host–Based autentizace. V SSH-1 byla klientova identita svázána s jeho síťovou adresou, díky čemuž byl tento způsob nepoužitelný např. pro majitele notebooků, které se často nacházejí v různých sítích nebo pro klienty za proxy serverem. V SSH-2 může klienta identifikovat jak síťová adresa, tak i jméno klientovy hostitelské stanice, které je serveru posláno v žádosti o autentizaci. Pokud se uvedené jméno nachází v databázi známých hostitelů a odpovídá i jeho klíč, pak může server údaj o adrese ignorovat a přijmout autentizaci.

6.1.2.5 Kontrola integrity

Kontrola integrity byla v SSH-1 jedním z nejslabších článků. Místo kryptograficky slabého kontrolního součtu jsou v SSH-2 již podporovány silné MAC algoritmy:

- hmac-sha1 – tento algoritmus musí všechny implementace podoporovat, ostatní jsou volitelné.
- hmac-md5
- hmac-sha1 96
- hmac-md5 96

Jedná se o algoritmy, u kterých se podařilo nalézt kolize (tj. block dat, který má stejný haš jako originál, ale neshoduje se s originálem), nicméně v případě sha-1, je nalezení kolize stále časově natolik náročnou záležitostí, že není pravděpodoné, aby bylo možné ohrozit probíhající relaci.

6.1.2.6 Obměna klíčů

Čím více dat zašifrovaných jedním klíčem má útočník k dispozici, tím je větší šance, že tento klíč objeví. Proto v SSH-2 našla uplatnění funkce, která umožňuje kterékoli straně iniciovat genreování nového klíče relace. Asymetrické šifry jsou používány na malá množství dat, tak jse jich uvedený problém příliš netýká, ale symetrickou šifrou mohou být šifrovány gigabyty dat např. při přesunu dat pomocí SCP, což dává útočníkovi velké množství imformací.

6.1.2.7 Zpětná kompatibilita

Jak jsem již uvedl, změny, které SSH-2 přináší jsou natolik výrazné, že není možné aby jedna strana komunikovala protokolem SSH-1 a druhá strana protokolem SSH-2. Ovšem je zde řešení, které umožňuje, aby si strana vybrala verzi protokolu po dohodě s druhou stranou – kontrétně tedy pokud budu mít klienta s protokolem SSH-1 a budu se chtít připojit na server, který podporuje jen SSH-2, je to proveditelné. Je to ovšem řešeno tak, že na dané stanici musí být přítomny moduly obou verzí protokolu – pak může server využít modulů starší verze. V praxi to není zrovna ideální řešení, protože se úměrně tomu zvedá počet možných chyb v konfiguraci (je potřeba mít bezpečně nakonfigurované dva samostatné protokoly) a zvětšuje dobu potřebnou pro připojení se SSH-1 klienta na SSH-2 server.

6.1.3 Algoritmy použité v protokolu SSH

6.1.3.1 Algoritmy s veřejným klíčem

SSH-1 má pro tuto část komunikace vyhrazen pouze algoritmus **RSA**. Pro připomenutí uvádím, že bezpečnost této metody je založena na problému faktorizace velkých čísel a v současnosti to lze považovat za dostatečnou záruku bezpečnosti. Samozřejmě bezpečnost algoritmu v praxi se také odvíjí hlavně od délky klíče a od parametrů použitých pro generování klíče, tj. spoléhá také na kvalitní výstup generátoru pseudonáhodných čísel.

SSH-2 přináší použití dalších dvou algoritmů. Prvním z nich je **DSA**, která je standartizovanou šifrou pro digitální podpis. Složitost jejího prolomení spočívá v řešení problému diskrétního logaritmu. V SSH-2 je použita pouze pro identifikaci hostitele.

Třetí šifrou, kterou lze v SSH-2 standartně nalézt, je již dříve zmíněný algoritmus **Diffie–Hellman**, který je znám již od roku 1976. Tento algoritmus slouží dvěma stranám k předání si určitého tajemství po nezabezpečeném kanále. V SSH-2 proto našel uplatnění při výměně klíče symetrické šifry použité pro další komunikaci (viz také [49]).

6.1.3.2 Symetrické šifry

Symetrické šifry, které jsou použity v SSH (podle [55]) jsou blíže popsány v kapitole 5, takže zde uvádím jen stručné shrnutí doplněné o informace specifické pro SSH:

- **DES** –algoritmus je podporován pouze v SSH-1, použití toho algoritmu nelze z důvodu nízké bezpečnosti doporučit.
- **3DES** – algoritmus, jehož podpora je vyžadována v implementaci obou verzí protokolu, bezpečnější než DES, ale za cenu malé rychlosti šifrování a dešifrování.
- **ARCFOUR (RC4)** – Protože jméno algoritmu RC4 je chráněno autorskými právy, je název ARCFOUR vodítkem, které má naznačovat, že se jedná o šifru založenou na kódu, který unikl z RSA Security, Inc., čiliže prakticky šifru RC4. Opět ji lze nalézt v obou verzích SSH. V SSH-1 používá 128 bitový klíč a různý pro každý směr komunikace, v SSH-2 může mít klíč různou délku. Její použití je v SSH-1 doporučeno jako náhrada za 3DES, nicméně kvůli slabinám popsaným v [26] bych místo ní doporučil spíše jiný algoritmus, např. Twofish. Jedná se o jedinou proudovou šifru použitou v SSH.
- **IDEA** – algoritmus podporovaný oběma verzemi protokolu, pro SSH-1 se jedná opět o doporučený algoritmus.
- **CAST–128** – algoritmus podporovaný v SSH-2 a jak již z názvu vypovídá, jedná se o verzi algoritmu se 128 bitovým klíčem.
- **Blowfish** – tímto se dostáváme k doporučeným algoritmům pro SSH-2. Prvním z nich Blowfish je starší pokus o nástupce DESu, je rychlejší než DES i IDEA, rychlosti RC4 ale nedosahuje. Při použití v SSH-2 pracuje s klíčem o délce 128 bitů.
- **Twofish** – další doporučený algoritmus pro SSH-2. Klíč má v tomto případě délku 256 bitů. Twofish je nástupcem Blowfish, je velmi rychlý a jeho použití není nikterak licečně omezeno.

6.1.3.3 Hašovací funkce

Hašovací funkce v protokolu SSH zajišťuji kontrolu integrity zasílaných dat. Je jim taky věnována samostatná kapitola, takže na tomto místě se nachází pouze stručný přehled:

- **CRC–32** – funkce pro kontrolní součet, nejedná se ani o kryptografickou funkci, umí pouze rozponat náhodné poškození dat, neochrání proti cílenému útoku [116].
- **MD5** – již poměrně starý algoritmus s délkou haše pouze 128 bitů. K datům hašovaným pomocí MD5 lze ovšem pomocí diferenciální kryptoanalýzy nalézt kolize

během několika sekund na standartním počítači typu Pentium4 3GHz (podrobný popis celého útoku je v [70]).

- **SHA–1** – jedná se v podstatě o vylepšení algoritmu MD4 a MD5, délka haše je zvětšena na 160 bitů.
- **RIPEMD–160** – další 160 bitový hašovací algoritmus, ale oficiálně není v SSH obsažen, pod názvem `hmac-ripemd160@openssh.com` jej využívá implementace OpenSSH.

6.1.3.4 Algoritmy pro kompresi dat

Obě verze protokolu SSH využívají pouze algoritmus **zlib**, který vychází z algoritmu **deflate** použitého poprvé v programu gzip.

6.1.4 SSH – implementace

Velmi záhy po svém prvním uvedení začaly vznikat různé programové implementace tohoto protokolu, ať v už komerční sféře nebo na poli open source softwaru. Ačkoli je verze SSH-2 na trhu již velmi dlouhou dobu, pozice SSH-1 je stále silná kvůli licenčním podmínkám jednotlivých protokolů. Protokol SSH-1 je volně poskytnut k použití, ale SSH-2 již byl uveden jako komerční produkt, který měl striktní licenční omezení. Ta se v pozdějších letech začala uvolňovat a protokol SSH-2 je k dispozici pro nekomerční využití v různých open source implementacích.

Protože jsou již dnes volně dostupné implementace podporující vyspělejší a bezpečnější verzi SSH-2, do krátkého přehledu nebudu zařazovat aplikace podporující pouze SSH-1.

6.1.4.1 Komerční produkty

- SSH Tectia Client/Server – řešení pocházející od firmy SSH Comunications Security, Inc., jejímž zakladatelem je autor protokolu SSH Tatu Ylönen. [56]
- SecureCRT – program společnosti VanDyke Software. [57]
- ZOC – produkt spoečnosti EmTec. [58]

Tyto produkty pokytují obvyklé služby, krom zabezpečeného spojení např. také bezpečný přenos dat. Jsou to řešení vhodná spíše pro podnikovou sféru, kvůli zajištění technické podpory svých produktů. Vyjmenované produkty nejsou jedinými na trhu, těch je velmi mnoho a proto jsem uvedl pouze pár příkladů.

6.1.4.2 Open source produkty

Tyto aplikace zvládají také všechny potřebné operace bez nějakých omezení, od komerčních produktů se většinou liší absencí technické podpory. Implementací SSH je velmi mnoho, proto uvádím jen pár známých příkladů.

- OpenSSH [59]
- PuTTY [60]
- portaPutty – upravená verze, která neukládá informace do registrů, ale do souborů a umožňuje tak spuštění např. z přenosné flash paměti [61]

6.1.5 SSH – shrnutí

Ačkoli je SSH velmi propacovaný systém, je nutné se při jeho použití řídit určitými pravidly, které útočníkovi znemožní zneužití chyb či slabin v návrhu. Pro začátek je vhodné mít preferované algoritmy, pro jednotlivé části spojení – mojí volbou by byly následující parametry spojení:

- **Autentizace** – zde bych jednoznačně využil autentizace pomocí veřejného klíče. Pokud je dobře vyřešeno uložení privátního klíče (kontrola přístupu k souboru), tak se jedná o jeden z nejbezpečnějších způsobů autentizace.
- **Symetrická šifra** – zde se mi jako ideální kandidát jeví šifra Twofish, pro svou rychlost a bezpečnost. Dalším využitelným kandidátem by mohla být IDEA.
- **Kontrola integrity** – důrazně nedoporučuji použití čehokoli slabšího než SHA-1, případně pokud to implementace dovoluje je také dobrá alternativa RIPEMD-160.

Další aspekty bezpepečného využití se týkají samotného nastavení SSH serveru a klienta. Při nastavování parametrů server bych nepovolil zpětnou kompatibilitu se starší verzí SSH-1. To zamezí využití slabin, které se již SSH-2 netýkají (např. problémy s kontrolou integrity). Také to značně zjednoduší správu serveru a urychlí jeho práci. Současně je vhodné mít zakázány nepoužívané metody autentizace a ani nenabízet šifry, které nejsou považovány za bezpečné, to vše aby byla minimalizována možnost zneužití chyby v některé části, která není pro správnou funkci potřebná.

Ve [109] autoři Song, Wagner a Tian poukazují na určité vlastnosti SSH, které mohou být zneužity. První vlastností je zarovnávání paketů na 8 bytovou hranici, to umožňuje určení přibližné velikosti zasílaných dat, díky tomu lze určit např. zda je heslo kratší než 7 znaků nebo ne. Další potenciální slabina se týká interactive módu, ve kterém je odeslán paket s daty po každém stisku klávesnice. Při přesné analýze procházejících dat tak útočník může získat přesnou délku hesla a existují i metody (také popsané ve [109]), které ukazují analýzu jednotlivých časových úseků mezi

stisky klávesnice a na základě těchto informací může útočník zůžit výběr možných hesel. Takže potlačení možnosti použití interactive módu je také vhodnou volbou.

Další předpokladem zaručení bezpečnosti je udržování SSH softwaru v aktuálním stavu, neboť v různých implementacích se nachází chyby, které nejsou dány návrhem SSH, ale špatnou implementací. Existuje několik stránek, které se zabývají upozorňováním na nově nalezené chyby, za všechny jmenujme např. web Secunia (www.secunia.com).

Na následující tabulce 6.1 lze vidět základní rozdíly mezi oběma verzemi SSH.

Vlastnost	SSH-1	SSH-2
Možnost uživatelsky definovaných parametrů spojení	Možnost výběru pouze symetrické šifry	Šifra pro autentizaci, kontrolu integrity, symetrická šifra
Podporované autentizační metody	Heslem, veřejným klíčem, Rhosts a RhostsRSA, systémy OTP, Kerberos	Heslem, veřejným klíčem, Host-based, systémy OTP, Kerberos, kombinace
Perfect forward secrecy	Pomocí klíče serveru	Pomocí algoritmu Diffie-Hellman pro výměnu klíčů
Kontrola integrity	CRC-32	SHA-1, MD5
Podpora certifikátů	Ne	Ano
Periodická obměna klíče relace	Ne	Ano

Tab. 6.1: Přehled rozdílů SSH-1 a SSH-2.

SSH je kvalitní nástroj pro práci se vzdálenými prostředky v nezabezpečeném prostředí, stejně tak nic nebrání uživatelům používat jej v síti, o které se domnívají, že je bezpečná. Navzdory popsaným nedostatkům první verze, je možné se s jejím použitím stále často setkat. Lze však vidět trend, kdy je z důvodu bezpečnosti upouštěno od podpory SSH-1 a uživatelé jsou tak nuceni přejít na SSH-2. Tomuto trendu nahrává také rostoucí dostupnost neplacených aplikací, které komunikují protokolem SSH-2.

Osobně bych nad použitím SSH-1 ani neuvažoval a rovnou sáhl po aplikaci, která mi poskytne výkon a bezpečnost protokolu SSH-2.

6.2 SSL/TLS

6.2.1 Vznik a verze protokolu

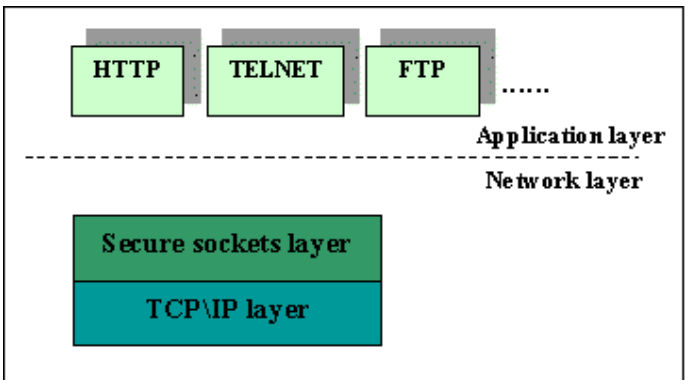
Protokoly Secure Sockets Layer a jeho následovník Transport Layer Security jsou vyvinuté pro zabezpečení komunikace na internetu. Stejně jako SSH je to protokol s modelem klient/server, byl však navržen pro širší využití – jeho cílem je zabezpečení informací při jejich zadávání na webových stránkách, při komunikaci e–mailem nebo pomocí internetových komunikátorů (instant messaging), apod.

Protokol SSL byl zaregistrován na americkém patentovém úřadě 25. srpna 1995 a uvedenými autory jsou Taher Elgamal a Kipp E. B. Hickman, tehdejší zaměstnanci firmy Netscape Communications [62]. Jeho vznik se však datuje do dřívějších let. První verze 1.0 nebyla zveřejněna, verze 2.0 z roku 1994 obsahovala četné bezpečnostní mezery, což vedlo k neustálému procesu updatů. Ne všechny problémy mohly být řešeny takto, takže v roce 1996 vyšla poslední verze SSL – 3.0. Následně byl vývoj tohoto protokolu převzán společností Internet Engeneering Task Force (IETF) a postupoval již pod názvem TLS.

TLS ve verzi 1.0 se příliš nelišilo od SSL 3.0, větší změny přinesla až aktuální verze 1.1, která mění některé parametry spojení v rámci ochrany před útoky. Momentálně již existuje návrh na verzi 1.2 [63]. V hlavičkách jednotlivých paketů chráněných pomocí SSL/TLS se také udává verze protokolů. Zde je možno vidět provázanost obou protokolů, protože hlavní číslo verze se do dob SSL 3.0 nezměnilo, TLS 1.0 má v těchto paketech hodnotu 3.1 a TLS 1.1 pak hodnotu 3.2, čili jsou brány jako updaty původního SSL. Vzhledem k provázanosti obou protokolů budu dále psát v názvu pouze TLS, které je aktuálnější, ale základní principy se dají vztáhnout i na SSL, vývoj se projevuje v opravě chyb a přidané funkcionalitě.

6.2.2 Základní principy

Hlavním úkolem TLS je ochrana před odposlechem dat, jejich padělání nebo neoprávněná změna při přenosu. Za tímto účelem je navržen tak, aby nebyl vázán na konkrétní protokol, ale zavádí se jako přidaná vrstva mezi konkrétní protokol v aplikační vrstvě a transportní vrstvu (viz Obr. 6.4).



Obr. 6.4: Umístění protokolu TLS ve zjednodušeném modelu TCP/IP, převzato z [67].

Protokol TLS se skládá z těchto dvou částí:

1) **Handshake Protocols** – sada protokolů pro vyjednání parametrů spojení mezi oběma stranami. Kromě samotných zpráv s parametry spojení obsahuje Alert protocol a Change Cipher Specification Protocol.
2) **Record Layer Protocol** – základní protokol TLS, zapouzdřuje pakety výše uvedených protokolů a stejně tak nakládá i s packety samotných aplikací, jejichž komunikace je takto zabezpečena. Spolu s Handshake protokolem tvoří podstatnou část TLS.

6.2.3 TLS Handshake Protocols

6.2.3.1 Alert Protocol

Sada zpráv, pomocí kterých lze upozornit druhou stranu, že vyvstal nějaký problém. Zprávy mohou mít dvě úrovně závažnosti:

- **fatal** – veškeré zprávy s touto úrovní závažnosti vedou k okamžitému ukončení relace. Příjem takové zprávy musí být následován okamžitým zneplatněním daného SessionID a zrušení všech dalších relací se stejným SessionID.
- **warning** – příjemce se může rozhodnout, zda takovouto zprávu bude ignorovat nebo na jejím základě ukončí spojení a postup bude stejný jako v předcházejícím případě.

Mimo tyto dva typy zpráv může kterákoli strana zaslat zprávu close_notify, která znamená ukončení relace, a že po odeslání této zprávy již budou v rámci dané relace ignorována veškerá příchozí data. Druhá strana musí odpovědět steným typem zprávy a uzavřít spojení.

Kompletní seznam chybových zpráv i s vysvětlením jejich významu a úrovní závažnosti lze nalézt v RFC 4346 [68]).

6.2.3.2 Change Cipher Specification Protocol

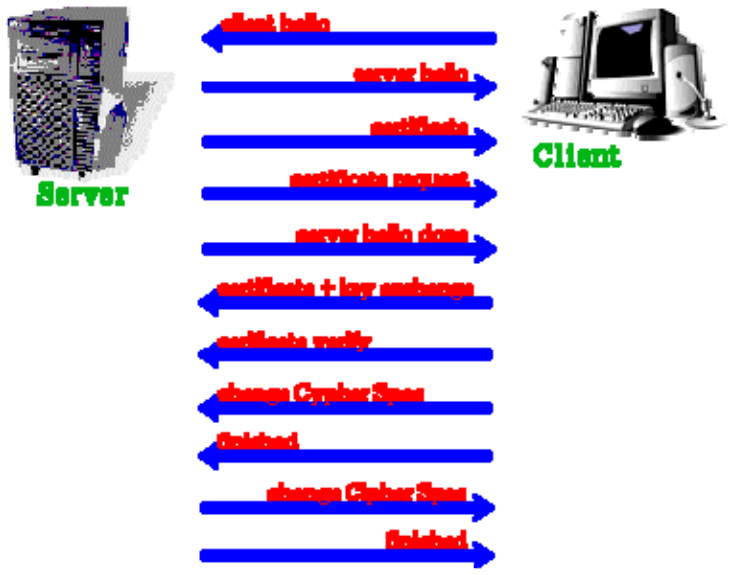
Protokol, který slouží jako signál příjemci, že odesílatel bude veškeré další zprávy vysílat dohodnutou šifrovací metodou s dohodnutým klíčem.

6.2.3.3 Handshake protocol

Někdy také zvaný jako key-exchange protocol, má na starosti ustanovení parametrů spojení, které jsou podporovány oběma komunikujícími stranami. Za tímto účelem při procesu vyjednávání proběhnou tyto kroky:

1. Autentizace serveru klientem
2. Předložení šifrovacích algoritmů podporovaných oběma stranami
3. Volitelná autentizace klienta serverem
4. Výměna parametrů spojení a výpočet klíče symetrické šifry (zašifrováno veřejným klíčem)
5. Zabezpečené spojení je navázáno

Na dalším obrázku (Obr. 6.5) je zobrazeno pořadí přenosu jednotlivých zpráv potřebných pro ustanovení spojení.



Obr. 6.5: Průběh použití Handshake protokolu, převzato z [64].

Zde je význam jednotlivých zpráv (podle [68]):

- **client_hello** – inicializační zpráva relace, klient zasílá serveru tyto parametry:
  - ***Version*** – nejvyšší verze TLS podporovaná klientem
  - ***Random*** – náhodná data

- ***SessionID*** – identifikátor relace
- ***CipherSuite*** – sada klientem podoporovaných šifer a algoritmů pro výměnu klíčů
- ***CompressionMethod*** – seznam klientem podporovaných kompresních algoritmů

- **sever_hello** – zpráva stejného formátu jako client_hello, server také podle SessionID zkontroluje, zda-li není již otevřené jiné spojení se stejným SessionID. Pokud ano, pak provede tzv. redukovaný handshake. V opačném případě založí nové SessionID a postupuje v provádění kompletního handshake, ze seznamu klientem podporovaných šifer jednu vybere, stejně jako algoritmus pro předání klíče této šifry a také metodu komprese.
- **server_certificate** – v další zprávě server zašle svůj certifikát.
- **server_key_exchange** – tato zpráva je vygenerována pouze pokud server nemá vlastní certifikát nebo certifikát neobsahuje dostatek informací pro to, aby klient mohl pokračovat ve výměně tzv. premaster klíč (hodnota, ze které obě strany vypočtou master klíč – mezikrok pro výpočet klíče symetrické šifry). Algoritmy používané pro výměnu klíče jsou RSA a Diffie–Hellman.
- **certificate_request** – volitelná zpráva, kterou server odešle v případě, že má nasteveno, aby autentizoval klienta.
- **server_hello_done** – oznámení serveru klientovi o splnění jeho počáteční části spojení a že server čeká na odpověď klienta.
- **client_certificate** – pokud server požádal klienta o autentizaci a klient má vlastní certifikát, pak je zaslán v této zprávě. V opačném případě je zaslána rovnou zpráva následující.
- **client_key_exchange** – tato zpráva musí okamžitě následovat zprávu client_certificate, pokud je zaslána. Pokud ne, tak to musí být první zpráva po přijeté server_hello_done. Obsah zprávy se dělí podle použití algoritmu RSA nebo D-H na tyto případy:
  - ***RSA*** – klient vygeneruje 48 bitový premaster klíč, který následně zašifruje veřejným klíčem serveru a odešle jej.
  - ***D*–H** – v případě, že klientův certifikát obsahuje parametry pro komunikaci pomocí algoritmu D–H, pak je zaslána prázdná zpráva, jinak jsou zaslány parametry pro výpočet premaster klíče.

- **certificate_verify** – zpráva sloužící pro verifikaci klientova certifikátu, pokud je zaslána, tak musí následovat bezprostředně za client_key_exchange.

V dalším kroku server i klient vypočtou master klíč na základě tohoto výpočtu [64]:

$$\text{master\_secret} = \text{MD5}(\text{pre\_master\_secret} + \text{SHA}(\text{A} + \text{pre\_master\_secret} + \text{ClientHello.random} + \text{ServerHello.random})) + \text{MD5}(\text{pre\_master\_secret} + \text{SHA}(\text{BB} + \text{pre\_master\_secret} + \text{ClientHello.random} + \text{ServerHello.random})) + \text{MD5}(\text{pre\_master\_secret} + \text{SHA}(\text{CCC} + \text{pre\_master\_secret} + \text{ClientHello.random} + \text{ServerHello.random})); \quad (6.1)$$

Po výpočtu klíče symetrické šifry je doporučeno zničit premaster klíč, aby případnému útočníkovi neposkytl informace použitelné k získání master klíče.

- **change_cipher_spec** – zpráva oznamující změnu parametrů šifrování.
- **finished** – první zpráva zašifrovaná novými parametry, slouží jako potvrzení, že autentizace proběhla úspěšně a parametry jsou správně nastaveny a musí následovat bezprostředně po change_cipher_spec. Po odeslání již jen daná strana očekává stejnou odpověď (tj. zprávu change_cipher_spec následovanou zprávou finished, již šifrovanou dohodnutou metodou).

Pokud obě strany přijaly korektně zašifrovanou zprávu finished, bezpečný kanál je ustanoven a komunikace může začít.

6.2.3.4 Redukovaný Handshake

Pokud sever podle SessionID v klientově zprávě client_hello nalezne již probíhající relaci s daným klientem, může provést redukovaný handshake. Ten spočívá ve vypuštění většiny kroků. Po počátečním client_hello následují pouze zprávy server_hello a obě strany zašlou change_ciper_spec a finished a parametry nastaví stejné, jaké má již probíhající spojení.

6.2.4 TLS Record Layer Protocol

Tento protokol zajišťuje samotné zapouzdření veškerých odesílaných packetů. Veškerá data, která přijme od vyšších aplikací dělí na fragmenty, se kterými dále pracuje – jeho úkolem je komprese dat, vytvoření MAC, šifrování a odeslání dat. Při příjmu data dešifruje, zkontroluje MAC, provede rozbalení a spojení fragmentů a výsledek předává příjemci z řad aplikačních protokolů.

Takto mohou být zpracovány čtyři různé typy zpráv:

- **handshake** – zpráva při vyjednávání parametrů spojení

- **alert** – zpráva s výstrahou
- **change_cipher_spec** – oznámení, že došlo ke změně v nastavení šifrování podle dohodnutých parametrů
- **application_data** – data daného protokolu, který komunikuje pomocí TLS

Existují čtyři možné stavy, ve kterých se může protokol nacházet. Tyto stavy definují použité kompresní a šifrovací algoritmy, algoritmus pro výpočet MAC, dále se zde nacházejí i hodnoty klíčů symetrických šifer a hodnota MAC. Stavy jsou rozlišené na aktuální a připravované, dále jsou také rozdělený kvůli různým klíčům pro odchozí a příchozí data:

- **aktuální čtecí stav** (z ang. current read state) – obsahuje parametry a klíče pro příchoz data.
- **aktuální zápisový stav** (z ang. current write state) – s parametry klíči pro odchozí data.
- **připravovaný čtecí stav** (z ang. pending read state) – parametry tohoto stavu jsou měněny během handshake algoritmu podle dohody.
- **připravovaný zápisový stav** (z ang. pending write state) – zápisový stav, který je připravován během handshake.

Jakmile jsou strany dohodnuty a dojde k výpočtu potřebných hodnot, je aplikován change_cipher_spec, který zruší aktuální stav a z připravovaných stavů vytvoří nové aktuální stavy. Připravované stavy jsou pak znovuinicializovány a nastaveny na žádnou kompresi, šifrování a MAC čekají na další handshake kdy dojde k jejich změně.

Samotný proces zpracování dat vypadá následovně:

1. **Fragmentace** – Data jsou rozdělena do fragmentů typu TLSPlaintext, které mohou mít velikost až 2^{14} bytů. Před tento blok je připojena hlavička o délce pět bytů, která obsahuje informace o typu zpávy (8 bitů), verzi TLS (16 bitů) a délce datové části (16 bitů).
2. **Komprese** – Pomocí algortimu dohodnutého v handshake fázi z původního TLSPlaintext vytvořena struktura TLSCompressed. Komprese musí být bezstrátová, pokud je při dekompresi nalezen fragment, který by po dekompresi měl větší velikost než 2^{14}, strana by měla ohlásit fatální chybu dekomprese a ukončit spojení.
3. **Výpočet MAC** – po kompresi je vypočítána kontrolního hodnota MAC. Použita je hašovací funkce domluvená při handshake a slouží k tomu tento výpočet:

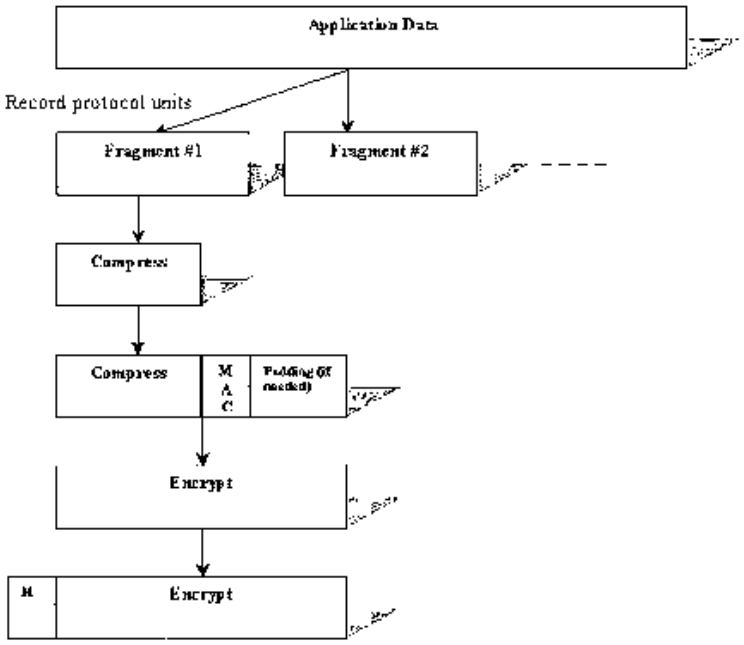
HMAC_hash (MAC_write_secret,seq_num + TLSCompressed.type +
TLSCompressed.version + TLSCompressed.length + TLSCompressed.fragment))
(6.2)

`MAC_write_secret` – tajná hodnota pro výpočet MAC.

`seq_num` – sekvenční číslo daného bloku, umožňuje detekci chybějících, přidaných nebo duplikovaných bloků.

4. **Šifrování** – TLS používá dvě možnosti šifrování – proudové a blokové (v tomto případě použity v CBC módu). V případě použití blokové šifry je potřeba doplnit velikost fragmentu na násobek délky bloku použité blokové šifry. Z fragmentu TLSCompressed se pak stává TLSCiphertext. Jednou ze změn, které přináší TLS 1.1 oproti TLS 1.0 a SSL všech verzí, je zavedení explicitního inicaliačního vektoru (IV), což bylo řešení slabiny, která poskytovala výhodu útočníkovi s možností chosen–plaintext útoku, blíže popsané v [69].

Na následujícím obrázku (Obr. 6.6) je dobře vidět proces zapouzdřování dat.



Obr. 6.6: Jednotlivé fáze zpracování dat před odesláním, převzato z [64].

V TLS je definován algoritmus pro výpočet všech potřebných klíčů pomocí master klíče, který byl vygenerován v průběhu handshake. Master klíč je zde použit jako jeden z parametrů pro vygenerování bloku bytů (tzv. klíčový blok), který je rozdělen na několik částí, z nichž každá představuje jeden klíč nebo tajemství. Algoritmus pro vygenerování klíčového bloku vypadá následovně:

key_block = PRF (SecurityParameters.master_secret, "key expansion",
SecurityParameters.server_random + SecurityParameters.client_random) (6.3)

Tento výpočet probíhá dokud není vygenerován dostatečně velký blok bytů, který by se dal rozdělit na tyto části:

client_write_MAC_secret[SecurityParameters.hash_size]
server_write_MAC_secret[SecurityParameters.hash_size]
client_write_key[SecurityParameters.key_material_length]
server_write_key[SecurityParameters.key_material_length]

Tyto klíče jsou pak použity při samotném šifrování dat.

6.2.5 Algoritmy použité v protokolu SSL/TLS

6.2.5.1 Hašovací algoritmy

Hašovací algoritmy jsou použity nejen při výpočtu MAC hodnot, ale také jsou obsažaženy ve formuli tvořící master klíč z premaster klíče a dalších parametrů.

- **MD5**
- **SHA-1**

V TLS je použita pro výpočet master klíče postupná aplikace obou algoritmů, aby byla posílena kryptografická síla takového výpočtu a byly co nejvíce potlačeny nevýhody jednotlivých algoritmů.

6.2.5.2 Algoritmy s veřejným klíčem

Pro výměnu premaster klíče mezi stranami používá TLS dva algoritmy:

- **RSA**
- **Diffie–Hellman**

Tyto algoritmy jsou při dostatečně dlouhém klíči považovány za bezpečné, ovšem je potřeba je ochránit proti útoku typu man-in-the-middle. Toho je dosaženo použitím certifikátů a důvěryhodných certifikačních autorit.

6.2.5.3 Symetrické šifry

V TLS se můžeme setkat se dvěma typy symetrických šifer – proudové a blokové. Výhodou proudových šifer je, že u nich není potřeba řešit doplňování dat na délku odpovídající násobku délky bloku použité šifry. Mezi blokovými šiframi je k dipozici větší výběr a hlavně jsou k dispozici bezpečnější algoritmy.

- **RC4** – jediný zástupce proudových šifer, vzhledem k existenci bezpečnějších algoritmů jeho použití nedoporučuji, její jedinou výhodou je rychlost, proto by byla pouze vhodná do prostředí, kde je rychlost šifrování kriticky důležitým parametrem.
- **DES** – v dnešní době zastaralá šifra, pro použití nevhodná.
- **3DES** – prodloužení klíče na trojnásobnou délku (tj. 168 bitů) je vykoupeno podstatným zpomalením šifrování.
- **IDEA** – šifra s lepším poměrem bezpečnosti a rychlosti šifrování než 3DES, ale bohužel není ve světě příliš rozšířená, takže ji mnoho implementací TLS nepodporuje.

- **AES** – jinak znám jako Rijndael, tento algoritmus v původním návrhu standartu TLS nebyl, ale byl přidán až po konci výběrového řízení na AES, jakožto náhradu již nedostatečně bezpečného DESu. Přidání AES je definováno v dokumentu [73]. Pro použití v TLS pracuje s délkou klíče 128 nebo 256 bitů a 128 bitovými bloky.

TLS má již přímo nastavené kombinace jednotlivých šifer, které budou při spojení použity. Jejich kompletní seznam uvádím v příloze B, zde pouze předložím dle mého názoru z hlediska bezpečnosti nejvhodnější variantu. Tou by měla být:

```
TLS_DH_DSS_WITH_AES_256_CBC_SHA
```

Význam jednotlivých zkratek je tento:

- **DH** – výměna premaster klíče probíhá pomocí algoritmu Diffie–Hellman.
- **DSS** – pro autentizaci jsou využity certifikáty podepsané algoritmem Digital Signature Standart (známý také jako Digital Signature Algorithm).
- **AES_256_CBC** – použitá symetrická šifra je AES s 256 bitovým klíčem v cipher block chaining módu.
- **SHA** – kontrola integrity je prováděna algoritmem SHA-1.

V případě, že klient a server použití této sady nepodporují, lze použít sadu, která je podle specifikace TLS povinná ve všech implementacích a tou je:

```
TLS_RSA_WITH_3DES_EDE_CBC_SHA
```

Výměna premaster klíče je provedena pomocí RSA a místo algoritmu AES je použit 3DES, také v cipher block chaining módu.

V souvislosti s nízkou bezpečností základního algoritmu DES a malým rozšířením algoritmu IDEA byl uveden dokument [72], který doporučuje vypuštění těchto dvou algoritmů ze stávajících implementací. V budoucí verzi TLS 1.2 se již s těmito algoritmy nepočítá.

6.2.6 Rozdíly mezi jednotlivými verzemi

V této kapitole proberu rozdíly, kterými se lišily jednotlivé verze protokolů SSL 2.0, SSL 3.0, TLS 1.0 a 1.1. Dřívější verze bezpečnostních protokolů (a softwaru obecně) vzniklých v USA byly poznamenány bezpečnostní politikou USA, která zamezovala export kryptograficky silných šifer na cizí území. Z těchto restrikcí pramení většina rizik a těží z nich také některé metody útoků. S uvolněním pravidel pro export šifer se mnoho problémů dočkalo řešení. Následující podkapitoly budou obsahovat vždy popis nových vlastností vyšší verze protokolu od té předchozí. Informace pro tuto kapitolu jsem čerpal převážně z [68] a [80].

6.2.6.1 Rizika SSL 2.0

- Při vyjednávání symetrické šifry může útočník nepozorovaně změnit seznam serverem podporovaných šifer tak, že na něm budou jen šifry se 40 bitovým klíčem.
- Konec spojení je indikován pouze přerušením TCP relace a tak může být vyvolán záměrně útočníkem.
- Před šifrováním nejsou data komprimována.
- V hlavičce record paketu je zbytečná informace o délce výplně, což poskytuje vodítka ke struktuře zasílaných dat.
- Pro šifrování i MAC je použit stejný klíč, při použití exportní šifry tak musí i MAC mít stejně krátký klíč (typicky 40 bitový) ačkoli to není nutné. SSL 2.0 používá pro autentizaci zprávy pouze MD5.

6.2.6.2 Porovnání SSL 2.0 a SSL 3.0

- SSL 3.0 přináší ochranu proti vynucení komunikace pomocí SSL 2.0, navzdory schopnosti obou stran použít SSL 3.0, což by útočníkovi přineslo jednodušší možnosti útoku. Děje se tak pomocí použití konkrétních hodnot pro prvních 8 bytů výplně bloku s daty pro tvorbu klíče. SSL 2.0 klienti tato data nepoužijí, takže se jich tato informace netýká, SSL 3.0 klienti tuto informaci zaznamenají a odmítnou komunikovat pomocí SSL 2.0.
- SSL 3.0 nabízí tři druhy spojení: s autentizací serveru, s autentizací serveru i klienta a anonymní. Anonymní způsob spojení je zranitelný útokem typu Man-in-the-middle, protože není žádná autorita, která by ověřila oprávněné vlastnictví RSA klíče daným serverem.
- Je zde také zavedena ochrana proti pozměnění zpráv při handshake procesu tak, že v závěrečné finish zprávě je zahrnut MAC všech předešlých zpráv včetně závěrečné. V

případě, že by tato hodnota nesouhlasila s hodnotou druhé strany, spojení bude uzavřeno.

- Ochrana proti útokům odseknutí dat – ve verzi 3.0 je přidána zpráva (closure_notify), která oznamuje konec spojení a tato událost již není signalizována pouze přerušením TCP spojení. Nečekané přerušení relace způsobí její neobnovitelnost.
- Od SSL 3.0 je vyžadována komprese dat.
- Přidána možnost výměny klíčů pomocí algortimu Diffie–Hellman.
- Při použití hašovacích funkcích ve výpočtech klíčů jsou použity obě metody – MD5 i SHA-1 pro zvýšení jejich kryptografické síly.

6.2.6.3 Porovnání SSL 3.0 a TLS 1.0

Změny již nejsou tak rozsáhlého charakteru jako v předchozím případě, proto je také také TLS 1.0 vedena jako minoritní upgrade (tj. z SSL 3.0 na SSL 3.1).

- Odlišný způsob výpočtu MAC.
- Mezi podporovanými sadami šifrovacích algoritmů musí být sada s algoritmy Diffie-Hellman, DSS a 3DES.
- Změny výpočtu klientského a serverového zápisového klíče.

6.2.6.4 Porovnání TSL 1.0 a TLS 1.1

- Po nečekaném ukončení relace již není vyžadován restart relace.
- Byla zrušena exportní omezení a exportní typy šifer a implementace je již nesmí podporovat.
- Formát certifikátu byl změněn na PKCS #1.
- Zavedení explicitního incializačního vektoru.
- Použitá chybová zpráva pro chyby ve výplni byla změněna z bad-record-mac na decryption-failed. Snaha o nevyzrazení útočníkovi potenciálně užitečných informací.
- Pro parametry spojení jsou definovány záznamy IANA [80].

6.2.7 Implementace SSL/TLS

Běžný uživatel se nejčastěji setká s implementací TLS zakomponovanou do internetového prohlížeče. Existenci bezpečného spojení HTTP protokolu může naznačit https místo obvyklého http v adrese stránky. Využití TLS se týká hlavně bankovních aplikací, platebních systémů. Další možností využití je ochrana poštovního protokolu SMTP – podrobné specifikace použití SMTP spolu s TLS jsou popsány v [74], specifikace HTTPS lze nalézt v [75].

Jednou z mála samostatných aplikací je Stunnel [71], který vytváří zašifrovaný tunel a veškerou komunikaci pro vybraný port zasílá tímto tunelem.

Další velmi zajímavou možností je vytvoření virtuální privátní sítě (VPN) pomocí TLS. Oproti VPN postaveným nad IPsec přináší VPN nad TLS možnost vytvoření připojení ke vzdálené VPN s minimálními prostředky – pro základní práci proveditelnou přes HTTP protokol uživateli stačí pouze počítač s internetovým prohlížečem, který podporuje SSL/TLS – takový se dá nalézt v každé internetové kavárně a jiných veřejných místech. K tomuto účelu se dají koupit i hotová hardwarová řešení VPN brán, které podporují přístup přes TLS, případně zprostředkovávají napojení VPN sítí nad TLS do již existující VPN nad IPsec infrastruktury.

Pro programátory je také volně k dispozici knihovna OpenSSL [76]), která zprostředkovává funkčnost SSL/TLS.

Mezi aplikace využívající zabezpečení pomocí SSL/TLS dále patří např.:

- **Apache-SSL** – webový server Apache s integrovanou podporou SSL/TLS. [77])
- **RaidenFTPD** – komerční FTP démon pro OS Windows. [78]
- **Samba** – volně dostupná implementace SMB/CIFS protokolu. [79]

6.2.8 SSL/TLS shrnutí

Protokol SSL/TLS je bezesporu důležitým prvkem při transportu důvěrných informací. Lze se s jeho využitím setkat i častěji, než s předešlým protokolem SSH. Naštěstí se dnes již příliš nesetkáme s verzí SSL 2.0, která se zdaleka nedá považovat za natolik bezpečnou, jako její nástupce v podobě SSL 3.0 a TLS 1.0 a 1.1. To je umocněno také tím, že nejpoužívanější webové prohlížeče mají v aktuálních verzích vypnutou podporu SSL 2.0, je to ochrana proti úrokům, při kterých útočník vnutí komunikujícím stranám použití nižší verze SSL, kterou může posléze snadněji napadnout.

Pro bezpečné použití TLS je, tam kde je to možné, vybírat šifrovací sady obsahující algoritmus AES. Pro výměnu klíče je vhodné použití Diffie-Hellman algoritmu vzhledem k útoku na spojení využívající algoritmus RSA, popsaném ve [104].

V dohledné době postoupí tento protokol do vyšší verze, TLS 1.2, na jehož specifikaci je již podán návrh. V této verzi přinese změny týkající se např. hašovacích funcí, hlavně použití kombinace MD5/SHA-1 v pseudonáhodné funkci bude nahrazeno pseudonáhodnou funkcí specifikovanou v šifrovací sadě, touto funkcí bude zřejmě SHA-256. Bude rozhodně zajímavé sledovat vývoj a nové funkce tohoto systému.

7 Útoky na kryptografické systémy

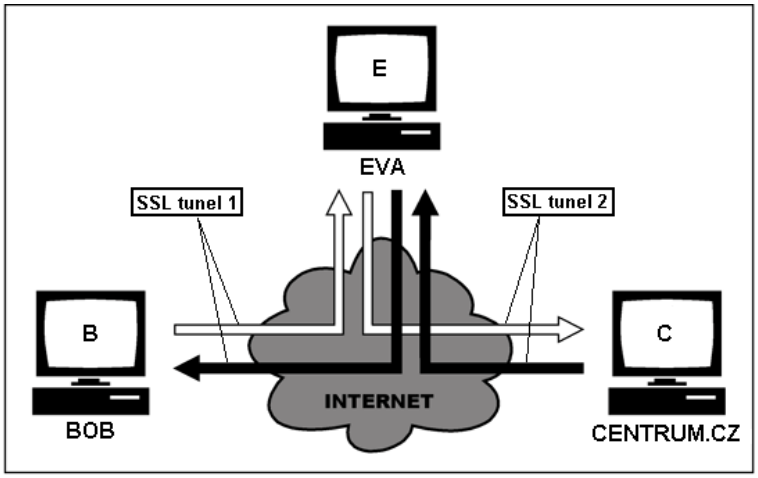
V kapitole 4 jsem se zabýval útoky na matematické jádro kryptografických algoritmů. V této části se zaměřím spíše na útoky na bezpečnostní systémy. Tyto útoky bývají založeny např. na špatném návrhu nebo vlastnostech síťových protokolů. Uvedu zde hlavně útoky týkající se protokolů SSH a SSL/TLS. Informace o specifičtějších verzích útoků jsem čerpal převážně z [68] a [87].

Man in the middle je název pro schéma situace, kdy se mezi klienta a server dostane útočník, přes kterého pak musí procházet všechna data. Z tohoto základního stavu vychází většina specitičtějších útoků, ale základní podstata je v tom, útočník v takovéto pozici má přístup k zasílaným datům, možnost je měnit, mazat nebo směrovat jinam, než byl původní záměr oběti.

Popis tohoto útoku lze provést na modelové situaci pro uživatele Boba a Alici a útočníka Evu. Bob a Alice si chtějí v nezabezpečeném prostředí vyměnit zprávu za pomocí kryptografie s veřejným klíčem. Eva má ovšem možnost zachytit všechny jejich zprávy, takže když si Bob a Alice navzájem zašlou své veřejné klíče, Eva je oba zachytí a uschová. Namísto toho oběma zašle svůj veřejný klíč. Poté když Bob zašle zprávu zašifrovanou Eviným veřejným klíčem (v domění, že komunikuje s Alicí a že šifroval jejím veřejným klíčem), Eva zprávu zachytí, dešifruje, uloží její kopii, zašifruje veřejným klíčem Alice a pošle dál. Takto by spolu Bob a Alice mohli komunikovat dál bez povšimnutí. Aby bylo možné této situaci zabránit, potřebují oba důvěryhodnou třetí stranu, která autoritativně potvrdí, že daný veřejný klíč je svázán s identitou daného uživatele. Touto stranou jsou v praxi certifikační autority.

7.1 Man in the middle attack

Pro ukázku jsem provedl útok Man-In-The-Middle za účelem získání přihlašovacích informací k účtu českého freemailového serveru Centrum.cz. Schéma útoku je na Obr. 7.1, kde lze vidět role jednotlivých entit vystupujících v tomto příkladu – útočníkem je Eva a klient Bob se chce přihlásit ke svému poštovnímu účtu.



Obr. 7.1: Schéma útoku.

Klient zde místo zamýšleného připojení na server centrum.cz navazuje nevědomky spojední s útočníkem, který jeho data dešifruje a vlastním ssl tunelem je zasílá na centrum.cz. V následujících odstavcích bude podrobně popsáno prostředí, ve kterém byl tento útok realizován a nástroje, které umožnily zachycení a analýzu datového toku.

Prostředí

Podmínky pro útok jsem vytvořil na dvou virtuálních strojích vytvořených v prostředí programu VMware Workstation. Tyto dva stroje simulovaly privátní síť, která byla připojena k internetu pomocí routeru, jehož funkci zde zastával hostitelský počítač. Ten zajišťoval NAT a vlastní připojení k internetu. Na obou vrituálních počítačích běžel operační systém Ubuntu 8.04 Beta.

Programové vybavení

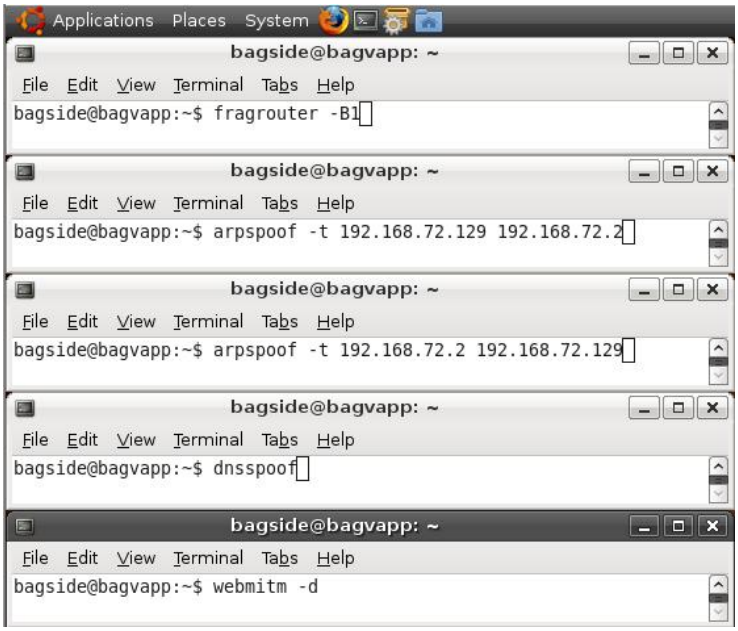
Klient pro svou práci potřebuje pouze internetový prohlížeč, tím byl v tomto případě Mozilla Firefox 3 Beta 4. Útočník již potřebuje ke své akci větší vybavení. První nástrojem použitým nástrojem je Wireshark 1.0.0. Tento program byl dříve znám jako Ethereal a jedná se o jeho přímého nástupce. Jeho úkolem je zachycení veškeré komunikace procházející síťovým rozhraním, které umožňuje ukládat pro pozdější zpracování.

Dále útočník potřebuje několik nástrojů, pro přeměrování klient ovy komunikace přes svoji stanici. Veškeré potřebné a některé další nástroje obsahuje balík programů snázvem dsniff [85]), jehož autorem je Dug Song. Funkci jednotlivých programů přiblížím až při jejich použití.

Posledním nástrojem, který útočník potřebuje je ssldump [86]), který výstup z Wiresharku upraví do čitelné podoby a v jeho výstupu pak může útočník nalézt hledané informace.

Postup

Veškeré následující úkony provádí útočník na své stanici, nepotřebuje fyzický přístup na klientův počítač. Podmínkou tohot útoku je ovšem fakt, že se klient i útočník budou nacházet ve stejné podsíti. Na následujícím obrázku (Obr. 7.2) je screen obrazovky útočníkova počítače, kde jsou vidět všechny nástroje potřebné pro přesměrování klientovy komunikace přes útočníkův počítač.



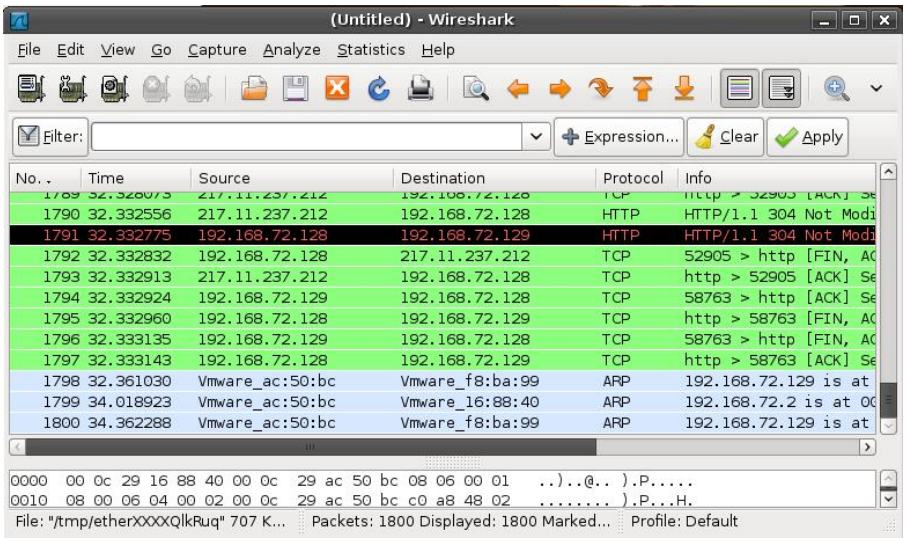
Obr. 7.2: Nástroje ze sady dsniff.

Jednotlivé nástroje mají tyto funkce:

- **fragrouter** – nástroj, který zajistí přeposílání dat (IP forwarding) z Evina počítače pro zajištění konektivity po arpspoofingu.
- **arpspoof** – tato utilita má na starosti padělání ARP odpovědí zasílaných v síti, jinými slovy v prvním kroku podstrčí útočník oběti informaci, že MAC adresa jejího počítače patří IP adrese brány, takže všechny klientovy zprávy směřující ven z jeho podsítě půjdou na útočníkův počítač. Druhý krok pro náš příklad není nutný, ale uvádím ho pro úplnost – ten přinutí bránu veškerá data zasílat na útočníkův počítač, tím kontroluje tok dat v obou směrech.
- **dnsspoof** – pracuje podobně jako arpspoof, ale týká se doménových názvů. DNS klient Bobova počítače, který pracuje nad UDP protokolem, vysílá dotazy pro zjištění IP adresy k danému DNS záznamu. Dnsspoof falšuje všechny odpovědi tak, že každý dotaz dostane odpověď s adresou útočníkova stroje.

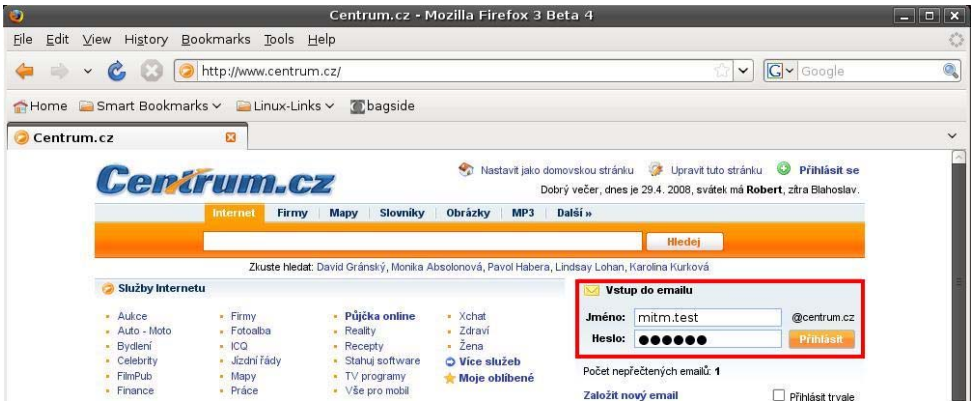
- **webmitm** – poslední nástroj pro aktivní ovlivnění dat. Při prvním spuštění útočník zadá informace potřebné k vytvoření falešného certifikátu. Tyto informace (včetně vytvořeného privátního klíče) uloží do souboru webmitm.crt, který bude mít pozdější využití. Pak má již webmitm na starosti komunikaci s SSL klientem počítače oběti a SSL serverem cílového serveru. Pomocí prvního SSL tunelu přijímá data od klienta, dešifruje je za použití vlastního privátního klíče a následně je druhým SSL tunelem zasílá cílovému serveru s platným certifikátem a analogicky k tomu pracuje v opačném směru tak, aby uživatel nezjistil, že se mezi ním a cílovým serverem někdo nachází. Jistým problémem je zde certifikát, který není podepsán žádnou certifikační autoritou, ale k tomuto se vrátím ještě později.

Těmito kroky si útočník přichystal síť ve které se nachází. Teď už jen potřebuje zachytit průchozí komunikaci programem Wireshark, jehož práce je dobře viditelná na Obr. 7.3.



Obr. 7.3: Aplikace Wireshark.

Útočníkovi teď zbývá jen počkat, až klient přijme jeho certifikát přihlásí do své emailové schránky.

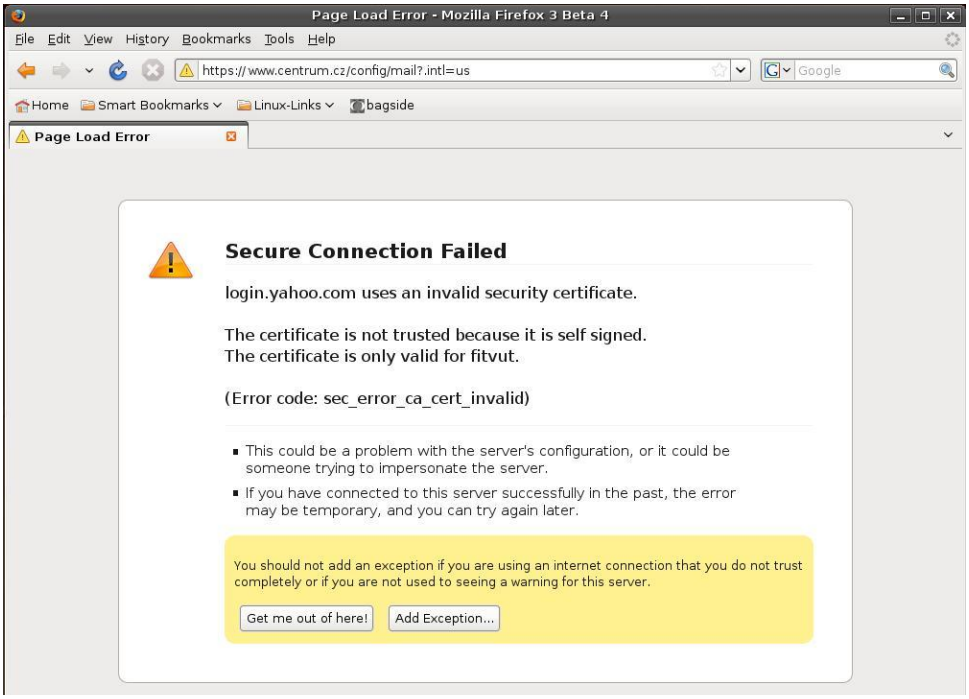


Obr. 7.4: Klientovo přihlašovací jméno.

Stiskem tlačítka přihlásit na Obr. 7.4 zasílá klient své přihlašovací jméno a heslo útočníkovi. Jakmile útočník ví nebo tuší, že se klient již přihlásil, může přerušit logování dat a získané informace uložit do souboru. Tento soubor pak použije jako vstupní data pro program ssldump, jako parametry určí dešifrování SSL komunikace a přidá i soubor webmitm.crt s privátním klíčem SSL tunelu vytvořeného mezi klientem a útočníkem. Výsledek si útočník může prohlédnout pomocí standartních linuxových nástrojů cat a grep, kterými může specifikovat, které řádky se mají pouze zobrazit, v tomto případě útočník hledá záznam pwd a u něj nalezne jméno **mitm.test** a heslo **fitvut**. Vše je vidět na Obr. 7.5.

Obr. 7.5: Získání údajů z odchycených dat.

Aby byl dobře vidět průběh útoku, úmyslně jsem přeskočil otázku certifikátů, ke kterým se teď vrátím. Prohlížeče mají totiž za úkol kontrolovat veškeré údaje certifikátů a v případě, že by některý z nich neseděl, musí varovat uživatele, že daná stránka nemá certifikát vpořádku. Dříve o tom pouze indikovala malá ikona, kterou nebyl problém přehlédnout, dnes je již informování o chybě certifikátu mnohem důraznější. Konkrétně použitý prohlížeč Mozilla Firefox 3 Beta 4 se snaží uživatele od použití takovéto stránky přímo odradit, viz Obr. 7.6 a 7.7.



Obr. 7.6: Oznámení o chybném certifikátu dané stránky.

Obr. 7.7: Dialog pro přijetí certifikátu.

Uživatel tak musí certifikát explicitně přijmout a nejde to provést jedním kliknutím, aby tak byl uživatel přinucen věnovat alespoň trochu pozornosti tomu, co se děje. Na Obr. 7.8 je pak samotný certifikát.

Obr. 7.8: Podvržený certifikát.

Z výše uvedeného je vidět, že bylo v poslední době věnováno hodně pozornosti důkladnému varování před podvrženými certifikáty. Existují pak tři možné scénáře:

- Útočník si dá práci s vytvořením opravdu věrohodně vypadajícího certifikátu, který uživatele přesvědčí, že komunikuje s důvěryhodnou stranou.
- Uživatel nedbá na bezpečnostní varování a všechny takové dialogy “odkliká”.
- Uživatel pojme podezření, že něco je v nepořádku a požadovanou službu použije jindy nebo v jiném prostředí.

Zde je vidět, že rozhodující je opět lidský faktor, takže mým doporučením je zdrženlivější chování v sítích, kde si člověk nemůže být jist úrovní bezpečnosti.

Systém SSH se tomuto útoku brání pomocí certifkačních autorit, případně databáze hostitelů a jejich klíčů, ovšem pokud se klient k připojuje k serveru poprvé, je tímto útokem zranitelný. Útok na SSL je podmíněn kooperací ze strany oběrti přijetím certifkátu.

7.2 Replay attack

Opět vychází z útoku Man in the middle, pro jeho provedení musí útočník zachytit nějaká validní data, např haš hesla, pomocí kterého uživatel potvrzuje svou identitu (takto se heslo kontroluje např. u autentizačního protokolu CHAP – Challenge Handshake Authentication Protocol [117]). Takovýto uložený haš hesla pak může útočník použít, když se snaží zfalšovat cizí identitu a server

nemá na výběr, než heslo přijmout. Jiným příkladem by bylo zachycení posloupnosti dat zasílané při bankovních převodech – útočník by tak mohl opakovaně provádět jednu platbu.

7.3 Connection hijacking

Přeložitelné do češtiny jako únos relace – útočník počká až proběhne autentizace a poté prakticky doslova ukradne TCP spojení. Tento útok ale využívá slabiny TCP, a protože SSH operuje nad TCP, tak proti únosu nemá žádné bezpečnostní mechanismy. Brání se proti této situaci kontrolou integrity zpráv, takže v případě, že by byly zaznamenány nějaké nesrovnalosti vzniklé změnou adresáta, došlo by k okamžitému uzavření relace. Jednou z metod connection hijacking je arpspoofing uvedený v příkladu.

7.4 Version rollback attack

Tento útok se týká SSH i SSL/TLS. Útočník se snaží využít chyb a vlastností nižší verze, proto ovlivňuje komunikaci tak, aby obě strany vybraly parametry odpovídající nižší verzi protokolu. V případě SSH může chtít útočník využít např slabou kontrolu integrity zpráv, v TLS se může snažit o použití protokolu SSL 2.0, kde by se dále mohl snažit vnutit stranám exportní 40bitové šifrování dat (které se již v aktuální verzi TLS nenálézá).

TLS se proti tomu brání specifickou výplní PKCS zpráv, které rozumí klienti verze SSL 3.0 a vyšší, takže pokud takovou zprávu dostanou, ví, že druhá strana je také SSL 3.0 nebo vyšší a odmítnou použít specifikace nižšího protokolu. Klient verze SSL 2.0 se výplní nezabývá.

Při pokusu o vnucení šifrovací sady, kterou by si jinak komunikující strany nezvolily (např. již zmíněné exportní sady) by musel útočník aktivně pozměnit jednu nebo více handshake zpráv. Tento fakt by byl ale zjištěn při zprávě finish, kdy by každý z účastníků vypočítal odlišné MAC hodnoty všech zaslaných a přijatých zpráv a vytváření spojení s danými parametry by selhalo.

Dalším vážný problém by nastal, kdyby se útočníkovi podařilo stranám komunikujícím pomocí SSL/TLS vnutit použití šifrovací sady TLS_NULL_WITH_NULL_NULL. V takovém případě by komunikace nebyla vůbec chráněna a útočníkovi by nic nebránilo veškerá zasílaná data odchytávat a číst.

7.5 Timing attack

Posledním útokem, který zmíním, je timing attack. Tento útok je postaven na monitorování časových úseků, mezi jednotlivými akcemi oběti. Jedním ze způsobů použití je hlídání odezvy napadaného serveru na jednotlivé dotazy. V praxi byl proveden pomocí dotazů na webový server Apache využívající SSL knihovnu OpenSSL.

Ve [104] David Brumley a Dan Boneh ukázali, že jsou schopni pomocí SSL tunelu získat timing útokem privátní klíč webového serveru. Celý útok probíhá v handshake fázi přípravy zabezpečeného spojení. Při standartní proceduře navazování spojení klient zašle zprávu `client-key-exchange`, ve které je veřejným klíčem serveru zašifrován premaster klíč. Pokud je formát dešifrované zprávy správný, server vypočte sdílenou hodnotu master klíče. Pokud mají zaslaná data nekorektní formát, serer vygeneruje vlastní náhodná data, ze kterých vytvoří master klíč, což ovšem vede k nerovnosti master klíčů komunikující stran a je to oznámeno zprávou `alert`.

Tento útok je realizován tak, že jsou serveru ve zprávě `client-key-exchange` zasílány vybrané hodnoty, které server dešifruje svým privátním klíčem, a podle délky času mezi odesláním `client-key-exchange` a přijetím `alert` je zjišťována doba dešifrování. Takto je hledán menší ze dvou dělitelů N asymetrické šifry RSA. Jakmile je tento dělitel nalezen, proběhne faktorizace čísla N a privátní klíč je prozrazen. Pro implementaci v OpenSSL jsou známy přesné charakteristiky dešifrování, které jsou také použity v tomto útoku a zájemce o hlubší studium této problematiky nalezne kompletní popis ve [104].

8 Závěr

V této diplomové práci jsem se zabýval aktuálními trendy v oblasti moderní kryptografie, s důrazem na použití symetrických a asymetrických algoritmů v praxi. Předvojem samotného porovnání algoritmů je část věnovaná modulární aritmetice, s jejíž využitím se můžeme v moderní kryptografii setkat. Ve stručnosti jsem zmínil podílové okruhy, použití operátoru mod, prvočísla, nerozložitelné polynomy a polynomy dělení kruhů a také jsem přiblížil princip funkce LFSR keystramů.

8.1 Šifrovací algoritmy

Před samotným výčtem algoritmů jsem popsal jejich dělení na symetrické, asymetrické a jako doplněk jsem probral několik vybraných hašovacích funkcí. Symetrické algoritmy se dále dají podle způsobu práce s daty dělit na blokové (kterých je aktuálně většina) a proudové. S tímto rozdělením jsem se začal zabývat vlastnostmi jednotlivých druhů šifer. Hlavně jsem probral jednotlivé módy blokových šifer, kde jsem také ukázal, že při použití Cipher Feedback a Output Feedback módu lze získat proudovou šifru za pomocí algoritmu blokové šifry. Tyto módy umožňují opuštění konceptu proudových šifer a zaměření se na tvorbu kvalitních blokových šifer.

Následně jsem prošel velké množství šifrovacích algoritmů, pro které jsem hledal informace o jejich bezpečnosti. Kompletní přehled všech popsaných algoritmů se nachází v příloze A, zde uvádím pouze algoritmy, o kterých jsem přesvědčen, že poskytují dostatečné zabezpečení. Podle mého názoru jsou pro další bezpečné použití vhodné tyto algoritmy:

- **DEAL** – výhodou tohoto algoritmu je pouze možnost jej implementovat na stávajících HW nebo SW řešeních určených pro DES, čímž může přinést úsporu nákladů, jinak díky své rychlosti neobstojí v konkurenci ostatních šifer.
- **GOST** – ruská odpověď na DES poskytuje velmi dobrou odolnost proti pokusům o kryptoanalýzu.
- **IDEA-NXT** – licenční podmínky bránící většímu rozšíření zřejmě z této kvalitní šifry dělají v očích kryptoanalytiků nezajímavý cíl. V kombinaci s dobrou odolností proti útokům dělá z šifry IDEA-NXT jeden ze mnou favorizovaných algoritmů.
- **MARS** – také opomíjený algoritmus, zatím byly úspěšné pouze útoky na značně redukované verze co do počtu provedených kol šifrování, takže ještě nabízí bezpečnostní rezervu.
- **RC6** – v tomto případě je obezřetnost na místě, zatím se podařilo prolomit 15 z 20 kol šifrování, ale prolomení všech 20 kol je stále mimo stávající možnosti.

- **Rijndael** – zde se opakuje situace z předchozího případu, ovšem tento stav je umocněn tím, že se jedná o standart AES, takže snahy o jeho prolomení budou jistě větší než např. snahy o prolomení šifry IDEA. Pro 128 bitový klíč bylo překonáno 7 z 10 kol.
- **Safer SK128** – po opravě výpočtu subklíčů tato šifra odolává pokusům o kryptoanalýzu.
- **Serpent** – má lepší výsledky co se odolnosti vůči kryptoanalýzy týče než Rijndael, který byl jeho soupeřem v boji o AES, za se podařilo překonat 10 kol z 32.
- **Skipjack** – opět velmi odolný proti kryptoanalýze, napadnutelný by byl při použití pouhých 16 kol, ovšem s 32 koly šifrování jej lze považovat za bezpečnou šifru.
- **Twofish** – poslední z doporučených algoritmů, zatím bezpečně odolává všem pokusům o prolomení.

V případě použití zde popsaných asymetrických šifer jsem nenalezl problémy, které by bránily jejich použití, proto mohu doporučit použití algoritmů **RSA**, **DSA** i **Diffie-Hellman**. Je ovšem třeba mít na paměti, že podstatnou část bezpečnosti nese také samotná implementace daného algoritmu. Jak jsem také uvedl v kapitole 7.5, díky specifickým vlastnostem implementace RSA v knihovně OpenSSL bylo možné tzv. timing útokem získat privátní klíč serveru.

U hašovacích algoritmů již situace zdaleka není tak jednoznačná. Bezkoliznost není natolik závažnou podmínkou pro použití ve funkcích generujících pseudonáhodné posloupnosti čísel, nicméně přesto se např. v příští verzi TLS s dalším využitím kombinace MD5/SHA-1 pro generování pseudonáhodných čísel nepočítá. Ovšem je to podmínka nutná při použití hašovacích algoritmů ve schématu digitálního podpisu. Možnost vytvoření dvou různých zpráv se stejnou haší podrývá celý koncept digitálního podpisu, protože ztrácíme záruku, že nikdo nevytvoří jinou zprávu, než kterou jsme vytvořili, se stejnou hodnotou haše. Na základě této podmínky nelze doporučit algoritmy MD5, RIPEMD a SHA-0. Pro tyto potřeby jsou zatím vhodné algoritmy:

- **SHA-2** – rozšířená verze algoritmu SHA-1, která přináší vyšší složitost útoku, ovšem vnitřní struktura algoritmu je velmi podobná, což by mohlo znamenat zneužitelnost stejných slabin jaké má SHA-1. Mezi SHA-2 jsou řazeny algoritmy SHA-224, SHA-256, SHA-384 a SHA-512.
- **RIPEMD160** – poměrně starý algoritmus, jehož neredukovaná verze zatím obstála ve všech pokusech o prolomení.

Dlouho jsem váhal, zda-li mám do doporučit i algoritmus SHA-1, jehož složitost nalezení kolize byla profesorkou Wangovou snížena z 2^{80} na 2^{69} (po dalších vylepšeních by složitost měla být snížena

dokonce na 2^{63}), což je sice na hranici praktické proveditelnosti, ovšem do budoucna tento algoritmus neslibuje potřebnou bezpečnost.

8.2 Útoky na algoritmy

V kapitole (cislo utoku) jsem představil několik metod útoků na samotné kryptografické algoritmy. Probral jsem dělení podle požadavků na možnosti útočníka, tj. chosen plaintext, chosen ciphertext apod. Dále jsem se zabýval již některými konkrétními metodami, které se aktuálně používají.

Mezi nejznámější patří Birthday attack, což je útok využívající narozeninového paradoxu, který snižuje složitost hledání kolizí hašovacích funkcí hrubou silou. Další, zřejmě nejrozšířenější metodou útoku, je diferenciální kryptoanalýza. Ta má několi různých variant, z nichž jsem si vybral bumerangový útok, který jsem podrobně prošel.

8.3 SSH

Další podstatnou částí mé práce byl rozbor protokolu SSH. Zde jsem probral použití výše zmíněných algoritmů do praxe, problémy s jejich implementací a následně jsem se zaměřil na bezpečnost uvedeného protokolu jako takového. Ze zjištěných poznatků jsem vytvořil několik doporučení, který by měla dopomoci pro bezpečné využití SSH:

- Provádění autentizace pomocí veřejných klíčů.
- Vhodnou symetrickou šifrou je Twofish, případně IDEA.
- Pro kontrolu integrity je vhodné použít minimálně SHA-1, pokud to implementace dovoluje tak RIPEMD-160.
- Nastavení serveru na striktní vyžadování SSH-2 protokolu a zakázání nepoužívaných prvků (např. jiné metody autentizace než jedna vybraná).
- Vypnutí možnosti interactive módu.

8.4 SSL/TLS

Podobně jako v případě SSH jsem se věnoval i dalšímu velmi rozšířenému protokolu, kterým je SSL/TLS. Opět jsem vytvořil několik doporučení, která se týkají použití algoritmů, vhodných verzí SSL/TLS a několik základních pravidel pro co nejbezpečnější využití:

- Vhodná šifrovací sada je TLS_DHE_DSS_WITH_AES_256_CBC_SHA.

- Důrazně nedoporučuji použití prázdné sady TLS_NULL_WITH_NULL_NULL, stejně jako sad, které neobsahující šifrování, autentizaci nebo kontrolu integrity, tím ztrácí použití TLS smysl.
- Rovněž bych se vyvaroval použití algoritmu RSA, protože byly předvedeny praktické útoky na získání privátního klíče serveru (popsán v kapitole 7.5 a podrobněji ve [104]).
- Aktuálně používaná verze by měla být minimálně SSL 3.0 a vyšší (tj. TLS 1.0 a 1.1), SSL 2.0 obsahuje vážné nedostatky popsané v kapitole 6.2.6.
- Na uživatele musí být kladeny požadavky na opatrnost při práci s certifikáty, jinak může snadno dojít k útoku typu man-in-the-middle jak je popsáno v kapitole 7.1.
- Jako doplněk lze TLS využít při vytváření VPN sítí.

8.5 Praktický útok

V kapitole 7 se věnuji praktickým útokům na bezpečnostní aplikace a protokoly. Pro co největší názornost jsem jeden z těchto útoků realizoval a spolu s podrobným popisem jsem ukázal jeho možnosti a kritická místa. Jedná se o útok typu Man In The Middle, který je vpodstatě základem pro většinu dalších útoků, takže jejich popis se více soustředí na jejich specifické vlastnosti.

Provedný útok je tzv. typu Man In The Middle a ukazuje schéma, kdy útočník, který se nachází ve stejné podsítí jako jeho oběť, změní arp záznamy oběti tak, aby veškerou komunikaci směřovala na útočníka místo brány sítě. Útočník pak předstírá, že je server, se kterým chtěla oběť navázat spojení. K tomu používá podvržené certifikáty, což je jediná možnost oběti, jak tento útok identifikovat. Zde řeším obecný problém, kdy běžní uživatelé bezhlavě příjmají bez bližšího prozkoumání jakékoli certifikáty, se kterými přijdou do styku. Následně se vytvoří dvě šifrovaná spojení – první směřuje od oběti k útočníkovi a druhé od útočníka k požadovanému server. V tuto chvíli má útočník k dispozici v nešifrované podobě veškerá data, která oběť odesílá a za pomoci určitých nástrojů může tato zpracovat a prohledávat, dokud nenalezne užitečné informace.

Ostatní popsané útoky rovněž vycházejí z pozice útočníka, který může na dané síti odposlouchávat komunikaci, takže už pouze uvádím jejich specifické vlastnosti, kterými se liší od Man-In-The-Middle-útoku.

8.6 Očekávaný vývoj

V souvislosti s bezpečností se nedá očekávat jakýkoli útlum vývoje, stále budou vznikat nové šifry a stále budou snahy o jejich prolomení. V nejbližší budoucnosti bude zřejmě zajímavé sledovat vývoj hašovacího algoritmu SHA-3, na nějž začne výběrové řízení v říjnu 2008. Vzhledem k potenciální

budoucí zranitelnosti i rodiny algoritmů SHA-2 (ta se bude odvíjet od výpočetní síly počítačů), tak bude zajímavé sledovat s jakými změnami přijdou kandidáti na šifru SHA-3.

Ve druhé polovině roku 2008 lze také očekávat novou verzi TSL 1.2, ovšem nepředpokládám, že by narozdíl od SHA-3 přinesla dramatické změny, spíše půjde o obměnu nepoužívaných či nedostatečně bezpečných algoritmů za novější a bezpečnější varianty (nahrazení šifer DES a IDEA standartem AES) a úpravy hašovacích funkcí.

Literatura

[1] BULANT, Michal. Kryptografie a některé její aplikace. In Sborník SLT 2001 . 2001. vyd. Brno : Konvoj, 2006. s. 169-177. [cit. 2008-05-14]. Dostupný z WWW: http://www.cstug.cz/slt/01/plne_texty/17.pdf. ISBN 8073020092.

[2] SERGIENKO, Alexander V. Quantum Communications and Cryptography. CRC press. 2005. 248 s. ISBN 0849336848.

[3] SINGH, Simon. Kniha kódů a šifer. Praha: Dokořán. 2003. 384 s. ISBN 80-86569-18-7.

[4] The Breakthrough of Frequency Analysis [online]. [2000] [cit. 2008-05-14]. Dostupný z WWW: http://cs-exhibitions.uni-klu.ac.at/index.php?id=279.

[5] SALES, Tony. The Principle of the Enigma [online]. [1995] [cit. 2008-05-14]. Dostupný z WWW: http://www.codesandciphers.org.uk/enigma/enigma1.htm.

[6] VONDRUŠKA, Pavel. Absolutně bezpečný systém – Vernamova šifra. Crypto-World [online]. 2001, ročník 3., č. 10 [cit. 2008-05-14], s. 3-4.

[7] ROBACK, Ed, SMID, Miles. Developing the Advanced Encryption Standard [online]. National Institute of Standards and Technology, U.S. Department of Commerce. 1999 [cit. 2008-05-14]. Dostupný z WWW: http://jya.com/aes-rsa99.htm.

[8] KNUDSEN, Lars. DEAL: A 128-bit block cipher : Technical Report 151 [online]. University of Bergen, Department of Informatics , 1998 [cit. 2008-05-14]. Dostupný z WWW: http://ii.uib.no/~larsr/papers/deal.ps.

[9] FEAL. Encyclopedia of Cryptography and Security [online]. Springer US. c2005. s. 219-220. ISBN 978-0-387-23473-1.

[10] LAI, X., MASSEY, J. A proposal for a New Block Encryption Standard [online]. Advances in Cryptology--- EUROCRYPT '90 Proceedings. Berlin: Springer-Verlag. 1991. s. 389-404. Dostupný z WWW: http://citeseer.ist.psu.edu/lai91proposal.html

[11] DAEMEN, J., GOVAERTS, R., VANDEWALLE, J. Weak keys for IDEA. Advances in Cryptology - Crypto '93. Springer-Verlag. 1994. s. 224-231.

[12] Cryptographic Algorithms [online]. 2003 [cit. 2008-05-14]. Dostupný z WWW: http://www.kremlinencrypt.com/algorithms.htm.

[13] BLAZE, M., SCHENIER, B. The MacGuffin block cipher algorithm [online]. Fast Software Encryption, volume 1008 of Lecture Notes in Computer Science. Springer Verlag. 1995. s. 97-110. [cit. 2008-05-14]. Dostupný z WWW: http://citeseer.ist.psu.edu/blaze95macguffin.html.

[14] WAGNER, D. The Security of MacGuffin [online]. 1995. [cit. 2008-05-14]. Dostupný z WWW: http://citeseer.ist.psu.edu/103809.html.

[15] BURWICK, C., COPPERSMITH, D., D'AVIGNON, E., et al. MARS - A Candidate Cipher for AES [online]. NIST AES Proposal. 1998. [cit. 2008-05-14]. Dostupný z WWW: http://citeseer.ist.psu.edu/burwick99mars.html.

[16] KELSEY, J., SCHNEIER, B. MARS Attacks!. Preliminary Cryptanalysis of Reduced-Round MARS Variants [online]. 2000. [cit. 2008-05-14]. Dostupný z WWW: http://www.schneier.com/paper-mars-attacks.ps.gz.

[17] KNUDSEN, L.R. , RIJMEN, V., RIVEST, R.L., ROBSHAW, M.P.J. On the design and security of RC2 [online]. Fast Software Encryption, Fift International Workshop. Springer Verlag. 1998. s. 206 - 221. [cit. 2008-05-14]. Dostupný z WWW: http://citeseer.ist.psu.edu/knudsen98design.html.

[18] KNUDSEN, L. R., MEIER, W., Improved Differential Attacks on RC5 [online]. 1996. [cit. 2008-05-14]. Dostupný z WWW: ftp://ftp.esat.kuleuven.ac.be/pub/COSIC/knudsen/rc5.ps.Z

[19] KNUDSEN, L.R., MEIER, W. Correlations in RC6 [online]. Preprint. 2000. [cit. 2008-05-14]. Dostupný z WWW: http://citeseer.ist.psu.edu/article/knudsen00correlation.html.

[20] SHIRRIFF, K. Differential Cryptanalysis of REDOC III [online]. [cit. 2008-05-14]. Dostupný z WWW: http://citeseer.ist.psu.edu/67768.html.

[21] DAEMEN, J., RIJMEN, V. AES Proposal: Rijndael [online]. 1998. [cit. 2008-05-14]. Dostupný z WWW: http://citeseer.ist.psu.edu/daemen98aes.html.

[22] BIHAM, E., DUNKELMAN, O., KELLER, N. The Rectangle Attack - Rectangling the Serpent [online]. Advances in Cryptology, Eurocrypt'01. 2001. [cit. 2008-05-14]. Dostupný z WWW: http://citeseer.ist.psu.edu/biham01rectangle.html.

[23] DAEMEN, J., RIJMEN, V., KNUDSEN, L. The block cipher Square [online]. Fast Software Encryption, volume 1267 of Lecture Notes in Computer Science. Springer-Verlag. 1997. s. 149-165. [cit. 2008-05-14]. Dostupný z WWW: http://citeseer.ist.psu.edu/daemen97block.html.

[24] KNUDSEN, L. R., ROBSHAW, M.J.B., WAGNER, D. Truncated differentials and skipjack [online]. Advances in Cryptology - CRYPTO'99, volume 1666 of LNCS. SpringerVerlag. 1999. s. 165-180. [cit. 2008-05-14]. Dostupný z WWW: http://citeseer.ist.psu.edu/knudsen99truncated.html.

[25] WAGNER, D., SCHNEIER, B., KELSEY, J. Cryptanalysis of ORYX [online]. 1997. [cit. 2008-05-14]. Dostupný z WWW: http://citeseer.ist.psu.edu/article/wagner97cryptanalysis.html

[26] RC4 [online]. [cit. 2008-05-14]. Dostupný z WWW: http://www.wisdom.weizmann.ac.il/~itsik/RC4/rc4.html

[27] FLUHRER, S. Cryptanalysis of the SEAL 3.0 pseudorandom function family. Fast Software Encryption Workshop (FSE'01). 2001. s. 135-143.

[28] Prime Number Hide-and-Seek: How the RSA Cipher Works [online]. [cit. 2008-05-14]. [cit. 2008-05-14]. Dostupný z WWW: http://www.muppetlabs.com/~breadbox/txt/rsa.html.

[29] MAURER, U., WOLF, S. The Diffie-Hellman protocol [online]. Designs, Codes, and Cryptography 19. KluwerAcademic Publishers. 2000. s. 147-171. [cit. 2008-05-14]. Dostupný z WWW: http://citeseer.ist.psu.edu/maurer99diffiehellman.html.

[30] Zip Password Recovery Key program [online]. [cit. 2008-05-14]. Dostupný z WWW: http://www.lostpassword.com/zip.htm.

[31] The RSA Laboratories Secret-Key Challenge [online]. [cit. 2008-05-14]. Dostupný z WWW: http://www.rsa.com/rsalabs/node.asp?id=2103.

[32] The RSA Laboratories DES Challenge III [online]. [cit. 2008-05-14]. Dostupný z WWW: http://www.rsa.com/rsalabs/node.asp?id=2108.

[33] Distributed.Net and EFF Win RSA's DES Challenge III [online]. [cit. 2008-05-14]. Dostupný z WWW: http://w2.eff.org/Privacy/Crypto/Crypto_misc/DESCracker/HTML/19990119_deschallenge3.html

[34] WRIGHT, E. L. Age of the Universe [online]. [cit. 2008-05-14]. Dostupný z WWW: http://www.astro.ucla.edu/~wright/age.html.

[35] Bletchleypark.net - Cryptanalysis [online]. [cit. 2008-05-14]. Dostupný z WWW: http://www.bletchleypark.net/cryptology/cryptanalysis.html.

[36] ARBAUGH, W. A. An Inductive Chosen Plaintext Attack against WEP/WEP2 [online]. 2001. [cit. 2008-05-14]. Dostupný z WWW: http://www.cs.umd.edu/~waa/attack/v3dcmnt.htm.

[37] FLUHRER S. R., MANTIN I., SHAMIR A. Weaknesses in the key scheduling algorithm of RC4. Selected Areas in Cryptography 2001, volume 2259 of Lecture Notes in Computer Science. Springer. 2001. s. 1-24.

[38] PYSHKIN A., TEWS E., WEINMANN R. P. Breaking 104 bit WEP in less than 60 seconds [online]. [cit. 2008-05-14]. Dostupný z WWW: http://eprint.iacr.org/2007/120.pdf.

[39] LORENC M. Kolize hašovacích funkci a narozeninový paradox. 2007 [online]. [cit. 2008-05-14]. Dostupný z WWW: http://www.isvs.cz/e-podpis-podatelny/kolize-hašovacich-funkci-a-narozeninovy-paradox-25-dil.html.

[40] BIHAM, E., SHAMIR, A. Differential Cryptanalysis of DES-like Cryptosystems [online]. Journal of Cryptology, Vol. 4 No. 1 1991. [cit. 2008-05-14]. Dostupný z WWW: http://citeseer.ist.psu.edu/biham91differential.html.

[41] Differential and Linear Cryptanalysis [online]. [cit. 2008-05-14]. Dostupný z WWW: http://www.quadibloc.com/crypto/co040501.htm.

[42] KNUDSEN, L. R., WAGNER, D. Integral cryptanalysis [online]. 2001. [cit. 2008-05-14]. Dostupný z WWW: http://citeseer.ist.psu.edu/knudsen01integral.html.

[43] KOHEL, D. R. Skripta kurzu Elementary Cryptography and Protocols Univerzity v Sydney, kapitola Modes of Operation [online]. [cit. 2008-05-14]. Dostupné z WWW: http://www.maths.usyd.edu.au/u/kohel/tch/MATH3024/Lectures/lectures_05.pdf

[44] Teorie blokových šifer [online]. [cit. 2008-05-14]. Dostupný z WWW: http://moon.felk.cvut.cz/~pjv/Jak/_info/i677/html/BlockCipher.htm.

[45] WAGNER, D. The Boomerang Attack. Fast Software Encryption - 6th International Workshop FSE'99. Springer-Verlag. 1999. s.156-170.

[46] KOHEL, D. R. Skripta kurzu Elementary Cryptography and Protocols Univerzity v Sydney, kapitola Modular Arithmetic [online]. [cit. 2008-05-14]. Dostupné z WWW: http://www.maths.usyd.edu.au/u/kohel/tch/MATH3024/Lectures/lectures_07.pdf

[47] RFC 4251. The Secure Shell (SSH) Protocol architecture [online]. 2006. [cit. 2008-05-14]. Dostupný z WWW: http://www.ietf.org/rfc/rfc4251.txt.

[48] RFC 4252. The Secure Shell (SSH) Authentication Protocol [online]. 2006. [cit. 2008-05-14]. Dostupný z WWW: http://www.ietf.org/rfc/rfc4252.txt.

[49] RFC 4253. The Secure Shell (SSH) Transport Layer Protocol [online]. 2006. [cit. 2008-05-14]. Dostupný z WWW: http://www.ietf.org/rfc/rfc4253.txt.

[50] RFC 4254. The Secure Shell (SSH) Connection Protocol [online]. 2006. [cit. 2008-05-14]. Dostupný z WWW: http://www.ietf.org/rfc/rfc4254.txt.

[51] BARRETT, D. J., SILVERMAN, R. E. SSH, The Secure Shell: The Definitive Guide [online]. 2001. ISBN 10: 0-596-00011-1. [cit. 2008-05-14]. Dostupný z WWW: http://www.unix.org.ua/orelly/networking_2ndEd/ssh/index.htm.

[52] HENDERSON, R. W., BLANKENBAKER, P. VPN: PPP Tunneled Over SSH Overhead Discussion [online]. 2003. [cit. 2008-05-14]. Dostupný z WWW: http://nst.sourceforge.net/nst/docs/user/ch07s05.html.

[53] RFC 4120. The Kerberos Network Authentication Service (V5)s [online]. 2005. [cit. 2008-05-14]. Dostupný z WWW: http://www.ietf.org/rfc/rfc4120.txt.

[54] How the Kerberos authentication work [online]. 2008. [cit. 2008-05-14]. Dostupný z WWW: http://learn-networking.com/network-security/how-kerberos-authentication-works.

[55] SSH Secure Shell for Servers Version 3.2.9 Administrator's Guide [online]. 2003. [cit. 2008-05-14]. Dostupný z WWW: http://www.ssh.com/support/documentation/online/ssh/adminguide/32/index.html

[56] SSH Tectia Client and SSH Tectia Server [online]. [cit. 2008-05-14]. Dostupný z WWW: http://www.ssh.com/products/client-server/.

[57] SecureCRT [online]. [cit. 2008-05-14]. Dostupný z WWW: http://www.vandyke.com/products/securecrt/index.html.

[58] ZOC • A Powerful Terminal Emulator and Telnet/SSH Client [online]. [cit. 2008-05-14]. Dostupný z WWW: http://www.emtec.com/zoc/index.html.

[59] OpenSSH [online]. [cit. 2008-05-14]. Dostupný z WWW: http://www.openssh.org/.

[60] PuTTY [online]. [cit. 2008-05-14]. Dostupný z WWW: http://www.chiark.greenend.org.uk/~sgtatham/putty/.

[61] portalPuTTY [online]. [cit. 2008-05-14]. Dostupný z WWW: http://socialistsushi.com/2005/11/17/portaputty.

[62] Patentový úřad USA. Patent č. 5657390. [cit. 2008-05-14]. Dostupný z WWW: http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PALL&p=1&u=%2Fnetahtml%2FPTO%2Fsrchnum.htm&r=1&f=G&l=50&s1=5657390.PN.&OS=PN/5657390&RS=PN/5657390.

[63] The Transport Layer Security (TLS) Protocol Version 1.2 [online]. 2008. [cit. 2008-05-14]. Dostupný z WWW: http://www.ietf.org/internet-drafts/draft-ietf-tls-rfc4346-bis-10.txt.

[64] HALABI, N., YUROVITSKY, B., KARIDI, L. SSL Tutorial [online]. 2001. [cit. 2008-05-14]. Dostupný z WWW: http://www2.rad.com/networks/2001/ssl/index.htm

[65] DIERKS, T., RESCORLA, E. Standards Track: The Transport Layer Security (TLS) Protocol Version 1.1 [online]. 2006. [cit. 2008-05-14]. Dostupný z WWW: http://www.ietf.org/rfc/rfc4346.txt.

[66] ODVÁRKA, P. SSL protokol [online]. [cit. 2008-05-14]. Dostupný z WWW: http://www.svetsiti.cz/view.asp?rubrika=Tutorialy&temaID=171&clanekID=187.

[67] KAČMAŘÍK, V. Výuková podpora předmětu Internetové technologie [online]. [cit. 2008-05-14]. Dostupný z WWW: http://linux456.vsb.cz/~tur038/int/diplomky/kacmarcik/index.html.

[68] The Transport Layer Security (TLS) Protocol Version 1.1 [online]. 2006. [cit. 2008-05-14]. Dostupný z WWW: http://www.ietf.org/rfc/rfc4346.txt.

[69] MOELLER, B. Security of CBC Ciphersuites in SSL/TLS: Problems and Countermeasures [online]. 2004. [cit. 2008-05-14]. Dostupný z WWW: http://www.openssl.org/~bodo/tls-cbc.txt.

[70] STEVENS, M. Fast Collision Attack On MD5. IACR ePrint archive Report [online]. 2006. [cit. 2008-05-14]. Dostupný z WWW: http://eprint.iacr.org/2006/104.pdf.

[71] Stunnel [online]. [cit. 2008-05-14]. Dostupný z WWW: http://stunnel.mirt.net/index.html.

[72] DES and IDEA Cipher Suites for Transport Layer Security (TLS) [online]. 2008. [cit. 2008-05-14]. Dostupný z WWW: http://www.ietf.org/internet-drafts/draft-ietf-tls-des-idea-01.txt.

[73] RFC 3268. Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS) [online]. 2002. [cit. 2008-05-14]. Dostupný z WWW: http://www.ietf.org/rfc/rfc3268.txt

[74] RFC 3207. SMTP Service Extension for Secure SMTP over Transport Layer Security [online]. 2002. [cit. 2008-05-14]. Dostupný z WWW: http://www.ietf.org/rfc/rfc3207.txt

[75] RFC 2818. HTTP Over TLS [online]. 2000. [cit. 2008-05-14]. Dostupný z WWW: http://www.ietf.org/rfc/rfc2818.txt.

[76] OpenSSL [online]. [cit. 2008-05-14]. Dostupný z WWW: http://www.openssl.org/.

[77] Apache-SSL [online]. [cit. 2008-05-14]. Dostupný z WWW: http://www.apache-ssl.org/.

[78] RaidenFTPD [online]. [cit. 2008-05-14]. Dostupný z WWW: http://www.raidenftpd.com/en/.

[79] Samba [online]. [cit. 2008-05-14]. Dostupný z WWW: http://us3.samba.org/.

[80] WEITH, L. Differences Between SSLv2, SSLv3, and TLS [online]. 2006. [cit. 2008-05-14]. Dostupný z WWW: http://www.yaksman.org/~lweith/ssl.pdf.

[81] Digital Signature Standard FIPS 186 [online]. 1994. [cit. 2008-05-14]. Dostupný z WWW: http://www.itl.nist.gov/fipspubs/fip186.htm.

[82] Draft for the third revision to the official DSA specification [online]. 2006. [cit. 2008-05-14]. Dostupný z WWW: http://csrc.nist.gov/publications/drafts/fips_186-3/Draft-FIPS-186-3%20_March2006.pdf.

[83] KLÍMA, V., ROSA, T. Achillova pata DSA [online]. Časopis Sdělovací technika, ročník 2005, číslo 22. [cit. 2008-05-14]. Dostupný z WWW: http://crypto-world.info/klima/2005/ST_2005_05_16_17.pdf.

[84] NGUYEN, P.-Q., SHPARLINSKI, I.-E. The Insecurity of the Digital Signature Algorithm with Partially Known Nonces [online]. Journal of Cryptology, Vol. 15, No. 3. Springer-Verlag. 2002. s. 151-176. [cit. 2008-05-14]. Dostupný z WWW: http://www.springerlink.com/content/1d2y9e47xp6a8ehf/fulltext.pdf.

[85] dsniff [online]. [cit. 2008-05-14]. Dostupný z WWW:http://monkey.org/~dugsong/dsniff/.

[86] ssldump [online]. [cit. 2008-05-14]. Dostupný z WWW:http://www.rtfm.com/ssldump/.

[87] Certiguide to security [online]. kapitola 1.4 Attacks. [cit. 2008-05-14]. Dostupný z WWW:http://www.certiguide.com/secplus/cg_sp_14Attacks.htm.

[88] Teorie symetrické a asymetrické kryptografie [online]. [cit. 2008-05-14]. Dostupný z WWW: http://www.ica.cz/home_cs/?acc=teorie_symetricke_a_asymetricke_kryptografie.

[89] VONDRUŠKA, P. Prolomení hašovacích funkcí MD5 a SHA1 a praktický dopad [online]. [cit. 2008-05-14]. Dostupný z WWW: http://crypto-world.info/vondruska/prezentace/gigacon_23_11_05.ppt.

[90] KLÍMA, V. Hašovací funkce, principy, příklady a kolize [online]. 2005. [cit. 2008-05-14]. Dostupný z WWW: http://cryptography.hyperlink.cz/2005/cryptofest_2005.htm.

[91] RFC 1321. The MD5 Message-Digest Algorithm [online]. 1992. [cit. 2008-05-14]. Dostupný z WWW: http://www.ietf.org/rfc/rfc1321.txt.

[92] DOBBERTIN, H. Cryptanalysis of MD5 compress [online]. 1996. [cit. 2008-05-14]. Dostupný z WWW: http://citeseer.ist.psu.edu/article/dobbertin96cryptanalysis.html.

[93] WANG, X., FENG, D, LAI, X., YU, H. Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD [online]. Rump session. Crypto 2004. [cit. 2008-05-14]. Dostupný z WWW: http://eprint.iacr.org/2004/199.pdf.

[94] KLÍMA, V. Tunnels in Hash Functions: MD5 Collisions Within a Minute [online]. 2006. [cit. 2008-05-14]. Dostupný z WWW: http://cryptography.hyperlink.cz/MD5_collisions.html.

[95] RIPEMD-160 [online]. [cit. 2008-05-14]. Dostupný z WWW: http://homes.esat.kuleuven.be/~bosselae/ripemd160.html.

[96] DOBBERTIN, H., BOSSELAERS, A., PRENEEL, B. RIPEMD-160, a strengthened version of RIPEMD [online]. 1996. [cit. 2008-05-14]. Dostupný z WWW: http://www.esat.kuleuven.ac.be/~cosicart/pdf/AB-9601/AB-9601.pdf.

[97] SHA-0 [online]. 1993. [cit. 2008-05-14]. Dostupný z WWW: http://security.isu.edu/pdf/fips180.pdf.

[98] RFC 3174. US Secure Hash Algorithm 1 (SHA1) [online]. 2001. [cit. 2008-05-14]. Dostupný z WWW: http://tools.ietf.org/html/rfc3174.

[99] WANG, X., YU, H., YIN Y. L. Efficient Collision Search Attacks on SHA-0 [online]. CRYPTO 2005. [cit. 2008-05-14]. Dostupný z WWW: http://www.cs.cmu.edu/~dbrumley/srg/spring06/sha-0.pdf.

[100 WANG, X., YU, H., YIN Y. L. Finding Collisions in the Full SHA-1 [online]. CRYPTO 2005. [cit. 2008-05-14]. Dostupný z WWW: http://people.csail.mit.edu/yiqun/SHA1AttackProceedingVersion.pdf.

[101] COCHRAN, M. Notes on the Wang et al. 2^{63} SHA-1 Differential Path [online]. [cit. 2008-05-14]. Dostupný z WWW: http://eprint.iacr.org/2007/474.pdf.

[102] Federal Information Processing Standards Publication 180-2 (+ Change Notice to include SHA-224) [online]. [cit. 2008-05-14]. Dostupný z WWW: http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf.

[103] Cryptographic hash Algorithm Competition [online]. 2007. [cit. 2008-05-14]. Dostupný z WWW: http://csrc.nist.gov/groups/ST/hash/sha-3/index.html.

[104] BONEH, D., BRUMLEY, D. Remote Timing Attacks are Practical [online]. Proceedings of the 12th USENIX Security Symposium. 2003. [cit. 2008-05-14]. Dostupný z WWW: http://citeseer.ist.psu.edu/article/boneh03remote.html.

[105] COPACOBANA - Special-Purpose Hardware for Code-Breaking [online]. [cit. 2008-05-14]. Dostupný z WWW: http://www.copacobana.org/.

[106] KELSEY, J., SCHNEIER, B., WAGNER, D. Related-Key Cryptanalysis of 3-WAY, Biham-DES,CAST, DES-X, NewDES, RC2, and TEA [online]. 1997. [cit. 2008-05-14]. Dostupný z WWW: http://citeseer.ist.psu.edu/318172.html.

[107] WENLING W., WENTAO Z., DENGGUO F. Improved Integral Cryptanalysis of FOX Block Cipher [online]. 2005. [cit. 2008-05-14]. Dostupný z WWW: http://eprint.iacr.org/2005/292.pdf.

[108] KFIHN, U. Improved Cryptanalysis of MISTY1 [online]. proceedings of FSE 9. Springer-Verlag Lecture Notes in Computer Science, vol. 2365. 2002. s. 61-75. [cit. 2008-05-14]. Dostupný z WWW: http://citeseer.ist.psu.edu/kfihn02improved.html.

[109] SONG, D. X., WAGNER, D., TIAN, X. Timing Analysis of Keystrokes and Timing Attacks on SSH [online]. 2001. [cit. 2008-05-14]. Dostupný z WWW: http://www.cs.berkeley.edu/~daw/papers/ssh-use01.pdf.

[110] Berlekamp-Massey algorithm [online]. [cit. 2008-05-14]. Dostupný z WWW: http://planetmath.org/encyclopedia/BerlekampMasseyAlgorithm.html.

[111] COPPERSMITH, D., KRAWCZYK, H., MANOUSR, Y. The Shrinking Generator [online]. 1998. [cit. 2008-05-14]. Dostupný z WWW: http://dsns.csie.nctu.edu.tw/research/crypto/HTML/PDF/C93/22.PDF.

[112] SCHNEIER, B. Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish). Fast Software Encryption, Cambridge Security Workshop Proceedings (December 1993). Springer-Verlag. 1994. s. 191-204.

[113] SCHNEIER, B., KELSEY, J., WAGNER, D., et al. The Twofish Encryption Algorithm: A 128-Bit Block Cipher. New York City: John Wiley & Sons. 1999. ISBN 0-471-35381-7.

[114] RFC 3280. Internet X.509 Public Key Infrastructure: Certificate and CRL Profile [online]. 2002. [cit. 2008-05-14]. Dostupný z WWW: http://www.ietf.org/rfc/rfc3280.txt

[115] GARFINKEL, S. PGP: Pretty Good Privacy. O'Reilly & Associates. 1991. ISBN 1-56592-098-8.

[116] KOOPMAN, P. 32-Bit Cyclic Redundancy Codes for Internet Applications. The International Conference on Dependable Systems and Networks: 459. 2002.

[117] RFC 1994. PPP Challenge Handshake Authentication Protocol (CHAP) [online]. 1996. [cit. 2008-05-14]. Dostupný z WWW: http://tools.ietf.org/html/rfc1994.

Přílohy

A. Tabulka s hodnocením bezpečnosti algoritmů symetrických šifer

Název šifry	Prolomena	Doporučení	Poznámka
3-Way	Ano	Ne	Prolomena pomocí related-key útoku, použito 2^{22} vybraných plaintextů [106]).
Blowfish	Ne	Ne	Prolomena pouze oslabená 4 kolová varianta, nikoli plná 16 kolová, ale doporučuji spíše nástupce, šifru Twofish.
CAST-256	Ne	Ne	Existují skupiny slabých klíčů, které mohou být využity při diferenciální kryptoanalýze.
CMEA	Ano	Ne	Velmi krátký klíč (64 bitů) a neobstojí při chosen-plaintext útoku.
DES	Ano	Ne	Prolomitelný i hrubou silou.
DEAL	Ne	**S výhradami**	Poskytuje větší bezpečnost než DES, ze kterého vychází, lze implementovat na stávajícím DES hw či sw (díky tomu se může vyplatit jeho použití), nevýhodou je malá rychlost šifrování.
FEAL	Ano	Ne	Vychází z DESu, verze s malým početem kol (4, 6, 8) velmi slabé.
GOST	Ne	**Ano**	Ruská obdoba DESu je překvapivě odolná vůči pokusům o kryptoanalýzu.
IDEA-NXT	Ne	**Ano**	Nástupce algoritmu IDEA, dosavadní útoky na integrální kryptoanalýzou na alg. s redukovaným počtem kol z 16 na 7 mají časovou složitost $2^{237.4}$ [107]), což poskytuje velkou rezervu pro bezpečné použití, současně kvůli nutnosti licenčním podmínkám předpokládám malé rozšíření a tudíž malý zájem o její praktické prolomení.
LOKI	Ano	Ne	Neobstojí při útoku diferenciální kryptoanalýzou.

MacGuffin	Ano	Ne	Napadnutelný diferenciální kryptoanalýzou, podle [14] je méně odolný než standartní 16 kolový DES.
MARS	Ne	**S výhradami**	Objevilo se několik pokusů o prolomení, ale stále je zde relativně bezpečná rezerva (bylo prolomeno 21 kol z 32)
MISTY	Ne	Ne	Na 4 kolovou verzi algoritmu byl úspěšně proveden tzv. slicing útok [108]), proto je doporučována aplikace minimálně 8 kol. Nicméně při existenci lepších algoritmů nevidím důvod pro další používání MISTY.
MMB	Ano	Ne	Obsahuje slabiny ve výpočtu sub klíčů a při návrhu ani nebyla brána v potaz lineární kryptoanalýza.
NewDES	Ano	Ne	Pokus o nástupce DES, ale má ještě menší odolnost vůči known-plaintext útokům.
RC6	Ne	**S výhradami**	Již 15 z 20 kol prolomeno, ale plný počet kol nebyl nikdy překonán.
Redoc-II	Ne	Ne	Sice nikdy nebylo překonáno 10 kol šifrování, ale tento algoritmus je velmi pomalý. Jeho rychlejší varianta REDOC-III byla prolomena known-plaintext útokem, takže nemohu doporučit ani ji.
Rijndael	Ne	**S výhradami**	Z 10 kol jich zatím bylo prolomeno 7, což se může zdát jako příliš malá bezpepčností rezerva.
Safer SK128	Ne	**Ano**	Verzi s opraveným výpočtem subklíčů se ještě nepodařilo prolomit.
Serpent	Ne	**Ano**	Má konzervativnější přístup k bezpečnosti než Rijndael, zatím se podařilo prolomit 10 kol z 32 použitých.
SQUARE	Ne	Ne	Algoritmus je odolný vůči diferenciální a integrální kryptoanalýze, ovšem byl vytvořen nový útok přímo na tento algoritmus, kterým bylo prolomeno 6 kol algoritmu. V případě použití je tedy vhodné minimum 8 kol.

Skipjack	Ne	**Ano**	Oslabený 16 kolový Skipjack je prolomen pomocí known-plaintext útoku, ale původní 32 verze úspěšně odolává útokům.
Twofish	Ne	**Ano**	Navzdory některým zprávám stále není k dipozici žádný praktický útok na Twofish.

B. Seznam možných šifrovacích sad protokolu TSL 1.1 (převzato z [68])

Šifrovací sada	Výměna klíčů	Šifra	Haš funkce
TLS_NULL_WITH_NULL_NULL	NULL	NULL	NULL
TLS_RSA_WITH_NULL_MD5	RSA	NULL	MD5
TLS_RSA_WITH_NULL_SHA	RSA	NULL	SHA
TLS_RSA_WITH_RC4_128_MD5	RSA	RC4_128	MD5
TLS_RSA_WITH_RC4_128_SHA	RSA	RC4_128	SHA
TLS_RSA_WITH_IDEA_CBC_SHA	RSA	IDEA_CBC	SHA
TLS_RSA_WITH_DES_CBC_SHA	RSA	DES_CBC	SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA	RSA	3DES_EDE_CBC	SHA
TLS_DH_DSS_WITH_DES_CBC_SHA	DH_DSS	DES_CBC	SHA
TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA	DH_DSS	3DES_EDE_CBC	SHA
TLS_DH_RSA_WITH_DES_CBC_SHA	DH_RSA	DES_CBC	SHA
TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA	DH_RSA	3DES_EDE_CBC	SHA
TLS_DHE_DSS_WITH_DES_CBC_SHA	DHE_DSS	DES_CBC	SHA
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA	DHE_DSS	3DES_EDE_CBC	SHA
TLS_DHE_RSA_WITH_DES_CBC_SHA	DHE_RSA	DES_CBC	SHA
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA	DHE_RSA	3DES_EDE_CBC	SHA
TLS_DH_anon_WITH_RC4_128_MD5	DH_anon	RC4_128	MD5
TLS_DH_anon_WITH_DES_CBC_SHA	DH_anon	DES_CBC	SHA
TLS_DH_anon_WITH_3DES_EDE_CBC_SHA	DH_anon	3DES_EDE_CBC	SHA
(sady přidané v [73])			
TLS_RSA_WITH_AES_128_CBC_SHA	RSA	AES_128_CBC	SHA
TLS_DH_DSS_WITH_AES_128_CBC_SHA	DH_DSS	AES_128_CBC	SHA
TLS_DH_RSA_WITH_AES_128_CBC_SHA	DH_RSA	AES_128_CBC	SHA
TLS_DHE_DSS_WITH_AES_128_CBC_SHA	DHE_DSS	AES_128_CBC	SHA

TLS_DHE_RSA_WITH_AES_128_CBC_SHA	DHE_RSA	AES_128_CBC	SHA
TLS_DH_anon_WITH_AES_128_CBC_SHA	DH_anon	AES_128_CBC	SHA
TLS_RSA_WITH_AES_256_CBC_SHA	RSA	AES_256_CBC	SHA
TLS_DH_DSS_WITH_AES_256_CBC_SHA	DH_DSS	AES_256_CBC	SHA
TLS_DH_RSA_WITH_AES_256_CBC_SHA	DH_RSA	AES_256_CBC	SHA
TLS_DHE_DSS_WITH_AES_256_CBC_SHA	DHE_DSS	AES_256_CBC	SHA
TLS_DHE_RSA_WITH_AES_256_CBC_SHA	DHE_RSA	AES_256_CBC	SHA
TLS_DH_anon_WITH_AES_256_CBC_SHA	DH_anon	AES_256_CBC	SHA