PŘÍRODOVĚDECKÁ FAKULTA UNIVERZITY PALACKÉHO
KATEDRA INFORMATIKY

DIPLOMOVÁ PRÁCE

Bezpečné kryptografické algoritmy

2012 Zdeněk Müller

Anotace

Diplomová práce hodnotí kryptografické metody a postupy z hlediska historického a moderního. Především se zaměřuje na současný vývoj kryptografie. Přehledně popisuje moderní symetrické a asymetrické algoritmy, kryptografické hašovací funkce, generátory pseudonáhodných čísel a autentizační protokoly. Text se zabývá i nejnovějším vývojem kryptografie v podobě kvantové kryptografie. Práce se zejména soustředí na úspěšné metody kryptoanalýzy, zranitelnost algoritmů a jejich náchylnost vzhledem k útokům. V praktické části práce je implementována řada ukázkových algoritmů a teoretická ukázka úspěšného útoku na SSL spojení.

Děkuji panu RNDr. Miroslavu Kolaříkovi, Ph.D., za jeho trpělivost a cenné rady při vedení diplomové práce. Zároveň bych rád poděkoval členům katedry Informatiky Univerzity Palackého v Olomouci i vedení Přírodovědecké fakulty za jejich vstřícnost a pomoc při studiu. Děkuji tímto své rodině, která mi byla po celou dobu oporou.

Obsah

Seznam obrázků

Seznam tabulek

1. Úvod

Z hlediska informačních techologií rozumíme pod pojmem bezpečnost ochranu odpovídajích informačních systémů a informací, které jsou v nich uchovány, zpracovány a přenášeny. Součástí pojmu bezpečnost je i komunikační bezpečnost, fyzická bezpečnost a personální bezpečnost. Bezpečnost nelze chápat jako stav, ale jako proces, který se může v čase měnit. Bezpečnost tedy nelze jednoduše vymezit, protože prostupuje širokou oblastí.

Z pohledu kryptografie vystupuje bezpečnost ve formě informační bezpečnosti, které lze dosáhnout pomocí kryptografických služeb jako jsou důvěrnost, integrita, nepopiratelnost a autentizace. Bezpečnostní cíle charakterizují kryptografické algoritmy a určují jejich míru bezpečnosti. Bezpečnost kryptografického algoritmu se obecně posuzuje podle míry bezpečnosti konstrukce. Kvalitní návrh je nutnou podmínkou, ale není podmínkou dostačující. Nezaručuje totiž jeho bezpečné použití v praxi. Většina úspěšných útoků nesouvisí s konstrukcí algoritmu, ale s jejich implementací a nasazením.

Kryptografie není stejná jako jiné informační systémy. Nelze k ní přistupovat jako k jednoznačnému prostředku zaručení bezpečnosti. Síla bezpečného algoritmu je rovna síle jeho použití. Moderní kryptografie se nezabývá pouze designem kryptografických algoritmů. Zkoumá i potenciální oslabení jejich nasazení. Bezpečnost kryptografického algoritmu není trvalá, mění se v čase. Algoritmus je označen za nevhodný, pokud byla prokázána jeho slabina v návrhu. Úspěšnost útoku nesouvisí pouze se zvýšujícími se výpočetními prostředky, ale i s nalezením kryptoanalytické metody resp. s prostředky pro její použití.

V následujících kapitolách si nejprve zavedeme základní pojmy kryptografie a následně se zaměříme na významné zástupce jednotlivých kategorií. Text má za úkol popsat konstrukci jednotlivých algoritmů a jejich bezpečnostní nedostatky.

2. Moderní kryptografie

Nástup nového směru kryptografie se datuje do sedmdesátých let dvacátého století. Význam kryptografie nastal se zvýšenou potřebou ochrany dat, ale také z důvodu rozvoje komunikačních a počítačových technologií i potřeby služeb, které se netýkaly jen utajování.

Cílem moderní kryptografie je dosáhnout informační bezpečnosti s využitím matematických metod.
Existuje několik druhů metod, které se používají pro dosažení informační bezpečnosti (fyzická, technická atd.). Kryptografie je důležitá součást procesu ochrany dat.

2.1. Bezpečnostní cíle kryptografie

Pro splnění bezpečnostních cílů existují vybrané kryptografické služby, jakými jsou *důvěrnost*, *integrita*, *nepopiratelnost* a *autentizace*. Samotné bezpečnostní cíle pak charakterizují použitý kryptografický algoritmus a určují také jeho míru bezpečnosti. Mezi hlavní bezpečnostní cíle moderní kryptografie patří:

- **Důvěrnost**
  Důvěrnost dat a použitých prostředků. Zabránění znalosti a přístupu k informacím před neoprávněným přístupem třetí strany. Důvěrnost dat lze zabezpečit různými způsoby, mezi které patří nejen šifrování, ale například i fyzické zabezpečení.

- **Integrita dat**
  Integritou dat rozumíme zamezení možnosti neoprávněné změny dat. Úmyslné či neúmyslné pozměnění původní informace pak vede k znehodnocení, nesrozumitelnosti atd.. Celistvost originálních dat může být porušena změnou, smazáním, nebo přidáním části informace.

- **Ověření entit**
  Ověřením (autentizací) entit je myšleno prověření kompetence všech objektů a účastníků (uživatelů, počítače, přenosové cesty, procesu zpracování atd.).

- **Ověření dat**
  Zjištění předpokládané identity dat. Ověření dat úzce souvisí s jejich samotnou integritou. Při autentizaci používaných dat se zkoumá jejich obsah, původ, autor, datum vytvoření atd.

- **Nepopiratelnost**
  Data, která již byla jednou vytvořena, mají být nepopiratelná a to i objektem, který je vytvořil. Nepopíratelnost je důležitá vlastnost při pozdějším

stanovení pravosti dat třetí stranou. Data mohou být nepopiratelná z několika hledisek: originalita autora (vytvoření zprávy), správný příjemce resp. odesílatel, přenos zprávy atd.

- **Důvěryhodnost**
  Důvěryhodností není myšleno pouhé zaručení pravosti objektů. Vedle toho se ověřuje garance připojení (trvalost, stabilita, bezpečnost vůči odposlechu třetí strany) a uchovávání přenosových podrobností (monitoring událostí).

2.2. Informačně bezpečnostní služba

Informačně bezpečnostní služba je způsob jakým se dosáhne splnění určitého bezpečnostního cíle.
Jako bezpečnostní cíl si vezmeme například nepopiratelnost dat. K jeho splnění potřebujeme službu, která nám zajistí, že autor prokáže pravost vytvoření na základě originální verze dat a dříve zveřejněné informace popisující zprávu. Ve skutečné situaci by se jednalo například o zaručení autorství pomocí zveřejněného hashe dokumentu pro jeho pozdější ověření.

2.3. Kryptografické prostředky

Kryptografické prostředky slouží k zajištění bezpečnostních cílů. Moderní kryptografie používá tyto prostředky jako nástroje a mechanismy k dosažení požadovaného cíle.
Nejčastější kryptografické nástroje:

- Šifrovací systémy s tajným klíčem.
- Šifrovací systémy s veřejným klíčem.
- Hašovací funkce.
- Pseudonáhodné generátory znaků.
- Schémata digitálních podpisů.
- Schémata výměny klíčů nebo dohody mezi klíči.
- Schémata autentizační a identifikační.
- Autentizační kódy zpráv.
- Hašové klíčované autentizační kódy zpráv.

2.4. Základní pojmy kryptografie

Mezi základní pojmy kryptografie patří: *kryptografický systém*, *algoritmus*, *zobrazení* a *transformace*.
V klasické kryptografii nebylo nutné pojmy vymezovat, neboť se pro zajištění informační bezpečnosti dat jednalo o jeden proces.
Moderní kryptografie začala uvedené pojmy oddělovat, i když můžeme chápat v rámci jednoho kontextu pojmy systém a algoritmus stejně.
Pro vymezení výše uvedených pojmů použijeme přístup od nejnižší úrovně k nejvyšší.

1. **Kryptografická transformace**
   Kryptografická transformace je funkce, která definuje zpracování dat pomocí zadaného klíče.

2. **Kryptografické zobrazení**
   Kryptografické zobrazení je zobrazení, které přiřadí všem klíčům nebo jiným parametrům kryptografického systému konkrétní kryptografickou transformaci.

   Příklad. Ověření autentičnosti pomocí digitálního podpisu kryptografickým nástrojem.
   Digitální podpis označíme symbolem p. Text, který chceme podepsat označíme m.

   - Pro specifický tajný klíč d dostáváme kryptografickou transformaci vytvoření podpisu $D_d : m \to p = D_d(m)$.
   - Pro specifický veřejný dešifrovací klíč e dostáváme kryptografickou transformaci ověření podpisu E_e, například $E_e(m, p) = \{\,\text{ANO, NE}\,\}$.

3. **Kryptografický algoritmus.**
   Pod pojmem kryptografický algoritmus rozumíme kolekci kryptografických zobrazení a jejich transformací pro uvažovaný kryptografický systém.

   Příklad. Kryptografický algoritmus u digitálního podpisu představuje uspořádaná trojice (G, S, V), kde

   - transformace G představuje generátor, který pro náhodné tajné počáteční nastavení k (seed) vytvoří dvojici veřejného a tajného klíče (e, d).
   - zobrazení S přiřadí každému tajnému klíči d transformaci vytvoření podpisu D_d.
   - zobrazení V přiřadí každému veřejnému klíčí e transfomaci ověření podpisu E_e.

4. **Kryptografický systém.**
   Moderní kryptografie využívá matematické metody k zajištění informační bezpečnosti. *Kryptografický systém* (kryptosystém) je použitá matematická metoda. Tedy kryptosystémem rozumíme proceduru zpracování dat a klíče zahrnující všechny použité kryptografické algoritmy, zobrazení, transformace a pravidla.

   Příklad. U digitálního podpisu představuje kryptografický systém způsob generování klíčů a definice algoritmů pro vytváření a ověřování digitálního podpisu.

   Příklad. Vermanova šifra.
   Kryptografický systém je tvořen:

   - transformacemi pro šifrování E_k a dešifrování D_k
   - kryptografickým algoritmem, který reprezentuje uspořádaná trojice (G, E, D), kde G je transformace generování hesla.
   - pravidlem, které určuje jak tvořit (náhodné generování klíče) a používat klíč (pouze jedno použití).

3. Šifrovací systémy

Šifrovací systémy (šifry) jsou kryptografické systémy, které zaručují informační bezpečnostní služby pomocí kryptografického nástroje šifrování dat.

Obecná definice šifry
Kryptografickým systémem pro šifrování zpráv (šifrou) nazveme uspořádanou pětici (M, C, K, E, D), kde:

- M je neprázdná konečná množina otevřených zpráv.
- C je neprázdná konečná množina šifrovaných zpráv.
- K je neprázdná konečná množina klíčů.
- E je zobrazení přiřazující každému klíči $k \in K$ transformaci E_k šifrování zpráv, $E_k : m \to c$ pro nějaké $m \in M$ a $c \in C$.
- D je zobrazení přiřazující každému klíči $k \in K$ transformaci D_k dešifrování zpráv, $D_k : c \to m$ pro nějaké $m \in M$ a $c \in C$.

Transformace E_k resp. D_k splňují pro každé $k \in K$ a $m \in M$ podmínku $D_k(E_k(m)) = m$.

3.1. Asymetrický systém pro šifrování zpráv

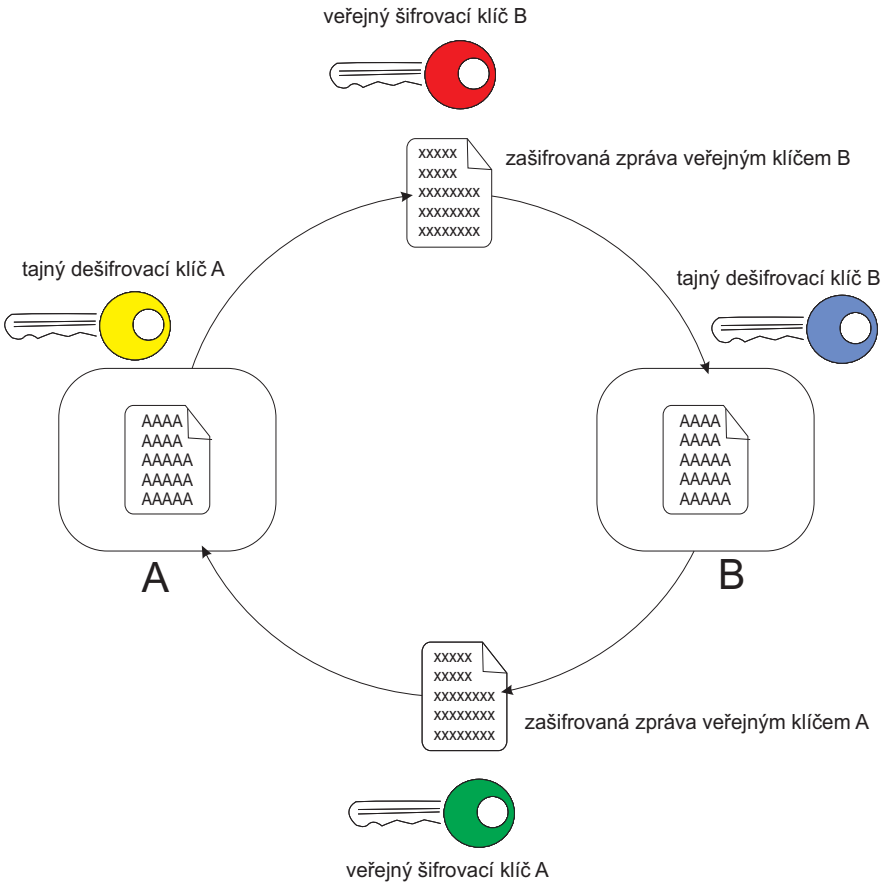
Definice: Asymetrický systém pro šifrování zpráv je takový šifrovací systém (šifra), kde pro skoro všechna $k \in K$ nelze z šifrovací transformace E_k určit dešifrovací transformaci D_k.

Klíč k slouží pro vytvoření utajeného počátečního nastavení, ze kterého pomocí transformace sestrojíme dvojici nových klíčů *(e,d)*. Klíč e je nazýván *veřejný* a je určen k zašifrování. Klíč d se nazývá *tajný* a je přístupný pouze oprávněným osobám (většinou jej využívá pouze autor). Klíče parametrizují šifrovací a dešifrovací transformace. Nepoužíváme pro přehlednost E_k a D_k, píšeme E_e a D_d.

3.2. Symetrický systém pro šifrování zpráv

Definice: Symetrický systém pro šifrování zpráv je takový šifrovací systém, kde pro skoro všechna $k \in K$ platí, že z šifrovací transformace E_k lze jednoznačně určit (převodem, odvozením) dešifrovací transformaci D_k.

Symetrické šifry mají klíč k sloužící k nastavení transformací tajný. Vedle toho může být způsob jakým probíhá transformace známý (DES). Symetrie mezi převodem transformací určuje název systému.



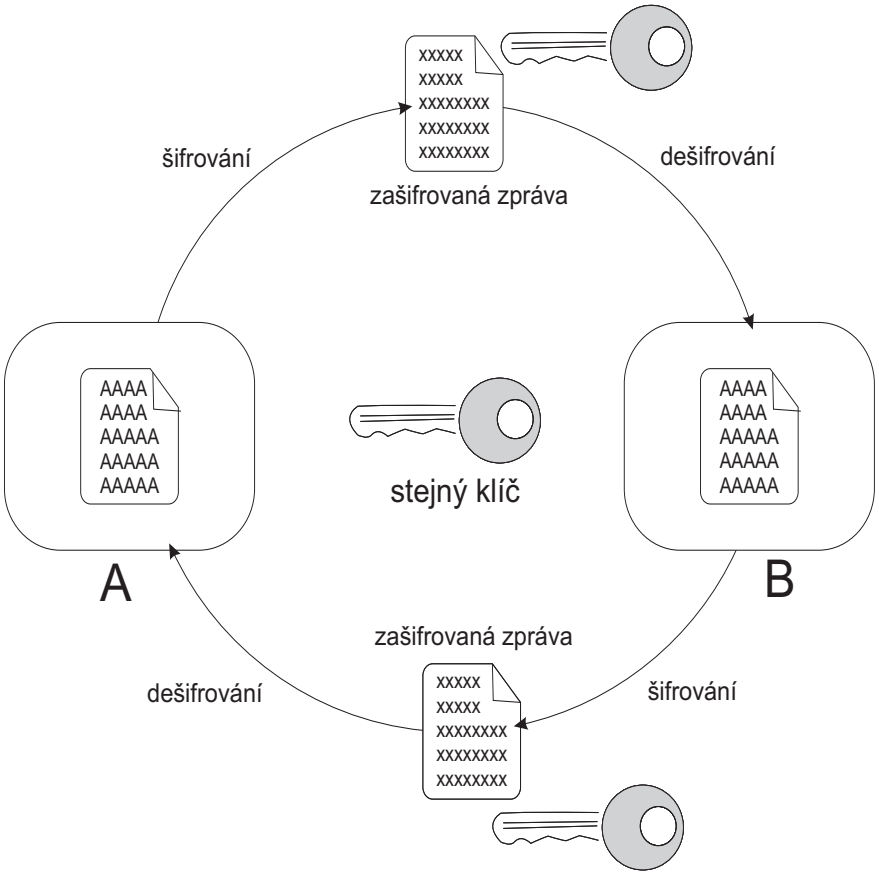
Obrázek 1. Asymetrický systém pro šifrování zpráv.

Příklad. Jednoduchý příklad symetrických šifer představuje *Caesarova šifra* [8]. Položme m = SYMETRIE a zašifrujeme jej pomocí klíče $k = 2$. Zašifrovaná zpráva $c = E_k(m)$ = UAOGVTKG.
Odesilatel provede šifrovací transformaci E_k s klíčem k tj. posune znaky o k pozic vpravo. Příjemce dešifruje zprávu pomocí transformace D_k, tedy posune znaky o k pozic vlevo.

Na příkladu Caesarovy šifry je vidět, že transformace E_k a D_k nejsou totožné, ale jednoduchou cestou odvoditelné.

3.3. Shannonova teorie

Shannon svými závěry položil základy pro *teorii šifrovacích systémů* a *teorii informace*. Pomocí jeho výsledků se posuzuje teoretická míra bezpečnosti použité šifry a to v případě, že máme k dispozici zašifrovanou zprávu.
Shanonova teorie vysvětluje přímou úměru mezi množstvím přijatého textu a



Obrázek 2. Symetrický systém pro šifrování zpráv.

schopností text dešifrovat.

3.3.1. Entropie

Základem teorie informace je změřit množství informace obsažené v textu, tedy entropii.
Entropie je průměrný počet bitů potřebný k *optimálnímu zakódování* zadaného textu. Optimální zakódování textu využívá nejméně bitů k jeho zakódování.
Pravděpodobnost s jakou se znaky resp. slova vyskytují v textu hraje roli s přínosem informace. Znaky vyskytující se s menší pravděpodobností jsou pro nás významnější, protože je jejich míra z pohledu entropie vyšší.

Definice (entropie).
Předpokládejme, že máme zdroj, který produkuje různé zprávy $M(1), M(2), \ldots, M(n)$ s různými pravděpodobnostmi $p(1), p(2), \ldots, p(n)$, přičemž platí $p(1) + p(2) + \cdots + p(n) = 1$.

Entropie je potom dána vztahem:
$H(M) = p(1) * log_2(1/p(1)) + p(2) * log_2(1/p(2)) + \cdots + p(n) * log_2(1/p(n))$.

Příklad.
Mějme dány zprávy jako odpovědi "ano" a "ne". Zdroj vydává 1 (ano), 0 (ne). Zprávy se vyskytují s rozdílnou pravděpodobností $p(0) = 0.7$, $p(1) = 0.3$.
Dostáváme
$H(M) = -p(0) * log_2(p(0)) - p(1) * log_2(p(1)) = -0.7 * log_2(0.7) - 0.3 * log_2(0.3) = 0.7 * 0.515 + 0.3 * 1.737 = 0.882$.

Definice (maximální entropie).
Předpokládejme, že máme zdroj, který produkuje zprávy se stejnou pravděpodobností. Entropie zakódování tohoto textu se nazývá *maximální*. Lze dokázat, že platí:
$H(M) = p(1) * log_2(1/p(1)) + p(2) * log_2(1/p(2)) + \cdots + p(n) * log_2(1/p(n)) \leq (1/n * log_2(n)) + \cdots + (1/n * log_2(n)) = log_2(n)$.

Máme-li dán zdroj, který vyprodukuje n zpráv se stejnou pravděpodobností, tedy $1/n$, pak dosahuje maximální entropie $log_2 n$.

3.3.2. Vzdálenost jednoznačnosti

Definice (absolutní bezpečnost šifry).
Řekneme, že šifra je *absolutně bezpečná*, jestliže se nedozvíme nic nového o otevřeném textu přijmutím libovolného množství šifrovaného textu.

Definice absolutní bezpečnosti vede k úvaze, kterou většina klasických šifer porušuje, protože se vzrůstajícím množstvím přijatého šifrovaného textu se zvyšuje schopnost rozpoznání otevřeného textu (informace o otevřeném textu). Ze zašifrovaného textu pak nemusí být otevřený text přímo patrný, ale dá se vhodnou metodou odvodit.

Poznámka.
Vermanova šifra je příkladem klasické šifry, která má vlastnost absolutní bezpečnosti.

Definice (vzdálenost jednoznačnosti).
Nechť jsme přijali určitý počet znaků šifrovaného textu. Pokud existuje vhodná metoda pomocí níž lze z textu odvodit jediný text otevřený, hovoříme o vzdálenosti jednoznačnosti. Vzdálenost jednoznačnosti označuje daný počet přijatých znaků.

Příklad.
Přijali jsme jeden znak šifrovaného textu "C". Předpokládejme použití nespecifikované šifry, která zašifruje 1 znak na 1 znak obecného přirozeného jazyka. Jeden znak nám však nedává žádnou informaci o otevřeném textu, protože při použití abecedy o 28 znacích je 28 možností. Pokud však přijmeme dva znaky "BC", informace o otevřeném textu se zvyšuje. Vylučujeme bigramy se stejnými písmeny

například "CC". Při získání desítek znaků jsme schopni rozpoznat umístění samohlásek. Při přijmutí tisíce znaků je znám již celý text.
Obecně: Při přijmutí k-tého znaku jsme dosáhli stavu, kdy existuje pouze jeden otevřený text.
Je patrné, že záleží na konkrétním použitém jazyku a šifře.

Z předchozího příkladu je vidět souvislost mezi entropií a vzdáleností jednoznačnosti.

3.3.3. Obsažnost a nadbytečnost jazyka

U přirozených jazyků je pravděpodobnost s jakou jsou produkovány jednotlivé znaky a slova závislá na konkrétní skladbě a druhu textu. Příkladem je výskyt odborných výrazů v matematickém a normálním laickém textu.

Definice (obsažnost jazyka).
Nechť máme dán jazyk L používající l znaků. Množina M_n obsahuje zprávy délky n jazyka L. *Obsažnost jazyka* pro zprávy délky n je definována vztahem: $R_n = H(M_n)/n$.
Obsažnost jazyka odpovídá průměrnému počtu bitů v jednom znaku (průměrná entropie na jeden znak).

Definice (absolutní obsažnost jazyka).
Nechť máme dán jazyk L používající l znaků. Množina M_n obsahuje zprávy délky n jazyka L.
Nechť jsou znaky jazyka L stejně pravděpodobné a zprávy z množiny M_n se vyskytují se stejnou pravděpodobností, pak výraz $R = (log_2(l^n))/n = log_2 l$ nazýváme *absolutní obsažností* jazyka L pro zprávy délky n.

Poznámka (vlastnosti přirozených jazyků).

1. Absolutní obsažnost je vlastnost, které přirozené jazyky nedosahují. Absolutní obsažnost by měl takový jazyk, jehož n-znakové řetězce, pro n přirozené, by se vyskytovaly se stejnou pravděpodobností.

2. Přirozené jazyky mají vlastnost, že čím je délka slova vyšší, tím je výskyt předpokládaného znaku větší (zmenšuje se množina pravděpodobných znaků). Se zvětšujícím se n klesá obsažnost jazyka, tedy průměrná entropie na jeden znak je nepřímo úměrná délce slova.

Pro přirozený jazyk se experimentálně dokázalo, že pro $n \to \infty$ odpovídá hodnota výrazu R_n konstantě. Platí, že $lim_{n\to\infty}(R_n) = r$, kde $R_n = H(M_n)/n$.
Konstantu r nazýváme *obsažností jazyka vzhledem k jednomu písmenu* a udává kolik bitů informace je průměrně obsaženo v jednom znaku daného jazyka.

Definice (nadbytečnost jazyka).
Nechť máme dán jazyk L tvořený l znaky. Uvažujme, že na jeden znak jazyka

L použijeme R bitů. Každé písmeno abecedy lze však reprezentovat průměrně r bity. Hodnota $D = R - r$ se nazývá *nadbytečnost jazyka L vzhledem k jednomu znaku.*

Procentuální nadbytečnost bitů vzhledem k jednomu písmenu získáme jako hodnotu výrazu D/R.
Pro smysluplnou zprávu délky n máme $n * r$ bitů informace. Smysluplných zpráv je 2^{r*n}, celkový počet zpráv je roven 2^{R*n}.

Nadbytečnost jazyka je důležitá pro stanovení zda je rozluštěný text správný nebo ne. Správnost se určuje podle smysluplnosti v použitém jazyce.

Příklad.
Vypočítáme procentuální nadbytečnost bitů vzhledem k jednomu písmenu abecedy o 32 znacích.
$l = 32$ znaků,
$R = log_2 l = 5$ bitů na znak,
$r = lim_{n\to\infty} R_n = 1.6$ bitů na znak,
$D = R - r = 3.4$ bitů na znak,
$D/R = 68\%$ nadbytečnost.

3.3.4. Realizace výpočtu vzdálenosti jednoznačnosti u obecné šifry

1. **Předpoklad:**
   Mějme dánu množinu otevřených textů M, množinu šifrovaných textů C, množinu klíčů K.
   Nechť je $H(K)$ neurčitost klíče a D je nadbytečnost jazyka. Prvky $k \in K$ jsou stejně pravděpodobné, tedy $|K| = 2^{H(K)}$. Označíme c_n zašifrovanou zprávu délky n, pro n přirozené. Dále nechť $X_k(c)$ jsou nezávisle, náhodně vybrané otevřené texty z M pro klíče $k \in K$. Celkem máme $|M| = 2^{R*n}$ zpráv, z toho 2^{r*n} smysluplných.

2. **Dešifrování:**
   Použitím všech $2^{H(K)}$ klíčů na dešifrování textu c dostáváme $2^{H(K)}$ možných zpráv. Smysluplných zpráv je pak $S = 2^{H(K)} * (2^{r*n}/2^{R*N}) = 2^H(K)/2^{D*n}$. Pro $S = 1$ dostáváme původní zprávu. Odtud $H(K) = D * n$, tedy $n = H(K)/D$.

3. **Závěr:**
   Vzdálenost jednoznačnosti je rovna n, kde $n = H(K)/D$.

Poznámka.
Úvaha uvedená v této kapitole není aplikovatelná na libovolnou šifru a je pouze ilustrativní.

Příklad.

1. Jednoduchá substituce nad českou abecedou.
   Pro českou abecedu je hodnota nadbytečnosti vzhledem k jednomu znaku $D = 3.4$ bitů na znak. Vzdálenost jednoznačnosti je $n = H(K)/D = log_2(32!)/3.4 = 118/3.4 = 34.6$. To znamená, že pro získání pouze jednoho možného otevřeného textu potřebujeme 35 znaků bloku zašifrovaného textu. Blok o 35 znacích obsahuje dostatečné množství informace pro dešifrování.

2. Bloková šifra DES
   Uvažujme blokovou šifru DES s blokem délky 56 bitů a s 56 bitovým klíčem. Vezmeme si otevřený text nad českou abecedou.
   Předpokládejme, že bude jeden znak otevřeného textu reprezentován jedním bajtem. Pro 8 bitů vstupu máme pouze 1.5 bitů informace, tedy nadbytečnost informace je 6.5 bitu na bajt.
   Vzdálenost jednoznačnosti: $n = H(K)/D = log_2(2^{56})/6.5 = 8.6$ bajtů otevřeného textu.
   K dešifrování jsou potřeba dva bloky šifrovaného textu resp. jeden celý blok a část druhého bloku.

Poznámka. Na předchozím případě je vidět jakou roli hraje vzdálenost jednoznačnosti, protože nám udává odhad množství potřebné informace. Vedle toho však existuje případná složitost úlohy, protože sice zhruba víme kolik potřebujeme šifrovaného textu, ale metoda dešifrování je většinou obtížná.

4. Kryptoanalýza

Kryptoanalýzou jsou souhrně označovány metody pro odhalování obsahu a smyslu šifrovaných informací. Cílem kryptoanalýzy je typicky zjištění šifrovacího klíče. V případě, že není známý popis šifrovacího procesu, snaží se kryptoanalytik popsat šifrovací algoritmus.

Kryptoanalýza se obecně nezabývá pouze kryptografickou formou útoku, tedy použitím nějaké matematické metody pro prolomení či zjednodušením složitosti použité šifry. Pod pojem kryptoanalýza se dají zahrnout i jiné metody útoku jako jsou krádeže citlivých údajů, fyzické prolomení, metody záznamu dat (stisknuté klávesy) apod. Jedná se o metody, které se nezaměřují na nedostatky skutečného šifrování, ale využívají chyb prostředí nebo lidského faktoru.

Kryptoanalýza byla důležitou součástí historických událostí a to nejen ve válečných konfliktech. Rozluštění Zimmermannnova telegramu [9], ve kterém nabízelo Německo podporu Mexiku za útok na USA, uspíšilo vstup USA do 1.světové války. Stejně tak i rozluštění kódu Enigma [12], které se podařilo spojencům utajit nacistickému Německu, napomohlo ukončení válečnému konfliktu v Evropě. Kryptoanalýza se díky podobným událostem stala důležitým zkoumaným oborem.

4.1. Klasické kryptoanalytické metody

Pojem kryptoanalýza je poměrně nový, ale klasické dešifrovací postupy jsou již dávno známé. Jedna z klasických kryptoanalytických metod je *frekvenční analýza*. Frekvenční analýza je metoda, která se používá u jednoduchých substitučních šifer jako je Caesarova šifra. Jedná se o šifry, které pracují s přirozenými jazyky. Některé znaky resp. digramy přirozeného jazyka se vyskytují častěji něž jiné, například pro češtinu je to "a" nebo "ch", samozřejmě záleží na typu textu a autorovově stylu. Frekvenční analýza využívá statistických vlastností otevřeného textu.

4.1.1. Kryptoanalýza Vigenerovi šifry

Jedna z nejznámějších klasických šifer je polyalfabetická Vigenerova šifra. Vigenerova šifra používá opakovaně stejný klíč na šifrování různých znaků. Byla dlouho považována za absolutně bezpečnou než byla v devatenáctém století definitivně prolomena. Nezávisle na sobě to dokázali Charles Babbage a později Friedrich Kasiski. Charles Babbage se zaměřil na prolomení upravené Vigenerovi šifry, jejímž autorem byl John Hall Brock Thwaites. Charles Babbage důkaz přímo nezveřenil, ale jak se později z jeho poznámek zjistilo, používal podobnou metodu jako Kasiski. Kasinski test je založen na skutečnosti, že šifrování často užívaných slovních spojení (např. pro anglické "the") se přímo promítá do stejných šifrovaných znaků, které tvoří opakující se znakové šifrované skupiny.

Hlavním nedostatkem Vigenerovi šifry je opakované použití stejného klíče. Podaří-li se odhalit délku klíče, pak lze text dle jeho délky rozdělit do bloků a aplikovat frekvenční analýzu. Existují dvě kryptoanalytické metody na zjištění délky klíče - Kasinski test a Friedmanův test.

Kryptoanalýza Vigenerovi šifry probíhá v následujích krocích:

1. zjištění délky klíče - Kasinski test nebo Friedmanův test.
2. rozdělení šifrovaného textu do řádků matice, která má počet sloupců dle délky klíče. Každý znak sloupce je zašifrován stejným znakem klíče.
3. použije se některá z kryptoanalytických metod pro Caesarovu šifru na sloupce matice.

Kasinski test hledá totožné úseky šifrovaného textu. Pokud se při šifrování aplikuje nevhodný klíč, stejný otevřený úsek se projeví stejným šifrovaným úsekem. Pro zašifrování následujícího textu se jako klíč použije text "bcde". Blok textu "alex" se v druhém případě projeví jako stejný šifrovaný blok. Uvažuje se anglická abeceda o 26 znacích.

Otevřený text:	**alex** is short form of **alex**andra and **alex**is.
Klíč:	bcde bc debcd ebcd eb cdebcdebc deb cdebcd.
Šifrovaný text:	**bnhb** ju vlptw jptp sg **coiy**cqhsd dre **coiy**kv.

Pokud by byl použit jiný klíč, pak by byl Kasinski test neefektivní a ani v prvním případě výskytu bloku textu "alex" není test úspěšný. Delší úseky textu test ještě zpřesňují, protože je zde větší šance, že se opakovaný úsek projeví. U následujícího textu se opakují stejné úseky a lze odhadnout délku klíče: **aic**xfdut**jtsnm**pqywemoy**aic**sa**jtsnm**. Vzdálenost mezi stejnými bloky "aic" je 21 a odtud dostáváme, že klíč může mít délku 21 nebo 7 nebo 3. Vzdálenost dalšího stejného bloku "jtsnm" je 18, tedy klíč má možné délky 18, 9, 6, 3, 2. Provede-li se průnik množin možných délek klíče, pak vychází délka klíče 3.

Friedmanův test (kappa test) je druhou kryptoanalytickou metodou pro nalezení délky klíče u Vigenerovi šifry. Využívá tzv. index koincidence (index of coincidence (IC)), který počítá výskyty totožných znaků na stejných pozicích u dvou textů resp. porovnáním samotného textu se sebou. Index se buď počítá pro libovolný text jazyka, nebo normalizovaně pro celý přirozený jazyk a je vyšší u přirozených jazyků. Většinou se náhodný index udává jako průměrná hodnota pro přirozený jazyk a to na základě relativních frekvencí znaků. Například relativní frekvence znaku "a" je pro angličtinu 8.167%, pro němčinu 6.51%, pro češtinu 6.19%. Jestliže jsou relativní frekvence označeny f a c udává počet znaků abecedy, pak lze průměrnou hodnotu indexu vypočítat jako

$$IC_{expected} = \sum_{i=1}^{c} \frac{f_i^2}{1/c}$$

Pokud by byly všechny znaky rovnoměrně rozmístěné, vyšel by index 1. Například pro agličtinu je hodnota indexu 1.73, pro němčinu 2.05. K výpočtu délky klíče je využita normalizovaná hodnota κ_p, což je pravděpodobnost, že jsou dva náhodně zvolené texty přirozeného jazyka stejné (pro angličtinu 0.067 = 1.73 / 26). Dále je k výpočtu potřeba hodnota $\kappa_r = 1/c$, což je předpokládaná pravděpodobnost náhodného rovnoměrného rozložení výběru abecedy (pro angličtinu 0.0385 = 1 / 26). Předpokládanou délku klíče pak udává hodnota výrazu

$$\frac{\kappa_p - \kappa_r}{\kappa_o - \kappa_r},$$

kde κ_o pro délku textu N, počet znaků abecedy c a pro frekvence c znaků abecedy $n_1 \dots n_c$ je dána vztahem:

$$\kappa_o = \frac{\sum_{i=1}^{c} = n_i(n_i - 1)}{N(N-1)}.$$

Vypočítaná délka klíče je pouze přibližná, je potřeba provést korelaci. K upřesnění výsledku se šifrovaný text rozdělí do řádků matice, která bude mít tolik sloupců kolik je předpokládaná délka klíče. Pro každý sloupec se spočítá $IC_{expected}$ pro daný text a provede se průměr vypočítaných hodnot. Proces se opakuje pro všechny předpokládané délky klíče, dokud se nenalezne nejlepší výsledek. Průměrný index, který se nejvíce blíží $IC_{expected}$ přirozeného jazyka odpovídá nejlépe délce klíče. Výpočet lze zjednodušit pomocí výsledků Kasinskiho testu, který vymezí možné délky klíče.

Jestliže je známá délka klíče, vytvoříme si matici, která bude mít řádky naplněny šifrovaným textem. Matice bude mít počet sloupců odpovídající délce klíče (podobně jako u kappa testu). Sloupce matice budou všechny šifrovány stejným znakem klíče, což odpovídá posunu u Caesarovi šifry. Při kryptoanalýze se pracuje s každým sloupcem zvlášť. Nyní se může aplikovat buď frekveční analýza nebo některá jiná metoda jako je například útok hroubou silou v podobě otestování všech možností. Jak bylo řečeno výše, některé znaky resp. digramy resp. trigramy přirozeného jazyka se vyskytují častěji a lze jejich časté frekvence vypozorovat a přímo odvodit znaky, které jsou jejich zašifrováním. Otestování všech možností je přímočařejší metoda. Například pro angličtinu je nutné vyzkoušet 26 posununů textu. Při hledání správného znaku klíče se počítá korelace mezi frekvencemi znaků dešifrovaného textu a relativními frekvencemi. Pokud je korelační hodnota nejvyšší, je nalezen znak klíče. Korelační koeficient se počítá ze vztahu:

$$\chi = \sum_{i=1}^{c} n_i f_i,$$

kde c je počet znaků jazyka, n_i jsou frekvence výskytů znaků v textu a f_i jsou relativní frekvence přirozeného jazyka.

4.2. Moderní kryptoanalytické metody

Kryptoanalytické metody se měnily v čase dle vývoje šifrovacích metod. Klasické kryptoanalytické metody využívaly známých mechanismů jako tomu bylo u frekvenční analýzy, kde se uplatnily statistické vlastnosti přirozených jazyků. Druhá světová válka je jedním z milníků, který přinesl jednak éru šifrovacích strojů jako byla Enigma, ale i propracované kryptoanalytické mechanismy jako byly Bomba kryptologiczna [10] nebo British Bombe [11]. Návrháři šifer pochopili, že je nutné vytvářet šifry, které budou alespoň dlouhodobě odolné.

Moderní šifry jsou ve většině případů natolik odolné, že není prakticky možné šifru přímo prolomit. Kryptoanalýza se mění a hledají se jiné mechanismy útoku jako jsou útoky bočním kanálem. Mnoho důležitých bezpečnostních chyb pochází z nedostatků v návrhu aplikací nebo protokolů, tedy nejde o problém moderní šifry, ale jejího nepřesného použití. Typickými příklady úspěšných kryptoanalytických útoků jsou chyby v generování náhodných klíčových hodnot pro protokol SSL v prohlížeči Netscape (rok 1996, verze Netscape 1.1), chyba v implementaci SSL protokolu (rok 2009) nebo útok formou záznamu PIN kódů na platební terminály Shell (rok 2006). I když jsou dodrženy všechny softwarové i hardwarové podmínky návrhu, existují další bezpečnostní hrozby ve formě phishingu (podvodné získávání hesel vydáváním se za důvěryhodný subjekt) nebo útoků v podobě sociálního engineeringu (podvodné shromažďování informací).

Existují však úspěšné kryptoanalytické útoky, které dokázaly nedostatky v návrhu používaných šifer. Z blokových šifer, které měly nahradit DES, jsou to například FEAL-4, Mandryga viz. 5.3.4.. Asi jeden z nejznámějších příkladů úspěšného útoku hrubou silou je útok na blokovou šifru DES viz. 5.3.4.. Z mobilních technologií to jsou například proudové šifry A5/1, A5/2 viz. 2, bloková CMEA. Dalším příkladem může být bezpečnostní protokol WEP, jehož nedostatkem je použití šifry RC4 a inicializačního protokolu IV viz. 5.2.6.. Z hašovacích funkcí je typickým příkladem nevhodnost použití MD5 u protokolu SSL. V případě MD5 u SSL bylo využito jednak kolizních útoků, ale i chyb vydavatele certifikátu.

4.2.1. Typy kryptoanalytických útoků

Kryptoanalytické útoky je možné rozlišit podle účinnosti s jakou představují nebezpečí pro reálný svět. Většina moderních útoků je pouze na teoretické úrovni a nejsou pro praktickou situaci plně použitelné. Případný útok také zavisí na cíli s jakým je útok prováděn, zda jsou známé nějaké tajné informace, výpočetní složitosti problému, zda je útok prováděn na kompletní kryptosystém nebo na její oslabenou verzi atd.

Jak bylo řečeno hned na začátku, typy útoků se také liší podle *stupně úspěšnosti* jakého útočník dosáhl. Lars Knudsen v roce 1998 navrhl klasifikaci útoků na blokové šifry, která byla založena na kvalitě získané informace.

- **Celková znalost** - útočník prozkoumal kompletně funkcionalitu přílušného algoritmu, ale nepodařilo se mu odhalit tajný klíč.
- **Lokální znalost** - útočník odhalit otevřené (šifrované) texty, které před tím nebyly známé.
- **Informační znalost** - útočník zjistil informace (z pohledu Shannovy teorie viz. 3.3.) o otevřeném (šifrovaném) textu.
- **Schopnost rozlišení algoritmu** - útočník je schopen rozlišit algoritmus od náhodných permutací.
- **Úplné prolomení** - útočník získal tajný klíč.

U kryptoanalýzy se také posuzuje *množtsví a druh prostředků* jako jsou čas, pamět a data. Čas je možno chápat jako počet výpočetních kroků při dešifrování. Data představují množství otevřených a šifrovaných textů. Často je obtížné stanovit přesné hodnoty požadovaných prostředků, protože útok posuzuje z teoretické úrovně a není možno provést testování.

Je veliký rozdíl zda se šifra posuzuje z *akademického* nebo z *reálného hlediska*. Akademický pohled je poměrně konzervativní, většinou se uvažují ideální podmínky, protože je hlavním úkolem ověření teoretických nedostatků šifry. Pokud je šifra prohlášena za kryptograficky nedokonalou, neznamená to, že je závěr užitečný pro reálný útok. Akademický přístup mnohdy zkoumá oslabené verze šifer, jako jsou blokové šifry s menším počtem rund, protože je útok na plnou verzi neproveditelný. Naproti tomu musí útočník v reálné situaci řešit další bezpečnostní mechanismy, které mají za úkol druhotnou ochranu systému. Stává se, že je šifra prohlášena za teoreticky dokonalou, ale praxe ukáže opak, protože záleží na konkrétní realizaci algoritmu. Je velmi důležité opatrně dodržovat a kontrolovat prvky návrhu, protože bylo dokázáno na mnoha příkladech, že může být i bezpečná šifra oslabena vedlejších kanálem.

Kryptoanalýza se dělí i podle toho jak moc je systém útočníkovi přístupný. Ideální systém by neměl útočníkovi poskytnout žádný vedlejší kanál, kterým by mohl testovat vstupy. I když se použije bezpečná šifra, může být oslabena nevhodně navrženým systémem. Většinou se předpokládá, že útočník plně zná princip šifrovacího algoritmu, což mu umožňuje hledat slabiny v návrhu bez nutnosti zkoumat systém jako černou skříňku. Existuje mnoho příkladů z historie, kdy se útoky na tajné šifry soutředily do formy špionáže, reverzního inženýrství atd. Tedy v podobě, kdy se šifra zpětně rekonstruovala z dostupných informací.

Kryptoanalýzu můžeme dělit podle *druhu informace* se kterou útočník pracuje:

- **Ciphertext-only attack (known ciphertext attack)** je typ útoku, kdy útočník zná pouze šifrované texty. Z šifrovaného textu se snaží získat informaci o příslušném otevřeném textu a pokud je velmi úspěšný odvodí i šifrovací klíč. Úspěchem je však mnohdy odhalení jakékoliv informace.

  Z historického hlediska byly útoky při znalosti šifrovaného textu založeny na frekvenční analýze (viz. 4.1.). Tedy na hledání statistických vlastností v šifrovaném textu. S nástupem moderní kryptografie přišel jeden ze základních požadavků, který se u každé šifry posuzoval, a to aby se jevil šifrovaný text jako náhodný šum bez statistických odchylek. Útočník by tedy neměl mít šanci rozlišit šifrovaný text od ostatního toku v přenosovém kanálu. Uveďme si dva příklady, kdy byl použit ciphertext-only útok. Point-to-Point Tunneling Protocol (PPTP) [13] byl protokol pro vituální soukromé sítě, který používal stejný RC4 klíč pro příjemce i odesílatele a tím dával prostor pro použití ciphertext-only útoku. Další z typických příkladů je algoritmus WEP [14].

- **Known-plaintext attack (KPA)** je typ útoku, kdy útočník zná sérii otevřených textů a příslušných šifrovaných textů, které využije k odvození šifrovacího klíče nebo ke konstrukci kódovací knihy (codebook), knihy pro konstrukci kódů.

  Pokud útočník dokáže zkonstruovat vlastní otevřený text, může příjemce mást nepravdivými zprávami. Velmi známý příklad náchylný na known-plaintext útok je ZIP archiv. V případě ZIP archivů stačí znát jen část archivu v podobě otevřené části a pak je možné odhalit použitý klíč. Klasické šifry jsou také velmi náchylné na tento typ útoku.

- **Chosen-plaintext** je typ útoku, kdy si útočník zvolí vlastní otevřený text a získá tak potenciálně příslušný šifrovaný text. Cílem útoku je získat informace, které oslabí bezpečnost šifrovacího schéma. V nejhorším případě odhalí útočník i šifrovací klíč. Chosen-plaintext se dělí na pevně stanovený útok (batch), kdy jsou otevřené texty voleny samostatně před dešifrováním a na volitelně stanovitelný útok (adaptive), kdy jsou texty voleny na základě předchozích textů.

  Útok se jeví nereálný, ale pokud má útočník přístup k šifrovacím prostředkům, může si požadavkem volit otevřené texty. Jedná se hlavně o šifrování s veřejným klíčem, kdy má ke klíči přístup. V případě šifrování s veřejným klíčem je důležité, aby se šifra jevila dostatečně náhodná. Tedy, aby nemohl útočník vytvořit slovník otevřený text - šifrovaný text, který by mu napomohl k odhalení klíče. Pokud je šifra odolná vůči chosen-plaintext útoku je odolná i vůči known-plaintext útoku a obecně i proti ciphertext-only útoku. Velmi známým příkladem je Gardening, což byl termín používaný anglickými kryptografy za 2.světové války k prolomení šifrovacího stroje Enigma.

Spojenci využívaly známé části otevřeného textu tzv. cribs získaných opakovaným výskytem v šifrovaných zprávách.

- **Chosen-ciphertext** je typ útoku, kdy si útočník zvolí vlastní šifrovaný text a získá tak potenciálně příslušný otevřený text. Cílem útoku je získat informace, které oslabí bezpečnost šifrovacího schéma. V nejhorším případě odhalí útočník i šifrovací klíč. Útok se opět rozdělují na adaptivní a neadaptivní. Speciálním typem jsou pak tzv. Lunchtime útoky, které jsou na pokraji adaptivních s omezením na dobu, po kterou je možno texty volit.

  Pokud je systém náchylný k chosen-ciphertext útoku, musí si dát návrháři velký pozor, aby neumožnili útočníkovi použít opakovaně systém a dešifrovat tak text. Ze známých příkladů uveďme například asymetrický šifrovací algoritmus El Gamal, který byl sématicky odolný vůči chosen-plaintext útoku, ale jednoduše se dal prolomit pomocí chosen-ciphertext útoku. Dalším příkladem může být nevhodné použití RSA (SSL), kdy mohl útočník odhalit oteveřený text pomocí úprav původního šifrovaného textu. Významnou oblastí, kde se musí počítat s možností tohoto útoku jsou čipové karty.

- **Adaptive chosen-ciphertext attack** je speciální typ chosen-ciphertext útoků. Jedná se o útok, kdy jsou texty voleny v závislosti na předchozích. Texty mohou být modifikací předchozích nebo se zvolí jiný text, pokud se s předešlým neuspělo atd. Cílem je volit texty tak, aby vedl výsledek k odhalení původního textu, nebo dešifrovacího klíče. Velmi dobrým příkladem jsou útoky viz. 11.2.1. 11.2.2., které jsou zpracovány a popsány v praktické části textu.

- **Related-key attack** je útok podobný chosen-plaintext útoku s tím rozdílem, že může útočník zvolit dva šifrované texty, které budou šifrovány různými neznámými klíči, přičemž existuje mezi klíči matematický vztah, např. se liší o jeden bit.

  Ačkoliv se tento útok zdá poměrně nereálný, existuje několik příkladů, které ukazují, že neprověřený systém může tvořit matematicky závislé klíče. Prevence útoku spočívá ve výběru vhodné metody pro generování klíčů, jako je například tvorba klíčů ze základu pomocí klíčové derivační funkce. Příkladem algoritmu náchylného na tento typ útoku je WEP (Wired Equivalent Privacy). Každý klient Wi-Fi síťové karty a access point sítě sdílel stejný WEP-klíč. K šifrovaní se používala proudová šifra RC4 s 24 bitovovým IV (asi 17 miliónů možností) pro každý paket zprávy. Pro proudovou šifru nesměl být použit 2x stejný klíč. RC4 klíč byl zřetězen s WEP-klíčem. WEP klíče se měly manuálně měnit, ale to se stávalo velmi zřídka. Vzhledem k narozeninovému paradoxu je pravděpodobné, že každý 4096 paket sdílel stejný IV a proto i stejný RC4 klíč. Mnohem horší je situace pokud se použijí slabé klíče.

Výše uvedené typy útoků se také liší v tom jak moc jsou použitelné v praxi. Některé útoky jsou mnohem realističtější jako je known-plaintext útok, jiné zase více teoretické jako je related-key útok.

4.3. Kryptoanalýza u symetrických šifrovacích systémů

4.3.1. Analýza nelineárních transformací

Principem statistické analýzy nelineárních transformací je nalezení vlastností, které spolu souhlasí s vysokou pravděpodobností. Vlastnosti jsou pak použity v mnoha řešeních jakou je obnova klíčů u blokových šifer nebo kolizní útoky na hašovací funkce. Mohou být použity samostatně i ve spojení s jinými vlastnosti.

Statistická analýza nelineárních transformací pracuje s pravděpodobnostním modelem, který je definován jako funkce f se vstupní vlastností $\mathcal{A}$ a výstupní vlastností $\mathcal{B}$. Říkáme, že model má pravděpodobnost p, jestliže platí:

$$\mathbb{P}\big[\mathcal{A} \xrightarrow{f} \mathcal{B}\big] = p.$$

Pravděpodobnostní modely se využívají v lineární a diferenciální kryptoanalýze.

Lineární kryptoanalýza
Lineární kryptoanalýza je metoda hledání afinních aproximací, které popisují princip šifry. Lineární aproximace je lineární relace mezi vstupními a výstupními bity transformace. Jde vlastně o metodu, která aproximuje nelineární komponenty pomocí lineárních funkcí. Lineární kryproanalýza je vedle diferenciální kryptoanalýzy nejčastněji používaná metoda útoku na blokové i proudové šifry. Metoda byla použita pro experimentální útok na DES (Matsui, 1993), který ale není v praxi možný, protože vyžaduje 2^{43} otevřených textů.

Lineární kryptoanalýza se skládá ze dvou fází. Nejprve je nutné sestrojit lineární rovnice pro příslušné otevřené texty, šifrované texty a bity klíčů. Rovnice pracují s pravděpodobnostním modelem (v prostoru všech možných hodnot proměnných), který má vysokou pravděpodobnost ($p \approx 0$ nebo $p \approx 1$). V druhé fázi se odvodí bity klíčů a to použitím lineárních rovnic na známé otevřené texty resp. šifrované texty.

Nejprve si popíšeme fázi *setrojení lineárních rovnic* pro příslušné otevřené texty, šifrované texty a bity klíčů. Vzhledem k tomu, že se řešení lineární kryptoanalýzy liší v pravděpodobnosti, nazývá se také jako lineární aproximace. Předpokládejme tedy, že máme dánu funkci f a hledáme boolovské vektory (maska) λ_a a λ_b, takové že platí

$$\lambda_a \cdot X = \lambda_b \cdot f(X)$$

s pravděpodobností $1/2 + \varepsilon$ pro dostatečně velké $|\varepsilon|$. Operace $\cdot$ představuje mezivýsledek. Jestliže je $f = f_K$ funkce pro klíč, pak vztah zahrnující bity klíče je dán

$$\lambda_a \cdot X \oplus \lambda_b \cdot f(X) = \lambda_k \cdot K. \tag{1}$$

Vztah si můžeme předvést na útoku na DES pro S-box S_5 aproximovaný vztahem $(0,0,0,1) \cdot X = (1,1,1,1) \cdot S_5(X)$, který má pravděpodobnost 1/2 - 5/16. Výsledná aproximace platí pro rundovní funkci F

$$X_{15} \oplus F_K(X)_7 \oplus F_K(X)_{18} \oplus F_K(X)_{24} \oplus F_K(X)_{29} = K_{22},$$

kde čísla závorek označují indexy bitů, které tvoří masku.

Popíšeme si princip jak se dá odvodit vztah 1. Nejprve najdeme aproximaci pro každou rundu, takovou že masku $(\lambda_I^i, \lambda_O^i, \lambda_K^i)$ lze propojit

$$\lambda_I^i = \lambda_O^{i+1}$$

tj. výstupní bit masky se shoduje s vstupním bitem masky další rundy. Nyní můžeme aplikovat následující pozorování.

Tvrzení (Piling-up lemma). Nechť $X_1, \cdots, X_n$ je n statisticky nezávislých náhodných boolovských proměných s $\mathbb{P}\big[X_i = 0\big] = 1/2 + \varepsilon_i$. Pak

$$\mathbb{P}\big[X_1 \oplus X_2 \oplus \cdots \oplus X_n = 0\big] = \frac{1}{2} + \frac{1}{2}\prod_i (2\varepsilon_i).$$

Za určitých předpokladů nezávislosti (podobně u diferenciální kryptoanalýzy), můžeme sečíst všechny aproximace a odvodit

$$\bigoplus_i (\lambda_I^i \cdot A^i \oplus \lambda_O^i \cdot A^{i+1} \oplus \lambda_K^i \cdot K_i) = 0 \Longrightarrow$$
$$\Longrightarrow \bigoplus_i \lambda_K^i \cdot K_i \oplus \lambda_I^1 \cdot A^1 \oplus \lambda_O^R A^{r+1} \Longrightarrow \lambda_I^1 \cdot P \oplus \lambda_O^R C = \bigoplus \lambda_K^i \cdot K_i, \tag{2}$$

jehož pravděpodobnost je dána $1/2(1 + \prod(2\varepsilon)^i)$, kde K_i je i-tý podklíč.

Rozšířením základní myšlenky může být použití více lineárních aproximací současně, což zahrnuje různé λ_K^i. Útočník tak může odvodit různé bity klíče současně, ale zvýší se tím časová složitost.

Postup konstrukce aproximací je rozdílný pro různé šifry, nejčastěji se lineární aproximace používá u blokových šifer, tedy u šifer se substitučně permutačním principem. Metoda se soustředí na S-boxy, nelineární části šifry. Pro dostačně malé S-boxy se dají sestrojit lineární rovnice a na základě vstupních a výstupních bitů se odvodí zkreslení a vybere se nejlepší aproximace.

V druhé fázi lineární kryptoanalýzy se *odvozují bity klíčů*. Použije se jednoduchý algoritmus tzv. vyčerpávající fáze hledání, kdy se z množiny kandidátů klíčů

odvodí bity klíčů. Pomocí známých otevřených textů se odhadují hodnoty bitů klíčů pro příslušnou aproximaci.

Pro každou množinu bitů klíče na pravé straně, pro tzv. částečný klíč, se počítá kolikrát je aproximace úspěšná (na množině dvojic otevřený - šifrovaný text). Jako nejpravděpodobnější množina bitů klíče je pak zvolena množina, která má nejvyšší počet úspěšností. Hodnota porovnání úspěšnosti se odvozuje jako absolutní rozdíl na poloviční množině dvojic otevřený - šifrovaný text, sleduje se totiž samotné statistické zkreslení částečného klíče od klíče šifry.

Diferenciální kryptoanalýza

Obecně můžeme diferenciální kryptoanalýzu popsat jako metodu, která využívá poznatků jak změny na vstupu ovlivňují změny na výstupu. V případě blokových šifer se rozdíly využívají k prozkoumání vnitřích transformací a můžou vést k celkovému odhalení tajného klíče.

Autorem metody je Eli Biham a Adi Shamir. Používá se jak na blokové, tak i na hašovací funkce. Nejčastěji se preferuje choosen-plaintext útok viz. 4.2.1.. Diferenciální kryptoanalýza byla úspěšně použita na FEAL-4 viz. 5.3.4., k prolomení stačí pouhých osm otevřených textů. Byla použita i v návrhu teoretického útoku na DES viz. 5.3.4.. Později však bylo prokázáno, že je DES odolná vůči diferenciální kryptoanalýze, protože její konstrukce zaručuje, že je algoritmus náchylný i malé změny a tak zabraňuje odhalení vnitřních transformací. Diferenciální kryptoanalýza je velmi úspěšná metoda, která vedla k prolomení mnoha šifer.

Myšlenka diferenciální kryptoanalýzy spočívá v použití dvojic otevřeného textu, které se liší o konstantní rozdíl tzv. diferenci. Nejčastěji se jedná o $\oplus$-diferenci

$$\Delta^{\oplus} P = P_1 \oplus P_2$$

mezi dvojicemi P_1 a P_2 otevřeného textu. Útočník počítá rozdíly v příslušných šifrovaných textech a snaží se najít statistické vlastnosti transformací. Výsledná dvojice diferencí se nazývá *diferenciál*. V případě blokových šifer je statistická vlastnost závislá na povaze S-boxů. Útočník zkoumá diference Δ_x, Δ_y, kde $\Delta_y = S(X \oplus \Delta_x) \oplus S(X)$ pro každý S-box S. Úkolem diferenciální kryptoanalýzy je nalezení tajného klíče, což vyžaduje šifrované texty pro velký počet dvojic otevřených textů. Předpokladem je, že diferenciál platí nejméně pro $(r-1)$ rund. Odtud je možné odvodit potenciální rundovní klíč, protože je diference statická. Pokud je rundovní klíč dostačně krátký, můžeme pomocí vyčerpávajícího dešifrování (podobně jako u druhé fáze lineární kryptoanalýzy) dvojic šifrovaných textů vyřadit všechny potenciální kandidáty rundovních klíčů. Je-li jeden rundovní klíč pravděpodobnější než ostatní kandidáti, pak se zřejmě jedná o hledaný rundovní klíč. Vhodná volba diferencí souvisí s úspěšností útoku. Pro různé fáze dešifrování se pak navrhují nejpravděpodobnější diference tzv. *diferenciální charakteristiky*.

Nyní si popíšeme diferenciál teoreticky. Diferenciál na transformaci F je pravděpodobnostní model, který závisí se vstupní diferencí Δ_I a výstupní diferenci

Δ_O:

$$\Delta_I \xrightarrow{F} \Delta_O.$$

Diferenciální pravděpodobnost (DP) je počet uspořádaných dvojic se vstupní diferencí Δ_I a výstupní diferencí Δ_O vydělený celkovým počtem dvojic se vstupní diferencí Δ_I:

$$DP_F(\Delta_I, \Delta_O) = \sharp\{\{x, y\} | x \oplus y = \Delta_I \text{ a } F(x) \oplus F(y) = \Delta_O\}/2^n,$$

kde n je výstupní délka.

Pro iterativní primitivy si můžeme zadefinovat diferenciální charakteristiky jako propojené diference:

$$\begin{cases} \Delta_1 \xrightarrow{f_1} \Delta_2; \\ \Delta_2 \xrightarrow{f_2} \Delta_3; \\ \cdots \\ \Delta_{k-1} \xrightarrow{f_{k-1}} \Delta_k. \end{cases} \quad \Longrightarrow \quad \Delta_1 \xrightarrow{f_1} \Delta_2 \xrightarrow{f_2} \cdots \xrightarrow{f_{k-1}} \Delta_k.$$

Hodnota diferenciální pravděpodobnosti určuje jak je design šifry slabý. Uvažujme náhodnou funkci počtu po sobě následujících dvojic: diferenciál je stochastická proměná s binomickým rozdělením se střední hodnotou 2^{n-m}, kde n je vstupní délka a m je výstupní délka. Binomické rozdělení popisuje četnost výskytu náhodného jevu v n nezávislých pokusech, v nichž má stále stejnou pravděpodobnost. V některých případech může být aproximován pomocí Poissonova rozdělení, tedy podle rozdělení řídkých jevů, které řídí četnost jevů s velmi malou pravděpodobností výskytu. Znamená to, že zatímco je počet uspořádaných dvojic s pevnou diferencí dán hodnotou 2^n, diferenciální pravděpodobnost se pohybuje okolo 2^{-m}. Platí tedy, že diferenciál s vysokou hodnotou diferenciální pravděpodobnosti značí potenciální slabost návrhu, protože náhodné zobrazení pravděpodobně nebude mít vysokou pravděpodobnost pro stejný diferenciál.

Diferenciální pravděpodobnost i diferenciální charakteristiky je poměrně těžké spočítat. Proto se pro diferenciální charakteristiky zavádí předpokládaná pravděpodobnost (EDP), která je definována jako násobek pravděpodobností vnitřních diferenciálů. V praxi, kde se používá nezávislost mezi rundami, jsou poznatky těžko prokazatelné pro další kroky. U iterativních blokových šifer je situace ještě komplikovanější, protože se zde část klíče stává součástí následující rundy. Nejlepší způsob jak se vyhnout závisloti klíčů mají šifry střídající klíče (key-alternating cipher), které aplikují operaci XOR na vnitří stav a na rundovní klíč před použitím v rundovní funkci. Pro nezávislé klíče pak platí, že DP = EDP. Jinak je diferenciální pravděpodobnost stochastická proměnná, jejíž rozdělení má střední hodnotu ve výši EDP.

Pro n-bitovou nelineární funkci je ideální dosáhnout co nejblíže k 2^{n-1}, aby byl diferenciál uniformní. Odvození klíče pomocí diferenciální kryptoanalýzy odpovídá útoku hrubou silou. Odolnost AES vůči diferenciální kryptoanalýze je popsána v 5.3.4.. Dalším případem rezistence vůči diferenciální kryptoanalýze jsou lichá pole, protože zde neexistuje bijekce pro vstupy - výstupy pro 2 uniformitu. Lichá pole (př. GF(2^7)) lze získat aplikací kubické funkce (cubing), nebo pomocí inverze. Tyto šifry jsou sice odolné vůči lineární i diferenciální kryptoanalýze, ale výhody zcela ztrácejí proti algebraické kryptoanalýze. Proto třeba AES používá affiní zobrazení po inverzi.

Diferenciální kryptoanalýzu lze dále dělit na speciální typy podle vlastností jak se volí diference:

- *Zkrácená diferenciální kryptoanalýza (Truncated differential cryptoanalysis)* je zobecněná verze diferenciální kryptoanalýzy pro blokové šifry. Zkrácená varianta uvažuje pouze diference, které jsou určeny jen částečně. Zatímco diferenciální kryptoanalýza analyzuje diference mezi plnohodnotnými otevřenými texty. Útok předpokládá jen některé bity místo celého bloku. Útok byl s úspěchem použit na SAFER, IDEA, Twofish atd.

- *Diferenciální kryptoanalýza vyššího řádu (Higher-order differential cryptanalysis)* byla stejně jako zkrácená varianta určená pro blokové šifry. Diferenciální kryptoanalýza analyzuje diference mezi dvěma otevřenými texty, metoda vyššího řádu analyzuje diference mezi diferencemi.

- *Diferenciální kryptoanalýzu nesplnitelnosti (Impossible differential cryptanalysis)* vyvinul jako všechny výše uvedené varianty Lars Knudsen. Diferenciální kryptoanalýza sleduje diference, které se šíří šifrou s nejvyšší pravděpodobností. Varianta nesplnitenosti hledá diference, které jsou nemožné (s pravděpodobností 0) a to v některých mezistavech algoritmu šifry. Metoda ukázala slabosti šifry DEAL a vyřadila ji z možných kandidátů na AES. Dále byla úspěšně použita na IDEA, MARS, Twofish, CRYPTON atd.

- *Boomerang útok (Boomerang attack)* umožňuje použít diferenciály, které pokrývají pouze část šifry. Diferenciální kryptoanalýza požaduje po diferenciálu, aby pokrýval celou nebo téměř celou šifru.

Ukázka útoku.
Principem útoku je předpoklad, že pro danou dvojici textů dochází ke změně pouze pro určité vstupní hodnoty. Pozorováním pak může útočník odvodit možné klíčové hodnoty. Předpokládejme, že se diferenciál $1 \Rightarrow 1$ vyskytuje s pravděpodobností 4/256 (možný příklad nelineární funkce pro AES_{256}). Implikací diference v LSB (Least significant bit) pro vstup vede ke změně LSB na výstupu. Pravděpodobnost udává, že je diferenciál určen dvěma dvojicemi (čtyři různé hodnoty). Dále předpokládejme, že existuje nelineární funkce, která před vyhodnocením

aplikuje na klíč operaci XOR a hodnoty diferenciálu jsou dané dvojicemi $\{2,3\}$, $\{6,7\}$. Pokud útočník pošle hodnoty $\{4,5\}$ a obdrží správné výstupní diference, pak je klíč buď $2 \oplus 4$, nebo $6 \oplus 4$, tedy $K = \{2,6\}$.

$$0010 \oplus 0100 = 0110$$
$$0110 \oplus 0100 = 0010$$
$$0010 \oplus 0101 = 0111$$
$$0110 \oplus 0101 = 0011$$

Primitiva s modulárním sčítáním
Modulární sčítání je jedna z nejpoužívanějších operací v aplikacích, protože je se jedná o jednu z nejrychlejších operací, které jsou pro návrháře šifer dostupné. Nejvýznamější bity (Most significant bit) ve argumentech sčítání neovlivní nejméně významné bity (LSB) výsledku, což činí difůzi nevyváženou. Jako protiopatření se používají rotace a posunutí. Mnoho primitiv obsahuje pouze tři operace: sčítání (addition), rotace (rotation) a operaci XOR, jedná o tzv. *ARX promitiva*.

4.4. Kryptoanalýza kryptografických hašovacích funkcí

Kryptoanalytické metody pro kryptografické hašovací funkce souvisí s jejich požadovanými vlastnostmi.

- Odolnost vůči nalezení vzoru (Preimage resistance) - pro haný hash h je výpočetně nezvládnutelné (teoreticky nemožné) nalézt zprávu M takovou, že $h = \text{hash}(M)$. Vlastnost souvisí s konstrukcí jednosměrných hašovacích funkcí a je základem preimage útoků.

- Odolnost vůči nalezení vzoru druhého stupně (Second-preimage resistance) - pro vstupní zprávu M_1 je výpočetně nezvládnutelné nalézt zprávu M_2, kde $M_1 \neq M_2$, takovou, že $\text{hash}(M_1) = \text{hash}(M_2)$. Jedná se o tzv. slabou odolnost vůči kolizím a je základem second-preimage útoků.

- Odolnost vůči kolizím (Collision resistance) - je obtížné nalézt dvě rozdílné zprávy M_1 a M_2 takové, že $\text{hash}(M_1) = \text{hash}(M_2)$. Pokud se podaří nalézt podobnou dvojici textů, hovoří se o kryptografické hašovací kolizi. Vlastnost se nazývá silnou podmínkou vůči kolizím.

Bezkoliznost neříká, že neexistují žádné kolize. Existuje mnoho zpráv se stejným hashem. Možných zpráv je totiž obrovské množství $(1+2^1+\cdots+2^D = 2^{D+1}-1)$, ale počet otisků je omezený např. pro SHA-1 existuje 2^{160} hashů. Bezkoliznost říká, že nalezení kolize je nad dostupné výpočetní prostředky.

Horní a dolní mez pro odolnost vůči kolizím určuje narozeninový paradox, který říká, že pokud kryptografická hašovací funkce produkuje výstup N-bitů, pak útočníkovi stačí spočítat $2^{N/2}$ hašovacích operací náhodných vstupů k nalezení dvou shodných výstupů. Pokud existuje jednodušší způsob jak je útok hrubou silou, pak je kryptografická hašovací funkce považována na slabou.

4.4.1. Kolizní útoky

Kolizní útok (collision attack) je metoda hledající dva různé vstupy se stejnou hodnotou otisku, přičemž nejsou předem stanoveny vstupy ani hodnota hashe. Kolizní útoky se dělí na klasické kolizní útoky na kolizní útoky s voleným prefixem.

Klasické kolizní útoky.
Klasické kolizní útoky hledají rozdílné texty M_1 a M_2 takové, že $\text{hash}(M_1) = \text{hash}(M_2)$. Útočník nemá vliv na obsah zpráv, jsou voleny algoritmem. Metodiku postupu při kolizním útoku určuje kokrétní kryptografická hašovací funkce. Platí, že pokud se vyskytne rychlejší kryptoanalytická metoda než je narozeninový útok, pak je kryptografická hašovací funkce za slabou a prolomitelnou.

Většina nalezených kolizních textů má specifické vlastnosti jako jsou konstantní délka a nestrukturovaný obsah. Nejsou tedy přímo použitelné pro praktický útok. Výsledky lze upravit na požadovaný formát dokumentů nebo obsah protokolů a to například dynamickou konstrukcí nebezpečného obsahu. Podobné dokumenty obsahují rozdílné úseky, které pak vrací kolidující hashe podle aktuálních podmínek. Mezi příklady jsou PostScriptové dokumenty, makra v Microsoft Word. Mezi další kolidující formáty patří dokumenty obsahující obrázky PDF, TIFF. Kolize jsou zde reprezentovány v podobě indexovaných barev, které se zobrazují buď světlou barvou splývající s pozadím a nebo tmavou barvou.

Kolizní útoky s voleným prefixem.
Kolizní útoky s voleným prefixem (Chosen prefix collision attack) jsou specifické kolizní útoky, které se zaměřují na kryptografické hašovací funkce založené na Merkle-Damgård konstrukci. Pro libovolné různé dokumenty M_1 a M_2 se počítají prefixy p_1 a p_2 takové, že výsledné spojené dokumenty mají stejnou hodnotu otisku $\text{hash}(p_1 \,.\, M_1) = \text{hash}(p_2 \,.\, M_2)$, kde $.$ je operace konkatenace. Kolizní útoky s voleným prefixem jsou mnohem efektivnější než klasické kolizní útoky.

Jedním z příkladů úspěšného použití kolizního útoku s voleným prefixem je Čínský útok na MD5 viz. 7.3.4.. Nejznámější příklad útoku je zfalšovaný certifikát X.509 viz. 7.3.4..

4.4.2. Narozeninové útoky

Narozeninový útok (birthday attack) je typ útoku, který závisí na hodnotě pravděpodobnosti výskytu kolizí mezi náhodnými pokusy útočníka a na pevném stupni permutací. Narozeninový útok je založen na narozeninovém paradoxu.

Narozeninový paradox.
Narozeninový paradox je prezentován na příkladu narozenin skupiny osob. Předpokládejme, že zjišťujeme narozeniny třiceti žáků třídy a zda existuje někdo kdo se narodil ve stejný den. Žáci, kteří mají stejné narozeniny tvoří kolizi skupiny. Intuitivně se kolize vyskytuje s malou pravděpodobností. Pokud hledáme pravděpodobnost žáka narozeného v konkrétní den, pak je hodnota dána jako

$1 - (364/365)^{30} = 0,079$, tedy $7,9\%$. Nicméně je pravděpodobnost, že má student narozeniny ve stejný den jako jiný téměř 70% $(1 - 365!/((365 - n)! * 365^n))$.

4.5. Kryptoanalýza u asymetrických šifrovacích systémů

Asymetrická kryptografie je založena na existenci dvou klíčů: veřejného a tajného. Řeší zcela jiné problémy než symetrická kryptografie, protože jsou útoky na symetrické šifry postavené na odlišných předpokladech. Bezpečnost asymetrických algoritmů je založena na propracovaných matematických problémech a je v současnosti široce zkoumána. Jedním z hlavních rozdílů asymetrické a symetrické kryptoanalýzy je možnost využití poznatků získaných z veřejných klíčů.

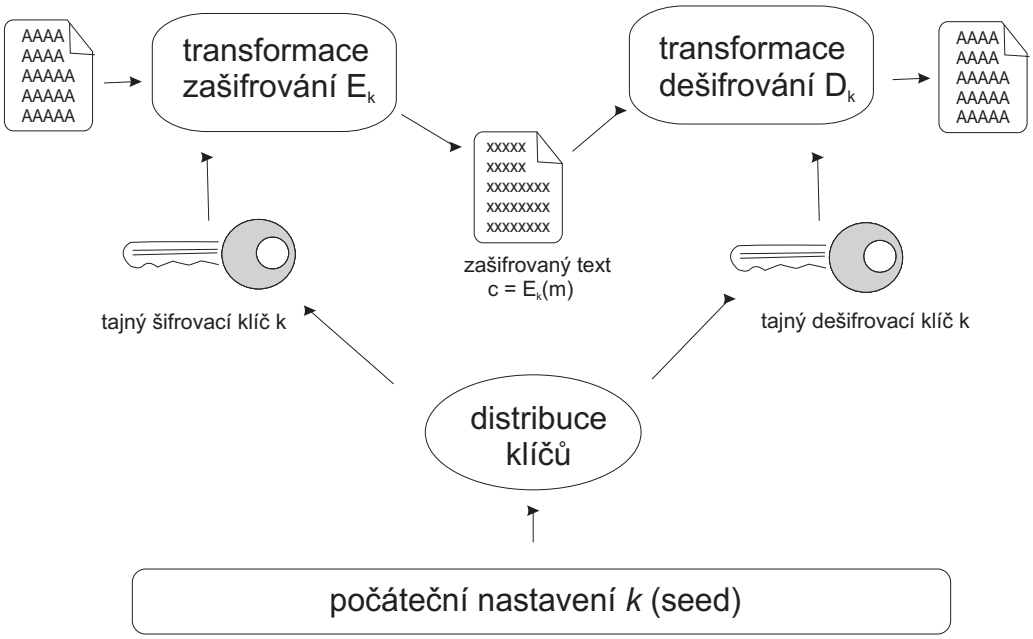
Asymetrické algoritmy jsou natolik odolné jak je dolný jejich matematický model. Kryptoanalýza se snaží najít rychlé řešení problému, které by model oslabilo. Například výměna klíčů mechanismem Diffie-Hellman je založena na složitosti řešení diskrétních algoritmů. V roce 1983 našel Don Coppersmith rychlou metodu pro hledání disktrétních algoritmů, což vedlo k potřebě používat mnohem obsáhlejší skupiny (typy skupin) algoritmů. Bezpečnost algoritmu RSA závisí na problému faktorizace velkých čísel, najde-li se rychlá metoda pro rozklad velkých čísel, nebude RSA vhodné.

Před třiceti lety bylo obtížné faktorizovat 50 místné číslo. Začátkem 21 století již nebylo ani 150 místné číslo považováno za dostatečně bezpečné pro klíče RSA. Přinesly to propracovanější metody faktorizace, ale především neustálé zrychlování výpočetních prostředků. Mooreův zákon předpovídá, že se budou rychlosti počítačů i nadále zvyšovat. Předpokladem je udržování předstihu před možnostmi kryptoanalýzy.

5. Symetrické šifrovací systémy

5.1. Druhy symetrických šifer

Na základě způsobu použití klíče pro zpracování vstupního otevřeného textu rozlišujeme dvojí druh šifer: blokovou a proudovou.



Obrázek 3. Obecný symetrický šifrovací systém.

Proudová šifra
Nechť máme dán otevřený vstupní text složený ze symbolů abecedy $|A| = t$. Proudová šifra zpracovává každý znak zvlášť a používá klíč k z množiny klíčů K pro šifrování jednotlivých znaků na vstupu.

Specifikace použití klíče.

1. Nejprve se z klíče k vygeneruje posloupnost $h(1), h(2), \ldots$
2. Každý znak vstupní posloupnosti se zašifruje speciální transformací $E_h(i)$ a to dle pozice znaku a hodnotě hesla na této pozici t.j. pro i-tou pozici se vezme hodnota hesla $h(i)$).

Bloková šifra.
Bloková šifra bere text po částech a šifruje jednotlivé bloky otevřeného textu. Blokové šifry mají předem stanovené velikosti zpracovávaných oddílů.

Specifikace použití klíče.

- Každý blok je šifrován (dešifrován) stejnou transformací $E_k(D_k)$ při použití klíče k.

5.2. Proudové šifry

Definice (Symetrická proudová šifra). Nechť A je abeceda s t symboly, M a C jsou množiny konečných řetězů nad abecedou A a K je množina klíčů. Proudová šifra se skládá z generátoru počátečního nastavení G a dvojice zobrazení E a D. Dále platí:

- Generátor G sestrojí pro každý klíč $k \in K$ posloupnost hesla $h(1), h(2), \ldots$ Prvky $h(i)$ představují libovolné substituce $E_h(1), E_h(2), \ldots$ nad abecedou A.

- Zobrazení E resp. D přiřazuje každému k z K transformaci zašifrování E_k resp. transformaci dešifrování D_k.

- Zašifrování otevřeného textu $m \in M$, kde $m = m(1), m(2), \ldots,$ odpovídá zápisu: $c(1) = E_h(1)(m(1)), c(2) = E_h(2)(m(2)), \ldots$

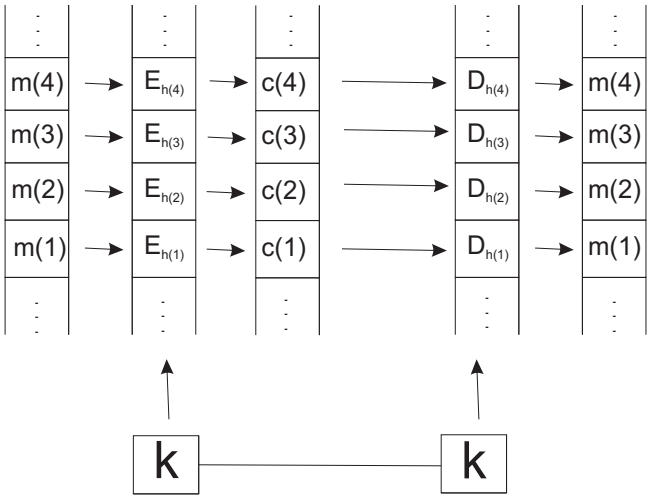
- Dešifrování šifrovaného textu $c \in C$, kde $c = c(1), c(2), \ldots,$ odpovídá zápisu: $m(1) = D_h(1)(c(1)), m(2) = D_h(2)(c(2)), \ldots$

Poznámka. Proudové šifry jsou příkladem tzv. *heslových systémů*. Posloupnost hesla $h(1), h(2), \ldots$ se nazývá *proud hesla*.
V případě, že se začne proud klíče od určitého místa opakovat, jedná se o periodické heslo a celkově hovoříme o *periodické šifře*. Příkladem periodické proudové šifry je Vigenerova šifra.

5.2.1. Vlastnosti proudových šifer

Proudové šifry se využívají u tzv. *linkových šifrátorů* a to v situaci, kdy se do komunikačního kanálu pravidelně nebo nepravidelně předávají jednotlivé znaky, které je potřeba ihned převádět a není možné čekat na celý blok.
Příkladem může být zabezpečená terminálová komunikace jako je vstup z klávesnice jednoho uživatele a výstup na monitor druhé strany. Druhým příkladem je situace, kdy šifrovací médium disponuje omezenou pamětí.

Výhodou proudových šifer je, že záměna znaku v komunikačním kanálu postihne pouze malou část celkové dešifrované informace, hovoří se o tzv. *propagaci chyby*. Pokud tedy nastane chyba na jednom šifrovaném znaku, projeví se jen na odpovídajícím znaku otevřeného textu, zatímco u blokových šifer má změna znaku vliv na celý blok následujících znaků.



Obrázek 4. Princip proudových šifer.

5.2.2. Synchronní a asynchronní proudové šifry

Pokud proud hesla (proud klíče, key-stream) nezávisí na otevřeném ani šifrovaném textu, hovoříme o *synchronních proudových šifrách*. Nevýhodou je nutnost synchronizace mezi komunikujícími stranami, protože výpadek jednoho šifrovaného znaku ovlivní po dešifrování celý následující otevřený text.
Nastane-li chyba v komunikačním kanálu, kdy jeden nebo více znaků přibude resp. vypadne, dojde ke ztrátě synchronizace proudu hesla a šifrovaného textu.

Princip šifrování.
Při vytváření šifry se použije generátor, který pro tajný klíč sestrojí posloupnost hesla $h(1), h(2), \ldots$. Samotné šifrování lze popsat jednoduše zápisem $c(i) = E_h(i)(m(i))$ a dešifrování $m(i) = D_h(i)(c(i))$. Většinou se jedná u $E_h(i)$ a $D_h(i)$ o prosté binární sčítání.

5.2.3. Princip současných proudových šifer

Současné moderní šifry pracují nad abecedou $A = \{0, 1\}$. Binární sčítání (sčítání modulo 2) se označuje symbolem $\oplus$ (operace *xor*).

Použitá substituce $E_h(i)$ nad otevřeným bitem abecedy $m(i)$ je transformace $E_h(i)(m(i)) = m(i) + h(i)$ nebo $E_h(i)(m(i)) = m(i) + h(i) + 1$, z důvodu reprezentace transformace stačí pouze $E_h(i)(m(i)) = m(i) + h(i)$.

Operace sčítání a odčítání jsou shodné, jedná se vlastně o diferenci bitů, proto jsou šifrování a dešifrování stejné transformace. Z principu symetrických šifer platí, že musí mít příjemce i odesilatel stejný klíč (heslo).

Hesla používaná pro transformace se mohou vytvářet zcela náhodně (Vermanova šifra), nebo s použitím generátoru tedy deterministicky šifrovacím algoritmem ze zadaného šifrovacího klíče.

Poznámka. Generátor bývá nesprávně nazýván přímo šifrovacím algoritmem, přestože je šifrovací algoritmus binární sčítání.

5.2.4. Vermanova šifra

Vermanova šifra je jednoduchou ukázkou symetrických proudových šifer. Princip šifrování spočívá ve faktu, že je použité jednorázové heslo (one-time pad) stejně dlouhé jako otevřený text. Po zašifrování textu se klíč odstraní a není již znovu použit pro následující šifrování.

Vermanovu šifru vynalezl ve dvacátých letech minulého století příslušník americké armády Joseph Mauborgn, i když jméno získala po držiteli patentu Gilbertu Vermanovi. Verman je autorem šifrovacího zařízení pro ochranu telegrafických zpráv, jehož princip byl založen na binárním načítání náhodné posloupnosti klíče na otevřený text. Klíčová posloupnost se generovala zcela náhodně a klíč byl po zašifrování ničen.

Příklad.

```
ODESÍLATEL
-------
zpráva:  0 0 1 0 1 1 0 1 0 1 1 1 ...
heslo:   1 0 0 1 1 1 0 0 1 0 1 1 ...
XOR      --------------------------
šifra:   1 0 1 1 0 0 0 1 1 1 0 0 ...

PŘÍJEMCE
---------
šifra:   1 0 1 1 0 0 0 1 1 1 0 0 ...
heslo:   1 0 0 1 1 1 0 0 1 0 1 1 ...
XOR      --------------------------
zpráva:  0 0 1 0 1 1 0 1 0 1 1 1 ...
```

Absolutní bezpečnost Vermanovy šifry

1. *Požadavek:* Pro ověření vlastnosti absolutní bezpečnosti je nutné ukázat, že se znalost otevřeného textu nezmění a to ani přijmutím libovolného množství textu zašifrovaného.

2. *Předpoklad:* Předpokládejme, že máme dán i-tý znak šifrovaného textu $c(i)$. Dále nechť $o(i)$ a $h(i)$ jsou po řadě i-tý bit vstupního textu a i-tý bit hesla. Uvažujme, že $P\{o(i) = 0\} = P\{c(i) \oplus h(i) = 0\} = P\{c(i) = h(i)\}$. Platí, že je tento zápis roven

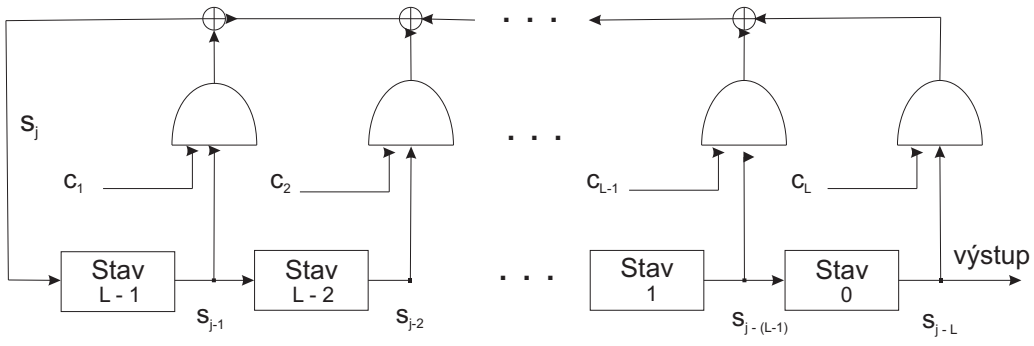
- $P\{h(i) = 1\}$, pokud $c(i) = 0$.
- $P\{h(i) = 0\}$, pokud $c(i) = 1$.

3. *Řešení:* Platí, že $P\{h(i) = 0\} = P\{h(i) = 1\} = 1/2$. Tedy v obou případech je výraz roven $1/2$ a celkově dostáváme, že $P\{o(i) = 1\} = 1/2$. Duálně pro $P\{o(i) = 0\} = 1/2$.

4. *Závěr:* Závěr je takový, že hodnota $P\{o(i) = 1\} = P\{o(i) = 0\} = 1/2$ nezávisí na hodnotě $c(i)$. Vermanova šifra splňuje podmínku kladenou na absolutně bezpečnou šifru, neboť šifrovaný text nenese žádnou informaci o otevřeném textu.

5.2.5. Zpětnovazebné registry

Zpětnovazebné registry (Feedback shift registers - FSR) jsou základem mnoha generátorů klíčů. Aplikace založené na FSR slouží například ke generování náhodných čísel, pseudonáhodných šumů atd.
Nejznámější z FSR jsou *lineární zpětnovazebné registry* (LFSR). Důvodem jejich významnosti v oblasti generátorů klíčů je snadná hardwarová a softwarová realizace a také schopnost generovat rozsáhlé periodické heslové sekvence. Vygenerované heslové posloupnosti mají dobře zkoumatelné statistické vlastnosti. Každý registr se skládá z konečné posloupnosti *stavů* uchovávající binární hodnotu 0 nebo 1.
V této kapitole si nejprve představíme LFSR a jejich vlastnosti, dále pak algoritmus *Berlekamp-Massey*, který slouží k výpočtu LFSR a nakonec nelineární zpětnovazebné registry (NFSR).



Obrázek 5. Lineární zpětnovazebný registr (LFSR) délky L.

Lineární zpětnovazebné registry (LFSR).
LFSR je zpětnovazebný registr jehož vstupní bit odpovídá hodnotě funkce předchozích stavů. Použitá funkce je lineární a jde o aplikaci operace xor na hodnoty

některých bitů uchovaných ve vybraných stavech registru.
Každý registr má zadané vstupní inicializační hodnoty (seed). Výpočet výstupní sekvence je deterministický a z důvodu konečnosti registrů je sekvence periodická. LFSR je vhodně navržený zpětnovazebný registr a je schopen produkovat náhodnou výstupní sekvenci bitů s velkou periodou.

Definice.
Lineární zpětnovazebný registr LFSR délky L má L stavů (stages), které jsou očíslovány $0, \ldots, L-1$. Každý ze stavů uchovává jeden bit 0 nebo 1 a má jeden vstup a výstup. LFSR má časovač, který řídí přesuny dat. Během jedné jednotky času jsou realizovány následující operace:

- obsah stavu 0 je výstupní a formuje část výstupní posloupnosti.
- obsah stavu i se přesune do stavu $i-1$ a to pro všechna $i \in \langle 1, L-1 \rangle$.
- nový obsah stavu $L-1$ se nazývá tzv. zpětnovazebný bit s_j (feedback bit) a jeho hodnota je výsledkem aplikace operace xor na všechny předchozí hodnoty stavů $0, \ldots, L-1$ modulo 2.

Obrázek 5. představuje základní strukturu LFSR. Každé c_i má hodnotu 0 resp. 1. Uzavřené polokruhy jsou tzv. AND brány (AND gates). Hodnota zpětnovazebného bitu s_j je rovna sumě všech takových hodnot stavů i, kde $i \in \langle 1, L-1 \rangle$ a $c_{L-1} = 1$.

Definice.
LFSR můžeme označit uspořádanou dvojicí $\langle L, C(D) \rangle$, kde L určuje počet stavů registru (délku) a $C(D) = 1 + c_1 D + c_2 D^2 + \cdots + c_L D^L \in \mathbb{Z}_2\langle D \rangle$ je konečný polynom. LFSR se nazývá *nesingulární*, jestliže je stupeň $C(D)$ roven L. Jestliže je počáteční obsah stavu i roven $s_i \in \{0, 1\}$, pak je $[s_{L-1}, \ldots, s_1, s_0]$ *inicializačním vektorem* LFSR.

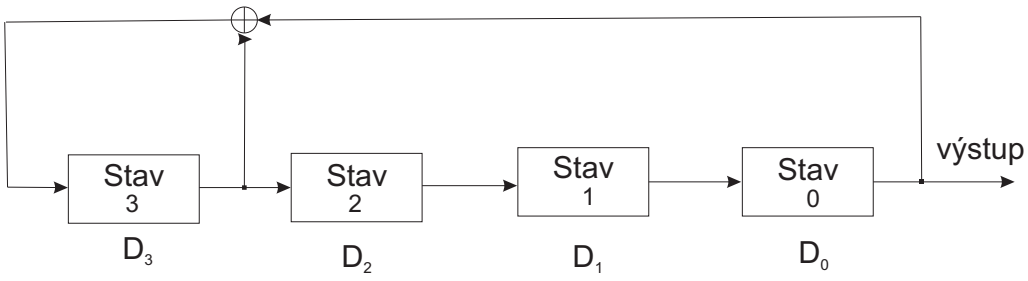
Jestliže je $[s_{L-1}, \ldots, s_1, s_0]$ inicializační vektor LSFR na obrázku 5., pak jsou hodnoty výstupní sekvence $s = s_0, s_1, \ldots$ jednoznačně určeny následující rekurzí:

$$s_j = (c_1 s_{j-1} + c_2 s_{j-2} + \cdots + c_L s_{j-L}) \, (\mod 2) \text{ pro všechna } j \geq L.$$

Výstupní sekvence tabulky 1. je s = 0101100100011110. Maximální perioda = $2^4 - 1 = 15$.

Příklad.
Předpokládejme, že máme dán LFSR z obrázku 6. $\langle 4, 1 + D + D^4 \rangle$. Pokud by byl inicilizační vektor $[0, 0, 0, 0]$, pak by byla výstupní sekvence nulová. Předpokládejme, že bude inicializační vektor $[1, 0, 1, 0]$. Následující tabulka 1. ukazuje hodnoty jednotlivých stavů D_3, D_2, D_1, D_0 v čase t.

Periodičnost výstupních sekvencí u LFSR ovlivňuje volba polynomu $C(D)$. Výstupní sekvence je periodická pouze tehdy a jen tehdy, pokud je stupeň $C(D)$ roven L, kde L je výše uvedená délka registru. Dále jestliže je $C(D) \in \mathbb{Z}_2$:



Obrázek 6. LFSR $\langle 4, 1 + D + D^4 \rangle$.

t	D_3	D_2	D_1	D_0
0	1	0	1	0
1	1	0	0	1
2	1	0	0	0
3	0	1	1	1
4	1	1	0	1
5	0	1	0	0
6	1	1	0	0
7	1	0	1	1

t	D_3	D_2	D_1	D_0
8	0	1	1	0
9	0	0	1	0
10	1	1	1	0
11	0	1	0	1
12	0	0	1	1
13	0	0	0	1
14	1	1	1	1
15	1	0	1	0

Tabulka 1. Výpočet LFSR $\langle 4, 1 + D + D^4 \rangle$ s inicializačním vektorem $[1, 0, 1, 0]$.

- ireducibilní, pak je perioda výstupní sekvence LFSR rovna nejméně nějakému N a platí, že $C(D)/1 + D^N$ a $N/2^L - 1$.
- primitivní, pak je perioda výstupní sekvence LFSR rovna nějakému N, kde $N = 2^L - 1$.

Je-li $C(D) \in \mathbb{Z}_2$ primitivní polynom stupně L, pak je $\langle L, C(D) \rangle$ *maximálně dlouhý* LFSR, výstupní sekvence je nenulová a nazývá se *m-sekvence*.
Pro posouzení náhodnosti požadovaných pseudonáhodných sekvencí existují Golombovi pseudohonáhodné postuláty (Golombs randomness postulates), které definují postačující podmínky na posuzované sekvence.

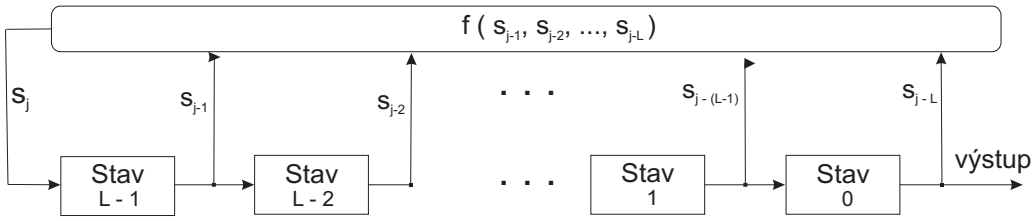
Další vlastnost, která se u výstupních m-sekvencí zkoumá je jejich *lineární složitost*, která výstup ovlivňuje. Cílem je sestrojení vhodného $\langle L, C(D) \rangle$, který bude vykazovat optimální výsledky. Lineární složitost je důležitá vlastnost výstupních sekvencí, protože popisuje vlastnosti periodičnosti a chování sekvencí a jejich výpočet v čase.

Algoritmus Berkamp-Massey
Algoritmus Berkam-Massey je efektivním algoritmem pro určení lineární složitosti

konečných sekvencí s^n délky n. Úkolem algoritmu je na základě zadané sekvence požadované délky určit neznámý LFSR, který ji generuje. Vstupem je sekvence s^n a jako výstup je lineární složitost L. Výpočet probíhá postupnou iterací subsekvencí s^n, která je přiblížením k požadovanému výsledku a udává nám i tvar použitého lineárního polynomu.

Příklad.
Předpokládejme, že je $s^n = 0, 0, 1, 1, 0, 1, 1, 1, 0$ pro $n = 9$. Nalezená lineární složitost je rovna 5 a algoritmus určil tvar polynomu $1 + D^3 + D^5$. Hledaný LFSR generující požadovanou sekveci s^n je určen uspořádanou dvojicí $\langle 5, 1 + D^3 + D^5) \rangle$.

Nelineární zpětnovazebné registry
Nelineárnost je prostředkem jak se dají odstranit nedostatky LFSR. Pro skupinu nelineárních zpětnovazebných registrů (NFSR) je nutné zavést pojem *Booleovské funkce n proměných.*



Obrázek 7. Obecný zpětnovazebný registr (FSR) délky L.

Definice.
Funkce, která má n binárních vstupů a pouze jeden binární výstup je nazývána Booleovskou funkcí n proměných. Existuje 2^{2^n} různých Booleovských funkcí n proměných.

Definice.
Obecný zpětnovazebný registr (FSR) 7. délky L je složen z L stavů, které jsou očíslovány $0, \ldots, L-1$. Každý ze stavů uchovává jeden bit a má jeden vstup a výstup. FSR má časovač, který řídí přesuny dat. Během jedné jednotky času jsou realizovány následující operace:

- obsah stavu 0 je výstupní a formuje část výstupní posloupnosti.
- obsah stavu i se přesune do stavu $i-1$ a to pro všechna $i \in \langle 1, L-1 \rangle$.
- nový obsah stavu $L-1$ se nazývá tzv. zpětnovazebný bit (feedback bit) $s_j = f(s_{j-1}, s_{j-2}, \ldots, s_{j-L})$, kde f tzv. zpětnovazebná funkce (feedback function) je Booleovská funkce a s_{j-i} je obsah předchozího stavu $L-i$ pro $1 \leq i \leq L$.

Pokud je zpětnovazebná funkce f lineární, pak se jedná o LFSR. Jestliže je $[s_{L-1}, \ldots, s_1, s_0]$ inicializační vektor z obrázku 7., potom můžeme vypočítat výstupní sekvenci $s = s_0, s_1, \ldots$ ze vztahu:

$$s_j = f(s_{j-1}, s_{j-2}, \ldots, s_{j-L}) \text{ pro všechna } j \geq L$$

Proudové šifry založené na LFSR
LFSR jsou široce používány v oblasti heslových generátorů a to díky svým vlastnostem jako je snadná hardwarová implementovatelnost, generování sekvencí s velikou periodou atd.
Vedle zmíněných výhod je ale možné pomocí algoritmu Berlekamp-Massey odvodit použitou heslovou sekvenci s pomocí nějaké subsekvence t sekvence s, která má délku nejméně $n = 2L$. Toho je možné v mnoha případech dosáhnout tehdy, pokud útočník zná jak otevřený, tak i šifrovaný text, protože lze korenspondující heslovou posloupnost získat jako výsledek $m_i \oplus c_i$, $1 \leq i \leq n$.

LFSR se z výše uvedených důvodů nepoužívá v jednoduché lineární podobě. Existuje několik metod jak se dá lineárnost u LFSR změnit:

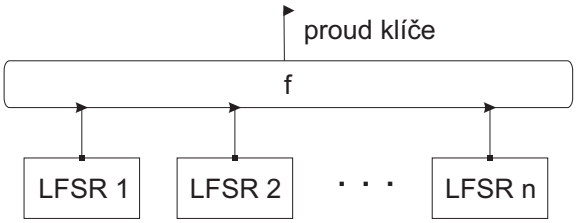
- použití nelineární kombinační funkce na výstup několika LFSR.
- použití nelineární filtrovací funkce na výstup z jednoho LFSR.
- použití výstupu jednoho (více) LFSR na kontrolu času druhého (více) LFSR.

Je nutné zmínit, že proudové šifry založené na LFSR vykazují pouze postačující podmínku kryptografické bezpečnosti, ale i tak se jedná o výpočetně bezpečné heslové generátory.

Nelineární kombinace generátorů
Jedna z technik jak změnit lineárnost je použití několika LFSR zapojených paralelně. Výstupní heslová sekvence je generována jako nelineární funkce f z výstupů zapojených LFSR viz. 8. Uvedený způsob se nazývá nelineární kombinace generátorů. Použitá funkce musí splňovat podmínky kladené na odolnost vůči kryptoanalýze.

Výsledek m různých proměnných je nazýván m-té pořadí proměných (m^{th} order product). Každou booleovskou funkci $f(x_1, x_2, \ldots, x_n)$ lze zapsat jako sumu m-tého pořadí modulo 2, pro $1 \leq m \leq n$. Jedná se o tzv. algebraickou normální formu. *Nelineární pořadí* funkce f je rovno pořadí termů v příslušné algebraické normální formě.

Příklad.
Pro boolovskou funkci $f(x_1, x_2, x_3, x_4, x_5) = 1 \oplus x_2 \oplus x_3 \oplus x_4x_5 \oplus x_1x_3x_4x_5$ je nelineární pořadí rovno 4.

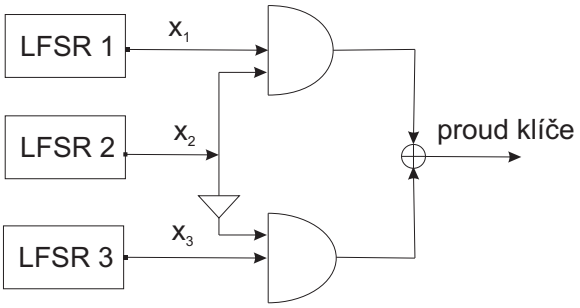


Obrázek 8. Nelineární kombinace generátorů.

Příklad *(Geffův generátor)*
Geffův generátor (Geffe generator) viz. obrázek 9. je jednoduchou ukázkou nelineární kombinace generátorů. Je složen ze tří maximálně dlouhých LFSR, jejichž délky jsou L_1, L_2, L_3. Jeho nelineární funkce je dána zápisem:

$$f(x_1, x_2, x_3) = x_1x_2 \oplus (1 + x_2)x_3 = x_1x_2 \oplus x_2x_3 \oplus x_3.$$

Heslová posloupnost je generována s periodou $2^{L_1} * 2^{L_2} * 2^{L_3}$. Lineární složitost je pak $L = L_1L_2 + L_2L_3 + L_3$.



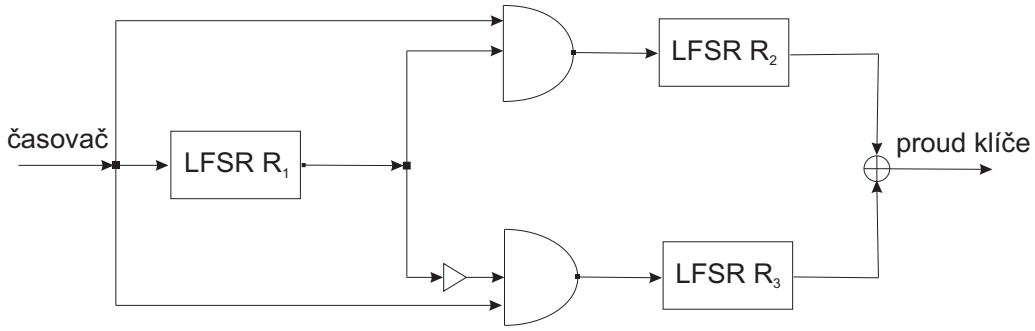
Obrázek 9. Geffův generátor.

Geffův generátor je kryptoanalyticky prolomitelný, protože informace o stavech LFSR1 a LFSR3 ústí do jedné výstupní sekvence. Geffův generátor je také náchylný vůči korelačním útokům, protože je jeho korelační pravděpodobnost malá. Existují však i jiné nelineární kombinace generátorů jako je např. *Sumační generátor* (Summation generator), které jsou mnohem odolnější i proti korelačním útokům a lze je použít v praxi.

Nelineární filtrovací generátory
Nelineární filtrovací generátor (Nonlinear filter generator) generuje sekvenci klíče jako nelineární funkci stavů jednoho LFSR. Použitá funkce f se nazývá *filtrovací funkce*.

Časově řízené generátory
Nelineární kombinace generátorů i nelineární filtrovací generátory pracují se stejným globálním časem (používají stejný časovač při přesunech mezi stavy). Základní myšlenka nelinearity časově řízených generátorů (Clock-controlled generators) spočívá ve faktu, že výstup jednoho LFSR ovlivňuje časovač druhého LFSR, tedy že řízení přesunů dat druhého LFSR závisí na vlastnostech výstupu prvního LFSR. Nepravidelnost přesunů pak zamezuje možnost výpočtu použitého LFSR pro získanou sekvenci. Uvedeme si dva základní zástupce časově řízených generátorů:

- **Generátor střídavých kroků.**
  Generátor střídavých kroků (The alternating step generator) používá LFSR R_1 pro řízení kroků LFSR R_2 a R_3. Celkový výstup tvoří použití xor operace na výstupní sekvence R_2 a R_3 viz. obrázek 10.



Obrázek 10. Generátor střídavých kroků.

Algoritmus (Generátor střídavých kroků).
Souhrn: Kontrolní LFSR R_1 střídavě řídí kroky (časuje, příkaz posuvu v registru) LFSR R_2 a R_3.
Výstup: sekvence vzniklá aplikace operace xor na výstupy R_2 a R_3.
Následující kroky se opakují, dokud se nevyčerpá výstupní sekvence z LFSR R_1.

1. Register R_1 je načasován (posunem získáme výstupní bit).
2. Jestliže je na aktuální bit sekvence R_1 logická 1, pak: R_2 je načasován (posun); R_3 není načasován, ale na výstup se pošle opakovaně jeho předchozí bit. (Pro první jednotku času (cyklu), se jako předchozí bit R_3 zvolí 0.)
3. Jestliže je na aktuální bit sekvence R_1 logická 0, pak: R_3 je načasován (posun); R_2 není načasován, ale na výstup se pošle opakovaně jeho

předchozí bit. (Pro první jednotku času (cyklu), se jako předchozí bit R_2 zvolí 0.)

4. Aplikace operace XOR na výstupní bity R_2 a R_3; Výsledek tvoří část výstupní sekvence.

Více formálněji: Nechť máme dány LFSR R_1, R_2 a R_3, jejichž výstupní sekvence jsou $a_0, a_1, a_2, \ldots$, $b_0, b_1, b_2, \ldots$ a $c_0, c_1, c_2, \ldots$. Nastavíme $b_{-1} = c_{-1} = 0$. Algoritmus produkuje výstupní sekvenci $x_0, x_1, x_2, \ldots$, kde $x_j = b_{t(j)} \oplus c_{j-t(j)-1}$ a $t(j) = (\sum_{i=0}^{j} a_i) - 1$ pro všechna $j \geq 0$.

Příklad (Generátor střídavých kroků).
Nechť máme dán generátor střídavých kroků s komponentami LFSR $R_1 = \langle 3, 1 + D^2 + D^3 \rangle$, $R_2 = \langle 4, 1 + D^3 + D^4 \rangle$, $R_3 = \langle 1, 1 + D + D^3 + D^4 + D^5 \rangle$. Předpokládejme, že jsou jejich inicializační vektory $[0, 0, 1]$, $[1, 0, 1, 1]$ a $[0, 1, 0, 0, 1]$. Výstupní sekvence R_1 má periodu 7 s cyklem:

$$a^7 = 1, 0, 0, 1, 0, 1, 1.$$

Výstupní sekvence R_2 má periodu 15 s cyklem:

$$b^{15} = 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0.$$

Výstupní sekvence R_3 má periodu 31 s cyklem:

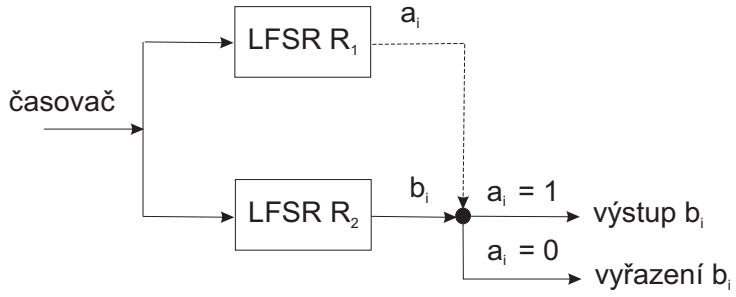
$$c^{31} = 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0.$$

Výsledná výstupní sekvence generátoru střídavých kroků je sekvence:

$$x = 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, \ldots$$

- **Generátor zkracující výstup**
  Generátor zkracující výstup (The shrinking generator) je novějším členem časově řízených generátorů. Je mnohem jednodušší než předchozí generátor a je i mnohem rychlejší a dává lepší výsledky. Je založen na principu, kdy se s pomocí zvoleného LFSR R_1 zkracuje výstup výchozího LFSR R_2. Jako výsledek dostaneme kratší část původní výstupní sekvence LFSR R_2. Jde o nepravidelné zmenšování výstupní posloupnosti, což zaručí odolnost vůči případné zpětné analýze a podmínku nelinearity.

  Algoritmus (Generátor zkracující výstup)
  Souhrn: Výchozí LFSR R_1 řídí výstup LFSR R_2.
  Následující kroky se opakují, dokud nebude nově vytvořená výstupní posloupnost požadované délky (heslová posloupnost).



Obrázek 11. Generátor zkracující výstup.

1. Registry R_1 a R_2 jsou načasovány (posun).
2. Je-li výstupní bit R_1 logická jedna, bude výstupní bit R_2 přidán k výstupní heslové posloupnosti (formuje výchozí heslovou posloupnost).
3. Je-li výstupní bit R_1 logická nula, bude výstupní bit R_2 vyřazen.

Více formálněji: Nechť máme dány LFSR R_1 a R_2, jejichž výstupní sekvence jsou $a_0, a_1, a_2, \ldots$ a $b_0, b_1, b_2, \ldots$. Algoritmus produkuje výstupní sekvenci $x_0, x_1, x_2, \ldots$, kde $x_j = b_{i_j}$ pro všechna $j \geq 0$, i_j je pozice j-té jedničky v sekvenci $a_0, a_1, a_2, \ldots$.

Příklad (Generátor zkracující výstup).
Nechť máme dán generátor zkracující výstup s komponentami LFSR $R_1 = \langle 3, 1 + D + D^3 \rangle$, $R_2 = \langle 5, 1 + D^3 + D^5 \rangle$. Předpokládejme, že jsou jejich inicializační vektory $[1, 0, 0]$ a $[0, 0, 1, 0, 1]$. Výstupní sekvence R_1 má periodu 7 s cyklem:

$$a^7 = 0, 0, 1, 1, 1, 0, 1.$$

Výstupní sekvence R_2 má periodu 31 s cyklem:

$$b^{31} = 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0.$$

Výsledná výstupní sekvence generátoru střídavých kroků je sekvence:

$$x = 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, \ldots$$

5.2.6. Algoritmicky generované proudové šifry

Původní přístup náhodného generování hesla a následnou distribuci jeho nosičů mezi komunikující strany vystřídala metoda algoritmického generování hesla. Ke generování se nejprve používaly mechanické, elektrické a později elektronické šifrovací mechanismy, které heslo vytvářely. Následovala distribuce v podobě šifrovacího klíče sloužícího jako inicializační nastavení těchto strojů.

Inicializační vektor (IV)

Z důvodu snížení potřeby časté změny klíče se zavedl *princip náhodně měnícího se inicializačního vektoru (IV)*. IV je blok bitů, který zaručuje proudovým i blokovým 5.3. šifrám generování nové heslové posloupnosti, i když použijí stejný tajný klíč. Pokud šifra ke generování hesla IV nevyužívá, pak musí opakovaně klíč zdlouhavě přepočítávat. Výběr IV probíhá náhodně pro každou novou zprávu a je většinou přenášen otevřenou formou. Šifrovací algoritmus se nastavuje na základě dodaného inicializačního vektoru do zcela náhodného počátečního stavu.

Velikost IV závisí na konkrétním algoritmu (protokolu) a udává ji buď velikost bloku, nebo velikost šifrovacího klíče. Příjemce musí znát IV, aby mohl zprávu dešifrovat. Toho lze docílit několika způsoby: IV je přenášen společně se šifrovaným textem, nebo je vypočítán (inkrementačně), nebo je odvozen na základě předepsaných podmínek např. aktuální čas, nebo kombinací více možností. Při generování IV je nutné počítat s problémy jako jsou kolize při náhodném generování, nebo otázka bezpečnosti jako je např. u inkrementačního výpočtu IV podmínka nonce (number used once) - ochrana vůči opakovanému použití (replay attack).

Inicializační vektor je rozdílně implementován u blokových viz. 5.3.5. a proudových šifer. Proudové šifry načítají IV na tajnou klíčovou posloupnost. Po přípravě klíčové posloupnosti proběhne několik šifrovacích rund. Počet rund se stejným klíčem by měl být co nejmenší. Problém nevhodného nízkého počtu rund souvisí jednak s možnou ztrátou entropie, ale i s obtížností konstrukce jedinečné šifry. Ukázkou problematického použití IV u proudových šifer je útok založený na principu souvisejících klíčů (Related-key attack) viz. 4.2.1., který lze vidět u protokolu WEP (Wired Equivalent Privacy)(viz. [19]). Dále jsou známy kryptoanalytické útoky na principu obnovy klíčů, které se provádí pomocí statistické analýzy IV viz. [20].

Mezi významné algoritmicky generované proudové šifry patří:

1. **Proudová šifra RC4**
   RC4 je příkladem proudové šifry, která nepoužívá princip inicializačního vektoru IV, a proto se musí tajný klíč generovat na začátku každého spojení. Komunikace spočívá ve faktu, že si musí účastníci předat tajný klíč pomocí nějaké asymetrické metody.
   Šifra byla vytvořena prof. Rivestem v roce 1987 a je jedna z proudových šifer patřící mezi standardy v rámci internetu.
   Její princip není oficiálně vydán, ale byl popsán neznámým hackerem, který jej získal disassemblováním z programu BSAFE společnosti RSA. Z důvodu ochrany licence se hovoří o šifře "Arcfour".
   RC4 obsahuje několik protokolů a standardů, jakými jsou například SSL a umožňuje volit délku klíče a to nejčastěji na 40 a 128 bitů.

   Popis RC4

- Substituce z klíče:
  Klíč se u šifry RC4 využívá k vytvoření tajné substituce bajt za bajt $0, \ldots, 255 \to 0, \ldots, 255$. Pro substituci se sestrojí tabulka S a na jejím základě se s pomocí konečného automatu generují jednotlivé bity hesla $h(0), h(1), \ldots$, které se pomocí operace *xor* aplikují na otevřený resp. šifrovaný text.
- Převod náhodné posloupnosti:
  Posloupnost r, kterou získáme bude mít 256 členů náhodných čísel $r(0), \ldots, r(255)$. Tato posloupnost může obsahovat některé členy několikrát resp. se některá čísla z 0 až 255 nemusí vyskytovat vůbec. Úkolem je vytvořit novou posloupnost P, která již bude permutací na množině $0, \ldots, 255$.
  Naplníme P identickou permutací, pro níž platí: $P(i) = i$, pro $i = 0, \ldots, 255$. Vezmeme si původní náhodnou posloupnost r a na jejím základě budeme prohazovat členy v P. Míchání provádíme pro $i = 0, \ldots, 255$. V každém kroku vyměníme prvky na pozicích i a $r(i)$, tedy prohodíme prvky $P(i)$ a $P(r(i))$. Protože během míchání projdeme všechny pozice, dojde k permutaci všech prvků v P. Výsledná posloupnost P je závislá na výchozí posloupnosti r.

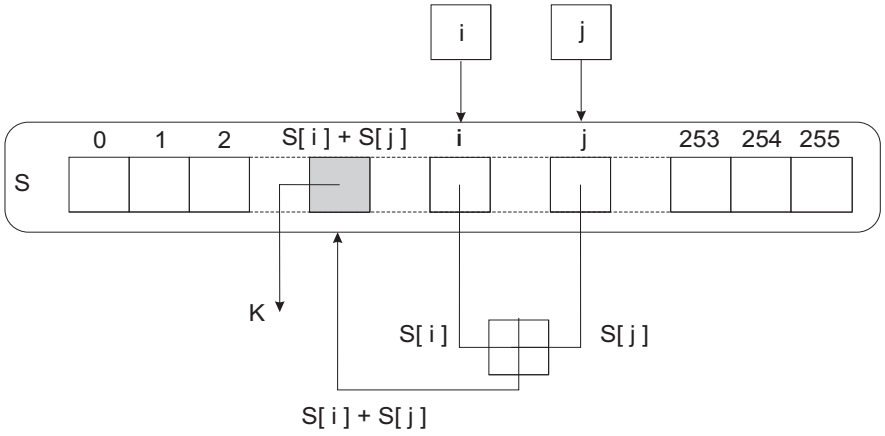
0	1	2	3		45		122		254	255
45	1	2	3		0		122		254	255
45	122	2	3		0		1		254	255
45	122	254	3		0		1		2	255

Obrázek 12. Vytvoření náhodné posloupnosti u RC4.

Heslo u RC4
Myšlenka aplikovaná v předchozím odstavci se používá i při tvorbě hesla. Operace jsou prováděny s bajty a sčítání je s modulo 256. Používají se dvě proměnné: index i, který slouží jako počítadlo od 0 do 255 a cyklicky dále; klíčově závislé j jako náhodný index. Generování heslové posloupnosti lze popsat následujícím cyklem:

```
i = j = 0;
for (k = 0; k < n; k++)
{
 i = i + 1;
```

```
 j = j + S(i);
 prohod_hodnoty(S(i),S(j));
 h(k) = S(S(i) + S(j));
}
```



Obrázek 13. Tvorba hesla u RC4.

2. **Proudová šifra A5**

Proudová šifra A5 se používá pro zabezpečení hovorů mezi mobilním zařízením a stanicí sítě GSM. Stejně jako u RC4 nebyla publikována. Nejznámější jsou používané varianty A5/1, A5/2 a bloková varianta A5/3. A5 je příkladem proudové šifry, která binárně načítá na bity otevřeného textu heslo.
V současné době se z proudových verzí používá pouze varianta A5/1, protože je mnohem odolnější vůči případné kryptoanalýze.

Popis A5
Heslo se používá v délce 228 bitů, ze kterého je 114 bitů určeno pro šifrování příchozích hovorů a 114 bitů pro šifrování odesílaných hovorů. Dané úseky se nazývají bursty a jsou číslovány 22 bitovými rámci TDMA.
Samotné šifrování hovorů je specifikované SIM kartou telefonu a to sice tajným klíčem K_i, který vygeneruje na základě 128 bitové náhodné výzvy RAND sítě GSM klíč K_c závislý jak na K_i tak i na hodnotě RAND. Hodnota klíče K_c je pak prostřednictvím A5 smíchána s rámci TDMA, čímž vznikne počáteční naplnění registrů A5. Před samotným vygenerováním hesla proběhne 99 ($A5/1$) resp. 100 ($A5/2$) kroků naprázdno, což zaručí použití jiného hesla na každý nový rámec. Hlavní rozdíl mezi A5/1 a A5/2 je ten, že druhá varianta používá registry R4.

5.3. Blokové šifry

Blokové šifry lze zjednodušeně popsat pomocí dvou algoritmů - algoritmu šifrování a algoritmu dešifrování. Jejich vstupem je n bitový blok textu a klíč délky k bitů. Výstupem je n bitový blok textu. Operace šifrování a dešifrování jsou obecně vzájemně inverzní operace.

Délka bloku textu je typicky 64 nebo 128 bitů (současnost). Klíč je v současnosti většinou volen v délce 64 až 256 bitů. Na délce klíče závisí odolnost šifry proti kryptoanalýze, doporučená délka klíče je minimálně 80 bitů.

Většina blokových šifer je iteračního charakteru tzv. iterační blokové šifry. Šifry vznikají opakovanou aplikací jednoduché funkce na vybraný blok textu. Jedna iterace je nazývána runda, typicky jich bývá 4 až 32. Použitá funkce se nazývá rundovní funkce.

Definice (Symetrická bloková šifra). Nechť A je abeceda s t symboly, M a C jsou množiny konečných řetězců nad abecedou A a K je množina klíčů. Bloková šifra je šifrovací systém (M, C, K, D, E), kde pro každé $k \in K$ určuje dvojice zobrazení E a D transformaci zašifrování E_k a dešifrování D_k s vlastnostmi:

- šifrování bloků otevřeného textu $m(1), m(2), m(3), \ldots$ je dáno vztahem $c(i) = E_k(m(i))$ pro každé $i \in \mathbb{N}$.
- dešifrování bloků šifrovaného textu $c(1), c(2), c(3), \ldots$ je dáno vztahem $m(i) = D_k(c(i))$ pro každé $i \in \mathbb{N}$.

Pro šifrování resp. dešifrování všech bloků otevřeného resp. šifrovaného textu je použita stejná transformace E_k resp. D_k.

5.3.1. Klasické blokové šifry

Definice (Substituční šifra). Nechť A je abeceda s t symboly, M a C jsou množiny konečných řetězců nad abecedou A a K je množina všech permutací na množině A. Substituční šifra je bloková šifra (M, C, K, E, D) s délkou bloku jednoho znaku s vlastnostmi:

- E a D jsou zobrazení definující pro každé $k \in K$ transformaci zašifrování $E_k = k$ a transformaci dešifrování $D_k = k^{-1}$.
  - šifrování znaku $m(i)$ otevřeného textu na šifrovaný text $c(i)$, pro každé $i \in \mathbb{N}$, je dáno vztahem $c(i) = E_k(m(i))$.
  - dešifrování znaku $m(i)$ otevřeného textu z šifrovaného textu $c(i)$, pro každé $i \in \mathbb{N}$, je dáno vztahem $m(i) = D_k(c(i))$.

Polygramové šifry.
Polygramové šifry jsou typickým příkladem substitučních šifer. Výchozí abeceda A je abecedou nějakého přirozeného jazyka s t symboly. Polygramová šifra přiřadí t-tici znaků nějakou jinou t-tici znaků. V případě, že je $t = 2$ jde o bigramovou šifru.

Playfair (1854).
Šifra Playfair je bigramovou šifrou, kde klíčem je matice 5x5 s 25 různými znaky. Pravidla pro šifrování bloku $M = m_1, m_2$:

- jestliže jsou m_1 a m_2 na stejném řádku, pak c_1 a c_2 jsou následovníci vzorů ve stejném řádku.
- jestliže jsou m_1 a m_2 ve stejném sloupci, pak c_1 a c_2 jsou následovníci vzorů ve stejném sloupci.
- jestliže jsou m_1 a m_2 na různých řádcích i sloupcích, pak jsou znaky c_1 a c_2 dalšími vrcholy obdélníku s hranami m_1c_1 a m_2c_2.
- jestliže je $m_1 = m_2$, pak se mezi znaky vloží nějaký nový znak.

Více informací k definici a kryptoanalýze šifry Playfair viz. [39].

Příklad (Playfair). Klíč šifry je dán $k = successfully$. Matice klíče, která je dána tabulkou viz. 2., je doplněna znaky bez výskytu duplicit. Otevřený text je dán bloky M = UB | LK | IJ | MO. Zašifrované bloky textu pak mají tvar C = YE | DQ | JD | NP.

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

Tabulka 2. Matice klíče šifry Playfair.

Definice (Transpoziční šifra). Nechť A je abeceda s t symboly, M a C jsou množiny řetězů délky t nad abecedou A a K je množina všech permutací na množině $\{1, 2, \ldots, t\}$. Transpoziční šifra je bloková šifra (M, C, K, E, D) s vlastnostmi:

- E a D jsou zobrazení definující pro každé $k \in K$ transformaci zašifrování E_k a transformaci dešifrování D_k.
  - $E_k = M \to C : m \to c = (c_1, c_2, \ldots, c_t) = (m_{k(1)}, m_{k(2)}, \ldots, m_{k(t)})$.

– $D_k = C \to M : c \to m = (m_1, m_2, \ldots, m_t) = (c_{l(1)}, c_{l(2)}, \ldots, c_{l(t)})$, kde $l = k^{-1}$.

- princip permutace: zápis $k(1)$ představuje původní pozici znaku před přesunem na první pozici.

Více informací k transpozičním šifrám [40] [5].

5.3.2. Statistické charakteristiky otevřeného textu

Problémem historických šifer byla skutečnost, že se dal otevřený text přímo odvodit z šifrovaného textu, protože šifrovaný text do sebe přímočaře přenášel statistické charakteristiky otevřeného textu. Shannon (viz. [6]) navrhl dvě metody řešení uvedeného problému: difúzi a konfúzi.

Příklad. Metody šifrování, které přenášejí statistickou charakteristiku otevřeného textu do šifrovaného textu:

- jednoduchá záměna - nemá vliv na četnost výskytu znaků (bigramů atd.), protože nemění jejich vazby.
- transpozice - ruší vazby mezi znaky, ale nedochází ke změně pravděpodobností jejich výskytů.
- Vigenerova šifra - částečně ruší rozdělení pravděpodobností jednotlivých znaků, ale v případě zjištění klíče může útočník odhadnout rozložení četnosti znaků na základě délky hesla.

1. **Difúze.**
   Difúze je metoda, která rozprostírá statistické vlastnosti otevřeného textu do delších úseků šifrovaného textu. Jedná se tedy obecně o vlastnost šifry zabývající se vlivem otevřeného textu a klíče na šifrovaný text.

   Příklad. Dobrou difúzi získáme zobrazením t znaků otevřeného textu do jednoho znaku šifrovaného textu. Uvažujeme abecedu o 26 znacích: $c(n) = (o(n-t+1) + o(n-t+2) + \cdots + o(n)) \mod 26$.

   Binární N-gram. Nechť máme dánu abecedu o k znacích. Podmnožina obsahující n znaků abecedy se nazývá uspořádaná n-tice. N-gram je zápis možného uspořádání znaků n-tice. Například pro n-tici 123456 jsou 1234, 1245, 1356 možné 4-gramy. Více viz. [41].

   Příklad (N-gramová substituce). Uvažujme N-gramovou substituci pro abecedu o 26 znacích, která má klíč v podobě invertibilní matice čísel z intervalu $< 0, 25 >$: $K = (k_{i,j})$, pro $i, j = 1, \ldots 10$. Budeme používat blokovou šifru s délkou bloku 10 znaků, šifrování probíhá dle zápisu:

$c(i) = (k_{i,1} * o(1) + k_{i,2} * o(2) + \cdots + k_{i,10} * o(10)) \mod 26$ pro $i = 1, \ldots, 10$. Uvedená šifra je dobře navržená z hlediska difúze, protože se podobné charakteristiky otevřeného textu promítají do všech znaků šifrovaného textu. Bloková šifra má hlavní nevýhodu, a to že může útočník při znalosti dostatečného množství bloků otevřeného a šifrovaného textu vypočítat použitou klíčovou matici.

2. **Konfúze.**
   Konfúze je metoda, která má učinit vztah (statistické vlastnosti) mezi klíčem a šifrovaným textem maximálně složitý. Cílem konfúze je znesnadnit útočníkovi možnou statistickou kryptoanalýzu. Obecně lze definovat konfúzi jako vlastnost šifry, která popisuje složitost kompozice mezi otevřeným textem, klíčem a šifrovaným textem.

   Příklad (N-gramová substituce). Popišme vlastnost konfúze N-gramové sustituce tvořené klíčovou maticí, viz. 1. Konfúze v tomto případě není zaručena. Samotná podmínka složitosti (obsáhlosti) vztahu klíče a šifrovaného textu je splněna, ale jednoduchým lineárním výpočtem lze klíč odvodit (výpočet soustavy lineárních rovnic).

Metody aplikace difúze a konfúze.
Existují dva základní způsoby k docílení kvalitní difúze a konfúze při tvorbě šifry: skládání šifer, iterace.

- **Skládání šifer.**
  Metoda skládání šifer je založena na principu opakované aplikace šifer na otevřený text. Při skládání šifer se využívá různých typů šifer (metody tvorby šifer), např. transpozice, substituce, lineární metody. Metoda opakovaného skládání šifer si klade za cíl efektivní promíchání klíče a otevřeného textu. Šifry, které využívají princip opakovaného skládání se nazývají *součinové šifry*. Součinové šifry používají jako klíč generátor posloupnosti, který se v každém kroku součinu aplikuje pro šifrování (dešifrování) aktuální verze textu. Více viz. [6].

  Příklad. Označme transpozici T (např. jednoduchá sloupcová transpozice), substituci S (např. polyalfabetická šifra) a lineární šifru L (např. aditivní šifra). Součinová šifra bude dána jako aplikace posloupnosti TLSLSTTL. Znamená to, že se nejprve na otevřený text aplikuje lineární šifra, následně se provede transpozice atd.

- **Iterované šifry.** Iterovaná šifra je speciálním případem součinové šifry, která má všechny použité šifry stejného druhu s jedinou výjímkou - první a poslední šifra může být jiná.

  Definice. Nechť máme dánu iterační šifru $I = I^n I^{n-1} \ldots I^1$. Iterace I^i, pro $i = 1, \ldots, n$, je nazývána rundou. Použité transformace šifrování E_i a

dešifrování D_i jsou tzv. *rundovní funkce* a jejich klíče k_i jsou tzv. *rundovní klíče.*

5.3.3. Vlastnosti a charakter moderních blokových šifer

Základní vlastností šifry by měla být dostatečná bezpečnost vůči možnému výpočtu klíče z množiny dvojic šifrovaný text, otevřený text. Informace, která se získá pomocí nějaké kryptoanalytické metody se nazývá užitečná informace např. blízké otevřené texty, možné šifrované texty atd. Kvalitní difúze a konfúze šifry přispívá ke splnění daného požadavku. Šifra by se neměla bez znalosti šifrovacího klíče lišit od náhodných permutací na množině {0,1}.

Většina moderních blokových šifer jsou iterační šifry ve tvaru $E_{k(n)}E_{k(n-1)}\dots E_{k(1)}$. Jednotlivé iterace jsou stejné transformace s různými rundovními klíči $k(i)$, pro $i = 1, \dots, n$. Klíče vznikají expanzí předchozího klíče. Expanze klíče je časově náročná operace a také proto existuje dvojí způsob tvorby klíčů:

1. Expanze probíhá před samotným šifrováním tzv. přípravná fáze šifrování. Rundovní klíče se uloží a jsou použity pro konkrétní iteraci. Tento způsob je nejčastější.

2. Expanze probíhá současně se šifrováním. Klíč se vytváří pro následující iteraci. Klíče nebývají ukládány (nedostatek paměti).

Je dobré dodržet podmínku rozdílnosti operací (poslední operace, první operace) u navazujících rundovních funkcí, protože by mohlo dojít při součinu ke zjednodušování.

Další metodou pro zkvalitnění konečné difúze a konfúze šifry je zašumění (DES, AES). Zašumění je aplikace rundovní funkce pomocí speciálních rundovních klíčů na otevřený text zcela nezávisle na samotné použité šifře. Rundovní klíče se použijí pouze jedenkrát. Zašumění je možno provést před i po samotném šifrování.

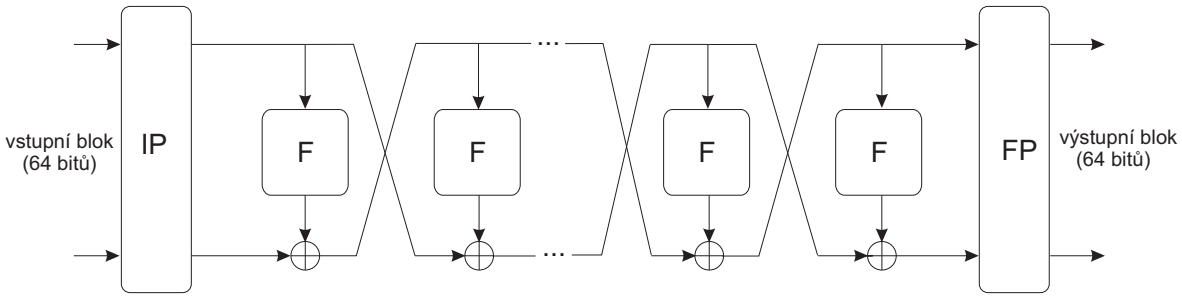
5.3.4. Významné blokové šifry

DES (The Data Encryption Standard) byla v roce 1976 přijata jako standard pro šifrování (Federal Information Processing Standard (FIPS)) v USA a později se prosadila v mezinárodním měřítku. Používá 56 bitový klíč. Od začátku se stala předmětem akademického zkoumání a to z důvodu své konstrukční povahy a relativně krátkého klíče. V současné době se nepoužívá, protože byla prokázána její nebezpečnost - krátký klíč, teoretické nedostatky v návrhu.

Obavy o bezpečnost a relativně časově náročný výpočet blokové šifry DES vedly ke snaze vytvořit vylepšenou alternativu, která by nedostatky odstraňovala.

V osmdesátých a devadesátých letech minulého století byly vytvořeny např. RC5, Blowfish, IDEA. Většina používala delší bloky textu a klíče (64, 128 bitový). Jako další řešení se jevilo opakované znovupoužití DES (Triple DES), což sice vede ke zvyšení bezpečnosti, ale celkově i ke zvýšení výpočetní náročnosti.

U mnohých nástupců DES se projevili nedostatky v návrhu. Dva typické příklady blokových šifer, na které byly provedeny úspěšné kryptoanalytické útoky jsou FEAL-4 a Mandryga. FEAL-4 je bloková šifra založená na F-funkci 15. a byla zveřejněna v roce 1987. Už od začátku se objevili důkazy nedostatků v jejím návrhu (například pomocí útoku volbou otevřených textů). Později byla šifra vylepšena zvětšením klíče i počtem rund, ale s příchodem diferenciální analýzy se i tato verze projevila jako nepoužitelná. Mandryga je bloková šifra, která byla vytvořena v roce 1984. Nebyla příliš rozšířená, i když byla jednoduše implementovatelná a efektivní. Byla jedna z prvních šifer využívající datově závislých rotací a její princip se stal součástí jiných šifer jako byly RC5 nebo RC6. V roce 1998 dokázal Biryukov a Kushilevitz pomocí diferenciální analýzy, že stačí pouze 16 párů při útoku volbou otevřených textů. Metodu útoku volby otevřených textů lze převést na ciphertext-only útok, což je jedna z kryptoanalytických metod, na kterou jsou náchylné moderní blokové šifry.

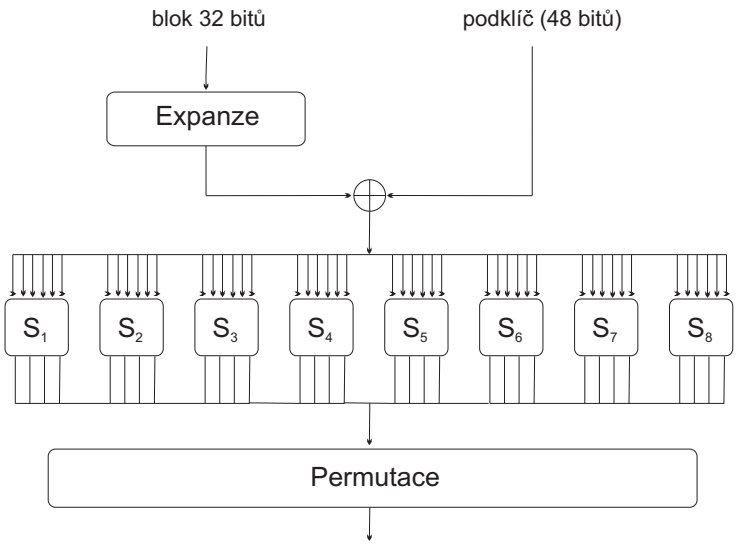


Obrázek 14. Formální popis blokové šifry DES.

.

Formální popis DES.
DES je archetypální blokovou šifrou, protože převádí otevřený text s pevnou délkou přes množinu komplikovaných operací na šifrovaný text stejné délky. Délka bloku je 64 bitů. Klíč je také 64 bitový, ale samotný algoritmus používá pouze 56 bitů. Zbývajících 8 bitů slouží k výpočtu parity a bity nejsou dále použity. Efektivní délka klíče je tedy pouze 56 bitů.

Algoritmus probíhá v 16 identických krocích (rundách). Na začátku a na konci se provedou dvě inverzní permutace (IP, FP), které mají spíše historický význam (rozložení bloků). Před každým krokem dojde k rozdělení bloku na dvě 32 bitové části, které jsou zpracovávány střídavě, jedná se o tzv. Feistelovo schéma. Na



Obrázek 15. Bloková šifra DES. Aplikace F-funkce na vybraný blok.

obrázku viz. 14. je znázorněn obecný princip DES, velkým F je označeno Feistelovo schéma. Fiestelovo schéma (F-funkce, rundovní funkce) viz. 15. zajišťuje, že šifrování a dešifrování jsou podobné procesy, jediný rozdíl je v opačném použití podklíčů. Schéma takto zjednodušuje hardwarovou implementaci. F-funkce se aplikuje na vybraný 32 bitový blok a její výstup je zkombinován (operace XOR) s druhým nepoužitým blokem. Před dalším krokem algoritmu dojde k prohození bloků. F-funkce zahrnuje čtyři operace:

1. expanze - použitím expanzí permutace na 32 bitový blok vznikne 48 bitový blok. Výstupem je osm 6 bitových bloků, které obsahují kopii 4 bitů ze vstupu a 2 bity jako kopii vybraných bitů sousedů (bloků).

2. smíchání s klíčem - výstup z procesu expanze je zkombinován (operace XOR) s 48 bitovým podklíčem. Existuje šestnáct 48 bitových podklíčů odvozených z původního klíče. Podklíč vzniká binárním posunem z předchozí verze původního klíče (key schedule).

3. substituce - výstup předchozí operace je rozdělen do osmi 6 bitových bloků, které tvoří S-boxy (základní struktura bezpečnosti DES). Následovně dojde k nelineární transformaci pomocí vyhledávací tabulky (Lookup table) - z 6 vstupních bitů se odvodí 4 výstupní bity.

4. permutace - aplikuje se pevná permutace na 32 vstupních bitů (S-boxy), vzniknou tzv. P-boxy. Permutace zajistí jiné rozdělení bitů v dalším kroku algoritmu.

S_3		Prostřední 4 bity (2^4 sloupců)					
		0000	0001	▪▪▪	0101	▪▪▪	1111
Krajní bity	00	1011	1101	▪▪▪	0100	▪▪▪	0110
	01	0011	0110	▪▪▪	0111	▪▪▪	0000
	10	0111	0010	▪▪▪	0001	▪▪▪	1100
	11	1001	1111	▪▪▪	1100	▪▪▪	0101

Obrázek 16. Bloková šifra DES. Princip substituce S-boxu F-funkce.

Použitím substituce S-boxy dochází k nelinearizaci výpočtu. V případě vynechání substituce a znalosti jednoho bloku otevřeného textu lze výpočet řešit jako pouhou soustavu lineárních rovnic s neznámou K. Na obrázku viz. 16. je ukázán princip substituce v S-boxu pomocí vyhledávací tabulky. Substituce nahrazuje vstupních 6 bitů výstupními 4 bity. Výběr použité substituce záleží na krajních 2 bitech vstupní šestice. Na obrázku je zvýrazněna substituce vstupních 6 bitů **0**01011**1** na 4 výstupní bity 0111.

Bezpečnost a kryptoanalýza DES.
Bezpečnost blokové šifry DES se stala hned od začátku své existence tématem diskuze a to hlavně v akademickém světě, což položilo základy výzkumu a vývoji moderních šifer. 56 bitový klíč byl stanoven z důvodu kapacity výpočetních prostředků a jevil se jako příliš krátký. Více informací k vývoji kryptoanalytických pokusů viz. [42].

Jako u každé šifry se nejprve zkoušela metoda útoku hrubou silou. Existovaly teoretické návrhy jak by se dala DES prolomit např. Diffie a Hellman (1977), Wiener (1993). Až v roce 1998 se objevila první reálná implementace (EFF - Electronic Frontier Foundation, Deep Crack), která potvrdila polemiky vedené kolem její bezpečnosti. Pozdější projekt COPACOBANA (2006) ukázal jednoduchost prolomení DES na komerční úrovni v řádu desítek hodin.

Existují rychlejší metody kryptoanalýzy než je útok hrubou silou viz. kapitola 4. Použití metod na DES je však víceméně teoretické a metody slouží spíše pro demostraci nedostatků v návrhu DES. Diferenciální kryptoalanýza (DC) potřebuje pro prolomení teoreticky 247 otevřených textů, ale při návrhu DES se s použitím DC počítalo. Lineární kryptoanalýza celkově snižuje potřebný počet textů pro odhalení použitého klíče až na 241 (Matsui (1994), Junod (2001)). Vedle uvedených metod se objevili výzkumy týkající se útoků na DES se sníženým počtem fází výpočtu např. Biham (2002) - DES s devíti fázemi (216 otevřených

textů).

DES má výpočetní vlastnost komplementárnosti:

$$E_k(P) = C \iff E_{k'}(P') = C'$$

, kde znak ' představuje bitové negace. Vlastnost udává, že lze v případě vhodně zvoleného otevřeného textu snížit složitost výpočtu útoku hrubou silou.

Posledním teoretickým nedostatkem blokové šifry DES je existence čtyř slabých (např. 0101 0101 0101 0101) a šesti poloslabých klíčů (např. (011F011F010E010, E1F011F010E010E01)).
V případě použití slabého klíče dostáváme:

$$E_k(E_k(P)) = P \text{ nebo ekvivalentně } E_k = D_k.$$

Poloslabé klíče vystupují ve dvojici (k_1, k_2) a šifrování jedním z dvojice odpovídá dešifrování s druhým klíčem dvojice:

$$E_{k_1}(Ek_2(P)) = P \text{ nebo ekvivalentně } E_{k_2} = D_{k_2}.$$

Slabé resp. poloslabé klíče jsou známy již od uvedení DES do praxe.

Triple DES (Triple Data Encryption Algorithm (TDEA)) je bloková šifra, která vzniká trojitou aplikací blokové šifry DES na stejný blok textu s použitím tří (různých) klíčů. Opakované použití DES zvyšuje délku klíče a tedy i odolnost vůči kryptoanalýze (útok hroubou silou), ale zároveň nemá TDEA složitější princip konstrukce než DES. TDEA vznikla jako alternativa DES, která měla odstraňovat její nedostatek v podobě délky klíče. V současnosti je stála používaná, ale díky existenci AES není v popředí významu. Více informací k použití TDEA viz. [43].

Formální popis Triple DES.
Triple DES používá sadu tří klíčů (K_1, K_2, K_3), které mají podobně jako u DES 56 bitů (výjma paritních bitů klíče). Šifrování lze obecně popsat zápisem:

$$C = E_{K_3}(D_{K_2}(E_{K_1}(P))).$$

Dešifrování je inverzní operace k šifrování:

$$P = D_{K_1}(E_{K_2}(D_{K_3}(C))).$$

Existují tři možnosti použití klíčů (K_1, K_2, K_3):

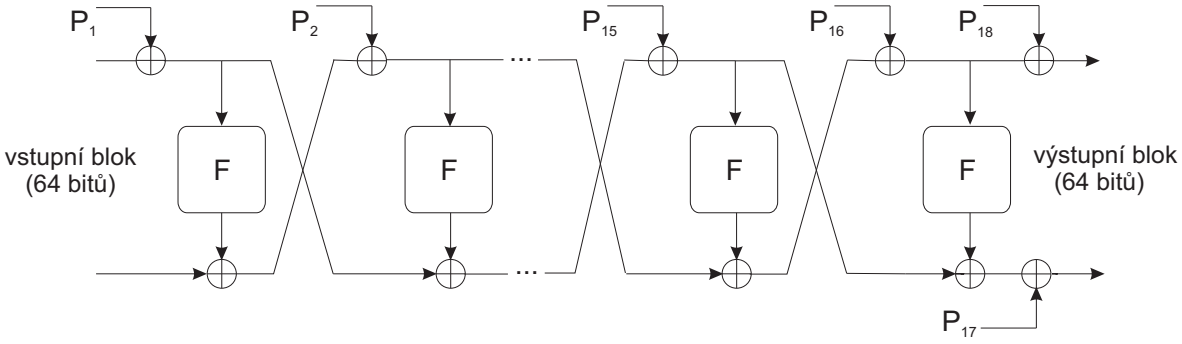
1. klíče jsou zcela nezávislé - nejlepší řešení z hlediska odolnosti vůči útoku, 168 bitový klíč.

2. klíče K_1 a K_3 jsou identické, klíč K_2 je na nich nezávislý - klíč je 112 bitový.

3. klíče jsou identické - výsledný klíč je pouze 56 bitový, odpovídá obyčejnému DES, není doporučováno používat.

Bezpečnost a kryptoanalýza Triple DES.
V případě použití klíčů v 1) a 2) možnosti je samotná Triple DES mnohem bezpečnější vůči útoku hrubou silou než tomu bylo u DES. Efektivní bezpečnost v případě 1) je z důvodu možného použití metody meet-in-midle viz. snížena na 112 bitů. V případě použití klíčů z bodu 2) je TDEA citlivější vůči možnému použití metody volby otevřených textů a metody známých otevřených textů. V současné době se podle organizace NIST (National Institute of Standards and Technology - USA) považuje TDEA při použití klíčů z 1) za výpočetně (pamětově, časově) odolnou.

Blowfish je symetrická bloková šifra, která byla navržena v roce 1993, autorem je Bruce Schneier. Je založena na principu klíčově závislých S-boxů (podobně jako DES) a disponuje propracovanou metodou odvozování klíčů (key schedule). Stala se součástí mnoha aplikací z důvodu své výpočetní rychlosti a dostatečné kryptoanalytické odolnosti. Blowfish není patentovaná a je veřejně přístupná. Více viz. [44].



Obrázek 17. Princip blokové šifry Blowfish.

Formální popis Blowfish.
Blowfish pracuje s 64 bitovými bloky textu a s klíči v rozmezí 32-448 bitů. Podobně jako DES používá F-funkce (Feistelovo schéma). Algoritmus šifry Blowfish je složen z šestnácti rund a využívá třech operací: klíčově závislé permutace, klíčově a datově závislé substituce, operace XOR. Algoritmus pracuje s jedním P-polem (pole permutací) a čtyřmi S-boxy. Každý S-box bere jako vstup 8 bitů a výsledkem subtituce je výstup 32 bitů. S-box má 256 položek. P-pole je složeno z osmnácti 32 bitových podklíčů $(P_1, P_2, \ldots, P_{18})$. Princip Blowfish je znázorněn na obrázku viz. 17.
Na začátku je 64 bitový blok rozdělen na dva 32 bitové bloky. V každém kroku

algoritmu (rundě) se střídavě aplikuje na jeden vybraný 32 bitový blok permutace (P-pole) a rundovní funkce (F-funkce). Na závěr, po proběhnutí šestnácté rundy, se použije permutace dle položek P-pole (P_{17}, P_{18}) na oba 32 bitové bloky. Samotný algoritmus šifrování lze popsat pseudokódem:

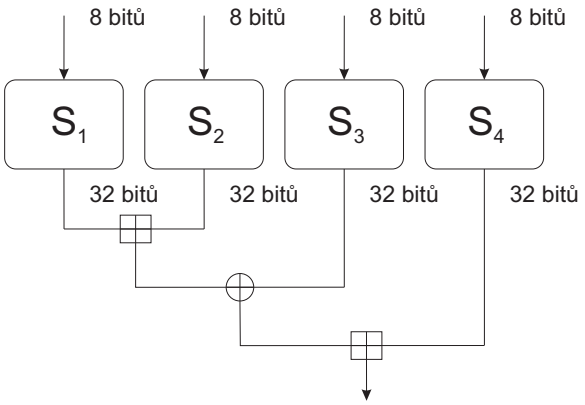
```
Rozdel x na dva 32 bitove bloky: xL, xR.
  for(i = 0; i < 16; i++)
  {
    xL = xL XOR Pi;
    xP = F(xL) XOR xP;
    Prohod(xL,xP);
  }
  Prohod(xL,xP);
  xR = xR XOR P17;
  xL = xL XOR P18;
Sluc xL a xP.
```

Princip F-funkce, viz obrázek 18.:

1. vstupní 32 bitový blok je rozdělen na čtyři 8 bitové bloky (a, b, c, d), které budou vstupem subtituce.

2. výpočet F-funkce probíhá dle zápisu:

$$F(xL) = ((((S_1(a) + S_2(b)) \mod 2^{32}) \oplus S_3(c)) + S_4(d)) \mod 2^{32}.$$



Obrázek 18. Bloková šifra Blowfish. Princip F-funkce.

Blowfish je specifický svou tvorbou podklíčů. Podklíče jsou vytvářeny před každým (de)šifrovaním. Tvorba podklíčů (nastavení P-pole a S-boxů) probíhá v následujících krocích:

1. na začátku proběhne inicializace P-pole a posléze i čtyř S-boxů pomocí pevného řetezu, který je odvozený od hexadecimálního tvaru konstanty π. Řetězec je v náhodném tvaru bez statistických vlastností (Nothing up my sleeve number).

2. pomocí operace XOR je tajný klíč cyklicky aplikován na položky P-pole: XOR P_1 s prvním bitem 32 bitového klíče, XOR P_2 s druhým bitem 32 bitového klíče atd.

3. zašifruje se nulový 64 bitový blok pomocí klíčů vytvořených v 1) a 2).

4. nahradí se (32 bitové) P_1 a P_2 šifrovaným výstupem z 3).

5. opakovaně se zašifruje výstup z 3) pomocí nových klíčů a následuje nahrazení P_3 i P_4 tímto šifrovaným výstupem.

6. šifrování pokračuje dále, dokud nejsou všechny položky P-pole a S-boxů nahrazeny změněnými výstupy. Algoritmus generující podklíče potřebuje ke změně celkově 512 iterací.

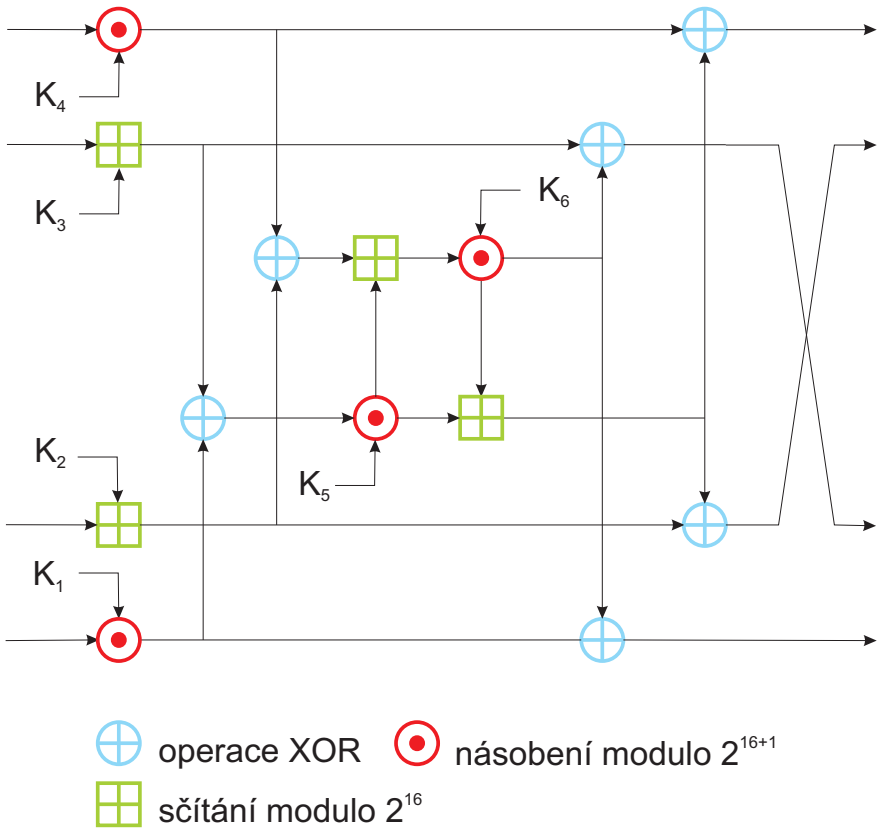
Šifrování je identické s dešifrováním, pouze u dešifrování dochází k opačnému použití prvků P-pole, tedy $(P_{18}, P_{17}, \ldots, P_2, P_1)$. Podobně jako u šifrování se musí nejprve vytvořit podklíče před zahájením dešifrování.

Bezpečnost a kryptoanalýza Blowfish.
V současnosti není známa efektivní metoda k prolomení blokové šifry Blowfish. Autor šifry navrhl zjednodušenou verzi Blowfish-Mini (Blowfish-32 (32 bitové bloky, 16 bitové podklíče)), na které demonstroval možnosti kryptoanalýzy. Doporučovaná verze Blowfish s šestnácti rundami je odolná vůči útoku hrubou silou. V roce 1996 bylo dokázáno, že by bylo potřeba k prolomení Blowfish pomocí metody známých textů celkem 2^{8r+1} textů, kde r značí počet rund. Při teoretickém použití slabých klíčů se celkový počet potřebných známých textů snižuje, ale u doporučené verze Blowfish neexistuje možnost odvození klíčově závislých S-boxů. Více k kryptoanalýze viz. [45].

Nástupci Blowfish.
Nástupci Blowfish jsou *Twofish* a pozdější *Threefish*. Twofish i Threefish, jejichž autor je Bruce Schneier, jsou opět otevřené nelicencované standarty. Twofish (1998) je jedním z finálních kandidátů, ze kterých byl vybrán standard AES. Twofish pracuje podobně jako Blowfish s klíčově závislými S-boxy, navíc používá principy jiných návrhů a má propracovanou správu klíčů - polovina klíče je použita jako skutečný klíč a druhá polovina upravuje šifrovací algoritmus. Twofish měla být nástupcem Blowfish, která by Blowfish zcela nahradila, ale vzhledem k odolnosti a době používání Blowfish, ale i díky existenci standardu AES, je Twofish v ústraní zájmu. Threefish (2008) již nepoužívá S-boxy. Šifra je tedy odolná vůči časovým útokům a útokům na základě vyhledávací tabulky. Její nelineárnost

je založena na třech operacích - střídavém sčítání, XOR, rotaci s pevnou délkou. Je součástí Skeinovi hašovací funkce (Skein hash function).

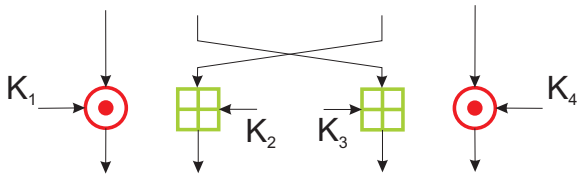
International Data Encryption Algorithm (IDEA) je bloková šifra, která vznikla v roce 1991 jako další z návrhů pro náhrazení standartu DES. Její autoři jsou Xuejia Lai a James Massey. IDEA je patentovaná, ale je volně přístupná pro nekomerční použití. Je založena na principu aplikace substitucí a permutací.



Obrázek 19. Bloková šifra IDEA. Princip rundy.

Formální popis IDEA.
IDEA pracuje s 64 bitovými bloky a s 128 bitovým klíčem. Její algoritmus je založen na sérii osmi identických rund a závěrečné poloviční rundě. IDEA je složena z následujích tří operací:

1. sčítání modulo 2^{16}.
2. násobení modulo 2^{16+1}, nulové slovo (0x0000) je interpretováno jako 2^{16}.
3. bitové operace XOR.



Obrázek 20. Bloková šifra IDEA. Princip poloviční rundy.

Výchozí 64 bitový blok je v rámci jedné rundy rozdělen na čtyři části, na které se použijí výše uvedené operace. Na obrázku viz. 19. je ukázán princip aplikace operací a vzájemné prolínání jednotlivých částí. Po proběhnutí osmi výchozích rund následuje poloviční runda, která je znázorněna na obrázku viz. 20.
Každá runda využívá šesti 16 bitových podklíčů, poslední poloviční runda již potřebuje pouze čtyři 16 bitové podklíče. Na obrázku, viz. 19., jsou klíče označeny symboly K_1 až K_6. Celkem je tedy potřeba padesátdva 16 bitových podklíčů. Prvních osm podklíčů je přímo derivováno z původního klíče a další klíče vznikají binárním posunem vlevo. K_1 je tvořen nejnižšími 16 bity výchozího 128 bitového klíče, K_2 je tvořeno následujícími 16 bity atd. Další skupina osmi klíčů vzniká rotací hlavního klíče o 25 bitů vlevo pro příslušnou skupinu osmi klíčů.

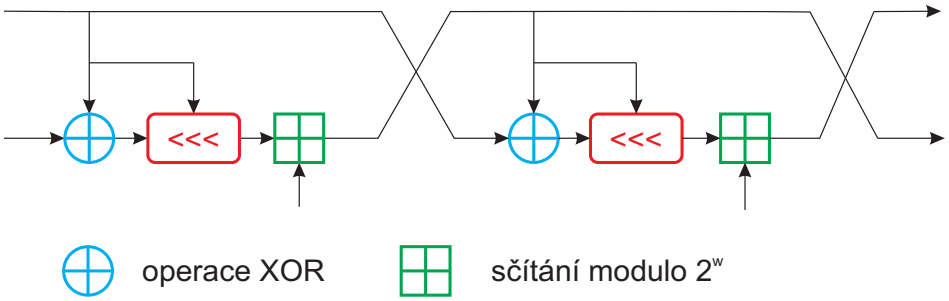
Bezpečnost a kryptoanalýza IDEA.
IDEA je v současné době považována za odolnou vůči diferenciální kryptoanalýze při dodržení předepsaných podmínek. Nejsou známy žádné lineární nebo algebraické nedostatky návrhu. V roce 2007 byl proveden útok s použitím všech klíčů na blokovou šifru IDEA, která byla zredukována na šest rund, viz. [46]. Z těchto závěrů bylo dokázáno, že by bylo potřeba nejméně 2^{128} operací k prolomení šifry. Existují i slabé klíče, které činí IDEA náchylnější pro případný útok. Jedná se o klíče s velkým počtem nul, ale problém lze vyřešit použitím operace XOR na klíč a konstantu. Samotná šifra není již doporučována k použití a to z důvodu pokroku v dešifrování, samotné existence mnohem rychlejších algoritmů i patentu IDEA.

RC5 je bloková šifra, která je charakteristická poměrně jednoduchým designem. Byla navržena Ronaldem Rivestem v roce 1994 a je definována v několika schématech, které se liší velikostí bloku, klíče a počtem rund. Její nástupce RC6 [28] byla jedním z kandidátů na AES. Podrobnější informace k RC5 [27].

RC5 se vyskytuje v různých velikostech bloků: 32, 64 a 128-bitů, velikost klíče nabývá hodnoty od 0 do 2040-bitů. RC5 lze definovat v rozdílném počtu rund a to od 0 do 255. Původní návrh doporučoval použití 32-bitových bloků, 128-bitový klíč a 12 rund. RC5 tedy vystupuje ve formátu RC5-w/r/b, kde w je velikost bloku v bitech, r je počet rund a b násobek 8-bitových hodnot klíče.

Rivest se při návrhu RC5 zaměřil na použití jednoduchých kryptografických

operací jako jsou OR, XOR, modulo a posunutí. Design šifrovacího resp. dešifrovacího algoritmu je poměrně jednoduchý, využívá Feistelových funkcí. Základem je specifické generování klíče, které je založeno na jednosměrné funkci, která kombinuje původní hodnoty klíče a dvě tzv. magické konstanty: eulerovo číslo a hodnotu zlatého řezu.



Obrázek 21. Princip rundovní funkce RC5. Na obrázku je znázorněna poloviční runda pro jeden ze dvou bloků vstupního textu.

Formální popis RC5.
Uvažujme obecnou RC5, která je určena blokem w, počtem rund r a hodnotou klíče b.

1. **Generování klíče:**

   - Definice magických konstant: $P_w = Odd((e-2)2^w)$ a $Q_w = Odd((\phi - 1)2^w)$, kde $e = (2,718281828459\ldots)$ a $\phi = (1,618033988749\ldots)$. Odd je celé liché číslo k výsledku x. Například pro $w = 32$ dostáváme $P_{32} = b7e15163$ a $Q_{32} = 9e3779b9$ v hexadecimální podobě.
   - Převedení tajného klíče: tajný klíč v bitové podobě $K[0,\ldots,b-1]$ převedeme do pole $L[0,\ldots,c-1]$, kde $c = [b/u]$ pro $u = w/8$ (bajt/slovo). V little-endian konvenci proces definuje pseudokód (pole L je na začátku vynulováno):

```
for(int i = b - 1; i >= 0; i--)
  {
    L[i/u] = (L[i/u] <<< 8) + K[i];
  }
```

   - Inicializace pole S: pole S se inicializuje pomocí magických konstant s použitím operace modulo 2^w:

```
S[0] = Pw;
for(int i = 0; i <= t - 1; i++)
  {
    S[i] = S[i - 1] + Qw;
  }
```

- Promíchání tajného klíče: v posledním kroku generování klíče dojde k promíchání tajného klíče ve třech průchodech s poli S a L. Vzhledem k možným rozdílům ve velikosti polí S a L se menší pole promíchá dle své velikosti:

```
int i = j = 0;
int a = b = 0;
int iteration = max(t, c);

for(int k = 0; k < iteration; k++)
 {
   a = S[j] = (S[i] + a + b) <<< 3;
   b = L[j] = (L[i] + a + b) <<< (a + b);
   i = (i + 1) mod(t);
   j = (k + 1) mod(c);
 }
```

2. **Šifrování bloku w:** blok w si rozdělíme na dvě části. Šifrovací algoritmus lze formálně popsat:

```
A = A + S[0];
B = B + S[1];
for(int i = 1; i <= r; i++)
 {
   A = ((A xor B) <<< B) + S[2*i];
   B = ((B xor A) <<< A) + S[2*i + 1];
 }
```

3. **Dešifrování:** Dešifrování je inverzní operace k šifrování:

```
for(int i = r; i > 1; i--)
 {
   B = ((B - S[2*i + 1]) >>> A) xor A;
   A = ((A - S[2*i]) >>> B) xor B;
 }
```

Bezpečnost a kryptoanalýza RC5
Vzhledem k poměrně jednoduchému designu byla RC5 hojně kryptoanalyticky zkoumána. RC5 je patentována společností RSA security. Byla vypsána odměna za úplné prolomení RC5, ale termín vypršel v roce 2007. Při použití doporučené velikosti klíče a počtu rund, je RC5 (RC5-64/18/16) zcela bezpečná.

U RC5-64/12/b byl uveřejněn lineární diferenciální útok, který používal 2^{44} volených šifrovaných textů [29]. Jako řešení se doporučuje použít 18 až 20 rund, které zarušují dostatečnou bezpečnost. Minimální velikost klíče by měla být 128-bitů. Doporučovaná velikost bloku je 64-bitů.

K prolomení RC5 byla využita distribuovaná síť Distributed.net [30], pomocí níž se podařilo hrubou silou prolomit šifrované texty zašifrované oslabenou verzi RC5 s 56 resp. 64-bitovým klíčem. Metoda hledá všechny možné klíče a testuje je na šifrovaném textu. V současnosti se hledají klíče pro RC5 s 72-bitovým klíčem.

Advanced Encryption Standard (AES) byla vybrána v roce 2001 organizací NIST ze skupiny uchazečů jako standard pro blokové šifrování. AES se stala mezinárodně uznávanou a nahradila problematické DES. Norma obsahuje tři blokové šifry s velikostí bloku 128 bitů: AES-128 (128 bitový klíč), AES-192 (192 bitový klíč), AES-256 (256 bitový klíč). Norma pochází z původní sbírky návrhů označovaných souhrnně jako Rijndael (autoři - Vincent Rijmen a Joan Daemen). AES využívá sérii substitucí a permutací (Substitution permutation network), ale již nepoužívá Feistelovo schéma jako tomu bylo například u DES. Skupina šifer označovaná Rijndael je specifická svými vlastnostmi jako je možná délka bloku (až 256 bitů), délka klíče (teoreticky neomezená), vlastnosti základní struktury (stavu) a také variacemi ve fázích výpočtu. AES je volně použitelná a to i pro komerční účely. Více informací k AES viz. [47] [48].

Formální popis AES.
Základní struktura používaná u AES je dvojrozměrné pole bajtů velikosti 4x4. Pole je nazýváno stavem a většina výpočtů je prováděna na konečné množině. Samotný algoritmus šifrování se skládá z posloupnosti transformačních rund. Dešifrování probíhá dle revezního pořadí rund při použití stejného klíče. AES používá deset rund pro 128 bitové klíče, dvanáct rund pro 192 bitové klíče a čtrnáct rund pro 254 bitové klíče. Algoritmus lze popsat pomocí následujících fází:

1. Expanze klíče: pomocí Rijndaelova procesu tvorby klíčů (Rijndael's key schedule) jsou z původního klíče vytvořeny rundovní podklíče.

2. Inicializační fáze:

   - AddRoundKey - pomocí operace XOR je každý bajt stavu zkombinován s rundovním podklíčem.

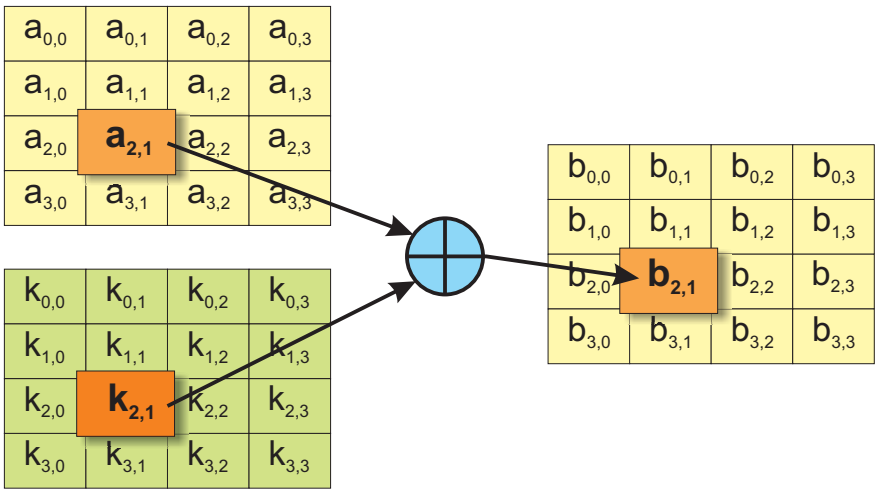
3. Hlavní rundovní fáze:

- SubBytes - nelineární substituční proces, ve kterém dochází k substituci všech bajtů stavu pomocí vyhledávací tabulky.
- ShiftRows - transpoziční proces, ve kterém jsou posunuty prvky řádků.
- MixColumns - proces směšování sloupců stavu, každé čtyři bajty sloupce jsou vynásobeny polynomem.
- AddRoundKey.

4. Závěrečná runda:

- SubBytes.
- ShiftRows.
- AddRoundKey.

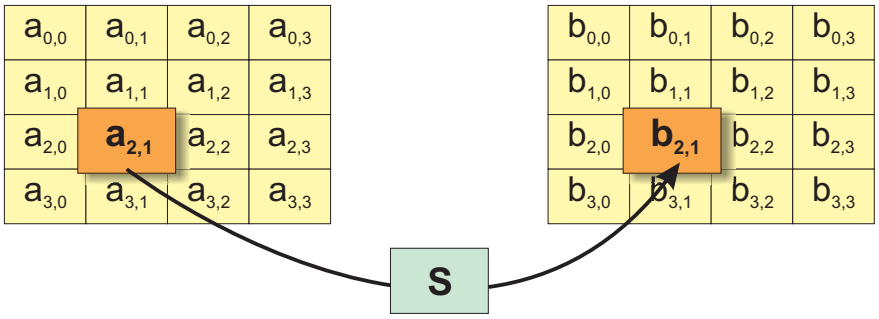
Inicializační fáze a závěrečná runda proběhnou pouze jedenkrát. Počet opakování hlavní rundovní fáze závisí na celkovém počtu rund, což se liší podle typu klíče u AES.



Obrázek 22. Bloková šifra AES. AddRoundKey - pomocí operace XOR je každý bajt stavu zkombinován s každým bajtem rundovního podklíče.

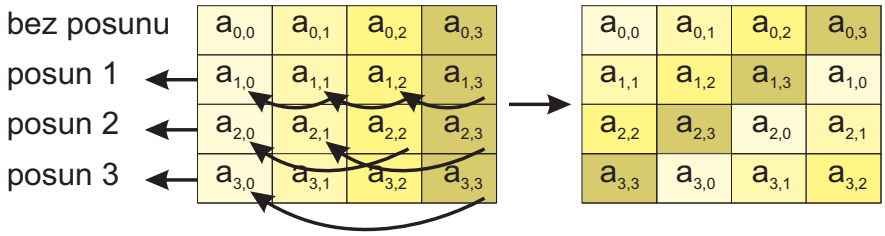
AddRoundKey viz. obrázek 22. je proces kombinace stavu a rundovního podklíče. Každý bajt stavu je pomocí operace XOR zkombinován s odpovídajícím bajtem aktuálního rundovního podklíče. Podklíč je výsledkem procesu expanze hlavního klíče. Délka rundovního podklíče a stavu je stejná.

SubBytes viz. obrázek 23. je proces, ve kterém dochází k substituci všech bajtů stavu dle 8 bitového S-boxu (Rijndael S-box). Samotný S-box je výsledkem



Obrázek 23. Bloková šifra AES. SubBytes - každý bajt stavu je substituován podle příslušné hodnoty v Rindael S-boxu.

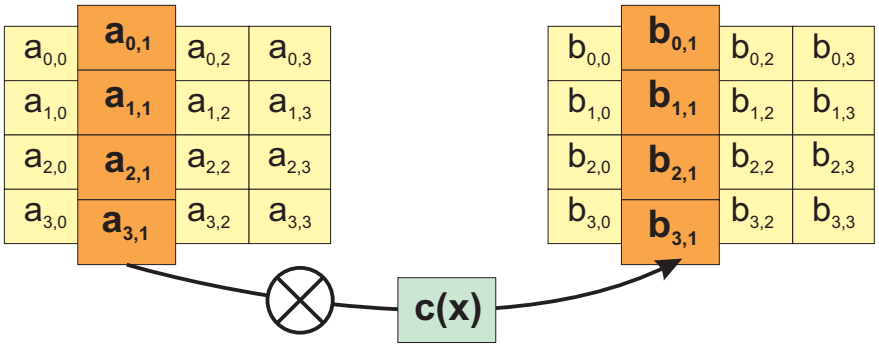
multiplikativní inverze na konečné množině následovné afinní transformací. S-box zaručuje dobré nelineární vlastnosti a je odolný vůči lineární a diferenciální kryptoanalýze. Rijndael S-box minimalizuje korelaci mezi lineární transformací vstupních a výstupních bitů a také minimalizuje rozdíl šíření pravděpodobnosti.



Obrázek 24. Bloková šifra AES. ShiftRows - prvky řádků jsou posunuty směrem vlevo. Hodnota posunutí je pro jednotlivé řádky rozdílná.

ShiftRows viz. obrázek 24. je proces, který přesouvá prvky řádků stavu směrem vlevo podle specifického ofsetu. První řádek zůstává nezměněn (ofset nula). Druhý řádek má ofset jedna, třetí má ofset dva a čtvrtý má ofset tři. Znamená to, že se změněné výstupní sloupce stavu skládají ze všech prvků původních sloupců. U variant Rijndael s délkou bloku 128 (AES) a 192 jsou ofsety nastavené podle výše uvedeného popisu. U verze s délkou bloku o 256 bitech dochází ke změně v použití ofsetu $(0, 1, 3, 4)$.

MixColumns viz. obrázek 25. aplikuje lineární transformaci na vstupní čtyři bajty. Hodnota každého vstupního bajtu ovlivní hodnoty všech výstupních čtyř bajtů. MixColumns a ShiftRows zajišťují difúzi u AES. MixColumns vlastně násobí každý sloupec stavu známou maticí, např. pro 128 bitový klíč je matice dána



Obrázek 25. Bloková šifra AES. MixColumns - každý sloupec je vynásoben s pevně daným polynomem $c(x)$.

zápisem:

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}$$

Hodnoty v matici udávávají specifickou operaci: násobení 1 hodnotu nemění, násobení 2 odpovídá posunu bitu vlevo a násobení 3 znamená posun vlevo a aplikaci XOR s původní hodnotou před posunen. Každý sloupec lze reprezentovat jako polynom v Galoisově konečném poli, které je reprezentováno dvěma konečnými poli s osmi členy GF(2^8). Galoisovo pole redukuje polynomy pro násobení. Takto reprezentovaný sloupec pak násobí pevný polynom $c(x) = 3x^3 + x^2 + x + 2$, dostáváme tzv. MDS matici (Maximum Distance Separable).

Nyní si popíšeme podmínky určující počet S-boxů u AES. Aktivní S-box je S-box s nenulovou vstupní diferencí. Dle teorie širokých cest (čím jsou cesty širší, tím se budou více užívat) existuje $\beta^2 = 25$ aktivních S-boxů ve 4 rundách. Spodní hranici získáme pomocí modifikované techniky pro hledání (Matsui) cest pro analýzu $AES[2, 8, 9, 12]$. Nechť $a(r)$ je minimální počet aktivních S-boxů pro r rund, pak hledané $a(r)$ z odhadnutých dostaneme $a(r) = a(r-4) + 25$ pro $(r \geq 4)$.

Bezpečnost a kryptoanalýza AES.
Doposud nebyl publikován úspěšný útok vůči plnohodnotné verzi AES. Design a odolnost všech klíčových verzí AES jsou dostatečné pro ochranu utajovaných informací. Pro zajištění vysoké ochrany je doporučeno používat verze AES-192 a AES-256. Pří implementaci AES je nutno také dbát na operační módy blokových šifer, což je ale při dodržení předepsaných podmínek u standardů vyřešeno.

Existuje možnost útoku vedlejším kanálem (side-channel attacks) na speciální implementaci AES, ale tato možnost přepokládá systém z něhož unikají

informace, což nesouvisí s bezpečností šifry. Jedna z prokázaných metod je cache-timing útok na AES z OpenSSL. V roce 2005 dokázal D.J. Bernstein získat klíč se serveru, který byl použit pouze pro šifrování. Server byl navržen tak, aby hlásil co nejvíce informací o časování a to počet strojových instrukcí použitých při šifrování. Princip útoku spočívá ve faktu, že čas potřebný na vytvoření modifikovaného vyhledávacího pole (stavu) je závislý na indexu pole a tedy i čas potřebný k výpočtu AES je na poli závislý. Tímto způsobem může útočník zjistit použité šifrovací podklíče. Útočník například sleduje čas, který je potřeba pro zpracování velkého množství vstupů, sčítá časy pro nějaký předpokládaný index vstupu a zjistí kdy je čas pro tento vstup maximální. Zároveň musí provádět experimenty se známými klíči a disponovat stejnými výpočetními prostředky (CPU, implementace AES) jako má server. Posléze může provést výpočty, které mu na základě hodnoty časů odhalí použitý podklíč. Předcházení uvedenému útoku je problematické, protože by bylo potřeba zaručit, aby operace výpočtu u AES trvaly konstatní čas, který je nezávislý na použitém klíči a vstupu. Zároveň autor předpokládal, že se jedná speciální server a bude možno získávat informace o přesném času výpočtu a o druhu výpočtu, což nezávisí na samotné bezpečnosti AES. Další z publikovaných útoků vedlejším kanálem je diferenční analýza chyb (Differential fault analysis), která zjišťuje stav systému navozením poruch (neočekávaných podmínek) do výpočetního prostředí, což může být například vystavení procesoru vysokým teplotám, magnetickému poli atd.

Byly popsány úspěšné útoky na AES s menším počtem rund a to sice pro AES-128 se sedmi rundami, AES-196 s osmi rundami a AES-256 s devíti rundami. AES má narozdíl od používaných blokových šifer poměrně jednoduchý algebraický popis, což vedlo ke snaze o nalezení jiné kryptoanalytické metody než je útok hrubou silou. Pro prolomení blokových šifer jako je AES byla setrojena metoda XSL útoku (XSL attack), ale bylo dokázáno, že je původní myšlenka XSL útoku na blokové šifry nepoužitelná. Dalším teoretickým útokem je použití metody souvisejících klíčů (related-key attack). Metoda předpokládá matematický vztah mezi použitými klíči, což se sice také jeví nereálně, ale z hlediska nevhodné implementace je myšlenka kryptoanalytické metody správná. Samotná metoda snižuje složitost útoku, ale opět se předpokládá snížený počet rund. Kryptoanalytická metoda souvisejících klíčů je poměrně nová a je předmětem zkoumání. Jedna z nejnovějších kryptoanalytických metod vyvíjených pro AES je metoda zjišťování známých klíčů (known-key distinguishing attack). Metoda je sice lepší než výše uvedené kryptoanalytiské návrhy a to sice z pohledu složitosti útoku, ale opět předpokládá snížený počet rund (osm rund pro AES-128).

Popíšeme si možnosti diferenciální kryptoanalýzy viz. 4.3.1. pro AES. Nelineární funkce u AES má maximální diferenciální pravděpodobnost 4/256 (nejvíce záznamů jsou však buď 0 nebo 2). Teoreticky je tedy možné určit klíč s polovičním množství práce jako u útoku hrubou silou. Design AES disponuje vysokými větvemi, které zabraňují použití tras s vysokou pravděpodobností přes více rund.

Ve skutečnosti by měla být AES odolná vůči diferenciální i lineární kryptoanalýze, i když by měla slabší nelineární funkci. Neuvěřitelně vysoké větve (aktivní počet S-boxů viz. 5.3.4.) od 25 přes 4R způsobují, že více než 8 rund zahrnují méně než 50 nelineárních transformací. Pravděpodobnost úspěchu není vyšší než $\mathbb{P}[\text{útok}] \leq \mathbb{P}[\text{nejlepší útok na S-box}]^5 0$. Například pro aktuální S-boxy AES vyzařuje bez pevného diferenciálu s pravděpodobností větší než $(4/256)^{50}$ nebo 2^{-300}, což je nižší než požadové hodnoty 2^{-128} pro 128-bitové blokové šifry. To umožnilo použít efektivnější S-boxy, které i přes diferenciální uniformitu o hodnotě 16, zaručují pravděpodobnost útoku pod 2^{-200}.

Přehled bezpečnosti blokových šifer

Tabulka 3. nabízí přehled nejznámějších blokových šifer včetně velikosti klíčů, počtu rund, velikosti bloků a struktury sítě. Struktura sítě je dána jako substitučně-permutační (SP), alternativy Feistelovy sítě (F), modulární sčítání + rotace + XOR (ARX). Doporučené velikosti jsou vyznačeny tučně. Tabulka 4. uvádí v současnosti nejúčinnější výsledky kryptoanalýzy.

Šifra	Velikost bloku	Délka klíče	Počet rund	Struktura
AES (Rijndael)	128	128, 192, 256	10, 12, 14	SP
BlowFish	64	32-448	16	F
CAST-128	64	40-128	12 nebo 16	F
DES	64	56	16	F
3DES	64	**168**, 128, 56	48	F
IDEA	64	128	8,5	ARX
RC2	64	8 - 128 (**64**)	18	F
RC5	32, **64**, 128	8-2048 (**128**)	12-255 (**20**)	F
RC6	128	128, 192, 266	20	F
SEED	128	128	16	F
SERPENT	128	128, 192, 256	32	SP
Skipjack	64	80	32	F
TEA	64	128	64	F
Twofish	128	128, 192, 256	16	F

Tabulka 3. Přehled nejznámějších blokových šifer. Velikosti klíčů a bloků jsou uvedeny v bitech.

5.3.5. Operační módy blokových šifer

Blokové šifry jsou definovány s požadavkem na pevně danou délku bloku, nejčastěji to bývá 64 nebo 128 bitů. Ve skutečnosti je většinou potřeba šifrovat nejen celé N bitové bloky, ale obecně libovolnou posloupnost bitů. Vzniká zde potřeba přizpůsobit blokovou šifru pro šifrování posloupnosti bitů jakékoliv délky.

Šifra	Zabezpečení	Nejlepší útok	Doporučení	Typ útoku
AES128	2^{128}	$2^{126.1}$ čas, 2^{88} text, 2^{8} pamět	ano	
AES192	2^{192}	$2^{189.7}$ čas, 2^{80} text, 2^{8} pamět	ano	
AES256	2^{256}	$2^{254.4}$ čas, 2^{40} text, 2^{8} pamět	ano	
BlowFish	2^{448}	4 z 16 rund	ano	diferenciální
CAST-128	2^{128}	2^{48} čas, 2^{17} chosen text	spíše ne	related-key
DES	2^{56}	plná verze	ne	brutal force
3DES	2^{168}	2^{113} čas, 2^{32} text, 2^{88} pamět	spíše ne	
IDEA	2^{128}	6 z 8,5 rund	ano	diferenciální
RC2	2^{64} až 2^{128}	2^{32} chosen plaitext	ne	related-key
RC5	2^{128}	12 rund	ano	diferenciální
RC6	2^{128} až 2^{256}	není znám	ano	
SEED	2^{128}	není znám	ano	
SERPENT128	2^{128}	10 z 32 rund	ano	lineární
SERPENT	2^{192} a 2^{256}	11 z 32 rund	ano	lineární
Skipjack	2^{80}	31 z 32 rund	ne	dif. nespl.
TEA	2^{128}	2^{32} čas, 2^{23} chosen plaintext	ne	related-key
Twofish	2^{128} až 2^{256}	6 z 16	ano	dif. nespl.

Tabulka 4. Přehled nejúspěšnějších kryptoanalytických útoků vybraných blokových šifer.

Vedle uvedené skutečnosti zde vzniká problém, že šifrování stejného otevřeného textu stejným klíčem dává stejný výstup. Z těchto důvodů vznikly operační módy blokových šifer. Operační módy blokových šifer určují způsob použití blokových šifer v kryptosystému. Více informací viz. [18].

Blokové šifry založené na substituci využívají pro zaručení bezpečnosti faktu, že samotná substituční tabulka je příliš veliká a proto ji nelze jednoduše vypočítat, např. u 128 bitového bloku obsahuje tabulka 2^{128} prvků. Neukládá se tedy celá tabulka, ale pouze algoritmus transformace. I když je obtížné samotnou tabulku vypočítat, stále zde platí nevýhody substituce jako je podobnost substituovaných bloků, či "vyzařování" šifrované informace.

Většina operačních módů, výjimku tvoří například ECB, využívají inicializační vektor IV viz. 5.2.6. k nastavení počátečního bloku, viz. [1]. IV slouží

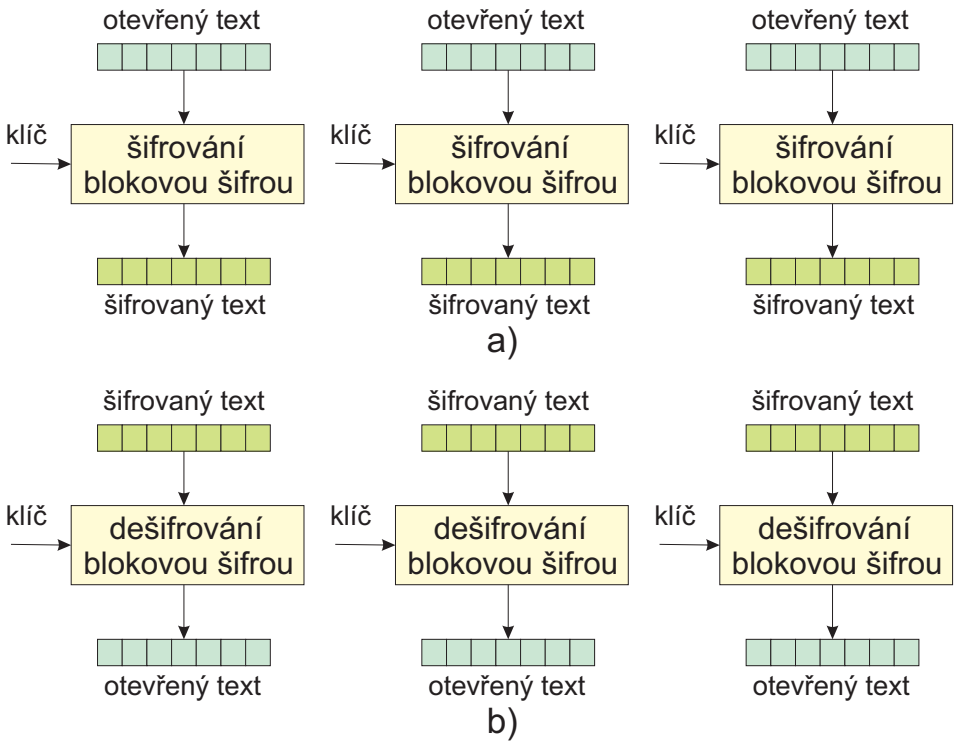
jako prostředek randomizace algoritmu šifrování. IV je inicializován různým způsobem, například u plain-IV je tvořen čísly sektorů, u typu ESSIV (Encrypted Salt-Sector IV) je vytvořen hašovací funkcí, více [23]. Většinou je IV přenášený otevřenou formou, ale nesmí být nikdy použit opakovaně se stejným klíčem. V otevřené formě vystupuje pokud nezávisí na klíči. Opakované použití může vést k úniku informací o prvním bloku (mód CBC), nebo se dokonce porušuje bezpečnost použitého šifrovacího algoritmu (mód OFB). Na principu úniku informací o IV je založena kryptoanalytická metoda TLS CBC IV útoku (TLS CBC IV attack), viz [22]. Útočník může na základě znalosti IV specifikovat další blok otevřeného textu, pokud je šifrován stejným šifrovacím klíčem. Například u CBC je řešením nepředvídatelnost IV v čase.

Módy blokových šifer jako jsou CBC, ECB, OFB a CFB poskytují současně pouze jednu z vlastností důvěrnost nebo integrita. Útočník je často schopen pozměnit text i bez znalosti klíče aniž by byla změna odhalitelná. Pro zaručení důvěryhodnosti se ověřuje IV a šifrovaný text, k čemuž slouží autentizační kódy jako je MAC. Jako jedno z kritérií u módů blokových šifer se uvažovala podmínka integrity, která je patrná u šíření chyb. Novější módy jako jsou OCB, CCM, IAPM, a GCM zajišťují integritu i důvěrnost v jedné vrstvě. Vedle výše uvedených módů existují specializované módy navržené pro bezpečné šifrování sektorů na disku. Do této skupiny patří LRW (tweakable narrow-block encryption) mód a módy CMC a EME (wide-block encryption), více [23].

V režimu módů ECB a CBC (PCBC) lze šifrovat (dešifrovat) pouze celé bloky. Je tedy potřeba text upravit, nebo použít nějakou metodu, která bloky připraví. Uvedené módy tedy nelze použít v případě, kdy zprávy nekončí přesně na hranicích bloku. Hovoří se procesu zarovnávání. Nejjednodušší metodou pro zarovnání je doplnění posledního bloku pravidelnou strukturou (nulami, jedničkami atd.), více k metodám zarovnávání viz. [21]. Pokud je ale potřeba, aby byl šifrovaný text stejně dlouhý jako nešifrovaný, například z důvodu zamezení potenciálních útoků, existují řešení:

- poslední kompletní blok se zašifruje ještě jednou.
- z posledního kompletního bloku se vezme n bitů a posloupnost se naxoruje na n bitů závěrečného necelého bloku.

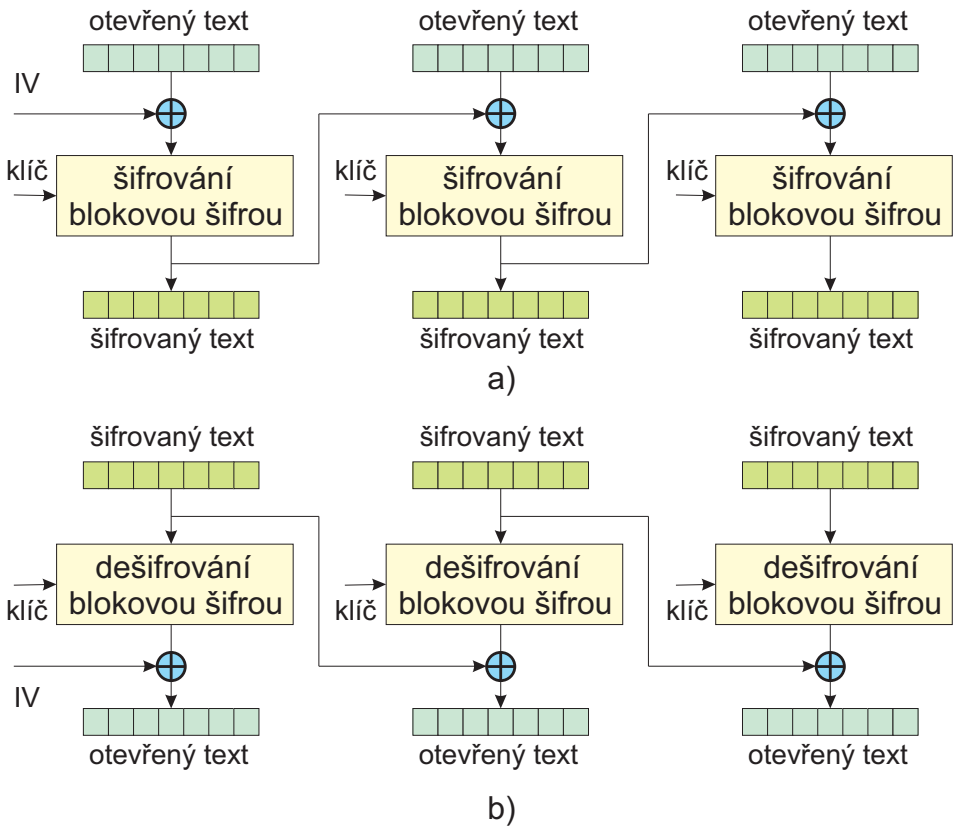
V režimech CFB, OFB, CTR lze pracovat s daty, které jsou rozdělené po menších částech. Dochází k transformaci blokové šifry na proudovou a to způsobem, kdy se šifrovací transformace použije ke generování proudu klíče. Blokové šifry v proudových modech (CFB, OFB, CTR) přenáší negativní vlastnost od synchronních proudových šifer a to je potenciální útok změnou zašifrovaného textu. Útok je založen na skutečnosti, že se změna $C \oplus x$ v zašifrovaného textu projeví v otevřeném textu jako $O \oplus x$.



Obrázek 26. Na obrázku a) je znázorněno šifrování v módu ECB. Na obrázku b) je znázorněno dešifrování v módu ECB.

Elektronická kódová kniha (Electronic codebook (ECB)) viz. schéma 26. je nejjednoduší způsob přípravy bloků na šifrování. Otevřený text je rozdělen do požadovaných bloků a každý blok je šifrován samostatně. Nevýhodou ECB je, že se stejné bloky otevřeného textu zašifrují do stejných šifrovaných bloků, což je jeden z problémů klasické substituce. Může tedy docházet k odkrytí obsahu zprávy. Dalším problémem u ECB je tzv. vyzařování zašifrovaného obsahu. Pokud se při šifrování bitmapy použije mód ECB, pak lze z šifrovaného obsahu vidět obrysy původních tvarů. Při použití jiného módu se již jeví obsah jako náhodný šum. Další nevýhodou použití ECB je možnost změny šifrovaného obsahu (replay attack), což vede ke ztrátě smyslu původní zprávy po dešifrování. Pokud útočník disponuje otevřeným i šifrovaným textem a dokáže pozměnit potřebnou část textu, stává se změna těžce odhalitelná. Dochází ke ztrátě integrity obsahu otevřeného textu. Šifrování v módu ECB se dá použít v případě, kdy jako vstup figurují náhodné binární řetězce.

Řetězení šifrového textu (Cipher-block chaining (CBC)) viz. schéma 27. je operační mód, který před procesem šifrování mění aktuální blok otevřeného textu předchozím blokem. Všechny bloky s výjimkou prvního jsou před samotným



Obrázek 27. Na obrázku a) je znázorněno šifrování v módu CBC. Na obrázku b) je znázorněno dešifrování v módu CBC.

šifrováním modifikovány pomocí operace XOR předchozím zašifrovaným blokem. Na prvním blok je naxorován inicializační vektor IV.

Jestliže se budou bloky číslovat vzestupně a prvnímu bloku bude přiřazen index jedna, pak lze šifrování v módu CBC popsat zápisem:

$$C_i = E_k(P_i \oplus C_{i-1}),\ C_0 = IV$$

Dešifrování v módu CBC lze popsat formulí:

$$P_i = D_k(C_i) \oplus C_{i-1},\ C_0 = IV$$

Moderní blokové šifry disponují dobrou difúzí i konfúzí, ale tato vlastnost je zaručena pro aktuální šifrovaný blok. Vlastnost inicializačního vektoru IV zajistí, že bude po šifrování první blok náhodný. Pokud se vychází ze skutečnosti, že je předchozí blok náhodný, pak bude i aktuální blok náhodný. Blok tedy závisí na posloupnosti předchozích bloků. Pokud by se tedy stejný otevřený text dvakrát

zašifroval v módu CBC, vypadal by díky přenosu náhodnosti výsledný šifrovaný text v obou případech odlišně. Výhodou závislosti na přechozích blocích je samosynchronizace, tedy schopnost obnovy při výpadku některého bloku. Vlastnost samosynchronizace umožní zrekonstruovat otevřený blok textu pomocí dvou po sobě jdoucích šifrovaných bloků. Nevýhodou jsou možné potenciální útoky založené na vzájemné závislosti bloků.

CBC mód byl jeden z nejpoužívanějších módů. Vzhledem k rozšířenosti módu CBC byla zkoumána bezpečnost použití a možné útoky na základě znalosti bloků, vodoznaků atd., více [23]. Problémem módu je samotná závislost bloků, protože je potřeba provádět šifrování sekvenčně. Dešifrování lze provést paralelně. Jednobitová chyba v šifrovaném bloku C_i poruší odpovídající blok P_i otevřeného textu a invertuje příslušný bit v následujícím bloku P_{i+1} otevřeného textu. Mód také požaduje, aby byl otevřený text přesně rozdělený na požadované délky bloků šifry. Některé texty nelze přesně rozdělit, proto je potřeba použít tzv. metodu krádeže šifrovaného textu (ciphertext stealing (CTS)) viz. [32]. Metoda krádeže šifrovaného textu umožňuje rozdělit text, který nelze jednoduše rozčlenit do rovnoměrných bloků bez expanzního vlivu na šifrovaný text.

Propagační CBC (propagating cipher-block chaining (PCBC), plaintext cipher-block chaining) viz. scéma 28. je mód, který vznikl z CBC z důvodu lepší odhalitelnosti případné chyby. Malá chyba (i jednobitová) se narozdíl od CBC šíří do dalších bloků a příjemce jednoduše výskyt chyby odhalí a text odmítne.

Pokud opět očíslujeme jednotlivé bloky textu, pak šifrování v módu PCBC lze popsat zápisem:

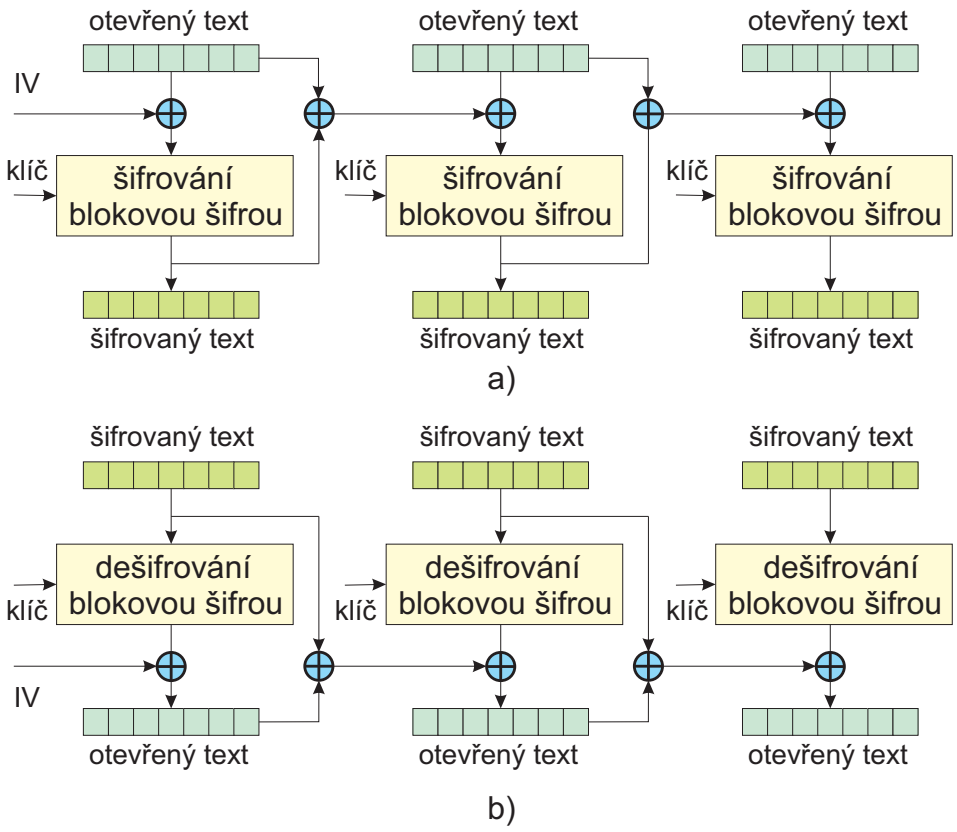
$$C_i = E_k(P_i \oplus P_{i-1} \oplus C_{i-1}),\ P_0 \oplus C_0 = IV$$

Dešifrování v módu CBC lze popsat formulí:

$$P_i = D_k(C_i) \oplus P_{i-1} \oplus C_{i-1},\ P_0 \oplus C_0 = IV$$

PCBC není standardizován ani není příliš rozšířen. Používá ho autentizační protokol Kerberos v4 (viz. [24]) a peer-to-peer protokol WASTE (viz. [31]). Pokud jsou dva vedlejší šifrované bloky prohozeny, nemá záměna vliv na výsledný dešifrovaný text. I z tohoto důvodu není PCBC používán v dalších verzích protokolu Kerberos.

Módy zpětné vazby z šifrovaného textu (CFB) a z výstupu (OFB) jsou operační módy, které transformují blokovou šifru na proudovou. Převod je vlastně použití blokové šifry ke generování hesla, které se podobně jako u proudových šifer naxoruje na otevřený text. Ke generování hesla je potřeba konečný automat, který se uvede do náhodné polohy dle inicializačního vektoru IV. První blok hesla tvoří zašifrování IV. Automat vzniklý šifrovaný text u módu CFB

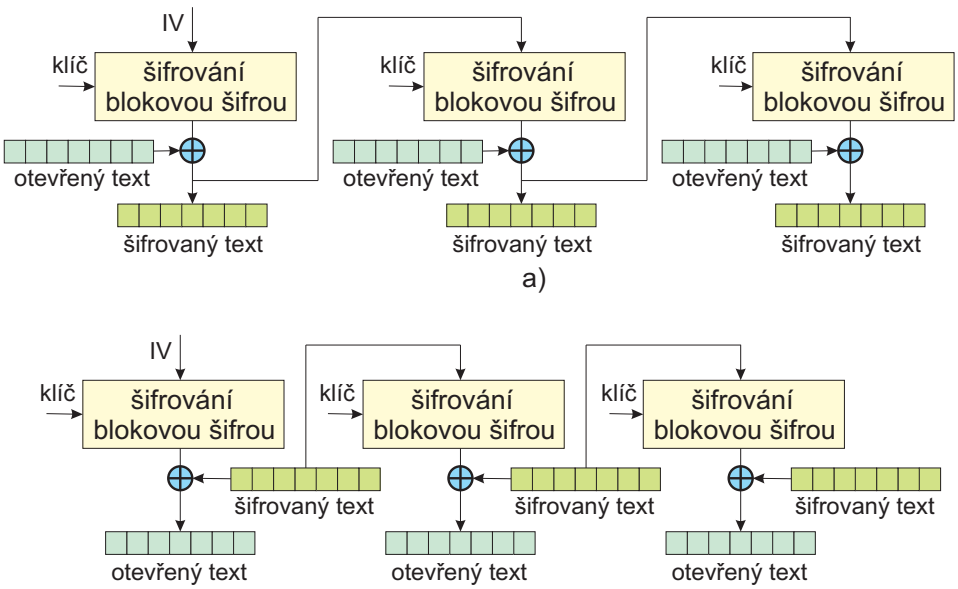


Obrázek 28. Na obrázku a) je znázorněno šifrování v módu PCBC. Na obrázku b) je znázorněno dešifrování v módu PCBC.

resp. vzniklé heslo u módu OFB pošle na vstup blokové šifry a jeho zašifrováním je vyprodukován následující blok zpětné vazby. Bloková šifra je potřeba u obou módů pouze jednosměrně. Používá se jen transformace E_k, což je vhodné pro harwarovou realizaci. Více informací viz. [33].

Pro ukládání a manipulaci se zpětnou vazbou je využit zpětnovazebný registr viz. 5.2.5. Při odebírání vstupu se nemusí použít celý blok, ale i jeho část. Pokud se tedy odebere x bitů hesla u módu OFB resp. x bitů vzniklého šifrovaného textu u módu CFB a vloží-li se x bitů zprava do registru, posune se obsah registru a x bitů zleva vypadne.

- **Mód zpětné vazby z šifrovaného textu (Cipher feedback (CFB))** viz. schéma 29. má vlastnost samosynchronizace, která závisí na délce zpětné vazby. Vlastnost samosynchronizace pomocí dvou po sobě jdoucích šifrovaných bloků je podobná jako u módu CBC. Problémem je, že opět potenciální ztráta jednoho bitu (bajtu) šifrovaného bloku C_i poruší odpo-



Obrázek 29. Na obrázku a) je znázorněno šifrování v módu CFB. Na obrázku b) je znázorněno dešifrování v módu CFB.

vídající dešifrovaný blok P_i. Jako řešení uvedeného problému CFB používá pro načítání vstupů zpětnovazebný registr.

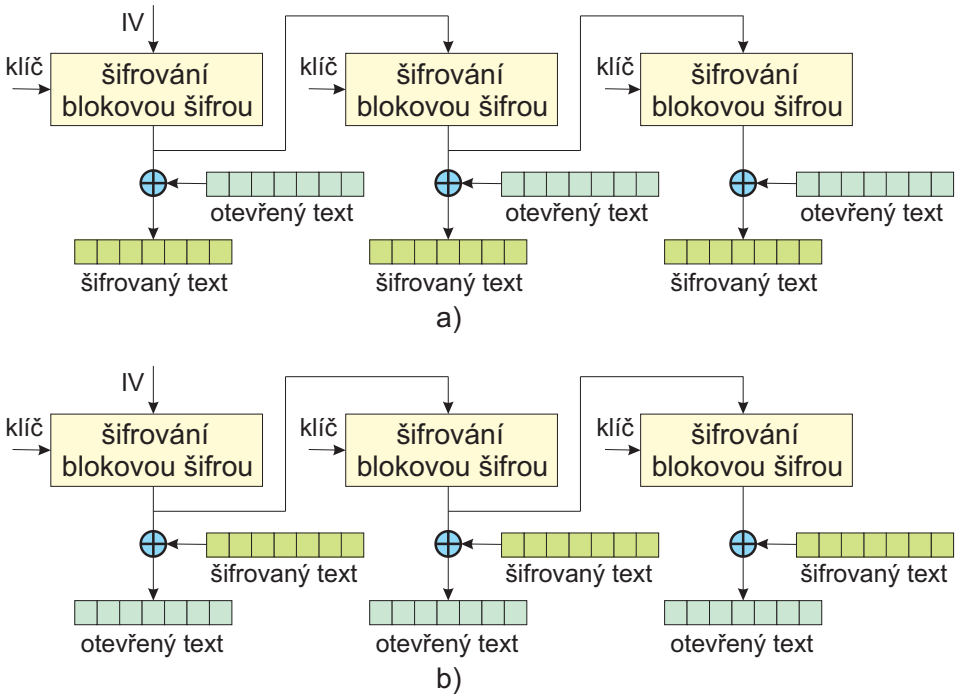
Použití módu CFB pro vytvoření samosynchronizované proudové šifry, která je schopna obnovit ztrátu x bitů, začíná počáteční inicializací zpětnovazebného registru inicializačním vektorem IV. Velikost registru je nastavena dle délky bloku. Nyní se obsah registru zašifruje blokovou šifrou a nejvyšších x bitů se naxoruje na x bitů otevřeného textu. x bitů z výstupu se vloží do registru, postupem popsaným výše. Proces se opakuje s dalšími x bity otevřeného textu. Dešifrování je obdobné, na začátku se provede inicializace dle IV, následuje šifrování pomocí blokové šifry. Dále se naxoruje nejvyšších x bitů výstupu s x bity šifrovaného textu a výsledkem je x bitů otevřeného textu.

Proces lze popsat pomocí následujího zápisu, kde S_i je i-tý stav zpětnovazebného registru, $b \ll x$ značí posunutí o x bitů vlevo, $head(b, x)$ značí nejvyšších x bitů z b a n je počet bitů inicilizačního vektoru.

$$P_i = head(E_k(S_{i-1}), x) \oplus C_i$$

$$C_i = head(E_k(S_{i-1}), x) \oplus P_i$$

$$S_i = ((S_{i-1} \ll x) + C_i) \mod 2^n$$



Obrázek 30. Na obrázku a) je znázorněno šifrování v módu OFB. Na obrázku b) je znázorněno dešifrování v módu OFB.

$$S_0 = IV$$

Pokud dojde ke ztrátě x bitů z šifrovaného textu, bude výstupní otevřený text nesprávný, ale zpětnovazebný registr si bude udržovat stav při šifrování, což bude bod od kterého proběhne resynchronizace. Podobně jako u CBC není možno provést fázi šifrování paralelně, protože se informace z otevřeného textu šíří do dalších bloků šifrovaného textu. Dešifrování lze realizovat paralelně. Jednobitová chyba v dešifrovací fázi ovlivní jednak příslušný blok otevřeného textu jednobitovou chybou, ale také kompletně poruší následující blok otevřeného textu.

CFB má podobně jako OFB a CTR dvě veliké výhody: bloková šifra je použita pouze pro šifrování a zprávu není potřeba dělit podle velikosti bloku šifry.

- **Mód zpětné vazby z výstupu (Output feedback (OFB))** viz. schéma 30. je režim tvořící synchronní proudovou šifru. Mód generuje proud hesla ve formě bloků, které jsou následovně naxorovány s bloky otevřeného textu. Výstupem jsou bloky šifrovaného textu. Podobně jako u proudových šifer

dochází při změně bitu v šifrovaném textu ke změně příslušného bitu v otevřeném textu, což umožňuje aplikaci samoopravných kódů (například Hammingovy kódy).

Vzhledem k symetrii operace XOR je šifrování a dešifrování stejné. Označíme-li výstup blokové šifry písmenem O a indexem aktuální blok, pak lze šifrování resp. dešifrování popsat zápisem:

$$C_i = P_i \oplus O_i$$

$$P_i = C_i \oplus O_i$$

$$O_i = E_k(O_{i-1})$$

$$O_0 = IV$$

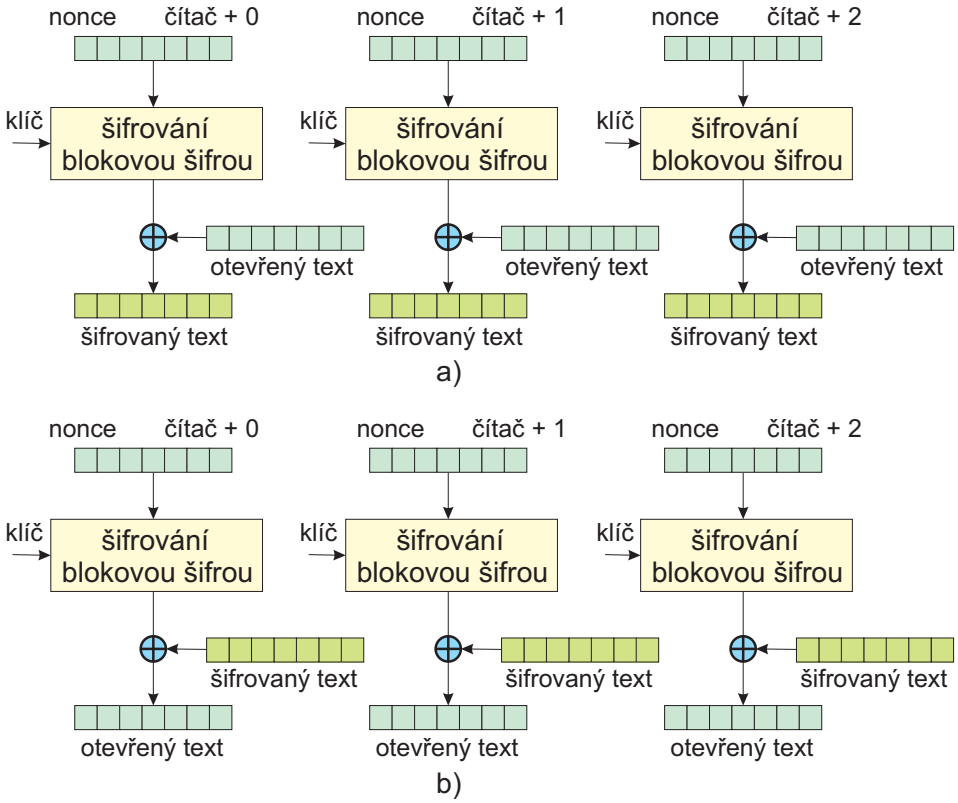
Generování hesla je založeno na principu konečného automatu, který má maximálně 2^N stavů. Po překročení počtu kroků se generování hesla periodicky opakuje. Délka periody hesla je maximálně 2^N bloků a pohybuje se v intervalu $< 1, N >$. Jeho délku určuje hodnota inicializačního vektoru IV. Struktura hesla je rozdílná a je závislá na délce periody a nejlepších výstupů dosahuje pokud je perioda maximální, viz [34].

Výstup zpětné vazby blokové šifry je závislý na předchozí operaci, takže je nutné provádět výpočet sériově. Vylepšením je postup, kdy se operace blokové šifry předem připraví a pak lze výpočet pomocí operace XOR provést paralelně.

Použití módu OFB je vhodné pro vysokorychlostní systémy s nepřístupným šířením chyb. Hodí se tedy pro systémy s vyšším výskytem chyb.

Čítačový mód (Counter (CTR)) viz. schéma 31. je mód, který transformuje blokovou šifru na symetrickou proudovou šifru. Je založen na podobném principu jako OFB. Proud klíče je generován pomocí čítače, což může být libovolná funkce, která garantuje neopakovatelnost již použité hodnoty. CTR umožňuje náhodný přístup během dešifrování. CTR se hodí pro operace na multiprocesorových počítačích s paralelním přístupem při šifrování. CTR se používá v případě potřeby šifrování resp. dešifrování libovolného bloku bez ohledu na ostatní. Heslo je možno vypočítat jen na základě inicializačního vektoru IV a pozice v otevřeném textu. CTR nemá možnost samosynchronizace a ztráta šifrovaného textu způsobí chybné odšifrování od místa chyby do konce textu.

CTR již nemá problém s neznámou délkou hesla jak tomu bylo u OFB. Délka hesla je dána periodou čítače, která je specifikována. Heslová posloupnost bude maximální a to díky vlastnosti bijektivního zobrazení u blokové šifry. Různé hodnoty čítače se zobrazí na různé hodnoty bloků hesla.



Obrázek 31. Na obrázku a) je znázorněno šifrování v módu CTR. Na obrázku b) je znázorněno dešifrování v módu CTR.

CTR používá inicializační vektor IV resp. nonce, který se na začátku načítá do do registru čítače. Po inicializačním načtení vzniká první blok hesla. Pro vygenerování aktuálního bloku čítače můžou být konkatenovány, sečteny, naxorovány atd. Nejčastěji dochází k aktualizaci čítače přičtením jedničky. Heslo může být využito v plné šíři bloku, nebo jen jeho část. Inkrementace jedničkou je jen jedna z možností, ale nikdy nesmí dojít k opakovanému vygenerování stejného hesla. Pokud by se heslo vyskytlo opakovaně, mohlo by dojít k porušení bezpečnosti.

Jestliže označíme jednotlivé bloky indexy, lze šifrování resp. dešifrování obecně popsat:

$$C_i = P_i \oplus E_k(IV, K_i)$$

$$P_i = C_i \oplus E_k(IV, K_i)$$

Při aktualizaci čítače přičtením jedničky lze šifrování pro B bitů čítače a pro $i = 1, 2 \ldots$ popsat:

$$CTR_i = (IV, i - 1) \mod 2^B$$

$$H_i = E_k(CTR_i)$$

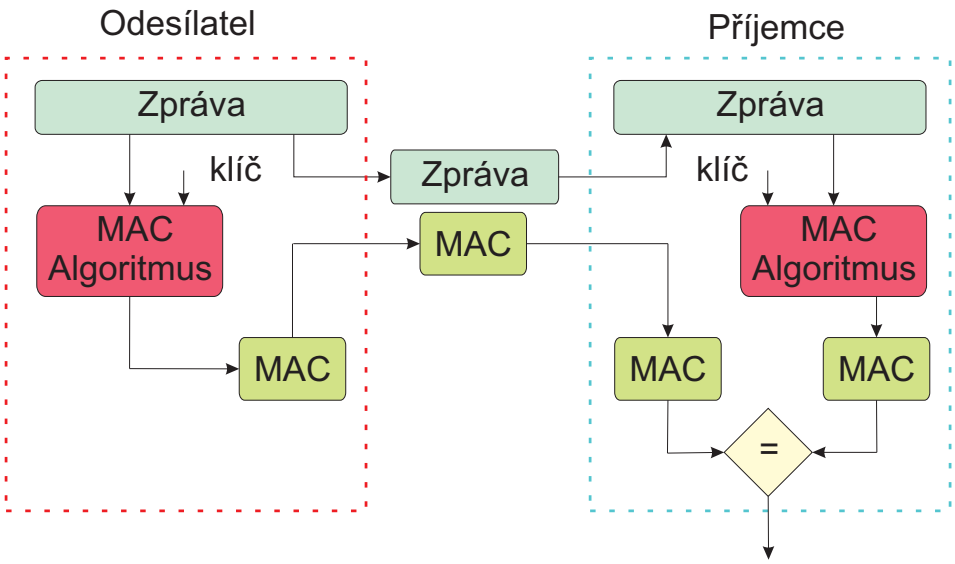
$$C_i = O_i \oplus H_i$$

a dešifrování:

$$CTR_i = (IV, i-1) \mod 2^B$$
$$H_i = E_k(CTR_i)$$
$$O_i = C_i \oplus H_i$$

CTR je díky své povaze oblíbený, i když bylo použití jednoduché vstupní deterministické funkce kritizováno jako zvyšování bezpečnostních rizik. V současné době je CTR uznáván, protože bylo prokázáno, že problémy vyplývající ze vstupu funkce jsou slabostí základní blokové šifry. Nicméně existují specializované útoky jako je Hardware Fault viz. [35], který je založen na využití jednoduchého počítadla vstupu funkce.

Autentizační kód zprávy (Message authentication code (MAC)) je operační mód, který zajišťuje integritu dat. Pomocí MAC se dá ověřit správnost a autenticita zprávy. MAC je krátký výstupní kód, který vznikne ze vstupu v podobě tajného klíč a zprávy pro ověření. Tajný klíč musí být jiný než je použit pro šifrování.



Obrázek 32. Autentizační kód zprávy (MAC). Princip komunikace mezi odesílatelem a příjemcem.

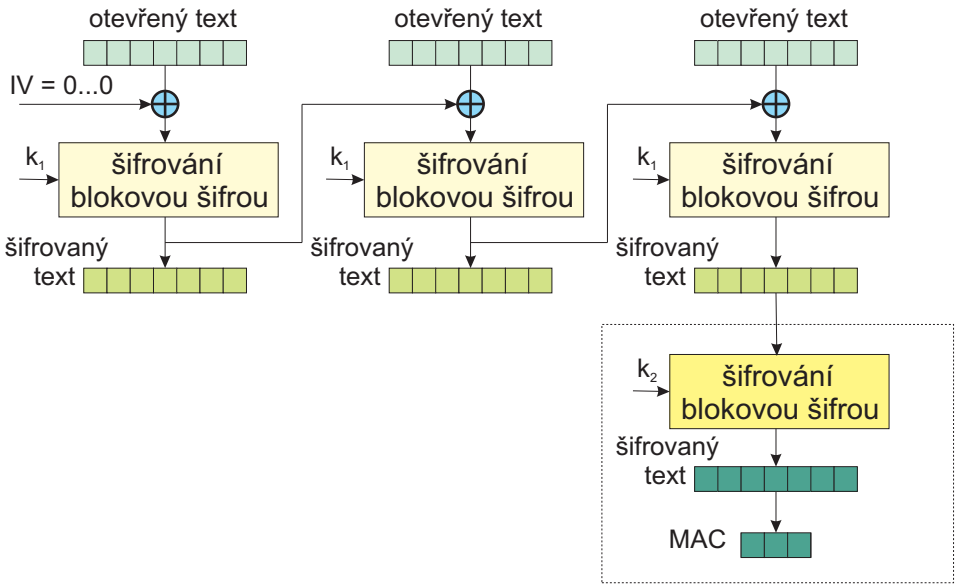
V souvislosti s MAC se používá termín MIC (Message integrity code), který nepoužívá tajný klíč. MIC nezaručuje integritu dat a je náchylný k neoprávněné

manipulaci. Algoritmus MAC je navržen tak, aby produkoval stejný kód jen pokud dostane stejná vstupní data v podobě zprávy, tajného klíče a inicializačního vektoru. Komunikující strany si kromě přenášené zprávy posílají i autentizační kód zprávy. Příjemce se rozhodne na základě porovnání mezi vypočtenou hodnotou MAC a doručenou hodnotou MAC zda je zpráva původní. Princip komunikace mezi příjemcem a odesílatelem je znázorněn na obrázku 32.

Algoritmus MAC je někdy nazýván klíčovou hašovací funkcí, která má narozdíl od klasické hašovací funkce rozdílné požadavky na bezpečnost jakými je odolnost vůči kryptoanalytické metodě volby otevřených textů (chosen-plaintext attack) a odolnosti vůči existenciálnímu padělání (existential forgery). Existenciální padělání je schopnost generovat pár zpráva-podpis, kde podpis není originální dle původního majitele. Více informací viz. [37]. Útočník je tedy schopen vytvořit MAC dle zadaného vstupu, ale nedokáže odhadnout MAC pro jiné zprávy, protože je to výpočetně neproveditelné. MAC je rozdílný i proti digitálnímu podpisu, protože obě komunikující strany používají pro ověření stejný tajný klíč. MAC je tedy symetrický a každá strana je schopna také generovat MAC pro jiné zprávy. Narozdíl tomu se u digitálních podpisů zastává asymetrický přístup, který zaručuje jedinečnost podpisu díky privátnímu klíči vlastníka. MAC díky své symetrické povaze nezaručuje nepopíratelnost dat, což neumožňuje třetí straně rozhodnout o autoru zprávy.

Algoritmy MAC jsou založeny na různých kryptografických metodách jako jsou hašovací funkce (například HMAC), nebo blokové šifry (například OMAC, PMAC, CBC-MAC). Nejvíce používané jsou MAC algoritmy založené na univerzálním hašování viz. [38] jako je například VMAC.

Jeden z možných výpočtů MAC, konkrétně se jedná o CBC-MAC, lze popsat pomocí upraveného operačního módu CBC, viz obrázek 33. Inicilializační vektor je nulový a pro MAC je použit poslední šifrovaný blok. Průběžné šifrované bloky se nikam neodesílají. Na poslední šifrovaný blok, je možno aplikovat dodatečné šifrování $E_k(2)$, jehož podoba je libovolná. Pro výsledný autentizační kód se dá použít pouze požadovaná délka bloku, která vznikne z původního bloku zkrácením.



Obrázek 33. Autentizační kód zprávy (MAC). Princip CBC-MAC.

6. Asymetrické šifrovací systémy

Asymetrický šifrovací systém je definován jako systém, kde pro skoro všechna $k \in K$ platí, že nelze z šifrovací transformace E_k určit dešifrovací transformaci D_k. Klíč k slouží k vytvoření počátečního utajeného nastavení, ze kterého se transformací odvodí dvojice tajný a veřejný klíč. Definice byla uvedena v kapitole 3.1.. Jednoduše řečeno, asymetrické šifrovací metody používají pro šifrování a dešifrování různé klíče. Asymetrické šifrovací systémy jsou obecnějším názvem pro systémy s veřejným klíčem, na které se v následujících kapitolách zaměříme.

Kryptografické systémy s veřejným klíčem lze rozdělit do tří základních skupin:

- šifrovací algoritmy s veřejným klíčem - veřejný klíč slouží k zašifrování textu a tajný klíč k dešifrování a je přístupný pouze oprávněným osobám. Klíče parametrizují šifrovací a dešifrovací transformace.
- systémy pro výměnu klíčů - umožňují komunikujícím stranám zajištění utajené komunikace v méně bezpečném prostředí. Komunikace je pak šifrována nějakou symetrickou metodou.
- digitální podpisy - autor zprávy text zašifruje (podepíše) pomocí tajného klíče a každý kdo má přístup k veřejnému klíči si podpisem zprávu ověří.

Systémy s veřejným klíčem jsou založeny na jednocestných funkcích, které využívají zakladní vlastnosti tzv. padacích dvířek. Odolnost systémů je přímo závislá na komplikovanosti nalezení inverzní funkce. Principy, na kterých bezpečné asymetrické systémy stojí, lze v současnosti řešit pouze s exponenciální složitostí.
Kryptografické systémy s veřejným klíčem jsou důležitou kapitolou kryptografie, jsou využívány v mnoha oblastech jako jsou digitální podpisy, transportní protokoly (např. TSL), systémy pro výměnu klíčů atd.

Základní rozdíly mezi symetrickým a asymetrickým systémem:

- Symetrické šifrovací systémy potřebují nějaký bezpečný kanál pro výměnu klíče. V případě asymetrických šifrovacích systémů stačí komunikujícím stranám veřejný klíč, kterým se zpráva zašifruje a pouze držitel tajného klíče zprávu dešifruje.

- Asymetrické šifrovací algoritmy mají většinou větší výpočetní náročnost než symetrické algoritmy. Jsou založeny na obtížně řešitelných úlohách jako je faktorizace velkých čísel na prvočísla, nebo výpočet diskrétního logaritmu.

Jedním z hlavních problémů, který systém s veřejným klíčem přináší, je důvěryhodnost zdroje. Můžeme sice mít veřejný klíč, ale není zaručená jeho autentičnost, tedy zda s ním nebylo manipulováno. Řešením problému je použití nějaké certifikační autority v podobě třetí strany tzv. PKI (public-key infrastructure) [93], která autentičnost zajistí. Certifikační autorita zaručují vlastnictví klíčových párů. Rozšířením jsou PGP (Pretty Good Privacy) [92], které vedle certifikace zaručí i spojení mezi uživateli tzv. web of trust. Doposud neexistuje uspokojivé řešení problému autentizace veřejných klíčů.

6.1. Bezpečnost kryptografických systémů s veřejným klíčem

Narozdíl od symetrických šifer mají asymetrické systémy hlavní nevýhodu a to je dostupnost veřejného klíče. Asymetrické systémy jsou náchylné na útoky hrubou silou, ale pro většinu používaných algoritmů neexistuje dostupný výpočetní výkon pro jeho provedení. Pokud by hrozilo nebezpečí prolomení, zvýší se délka použitého klíče.

Bezpečnost šifrovacích algoritmů s veřejným klíčem závisí i na jejich realizaci. V praktické části je ukázáno nevhodné použití bezpečného RSA. Problém je viditelný u certifikačních autorit, protože ručí za identitu přidělených klíčů a vydaných certifikátů. Vydaný certifikát má většinou delší platnost a závisí na něm významná soukromá data jako jsou bankovní certifikáty. Špatně navržené certifikáty mohou být náchylné na man-in-the-middle útoky, což je typ útoku

aktivního odposlechu mezi komunikujícími stranami, vzniknou tak podvržené certifikáty, které se pak vydávájí za originál. Útoky jsou realizovatelné hlavně v nezabezpečených sítích jako jsou otevřené bezdrátové sítě.

Je známo několik slibně vypadajících asymetrických algoritmů, které vykázaly značné nedostatky. Mezi ně patří algoritmus založený na úloze batohu [94]. Dalším z často používaných útoků je útok bočník kanálem. Mezi úspěšné příklady patří útoky, které jsou založeny na přesném měření času šifrování otevřeného textu s jehož pomocí se získaných údajů odvodí dešifrovací klíče.

6.2. Diffie-Hellman

Diffie-Hellman (Diffie-Hellman Key Agreement Method) protokol byl navržen k bezpečné výměně klíčů. Je založen na problému řešení diskrétního logaritmu. Umožňuje dvěma stranám vytvořit na otevřeném kanálu bezpečnou šifrovanou komunikaci. Protokol nese jméno po Whitfield Diffie a Martin Hellman, kteří jej v roce 1976 věřejně publikovali. Je prvním úspěšným protokolem pro výměnu klíčů. Pomocí asymetrického šifrování dojde k výměně klíčů a následovná komunikace probíhá již pomocí symetrického šifrování. Platnost Diffie-Hellman protokolu skončila v roce 1997.

6.2.1. Princip Diffie-Hellman

Přepokládejme, že existují dvě komunikující strany A a B, které se shodnou na parametrech u a p. Strana A si zvolí tajný klíč x a odvodí si zprávu $w = u^x \,(mod\, p)$. Strana B si zvolí tajný klíč y a odvodí si zprávu $w' = u^y \,(mod\, p)$. Nyní si strany zprávy w a w' vymění. Nyní si strana A vypočítá klíč $k = w'^x \,mod\, p$ a $k' = w^y \,(mod\, p)$. Z vlastnosti cyklické grupy platí, že

$$k = (u^x)^y \,(mod\, p) \;=\; (u^y)^x \,(mod\, p) = k'.$$

Komunikující strany se dohodly na tajném klíči k, kde $g = k = u^{xy}$ je prvek grugy $|G|$, který budou používat v další symetricky šifrované komunikaci.

Diffie-Hellman lze použít pro výměnu klíčů mezi více komunikujícími stranami.

6.2.2. Bezpečnost Diffie-Hellman

Protokol je považován za bezpečný, pokud je dobře zvolena grupa G (dostatečně velké prvočíslo) a její prvek g. Útočník, který zná jen veřejně dostupné data (u, p, w a w') musí řešit problém diskrétního logaritmu (nalezení x a y), na který v současnosti neexistuje efektivní řešení. Pro výpočet p je doporučeno používat bezpečné prvočíslo q, kde $p = 2q+1$. Grupa G je pak dělitelná 2 a tímto prvočíslem. Tajná čísla x a y jsou použita pouze pro jedno sezení.

Diffie-Hellman je náchylný k man-in-the-middle útokům. Pokud se třetí strana napojí do komunikace mezi A a B, může předstírat komunikaci druhé strany, přijmout jejich identitu a zasílat příjemcům podvržené zprávy. Během používání Diffie-Hellman protokolu vyplula otázka autentizace komunikujícího, který řešil např. protokol Station-Station (STS) [95]. Problém lze vyřešit pomocí komunikace vyžadující ověření důvěryhodnosti podpisu vydaného třetí stranou.

6.3. ElGamal

ElGamal je asymetrický šifrovací algoritmus, který je založený na řešení problému diskrétního algoritmu a vychází z Diffie-Hellman schématu pro výměnu klíčů. Byl popsán Taherem Elgamalem v 1984 a je stejně jako RSA využíván k šifrování, tak i pro digitální podpisy ve variantě DSA (Digital Signature Algorithm). Šifrování ElGamal je definováno na libovolné cyklické grupě G. V praxi se však více využívá RSA, protože je pro šifrování mnohem efektivnější. ElGamal potřebuje pro zašifrování stejného textu asi dvakrát více místa než RSA.

6.3.1. Princip ElGamal

ElGamal se skládá se tří fází:

1. Generátorování klíčů - strana A zvolí vhodná celá čísla x, g, y a p, kde $q < p$, p je prvočíslo a x je tajný klíč, pro který platí, že $y = g^x \,(mod\, p)$. Veřejný klíč je dán jako (g, p, y).

2. Šifrování - strana B si nejprve zvolí nějakou jednorázovou tajnou hodnotu k a vytvoří si dvě zprávy $a = g^k \,(mod\, p)$ a $b = y^k \cdot M \,(mod\, p)$, kde M je otevřený text. Jako šifrovaná zpráva složí dvojice (a, b).

3. Dešifrování - strana A dostane dvojici (a, b). Původní zprávu M dešifruje pomocí tajného klíče x z předpisu:

$$M = \frac{a}{b^x} \,(mod\, p)$$

.

Chceme-li vyjádřit zápis přesněji, budeme pracovat s multiplikativní grupou G řádu q s generátorem g. Veřejný klíč je pak dán jako (G, g, p, y). Šifrová zpráva je dána ve tvaru

$$(a, b) \;=\; (g^k, M' \cdot y^k) \;=\; (g^k, M' \cdot (g^x)^k),$$

kde M' je zkonvertovaná zpráva M jako element grupy G. Dešifrování probíhá dle zápisu:

$$b \cdot s^{-1} = M' \cdot y^k \cdot (g^{xk}) = M' \cdot g^{xk} \cdot g^{-xk} = M',$$

kde $s = a^x$.

ElGamal se používá v kombinaci se symetrickým šifrováním. Nejprve se zpráva zašifruje nějakým symetrickým šifrováním a šifrovací klíč je zašifrován pomocí ElGamal. Jedná se o tzv. hybridní kryptosystémy.

6.3.2. Bezpečnost ElGamal

Bezpečnost ElGamal závisí na volbě grupy G a jejich vlastností. Důležitá je také volba způsobu doplnění zprávy M resp. její konverze do elementu grupy G. Hovoří se o rozhodovacích předpokladech Diffie-Hellman (Decisional DiffieHellman assumption (DDH)), které se také používají pro stanovení grup u ElGamal. Jedná se o mechanismus testování a výběru vhodných podgrup grupy G při řešení diskrétních logaritmů na cyklické grupě G.

Z vlastností ElGamal plyne, že je náchylný na chosen-ciphertext útoky. Platí totiž, že lze pro šifrovanou dvojici (a, b) nějakého neznámého M setrojit šifrovanou dvojici $(a, 2b)$ pro zprávu $2M$. Chosen-ciphertext útokům se dá vyvarovat třeba použitím nějakého bezpečného mechanismu doplňování zpráv. Alternativy ElGamal jako je například Cramer-Shoup [98] vycházejí z předpokladu, že k zamezení chosen-ciphertext útoku stačí přísné dodržování podmínek pro DDH.

6.3.3. Schéma digitálního podpisu ElGamal

Algoritmus digitálního podpisu ElGamal není tak rozšířený jako jeho varianta DSA, kterou vyvinulo NSA. Podpis umožňuje ověření autentičnosti při komunikaci na nezabezpečeném kanále.

Algoritmus používá nějakou bezkolizní hašovací funkci H. Zvolí si dostatečně velké prvočíslo p, pro které je výpočet diskrétního logaritmu *modulo p* obtížná úloha. Nyní si zvolí nějaký generátor g mutiplikativní grupy celých čísel G.

1. Generování klíče - nejprve se zvolí nějaké tajné x, kde $1 < x < p - 1$. Nyní se vypočítá hodnota $y = g^x \,(mod\, p)$ a publikuje se veřejný klíč (p, g, y).

2. Generování podpisu - autorita si zvolí nějaké tajné k, kde $0 < k < p - 1$ a $gcd(k, p - 1) = 1$. Podpis zprávy M tvoří dvojice (r, s), kterou dostaneme jako
$$\begin{aligned} r &\equiv g^k \,(mod\, p). \\ s &\equiv (H(M) - xr)k^{-1} \,(mod\, p). \end{aligned}$$
Pokud je $s = 0$, pak se výpočet podpisu opakuje. Proces generování podpisu probíhá pro každou novou zprávu M.

3. Ověření podpisu - pro ověření podpisu (r, s) zprávy M se nejprve zjistí, zda je $0 < r < p$ a $0 < s < p - 1$. Pokud první podmínka platí, ověří se podpis ze vztahu $g^{H(M)} \equiv y^r r^s \,(mod\, p)$. Pokud obě podmínky platí, je podpis korektní.

Bezpečnost digitálního podpisu ElGamal je založena na problému řešení diskrétního logaritmu a na problému nalezení kolizní hašovací funkce $H(M) \equiv H(M) \,(mod\, p)$. Oba problémy jsou v současnosti obtížně řešitelné. Pro praktický útok je potřeba zjistit tajné k, které se volí při generování podpisu. Problém nastává, pokud se k použije vícekrát.
Vedle již zmíněných předpokladů byla zveřejněna metoda potenciálního útoku pomocí falšování podpisu.

6.4. Digital Signature Algorithm

Digital Signature Algorithm (DSA) je americký standard pro digitální podpisy. Byl vydán v roce 1991 agenturou NIST jako Digital Signature Standard (DSS). Poslední rozšíření standardu proběhlo v roce 2000 (FIPS-186) a zohledňuje i hašovací funkce z rodiny SHA-2. DSA je variantou schéma digitálního podpisu ElGamal, která využívá pouze podgrupu prvočíselného řádu.

První verze DSS pracuje s klíči o délce 512 až 1024-bitů, druhá verze pracuje s pevnou délkou klíčů 1024-bitů a aktuální třetí verze již bere v úvahu klíče 2048-bitů resp. 3072 bitů.

Mezi varianty DSA se řadí ECDSA (Elliptic Curve DSA) [99], které pracuje s podgrupou eliptických křivek a to záměnou multiplikativní grupy přirozených čísel G_p^* (DSA) za náhodně vygenerovanou podgrupu eliptických křivek. ECDSA používá menší velikosti klíčů než DSA, vychází to z povahy algoritmu ECDSA. Například pro úroveň zabezpečení ECDSA 80-bitů útočník potřebuje ekvivalent 2^{80} pokusů pro případné vygenerování soukromého klíče. Velikost veřejného klíče u DSA je nejméně 1024-bitů, zatímco si ECDSA vystačí pouze s 160-bitovým klíčem (to odpovídá položce tabulky RSA 512-bitů). Princip odvození velikosti podpisu je stejný jak u DSA, tak i u ECDSA: $4t = 4 \cdot 80$, pro úroveň zabezpečení 80-bitů ECDSA.

6.4.1. Princip algoritmu DSA

Předpokládejme, že máme dány přirozené proměnné x, y, p, k, pro které platí:

$$x = y^k \,(mod\, p).$$

Disktrétním logaritmem budeme rozumět proměnnou k.
Princip algoritmu spočívá v rovnosti problému diskrétního algoritmu a malé Fermatovy věty:

$$g = h^{(p-1)/q} \,(mod\, p) \Rightarrow g^q \equiv h^{p-1} \equiv 1 \,(mod\, p).$$

Algoritmus pracuje s multiplikativní grupou přirozených čísel G_p^*, grupa je generovaná prvočíslem p řádu q.

1. Generování klíče - probíhá ve dvou fázích, v první fázi se odvodí parametry, které mohou být sdíleny uživately systému a v druhé fázi se generují veřejné a soukromé klíče pro jednotlivého uživatele.

   (a) Generování parametrů:

   - Setrojíme si hašovací funkci H, kterou můžeme zkrátit na délku klíčů. Nejnovější verze DSS doporučuje použití SHA-2.
   - Nyní je potřeba zvolit dvojici klíčových délek L a N. Pro nejnovější třetí verzi DSS to jsou páry $(1024, 160)$, $(2048, 224)$, $(2048, 256)$ a $(3072, 256)$.
   - Najdeme si prvočíslo p, které má N-bitů. $N \leq$ délka výstupu hašovací funkce.
   - Najdeme L-bitové prvočíslo modulo p, takové že $(p-1) = q$.
   - Nyní si zvolíme nějaký generátor g grupy G_p^*. Pro generátor platí, že $g = h^{(p-1)/q} \, (mod\, p)$, pro nějaké $1 < h < p-1$.
   - Parametry (p, q, g) mohou být sdíleny uživately systému.

   (b) Generování klíčů uživatele:

   - Zvolíme si nějaké tajné x, kde $0 < x < q$.
   - Vypočítáme si $y = g^x \, (mod\, p)$.
   - Veřejný klíč je dán jako čtveřice (p, q, g, y), (p, q, g, x) je klíč soukromý.

2. Generování podpisu - Nechť H je hašovací funkce a M je podepisovaná zpráva.

   - Pro každou zprávu si vygenerujeme hodnotu k, kde $0 < k < q$.
   - Vypočítáme si $r = g^k \, (mod\, p)$. Jestliže je $r = 0$, pak se vrátíme ke generování k.
   - Vypočítáme si $s = k^{-1}(H(M) + xr) \, (mod\, p)$. Jestliže je $s = 0$, pak se vrátíme na začátek a vygenerujeme nové k.
   - Jako podpis figuruje dvojice (r, s).

3. Ověření podpisu:

   - Nejprve ověříme, zda je $0 < r < q$ a $0 < s < q$.
   - Spočítáme $w = s^{-1} \, (mod\, q)$.
   - Spočítáme $a = H(M)w \, (mod\, q)$.
   - Spočítáme $b = rw \, (mod\, q)$.
   - Spočítáme $v = (g^a y^b \, (mod\, p)) \, (mod\, q)$.
   - Podpis je přijat, pokud platí $v = r$.

6.4.2. Bezpečnost DSA

Jedním z hledisek zabezpečení DSA je potřeba utajení hodnoty k, protože při jejím odhalení hrozí možnost úspěšného útoku. I proto je potřeba vygenerovat nové k pro každou zprávu M.
Obě varianty DSA i ECDSA jsou považovány za bezpečné. V případě ECDSA existuje metoda na hledání parametrů tzv. Pollardova [100], jejíž složitost se pohybuje okolo $\sqrt{n}$, kde n je počet bodů křivky. Metoda se kombinuje s Pohlig-Hellmanovou redukcí [101], která neřeší problém diskrétního algoritmu, ale redukuje jej na dílčí problémy v grupě prvočísel.
V roce 2011 byl uveřejněn článek popisující metodu načítání soukromého TSL klíče serveru pomocí OpenSLL (verze 1.0.0e). Metoda ověřuje podpis přes binární pole pomocí timing útoku [102].

6.5. Rabin

Asymetrický kryptografický systém Rabin je alternativou RSA, která bere jako veřejný exponent číslo 2. Rabin má jednoznačné určení bezpečnosti, které přímo závisí na problému faktorizace velkých čísel. Rabin má jednu velikou nevýhodu, která vyplývá z jeho konstrukce. Jako výsledek dešifrování dostaneme čtyři možné otevřené texty, ze kterých je potřeba vybrat ten správný a proto se Rabin často nepoužívá. Rabin vznikl v roce 1979 a jeho autorem je Michael O. Rabin.

6.5.1. Princip algoritmu Rabin

Rabin pracuje ve třech fázích:

1. Generování klíčů:
   - Nejprve se zvolí dvě různá prvočísla p a q, která mají podobnou délku v bitech. Pro ověření prvočíselnosti můžeme využít nějaký test provočíselnosti například nějaký rychlý deteministický test jako je AKS, MillerRabin.
   - Pomocí prvočísel p a q (forma $4k+3$) se vypočítá hodnota pro modulo $n = pq$.
   - Jako veřejný klíč bude vystupovat hodnota n, soukromý klíč tvoří dvojice (p, q). Veřejný exponent není potřeba odvozovat, protože jej bude tvořit hodnota 2. Stejně tak i soukromý exponent není definován.
2. Šifrování vstupního textu:
   - Maximální délka vstupního textu M musí být menší jak hodnota $n-1$. Delší text se rozdělí do bloků a každý blok je zašifrován samostatně.

- Šifrování je poměrně jednoduché $C = M^2 \, (mod \, n)$. V praxi text vystupuje ve formě celého čísla.

3. Dešifrování textu:

   - Pro dešifrování M z C je potřeba odvodit r ze vztahu: $M^2 \equiv C \, (mod \, r)$. Odvození není z hlediska konstrukce šifry Rabin jednoduché, je potřeba odvodit kvadratické kořeny.
   - Kvadratické kořeny odvodíme pomocí theorému Chinese remainder:

$$\begin{aligned} M_p &= \sqrt{C} \, mod \, p. \\ M_q &= \sqrt{C} \, mod \, q. \end{aligned}$$

     Zápis můžeme upravit do podoby:

$$\begin{aligned} M_p &= C^{\frac{p+1}{4}} \, mod \, p. \\ M_q &= C^{\frac{q+1}{4}} \, mod \, q. \end{aligned}$$

   - Aplikujeme rozšířený Eukleidiův algoritmus (extended Euclidean algorithm), protože potřebujeme najít proměnné y_p a y_q ze vztahu: $y_p \cdot p + y_q \cdot q = 1$.
   - Nyní si můžeme vypočítat čtyři kvadratické kořeny r, $-r$, s, $-s$ pro $(C + n\mathbb{Z}) \in \mathbb{Z}/n\mathbb{Z}$:

$$\begin{aligned} r &= (y_p \cdot p \cdot M_q + y_q \cdot q \cdot M_p) \, mod \, n \\ -r &= n - r \\ s &= (y_p \cdot p \cdot M_q - y_q \cdot q \cdot M_p) \, mod \, n \\ -s &= n - s \end{aligned}$$

   - Výsledkem dešifrování Rabin algoritmu jsou čtyři otevřené texty, ze kterých jeden odpovídá původnímu M.

6.5.2. Vlastnosti kryptosystému Rabin

Pokud je původní otevřený text dán nějakým speciálním formátem, nebo byl M číslo, nastává problém v jeho výběru. Byly navrženy metody řešení v podobě úpravy textu před šifrování do předepsané podoby, nebo doplnění textu nějakou speciální vycpávkou. Další řešení uvedli Blum a Williams. Při generování prvočísel se vyberou prvočísla shodné s 3 modulo 4 a tím se omezí problém na doménu kvadratických zbytků, což vede k odstranění nejednoznačnosti výsledku.

Rabin je ve fázi šifrování ryhlejší jak RSA. Ve fázi dešifrování jsou obě metody srovnatelné. Odstranění nejednoznačnosti přináší další režii a zabránilo jeho širokému rozšíření.

Rabin byl původně definován ve zobecněné podobě, která definuje veřejný klíč jako dvojici (n, B), kde $0 \leq B \leq n-1$. Veřejný klíč je stejný tj. (p, q). Šifrování je dáno vztahem: $E(x) = x(x+B)\,(mod\, n)$. Dešifrování je dáno vztahem: $D(y) = \left(\sqrt{\frac{B^2}{4} + y} - \frac{B}{2}\right)(mod\, n)$.

6.5.3. Bezpečnost kryptosystému Rabin

Rabin je tak bezpečný jak je bezpečná samotná faktorizace velkých čísel. Autor uvádí, že pokud někdo odvodí hodnoty r a s, pak bude moci nalézt faktorizaci n, protože platí: $gcd|r-s| = p$, nebo $gcd|r-s| = q$.

Problém faktorizace u Rabin je odlišná než u RSA. V jistém smyslu je tedy bezpečnější jak RSA, protože stojí pouze na problému faktoritace. Doposud neexistuje rychlá a účinná metoda faktorizace velkých prvočísel. S příchodem kvantových počítačů bude možné využít jejich kvantové počítání, které rapidně snižuje složitost problému faktorizace.

Nicméně, stejně jako RSA, Rabin je náchylný vůči chosen-cipher-text útoku. Jisté řešení nabízí redundance textu, například v podobě opakovaní posledního bloku textu. Rabin pak produkuje pouze jeden kořen, který již útočník zná. Při použití této metody není již bezpečnost Rabin rovna problému faktorizace.

6.6. RSA

RSA je šifrovacích algoritmus s veřejným klíčem, který je založený na problému faktorizace velkých čísel a na problému nalezení kořenu m v operaci m^e. Hovoří se o tzv. RSA problému. Faktorizace čísla x spočívá v nalezení rozkladu součinu prvočísel p_1 a p_2. RSA byl publikován v roce 1978 a nese název podle jeho tvůrců - Ron Rivest, Adi Shamir a Leonard Adleman. V současné době neexistuje žádný účinný algoritmus na řešení uvedených problémů, problém pravděpodobně nastane s příchodem kvantových počítačů. Případné ohrožení manuální faktorizací použité úrovně zabezpečení klíče RSA se řeší pomocí jeho navýšení.

6.6.1. Princip algoritmu RSA

Algoritmus RSA se skládá ze tří částí:

1. Generování klíče:
    - Náhodně si zvolte dvě různá prvočísla p a q, která mají podobnou délku v bitech. Pro ověření prvočíselnosti můžeme využít nějaký test provočíselnosti například nějaký rychlý deteministický test jako je AKS [103], Miller-Rabin [104].
    - Vypočítejte $n = pg$.

- Vypočítejte $\varphi(n) = (p-1)(q-1)$, kde ϕ je Eulerova funkce. Eulerova funkce $\varphi(n)$ udává počet přirozených čísel, které jsou menší nebo rovny n a jsou nesoudělná s n. Jinak řečeno pro $n \in N$, $\varphi(n) = k$, kde $1 \leq k \leq n$ platí $gcd(n,k) = 1$. Funkce $\varphi(mn)$ je multiplikativní tj. $\varphi(mn) = \varphi(m)\varphi(n)$, kde $m, n \in N$ jsou nesoudělná.
- Zvolte si exponent $e \in N$ veřejného klíče, kde $1 < e < \varphi(n)$ a $gcd(e, \varphi(n)) = 1$. Jesliže je bitová délka e malá a má malou Hammingovu váhu (počet nenulových bitů řetězce), je šifrování efektivnější. S ohledem na bezpečnost nevolte e příliš malé.
- Odvoďte si exponent d soukromého klíče ze vztahu $d \equiv e^{-1} \, (mod\, \varphi(n))$, d je multiplikativní inverse k $e \, (mod\, \varphi(n))$. Vztah lze odvodit pomocí rozšířeného Eukleidova algoritmu pro $(de) \, (mod\, \varphi(n)) = 1$ resp. $de = 1 \, (mod\, \varphi(n))$, platí $dx + ey = gcd(d, e)$.

Soukromý klíč je dán exponentem d a operací modulo n. Veřejný klíč je dán jako exponentem e a operací modulo n. Utajené zůstávají prvočísla p, q a $\varphi(n)$.

2. Šifrování: Veřejný klíč je dán dvojicí (e, n). Zprávu M převedeme do celočíselné podoby pomocí reversibilního protokolu, který pracuje s metodou doplňování zpráv OAEP (Optimal Asymmetric Encryption Padding). Pro M platí $0 < M < n$. Samotné šifrování je dáno vztahem $C = M^e \, (mod\, n)$. Pro umocňování můžeme použít metodu opakování čtverců (na okruhu modulo n), která je založena na rozkladu mocnitele na mocniny dvojky.

3. Dešifrování: Soukromý klíč je dán dvojicí (d, n). Otevřený text M lze dešifrovat z C pomocí vztahu: $M = C^d \, (mod\, n)$. Text M je v neupraveném formátu, proto se musí převést pomocí reverzibilního schéma. Dešifrování je poměrně náročná operace, proto se jako vylepšení používá algoritmus zbytků (Chinese remainder algorithm), který si při generování soukromého klíče uchovává pomocné hodnoty.

6.6.2. Bezpečnost RSA

Bezpečnost RSA spočívá v RSA problému, který je založen na existenci součinitelů p a q a na problému nalezení kořenu m v operaci m^e. Pokud by se útočníkovi podařilo p a q odhalit, odvodil by si Eulerovu funkci $\varphi(n) = (p-1)(q-1)$ a mohl by si vypočítat soukromý klíč $d = e^{-1} \, (mod\, \varphi(n))$.
Faktorizace velkých čísel na prvočísla je stále omezená. Nejúspěšnější metodou je GNFS, která však pracuje s exponeciální složitostí. Za efektivní metodu rozkladu čísla na prvočísla považujeme metodu s polynomiální složitostí.
V roce 1999 byla uveřejněna úspěšná faktorizace RSA-150 (klíč 512-bitů). V roce 2010 byla zveřejněna poslední úspěšná faktorizace obecného algoritmu s 768 bity

za pomoci distribuovaného výpočtu. RSA běžně používá klíče o délce 1024 až 2048-bitů. Předpokládá se, že bude možné v nejbližší budoucnosti faktorizovat RSA s 1024-bity, proto se doporučuje u RSA velikost n o 2048-bitech.

Z hlediska bezpečnosti je také důležitá správná volba p a q. Prvočísla by neměly být podobné velikosti, protože hrozí jejich odhalení pomocí Fermatovy faktorizace. Další podmínkou je dostatečný faktor prvočísel p a q, protože totiž hrozí faktorizace n pomocí Pollardova $(p-1)$ algoritmu.
Dalším problémem může být klíč d o malé velikosti. Většinou platí, že je $p \in (q, 2q)$ a pak $d = \frac{n^{-4}}{3}$, ale tím se snižuje složitost výpočtu d z (e, n). Uvedeného nedostatku využívá Wienerův útok.
Jak již bylo řečeno výše, důležitou roli u RSA hraje i klíč e. NIST doporučuje použití $e > 65537$, které zajišťuje ochranu proti potenciálním útokům na malé e i stále efektivní šifrování.

6.6.3. Kryptoanalýza RSA

Mezi potenciální úspěšné útoky se řadí Shorův algoritmus [105], který dokáže faktorizovat čísla v polynomickém čase. Shorův algoritmus je však možno implementovat pouze na kvantovém počítači, pro který doposud neexistuje praktická realizace. Shorův algoritmus zjednodušuje problém faktorizace čísel na hledání periody nějaké periodické funkce $F(a) = y^a \, mod \, n$, kde n je faktorizované číslo. Shor navrhl i teoretické řešení úlohy diskrétních algoritmů a potenciální řešení problému eliptických křivek. S příchodem kvantových počítačů bude nutno řešit budoucnost asymetrických algoritmů.

RSA je náchylné vůči choosen-ciphertext útokům, protože platí $M_1^e M_2^e \equiv (M_1 M_2)^e \, (mod \, n)$, protože je multiplikativní. Pokud chce útočník zjistit výsledek dešifrování $C = M^e \, (mod \, n)$, požádá vlastníka soukromého klíče o dešifrování podvrženého šifrovaného textu $c' = Cr^e \, (mod \, n)$, kde r hodnota volená útočníkem. Více informací v praktické části 11.2.1..

Vzhledem k tomu, že je RSA deterministický, je náchylný na chosen-plaintext útokům. Tento problém řeší RSA metodou doplňování otevřených textů. Pokud není schopen útočník rozlišit mezi výsledky šifrování předem známých otevřených textů, pak se RSA nazývá sématicky bezpečný.

Dalším typem útoků na RSA jsou timing útoky. Jsou založeny na skutečnosti, že při znalosti přesných časů dešifrování lze nasimulovat na stejném hardwarovém vybavení odvození tajného klíče d. V roce 2003 byl uveřejněn úspěšný útok na schéma digitálního podpisu RSA při síťové komunikaci [106].
Pro zamezení timing útoků lze zavést konstantní dobu dešifrování, což ale snižuje efektivnost algoritmu. Lepší metodou je tzv. zaslepení (blinding), která využívá multiplikativní vlastnosti RSA. Metoda odstraňuje přímou závislost času dešifrování s hodnotou na vstupu. Pro každé dešifrování $C^d \, (mod \, n)$ je zvolena tajná

hodnota r, pro kterou z Eulerova teorému platí, že z $(r^eC)^d\,(mod\,n)$ dostaneme $rC^d\,(mod\,n)$ tj. účinek r se odstraní vynásobením jeho inverzní hodnoty.

Mezi další útoky na slabiny používání RSA patří útok publikovaný Johanem Hastadem, který později vylepšil Don Coppersmith (Coppersmith útok [107]). Útok je založen na situaci, kdy se posílá stejný otevřený text v šifrované podobě pomocí klíče e více stranám, které používají stejný klíč e, ale rozdílné p a q resp. n. Útočník si může otevřený text dešifrovat pomocí algoritmu zbytků (Chinese remainder theorem). I když jsou otevřené texty různé, útočník ví, že mezi nimi existuje lineární relace.
Útoku se dá jednoduše vyvarovat jedinečným e pro každého uživatele.

6.6.4. Schéma digitálního podpisu RSA

Algoritmus RSA je nejrozšířenějším schématem digitálních podpisů. Popíšeme si standart PKCS#1.

1. Generování klíčů - generování klíčů je stejné jako u RSA. Od verze 2.1 se používají multiprvočíselné klíče, kde počet klíčů může být dva a více:

   $$n = r_1 \cdot r_2 \cdot \cdots \cdot r_u,\ kde\ u \geq 2.$$

   Veřejný klíč je pak dán jako dvojice (n, e). Tvar soukromého klíče se liší podle počtu klíčů.

2. Generování podpisu - odesílatel vypočte ze svých klíčů $s \equiv z^d\,(mod\,n)$ a odešle dvojici (s, z), kde z je podpis.

3. Ověření podpisu - Příjemce ověří zprávu pomocí veřejného klíče e: $z \equiv s^e\,(mod\,n)$.

Aktuální verze 2.1 je kryptograficky bezpečná. Kryptoanalýze PKCS#1 verze 1.5 je věnována kapitola 11.2.1..

7. Hašovací funkce

Obecně můžeme hašovací funkci popsat jako funkci, která mapuje nějakou větší množinu dat (klíče) na menší množinu dat. Mapování je omezeno přesně danými podmínkami, které musí algoritmus hašovací funkce dodržet. Hašovací funkce (hashe) se používají v mnoha oborech, text se bude především zabývat *kryptografickými hašovacími funkcemi.*

Hašovací funkce mají široké použití. Mezi základní příklady použití hašovací funkcí patří hašovací tabulky, budování cache, vyhledávání duplicitních záznamů, hledání stejných výrazů (podvýrazů), geometrické hašování (rozdělení prostoru do sítě buněk).

Mezi obecné vlastnosti hašovacích funkcí patří:

- *referenční transparentnost*, která udává, že pro dva stejné vstupy (posloupnosti stejných znaků) dává hašovací funkce stejné výsledky.
- *determinismus procedur*, u kterých se v čase nebo charakterem nemění vstupy.
- *rovnoměrnost rozložení (jednotnost)*, generování hodnot se stejnou pravděpodobností, ideální případ tzv. perfektní hašování.
- *dynamičnost* hašovací funkce a jejich proměných. Hašovací funkce by měla být použitelná, i když se změní podmínky (např. počet proměnných).
- *příznivá složitost algoritmu*

Některé hašovací funkce způsobují kolize (např. u hašovací tabulky), protože můžou namapovat klíče na stejnou hašovací hodnotu. V případě hašovacích tabulek se snaží funkce o rovnoměrné rozložení záznamů. Kolize se řeší úpravou hašovací funkce jako je dvojité hašování.

7.1. Kryptografické hašovací funkce

Hašovací funkci lze z pohledu kryptografie definovat jako deterministickou funkci (algoritmus), která mapuje libovolný vstupní blok (zprávu) M, $M \in \{0,1\}^*$, na blok pevné délky (hash, otisk) $h = H(M)$, $h \in \{0,1\}^d$. Z bezpečnostního hlediska se ideální kryptografická hašovací funkce chová jako náhodné orákulum. Hašovacími funkcemi budeme dále myslet kryptografické hašovací funkce.

Vlastnosti ideálních hašovacích funkcí:

1. malá změna zprávy změní i hash, není možné změnit zprávu aniž by se nezměnil hash.
2. odvození hashe je výpočetně jednoduché.

3. z hashe je prakticky nemožné ($\in NP$) získat původní zprávu tzv. jednosměrná funkce.

4. je teoreticky nemožné najít dvě různé zprávy M_1 a M_2, které mají stejný hash tj. $h(M_1) = h(M_2)$.

5. je teoreticky nemožné nalézt pro zadanou zprávu M jinou zprávu $M' \neq M$, která má stejný hash tj. $h(M) = h(M')$.

Hašování lze použít pro důkaz rovnosti, nezachovává uspořádání ani podobnost. Vlastnost 3) je nazývána **podmínkou resistence proti nalezení vzoru** (preimage resistance). Vlastnost 4) je nazývána **podmínkou resistence proti kolizím**. Vlastnost 5) se nazývá **druhá podmínka resistence proti nalezení vzoru** (2nd preimage resistance). Složitost výpočtu, vlastnost 2), by měla být $O(n)$ pro řetězec délky n. S uvedenými podmínkami souvisí útoky na kryptografické hašovací funkce 4.4..

Vlastnost odolnosti vůči kolizím tzv. **bezkoliznost** je důležitá pro digitální podpisy, kde tato vlastnost zaručuje jedinečnost ověření pro autoritu. Dalším z příkladů jsou proof-of-work systémy, které používají vlastnost bezkoliznosti jako důkaz bezpečnosti systému pro uživatele. Dalším použitím vlastnosti bezkoliznosti jsou distribuované systémy, kde se jedinečnost hashe bere jako ověření stejné verze souboru.

Hašovací funkce jsou v kryptografii používány pro ověření autentičnosti zpráv, zaručení integrity dat, v aplikacích informační bezpečnosti, kontrolních součtech atd. Hašovací funkce lze použít k vytvoření dalších kryptografických zástupců, u kterých je ale potřeba dbát na korektní design zaručující kryptografickou bezpečnost. Jedním z příkladů je HMAC, varianta MAC viz. 5.3.5., který je často vytvořen z hašovací funkce. Jedna z konstrukčních metod návrhu hašovacích funkcí je použití blokových šifer. Opačně lze hašovací funkce použít ke konstrukci blokových šifer. Mezi takové zástupce patří např. SHACAL, což je 160-bitová bloková šifra založená na SHA-1. Dalším příkladem použití hašovacích funkcí jsou varianty pseudonáhodných generátorů čísel, které kombinují náhodný seed s hashem protiproudu. Dalším příkladem využití je proudová šifra SEAL, která aplikuje hašovací funkci SHA-1 pro generování interních tabulek pro generátory klíčů.

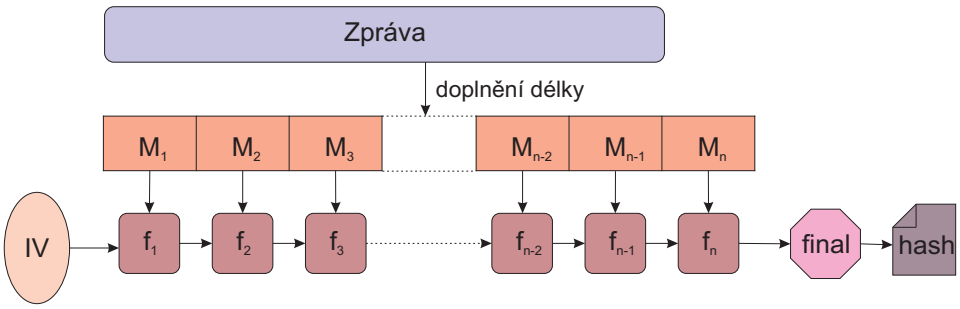
7.2. Konstrukce kryptografických hašovacích funkcí

7.2.1. Merkle-Damgård kostrukce

Merkle-Damgård kostrukce je metoda návrhu resistentních hašovacích funkcí vůči kolizím z jednosměrných hašovacích funkcí. Metoda splňuje požadavek kladený na zabezpečené kryptografické hašovací funkce, protože dokáže přijmout řetezec libovolné délky a převést jej pomocí kompresní funkce na řetězec pevné

délky. Hašovací funkce založená na Merkle-Damgårdově kostrukci je odolná vůči kolizím tak jak je odolná kompresní funkce tj. pokud útočník použije řadu různých postupů (padding scheme, schéma útoku) na kompresní funkci a kompresní funkce je dále odolná vůči kolizím, pak je odolná vůči kolizím i hašovací funkce pro dané schéma útoku. Myšlenka Merkle-Damgård kostrukce byla popsána v roce 1979, autoři jsou Ralph Merkle a a Ivan Damgård.

Metoda je založena principu rozdělení textu na stejně dlouhé bloky, na které je sekvenčně aplikována jednosměrná kompresní funkce. Přesněji se zřetězí iterativní komprese výsledku předchozí operace s dalším blokem. Všechny bloky zprávy jsou tedy součástí výsledné hashe. Hašovací funkce je určena její kompresní funkcí.



Obrázek 34. Merkle-Damgård kostrukce

Algoritmus Merkle-Damgård kostrukce
Každou zprávu M lze rozdělit jako zřetězení jeho n bloků. Každý blok má pevnou délku l (512, 1024 bitů). Poslední blok M_n je doplněn (např. nulami) do potřebné délky, bity reprezentují délku zprávy M, což je další bezpečnostní opatření.

$$M = M_1 \circ M_2 \circ \cdots \circ M_n$$

Jednosměrná kompresní funkce f_C bere mezivýsledek nebo zřetězenou hodnotu s_{i-1} a blok vstupních dat m_i a z nich spočítá další mezivýsledek s_i. Hodnota s_i představuje stav hašovací funkce pro blok M_i. Následující zřetězená hodnota s_i má délku k:

$$f_C : \{0,1\}^k \times \{0,1\}^l \rightarrow \{0,1\}^k$$

Algoritmus začíná s počáteční hodnotou s_0, obvykle jde o inicializační vektor IV

nebo o nějakou počáteční zřetězenou hodnotu.

$$
\begin{aligned}
s_0 &= IV \\
s_1 &= f_C(s_0, M_0) \\
s_2 &= f_C(s_1, M_1) \\
&\vdots \\
h(M) = s_{n+1} &= f_C(s_n, M_n) \\
h(M) &= f_C\Big(\cdots f_C(\underbrace{f_C(s_0, M_0)}_{s_1}, M_1) \cdots \Big) \\
&\qquad\qquad \underbrace{\qquad\qquad\qquad\qquad}_{s_2}
\end{aligned}
$$

Transformace je dokončena finální transformací, která může mít různé účely. Samotné mezivýsledky mohou být (dle typu kompresní funkce) delší než je požadovaná velikost bloku. Finální funkce může být kompresí do požadované velikosti nebo zajišťuje lepší promíchání a *lavinový efekt* (Avalanche effect). Lavinový efekt zajišťuje, že i malá změna na vstupu (jeden bit) vede k výrazné změně výstupu (např. poloviny bitů).

7.2.2. Bezpečnost hašovacích funkcí Merkle-Damgård kostrukce

Během let, co byla Merkle-Damgård kostrukce publikována, byly objeveny nedostatky v návrhu. Některým se dá úspěšně vyhnout, další jsou pouze teoretické, protože se vztahují na extrémně dlouhé zprávy. Více informací [68].

Jak bylo již jednou řečeno, hašovací funkce založená na Merkle-Damgård kostrukci je odolná vůči kolizím tak jak je odolná kompresní funkce. Konstrukce má několik nežádoucích vlastností:

- *pseudo-kolize* - je možné najít dvě zprávy M a M' s mezivýsledky s a s', pro které platí $f_C(s, M) = f_C(s', M')$. Nalezení pseudokolize má stejnou složitost jako nalezení kolize.

- *rozšíření délky* - pokud existují dvě stejně dlouhé zprávy M a M', které kolidují, pak také kolidují $x \circ M$ a $x' \circ M'$. Důsledek je základem rozšiřujících útoků (extension attacks). Metodě se nedá jednoduše vyhnout, záleží na volbě orákula.

- *pevné body (fixed points)* jsou k dispozici pro všechny hašovací funkce, které mají reverzibilní kompresní funkce. Jedná se o hašovací funkce, které se řídí Davies-Meyer konstrukcí (např. MD4, MD5 nebo SHA-1). Pro kompresní funkci a blokovou šifru $E_k(k, X)$ zkombinovanou s mezivýsledkem s_i pomocí operace + (modulárního sčítání, XOR), dostaneme:

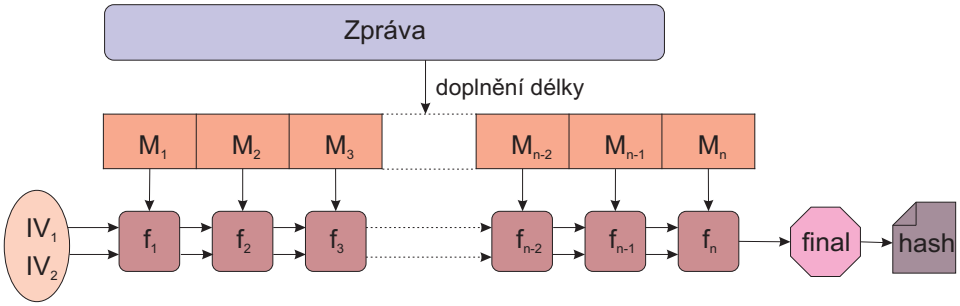
$$f_C(s_i, M_i) = E(M_i, s_i) + s_i$$

Pro danou kompresní funkci lze pevné body p odvodit pro blok zprávy M_p jako $p = E^{-1}(M_p, 0)$, kde p splňuje $p = f_C(p, M_p)$. Pokud je p stav hašovací funkce, pak lze získat hash bloku M_p bez změny stavu. Pokud by byl jako inicializační vektor zvolen pevný bod p nějakého tajného bloku M_p, pak by útočník mohl dosáhnout kolize (nebo 2nd preimages) a to $h(M) = h(M_p \circ M) = h(M_p \circ M_p \circ M_p)$. Útočník tak může kontrolovat blok zprávy, ale i vytvářet předlohu (preimage) ze zřetězené hodnoty pro p. Útok se dá ještě zjednodušit pomocí narozeninového útoku (birthday attack), který umožní odvodit $2^{n/2}$ pevných bodů (p_i, M_{p_i}) a $2^{n/2}$ zpráv M_j ke generování rozšířené zprávy pro n-bitovou hašovací funkci: $h(M_j) = h(M_j \circ M_{p_i})$. Kolidující zprávy jsou pak členy rozšíření zprávy.

- *multikolize* - pokud existují kolize pro nějaké dvě dvojice kolidujících zpráv, pak lze nalézt další čtyři zprávy se stejným hashem, které kolidují. Schéma lze rozšířit k nalezení libovolného počtu práv se stejným hashem. Tento poznatek lze použít v důkazu nízké bezpečnosti při řetězení dvou (více) hašovacích funkcí. Bezpečnost zřetězených hašovacích funkcí není vyšší než je bezpečnost silnější funkce.

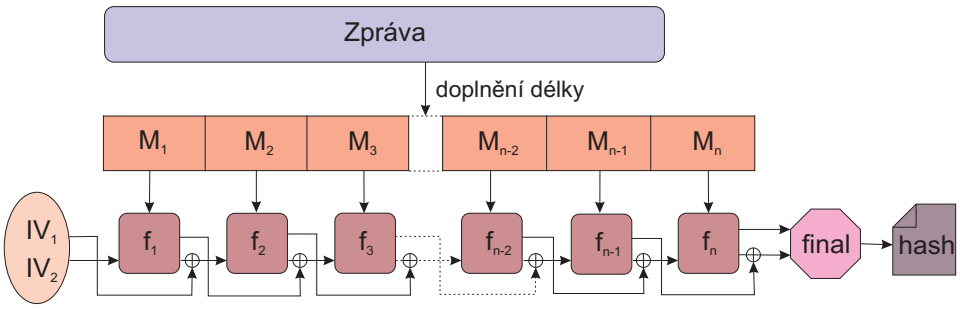
7.2.3. Varianty Merkle-Damgård kostrukce

Vzhledem k bezpečnostním nedostatkům, které má původní Merkle-Damgård kostrukce, byly navrženy varianty, které nejsou náchylné na rozšíření délky a multikolize.



Obrázek 35. Wide-pipe konstrukce. Zdvojené zřetězené mezivýsledky.

Wide-pipe konstrukce
Metoda Wide-pipe (metoda širokého potrubí) má bezpečnější vnitřní stav, který jako by zdojuje zřetězené mezivýsledky. Počet bitů mezivýsledku je větší než počet bitů výstupu. Pokud je potřeba odvodit hash pro n bitů, pak bere kompresní funkce $2n$ bitů zřetězeného mezivýsledku a l bitů zprávy a výsledkem je pak $2n$ bitů. Finální kompresní funkce převede $2n$ bitů na požadovaných n bitů.



Obrázek 36. Fast Wide-pipe konstrukce. Polovina zřetězených mezivýsledků je použita v kompresní funkci.

Fast Wide-pipe konstrukce
Fast Wide-pipe konstrukce (rychlá metoda širokého potrubí) je zefektivněná varianta Wide-pipe konstrukce, která rozděluje mezivýsledek na dvě části. Jednu z nich použije jako vstup pro kompresní funkci a druhou zkombinuje pomocí operace XOR s výsledkem kompresní funkce. Výsledek po kombinaci je dalším mezivýsledkem zřezězení. Metoda je až dvakrát rychlejší než je původní Wide-pipe konstrukce. Pokud se použije stejná kompresní funkce jako je u Wide-pipe konstrukce, je potřeba upravit bloky zprávy M, ale i tak je rychlost metody vyšší.

7.2.4. Konstrukce hašovacích funkcí z blokových šifer

Jednoduché kryptografické hašovací funkce lze sestrojit z mnoha typů blokových šifer. Konstrukce setrojí jednosměrnou kompresní funkci, která je založena na šifrování blokových šifer. Metody kostrukce se podobají operačním módům blokových šifer. Základní podmínkou designu je, aby nebyla funkce bijektivní pomocí zpětné vazby. Mezi významné hašovací funkce založené na této metodě patří MD4, MD5, SHA-1, SHA-2.

Schéma Davies-Meyer
Schéma Davies-Meyer je metoda konstrukce jednosměrné kompresní funkce. Předpokládejme, že existuje blok M_i jako klíč a zřetězený mezivýsledek s_{i-1} jako otevřený text, výsledkem šifrovací rutiny je následující mezivýsledek s_i. Konstrukce není bezpečná, protože je reverzibilní. Pro zaručení požadované podmínky bezpečnosti se aplikuje operace XOR na výstup šifrování a mezivýsledek s_{i-1}. Místo operace XOR je možné použít modulární sčítání. V první rundě, kdy neexistuje předcházející zřetězený mezivýsledek, se použije konstantní inicializační hodnota s_0.

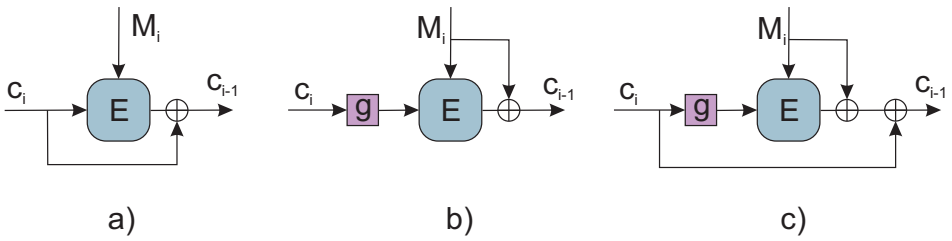
$$s_i = E_{M_i}(s_{i-1}) \oplus s_{i-1}$$

Schéma má poměr rozdělení dán vztahem:

$$R_{DM} = \frac{k}{1 \cdot n} = \frac{k}{n},$$

kde k je velikost klíče. Pokud bloková šifra používá např. 256-bitové klíče, pak je má blok M_i 256 bitů.

I když je bloková šifra zcela bezpečná, existují pevné body, které jsou potenciálním nedostakem schéma Davies-Meyer. Pro každé M je možné najít hodnotu h takové, že $E_M(h) \oplus h = h$, pak stačí stanovit $h = E_M^{-1}(0)$. Doposud nebyl popsán praktický útok, který by využil pevných bodů pro úspěšný útok. Pro prevenci útoku pomocí pevných bodů používá schéma Davies-Meyer metodu prodloužení, kdy se ke zprávě na konec připojí její délka. Pak nelze najít kolizi s větší úspěšností než udává narozeninový paradox $2^{n/2}$, kde n je délka zprávy v bitech. Dalším použitelným útokem, který využívá existence pevných bodů, je preimage útok.



Obrázek 37. a) Davies-Meyer jednosměrná kompresní funkce. b) Matyas-Meyer-Oseas jednosměrná kompresní funkce. c) Miyaguchi-Preneel jednosměrná kompresní funkce.

Schéma Matyas-Meyer-Oseas

Schéma Matyas-Meyer-Oseas dostaneme tak, že se u Davies-Meyer schéma prohodí hodnoty M_i a s_i:

$$s_i = E_{s_{i-1}}(M_i) \oplus s_{i-1}$$

V první rundě, kdy neexistuje předcházející zřetězený mezivýsledek, se použije konstantní inicializační hodnota s_0. Pokud má bloková šifra jinou velikost bloku než je zřetězená hodnota, pak bude mít hodnota špatnou velikost při použití pro klíč. Šifra může mít i jiné požadavky na klíč, proto je potřeba převést hodnotu pomocí funkce g, která hodnotu upraví podle potřeby (zkrátí, zkonvertuje). Schéma má poměr rozdělení dán vztahem:

$$R_{MMO} = \frac{n}{1 \cdot n} = 1.$$

Na schéma Matyas-Meyer-Oseas lze použít 2nd-preimage útok, který může být proveden pro 2^k bloků zprávy v čase $k \cdot 2^{n/2+1} + 2^{n-k+1}$.

Schéma Miyaguchi-Preneel
Je rozšířením Matyas-Meyer-Oseas metody, kdy se jako poslední krok kompresní funkce přidá blok zprávy M_i. Jako otevřený text se použije blok M_i s klíčem s_{i-1}, na výsledek šifrování a zřetězený mezivýsledek s_{i-1} se aplikuje operace XOR a následovně je opakovaně aplikována operace XOR na doposud vypočítanou hodnotu hashe a blok zprávy M_i:

$$s_{i+1} = E_{s_{i-1}}(M_i) \oplus s_{i-1} \oplus M_i.$$

Stejně tak jako v obou předchozích metodách neexistuje žádná počáteční hodnota s_0, proto je potřeba zadat inicializační hodnotu. Metoda se hodí v případě, kdy je blok zprávy a zřetězený mezivýsledek stejně dlouhý. Lze ji použít i když se délka v bitech liší, ale je potřeba blok upravit do požadované podoby. Na schéma Miyaguchi-Preneel lze použít 2nd-preimage útok, který může být proveden pro 2^k bloků zprávy v čase $k \cdot 2^{n/2+1} + 2^{n-k+1}$.

7.3. Významné kryptografické hašovací funkce

7.3.1. Historický přehled

První hašovací funkce byly navrženy s ohledem na prokazatelnou bezpečnost. Autoři často využívali obtížně řešitelné problémy, které zařadili do svého návrhu. Ivan B. Damgård navrhl kryptografickou hašovací funkci, která byla založena na NP-úplném problému Úloze batohu. Později autoři opustili podmínku prokazatelné bezpečnosti a zaměřili se na efektivitu návrhu jednoduché kompresní funkce. Design je zaměřen na opakování rund a bezpečnosti konstrukcí.

První široce používaná hašovací funkce byla **MAA** (Message Authenticator Algorithm) [69], která byla navržena v roce 1983 pro finační sektor jako zabezpečení komunikace mezi finačními ústavy. Součástí algoritmu je kryptografická hašovací funkce s 64-bitovým klíčem. Algoritmus dokázal zpracovat i zprávy s délkou větší než 1024-bitů a vracel 32-bitový klíč. Hašovací funkce byla poměrně rychlá. Kryptoanalýza MAA byla zveřejněna v roce 1997, ukazuje možnost padělání zprávy a odhalení klíče (pro 2^{32} zpráv s jedním blokem) i existenci 2^{33} slabých klíčů, které umožňují snadný výpočet kolizí. V období 1990-1992 vzniklo mnoho hašovacích funkcí, které byly krátce po uveřejnění prolomeny. I když nebyla hašovací funkce prolomena, i úspěšný útok na oslabenou verzi ohrozil důvěru odborníků a to vedlo k jejímu opuštění. Například v roce 2004 byly nalezeny nedostatky u několika populárních hašovacích funkcí, včetně SHA-0, RIPEMD a MD5. To zpochybnilo dlouhodobou bezpečnost u jejich následovníků, zejména SHA-1 (posílená SHA-0), RIPEMD-128, RIPEMD-160 (posílené RIPEMD). I přes nalezené nedostatky jsou následovníci nadále používáni. V roce 2005 byl popsán úspěšný útok na SHA-1, který je schopný najít kolize pro 2^{69} hašovacích operací, další publikovaný útok vyžadoval 2^{63} operací. Prokázaná slabost návrhu

vede k závěru, že SHA-1 bude v budoucích letech prolomena, proto se přechází k následovníkům z rodiny SHA a to SHA-2, nebo jiným hašovacím technikám, které jsou odolné vůči kolizím např. techniky s náhodným hašováním. Řešení v oblasti hašovacích funkcí by měla přinést SHA-3, která je plánována na rok 2012.

7.3.2. MD2

MD2 je první z řady kryptografických hašovacích funkcí navržené Rolandem L. Rivestem, MD kandidovala na Message-Digest. MD1 ani MD3 nebyly nikdy publikovány, MD5 je rozšířením MD4. Všechny tři funkce vrací 128-bitový hash.

MD2 byla publikována v roce 1988 pro bytestreamové hashe. Velikost každého bloku je 16 bajtů, text je tedy rozdělen do bloků 16-bajtové velikosti a následovně je připojeno 16 bajtů kontrolního součtu. V centru MD2 je 256 bajtový S-box, který substituuje bajty na bajty, hodnoty S-boxu jsou dány jako zlomky čísla π. Jako vnitřní stav se používá pomocné pole X_k o velikosti 48 bajtů. Algoritmus probíhá v 18-ti krokové smyčce, ve které dochází k permutaci každého bajtu pomocného pole s každým bajtem 16-bajtového vstupu. Jakmile jsou všechny vstupní bloky zpracovány, první dílčí blok pomocného pole tvoří hodnotu hashe.

Nyní si popíšeme algoritmus detailně: pro pohyb v S-boxu se používá pomocný index t modulo 256. Všechny hodnoty jsou inicializovány na 0. 16-bajtový vstupní blok se načte do pole X_k: pro $k \in \{16, \ldots, 31\}$ normálně, pro $k \in \{32, \ldots, 48\}$ jako výsledek operace XOR na vstupní hodnoty a na $X_0, \ldots, X_{15}$. MD2 iteruje následující funkci přes všechny 48-bitové slova v 18-ti rundách:

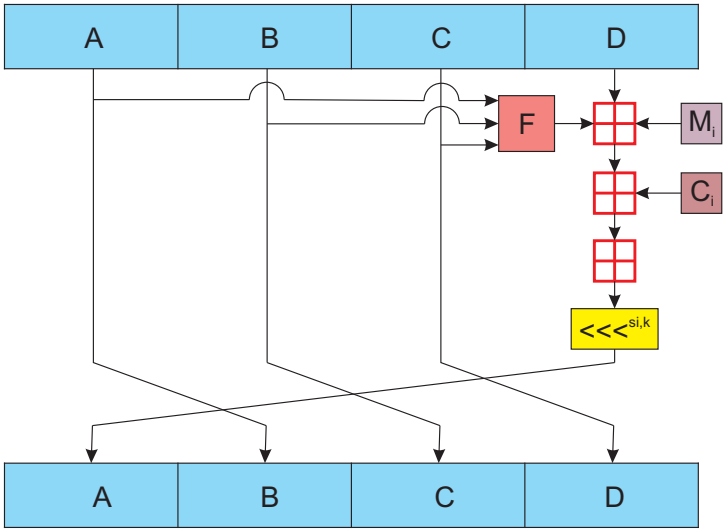
$$\text{Set } t \text{ and } X_k \text{ to } (X_k \oplus S[t])$$

Jakmile je zpracován poslední blok, algoritmus MD2 vrací jako výsledek hash tvořený hodnotami $X_0 \ldots X_1 5$.

Kryptoanalýza MD2
Kryptoanalýza ukázala závažné nedostaky v návrhu MD2. V roce 1995 byl publikován útok pomocí speciální kompresní funkce, která dokáže nalézt mnoho kolizí, ale na výslednou složitost má velký vliv vnitřní hodnota hashe. Složitost je dána jako $256^{17-z} = 2^{8\cdot(17-z)}$, kde z počet koncových nulových bajtů mezivýsledných hašů, pro hash prvního bloku je $z = 16$. Důvod proč nebyly zveřejněny kolize pro normální MD2 je kontrolní součet. Byly nalezeny pouze pseudokolize a to pomocí upraveného preimage útoku se složitostí 2^{16}. Je nutné si uvědomit, že se kontrolní součty obou zpráv rovnaly pouze v případě, kdy byly zprávy nastaveny na 0 a navíc byly vypočítány rozdílné vnitřní hashe h a h', což umožnilo rovnost pseudokolizí $H(h, M)$ a $H(h', M)$ pro $M = 0^{128}$.
Mezi úspěšné útoky patří preimage útok (rok 2004 s časovou složitostí 2^{104}), (rok 2008 [70] s časovou složitostí 2^{73} pro kompresní funkci a pamětovou složitostí 2^{52} bloků zprávy), kolizní útok (2009) [71]. V roce 2009 byly vydány bezpečnostní aktualizace MD2 pro OpenSSL, GnuTLS, a NSS [72].

7.3.3. MD4

MD4 vznikla v roce 1990 jako posílená verze Merkle-Damgård kostrukce pro bytestreamy. MD4 byla navržena pro 32-bitové procesory a byla velmi rychlá. Princip doplnění zprávy (padding, zarovnání zprávy) MD4 se stal součástí jiných hašovacích funkcí např. SHA-1, MD5. Velikost otisku (digest) je 128 bitů.



Obrázek 38. MD4 se skládá z podobných 48 operací (3 rundy s 16 operacemi). F je nelineární funce. M_i označuje 32-bitový vstupní blok zprávy, C_i označuje 32-bitovou konstantu, která je různá pro každou operaci. $\boxplus$ značí sčítání modulo 2^{32}.

Algoritmus MD4

1. *Doplnění zprávy.*
   Ke zprávě je doplněn jeden 1 bit a pak tolik 0 bitů kolik je potřeba pro délku zprávy shodnou s 448 modulo 512. Dále je doplněna délka zprávy v podobě dvou 32-bitových slov, z nichž druhé je více signifikantní. Pokud je délka zprávy větší než 2^{64} bitů, pak je použito pouze 64 nejméně významných bitů (LSB). To představuje bezpečnostní riziko, ale zprávy delší než 16 exabajtů (2^{60}) jsou spíše teoretické.

2. *Rozdělení textu do bloků.* Zpráva je následovně rozdělena na 512-bitové bloky (MD4 bloky).

3. *Inicializace bufferu.*
   MD4 používá osm 32-bitových slov pro udržení vnitřního stavu, z nichž čtyři (A, B, C, D) jsou uplatněny v každé rundě a další čtyři jsou pro zřetězené

proměnné. Jakmile je zpracován aktuální blok zprávy pomocí kompresní funkce, jsou hodnoty aktualizovány: součet zřetězené proměnné a korenspodující rundovní proměnné je uložen do obou proměnných (modulo 2^{32}). Nyní jsou hodnoty rundovních proměnných inicializovány (inicializační vektory):

A	=	0x67452301	01 23 45 67
B	=	0xefcdab89	89 ab cd ef
C	=	0x98badcfe	fe dc ba 98
D	=	0x10325476	76 54 32 10
		$(hexadecimal)$	$(little-endian\ notace)$

4. *Zpracování zprávy v 16-slovních blocích*
   Kompresní funkce MD4 používá následující funkce:

$$\begin{aligned} F(x,y,z) &= (x \wedge y) \vee (\bar{x} \wedge z) \\ G(x,y,z) &= (x \wedge y) \vee (x \wedge z) \vee (y \wedge z) \\ H(x,y,z) &= (x \oplus y \oplus z) \end{aligned}$$

   Pro každý blok je každá z funkcí provedena v 16-ti průchodech se třemi ze čtyř proměnných jako s argumenty v různém pořadí. MD4 má tři rundy, každá se skládá z volání funkce F, G nebo H. Součet výsledků (vstupní slovo a konstanta) je přidán do čtvrté proměnné. Poté je proměnná kruhově posunutá (circular-shifted) o lichou hodnotu.
   Operace každého kroku MD4 lze sumarizovat zápisem:

$$w = (w + \phi_i(x,y,z) + M_{\pi_i(k)} + C_i)^{\lll s_{i,k}},$$

   kde w, x, y a z jsou každá z proměnných A, B, C a D, i je aktuální runda ($i \in \{1,2,3\}$), k je číslo aktuálního průchodu ($k \in \{0,\ldots,15\}$) každé rundy, ϕ_i označuje jednu z funkcí F, G nebo H. $M_{\pi_i(k)}$ je slovo $\pi_i(k)$ vstupního bloku (π_i je jednoduchá permutace) a C_i je aditivní konstanta rundy i (buď 0x0, 0x5a827999 ($= [2^{30}\cdot\sqrt{2}]$) nebo 0x6ed9eba1 ($= [2^{30}\cdot\sqrt{3}]$)). $\alpha^{\lll s_{i,k}}$ značí kruhové posunutí α o $s_{i,k}$ bitů ($s_{i,k} \in \{3,5,6,9,11,13,15,19\}$), každá runda má čtyři různé hodnoty posunutí.

5. *Výstup.*
   Jakmile je zpracován poslední blok zprávy, algoritmus MD4 vrací otisk (message digest) tvořený $ABCD$ v little-endian notaci.

Rozšířená MD4 (Extended-MD4) používá dvě paralelní instance MD4. Rozdíl je v hodnotách inicializačních vektorů a aditivních konstant. Jedna instance MD4 používá inicializační vektory a aditivní konstanty popsané výše, druhá instance má vektory dány hodnotami 0x33221100, 0x77665544, 0xbbaa9988, 0xffeeddcc a adiditní konstanty 0x0, 0x50a28be6 a 0x5c4dd124. Na konci kompresní

funkce jsou hodnoty proměnných A jednotlivých MD4 instancí vzájemně prohozeny. Rozšířená MD4 vrací 256-bitový hash, který je tvořen zřetězením výsledků obou instancí.

Kryptoanalýza MD4
První kryptoanalytický útok na MD4 byl publikován v roce 1991. Bylo prokázáno, že pokud se vynechá první runda resp. poslední runda, pak je nalezení kolizí mnohem jednodušší. Poznatky o případné slabosti návrhu vedly k vytvoření MD5. V roce 1996 byl zveřejněn první úspěšný kolizní útok na plnou verzi MD4 a na upravenou rozšířenou MD4, protože obě verze používají stejný inicializační vektor. V roce 2005 byl publikován sofistikovanější útok, který umožnil snadné nalezení kolizí s pravděpodobností blížící se jedné. Později bylo dokázáno, že pro splnění podmínky preimage útoku na první a druhou rundu MD4 vyžaduje pouze jednu hodinu. Pro splnění podmínky pro druhý preimage útok stačí dokonce pár minut. MD4 je jedna z nejméně bezpečných z dopusud používaných hašovacích funkcí. Vzhledem k tomu, že existuje mnoho dalších hašovacích funkcí, které jsou založeny na designu MD4, lze pochybovat i o jejich bezpečnosti. MD4 je i přes uvedené nedostatky hojně rozšířená a je stále v zájmu kryptoanalytiků. Více ke kryptoanalýze MD4 [73].

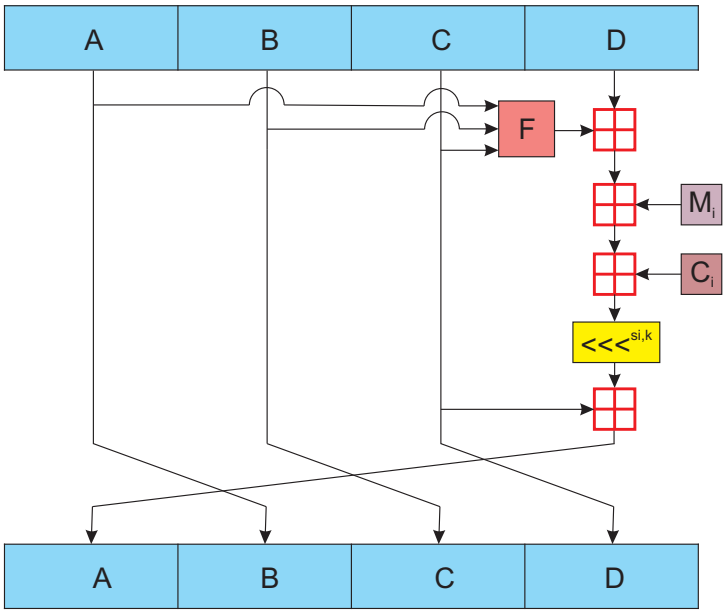
7.3.4. MD5

MD5 je široce rozšířená kryptografická hašovací funkce, která byla publikována v roce 1991 (Ron Rivest). MD5 zpracovává zprávy libovolné délky a vrací 128-bitový otisk (hash), který je obvykle vyjádřený jako 32 místné hexadecimální číslo. MD5 nahradila dřívější MD4, u které byla opomenuta otázka bezpečnosti na úkor rychlosti. MD5 lze najít v mnoha bezpečnostních aplikacích např. jako nativní nástroj v mnoha verzích Linuxu nebo jako nástroj pro kontrolu integrity dat. Je založena na Merkle-Damgård konstrukci.

MD5 vznikla jako nástupce MD4, který měl odstranit její bezpečnostní nedostaky. Přesto bylo dokázáno, že není MD5 odolná vůči kolizím, proto je nevhodné aplikovat MD5 v aplikacích jako jsou SSL certifikáty nebo digitální podpisy. MD5 je považována za kryptograficky prolomitelnou a nevhodnou pro další použití. Jako vhodný zástupce je doporučována SHA-2.

Algoritmus MD5
V mnoha ohledech se MD5 podobá MD4. Používá stejný postup (padding, připojení délky zprávy) pro zpracování zprávy před rozdělením do bloků. Velikost bloků je stejná jako u MD4, MD5 má i stejné inicializační vektory pro proměnné A, B, C a D. MD5 má čtyři rundy a navíc jednu rundovní funkci. Zatímco funkce F a H zůstaly nezměněny, G je vyměněná a I byla nově přidaná. Každá runda má 16 průchodů, ale operace každého kroku byly značně změněny.

1. *Doplnění zprávy.*



Obrázek 39. MD5 se skládá z podobných 64 operací (4 rundy s 16 operacemi). F je nelineární funce. M_i označuje 32-bitový vstupní blok zprávy, C_i označuje 32-bitovou konstantu, která je různá pro každou operaci. $\boxplus$ značí sčítání modulo 2^{32}.

Vzhledem k tomu, že je zpráva rozdělena do 512 bloků, je jí potřeba doplnit tak, aby byla délka dělitelná 512. Doplnění textu je stejné jako u MD4.

2. *Rozdělení textu do bloků.* Zpráva je následovně rozdělena na 512-bitové bloky (šestnáct 32-bitových čísel pro little-endian).

3. *Inicializace bufferu.*
   Inicializace je podobná jako u MD4. Algoritmus MD5 pracuje s 128-bitovým stavem, který je rozdělen do čtyř 32-bitových slov (A, B, C, D), inicializační vektory jsou nastaveny na stejné hodnoty jako u MD4. Rozdíl je v samotných aditivních konstantách, pro které se používá tabulka $T[i]$ $(i \in 1, \ldots, 64)$. Tabulka je vygenerována pomocí sinus funkce: $T[i] = \lfloor |\sin i| \cdot 2^{32} \rfloor$. Posunutí je u MD5 realizováno pomocí šestnácti hodnot z nichž osm je lichých. Stejně jako u MD4 má každý čtvrtý průchod jedné rundy stejnou konstantu posunutí, ale stejné hodnoty nejsou použity v jiné rundě. V každém kroku je k operaci přidán předchozí výsledek.

4. *Zpracování zprávy.*

Kompresní funkce MD5 používá následující funkce:

$$\begin{aligned} F(x,y,z) &= (x \wedge y) \vee (\bar{x} \wedge z) \\ G(x,y,z) &= (x \wedge z) \vee (y \wedge \bar{z}) \\ H(x,y,z) &= (x \oplus y \oplus z) \\ I(x,y,z) &= y \oplus (x \vee \bar{z}) \end{aligned}$$

Funkce F a G jsou stejné, spolu s funkcí H se používají i u MD4. Funkce G MD4 nebyla u MD5 nikdy použita, protože vede k nežádoucí symetrii a snížení výkonu. Podobné je to i u jiných hašovacích funkcí, které jsou na principu MD4 založeny.
Operace každého kroku MD5 lze sumarizovat zápisem:

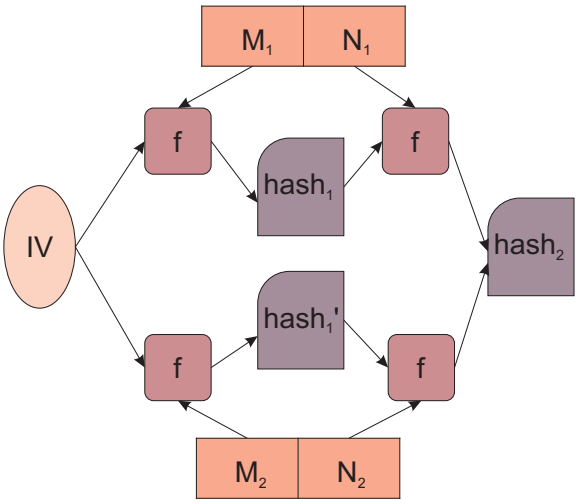
$$w = v + (w + \phi_i(x,y,z) + M_{\pi_i(k)} + T[i \cdot 16 + k])^{\lll s_{i,k}},$$

kde w, x, y a z jsou každá z proměnných A, B, C a D, v je výsledek předchozího kroku, i je aktuální runda ($i \in \{1,2,3\}$), k je číslo aktuálního průchodu ($k \in \{0,\ldots,15\}$) každé rundy. ϕ_i označuje jednu z funkcí F, G, H nebo I. $M_{\pi_i(k)}$ je slovo $\pi_i(k)$ vstupního bloku, $T[k]$ je aditivní konstanta získaná s tabulky T. $\alpha^{\lll s_{i,k}}$ značí kruhové posunutí α o bitů, sčítání je provedeno modulo 2^{32}.

5. *Výstup.*
   Jakmile je zpracován poslední blok zprávy, algoritmus MD5 vrací otisk (message digest) tvořený $ABCD$ v little-endian notaci.

Kryptoanalýza MD5.
První kryptoanalytické útoky se zaměřily na hledání pseudokolizí (1993) a na kolizi v kompresní funkci MD5 (1996), která by umožnila potenciální útok. Vzhledem k tomu, že má MD5 pouze 128-bitový otisk, uvažovalo se o použití narozeninového útoku (projekt MD5CRK).
Teprve až v roce 2004 byl uveden postup pro potenciální kolizní útok na plnou verzi MD5 (i na MD4, RIPEMD a HAVAL-128). Autorem byla Xiaoyun Wang. Metoda byla později upravena a je nazývána Čínským útokem [65]. Čínský útok je založen na dvou základních myšlenkách: diferenční schéma (úloha diferenčních konstant) a metoda modifikace zpráv. V případě diferenčního schéma se kolidující zprávy liší v šesti 32-bitových slovech o definovaný aritmetický rozdíl. Útok hledá kolize ve dvou 1024-bitových zprávách. Nejprve se sestrojí dvě různé 512-bitové bloky M_1 a M_2 a potom se k nim najdou dvě různé 512-bitové bloky N_1 a N_2 tak, že mají složené zprávy (M_1, N_1) a (M_2, N_2) stejný hash. Útočník hledá konstantu C (512-bitů), která udává jak se mají lišit první poloviny textu M_1 a M_2 a druhé poloviny textu N_1 a N_2. Nejprve se musí pomocí C nalézt M_1, tak aby $M_2 = M_1 + C$, při hašování pak vznikají dva kontexty H_1 a H_1'. Odlišnost je srovnána při hašování bloků N_1 a N_2 ($N_2 = N_1 - C$), kontexty H_2 jsou stejné.

Protože jsou zprávy 1024-bitové, jsou doplněny dalším blokem (padding), který je pro obě zprávy stejný. Výsledná hash H_3 je stejná. Princip Čínského útoku je znázorněn na obrázku 40..



Obrázek 40. Princip Čínského útoku.

Jako ukázku kolize je uvedeno ověření hashe MD5 u dvou řetězců [63]. První řetězec je dán:

d131dd02c5e6eec4 693d9a0698aff95c2fc ab58712467eab400458
3eb8fb7f8955ad34060 9f4b30283e488832571 415a085125e8f7cdc99
fd91dbdf280373c5b96 0b1dd1dc417b9ce4d89 7f45a6555d535739ac7
f0ebfd0c3029f166d10 9b18f75277f7930d55c eb22e8adba79cc155ce
d74cbdd5fc5d36db19b 0ad835cca7e3

Druhý řetezec:

d131dd02c5e6eec4 693d9a0698aff95c2fc ab50712467eab400458
3eb8fb7f8955ad34060 9f4b30283e4888325f1 415a085125e8f7cdc99
fd91dbd7280373c5b96 0b1dd1dc417b9ce4d89 7f45a6555d535739a47
f0ebfd0c3029f166d10 9b18f75277f7930d55c eb22e8adba794c155ce
d74cbdd5fc5d36db19b 0a5835cca7e3

Společná hash: a4c0d35c95a63a805915367dcfe6b751.
V roce 2006 ukázal Vlastimil Klíma algoritmus (pro jakýkoliv inicializační vektor), který dokáže najít kolize během jedné minuty na obyčejném notebooku, metoda se nazývá tunelování (tunneling) [64]. V roce 2010 byl popsán kolizní útok na jednoblokovou verzi MD5, pro dvě 64-bajtové zprávy byl nalezen stejný MD5 otisk. Dříve uvedené útoky se zaměřovaly na multi-blokové zprávy. Přesný

postup útoku nebyl z bezpečnostní důvodů publikován. V roce 2011 byla schválena bezpečnostní aktualizace MD5 a HMAC-MD5 [66].
Pro řešení problému kolizí u MD5 vznikl projekt hashclash [67], který hledá tzv. cílené kolize. Skládá se ze zpráv $m_1 \circ b_1$ a $m_2 \circ b_2$, kde $m_1 \neq m_2$ a $b_1 \neq b_2$, pro které mají stejné hashe: $H(m_1 \circ b_1) = H(m_2 \circ b_2)$. Termín cílené kolize znamená, že pro zvolenou dvojici různých zpráv m_1 a m_2 můžeme sestavit dvojici různých zpráv b_1 a b_2, pro které platí, že $H(m_1 \circ b_1) = H(m_2 \circ b_2)$. Obě zprávy obsahují důležitá data. Díky projektu (2008) byly nalezeny dva certifikáty X.509 s různými veřejnými klíči a se stejným MD5 otiskem. X.509 je důležitý certifikát pro certifikaci s veřejnými klíči. Certifikáty jsou používány jako osvědčení pravosti hostitele při navazování bezpečné (šifrované) komunikace např. pomocí TLS (Transport Layer Security). Útočník se mohl vydávat za SSL zabezpečenou stránku jako man-in-the-middle (útočník naslouchající mezi účastníky) a rozvracel oveřené certifikáty ve webových prohlížečích. Navíc bylo možné falešný certifikát podstrčit ověřovací autoritě a získat tak neomezenou expiraci.

7.3.5. RIPEMD

RIPEMD (RACE Integrity Primitives Evaluation Message Digest) vznikla v roce 1992 jako posílená verze MD4. RIPEMD obsahuje dvě upravené instance MD4, které běží paralelně. Nejedná se však o Rozšířenou MD4, protože se algoritmus liší v několika detailech.

Algoritmus RIPEMD.
Jedním z rozdílů jsou aditivní konstanty, první instance má aditivní konstantu stejnou jako je u MD4, druhá instance má aditivní konstanty `0x50a28be6`, `0` a `0x5c4dd124`. Další odlišnost je v kruhovém posunutí, každá instance má 16 různých posunutí. Inicializační vektory jsou stejné jako u MD4. Jakmile proběhne kompresní funkce, jsou výsledné proměnné A^l a A^r, B^l a B^r, C^l a C^r, D^l a D^r pro instance l a r sečteny s zřetězenými proměnnými. Operace každého kroku RIPEMD lze zapsat jako

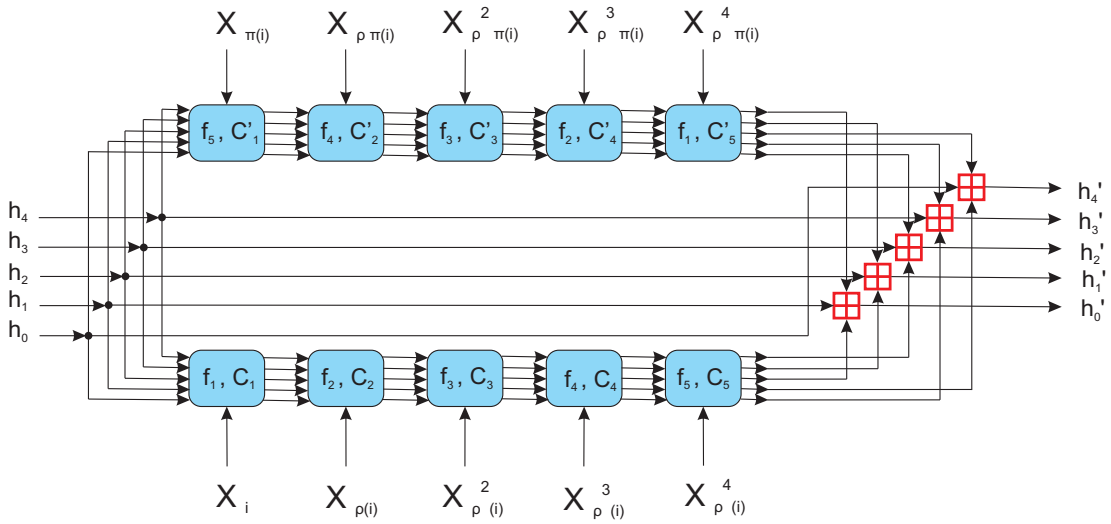
$$w^l = (w^l + \phi_i(x^l, y^l, z^l) + M_{\pi_i(k)} + C_i^l)^{\lll s_{i,k}}$$
$$w^r = (w^r + \phi_i(x^r, y^r, z^r) + M_{\pi_i(k)} + C_i^r)^{\lll s_{i,k}},$$

kde l a r jsou instance. Slovo w je dáno jako součet předchozího slova s výstupem funkce ϕ_i (jedna z MD4 funkcí F, G a H) proměnných x, y a z, se slovem $M_{\pi_i(k)}$ vstupu a konstanty C. Součet je pak kruhově posunut podle poslední operace. Výsledný otisk je dán jako hodnota zřetězených proměnných v little-endian notaci.

Kryptoanalýza RIPEMD.
Poznatky kryptoanalýzy u MD4 měly důsledek na RIPEMD. Nejprve byl uveřejněn útok na RIPEMD se sníženým počtem rund. Teprve až Čínský útok (2005) ukázal náchylnost RIPEMD vůči kolizím [74].

7.3.6. RIPEMD-160

RIPEMD-160 byla vydána v roce 1996 spolu s RIPEMD-128. Jedná se o bezpečnější náhradu původní RIPEMD, proto bývá nazývána jako RIPEMD-1. Označení obsahuje čtyři hašovací funkce: s 128-bitové, 160-bitové, 256-bitové a 320-bitové verze. RIPEMD-256 a RIPEMD-320 byly uvedeny později, jedná se o mírně upravené verze 128-bitové a 160-bitové verze, ale vlastně nepřidávají další bezpečnostní opatření proti kryptoanalýze. Obě pozdější verze snižují pravděpodobnost vůči kolizím díky hodnotám hashe. RIPEMD-128 byla považována v době návrhu za potenciálně slabou (z hlediska narozeninového paradoxu), proto byla upřednostněna RIPEMD-160.



Obrázek 41. Princip RIPEMD-160. Vstupem jsou 16-slovní bloky zprávy X_i a 5-slovní zřetězené proměnné h_0, h_1, h_3, h_4, výstupem jsou nové hodnoty zřetězených proměnných.

Algoritmus RIPEMD-160.
Bezpečnostní vylepšení RIPEMD-160 vycházejí z použití dvou paraleních linií hašování. Algoritmus pracuje s pěti 32-bitovými proměnnými pro každou linii. Rundovní funkce bere jako vstup 5-slovní zřetězenou proměnou a 16-slovní blok zprávy a mapuje je do nové zřetězené proměnné. Kompresní funkce má pět rund a pět boolovských proměnných. RIPEMD-160 používá stejné doplňování zpráv jako MD4, stejně jako inicializační hodnoty MD4, pátá inicializační hodnota začíná 0xc3d2e1f0 (f0 e1 d2 c3). Po doplnění je zpráva rozdělena na t 16-slovních bloků $X_i[j]$ ($0 \neq i \leq t-1$ a $0 \neq j \leq 15$). Obě linie mají rozdílné kruhové posunutí ($s^l_{i,k}$ a $s^r_{i,k}$) a rozdílný výběr permutací pro slova zprávy (π^l_i a π^r_i). Všechna specifická kriteria byly voleny s ohledem na maximální bezpečnost návrhu. Boolovské funkce

jsou dány jako

$$\begin{aligned} f_1(x,y,z) &= x \oplus y \oplus z \\ f_2(x,y,z) &= (x \wedge y) \vee (\bar{x} \wedge z) \\ f_3(x,y,z) &= (x \vee \bar{y}) \oplus z \\ f_4(x,y,z) &= (x \wedge z) \vee (y \wedge \bar{z}) \\ f_5(x,y,z) &= x \oplus (y \vee \bar{z}) \end{aligned}$$

$f_4(x,y,z) = f_2(z,x,y)$ a $f_5(x,y,z) = f_3(y,z,x)$. Aditivní konstanty rund jsou celočíselné části druhé odmocniny (linie l) a třetí odmocniny (linie r) prvních čtyř prvočísel krát 2^{30} a 0 ($C_i^l = \sqrt{p} \cdot 2^{30}$ a $C_i^r = \sqrt[3]{p} \cdot 2^{30}$, kde $p \in \{2,3,5,7\}$; $C_i^l = 0$ a $C_i^r = 0$). Existuje deset aditivních konstant. Rozdělení bloků do linií pro kompresní funkce se řídí podle pemutací, které udávají pořadí slov. Operace každého kroku RIPEMD-160 je dána jako

$$\begin{aligned} w^l &= (w^l + f_i(x^l, y^l, z^l) + M_{\pi_i^l(k)} + C_i^l)^{\lll s_{i,k}^l} + v^l; \quad y^l = y^{l \lll 10} \\ w^r &= (w^r + f_i(x^r, y^r, z^r) + M_{\pi_i^r(k)} + C_i^r)^{\lll s_{i,k}^r} + v^r; \quad y^r = y^{r \lll 10} \end{aligned}$$

pro každou linii l a r, použitím stejného značení jako již bylo uvedeno výše. Funkce použité v jednotlivých liniích jsou různé, jedna linka má f_i, druhá f_{5-i}, v rundě i. Jedna proměnná je přičtena v každém kroku, další je posunutá o 10 bitů, což je hodnota, která již není použita v žádném jiném posunutí. Obě množiny zřetězených proměnných (vstupy kompresní funkce) se odvodí jako střední hodnoty hashů (h_0, h_1, h_2, h_3), které vrátila předchozí operace. Jakmile proběhne všech osmdesát operací, sečtou se výsledky (modulo 2^{32}) obou linií se zřetězenými proměnnými, což tvoří výsledný hash (160-bitů).

RIPEMD-128 se značně liší od RIPEMD-160. Má pět rund, ale používá se pouze čtyři 32-bitové proměnné v každé linii, dále má jen dvě aditivní konstanty v každé rundě a funkci f_5 vůbec nemá. RIPEMD-256 a RIPEMD-320 jsou dosti podobné RIPEMD-160. Rozdíl je ve vynechání sčítání (modulo 2^{32}) proměnných na konci kompresní funkce. Aby se zachovala interakce mezi oběma liniemi, jsou po každém kroku vyměněny odpovídající proměnné obou linií. RIPEMD-256 je tedy rychlejší než RIPEMD-160, ale potenciálně méně bezpečnější.

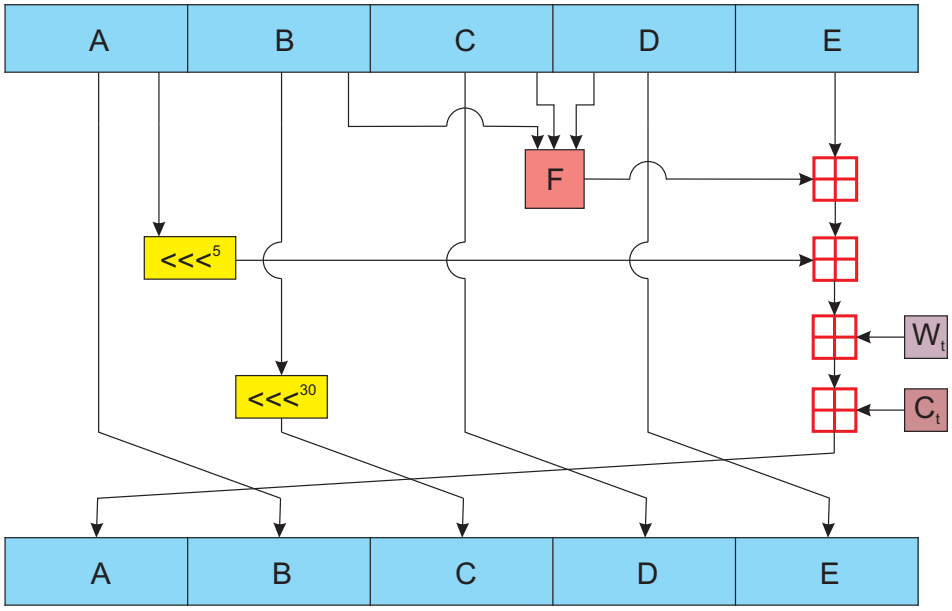
Kryptoanalýza RIPEMD-160
Na zesílené plné verze RIPEMD nebyly doposud zveřejněny žádné úspěšné kryptoanalytické výsledky. V roce 2006 byl zveřejněn útok na oslabenou verzi s 48 rundami s časovou složitostí 2^{51} [75].

7.3.7. SHA-0, SHA-1

SHA (secure hash algorithm) vznikla jako vladní standard pro NSA (National Security Agency) a byla publikována agenturou NIST. Existují tři skupiny SHA: SHA-0 (1993), SHA-1 (1995) a SHA-2 (2001). SHA-1 je vylepšená SHA-0, opravuje chyby, které obsahovala původní specifikace. SHA-2 se výrazně liší od

SHA-1, ale je stále algoritmicky podobná. SHA-0 a SHA-1 produkuje 160-bitové hashe. Struktura obou verzí je podobná MD4 a MD5, ale obsahují několik změn, které zvyšují jejich bezpečnost. V současné době je vybírán nejlepší kandidát pro SHA-3.

SHA-1 je rozšířena v bezpečnostních aplikacích, je součástí mnoha protokolů jako jsou například TSL/SSL, PGP (Pretty Good Privacy), S/MIME (Secure / Multipurpose Internet Mail Extensions) a IPSec (Internet Protocol Security). SHA-1 se nachází i v distribuovaných revizních systémech (např. Mercurial) jako prostředek pro kontrolu inregrity dat. SHA-1 a SHA-2 jsou standardem používaným ve vládních aplikacích, pro ochranu zabezpečených informací, v soukromých i obchodních institucích. Dalším důvodem existence rodiny hašovacích funkcí SHA byl také Digital Signature Standard.



Obrázek 42. Princip hašovací funkce SHA-1. SHA-1 se skládá z podobných 80 operací. Proměnné A, B, C, D, E jsou 32-bitová slova pro stav uvnitř kompresní funkce. F je nelineární funkce. Posunutí se liší pro každou operaci. C_t je unikátní konstanta rundy t. W_t je expandované slovo rundy t. $\boxplus$ značí sčítání modulo 2^{32}.

Algoritmus SHA-1

1. *Doplnění zprávy.*
   SHA používá jiný způsob doplnění zprávy jako MD4. SHA sice přidává bitovou 1 a dále potřebný počet 0, ale délka zprávy je připojena jako 64-bitové celé číslo v big-endian notaci.

2. *Rozdělení zprávy do bloků*
   SHA-1 dělí zprávu do 512-bitových bloků. Používá pět 32-bitových zřetězených proměnných a dalších 5-slov pro stav uvnitř kompresní funkce.

3. *Inicializace bufferu.*
   Čtyři zřetězené proměnné a, b, c, d jsou inicializovány stejnými hodnotami jako MD4, pátá proměnná e je inicializována stejnou hodnotou jako RIPEMD-160 tj. 0xc3d2e1f0 (f0 e1 d2 c3).

4. *Zpracování zprávy.*
   Kompresní funkce SHA-1 používá následující funkce:

$$\begin{aligned} f_1(x,y,z) &= (x \wedge y) \oplus (\bar{x} \wedge z) \\ f_2(x,y,z) &= x \oplus y \oplus z \\ f_3(x,y,z) &= (x \wedge y) \oplus (x \wedge z) \oplus (x \wedge z) \\ f_4(x,y,z) &= x \oplus y \oplus z \end{aligned}$$

   Před vyvoláním kompresní funkce dojde k expanzi 16-slovního vstupního bloku do 80 slov W_t. Prvních 16 slov je stejných jako je vstup, dalších 64 slov je vygenerováno pomocí operace XOR na předchozí čtyři slova a následovného posunutí o jeden bit:

$$W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16})^{\lll 1} \quad \text{pro } 16 \geq t \leq 79$$

   Kruhové posunutí tvoří rozdíl mezi SHA-1 a SHA-0, které nemá v této fázi posunutí definováno. Kompresní funkce má osmdesát operací ve čtyřech rundách. V každém kroku je jedna proměnná kruhově posunutá, všechny proměnné jsou prohozeny a rundovní funkce probíhá dle zápisu:

$$T = a^{\lll 5} + f_i(b,c,d) + e + C_i + W_i; \quad b = b^{\lll 30}$$

   Každá runda má unikátní konstantu C_i, které jsou $\lfloor \sqrt{r} \cdot 2^{30} \rfloor$, pro $r \in \{2,3,5,10\}$.

5. *Výstup.*
   Jakmile je ukončena kompresní funkce, je výsledek přidán k zřetězeným proměnným, které tvoří výsledný otisk. Velikost otisku je 160-bitů.

Kryptoanalýza SHA-0, SHA-1

Vzhledem k tomu, že jsou SHA algoritmy závislé na struktuře bloků a na jejich iterativní povaze, jsou SHA funkce náchylné na prodlužovací útoky (length-extension attack) a na částečné kolizní útoky (partial-message collision attacks). Útočník může vytvořit zprávu, která je podepsaná hashem s klíčem (klíčově závislá) tj. SHA(zpráva $||$ klíč) nebo SHA(klíč $||$ zpráva) a to tak, že prodlouží zprávu a vytvoří nový hash bez znalosti klíče. Způsob jak by se dalo útokům předejít

je dvojí hašování tj. $\mathrm{SHA}_a(\text{zpráva}) = \mathrm{SHA}(\mathrm{SHA}(0^b \,||\, \text{zpráva}))$, kde 0^b je nulový blok, jehož délka odpovídá velikosti bloku hašovací funkce.

První úspěšná kryptoanalytická metoda na SHA-0 byla publikována v roce 1998 (Crypto 98). Autoři uvádějí, že lze nalézt kolizi se složitostí 2^{61}. Závěry nejsou aplikovatelné na SHA-1, což naznačuje lepší bezpečnost oproti SHA-0. V roce 2004 byly nalezeny **blízké kolize** (near collisions). Dvě zprávy mají blízké kolize, pokud jsou jejich hashe velmi podobné. Hashe se lišili v 18 bitech, generování zpráv probíhá se složitostí 2^{40}. Autoři také ukázali potenciální útok na SHA-0 se sníženým počtem rund.
Později byl uveden útok na plnou verzi SHA-0, který byl schopen najít blízké kolize ve čtyřech blocích zprávy se složitostí 2^{51}. Teprve až v roce 2005 byl publikován útok, který dokázal vygenerovat kolize pro 2^{39} hašovacích operací [76].

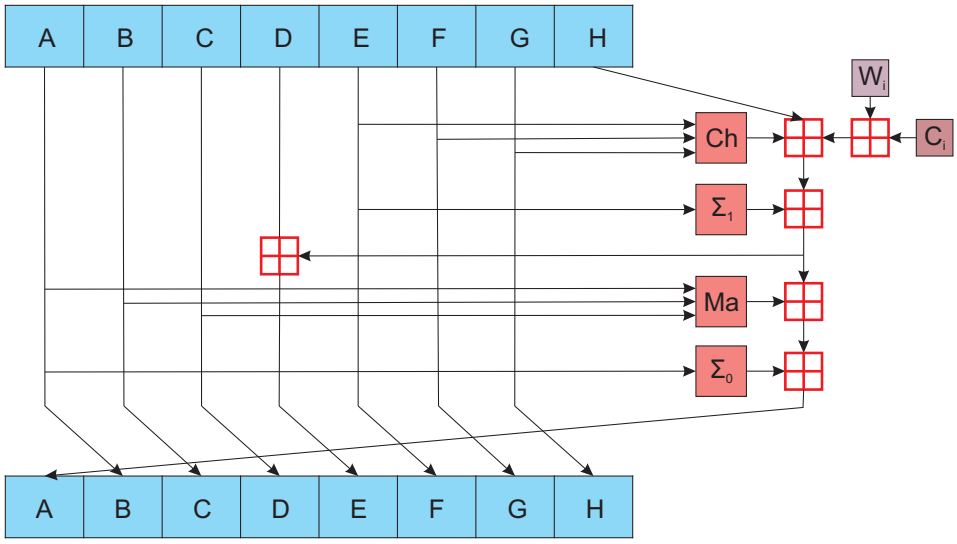
Některé metody použité při hledání kolizí u SHA-0 byly použity i na SHA-1. Existuje několik útoků na oslabenou verzi SHA-1. V roce 2005 (Xiaoyun Wang a kolektiv) byl publikován útok na SHA-1, který dokázal najít kolize pro méně jak 2^{69} hašovacích operací [77]. Dalším příkladem je kolizní útok na SHA-1 s 64 rundami. V roce 2007 vznikl projekt pro hledání kolizí na plnou verzi SHA-1, který využívá distribuovaný výpočet pomocí propojení skrz BOINC (Berkeley Open Infrastructure for Network Computing). Výzkum kolizí zahrnuje hledání dvou podobných kolizí, kdy druhá kolize vyrovnává rozdíly první. Odhaduje se složitost pro 2^{60} volání kompresní funkce. Další z projektů zaměřených na kryptoanalýzu SHA-1 je projekt HashClash s [67].

7.3.8. SHA-2

SHA-2 označuje skupinu hašovacích funkcí SHA-224 (224-bitů hashe), SHA-256 (256-bitů), SHA-384 (384-bitů) a SHA-512 (512-bitů), které vznikly pro NSA a byly publikovány agenturou NIST jako vládní standard (U.S. Federal Information Processing Standard). První tři byly přijaty v roce 2002 a čtvrtá v roce 2004. Hašovací funkce SHA-2 vznikla z potřeby vytvořit hašovací funkci s lepší bezpečností a to zejména z hlediska délky hashe. Algoritmy verzí SHA-2 jsou dosti podobné, ale značně odlišné od předchůdce SHA-1, což zajistilo jejich odolnost vůči kryptoanalytickým výsledkům SHA-1. V současné době probíhá výběr nástupce SHA-3.

Algoritmus SHA-2 (SHA-256, SHA-512)
SHA-256 používá 32-bitová slova, SHA-512 používá 64-bitová slova. SHA-256 má 64 rund a SHA-256 má 80 rund.

1. *Inicializace bufferu.*
   Obě funkce používají stejné inicializační vektory, které se skládají z osmi slov vygenerovaných z druhých odmocnin prvních osmi prvočísel. Všechny konstanty jsou uvedeny v big-endian notaci. Tedy například pro SHA-256 jsou inicializační vektory tvořeny prvními 32-bity $h(p)$ ($h[0 \dots 7] :=$



Obrázek 43. Princip hašovací funkce SHA-2. Jedna z iterací kompresní funkce, která využívá následující funkce: $Ch(E,F,G) = (E \wedge F) \oplus (\bar{E} \wedge G)$. $Ma(A,B,C) = (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C)$. $\Sigma_0(A) = (A^{\gg 2}) \oplus (A^{\gg 13}) \oplus (A^{\gg 22})$. $\Sigma_1(E) = (E^{\gg 6}) \oplus (E^{\gg 11}) \oplus (E^{\gg 25})$. Rotace jsou různé pro jednotlivé varianty. Uvedené hodnoty rotací jsou pro variantu SHA-256. $\boxplus$ značí sčítání modulo 2^{32}.

0x6a09e667, 0xbb67ae85, 0x3c6ef372, 0xa54ff53a, 0x510e527f, 0x9b05688c, 0x1f83d9ab, 0x5be0cd19).
Kompresní funkce využívá osmi vnitřních stavů a osmi zřetězených proměnných (a,b,c,d,e,f,g,h). Dále mají obě verze unikátní konstanty C_t, které lze odvodit jako hodnotu třetích odmocnin prvních šedesátičtyř resp. osmdesáti prvočísel ($C_t = \lfloor (\sqrt{p}[3] - \lfloor \sqrt{p}[3] \rfloor) \cdot 2^{32} \rfloor$, kde p_t je p-té prvočíslo).

2. *Doplnění zprávy.*
   Doplnění zprávy do požadovaného tvaru je stejné jako u SHA-1.

3. *Rozdělení zprávy, expanze bloků.*
   Každý vstupní blok zprávy 16 slov (512 nebo 1024) je rozšířen na 64 slov pro SHA-256 a 80 slov pro SHA-512. Nicméně rozšíření není stejné jako u SHA-1, je složitější. Skládá se z operací XOR, modulárního sčítání, kruhového posunutí a s dvou speciálních funkcí míchání:

$$\sigma_i \quad = \quad x^{\lll s_{i,1}} \oplus x^{\lll s_{i,2}} \oplus x^{\gg s_{i,3}} \quad i \in \{0,1\}$$

   SHA-256 a SHA-512 mají rozdílné hodnoty posunutí $s_{i,j}$. V zápisu je uvedeno vedle kruhového posunutí logické posunutí $\alpha^{\gg s}$, což uvádá posunutí bitů α o s bitů do prava, přičemž dojde k vyplnění levých s bitů 0 od základní pozice. Samotné rozšíření je realizováno vygenerováním nových 49

resp. 64 nových slov. Nejprve se vypočítají pro dvě slova hodnoty σ_0, σ_1, výsledky se sečtou s dalšími dvěma nezměněnými slovy a tím se vypočítá nové slovo W_t:

$$W_t = \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16}$$

Například pro SHA-256 lze rozšíření šestnácti 32-bitových slov na šedesátčtyři 32-bitových slov zapsat pomocí pseudokódu:

```
for i from 16 to 63
    s0 := (w[i-15] rightrotate 7) xor (w[i-15] rightrotate 18)
          xor (w[i-15] rightshift 3);
    s1 := (w[i-2] rightrotate 17) xor (w[i-2] rightrotate 19)
          xor (w[i-2] rightshift 10);
    w[i] := w[i-16] + s0 + w[i-7] + s1;
```

4. *Kompresní funkce*
   Kompresní funkce využívá dvě funkce f_1, f_2 a další dvě mixovací funkce Σ_0 a Σ_1.

$$\begin{aligned} f_1(x,y,z) &= (x \wedge y) \oplus (\bar{x} \wedge z) \\ f_2(x,y,z) &= (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \\ \Sigma_i(x) &= x^{\lll s_i,4} \oplus x^{\lll s_i,5} \oplus x^{\lll s_i,6}, \quad i \in \{0,1\} \end{aligned}$$

   SHA-256 a SHA-512 mají i zde rozdílné posunutí. Zde je jeden z rozdílů oproti SHA-1, protože ta má vedle f_1 a f_2 navíc funkci $f(x,y,z) = x \oplus y \oplus z$. Dalším rozdílem u SHA-2 je, že SHA-2 navíc používá uvedené funkce v každé rundě pro odvození nových hodnot T_1 a T_2:

$$\begin{aligned} T_1 &= h + \Sigma_1(e) + f_1(e,f,g) + C_t + W_t \\ T_2 &= \Sigma_0(a) + f_(a,b,c) \end{aligned}$$

   Nyní se prohodí proměnné a hodnoty T_1 a T_2 jsou použity pro odvození proměnné a a proměnná h je zahozena. Pomocí hodnoty T_1 se odvozuje proměnná e. Konstanty C_t jsou unikátní pro každou rundu.
   Jakmile proběhnou všechny rundy (algoritmus zpracuje všechny expandované bloky zprávy) jsou všechny proměnné $a, \ldots, f$ přidány ke zřetězeným proměnným.

5. *Odvození hashe.*
   Výslednou hodnotu otisku tvoří zřetězení všech doposud vypočtených částí hashe: digest $= h_0 + h_1 + h_2 + h_3 + h_4 + h_5 + h_6 + h_7$.

Další členové SHA-2 rodiny jsou SHA-224 a SHA-384. SHA-384 je zkrácená verze SHA-512 s jinou sadou konfiguračních proměnných. Zatímco SHA-224 je odvozen od SHA-256 zcela stejným způsobem. Kratší délku hashe lze jednoduše

Algoritmus	Délka hashe	Velikost bloku	Počet rund
GOST	256	256	32 (256 operací)
MD2	128	128	18 (864 operací)
MD4	128	512	3 (48 operací)
MD5	128	512	4 (64 operací)
RIPEMD	128	512	3 (48 operací)
RIPEMD-128/256	128/256	512	5 (64 operací)
RIPEMD-160	160	512	5 (80 operací)
RIPEMD-320	320	512	5 (80 operací)
SHA-0	160	512	80
SHA-1	160	512	80
SHA-256/224	256/224	512	64
SHA-512/384	512/384	1024	80
Tiger(2)-192/160/128	192/160/128	512	24
WHIRLPOOL	512	512	10

Tabulka 5. Přehled nejznámějších kryptografických hašovacích funkcí. Velikosti jsou uvedeny v bitech.

získat zkrácením výstupu. Jedním z rozdílů uvedených verzí jsou inicializační vektory.

Kryptoanalýza SHA-2.
Současné kryptoanalytické výsledky nepředstavují nebezpečí vůči kolizní resistanci SHA-2 i vůči její preimage resistanci (nalezení vzoru). Existuje několik prací, které například poukazují odolnost SHA-256 a SHA-512 vůči dříve zveřejněným úspěšným útokům na jiné hašovací funkce (Henri Gilbert a Helena Handschuh). Stejní autoři ukázali, že i mírné změny na SHA-256 verzi vedou k jejímu zásadnímu oslabení. Další práce ukázala důležitost funkcí Σ a σ pro bezpečnost SHA-2, bez nich mohou být kolize nalezeny se složitostí 2^{64}. Existují meet-in-the-middle útoky na oslabenou verzi SHA-2 s menším počtem rund. První z útoků je zaměřen na 41 rundovní verzi SHA-256 (časová složitost $2^{253.5}$, paměťová složitost 2^{16}) a na 46 rundovní verzi SHA-512 (časová složitost $2^{511.5}$, paměťová složitost 2^3), druhý na 42 rundovní verzi SHA-256 (časová složitost $2^{251.7}$, paměťová složitost 2^{12}) a 42 rundovní verzi SHA-512 (časová složitost 2^{502}, paměťová složitost 2^{22}) [78].

7.3.9. Přehled kryptografických hašovacích funkcí

V tabulce 5. jsou uvedeny nejznámější kryptogragické hašovací funkce. V tabulce 6. jsou vypsány nejlepší dosažené kryptoanalytické výsledky. Útoky jsou brány vzhledem k požadovanému času útoku. Pokud se jedná o útok na algorit-

mus s menším počtem rund, je její počet uveden v závorce.

Algoritmus	Kolize	2nd preimage	Preimage
GOST	ANO (2^{105})	ANO (2^{192})	ANO (2^{192})
MD2	ANO ($2^{63.3}$)	NE	ANO (2^{73})
MD4	ANO (3 operace)	ANO (2^{64})	ANO ($2^{78.4}$)
MD5	ANO ($2^{20.96}$)	NE	ANO ($2^{123.4}$)
RIPEMD	ANO (2^{18})	NE	NE
RIPEMD-128/256	NE	NE	NE
RIPEMD-160	ANO (48 rund) (2^{51})	NE	NE
RIPEMD-320	NE	NE	NE
SHA-0	ANO ($2^{33.3}$)	NE	NE
SHA-1	ANO (2^{51})	NE	NE
SHA-256/224	ANO (24 rund) ($2^{28.5}$)	NE	ANO (42 rund) ($2^{248.4}$)
SHA-512/384	ANO (24 rund) ($2^{32.5}$)	NE	ANO (42 rund) ($2^{494.6}$)
Tiger(2)-192/160/128	ANO (19 rund) (2^{62})	NE	ANO ($2^{184.3}$)
WHIRLPOOL	ANO (4.5 rundy) (2^{120})	NE	NE

Tabulka 6. Přehled nejúspěšnějších kryptoanalytických útoků na vybrané kryptografické hašovací funkce.

8. Generátory pseudonáhodných čísel

Generátory pseudonáhodných čísel (pseudorandom number generator, dále jen PRNG) jsou nedeterministické funkce, které vytvářejí výstup se statisticky podobnými vlastnostmi jako mají reálné náhodné generátory. Vygenerovaná sekvence aproximuje k reálným náhodným číslům. Sekvence je tvořena počáteční náhodnou množinou inicializačních hodnot, které se nazývají PRNG stav. K tomu, aby měl výstup požadovné pseudonáhodné vlastnosti, je potřeba zavést do průběhu generování prvek nedeterminismu. Nedeterminismus funkce bývá v reálné situaci tvořen nějakým hardwarovým vstupem jako je např. vstup z klávesnice, systémový čas, šumová dioda.

Pseudonáhodná čísla jsou důležitá z pohledu rychlosti generování a jejich reprodukovatelnosti. Používají se v kryptografii pro generování klíčů, v simulacích a v procedurální generaci. Klasickým algoritmům pro gererátory pseudonáhodných čísel je věnována kapitola Lineární zpětnovazebné registry 5.2.5..

Význam správně navrženého algoritmu generátoru pseudonáhodných čísel je patrný v případě použití u klíčů šifry. Předpokládejme, že máme šifru, která pracuje s 56-bitovým klíčem. Počet možných kombinací šifer je 2^{56}. Pokud by byl použit generátor, který by vygeneroval omezenou třídu klíčů s pouze 2^8 členy, klesla by původní složitost algoritmu na 2^8. Dojde k vnějšímu ovlivnění původního šifrovacího algoritmu, protože je útočník schopen najít jednodušeji klíč.

Většina PRNG vytváří rovnoměrné rozdělení pravděpodobností, které se dá jednoduše upravit podle požadavků např. na normální rozdělení. Teoretická kryptografie se zabývá otázkou jak se dají odlišit výstupy PRNG od náhodných sekvencí. Bezpečnost kryptografických algoritmů (např. proudové šifry), které používají PRNG, je založena předpokladu, že není možné odlišit sekvenci vyprodukovanou pomocí PRNG od sekvence reálného náhodné generátoru.

8.1. Periodicita

Každý PRNG začíná v počátečním stavu s využitím inicializační hodnoty tzv. seed. Pokud se použije stejný stav se stejným inicializačním vektorem, pak je výsledná sekvence stejná. *Perioda PRNG* je definována jako maximální délka generování jedinečných sekvencí nad množinou vstupních stavů.

Perioda je u PRNG důležitou vlastností, v praxi ji udává velikost stavu v bitech. Přidáním bitu ke stavu se perioda teoreticky zdvojnásobí, proto není složité sestavit PRNG s dostatečnou periodou. Zvýšením velikosti stavu však roste výpočetní složitost. Pokud tedy má stav PRNG n bitů, není perioda delší jak $2n$ bitů. V praxi je to většinou méně, což ale neznamená, že bude periody dosaženo. Většinou bývá vnitřní stav PRNG větší než požadovaný výstup např. PRNG s 1-bitovým výstupem. Více v kapitole Lineární zpětnovazebné registry 5.2.5..

8.2. Kryptograficky bezpečné generátory pseudonáhodných čísel

Kryptograficky bezpečné generátory pseudonáhodných čísel (CSPRNG) je třída PRNG, která musí vedle požadavku dostatečně náhodných dat splňovat požadavky na rezistenci vůči kryptoanalýze a proti možnosti zjištění stavu, ve kterém se generátor právě nachází.

V kryptografii se používají CSPRNG ke generování klíčů, generování hodnot na jedno použití tzv. *nonce* (number used once), což je pseudonáhodné číslo vydané autentizačním protokolem pro zamezení replay útoků, dále pro solení v digitálních podpisech nebo jako jednorázové šifry (one-time pad). Požadavky na kvalitu náhodných dat se liší podle typu použití. Například u nonce je požadavek na jedinečnost, generátory klíčů požadují vysokou kvalitu náhodnosti.

Narozdíl od klasických PRNG, kde je nutné vytvořit sekvenci odolnou vůči statistickým testům, požadují CSPRNG odolnost vůči statistickým testům v polynomickém čase na množině stavů. V případě CSPRNG má útočník při neznalosti stavu zanedbatelnou výhodu k rozlišení výstupní sekvence generátoru od náhodného pořadí.

8.2.1. Požadavky kladené na CSPRNG

Na CSPRNG je vedle podmínky odolnosti vůči statistickým testům kladena podmínka odolnosti v situaci, kdy útočník zná část počátečního stavu nebo části stavu, ve kterém se generátor aktuálně nachází.

Podmínky kladené na CSPRNG:

- **Next-bit test** - bez ohledu na znalost předchozích k-bitů sekvence, není schopen útočník vypočítat v polynomickém čase $(k+1)$-bit s pravděpodobností úspěchu lepší než 50%.
- **Odolnost vůči rozšíření stavu** - útočník není schopen odvodit předchozí stavy ani při znalosti části nebo celého aktuálního stavu generátoru. Navíc entropie zajišťuje, že nebude útočník schopen předpovídat budoucí stavy CSPRNG.

Většina klasických PRNG nesplňuje ani jednu z výše uvedených podmínek, protože nejsou odolné vůči reversibilnímu inženýrství ani vůči schopnosti zrekonstruovat předchozí stavy při znalosti aktuálního stavu.

8.2.2. Konstrukce CSPRNG

CSPRNG lze rozdělit do tříd podle typu kontrukce. První třídu tvoří generátory založené na kryptografických primitivech jako jsou hašovací funkce. Druhou

třídu tvoří generátory založené na obtížně řešitelných matematických problémech jako je problém kvadratických zbytků. Třetí třídu tvoří speciální generátory, které mají odlišný způsob tvorby výstupní sekvence, která není založena zcela na původním stavu.

Nyní si popíšeme některé příklady použití kryptografických primitiv jako CSPRNG.

- Proudové šifry: většina proudových šifer používá operaci XOR na vygenerované pseudonáhodné sekvence a otevřený text. Pokud se šifra nastaví do formy čítače, jsou generovány sekvence s delší periodou. Volba proudových šifer nebývá vhodná, protože jde především o bezpečnost původní šifry.
- Hašovací funkce: kryptograficky bezpečné hašovací funkce mohou vystupovat jako CSPRNG. Počáteční hodnota čítače musí být tajná a náhodná.
- Blokové šifry: CSPRNG může být ve formě bezpečné blokové šifry nastavené v režimu čítače. Klíč je opět volen jako náhodný a šifrování začíná s nějakou nastavenou hodnotou čítače. Např. šifrování 0, pak šifrování 1, šifrování 2 atd. Pro n-bitové šifry může čítač nabývat hodnoty $2n$. Inicializační hodnoty, v tomto případě klíč a otevřený text, musejí být tajné.

8.3. Vybrané typy PRNG

V následující kapitole se ukážeme klasické typy PRNG a CSPRNG, které patří do druhé a třetí třídy. Mezi další typy pseudonáhodných generátorů patří Lineární zpětnovazebné registry, kterým je věnována celá kapitola viz. 5.2.5..

8.3.1. Lineární kongruentní generátor

Lineární kongruentní generátor (LCG) je nejjednoduší a také nejméně bezpečný z níže uvedených generátorů pseudonáhodných čísel. Je založen na rekurentní relaci:

$$x_{i+1} = (ax_i + b)\,(mod\, m)$$

kde a, b, m jsou kladné, celé konstanty a x_0 je inicializační hodnota (seed).

Délka periody je maximálně m. Generátor vrací hodnoty z intervalu 0 do m. LCG není doporučeno používat v praxi, protože produkuje nekvalitní rovnoměrné rozložení. I když algoritmus produkuje pseudonáhodné čísla je citlivý na volbu parametrů a, m a c.

Mezi hlavní výhody algoritmu patří jeho pamětová nenáročnost, protože je algoritmus lineární. Není jej dobré používat v simulacích ani v kryptografii, protože trpí sériovými korelacemi. Problémem je také délka periody u lower-order bitů generované sekvence. Máme-li n-tý least significant bit základu x výstupní sekvence, kde $x^i = m$ pro nějaké celé číslo i, pak se sekvence opakuje nejvýše s periodou x^n.

8.3.2. Blum Blum Shub

Blum Blum Shub patří do třídy CSPRNG, které jsou založeny na matematicky obtížně řešitelném problému a to na problému kvadratických zbytků (Quadratic residuosity problem) [79]. Byl publikován v roce 1986, autory byly manželé Lenore Blum, Manuel Blum a Michael Shub. Doposud je známé pouze jediné řešení problému a to modulární faktorizace, proto je považován Blum Blum Shub za kryptograficky bezpečný. Algoritmus je však velmi neefektivní, proto je málo používaný a hodí se pouze v situaci, kdy je potřeba extrémní zabezpečení.

Princip generování pseudonáhodných čísel vychází z následujícího vzorce:

$$x_{i+1} = x_i^2 + (mod\, M)$$

M je definována jako součin dvou prvočísel tzv. Blumova prvočísla. Z intervalu přirozených čísel, které jsou menší jak M, se vygeneruje inicializační hodnota x_0. Další pseudonáhodné číslo je pak vygenerováno dosazením předchozího čísla do vzorce.

Bezpečnost algoritmu plyne z problému určení kvadratického residua. Pokud existuje x takové, že platí:

$$x^2 \equiv a\, (mod\, M)$$

nazýváme všechna taková x jako a kvadratický zbytek modulo M. Pokud neznáme rozklad čísla M na původní Blumova prvočísla, pak je obtížné rozhodnout, zda a paří do množiny kvadratických zbytků modulo M.

8.3.3. Fortuna

Fortuna je speciální typ CSPRNG, který kombinuje několik přístupů pro generování pseudonáhodné sekvence. Není to tedy "čistý" generátor. Fortuna je pokračovatelem šifry Yarrow [80] a její autoři jsou kryptologové Niels Fergusson a Bruce Schneier. Fortuna je kryptograficky bezpečný generátor.

Fortuna se skládá ze tří základních částí:

- **Generátor** - generuje po nastavení nespecifikované množství pseudonáhodných čísel.
- **Akumulátor entropie** - sbírá reálná náhodná data z několika zdrojů. Jakmile dosáhne dostatečné náhodnosti, přenastaví s jejich pomocí seed generátoru.
- **Seed soubor (seed file)** - slouží k uložení stavu generátoru. Jakmile získá generátor dostatečnou náhodnost, spustí pomocí seedu proces generování nového stavu.

Princip generátoru Fortuna:

1. Generátor je založen na nějaké bezpečné blokové šifře např. Twofish, AES. Bloková šifra se používá v čítačovém módu (CTR), který šifruje následující krok inkrementace. Dochází však ke vzniku statisticky zjistitelných odchylek, které jsou v čítačovém módu ošetřené, protože se klíč mění po každém vyžádání nových dat.

2. Akumulátor entropie je založen na hašovací funkci SHA-256. Akumulátor využívá celkem 32 zdrojů entropie a nastavuje se buď periodicky, nebo až nasbírá dostatečné množství entropie tzv. operace reseed. Akumulátor je odolný vůči injekčním útokům s ohledem na nenáročnost získání entropie. Zdroje entropie slouží k rovnoměrnému rozložení do výsledného seedu. Máme-li n-tý krok reseedu, je použit zdroj k pokud je n dělitelné $2k$. Ve fázi reseedu jsou méně využívány zdroje s vyšším číslem, ale zato shromažďují větší množství entropie mezi jednotlivými fázemi reseedu. Reseeding je založen na hašování specifického zdroje na klíč blokové šifry, hašování využívá dvou iterací SHA-256.

3. Seed soubor obsahuje 64 bajtů náhodných dat. Systém po restartu přečte náhodná data a provede reseed a znovu uloží aktuální hodnotu do seed souboru.

Bezpečnost Fortuny souvisí s rozložením entropie do jednotlivých zdrojů. Útočník může sice kontrolovat jednotlivé zdroje, ale některé zdroje shromažďují větší množství entropie. Jejich použitím se přímoúměrně zvyšuje množství entropie v systému, proto je odolnost vůči možné injekci zdrojů dostatečně zajištěna. Fortuna je považována za dostatečně bezpečnou pro praktické účely, zvýšením počtu zdrojů dosáhneme větší bezpečnosti.

8.3.4. Mersenne twister

Mersenne twister (MT) je pseudonáhodný generátor založený na matici lineární rekurence na konečném binárním poli F_2. Mersenne twister byl uveřejněn v roce 1997 kolektivem autorů Makotem Matsumotem a Takuji Nishimurou. Je velmi efektivní z hlediska generování kvalitních pseudonáhodných čísel i z hlediska jeho rychlosti. Použití MT v praxi není doporučeno, protože je založen na rekurzi a umožňuje útočníkovi při znalosti dostatečně dlouhého výstupu odhalení následujícího výstupu.

Generátor je pojmenovaný podle Mersennova prvočísla, což je obecně takové prvočíslo x splňující podmínku $x = 2^n - 1$. Prvočíslo určuje i periodu generátoru. MT vznikl pro Monte Carlo simulace a je základem mnoha generátorů jazyků jako je např. Python.

Mezi hlavní výhody MT patří dlouhá perioda a jeho výborná statistická odolnost. Nejznámější variantou MT je MT19937. Tato varianta produkuje sekvenci

32-bitových celých čísel a má poměrně velkou periodu $2^{19937} - 1$. Mezi výhody patří k-distribuovatelnost s 32-bitovou přesností, pro každé $k \in < 1, 623 >$. Hodnota 624 udává velikost stavového vektoru.

Nevýhodou je již citovaná rekurze, která umožňuje útočníkovi již při 624 iteracích u varianty MT19937 předpovědět budoucí iterace. Další nevýhodou je poměrně dlouhá doba vygenerování nenáhodného počátečního stavu (např. s řadou nul) do výstupu.

9. Kvantová kryptografie

Kvantová kryptografie (kvantová komunikace, kvantová distribuce klíčů) využívá principů kvantové mechaniky k vytvoření bezpečné komunikace. Základní vlastností kvantové kryptografie je její schopnost odhalit přítomnost třetí osoby, která komunikaci odposlouchává, protože dochází k narušení stavu přenosu.

Odposlouchávání komunikačního kanálu je vlastně proces měření veličin. Důsledkem kvantové mechaniky je, že měření ovlivňuje stav systému a změnu je možno odhalit pomocí fyzikálních metod. Jedná se o man-in-the-middle útoky, kterým nedokáže kvantová kryptografie zabránit, ale dokáže je obecně odhalit. K detekci narušené komunikace existují dvě základní metody: **měření polarizace fotonů** (kvantová superpozice) a **metoda propletení fotonů**.

9.1. Kvantová informace

Kvantová informace zajišťuje, že nelze neznámý kvantový stav bezchybně zkopírovat. Z pohledu kryptografie je zajímavá také možnost kvantového počítání. Kvantová informace umožňuje díky principu superpozice paralelní zpracování velkého množství kvantových bitů tzv. **qubit**. Metoda využívá linearity a principu superpozice kvantových stavů, což v případě složitějších úloh radikálně snižuje složitost výpočtu. Praktické použití metody souvisí s problémem konstrukce kvantových počítačů, které jsou doposud v teoretické úrovni.

9.1.1. Základní pojmy kvantové informace

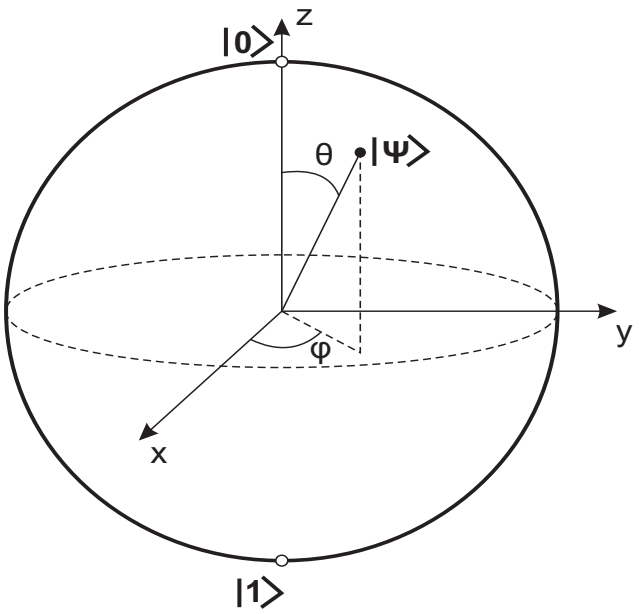
Kvantová informace je založena na sekvenci kvantových bitů. Qubit viz. 44. je reprezentován logickou nulou, logickou jedničkou nebo kvantovou superpozicí, která umožňuje aby se fyzikální systém nacházel částečně ve všech možných stavech současně, ale výsledné měření vede pouze k jednomu výsledku ze všech možných konfigurací. Pro trojici qubitů dostaneme superpozici osmi stavů. Obecně, n qubitů se pro libovolnou superpozici může současně nacházet v 2^n stavech.

Kvantová informace je vyjádřena pomocí bázových stavů v dvoudimenzionálním Hilbertově prostoru. Hilbertův prostor je vektorový prostor, kde je možné měřit velikosti vektorů a jejich úhly a zároveň ortogonálně projektovat vektory na podprostory. Kvantový bit je libovolná superpozice bázových stavů viz. 45.:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle,$$

kde α a β jsou komplexní čísla, pro které platí $|\alpha|^2 + |\beta|^2 = 1$. Vzhledem k tomu, že jsou pravděpodobnostní amplitudy tvořeny komplexními čísly, hraje fáze mezi jednotlivými stavy významný parametr.

Pro změření kvantového stavu je potřeba provést projekční měření. Měření je založeno na pravděpodobnostním charakteru kvantového světa, který vede k



Obrázek 44. Reprezentace qubitu v Blochově prostoru. Pravděpodobnostní aplitudy jsou dány vztahy $\alpha = \cos(\frac{\theta}{2})$ a $\beta = e^{i\phi} \sin(\frac{\theta}{2})$

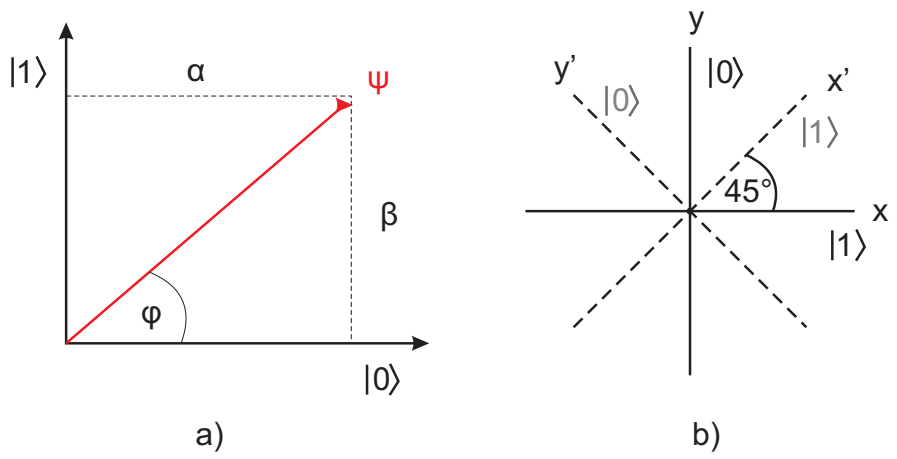
projekci $|\psi\rangle$ do stavu $|0\rangle$ s pravděpodobností $|\alpha|^2$ nebo do $|1\rangle$ s pravděpodobností $|\beta|^2$. K určení stavu jedním měřením je potřeba, aby $|\psi\rangle$ vedlo k jednomu z bázových stavů $|0\rangle$ nebo $|1\rangle$. Určení úhlu měření φ není jednoduché, protože z vlastnosti kvantové mechaniky plyne, že měření mění kvantový stav. Pro měření $|\psi\rangle$ je tedy potřeba měřit soubor kvatových stavů, které vznikly za stejných podmínek. Na vstup se tedy pošle soubor kvantových stavů. Následně se pro měření použije takový výsledek, kdy měření kvantového bitu odpovídá bitu klasické informace.

9.1.2. Měření kvantové informace

Pro měření kvantové informace je potřeba vytvořit takové médium, které umožňuje jednoduchou manipulaci a změnu kvantového stavu. Stavy musí mezi sebou interagovat a musejí být snadno měřitelné. Kvantový stav se nesmí volně měnit ani v čase ani bez interakce ani měřením. Dodržení všech podmínek je poměrně složité.

Existuje několik metod, které se dají pro měření kvantové informace použít, ale jejich složitá technická realizace resp. případné nevýhody vedou k jejich nepoužitelnosti v praxi.

Pro kódování kvantových stavů se nejlépe hodí kódování do vlastností **fotonů**.



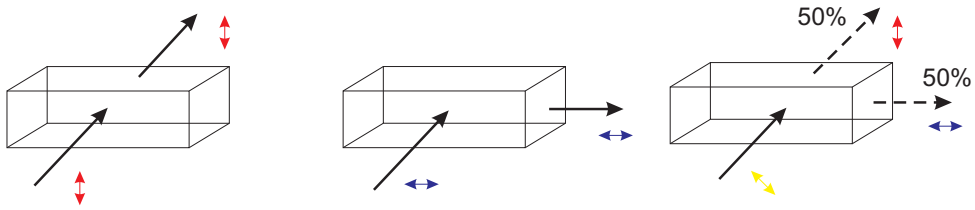
Obrázek 45. Reprezentace kvantové informace. a) Kvantový stav, b) Polarizační báze.

Foton je kvantum světla předávané atomem při absorbci světla, atom předává energii diskrétně. Kvantovou informaci určuje polarizační stav fotonu. Polarizované světlo je takové světlo, u kterého vektor elektrického pole kmitá ve stále stejného směru. Bázové stavy mohou například tvořit:

- vertikální nebo horizontální lineární polarizace. Superpozice stavů je pak eliptická polarizace.
- kódování do dráhy fotonu. Bázi tvoří optická vlákna (prostorové dráhy). Foton se může šířit oběma drahami současně (oddělené časové intervaly) s určitou pravděpodobností.
- kódování stavu do spektra.

Jako polarizátor je použit nějaký skleněný hranol, nebo polarizační destička viz. 46.. Kvantová kryptografie je založena na navzájem kolmých lineárních polarizacích ze dvou polarizačních bází pootočených o 45°. Problémem je dosažení kvalitní interakce kvantových bitů mezi sebou. Důležitá je nerozlišitelnost fotonů, což je poměrně náročné. I přes nevýhody účinnosti detektorů fotonů, jsou fotony nejrozšířenější nosiče kvantové informace.

Praktické měření je založeno na skutečnosti, že hranol nechá horizontálně polarizované fotony projít skrz a vertikálně polarizované fotony odklání mimo osu vstupních fotonů. Diagonálně polarizované fotony se s poloviční pravděpodobností odkloní (nyní vertikální polarizace) nebo projdou skrz (nyní horizontální polarizace). Vlastnost komplementárnosti bází vede k závěru, že nelze změřit současně foton v obouch fázích aniž by se polarizace změnila. Kvantová kryptografie



Obrázek 46. Měření polarizace.

využívá této vlastnosti, jedná se Heisenbergův princip neurčitosti. Při odposlechu se změní polarizace fotonu a příjemce je schopen útok odhalit. Vzhledem k tomu, že je jeden bit vyslán na trilionech fotonů, používá se jako médium optické vlákno.

9.1.3. Kvantová dekoherence

Kvantová dekoherence se ztráta soudržnosti nebo uspořádání fázových úhlů mezi prvky kvantové superpozice. Základní myšlenka dekoherence spočívá ve vysvětlení kolapsu vlnové funkce vzniklé provázáním superponovaných stavů s okolním světem. Dekoherence nastane, pokud systém interaguje se svým prostředím nevratným způsobem. Obecně řečeno, kvantovou dekoherenci lze považovat za nevratnou ztrátu údajů. Řešení dekoherence je jedním z problémů, který omezuje stavbu kvantového počítače.

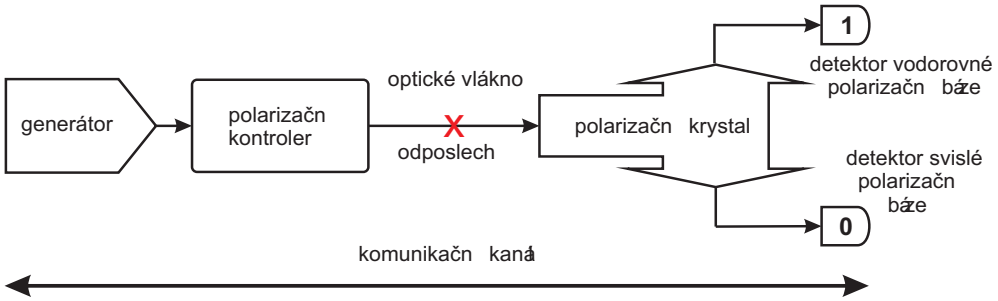
Jednou z metod řešení kvantové dekoherence je kvantová oprava chyb, která ale dokáže opravit ztrátu údajů v omezené míře a navíc zvyšuje složitost výpočtu, protože potřebuje vetší množství qubitů.
Z důvodu kvantové dekoherence se zavádí míra chybovosti, která určuje procento tolerovaných chyb.

9.2. Kvantová komunikace

Kvantová komunikace (kvantová ditribuce klíčů (QKD)) je bezpečná výměna klíčů pomocí poznatků kvantové mechaniky. Kvantová komunikace zajišťuje bezpečnou výměnu klíčů mezi dvěmi stranami, které pak pomocí klíče šifrují zasílané zprávy. Z poznatků kvantové mechaniky plyne, že jsou komunikující strany schopny detekovat odposlech třetí strany, protože měření klíče mění přenášenou informaci. Pokud je míra chybovosti pod stanovenou hranicí, komunikace probíhá dále, v opačném případě je klíč zahozen.

9.2.1. Příklad kvantové komunikace

Předpokládejme, že spolu chtějí komunikovat dvě strany A (Alice) a B (Bob).



Obrázek 47. Příklad kvantové komunikace.

Předpokládejme, že strany používají pouze kolmou polarizační bázi (x, y) viz. 45.. A zasílá vertikálně a horizontálně polarizované fotony. Strana B přijímá stejnou sekvenci fotonů, protože používá stejnou bázi - chování fotonů na polarizátoru je plně deterministické. Předpokládejme, že se třetí strana E (Eva) napojí na komunikační kanál a chce komunikaci odposlouchávat. Kvantová mechanika zajišťuje, že nelze foton rozdělit a pasivní komunikace není možná. Foton buď bude pokračovat ve směru komunikace, nebo bude odkloněn. Příjemce pak útok detekuje jako pokles výkonu.

Útočník vloží do komunikačního kanálu vlastní zařízení, které pracuje stejně jako zařízení na straně příjemce. Protože nezná bázi stavů používanou stranami A a B, odesláním nového bitu straně B způsobí chyby. Komunikující strany si po dokončení první komunikace vyberou pouze bity pro zvolenou bázi. Báze je nutné měnit, protože by byl útočník při její znalosti neodhalitelný.

Komunikující strany mají předem definované báze a po ukončení bezpečné komunikace bez detekce odposlechu, si vzájemně oznámí jakou bázi použili. Pokud útočník zná polarizační báze, je schopen zvolit správnou s 50% úspěšností. Nepřetržitý odposlech způsobí průměrně 25% chyb. Pravděpodobnost odhalení lze vyjádřit vztahem:

$$P = 1 - \left(\frac{3}{4}\right)^n,$$

kde n je počet testovaných bitů. Pro dvacet testovaných bitů dosahuje pravděpodobnost odhalení útočníka 99,68%.

9.2.2. Vlastnosti kvantové komunikace

Základním kódovacím elementem kvantové komunikace je foton. Moderní přenosové sítě využívají těchto prvků. Použití fotonů má své výhody, ale také nevýhody. Kvantová kryptografie je **bezpodmínečně bezpečná**, což znamená, že bezpečnost není podmíněna schopnostmi útočníka.

Kvantová komunikace zabezpečuje bezpečnou distribuci klíčů s možností detekce odposlechu, ale neřeší již problém autentizace tj. případného přerušení komunikačního kanálu a odcizení identity jedné ze stran. Zprávy je potřeba autentizovat heslem, které se po každém přenosu nahradí.

Samotné přenosové prostředí přináší nevýhody v podobě rušení a to hlavně polarizační vidovou disperzí, která ovlivňuje polarizační stavy. Přenos je vždy spojovám se zanesením šumu do přenosového kanálu, čímž vzniká problém pro stranu B ve schopnosti odlišení rušení a odposlechu. Mezi další nevýhody patří cenová a technická náročnost síťí. Pro kvantovou kryptografii doposud neexistují dostatečně účinné a rychlé metody a technické zázemí při řešení úloh.

Pro obnovení ze zanesení šumu, který je vyvolaný rušením resp. odposlechem existují metody rekorelace. Rekorelace má za úkol zajištění identičnosti klíče strany A a B. Ke korelaci vznikl kaskádový protokol, který v několika rundách kontroluje paritu po sobě jdoucích bloků klíče. Protokol vytváří kaskádová jména a přerovnává bloky klíče strany A a B. Na konci budou mít obě strany s vysokou pravděpodobností stejný klíč. Pro zajištění vyšší bezpečnosti klíče se používá metoda rozšíření bezpečnosti, která vytváří nový kratší klíč například pomocí hashe a zabraňuje tak použití i částečné znalosti klíče straně E.

9.2.3. Protokol BB84

Protokol BB84 je založený na principech kvantové mechaniky a umožňuje vytvoření a distribuci náhodného klíče. Přenášené bity jsou určeny polarizací fotonů. BB84 vznikl v roce 1984 a nese název podle svých autorů: Charles H. Bennett a Gilles Brassard. Komunikující strany jsou propojeny optickým vláknem a ke komunikaci používají ještě jiný druh kanálu, kterým si ověřující správnost příchozích hodnot. Metoda je poměrně oblíbená, ale vzhledem k omezení zapojení zesilovačů, dovoluje komunikaci jen na krátkou vzdálenost.

Metoda vychází z poznatků uvedených výše. Polarizovaný foton buď projde polarizačním filtrem, nebo jím neprojde. Projdou fotony, které jsou polarizované shodně s rovinou daného filtru. Foton polarizovaný kolmo na rovinu, neprojde nikdy. U fotonů, které jsou polarizované v jiném úhlu roviny, dojde k jejich náhodnému výběru na základě pravděpodobnosti danou skutečným úhlem polarizace.

Pro vlastní zakódování bitů jsou využity čtyři stavy polarizace, kde každé dva a dva jsou vzájemně kolmé. Dvojice ortogonálních stavů tvoří bázi. Polarizace dvojic je buď shodná (vertikální) s rovinou tj. 0°, nebo je horizontální 90°, nebo je báze diagonální 45° a 135°, nebo jsou báze kruhově posunuté.

Princip komunikace si ukážeme na příkladu stran A a B, kde A je odesílatel:

1. Odesílatel A si náhodně vybere vysílací polarizační bázi 7..
2. Odesílatel A zakóduje zprávu do polarizací vysílaných fotonů 8..

Báze	0	1
+	↗	↘
×	→	↑

Tabulka 7. Polarizační báze.

3. Příjemce B si náhodně vybírá přijímací polarizační bázi a dekóduje přijatou zprávu. 7., 8..
4. Strana B oznámí straně A použité polarizační báze a strana A dobře použité báze potvrzuje.
5. Pokud nedošlo k rušení ani nebyl přítomen útočník, má strana B odeslanou zprávu.
6. Nyní provede B test na odhalení třetí strany E. K tomu potřebuje nějaké bity z přijaté zprávy, ty pak zahodí.
7. Strana A potvrdí použité bity pro test a bity ze zprávy odstraní. Pokud byl přítomen útočník, jsou detekovány odchylky.
8. Jako klíč slouží zbylé bity.

Sekvence bitů strany A	1	1	0	0	1	1	1	0
Náhodně zvolená báze strany A	+	+	×	+	×	+	×	×
Polarizované fotony strany A	↘	↘	→	↗	↑	↘	↑	→
Měření na základě báze strany B	×	+	×	×	+	×	+	+
Polarizace změřené stranou B	↑	↘	→	→	↘	↑	↘	↗
Potvrzení bází a test narušení								
Tajný klíč		1	0		1			0

Tabulka 8. Ukázka komunikace pomocí BB84 protokolu.

9.2.4. Protokol E91

Protokol E91 je založen na tzv. propletenosti fotonů (metoda korelovaných stavů dvojice částic). Metoda je založena na stavu dvojic částic. Pokud dojde k měření jedné částice z dvojice, vede to ke kolapsu vlnové funkce, která dvojici popisuje a dojde ke změně stavu druhé částice z dvojice.

Kolaps vlnové funkce znamená redukci ze superpozice několika vnitřních stavů na jeden z těchto vnitřních stavů. Vlnová funkce je řešením vlnové rovnice, která je

většinou parciální diferenciální rovnicí prvního nebo druhého řádu. Vlnové rovnice popisují stav fyzikálního systému. U vlnových funkcí lze určit pravděpodobnost s jakou dojde k naměření určité hodnoty fyzikální veličiny.

Protokol generuje díky zdroji dvojice, která mají spiny ve stavu superpozice nahoru a dolů, ale zároveň jsou díky vlastnosti propletenosti vzájemně korelované. Spin je jedinečná kvantová vlastnost, která udává vnitřní moment hybnosti částice ve smyslu ovlivnění momentu hybnosti celkové soustavy.

Princip protokolu je založen na skutečnosti, že výsledek měření je zcela náhodný a nepředvídatelný. Distribuce tajného klíče protokolem E91:

- Zdroj, který generuje fotony může být umístěn na straně A nebo B nebo úplně někde jinde.
- Jeden foton se pošle straně A a druhý z dvojice fotonů se pošle straně B.
- Strana A provede měření a zaznamená si hodnotu bitu.
- Strana B si dekóduje příchozí foton a z vlastností stavu superpozice ví, že je to opačná hodnota než má strana A.
- Proces generování a dekódování se opakuje, dokud nezískají obě strany tajný klíč.
- Protokol vygeneruje zcela náhodný klíč, který mohou strany A a B použít pro další komunikaci.

Pokud by se pokusila strana E přenos odposlouchat a došlo by při měření fotonů, porušila by se korelace mezi výsledky A a B. Začnou platit Bellovy nerovnosti, které kvantová mechanika porušuje. Po přijetí klíče obětují strany A a B část bitů a ověří si správnost výsledků. Metoda je omezená dekoherencí, protože jsou propletené stavy fotonů stavem superponovaným.

9.2.5. Bezpečnost kvantové komunikace

Z hlediska kvantové mechaniky je kvantová komunikace bezpečná. Útoky jsou realizovány pomocí oslabení metod jako je například zmatení příjemce pomocí oslepení detekčních diod. Jak již bylo řečeno výše, kvantová komunikace je v ideálním případě bezpodmínečně bezpečná. Komunikující strany musí zajistit následující podmínky:

- Strana E nemá přístup k mechanismu kódování resp. dekódování stran A a B.
- Samotné šifrování zprávy je bezpodmínečně bezpečné.

- Ke generování náhodných hodnot používají strany A a B bezpečné a dostatečně náhodné generátory (kvantové generátory náhodných čísel).
- Proces ověřování hodnot jako jsou kvantové báze je realizován bezpečnou cestou.

Kvantová komukace je náchylná k **man-in-the-middle útoku** a není schopna odposlechu zabránit. Poznatky kvantové mechaniky sice zajišťují schopnost detekce, ale vnější vlivy, jako je rušení charakterem komunikačního kanálu, detekci ztěžují. Samotná kvantová komunikace neřeší autentizaci stran A a B. Ověření autentičnosti stran může řešit předem stanové počáteční nastavení, které se pak zanese do procesu generování klíče.

Kvantová komunikace je založena na polarizaci fotonů. U protokolu BB84 jsou kvantové stavy přenášeny pomocí fotonů. V praxi jsou fotony generovány pomocí pulsů laseru a jsou distribuovány pomocí Poissonovy distribuce. V ideálním případě obsahuje puls jeden foton. Někdy pulsy neobsahují žádné fotony (není odeslán), nebo naopak mají více fotonů. Této skutečnosti využívají **splitting útoky** (například PNS útok), kdy se puls rozdělí, jeden foton se pošle straně B a strana E si zbytek ponechá. Strana B odposlech nedetekuje, strana E si ukládá fotony do kvantové paměti a na konec získá částečnou informaci o klíči. Dochází ke snížení stupně bezpečnosti klíče. Řešení se nabízí v podobě jednotného zdroje fotonů, které ale v současnosti nedosahují potřebné účinnosti. Jako další řešení se nabízí použití protokolu SARG04 [58], který nabízí vyšší stupeň bezpečnosti klíče pro potenciální PNS útoky. Pro detekci PNS útoků vznikla metoda, při které strana A náhodně posílá menší než průměrný počet fotonů tzv. návnadu, strana E nemá šanci odhalit které pulsy jsou testovací a strana B pak případný PNS útok odhalí.

S bezpodmínečnou bezpečností souvisí tzv. **kvantové závazky**. Pokud strana A zaváže nějakou hodnotu, pak nikdo, kromě strany A, nemůže hodnotu měnit a zároveň ani strana strana B nemůže hodnoty změřit aniž by je strana A odhalila. Mayers ukázal, že je podmínka závazku nesplnitelná, ale tím není vyloučeno, že jí nebude možno dosáhnout. Možným řešením jak kvantové závazky realizovat je **metoda omezeného kvantového prostoru** (Bounded quantum storage model (BQSM)). BQSM využívá skutečnosti, že při generování velkého množství quitů, musí útočník část uložených údajů zahodit, protože disponuje omezenou kapacitou prostoru. Tím nemůže po skončení provést rekonstrukci klíče. Skladování qubitů po delší dobu je poměrně náročné. Rozšířením BQSM je metoda NQSM (noisy), která zanáší do komunikačního kanálu chybovost v podobě rušení a útočník disponuje daty, které nedokáže opravit. Metody jsou pro současné protokoly nepraktické, protože vyžadují po komunikujích stranách velké množství paměti.

Další publikovanou metodou vůči potencionálním útokům je metoda **quantum tagging**, která omezuje možnost dekódování na určité zeměpisné místo.

Později byla tato metoda zpochybněna protokolem, který využíval velký počet propletených fotonů, ale metodu lze použít v kombinaci s BQSM.

V roce 2011 byl uveřejněn útok tzv. **oslepením**, kterým se podařilo zkopírovat klíč aniž by došlo k detekci odposlechu. Metoda byla nasazena na BB84 protokol, který pro detekci používal lavinové fotodiody, které se dají zmást silným světlem, takže ztrácejí schopnost detekce jednotlivých fotonů a reagují pouze na intezitu světla. Strana B není schopna správně přiřazovat polarizéry a při procesu porovnávání klíčů odhalit odposlech. Autoři navrhli i řešení tím, že umístili před detektor zdroj jednotlivých fotonů pro ověření, zda je detektor schopen zpracovat jednotlivé fotony.

Mezi další typy útoků se řadí útoky zaměřené na nedostatky kvantových generátorů, nebo útoky trojským koněm. Druhý typ útoků neměří průchozí fotony, ale vyšle impuls světla straně A a zpětnou vazbu využije k odhadu polarizačního stavu strany A. Dalším typem útoku na je útok s časovým posunem, který byl jako první aplikován na komerční kvantovou komunikaci.

9.2.6. Budoucnost kvantové komunikace

Rozšíření kvantové komunikace v současnosti brání její nedostatky jako jsou omezenost vzdálenosti komunikujících stran, přítomnost rušení kanálu, cena realizace sítí nebo celková složitost realizace úloh a pamětová náročnost. Kvantová komunikace se nyní hodí pro oblasti, které vyžadují zvýšenou bezpečnost utajení informací. Budoucnost sítí v podobě optických vláken dává kvantové komunikaci velký potenciál.

10. Autentizační protokoly

Autentizační protokoly jsou schémata, která umožňují předvést znalost sdíleného tajemství. Zároveň nesmí poskytovat útočníkovi žádnou užitečnou informaci, kterou by mohl útočník použít pro opakovanou autentizaci a následný přístup do systému. Autentizační protokoly jsou nejčastěji budovány na základě nějakého silného kryptografického návrhu a pracují principem dotaz - odpověď.

Mezi základní pojmy patří **autentizace** a **identifikace**. Identifikace subjektu je tvrzení o existenci určité entity z hlediska totožnosti. Autentizace je proces ověření identifikace subjektu. Autentizace probíhá ve dvou krocích:

1. Identifikace - subjekt předá systému identifikátor.
2. Ověření - subjekt předá systému svoji autentizační informaci, která potvrzuje jeho svázanost s identifikátorem.

Konkrétní realizace autentizace záleží na povaze systému, která určuje jeho roli v procesu autentizace.

Autentizační protokoly musí splňovat kritéria bezpečnosti jako je například test CIA (Confidentiality-Integrity-Availability). Principem CIA je zajištění integrity dat, dostupnosti a důvěrnosti.

10.1. Všeobecný pohled na autentizaci

Autentizaci je možné definovat dle charakteru použitého prostředku autentizace: identifikátor + heslo, mechanická identifikace (např. karta), osobní identifikace (biometrie).

10.1.1. Klasická hesla

Hesla jsou nejčastějším prostředkem identifikace subjektu v systému. Jsou nejsnadnějším způsobem, ale také nejméně bezpečným. Z hlediska potenciálních útoků nabízejí širokou škálu možností jako je prosté odvození, slovníkový útok, hrubá síla, napadení hesel v databázi, keyloggery atd. Z hlediska klasických hesel je důležitá konkrétní implementace autetizačního protokolu. Mechanismus tvorby hesel může být doplněn tzv. solením, což je metoda tvorby hesla generováním náhodné hodnoty, která odpovídá páru hesla. Teprve ve spojení s heslem dojde k vytvoření otisku.

Speciálním typem hesel jsou jednorázová hesla, která se například používají v oblasti bankovnictví a to ve spojení s heslem nebo PINem klienta. Vedle klasických textových hesel se zavádějí grafická hesla. Dalším typem zabezpečení jsou časové známky, které po vypršení zamezí další komunikaci. U hesel se klasicky testuje síla hesla, kterou si systém sám definuje. Autentizační protokol si tak vynucuje vyšší bezpečnost a zabraňuje klasickým chybám ze strany uživatele.

10.1.2. Pokročilejší mechanismy tvorby hesla

Mezi pokročilejší mechanismy tvorby hesel řadíme autentizační protokoly, které umožňují bezpečnou komunikaci při použití hesla uživatele. Protokoly se zaměřují na odstranění bezpečnostních mezer, které umožňují útoky citované výše.

K bezpečnému spojení na otevřené lince vznikla modifikovaná verze Peyravian-Jeffries protokolu, která je nazývána podle svých autorů: **Hölbl, Welzer, Brumen** [81]. Protokol je založen na Diffie-Hellmanově výměně klíčů a využívá k výpočtu klíčových hodnot nějakou bezpečnou hašovací funkci. Není vázán platformou ani systémem. Protokol umožňuje bezpečnou komunikaci mezi klientem a serverem za použití identifikátoru a hesla uživatele. Na základě uloženého identifikátoru a zprávy si klient a server vypočítá hodnot MAC viz. 5.3.5.. Komunikace probíhá výměnou MAC hodnot serveru a klienta:

- Server si defaultně vypočítá na základě dodaného identifikátoru a hesla autentizační hodnotu, kterou si uloží.
- Klient naváže spojení se serverem a požádá o ověření.
- Server si na základě uloženého identifikátoru vypočte kontrolní hodnotu a pošle ji klientovi.
- Klient si nejprve ověří z přijaté zprávy, že se jedná o server a teprve pak pošle serveru kontrolní hodnotu.
- Server kontrolní hodnotu ověří podle uloženého identifikátoru a teprve nyní klienta autetizuje.

Změna hesla je poměrně jednoduchá. Po úspěšné autentizaci pošle klient žádost o změnu hesla s novou kontrolní hodnotou. Server si hodnotu ověří a změní si uložený identifikátor, který je příslušný k novému heslu.

Dalším příkladem je autentizační protokol **S/KEY**, který vznikl pro Unixové systémy poskytující nedokonalé zabezpečení. Protokol generuje jednorázové heslo, které vznikne opakovým použitím hašovací funkce. Heslo se generuje pomocí rekurentního procesu, který pomocí čítače opakovaně mění hodnotu hesla. Hesla vznikají na straně klienta a server pouze hesla ověřuje. Klient si buď předem připraví seznam jednorázových hesel, nebo, což je častější, vznikají hesla jednorázově na požádání.
Princip protokolu S/KEY:

- Nejprve si klient vytvoří nějaké heslo W, které bude inicializační hodnotou hašovacích funkcí.
- Klient a server se domluví na nějaké tajné hodnotě N. N bude určovat hodnotu čítače, tedy počet opakování hašovací funkce.

- Klient si rekurentně vytvoří $H(W), H(H(W)), \ldots, H^N(W)$.
- Původní hodnota W se zničí a klient serveru pošle hodnotu $H^N(W)$. Server žádnou jinou hodnotu nemá.
- Při autentizaci požádává server klienta o hodnotu $H^{N-1}(W)$.
- Klient pošle serveru svoji uloženou hodnotu $H^{N-1}(W)$ a server si vypočítá $H(H^{N-1}(W))$ a porovná ji se svojí hodnotou $H^N(W)$.

Bezpečnost S/KEY závisí na bezpečnosti hašovací funkce. Protokol je náchylný na man-in-the-middle útoky. Útočník se napojí na server a pokouší se určit N-1 znaků hesla pomocí opakovaného zasílání požadavků pro zjistění platných znaků na N-té pozici.

10.1.3. Bezpečnostní tokeny

Hardwarové tokeny (bezpečnostní tokeny) využívají ke své autentizaci nějaké fyzické zařízení jako jsou jednočipové karty, USB tokeny atd. Výhoda bezpečnostních tokenů je skutečnost, že si uživatel nemusí pamatovat přihlašovací údaje, protože jsou součástí zařízení. Bezpečnostní tokeny jsou rozšířeny druhotnou autentizací, která token chrání před jeho zneužitím.

Hardwarové tokeny zajištují neproniknutelnost k privátnímu klíči tzv. tamper resistance. Použitím bezpečnostního tokenu odpadá nutnost zadávat identifikátor a heslo. Problém nastává při ztrátě tokenu, protože může dojít ke zneužití legitimity majitele. Proto bezpečnostní tokeny požadují ověření autentizace ve dvou fázích:

1. fyzické držení tokenu
2. zadání bezpečnostního údaje jako je PIN, u kterého je omezen počet neúspěšných zadání.

Bezpečnost tokenu je také závislá na konstrukci šifrovacího algoritmu a délce použitého klíče. Hardwarové tokeny jsou nejrozšířenější v podobě jednočipových karet a to hlavně v bankovní oblasti. Příkladem nevhodného bezpečnostního tokenu je SecureID (hardwarový autentizační klíč společnosti RSA), který obsahoval hašovací funkci SecurID Hash Function a speciální blokovou šifru. U hašovací funkce byly nalezeny kolize a bloková šifra byla prolomena pomocí chosen-plaintext útoku s 2^{48} otevřených textů.

10.1.4. Biometrika

Biometrika (biometrické systémy) je založena osobní identifikaci. Osobní identifikace probíhá na základě jedinečných tělesných znaků. Výhoda biometriky je

neměnnost identity v čase a zároveň si subjekt nemusí pamatovat přihlašovací údaje ani nosit bezpečnostní token.

Mezi hlavní podmínky biometriky patří: jedinečnost, univerzálnost, neměnnost v čase, měřitelnost údajů a uživatelská příjemnost. Mezi měřené biometrické prvky patří: otisk prstu, geometrie ruky, duhovka oka, hlas, geometrie tváře atd. Pro sjednocení podmínek biometrie vznikl mezinárodní standart BioAPI, který definuje rozhraní mezi moduly od různých výrobců.

10.2. Přehled autentizačních protokolů

V následující kapitole si popíšeme nejznámnější autentizační protokoly. Budeme postupovat podle modelu ISO/OSI a to od nejnižší logické komunikační vrstvy. Vzhledem k rozsahu problematiky se zaměříme jen na vysvětlení základních principů.

10.2.1. Password authentication protocol

Password authentication protocol (PAP) je protokol používaný v Point to Point protokolu (PPP) [82] pro ověření autentizace před přidělením povolení k přístupu na server. PAP byl hojně rozšířený v systémech podporující remote control server. PAP je jedním z nejstarších protokolů a je považován za nebezpečný, protože přenáší hesla v otevřené podobě. V současnosti se používá pouze v případě, že server jiný protokol nepodporuje. Více informací k PAP [83].

PAP funguje pomocí dvoucestného handshake, kdy se nejprve vytvoří klientská výzva (Authentication-request), při které klient posílá serveru dvojici (identifikátor, heslo). Server pošle odpověď (authentication-ack), která je buď kladná nebo záporná. Klient opakovaně posílá serveru rámce a pokud mu server po stanovém počtu odeslaných rámců neodpoví, komunikaci ukončí.

PAP nepodporuje žádné kryptografické zabezpečení. Heslo je přenášeno v otevřené podobě. Pokud se útočník napojí na komunikaci mezi klienta a server a odcizí přenášené údaje, pak se může jednoduše vydávat za klienta.

10.2.2. Challenge-Handshake Authentication protocol

Challenge-Handshake Authentication protocol (CHAP) je nástupcem PAP, který používá k autentizaci hašovací funkci. Protokol opakovaně mění identifikátor a tajnou hodnotu. Heslo není nikdy přenášeno v otevřené podobě. Více informací k CHAP [53] [84].

CHAP je třífázovým handshake protokolem, který v pravidelných intervalech ověřuje totožnost klienta. První ověření je realizováno pomocí inicializační hodnoty hned při navázání komunikace. Autentizace je založena na sdíleném tajemství. Princip CHAP:

1. Server je iniciátorem spojení. Pošle klientovi výzvu (Challenge), jedná se o nějakou náhodnou hodnotu serveru.

2. Klient přijme zprávu (výzvu) a zprávu zahešuje a pošle zpět serveru.

3. Server porovná získanou hodnotu s uloženou hodnotou a na základě výsledku spojení povolí, nebo zakáže.

4. Server v pravidelných intervalech opakuje žádost o autentizaci. Každá výzva musí bý unikátní.

Bezpečnost CHAP je vyšší než u PAP, protože nepřenáší hesla v otevřené podobě a pravidelně klienta autentizuje. Nevýhodou je, že server ukládá heslo v otevřené podobě. CHAP je odolný vůči replay útokům, díky své recyklaci údajů.

Vylepšenou variantou CHAP je MS-CHAP (Microsoft Challenge CHAP), v současnosti je verzi V2. Více informací [53] [85]. První verze MS-CHAP-V1 (1998) byla podobná CHAP a umožňovala změnu hesla. MS-CHAP-V1 používá ke kódování a změně hesla LAN Manager. Princip MS-CHAP-V1:

1. Klient požádá o výzvu serveru.

2. Server pošle klientovi náhodnou výzvu, která je složena z identifikátoru relace a individuálního řetězce výzvy.

3. Klient si nejprve pomocí LAN-Manager-hash vytvoří tři hodnoty z výzvy, identifikátoru relace a hesla. Hodnoty jsou použity pro šifrování a spolu s uživatelským jménem je odešle na server.

4. Server hodnoty dešifruje a ověří je a následně povolí (zakáže) komunikaci.

Z bezpečnostního hlediska poskytuje MS-CHAP-V1 slabé šifrování (šifra DES). Dále je možné pouze jednostranné ověřování. Klient nemá šanci zjistit zda se spojuje se serverem a zda se nejedná o podvržené spojení.

MS-CHAP-V2 odstraňuje nedostatky předchozí verze. Kryptografický klíč je generován nejen z hesla zadaného uživatelem, ale i z individuálního řetězce výzvy. Navíc narozdíl od předchozí verze používá pro každý směr komunikace jiný klíč. Autentizace probíhá v obou směrech. Bruce Schneier a kol. dokázal, že bezpečnost závisí na složitosti zadaného hesla [86]. Pokud je provedena dostatečná kontrola síly hesla, lze považovat MS-CHAP-V2 za bezpečný.

10.2.3. Extensible Authentication Protocol

Extensible Authentication Protocol (EAP) zahrnuje širokou skupinu autentizačních protokolů. EAP vznikl jako nástupce PAP a CHAP a lze ho využít i mimo PPP, protože není použit ve fázi LCP (Link Control Protocol), pouze je

v této fázi smluvena jeho aplikace. Podporuje řadu autentizačních metod jako jsou jednorázová hesla, generické tokeny nebo MD5-challenge. Více informací k definici EAP a jeho variant [87] [88].

Obecné komunikační schéma EAP probíhá ve fvou fázích: nejprve se komunikující strany dohodnou na metodě ověření a ve druhé fází probíhá samotná autentizace. Obě strany provádí autentizaci protistrany. EAP podporuje více autentizačních mechanismů. Princip EAP (RFC 3748):

- Komunikace probíhá mezi klientem a přístupovým bodem.
- Přístupový bod zvolí typ autentizace a pošle klientovi žádost o autentizaci.
- Pokud klient souhlasí s komunikací, pošle přístupovému bodu kladnou odpověď na žádost.
- Nyní pošle bod klientovi výzvu.
- Klient odešle odpověď na výzvu.
- Bod potvrdí resp. odmítne klienta.

Protected Extensible Authentication Protocol (PEAP) je jednou z nejrozšířenějších alternativ EAP, která měla odstranit jeho nedostatky. PEAP je společným protokolem společností RSA, Cisco a Microsoftu. EAP předpokládá použití na zabezpečeném kanálu např. pomocí fyzického zabezpečení. PEAP zapouzdřuje EAP protokol pro šifrování a autentizaci uvnitř TLS (Transport Level Security) kanálu.

Proces ověřování pomocí protokolu PEAP probíhá ve dvou fázích:

1. pomocí protokolu PEAP se vytvoří zabezpečený kanál mezi klientem a serverem.
2. oveřování mezi klientem a serverem je realizován pomocí EAP.

Nejrozšířenější verzí PEAP je PEAPv0, protože je nejvíce podporovaným protokolem. Vzhledem k nejednotnosti následujících verzí existuje PEAP v několika variantách s protokolem EAP např. PEAPv0 s EAP-MSCHAPv2 (Microsoft) nebo PEAPv1 s EAP-GTC (Cisco). Hlavním rozdílem mezi EAP a PEAP je vytvoření šifrovaného kanálu pro druhý ověřovací algoritmus EAP. PEAP navíc podporuje chráněné hlavičky výstupu autentizace pro fáze zamítnutí resp. pro povolení spojení.

PEAP lze použít jako metodu ověření klientských počítačů v bezdrátové síti, ale nehodí se však u klientů virtuální privátní sítě (VPN) nebo jiných klientů vzdáleného přístupu. Vytvoření šifrovaného kanálu mezi klientem a serverem je

realizované pomocí PKI certifikátů, které poskytují klientovi šifrování s veřejnými klíči. Některé implementace PEAP jsou náchylné na man-in-the-middle útoky, protože může nastat přeskočení fáze ověřování důvěryhodného serveru třetí strany a tím dochází k omezení bezpečnosti zadávaného hesla.

Lightweight Extensible Authentication Protocol (LEAP) je další rozšířenou variantou EAP, která je proprietárním autentizačním mechanismem pro bezdrátové sítě. Protokol byl vyvinut společností Cisco a původní varianta byla modifikací MS-CHAPv2. Hlavní výhodou LEAP je jeho pravidelné ověřování identity klientů, kdy dostane klient po každém úspěšném ověření nový WEP klíč. Životnost klíčů je omezená z důvodu potenciálního prolomení. Protokol je součástí zařízení firmy Cisco a také je definována v kompatibilních zařízeních jako je například WLAN.

U protokolu LEAP byly prokázány bezpečnostní nedostatky podobně jako u WEP. V roce 2003 (Wright, nástroj Asleap [89]) byla uveřejněna úspěšná metoda prolomení hesla v offline režimu. Metoda pracovala na principu slovníkového útoku, který prolamoval odchycené pakety hesla. Autor útoku v roce 2007 upravil metodu také pro MS-CHAPv2. Doporučené řešení problému je vynucená síla hesla na straně klienta, nebo nahrazení LEAP novějším protokolem Cisco EAP-FAST, nebo PEAP. Vzhledem k existenci efektivních nástrojů pro slovníkové útoky je použití LEAP nevhodné.

Jako řešení problémů s LEAP vznikl EAP-FAST, který měl odstranit nedostatky LEAP, ale zachovat efektivní provedení LEAP. EAP-FAST je založen na PAC (Protected Access Credential), pomocí nějž vytváří zabezpečený tunel TLS. EAP-FAST probíhá ve třech fázích:

1. první fáze je volitelná, zde se dynamicky nebo manuálně inicializuje PAC. Existuje řada implementací inicializace PAC, například probíhá na straně serveru pomocí údajů klienta.

2. v druhé fázi je vytvořen TLS tunel dle hodnot PAC.

3. třetí fáze je určena pro autentizaci klienta v šifrovaném kanálu.

Použití PAC má několik nevýhod. Pokud útočník zachytí PAC, pak jej může zneužít k autentizaci uživatele. Řešením je manuální vydávání PAC v první fázi protokolu. Problémem je také nevhodné přidělování PAC. Například pokud útočník odcizí PAC klienta a v síti se objeví dva body se stejným identifikátorem, server autentizaci odmítne. Při připojení nového klienta do sítě musí tento bod dostat PAC jako první, proto není možné provozovat EAP-FAST v anonymních sítích, kde nejsou klienti známí.

Mezi další varianty EAP patří například **EAP-TLS**, který je standardem podporovaným většinou bezdrátových sítí. Bezpečnost EAP-TLS je založena na

schopnosti detekovat falešné výzvy o autentizaci. EAP-TLS je konstrukčně dostatečně bezpečný, ale používá pro komunikaci nějaký PKI certifikát, který může tvořit slabinu protokolu. Jedním z příkladů nevhodného nasazení je EAP-TLS s Cisco Secure ACS, který umožňoval přeskočení fáze autentizace. EAP-TLS je vhodné používat s metodami poskytujícími dostatečné zabezpečení privátního klíče (například čipové karty).

10.2.4. Kerberos

Kerberos je autentizační protokol, který je založen na přidělování tiketů klientům, které umožňují autentizaci mezi komunikujícími uzly na nezabezpečené síti. Kerberos je založen na symetrické kryptografii, ale dovoluje aplikaci asymetrické kryptografie pro některé fáze autentizace. Kerberos je protokolem typu klient-server a zajišťuje datovou integritu. Protokol je rozšířený v systémech Windows, ale existuje i ve volně šiřitelné verzi (MIT).

Kerberos vznikl v osmdesátých létech a během vývoje prošel několika verzemi. Například verze Kerberos verze 4 používala DES, která se prokázala jako nedostatečná. Verze 4 je náchylná vůči replay útokům, kdy je zneužita autentizace a časové razítko tiketu. Verze 5 přešla na 3DES s CBC módem. Obě verze mají implementační nedostatky jako je zneužití útočníkova vlastního kódu pomocí přetečení zásobníku. V současnosti existuje Kerberos ve verzi 5 V2 ve formátu GSS-API (The Kerberos Version 5 Generic Security Service Application Program Interface). Používá k šifrování AES a vedle šifrování obsahuje kontrolní součty dat. Více k definici protokolu Kebreros [90].

Princip autentizace protokolu Kerberos
Kerberos je protokol, který využívá více entit: klient, autentizační server (AS), tiket server (TGS) udělující časová razítka a servisní server (SS). Průběh autentizace začíná u klienta, který požádá AS o autentizaci. AS předá identifikátor KDS (Key Distribution Center), který identifikátor zašifruje a zajišťuje vytvoření tiketu (časového razítka) pomocí služby TGT (Ticket Granting Ticket). Komunikaci mezi uzly zajišťuje KDS pomocí zasílání TGT. Po ověření TGT je povolen klientovi přístup ke službě. Klient službu (SS) kontaktuje.

Podrobnější popis procesu autentizace pomocí protokolu Kerberos:

1. Klient na začátku zadává jméno a heslo, ze kterého je vypočten hash.
2. Klient pošle serveru AS žádost o autentizaci, která je složena pouze z identifikátoru. Klient neposílá AS heslo a to ani v šifrované podobě. Server AS vypočítá hash z hesla, které má uložené v databázi (Active directory).
3. Pokud server hash vypočítal (našel heslo v databázi), pošle klientovi:
   (a) klíč sezení (zpráva A), který je zašifrovaný privátním klíčem uživatele.

(b) tiket (zpráva B), který klienta identifikuje v síti. Tiket obsahuje ID klienta, síťovou adresu klienta a dobu platnosti. Tiket je zašifrován pomocí soukromého klíče TGS.

4. klient zprávu A dešifruje a získá tak svůj klíč sezení, který bude používat pro komunikaci s TGS. Zprávu B bude používat jako svůj tiket v síti.

Autentizace klienta při přístupu k službám serveru:

1. Pokud chce klient přistupovat ke službám serveru, musí poslat TGS dvě zprávy:

   (a) zprávu C, která je složena s identifikátoru služby a zprávy B, kterou získal od AS ve fázi autentizace.

   (b) zprávu D tzv. autentizátor, což je šifrovaná zpráva časového razítka s klíčem daným klientovým ID sezení.

2. TGS obdrží zprávy C a D. Získá zprávu B z C a dešifruje B pomocí příslušného uloženého klíče. Výsledek dešifrování udává klient/TGS klíč sezení. Zprávu D dešifruje pomocí klíče sezení. Nyní pošle klientovi dvě zprávy:

   (a) zprávu E: Klient-Server tiket, který je složen z identifikátoru klienta, síťové adresy klienta, doby platnosti tiketu a klient-server klíče sezení. Tiket je zašifrovaný pomocí tajného klíče TGS serveru.

   (b) Zprávu F: Klient-Server klíč, který je zašifrovaný pomocí klient/TGS klíče sezení.

Žádost o přístup k službám SS:

1. Klient se připojí k SS a posílá dvě zprávy:

   (a) zprávu E.

   (b) zprávu G, která je složena z identifikátoru služby a tiketu. Zpráva G je zašifrována pomocí Klient-Server klíče.

2. SS z G dešifruje tiket pomocí vlastního Klient-Server klíče. Dále s použitím klíče sezení dešifruje E a získá autentizátor klienta. Při úspěšném dešifrování hodnoty tiketu a autentizátoru odešle SS klientovi povrzení o úspěšné autentizaci a ochotě poskytovat spojení:

   (a) zpráva H, která je složena z časového razítka (odvozené z autentizátoru) + 1. Zpráva G je zašifrovaná pomocí Klient-Server klíče.

3. Klient si dešifruje zprávu G pomocí Klient-Server klíče a zkotroluje, zda je časové razítko správně aktualizované. V kladném případě může SS důvěřovat a začne žádat o jeho služby.

4. SS po dobu platnosti časového razítka poskytuje klientovi požadované služby.

11. Praktická část

11.1. Aplikace Bezpečné kryptografické algoritmy

Praktická ukázka vybraných kryptografických algoritmů je realizována aplikací Bezpečné kryptografické algoritmy. Aplikace byla iplementována v jazyce C# prostřednictvím vývojového nástroje Microsoft Visual Studio 2010 .NET Framework 4, MS Windows Forms.

Aplikaci lze formálně rozdělit do tří základních částí: blokové šifry, šifrování s veřejným klíčem, kryptografické hašovací funkce. Každá z částí obsahuje realizaci řady vybraných algoritmů.

11.1.1. Zadání a cíle aplikace

- Zadání diplomové práce nespecifikovalo konkrétní algoritmy.
- Algoritmy byly voleny s ohledem na bezpečnost, kterou v současnosti poskytují.
- Cílem aplikace byla praktická ukázka vybraných algoritmů, které jsou popsány v textu diplomové práce.
- Aplikace nemá za úkol nahradit stávající kryptografické knihovny jako je například .NET Framework 4 System.Security.Cryptography Namespace.
- Algoritmy byly implementovány dle doporučených bezpečnostních postupů.
- Algoritmy jsou implemetovány dle původních návrhů bez dalších optimalizací. Jednou z optimalizací je například použití čtyř vyhledávacích tabulek u AES [49].

11.1.2. Blokové šifry

Praktická ukázka vybraných algoritmů je demonstrována ve formuláři Blokové šifry. Formulář umožňuje nastavení vybrané šifry, jejího klíče a módu blokové šifry. Vedle šifrování resp. dešifrování lze vygenerovat náhodný klíč požadované délky. Zdrojové kódy jednotlivých algoritmů se nachází v projektu BlockCiphers. Třídy jsou doplněny komentáři pro pochopení jednotlivých funkcí.

Nastavení blokových šifer:

- Doplnění bloku zprávy (vycpávka) je realizováno doplněním nulových bitů do velikosti bloku.

Šifra	Velikost bloku (bit)	Délka klíče (bit)	Počet rund
AES	128	128, 192, 256	10, 12, 14
BlowFish	64	32-448	16
IDEA	64	128	8,5
RC5	32, 64, 128	0-2040	0 - 255

Tabulka 9. Přehled implementovaných blokových šifer.

- Klíč je potřeba zadávat v hexadecimální podobě. Velikost klíče musí odpovídat požadované délce klíče blokové šifry, klíč si můžete vygenerovat. Více informací k velikosti klíčů vybraných blokových šifer v tabulce 9..
- Otevřený text lze zadávat jako prostý text, nebo v hexadecimální podobě.
- Šifrovaný text je vrácen v hexadecimálné podobě.

Aplikace nabízí následující blokové šifry:

1. **AES (Advanced Encryption Standard).** Formální popis viz. 5.3.4.. Základní údaje o velikosti klíčů, počtu rund a velikosti bloku najdete v tabulce 9.. Patentováno U.S. FIPS PUB 197. AES byla otestována pro všechny tři velikosti klíčů a výsledky byly porovnány s údaji třídy AES .NET Framework 4 (System.Security.Cryptography.Aes).
2. **BlowFish.** Formální popis viz. 5.3.4.. Základní údaje o velikosti klíče, počtu rund a velikosti bloku najdete v tabulce 9.. BlowFish není patentována, více informací na stránkách autora [44]. BlowFish byla otestována pomocí standardního testovacího souboru šifry DES, který doporučuje i Bruce Schneier.
3. **IDEA (International Data Encryption Algorithm).** Formální popis viz. 5.3.4.. Základní údaje o velikosti klíčů, počtu rund a velikosti bloku najdete v tabulce 9.. Patentováno společností MediaCrypt, pro nekomerční použití volně k dispozici. IDEA byla otestována v CBC módu.
4. **RC5-32/20/16.** Formální popis viz. 5.3.4.. Počet rund byl volen dle bezpečnostních doporučení, verze RC5-32/12/16 s dvanácti rundami není považována za dostatečně bezpečnou. Základní údaje o velikosti klíče, počtu rund a velikosti bloku najdete v tabulce 9.. Patentováno společností RSA Security. RC5 byla otestována ve verzi RC5-32/12/16.

11.1.3. Šifrování s veřejným klíčem

Z asymetrických šifrovacích systémů byly vybrány šifrovací algoritmy s veřejným klíčem. Algoritmy jsou realizovány ve formuláři Šifry s veřejným klíčem. Zdrojové kódy lze nalézt v projektu PublicKeyCiphers. Aplikace umožňuje generovat prvočísla podle zadané velikosti v bitech. Testování prvočíselnosti probíhá pomocí Miller-Rabin testu prvočíselnosti. Aplikace nabízí vygenerování veřejného a soukromého klíče a šifrování resp. dešifrování zadaného textu.

Šifra	Max. velikost bloku (bit)	Délka klíče (bit)	Počet rund
Rabin	délka klíče - 1	512 to 4096	1
RSA	délka klíče - 1	512 to 4096	1

Tabulka 10. Přehled implementovaných šifer s veřejným klíčem.

Nastavení šifrování s veřejným klíčem:

- Generování veřejného resp. tajného klíče je závislé na vybrané velikosti klíče. Pokud použijete rozdílné velikosti klíčů, získáte špatné výsledky.
- Generování prvočísel je časově náročná operace. Pro test prvočíslenosti je doporučeno používat menší velikosti klíčů. Miller-Rabin test provočíselnosti je rychlá deterministická metoda, ale samotný výběr testovaného čísla není nijak optimalizován. Pro výběr náhodného čísla je použita standardní metoda Random() resp. RNGCryptoServiceProvider(). Jako optimalizaci by se dala použít nějaká metoda generování kandidátů na prvočíslo.
- Veřejný exponet e u RSA je nastaven na hodnotu 0x10001 = 65,537, což je doporučovaná hodnota pro zajištění dostatečné bezpečnosti.
- V případě šifry Rabin je demonstrována jeho hlavní nevýhoda. Při dešifrování vrací Rabin čtyři možné výsledky.

Aplikace nabízí následující šifry s veřejným klíčem:

- **Rabin kryptosystém.** Formální popis viz. 6.5.. Základní údaje o velikosti klíčů, počtu rund a maximální velikosti bloku najdete v tabulce 10.. Rabin není patentován. Pro demostraci byla vybrána nezobecněná varianta s veřejným exponentem 2. Rabin kryptosystém byl zvolen z důvodu demostrace jeho hlavní nevýhody.
- **RSA.** Formální popis viz. 6.6.. Základní údaje o velikosti klíčů, počtu rund a maximální velikosti bloku najdete v tabulce 10.. RSA byl patentovaný (U.S. Patent 4,405,829), patent vypršel v roce 2000, nyní je volně k použití.

11.1.4. Kryptografické hašovací funkce

Kryptografické hašovací funkce jsou realizovány ve formuláři Kryptografické hashe. Aplikace umožňuje vytvoření hashe ze zadaného textu. Kryptografické hašovací funkce jsou vybrány pro docílení bezpečnostních předpokladů a dle snahy popsat zástupce jednotlivých rodin hashů. Proto nebyly implementovány pouze hashovací funkce z rodiny SHA-2, i když jsou v současnosti nejrozšířenější a nejbezpečnější. Zdrojové kódy lze nalézt v projektu CryptographicHashFunctions.

Nastavení kryptografických funkcí:

- Aplikace je z hlediska nastavení velmi jednoduchá. Výběr kryptografické hašovací funkce je realizován pomocí combo boxu.
- Hash vytvoříte pomocí stejnojmenného tlačítka.

Algoritmus	Délka hashe (bit)	Velikost bloku (bit)	Počet rund
RIPEMD-160	160	512	80
SHA-1	160	512	80
SHA-256	256	512	64

Tabulka 11. Přehled implementovaných kryptografických hašovacích funkcí.

Aplikace nabízí následující kryptografické hašovací funkce:

- **RIPEMD-160.** Formální popis viz. 7.3.6.. Základní údaje o délce hashe, počtu rund a velikosti bloku najdete v tabulce 11.. RIPEMD-160 není omezen žádným patentem. RIPEMD-160 byl vybrán z důvodu jeho oblíbenosti.
- **SHA-1.** Formální popis viz. 7.3.7.. Základní údaje o délce hashe, počtu rund a velikosti bloku najdete v tabulce 11.. SHA-1 byla vybrána z důvodu její vysoké oblíbenosti z minulosti, i když je z bezpečnostního hlediska v současnosti její použití nedoporučované. SHA-1 nebyla doposud prolomena, ale je náchylná vůči kolizím.
- **SHA-256.** Formální popis viz. 7.3.8.. Základní údaje o délce hashe, počtu rund a velikosti bloku najdete v tabulce 11.. SHA-256 patří do rodiny SHA2 a byla volena vzhledem k popisu algoritmu v textu diplomové práce.

11.2. Bezpečné použití RSA.

RSA je jedna z nejpoužívanějších asymetrických šifer, jedním z praktických příkladů jejího využití je šifrování klíčů v protokolu SSL. Pokud se při šifrování použije dostatečně velký klíč, je odolná vůči kryptoanalytickým útokům, protože doposud neexistuje úspěšná metoda pro faktorizaci prvočísel. Šifra je tedy bezpečná, ale problémem může být nevhodný způsob použití a realizace. Existuje několik praktických případů dokazujících, jak může být nesprávné použití bezpečné šifry slabinou. V následujících kapitolách si popíšeme prokázané kryptoanalytické útoky na protokol SSL.

11.2.1. Bleichenbacherův útok bočním kanálem

Bleichenbacherův útok je adaptivní útok volbou šifrovaných textů vůči protokolům založeným na algoritmu RSA ve standardu PKCS#1. Adaptivní útoky používají výstupy z předchozích útoků, v případě Bleichenbacherova útoku je základem zachycený šifrovaný text c. Útok je založen na základním nedostatku PKCS#1, který byl v pozdějších verzích ošetřen, a to multiplikativní vlastnosti RSA, kterou můžeme symbolicky zapsat jako $RSA(a * b) = RSA(a) * RSA(b)$. Standard PKCS#1 umožňuje útočníkovi odhalit konstrukci zprávy, která je přijatelná serverem a díky které je schopen v konečném důsledku odhalit jednak původní zprávu m tak i použitý klíč. Bleichenbacherův útok popsaný v roce 1998 je možno nalézt pod názvem EME-PKCS1-v1_5 určujícím útok na verzi PKCS#1. Bleichenbacher používal pro testy svého útoku SSL protokol ve verzi 3.0.

Obsah standardu PKCS#1.
PKCS#1 je standard, který definoval operaci zašifrování a dešifrování pomocí algoritmu RSA. Obstarával přípravu a formát vstupních i výstupních dat pro algoritmus RSA. Byl stanoven pro bezpečnotní protokol SSL a v první použitelné verzi PKCS#1.5 byl používán od roku 1993 až do nahrazení verzí PKCS#2. PKCS#1 i jeho pozdější verze stanovovaly formu doplnění dat do plného bloku RSA, protože šifrovací klíče i hashe pro elektronický podpis tvořili pouze malou část bloku RSA. Dále také popisoval způsob převodu bitového řetězce na čísla, aby se s nimi mohla provést operace $c^d \mod n$, ale také jak výsledné číslo převést zpět na bitový řetězec podle normy ASN.I.

V tabulce viz. 12. jsou popsány základní označení a symboly stanovené dle PKCS#1. Standard používal oktety (osmibitové řetězce), které se převáděly na čísla a zpět. Číslo se při konverzi na řetězec oktetů doplní zleva nulovými bity, tak aby mělo binární vyjádření zarovnané na oktety. Procedura má označení I2OSP (Integer-to-Octet-String Primitive). Osmice bitů se převedou na číslo tak, že jsou oktety nejvíce vlevo chápány jako bajty s nejvyšší vahou. Opačná procedura má označení OS2IP (Octet-String-to-Integer Primitive).

Příprava dat dle verze PKCS#1.5.
Nechť jsou n, e veřejné klíče algoritmu RSA a p, q, d jsou korenspodující tajné

Symbol	Význam symbolu
xy	vybraný oktet, osmibitový řetězec vyjádřený hexadeximálně
X ‖ Y	zřetězení X a Y
EB	doplněný blok, řetězec oktetů (Encryption Block) připravený ke konverzi na číslo zašifrované pomocí algoritmu RSA
ED	výsledný oktetový řetězec (Encrypted Data), výsledek konverze šifrovaného výstupu RSA
BT	typ bloku (Block Type), nabývá hodnoty 01 nebo 02
PS	doplňující řetězec (Padding String)
D	bitový řetězec dat (Data)
M	originální zpráva určená k podpisu (Message)
MD	hašovací kód (Message Digets)

Tabulka 12. Symboly a označení dle standardu PKCS#1. Řetězce bitů a oktety jsou označeny velkým písmenem, čísla a písmena jsou označeny malým písmem.

klíče. Platí $n = pq$ a $d = e^{-1}(\mod \varphi(n))$. Vstupem RSA byla data D, která měla nejčastěji délku 40 oktetů a jednalo se o klíče nebo hashe. Bylo je nutné doplnit na k oktetů a to do velikosti bloku modulu RSA. Doplněný blok EB je definován jako řetězec k oktetů:

$$EB = 00||BT||PS||00||D.$$

Před data D se umístil separátor ve formě oktetu 00, dále doplňující řetězec několika oktetů označený PS, dále jeden oktet BT určující typ bloku a začátek EB tvořil vedoucí oktet 00. Vedoucí oktet pomáhá při převodu EB na číslo, protože nejvýznamější bajt je pak nulový. Výsledné číslo je menší než modul RSA, což zaručuje správnost při dešifrování. Délka doplňující řetězce PS byla volena tak, aby EB dosáhl požadované délky k oktetů. Doplňující řetězec obsahoval náhodně volené bity, které byly nenulové a jeho délku lze vyjádřit vztahem $P = k-3-|D|$. Vzhledem k tomu, že D nesmělo překročit hodnotu $k - 11$, minimální délka PS byla osm oktetů. Typ bloku BT nabývá hodnoty 01 pro podpis dat nebo 02 pro šifrování klíčů.

V případě bloku s typem 01 byl PS tvořen stejnými oktety s hodnotou FF. Data D vystupovala ve formě hašovacího kódu MD zprávy M a byla doplněna konstantním identifikačním řetězcem (dle ASN.I DigestAlgorithmIdentifier), jehož hodnota byla odvozena podle typu hašovací funkce (např. MD5).
V případě bloku s typem BT 02 byl volen PS ve formě nenulových náhodných oktetů. Náhodné doplnění zaručuje jiné šifrované obrazy pro stejná data D. Podmínka pro PS se jeví silná, ale právě volba náhodných dat vede k možnosti luštění a umožňuje odhalení šifrovacích klíčů určených pro šifrování přenášených dat.

U protokolu SSL si klient na začátku sezení vygeneruje náhodný klíč, zašifruje jej pomocí algoritmu RSA a pošle jej serveru. Obě strany potom komunikují pomocí symetrické šifry. Pokud je tedy dán například 128 bitový klíč D (d = 16 oktetů) a modul RSA má 1024 bitů (k = 128 oktetů), pak je potřeba doplnit blok D do plného 128 oktetového bloku EB a to následujícím způsobem: $EB = 00||02||PS||00||D$, kde $PS = 128 - 3 - 16$.

Šifrování a digitální podpis dle PKCS#1.5.

Odesílatel:

1. připravený EB se převede pomocí OS2IP na číslo.
2. aplikuje se algoritmus RSA s tajným nebo veřejným klíčem.
3. číslo se převede pomocí I2OSP na výsledný oktetový řetězec ED (Encrypted Data).

Příjemce:

1. na doručený oktetový řetězec se zavolá OS2IP.
2. aplikuje se algoritmus RSA s odpovídajícím párovým klíčem.
3. číslo se převede pomocí I2OSP na oktetový řetězec.
4. z výstupu se zjistí zda doručený formát dat souhlasí s formátem předpokládaného typu bloku.
5. kontroluje se nulový vedoucí oktet, vlastnosti PS, existence separátoru, délka dat D.
6. pokud se ověřuje podpis, kontroluje se identifikátor hašovacího algoritmu, ale také dojde k porovnání MD s hašem získaným z doručených podepsaných dat MD'.

Pokud příjemce obdrží šifrovaný blok, musí jej odšifrovat a ověřit zda má získaný EB' typ 02. Musí totiž z bloku zjistit klíč obsažený v části D, který pak obě strany používají pro komunikaci. Proto ověřuje zda jsou první dva oktety 00 a 02, dále zda následuje alespoň osm nenulových oktetů (část PS) a za nimi separátor 00. Pokud je vše splněno, pak lze blok akceptovat a pokračovat v komunikaci. V opačném případě vrátí chybové hlášení, což je podstatou útoku uvedeného níže.

Princip Bleichenbacherova útoku

Dříve než přejdeme k popisu útoku je potřeba zavést definici PKCS vyhovující zprávy.

Definice (PKCS vyhovující otevřený text).
Nechť je dán otevřený text $P = \sum_{i=1}^{k}(P_i * 256^{k-i})$, $0 \leq P_i \leq 255$, kde P_1 je nejvýznamější bajt otevřeného textu. Řekneme, že P je PKCS vyhovující (přijatelný) splňuje-li podmínky:

1. $P_1 = 0$
2. $P_2 = 2$
3. $P_j \neq 0$ pro všechny $j \in < 3, 10 >$
4. $\exists j$, $j \in < 11, k >$, $P_j = 0$; řetězec $P_{j+1} \| \ldots \| P_k$ je nazýván zprávou M, nebo daty (data payload).

Šifrovaný text c nazveme PKCS vyhovující, jestliže jeho dešifrováním získáme PKCS vyhovující otevřený text.

PKCS vyhovující zpráva vlastně určuje tvar akceptovatelné zprávy ze strany serveru. Bleichenbacher navrhl svůj útok pro PKCS#1, později si u útoku Klíma-Pokorný-Rosa ukážeme jak byl modifikován Bleichenbacherův útok pro PKCS#1.5. Bleichenbacherův útok není použitelný pro PKCS#1.5 (protokoly SSL 3.0 a vyšší), protože zde již byla navržena ochrana vůči útoku ve formě kontroly čísla protokolu. Pokud nedošlo ke korektnímu ověření čísla protokolu, server pošle chybovou zprávu a komunikaci ukončí. Více podrobností viz. 11.2.2..

Nyní přejdeme k samotnému popisu Bleichenbacherova útoku. Symbolem m označíme zprávu, kterou bude odesílatel šifrovat a symbolem c označíme šifrovaný obraz. Předpokládejme, že útočník naslouchá v komunikačnímu kanálu, zachytí šifrovaný blok c a pokračuje v odebírání přenášených dat. Útočník počká až komunikace skončí a začne serveru posílat modifikované šifrované bloky c_i. Z přijatých chybových hlášení získá dostatek informací, aby mohl určit zprávu m a celkově odvodit klíč odposlouchávané komunikace.

Z krytologického hlediska je modifikace c_i vyjádřena vztahem $c_i = c(r_i)^e \, (mod\, n)$, kde c je původní zachycený šifrovaný blok a r_i je volitelná konstanta. Příjemce zprávu odšifruje a zjistí zda je typu 02, pokud není, vrátí chybové hlášení. Po určité době se útočníkovi podaří zvolit konstantu r_i takovou, že chybové hlášení nepřijde. Znamená to, že se na straně příjemce úspěšně dešifrovala zpráva $m(i)$, která má požadovaný formát 02. Útočník nyní ví, že $(m * r_i) \, (mod\, n)$ začíná 00 02 a že tedy zná 2 bajty. Platí totiž:

$$m_i = c_i^d \, mod(n) = (c_i * r_i^e)^d \, mod(n) = (c^d \, mod(n)) * (r_i^{ed} \, mod(n)) = (m * r_i) \, mod(n).$$

Pomocí k oktetového čísla označeného symbolem $B = 00\,02 \ldots 00$ se vymezuje interval $2B \leq m * r(i)(\mod n) < 3B$, který vede k zúžení intervalu m, kde $0 < m < n$, na řadu menších intervalů. Opakovaným útokem pro jiné c_j, které dávají nové nerovnosti a nové zúžení intervalu. Po určitém počtu kroků lze zcela

odvodit m. Bleichenbacher dokázal, že je potřeba asi 2^{20} modifikovaných c_i pro 512 a 1024 bitový RSA modul. Průměrně se požadovaný počet modifikovaných šifrovaných textů pohyboval od 300 000 po 2 000 000. Útok se tedy skládá ze tří fází. V první fázi získá útočník šifrovaný text c_0, který korenspoduje s neznámou zprávou m_0. V druhé fázi je nutno zvolit nejmenší možnou konstantu r_i, která zaručí úspěšné přijetí modifikovaného šifrovaného textu na straně serveru. Výsledek vymezí intervaly, ve které se nachází původní zpráva m_0. Třetí fáze začne až zbyde pouze jeden možný interval. Nyní má útočník dostatečné množství informací o původním textu m_0 a může zvolit konstantu r_i takovou, že $c_i = (c_0 * (r_i)^e)\,(mod\, n)$ vyhovuje lépe než náhodně volené zprávy. Konstanta je postupně zvyšována, až dojde k nalezení jediné možné hodnoty pro původní zprávu m_0.

Praktická realizace útoku.
Předpokládejme, že chce útočník nalézt text $m = c^d\,(mod\, n)$, kde c je známé celé číslo. Nechť M_i je množina uzavřených intervalů, které byly vypočítány pro aktuální hodnotu konstanty r_i, která zaručí úspěšné přijetí modifikovaného šifrovaného textu, hovoříme o PKCS vyhovujícím textu viz. 11.2.1.. Zpráva m_0 se nachází v jednom z intervalů M_i. Dále je známo číslo $B = 2^{8(k-2)}$, kde k je délka B v bajtech.

1. **Nalezení r_0.**
   Nechť je r_0 náhodně zvolené pozitivní celé číslo. Dokud není $c(r_0)^e$ PKCS vyhovující, zvol jiné r_0 a testování opakuj. Při úspěchu nastav:

$$\begin{aligned} c_0 &= c(r_0)^e\,(mod\, n) \\ M_0 &= [2B, 3B-1] \\ i &= 1 \end{aligned}$$

2. **Hledání PKCS přijatelných zpráv.**

   (a) **Zahájení hledání.**
   Jestliže $i = 1$, pak hledej nejmenší pozitivní celé číslo r_1, pro které platí $r_1 \geq n/(3B)$, a pro které je šifrovaný text $c_0(r_1)^e\,(mod\, n)$ PKCS vyhovující.

   (b) **Zbývá-li více intervalů.**
   Jestliže je $i > 0$ a zároveň existují nejméně dva intervaly v M_{i-1}, pak hledej nejmenší pozititivní celé číslo $r_i > r_{i-1}$ takové, že je šifrovaný text $c_0(r_i)^e\,(mod\, n)$ PKCS vyhovující.

   (c) **Zbývá-li pouze jeden interval.**
   Jestliže obsahuje množina M_{i-1} pouze jeden interval, tedy $M_{i-1} = [a, b]$, opakovaně vol nejmenší pozitivní celá čísla r_i a t_i takové, že platí:

$$t_i \geq 2\frac{br_{i-1} - 2B}{n} \tag{3}$$

$$\frac{2B + t_i n}{b} \leq r_i \leq \frac{3B + t_i n}{a} \tag{4}$$

než je text $c_0(r_i)^e$ PKCS vyhovující.

3. **Zúžení množiny řešení.**
   Po nalezení r_i vymezíme množinu M_i podle následujícího vztahu:

$$M_i = \bigcup_{(a,b,t)} \left\{ \left\lceil max\left(a, \frac{2B + tn}{r_i}\right), min\left(b, \frac{3B - 1 + tn}{r_i}\right) \right\rceil \right\} \tag{5}$$

   pro všechny $[a, b] \in M_{i-1}$ a

$$\frac{ar_i - 3B + 1}{n} \leq t \leq \frac{br_i - 2B}{n}.$$

4. **Výpočet řešení.**
   Jestliže obsahuje M_i jediný interval jehož délka je rovna jedné, tedy platí $M_i = [a, a]$, potom polož $m = a(r_0)^{-1} \, (mod\, n)$ a vrať m jako řešené zadané rovnice $m = c^d \, (mod\, n)$. V opačném případě polož $i = i + 1$ a přejdi na začátek kroku 2.

Poznámka.

- Krok 1 může být vynechán je-li c PKCS vyhovující (jedná se o ukradenou šifrovanou zprávu). V tomto případě položíme $r_0 = 1$. Krok 1 se nutné spočítat vždy když je potřeba ověřit podpis zprávy, nebo když nechceme použít předen získanou zprávu.
- V kroku 2.a se začíná na hodnotě $r_1 \geq n/(3B)$, protože pro menší hodnoty není $m_0 r_1$ PKCS vyhovující.
- Podmínka 2c se používá, protože je potřeba v každé iterací rozdělit interval na půl.
- Útok může být modifikován z doplňujích informací. Pokud je například zpráva $m_0 r_i$ použita pro vygenerování klíčů pro sezení, umožní to útočníkovi lepší dohledání z druhotných informací.

Navrhované protiopatření

Úspěšnost útoku spočívá ve vysoké pravděpodobnosti, se kterou se útočník trefuje do požadovaného tvaru. Server totiž kontroluje po dešifrování pouze dva první bajty. Jedno z navrhovaných řešení je kontrola většího počtu bajtů po dešifrování doručeného bloku. Úspěšnost správného výběru konstanty se pak rapidně snižuje. Může se například kontrolovat existenci a správné umístění separátoru 00 oddělujícího D, protože je jeho umístění pro příjemce díky SSL známé. Server totiž ví

jakou délku má mít klíč a tedy i kde je přesně separátor umístěný. Navíc některé verze SSL obsahovali nadbytečné informace o verzi protokolu SSL, které se předávali v rámci dat D. Pokud by se po doručení kontrolovalo i číslo verze, útočník musel podle Bleichenbachera vyzkoušet až bilion c_i. Bleichenbacher navrhoval další protiopatření v podobě sjednocení chybových hlášení, protože bylo možné odlišit různé typy chybových zpráv, což umožňovalo útočníkovi lepší orientaci v nastalé situaci.

11.2.2. Klíma-Pokorný-Rosa útok bočním kanálem

V roce 2003 přišla skupina českých kryptologů pod vedením Vlastimila Klímy s modifikovaným Bleichenbacherovým útokem na session protokolů SSL/TSL založených na algoritmu RSA (Attacking RSA-based Sessions in SSL/TLS). Útok byl zaměřen na protokoly na bázi standardu PKCS#1.5. Podstata útoku těží z manipulace s tajnou premaster-secret hodnotou (viz. 11.2.2.), která byla šifrována algoritmem RSA a která se využívala k odvození tajným klíčů pro session.

Oba útoky jsou založeny na velmi známém tvrzení, které můžeme očekávat i u ostatních kvalitních cifer:

> Uniká-li z dešifrátoru RSA informace byť jen o jediném bitu odšifrované hodnoty m, potom může útočník pro libovolný šifrovaný text vyluštit celou hodnotu otevřeného textu.

Útočník může při znalosti premaster-secret hodnoty dešifrovat zachycenou session SSL/TSL. Jedna z modifikací PKCS#1.5 totiž ověřovala verzi protokolu v otevřeném textu, ale tím nechanismus poskytl vytvoření bočního kanálu, z jehož informací se dalo invertovat RSA šifrování. Pro demostraci principu útoku navrhli autoři tzv. bad-version oracle (BVO). Zároveň přišly s několika úpravami původního Bleichenbacherova algoritmu, které zrychlily dobu původního útoku.

Když se objevil Bleichenbacherův útok, návrháři standardu doporučovali další kontrolu prvků specifických pro SSL/TSL protokoly, jakým byla například verze použitého protokolu pro komunikaci. Verze protokolu byla tvořena dvěma nejlevějšími bajty premaster-secret hodnoty. Jednalo se však o další protiopatření, které otevřelo možnost útoku bočním kanálem. Nebylo totiž bezpečně specifikováno jak se má server zachovat při nesplnění podmínky a jaké chybové zprávy odeslat a zda komunikaci ukončit. Autoři útoku uvádí, že byly k útoku náchylné více jak 2/3 ze stovek náhodně vybraných serverů.

Bleichenbacherův útok ukázal jak se dá zneužít informace získáná bočním kanálem k prolomení konkrétní realizace RSA. Pomocí Bleichenbacherova útoku může útočník dešifrovat získaný šifrovaný text. Doporučovaná varianta PKCS#1.5 zamezující tomuto útoku se nazývá EME-OAEP metoda. Metoda neumožňuje útočníkovi odhalit podrobnosti o postupu dešifrování, které byly základem Bleichenbacherova útoku. V případě, že se nejedná o vyhovující zprávu

(S-PKCS vyhovující zprávu), komunikaci ukončí a zamezí tak dalším dotazům útočníka, které byly využity ke konkretizaci intervalu řešení. Bleichenbacher si byl této skutečnosti vědom, hovoří o ní v případě popisu možností jeho útoku na protokol SSL ve verzi 3.0. Zároveň navrhl možnost jak by se dalo ověřování obejít, což později ukázali čeští kryptologové v modifikaci jeho útoku.

TSL protokol byl základem SSL protokolu ve verzi 3.1. SSL protokol byl ve verzi 3.0 označován jako prostý (plain) SSL. Protokoly se lišili, ale pro útok byly rozdíly nepodstatné, proto SSL/TSL.

Bad-Version Oracle (BVO).
Pro BVO bylo potřeba definovat S-PKCS vyhovující otevřený RSA text. Doposud uvažovaná PKCS vyhovující podmínka zůstává stejná. Dříve zavedená definice viz. 11.2.1. určuje PKCS vyhovující zprávy pro PKCS#1. Vzhledem k tomu, že další verze PKCS#1.5 zavádí další specifikaci PKCS přijatelnosti a to S-PKCS přijatelnost, je potřeba definici rozšířit.

Definice (S-PKCS vyhovující otevřený text).
Řekneme, že P je S-PKCS vyhovující (přijatelný), jestliže je PKCS vyhovující a navíc splňuje-li následující podmínky:

1. $P_j \neq 0$ pro všechny $j \in < 3, k - 49 >$
2. $P_{48} = 0$

Šifrovaný text C nazveme S-PKCS vyhovující, jestliže jeho dešifrováním získáme S-PKCS vyhovující otevřený text.

S-PKCS přijatelnost tedy kontroluje existenci separátoru a přesně specifikuje délku dat (48 bajtů), jak bylo popsáno v části 11.2.1. v kapitole Bleichenbacherova útoku.

SSL/TSL protokoly představili speciální interpretaci prvních dvou bajtů P_{k-47} a P_{k-46} a to jako hlavní (major) a vedlejší (minor) čísla verze protokolu. Rozšíření mělo zabránit tzv. rollback útokům. *Premaster-secret hodnotou* jsou nazývána data (data payload), která lze vymezit jako $P_{k-47} \| P_{k-46} \| P_{k-45} \| \ldots \| P_k$. Vytvoření, výměnu a kontrolu hodnoty lze popsat v následujích krocích:

1. klient náhodně vygeneroval $P_{k-45} \| \ldots \| P_k$,
2. přidal hlavní a vedlejší číslo protokolu dle dohodnuté verze,
3. zašifroval celou premaster-secret hodnotu veřejným klíčem serveru a poslal zašifrovaný text C serveru,
4. server dešifroval přijatý text soukromým klíčem, vytvořil si vlastní kopii premaster-secret hodnoty a rozhodl o dalším postupu.

Proces popsaný výše je základem Handshake podprotokolu. Server nemusí po přijetí PKCS nevyhovující zprávy P informovat klienta. Ve skutečnosti si vytvoří novou premaster-secret hodnotu a to i pokud komunikace skončí zasláním Finish zprávy. Klient i server má rozdílnou premaster-secret hodnotu. Útočník vlastně ani není zda byla chyba v premaster-secret hodnotě nebo v nevyhovující PKCS zprávě.

Nechť server používá výše uvedenou ochranu vůči Bleichenbacherovu útoku a dále nechť server kontroluje všechny rozšíření podle následující podmínky.

Tvrzení (Přepokládané chování serveru)

1. Server kontroluje zda je přijatý otevřený text P S-PKCS vyhovující. Jestliže není, vytvoří si novou premaster-secret hodnotu a ukončí komunikaci jakmile klient pošle Finish zprávu.

2. Server kontroluje zda každý přijatý S-PKCS vyhovující text obsahuje správné hlavní a vedlejší čísla protokolu. Pokud čísla nesedí, odešle klientovi chybové hlášení.

Definice (Bad-Version Oracle (BVO).).
BVO je zobrazení $Z_N \to \{0,1\}$. $BVO(C) = 1$ právě tehdy když $C = P^e \,(mod\, N)$, kde e je veřejný klíč serveru, N je modulus serveru a P je S-PKCS vyhovující otevřený text takový, že buď $P_{k-47} \neq major$ nebo $P_{k-46} \neq minor$, kde major.minor jsou hlavní a vedlejší čísla protokolu očekávané serverem. V opačném případě je $BVO(C) = 0$.

Přepokládejme, že útočník zašle serveru šifrovaný text C. Jestliže server odešle chybovou zprávu (Tvrzení bod 2)), pak nastavíme $BVO(C) = 1$. BVO je tedy modelem serveru, který je náchylný na Klíma-Pokorný-Rosa útok bočním kanálem. Autoři zavedly model BVO, protože bylo potřeba definovat podmínky za kterých je server náchylný k útoku, protože obecně nebyl každý server BVO typem. Nyní si ukážeme jak využít definici BVO ke stanovení zda je text S-PKCS vyhovující.

Použití BVO.
Nechť známe pro BVO jeho veřejnou konstantu e, modulus N, dvojici major.minor a dále máme zašifrovaný text C pomocí RSA. Potom podmínka $BVO(C) = 1$. implikuje skutečnost, že $C = P^e \,(mod\, N)$, kde P je S-PKCS vyhovující otevřený text.

Pravděpodobnosti spojené s BVO.
Nechť $Pr(A) = B/A$ je pravděpodobnost jevu A, že podmínky 1) a 2) uvedené v definici PKCS vyhovujícího textu viz. 11.2.1. platí pro náhodně zvolený otevřený text. Nechť $Pr(S-PKCS|A)$ je podmíněná pravděpodobnost, že je prostý text P S-PKCS vyhovující a to za předpokladu, že pro P nastal jev A. Nechť

$Pr(BVO|S-PKCS)$ je podmíněná pravděpodobnost, že $BVO(P^e \, (mod\, N)) = 1$ s předpokladem, že je P S-PKCS vyhovující.

Autoři útoku využívali výše uvedených pravděpodobností k výpočtům složitosti útoku. Bližší informace viz. [16]. Uvedeme zde pouze základní informace, protože bude pravděpobnost využita u PT metody viz. 3. Pro $Pr(A)$ máme $256^{-2} < Pr(A) < 256^{-1}$ jak je popsáno v [15]. Pravděpobobnost $Pr(S-PKCS|A)$ lze vyjádřit jako $Pr(S-PKCS|A) = (255/256)^{(k-51)} * 256^{-1}$, kde délka nenulové PS je rovna $k-51$. BVO akceptuje jednu z variant čísla protokolu. Odtud, $Pr(BVO|S-PKCS) = 1-256^{-2}$. Hodnota $Pr(BVO|S-PKCS) * Pr(S-PKCS|A) * Pr(A)$ je pravděpodobnost, že náhodně zvolený šifrovaný text C splňuje $BVO(C) = 1$.

Princip Klíma-Pokorný-Rosa útoku.
Útok je modifikací Bleichenbacherova útoku, ale existují zde základní rozdíly spojené s S-PKCS vyhovující podmínkou a použitím BVO. V případě SSL/TSL je útok schopen odhalit jednak premaster-secret hodnotu pro získanou session, ale i digitální podpis serveru.

Autoři útoku používají pro zjednodušení následující značení: $E = 2B$, $F = 3B-1$, kde $B = 256^{k-2}$. Zavádí si je také z důvodu stanovení nových hranic E' a F'. Předpokládejme, že útočník analyzuje C_0, kde $BVO(C_0) = 0$, a chce zjistit tvar premaster-secret hodnoty. Předem ví, jaké jsou hlavní ($P_{0,k-47}$) a vedlejší ($P_{0,k-46}$) čísla protokolu pro nějaký S-PKCS P_0, kde $P_0 = C_0^d \, (mod\, N)$. Dále také ví přesnou pozici separátoru $P_{0,k-48} = 0$. V následujícím výčtu se zaměříme na základní modifikace, ve kterých se liší Klíma-Pokorný-Rosa útok od původního Bleichenbacherova útoku viz. 11.2.1.. V prvním bodě jsou zmíněny základní odlišnosti, které souvisejí s S-PKCS a BVO. V druhém bodě jsou uvedeny základní optimalizace, které autoři navrhli pro zlešení výpočtu algoritmu. Třetí bod popisuje paralelní metodu určenou pro výpočet subintervalů M_i.

1. **S-PKCS a BVO vlastnosti.**
   V případě Bleichenbacherova útoku se zavedly dvě hranice E a F, které vymezovaly PKCS vyhovující text $E \leq P \leq F$. Hodnoty se používaly v celém algoritmu inverze RSA. Protože v případě BVO pracujeme pouze s S-PKCS vyhovujícími texty, proto se zavádí upravené hranice, které omezují S-PKCS vyhovující text $E' \leq P \leq F'$. E' je stanoveno z minimální hodnoty doplňujícího řetězce PS a F' se počítá vzhledem k pevné pozici separátoru 00.

$$E' = 2B + 256^{k-3} + 256^{k-4} + \cdots + 256^{49} = 2B + 256^{49}(256^{k-51} - 1)/255.$$

$$\begin{aligned} F' &= 2B + 255(256^{k-3} + 256^{k-4} + ... + 256^{49}) + 0 \\ &\quad + 255(256^{47} + 256^{46} + ... + 256^{0}) = 3B - 255(256^{48}) - 1. \end{aligned}$$

Vzhledem k tomu, že útočník zná čísla verze protokolu a dále také zná pozici separátoru, navrhli autoři další zefektivnění algoritmu spočívající v úpravách hraničních hodnot $[a, b]$. Přesné stanovení hodnot a a b autoři neuvádí.

2. **Základní zobecněné optimalizace**.
Autoři útoku zavedli tzv. vhodný násobek r pro šifrovaný text C (celé číslo). Jeho použitím dostáváme stejně jako u Bleichenbachera $C' = C(r^e)\,(mod\,N) = (P')^e\,(mod\,N)$, kde P' je S-PKCS vyhovující otevřený text. Následující β-metoda byla navržena k zefektivnění hledání nejmenšího čísla r_i v bodě 2(b) Bleichenbacherova algoritmu. Z následujícího tvrzení plyne, že nejlepším vhodným násobkem, který slouží jako vstup dalšího kroku inkrementálního hledání případně PT metody, je takové $r_j < N/2$.

Lineární kombinace (β-metoda)
Nechť C_i a C_j jsou šifrované texty takové, že $C_i = (r_i)^e\,(mod\,N)$, $C_j = (r_j)^e\,(mod\,N)$, kde r_i a r_j jsou vhodné násobky C_0. Tj.

$$P_i = C_i^d\,(mod\,N) = 2B + 256^{49}PS_i + D_i$$

a

$$P_j = C_j^d\,(mod\,N) = 2B + 256^{49}PS_j + D_j,$$

kde $0 < PS_{i,j}$ a $0 \leq D_{i,j} < 256^{48}$. Pak pro C, kde $C = (r)^e\,(mod\,N)$, a $\beta \in \mathbb{Z}$, kde $r = [(1-\beta)r_i + \beta r_j]\,(mod\,N)$ platí

$$C^d\,(mod\,N) = P,$$

kde $P = [2B + 256^{49}((1-\beta)PS_i + \beta PS_j) + (1-\beta)D_i + \beta D_j]\,(mod\,N)$.

Pokud tedy máme dány dva vhodné násobky r_i a r_j pro C, můžeme najít další vhodný násobek jako jejich lineární kombinaci. V praxi se mohou testovat nejmenší pozitivní a negativní hodnoty β a vyzkouší se zda lineární kombinace r dá S-PKCS vyhovující text P. Předpokládáme, že $gcd(r_j - r_i, N) = 1$ a tedy můžeme nalézt konkrétní hodnotu pro každou trojici vhodných násobků (r_i, r_j, r). Autoři uvádějí, že vzniká závislost mezi množstvým informací získaných z r a velikostí β. Platí, že pro malé β se nesnižují hodnoty M_i tak jako pro β blízsko $N/2$. Tedy nejlepší dělení intervalu M_i dosahuje hodnota β pokud se pohybuje blízko $N/2$, ale na druhou stranu je obtížné najít příslušnou hodnotu β hrubou silou. Z uvedených důvodů navrhují autoři získání co největšího množství z rozumně volených hodnot β a pak pokračovat buď v inkrementální podobě původního Bleichenbacherova algoritmu, nebo použít paralelní metodu PT, která bude uvedena níže.

V případě, že použijeme β-metodu (negativní hodnoty) z předchozího odstavce, můžeme dostat hodnotu r_i blízsko N. Získaná hodnota nemůže být

použita, protože vede ke vzniku velkého intervalu pro r v bodě 3) Bleichenbacherova algoritmu. Následující tvrzení upravuje algoritmus pro malé pozitivní hodnoty r i malé negativní hodnoty r modulo N.

Symetrizace.
Nechť máme dány celá čísla r, P a N splňující podmínku $E_1 \leq rP\,(mod\,N) \leq F1$, kde $E_1, F_1 \in \mathbb{Z}$. Potom existuje celé číslo v, $v = N - r$ takové, že $E_2 \leq vP\,(mod\,N) \leq F_2$, kde $E_2 = N - F_1$, $F_2 = N - E_1$.

Jestliže dostaneme použitím β-metody vysokou hodnotu r, pak můžeme využít poznatek z předchozího tvrzení a vytvoříme symetrickou hodnotu v, kterou použijeme v kroku 3) Bleichenbacherova algoritmu. Zároveň je potřeba upravit hranice místo originálních $E_1 = E$, $F_1 = F$ použijeme E_2 a F_2.

3. **Metoda paralelního použití více vlákem (Threads (PT) Method).**

   Složitost Bleichenbacherova algoritmu v kroku 2) závisí na počtu prvků v M_{i-1}. Algoritmus je rychlejší pokud $|M_i| = 1$, protože existuje pouze jeden interval aproximující k hodnotě P_0 a může být tedy hledán další vhodnější r_i. Autoři uvádí, že je lepší vytvořit pro každé $I \in M_{i-1}$ vlastní vlákno jako kdyby zbýval pouze jeden interval. Je-li $|M_{i-1}| = w$, pak dostáváme vlákna $T_1, \ldots, T_w$. Každé vlákno vykonává 2c) a jednotlivě volá BVO. Výsledky nalezené vláknem T_j se promítnou do celé skupiny intervalů všech vlákem. Dojde k vyřazení vláken, jejichž intervaly spadají do vyzkoušených intervalů.

   Autoři uvedli kdy je dobré použít paralelní metodu:

   $$|M_{i-1}| < (2\epsilon Pr(A))^{-1} + 1.$$

   ϵ určuje počet průvodů potřebných od začátku do výskytu jediného intervalu, tj. $|M_i + \epsilon - 1| = 1$, kdy PT metoda začne v kroku i. $Pr(A)$ se používá při výpočtu pravděpodobnosti zda pro nějaký C platí $BVO(C) = 1$. Autoři získali pozorováním $\epsilon = 2$.

Praktická realizace útoku.
Předpokládejme, že chce útočník nalézt text $m = c^d\,(mod\,n)$, kde c je známé celé číslo a d je neznámý tajný RSA exponent. Nechť M_i je množina uzavřených intervalů, které byly vypočítány pro aktuální hodnotu konstanty r_i. V případě, že budeme hovořit o vyhovujícím textu, budeme myslet S-PKCS vyhovující text. Zpráva m_0 se nachází v jednom z intervalů M_i. Dále je známo číslo $B = 2^{8(k-2)} = 256^{k-2}$, kde k je délka B v bajtech. Budeme používat značení $E = 2B$, $F = 3B - 1$.

Modifikace Bleichenbacherova útoku souvisí s BVO a vlastností S-PKCS pro otevřené texty. Dále zde zohledníme autory navrhované optimalizace.

- **nové hranice E' a F'** - důvody zavedí nových hranic vymezujících S-PKCS vyhovující texty jsou uvedeny v části *S-PKCS a BVO vlastnosti* viz. 1

- **úprava hraničních hodnot** $[a, b]$ - úpravy se provádí z důvodu znalosti čísel protokolu (major.minor pro premaster-secret hodnotu) a znalosti přesné pozice separátoru, část *S-PKCS a BVO vlastnosti* viz. 1

- **lineární kombinace** ($\beta - metoda$) - pokud jsme nalezli dva vhodné násobky, můžeme je použít při hledání aktuální hodnoty r_i viz. 2

- **symetrizace** - pokud dostaneme použitím $\beta - metoda$ velikou hodnotu r_i, musíme upravit stávající hranice dle definice uvedené v části *Symetrizace* viz. 2.

- **použití paralelní metody PT** - metoda se použije jen je-li splněna podmínka pro počet intervalů $|M_{i-1}| < (2\epsilon Pr(A))^{-1}+1$ viz. 3. Paralelní metoda se nemusí používat a lze zachovat stávající iterativní charakter Bleichenbacherova algoritmu!

Modifikovaný Bleichenbacherův algoritmus:

1. **Nalezení r_0.**
   Nechť je r_0 náhodně zvolené pozitivní celé číslo. Dokud není $c(r_0)^e$ S-PKCS vyhovující, zvol jiné r_0 a testování opakuj. Při úspěchu nastav:

$$c_0 = c(r_0)^e \, mod(n)$$

$$M_0 = [E', F']$$

$$i = 1$$

2. **Hledání S-PKCS přijatelných zpráv.**

   (a) **Zahájení hledání.**
   Jestliže $i = 1$, pak hledej nejmenší pozitivní celé číslo r_1, pro které platí $r_1 \geq n/(F' + 1)$, a pro které je šifrovaný text $c_0(r_1)^e \, (mod\, n)$ S-PKCS vyhovující.

   (b) **Zbývá-li více intervalů.**
   Jestliže je $i > 0$ a zároveň existují nejméně dva intervaly v M_{i-1}, pak hledej nejmenší pozitivní celé číslo $r_i > r_{i-1}$, kde $r_i = [(1 - \beta)r_k + \beta r_j] \, (mod\, N)$ viz. 2, takové, že je šifrovaný text $c_0(r_i)^e \, (mod\, n)$ S-PKCS vyhovující. Pokud platí $|M_{i-1}| < (2\epsilon Pr(A))^{-1} + 1$, kde $\epsilon = 2$ viz. 3, vytvoříme pro každý interval vlákno, které bude vykonávat následující část 2c).

(c) **Zbývá-li pouze jeden interval.**
Jestliže obsahuje množina M_{i-1} pouze jeden interval, tedy $M_{i-1} = [a, b]$, opakovaně vol nejmenší pozitivní celá čísla r_i a t_i takové, že platí:

$$t_i \geq 2\frac{br_{i-1} - E'}{n} \tag{6}$$

$$\frac{E' + t_i n}{b} \leq r_i \leq \frac{F' + t_i n}{a} \tag{7}$$

než je text $c_0(r_i)^e$ S-PKCS vyhovující. Před vykonáváním zkontrolujeme zda neexistuje řešení (získané již ukončeným vláknem), které by spadalo do hledaného intervalu. Pokud ano vlákno ukončíme a pokračujeme ve stávajícím řešení a vyhneme se tak opakovanému výpočtu.

3. **Zúžení množiny řešení.**
Po nalezení r_i nejprve zkontrolujeme zda není r_i příliž veliké tj. $r_i \approxeq n$. Pokud ano, spočítáme novou hodnotu $v = N - r$ takovou, že $E_2 \leq vP \,(mod\, N) \leq F_2$, kde $E_2 = N - F'$, $F_2 = N - E'$. Položíme $r_i = v$ a upravíme stávající hranice $E' = E_2$ a $F' = F_2$. Vymezíme množinu M_i podle následujícího vztahu:

$$M_i = \bigcup_{(a,b,t)} \left\{ \left[max\left(a, \frac{E' + tn}{r_i}\right), min\left(b, \frac{F' - 1 + tn}{r_i}\right) \right] \right\} \tag{8}$$

pro všechny $[a, b] \in M_{i-1}$ a

$$\frac{ar_i - F'}{n} \leq t \leq \frac{br_i - E'}{n}.$$

4. **Výpočet řešení.**
Jestliže obsahuje M_i jediný interval jehož délka je rovna jedné, tedy platí $M_i = [a, a]$, potom polož $m = a(r_0)^{-1} \,(mod\, n)$ a vrať m jako řešené zadané rovnice $m = c^d \,(mod\, n)$. V opačném případě polož $i = i + 1$ a přejdi na začátek kroku 2.

Závěr

Cílem diplomové práce bylo přiblížit čtenáři rozsáhlou oblast kryptografie, která člověka provází od ranných dob civilizace a svoji důležitost získala s příchodem moderních technologií. Z hlediska celistvosti a zvoleného formátu popisu problému nebylo možné vynechat některou oblast, kterou se kryptografie zabývá. Důvodem, proč jsem zvolil tento formát popisu byl nedostatek podobného souhrnného materiálu českém jazyce. Většina textů neobsahuje průřez od teoretických pojmů po názorné vysvětlení algoritmu s hodnocením bezpečnosti pro jeho možné použití v praxi. Práce nemá na úkol nahradit existující učební materiály, které jsou psány zkušenými odborníky v oblasti. Rozsah textu je z hlediska diplomové práce překročen, lepším řešením by byla volba jedné z oblastí kryptografie. Práce neobsahuje přehled PKI (Public Key Infrastructure) v podobě certifikátů určených pro šifrování s veřejnými klíči.

Teoretická část se zaměřila na vysvětlení základních pojmů, na kterých je kryptografie postavena. Zejména jsou popsány pojmy, které stojí za bezpečností konstrukce kryptografických algoritmů. Pokud by došlo k vynechání základních pojmů, nebylo by možné definovat kryptografické systémy.

Historické období kryptografie je popsáno pouze na příkladech některých šifer. Práce se zejména zaměřuje na moderní kryptografii a vybírá pouze nejznámější kryptografické algoritmy. Práce se zejména soustředí na úspěšné metody kryptoanalýzy, zranitelnost algoritmů a jejich náchylnost vzhledem k útokům. Práce ukazuje základní principy kryptoanalytických útoků a nemá za úkol demonstrovat jejich praktickou realizaci. Nejnovější trend směřování kryptografie je reprezentován kvantovou kryptografií.

Praktická část je rozdělena na dvě části. První část obsahuje implementaci vybraných zástupců symetrických, asymetrických šifrovacích systémů a kryptografických hašovacích funkcí. Algoritmy byly vybírány z hlediska bezpečnosti, kterou v současnosti poskytují.
Druhá část praktické části popisuje úspěšný útok na SSL spojení. Oba dva typy útoků jsou v práci popsány pomocí postupu na implementaci. Druhý z typu útoků (Klima-Rosa-Pokorny) jsem popsal i s navrhovanými optimalizacemi, které autoři popisují pouze teoreticky. Praktická realizace nebyla provedena z důvodu komplikovanosti problému a vzhledem k rozsáhlosti zadání práce.

Conclusions

Main goal of text was representation extensive domain of cryptography, which is going along people from the earliest times of civilization. Modern cryptography is study of techniques for secure communication in the presence of third parties. Cryptography is extensive area so it was important describe all parts of it. The reason why I chose this format was deficit of similar description in Czech language materials. Most of the texts does not cross cryptography from the theoretical terms to sake of explanation algorithm for the evaluation of its safety. Material hasn't the task to replace the existing publications which are written by experienced professionals. Text dosn't contains an overview of PKI (Public Key Infrastructure) in the form of certificates for public key encryption.

The theoretical part is focused on explanation main terms of cryptography. In particular, text describes the terms behind safety design of cryptographic algorithms. If there was a fundamental omission terms, it would be impossible to define cryptographic systems.

Classical cryptographics algorithms were explain on basic cryptographics examples. The text mainly focuses on modern cryptography and selects only the best known cryptographic algorithms. The thesis is mainly focused on successful methods of cryptoanalysis and vulnerability of algorithms and their susceptibility due to the attacks. Material shows the basics principles of cryptoanalytic attacks. Text isn't intended to demostrate their practical implementation. The latest trends of cryptography are represented of quantum cryptography.

The practical part is divided into to main parts. The first part contains the implementation of selected representatives of symmetric, asymmetric encryption and cryptographic hash functions. Algorithms were selected in terms of safety, which currently provide.
The second part describes the practical successful attack on SSL connections.

Reference

[1] Ferguson, Niels, Schneier, Bruce. *Practical Cryptography.* Wiley, 2003.

[2] Schneier, Bruce. *Applied Cryptography.* Wiley, 1996.

[3] Klíma, Vlastimil. *Základy moderní kryptologie Symetrická kryptografie I., II., III.* Karlova Univerzita v Praze, Matematicko-fyzikální fakulta. 2007.

[4] Singh, S. *Kniha kódů a šifer.* Dokořán, 2003.

[5] Janeček, Jiří. *Odhalená tajemství šifrovacích klíčů minulosti.* Naše Vojsko Praha, 1994.

[6] Shannon, Claude E. *Communication Theory of Secrecy Systems.* Bell System Technical Journal, vol.28-4, page 656–715, 1949.

[7] Menezes, Alfred J., Oorschot, Paul C. van, Vanstone, Scott A.. *Handbook of Applied Cryptography.* Fifth Printing, 2001.

[8] *Caesar cipher*

[9] *Zimmermann Telegram*

[10] *Bomba kryptologiczna*

[11] *Bombe*

[12] *Enigma machine*

[13] *Point-to-Point Tunneling Protocol*

[14] *Wired Equivalent Privacy*

[15] Bleichenbacher, David. *Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS#1.* in Proc. of CRYPTO '98, pp. 1-12, 1998.

[16] Klíma, Vlastimil, Pokorný, Ondřej, Rosa, Tomáš. *Attacking RSA-based Sessions in SSL/TLS.* in Proc. of CHES '03, Cologne, Germany, September 7-11, 2003.

[17] Khovratovich, Dmitry. *New Approaches to the Cryptanalysis of Symmetric Primitives.* The Faculty of Sciences, Technology and Communication. 2010.

[18] Dworkin, Morris. *Recommendation for Block Cipher Modes of Operation, Methods and Techniques.* NIST Special Publication 800-38A, 2001.

[19] Borisov, Nikita, Goldberg, Ian, Wagner, David. *Intercepting Mobile Communications: The Insecurity of 802.11.* Elektronická publikace, 2006.

[20] Fischer, Simon, Khazaei, Shahram, Meier, Willi. *Chosen IV Statistical Analysis for Key Recovery Attacks on Stream Ciphers.* Springer, 2008.

[21] Ireland, David. *Using Padding in Encryption.* Elektronická publikace, 2010.

[22] Moeller, Bodo. *Security of CBC Ciphersuites in SSL/TLS: Problems and Countermeasures.* Elektronická publikace, 2004.

[23] Fruhwirth, Clemens. *New Methods in Hard Disk Encryption.* Vienna University of Technology, 2005.

[24] Kohl, J. *The Use of Encryption in Kerberos for Network Authentication* Crypto '89, 1989.

[25] Daemen, Joan. *Cipher and Hash Function Design, Strategies Based on Linear and Differential Cryptanalysis* Katholieke Universiteit Leuven, 1995.

[26] Baldwin, Robert, Ronald. *The RC5, RC5-CBC, RC5-CBC-Pad, and RC5-CTS Algorithms* MIT Laboratory for Computer Science and RSA Data Security, Inc., 1996.

[27] Rivest, R. L. *The RC5 Encryption Algorithm* Proceedings of the Second International Workshop on Fast Software Encryption (FSE) 1994e. pp. 8696.

[28] Rivest, R.L.; Robshaw, M.J.B.; Sidney, R.; Yin, Y.L.. *The RC6 Block Cipher* 1998.

[29] Biryukov, A.; Kushilevitz, E.. *Improved Cryptanalysis of RC5.* EUROCRYPT 1998.

[30] *distributed.net*

[31] WASTE Again Team. *WASTE Again: Introduction.* WASTE Again, 2008.

[32] NIST. *Proposal To Extend CBC Mode By "Ciphertext Stealing".* NIST, 2007.

[33] Hudde, Christoph. *Building Stream Ciphers From Block Ciphers and Their Security.* Seminararbeit Ruhr-Universitat Bochum, 2009.

[34] Davies, D. W., Parkin, G. I. P. *The Average Cycle Size of The Key Stream in Output Feedback Encipherment.* Springer, 1983.

[35] Tirtea, R., Deconinck, G. *Specifications overview for counter mode of operation. security aspects in case of faults.* MELECON, 2004.

[36] Lipmaa, Helger, Rogaway, Phillip, Wagner, David. *Comments to NIST concerning AES modes of operation: CTR-mode encryption.* NIST, 2000.

[37] Vaudenay, Serge. *A Classical Introduction to Cryptography: Applications for Communications Security.* Springer, 2005.

[38] CFRG Working Group. *VMAC: Message Authentication Code using Universal Hashing* CFRG Working Group, 2010.

[39] *Playfair cipher.*

[40] *Transpozični šifry.*

[41] Ullmann, J.R.. *A binary n-gram technique for automatic correction of substitution, deletion, insertion and reversal errors in words.* The Computer Journal, 1977.

[42] RSA Laboratories. *Has DES been broken?* RSA Laboratories, 2009.

[43] Barker, William C.. *Recommendation for the Triple Data Encryption Algorithm (TDEA).* NIST, 2008.

[44] Schneier, Bruce. *The Blowfish Encryption Algorithm.* Dr. Dobb's Journal, 1995.

[45] Rijmen, Vincent. *Cryptanalysis and Design of Iterated Block Ciphers* Ph.D thesis, 1997.

[46] Biham, E.; Dunkelman, O.; Keller, N.. *A New Attack on 6-Round IDEA.* Springer-Verlag, 2007.

[47] Daemen, Joan; Rijmen, Vincent. *The Design of Rijndael: AES - The Advanced Encryption Standard.* Springer, 2002.

[48] *Advanced Encryption Standard*

[49] Bertoni, Guido; Breveglieri, Luca; Fragneto, Pasqualina; Macchetti, Marco; Marchesin, Stefano. *Efficient Software Implementation of AES on 32-Bit Platforms.* Lecture Notes in Computer Science, 2003, Volume 2523/2003.

[50] Goldwasser, Shafi; Bellare, Mihir. *Lecture Notes on Cryptography.* Massachusetts Institute of Technology (MIT), 1996-2008.

[51] Gruska, Josef. *Public-key cryptosystems, II. Other cryptosystems, security, PRG, hash functions.*

[52] Bareš, Ladislav. *Autentizační protokoly.* Univerzita Pardubice, 2010.

[53] *Metody ověřování pro použití se službou IAS*

[54] Reichert, Pavel; Abilov, Albert; Šifta, Radim. *Kvantová kryptografie v optickém přenosovém systému.* Fakulta elektrotechniky a komunikačních technologií VUT v Brně, 2011.

[55] Černoch, Antonín. *Kvantová informatika pro komunikace v budoucnosti.* Univerzita Paleckého v Olomouci, 2011.

[56] *Quantum cryptography*

[57] *Quantum key distribution*

[58] *http://arxiv.org/abs/quant-ph/0510025v1*

[59] Knopf, Christian. *Cryptographic Hash Functions.* Leibniz Universitat Hannover, 2007.

[60] *The hash function RIPEMD-160*

[61] Klíma, Vlastimil. *Nedůvěřujte kryptologům.* DCD Publishing, Hotel Diplomat, Praha, 10. - 11. 11. 2004.

[62] Klíma, Vlastimil. *Hašovací funkce, principy, příklady a kolize.*

[63] Klíma, Vlastimil. *Hašovací funkce MD5 a další prolomeny!*

[64] Klíma, Vlastimil. *Tunnels in Hash Functions: MD5 Collisions Within a Minute*

[65] Black, J.; Cochran, M., Highland, T. *A Study of the MD5 Attacks: Insights and Improvements*

[66] *Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms*

[67] Stevens, Marc. *HashClash*

[68] Bouillaguet, Charles. *Herding, Second Preimage and Trojan Message Attacks Beyond Merkle-Damgaard*

[69] Davies, D.W.. *Message Authenticator Algorithm*

[70] *An improved preimage attack on MD2*

[71] *Cryptanalysis of MD2*

[72] *CVE-2009-2409*

[73] Leurent, Gaetan. *MD4 is Not One-Way* FSE 2008.

[74] Wang, Xiaoyun; Feng, Dengguo; Lai, Xuejia; Yu, Hongbo. *Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD*

[75] Mendel, Florian; Pramstaller, Norbert; Rechberger, Christian; Rijmen, Vincent. *On the Collision Resistance of RIPEMD-160* Lecture Notes in Computer Science, 2006, Volume 4176/2006.

[76] *Shandong University* Shandong University.

[77] *Cryptanalysis of SHA-1*

[78] Guo, Jian; Matusiewicz, Krystian. *Preimages for Step-Reduced SHA-2*

[79] *Quadratic residuosity problem*

[80] Schneier, Bruce; Kelsey, John. *Yarrow*

[81] Hölbl, Marko; Welzer, Tatjana; Brumen, Boštjan. *Improvement of the Peyravian-Jeffriess user authentication protocol and password change protocol.* Faculty of Electrical Engineering and Computer Science, University of Maribor, Smetanova ulica 17, 2000 Maribor, Slovenia

[82] *Point-to-point protocol*

[83] *PPP Authentication Protocols*

[84] *PPP Challenge Handshake Authentication Protocol (CHAP)*

[85] *Microsoft PPP CHAP Extensions, Version 2*

[86] *Cryptanalysis of Microsoft's PPTP Authentication Extensions (MS-CHAPv2)*

[87] *Extensible Authentication Protocol (EAP) Key Management Framework*

[88] *Extensible Authentication Protocol (EAP)*

[89] *Asleap - exploiting cisco leap*

[90] *What Is Kerberos Authentication?*

[91] *Kerberos (protocol)*

[92] *Pretty Good Privacy*

[93] *Public-key infrastructure*

[94] *Knapsack problem*

[95] *Station-to-Station protocol*

[96] ElGamal, Taher. A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms IEEE Transactions on Information Theory 31, 1985.

[97] Nyberg, K.; Rueppel, R. A.. *Message recovery for signature schemes based on the discrete logarithm problem*

[98] *CramerShoup cryptosystem*

[99] *Elliptic Curve DSA*

[100] *Pollard's p - 1 algorithm*

[101] *PohligHellman algorithm*

[102] Ping, Zhou; Yingzhan, Kou; Caisen, Chen; Jilao, Zhang. *Research on Key Technology for Data Cache Timing Attack on DSA* IEEE Computer Society Washington, DC, USA. 2011

[103] *AKS primality test*

[104] *MillerRabin primality test*

[105] *Shor's algorithm*

[106] *Remote timing attacks are practical.*

[107] *Coppersmith's Attack*

A. Obsah přiloženého CD

`bin/`
: Adresář obsahuje instalátor *BKA.msi* a podadresář PROGRAM s aplikací *BKA.exe*, která je volně spustitelná přímo z CD.

`doc/`
: Adresář obsahuje text diplomové práce *Bezpečné.kryptografické.algoritmy.pdf*. Dále se zde nachází ZIP archiv BKA.zip, který obsahuje zdrojový text dokumentace včetně obrázků.

`src/`
: Adresář obsahuje kompletní zdrojové texty programu *Bezpečné kryptografické algoritmy* včetně všech potřebných knihoven. Zdrojové texty najdete v archivu *BKA.zip*.

`readme.txt`
: Textový dokument obsahuje instrukce pro instalaci a spuštění programu *Bezpečné kryptografické algoritmy*, včetně požadavků pro jeho provoz.

Navíc CD obsahuje:

`data/`
: Adresář obsahuje textové dokumenty s testovacími hodnotami pro jednotlivé algoritmy.