Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Porovnání současné a postkvantové kryptografie

Jan Schlosárik

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Jan Schlosárik

Informatika

Název práce

Porovnání současné a postkvantové kryptografie

Název anglicky

Comparasion of current and post-quantum cryptography

Cíle práce

Cílem práce je porovnat současné běžně používané a postkvantové šifrovací metody mezi sebou pomocí jejich časové složitosti.

Metodika

Práce je založena na studiu odborné a vědecké literatury.

Teoretická část obecně popisuje kryptografii, současně používané kryptografické metody a kvantovou kryptografii.

V praktické části je použit program, který využívá současně používané šifrovací metody a šifrovací metody postkvantové. Je změřen čas při zašifrování a dešifrování použitou metodou. Tyto testy jsou provedeny na více typech souborů.

Na základě poznatků z případové studie a z analýzy zdrojů je vytvořen závěr práce.

Doporučený rozsah práce

30 – 40 stran

Klíčová slova

asymetrické šifrování, kryptografie, postkvantové šifrování, symetrické šifrování, šifrování, zabezpečení

Doporučené zdroje informací

AUMASSON, Jean-Philippe, 2018. Serious cryptography: A Practical Introduction to Modern Encryption. San Francisco: No Starch Press. ISBN 15-932-7826-8.

OPPLIGER, Rolf, 2021. Cryptography 101: From Theory to Practice. Artech House. ISBN 978-1630818463.

STALLINGS, William, 2020. Cryptography and Network Security: Principles and Practice. 8th Edition. Pearson. ISBN 9780135764251.

WONG, David, 2021. Real-World Cryptography. Manning Publications. ISBN 978-1-61729-671-0.

Předběžný termín obhajoby

2021/22 LS – PEF

Vedoucí práce

Ing. David Buchtela, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 8. 3. 2022

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 10. 3. 2022

doc. Ing. Tomáš Šubrt, Ph.D.

Děkan

V Praze dne 15. 03. 2022

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Porovnání současné a postkvantové kryptografie" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 15.3.2022 _____

Poděkování

Rád bych touto cestou poděkoval svému vedoucímu práce Ing. Davidu Buchtelovi, Ph.D. za pomoc, dohled a cenné rady při psaní této práce.

Porovnání současné a postkvantové kryptografie

Abstrakt

Cílem bakalářské práce je komparovat současné a postkvantové šifrovací metody dle jejich rychlosti mezi sebou.

V teoretické části práce je popsána současná a postkvantová kryptografie, a témata úzce spojena s nimi. Tato část je vypracována pomocí poznatků získaných studiem odborné literatury. Jsou v ní popsány způsoby fungování jednotlivých kryptografických metod a taktéž základní koncepty s nimi používané.

Vlastní práce obsahuje popis programů realizující kryptografické metody. Tyto programy jsou použity pro získání měření. Na základě toho je vypočítána rychlost jednotlivých metod a podle výsledků jsou metody porovnány.

Na základě poznatků z teoretické části a výsledků měření z části praktické je zpracován závěr.

Klíčová slova: asymetrické šifrování, kryptografie, postkvantové šifrování, symetrické šifrování, šifrování, zabezpečení

Comparasion of current and post-quantum cryptography

Abstract

The aim of the bachelor thesis is to compare current and post-quantum encryption methods according to their speed with each other.

The theoretical part of the thesis describes current and post-quantum cryptography, and topics closely related to them. This part is elaborated with the help of knowledge gained from the study of literature. It describes the ways in which individual cryptographic methods work, as well as the basic concepts used with them.

The thesis contains a description of programs implementing cryptographic methods. These programs are used to obtain measurements. Based on this, the speed of each method is calculated, and the methods are compared according to the results.

The conclusion is based on the knowledge from the theoretical part and on the results of measurements in the practical part.

Keywords: asymmetric encryption, cryptography, postquantum encryption, symmetric encryption, encryption, security

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Kryptografie studuje techniky bezpečné komunikace, umožňuje zobrazit obsah pouze odesílateli a zamýšlenému příjemci. Při přenosu elektronických dat je její nejběžnější využití šifrování a dešifrování e-mailů a jiných textových zpráv.

Bakalářská práce je zaměřena na současnou běžně používanou a postkvantovou kryptografii. Jejich metody jsou mezi sebou vzájemně porovnány pomocí jejich časové složitosti. Práce se zabývá otázkou, zda by postkvantová kryptografie dokázala plně nahradit kryptografii současnou.

Současná kryptografie využívá složitosti výpočtu různých matematických problémů tak, že by mělo trvat rozšifrování bez znalosti klíče velmi dlouho, v řádu až tisíců let. Existuje však teoretický model pro kvantové počítače, které by využívaly pro vykonávání výpočtů fenomény z kvantové mechaniky. Tento počítač by mohl představovat problém pro současnou kryptografii, jelikož by mu výpočty netrvaly tak dlouhou dobu, tím pádem by tato kryptografie již nebyla bezpečná.

V důsledku tohoto teoretického modelu vznikla postkvantová kryptografie. Ta by měla pokrývat metody, které by ani samotné kvantové počítače neměly dokázat vyřešit rychleji. Jednalo by se tedy o stále bezpečnou metodu šifrování i v případě úspěšného sestavení teoretického modelu kvantového počítače.

2 Cíl práce a metodika

2.1 Cíl práce

Cílem práce je porovnat současné běžně používané a postkvantové šifrovací metody mezi sebou pomocí jejich časové složitosti. Na základě tohoto porovnání je zjištěno, zda by metody postkvantové dokázaly nahradit současné.

2.2 Metodika

Bakalářská práce je rozdělena na dvě části – teoretická východiska a vlastní práci.

Teoretická část je zpracována na základě sběru dat a studia odborné literatury z oblasti současné a postkvantové kryptologie. Je sestaven ucelený přehled o charakteristice kryptologie, jejím dělením a kryptografických metodách. S větším důrazem jsou popsány i konkrétní metody z těchto oblastí, které jsou využity pro zpracování části praktické.

Ve vlastní práci je nejprve představen hardware, na kterém je tato část vypracována a na kterém je provedeno měření. Následně je představen software, jenž je použit pro demonstraci a měření jednotlivých šifrovacích algoritmů. Je popsáno ovládání a výpis tohoto softwaru, aby bylo zřejmé odkud pochází naměřené hodnoty. Měření doby běhu jednotlivých algoritmů je pro demonstraci rychlosti provedeno na souborech více velikostí. Naměřený čas je poté přepočítán na přenosovou rychlost v jednotkách MB/s a v těchto jednotkách jsou algoritmy porovnány pomocí metody komparace.

3 Teoretická východiska

3.1 Problematika kryptologie

Původním záměrem kryptologie bylo skrýt význam slov, a ochrana důvěrnosti a tajemství příslušných údajů. Tento smysl přetrvává dále, avšak tento pojem je používán pro mnoho ostatních účelů a aplikací spojených s tématem bezpečnosti kromě ochrany důvěrnosti a utajení dat. Konkrétněji se kryptologie týká matematické vědy a studijního oboru, který zahrnuje kryptografii a kryptoanalýzu. (Oppliger, 2021)

3.1.1 Kryptografie

Význam termínu kryptografie lze parafrázovat jako skryté psaní. Kryptografie se týká matematické vědy, která se zabývá transformací dat tak, aby byl jejich význam nesrozumitelný, to znamená, aby se skryl jejich sémantický obsah, aby se zabránilo jejich neodhalené změně nebo neoprávněnému použití. Pokud je tato transformace oboustranná, kryptografie se zabývá také obnovou zašifrovaných dat do srozumitelné podoby. (Oppliger, 2021)

Termín kryptografie je více popisován obecněji jako studium matematických technik souvisejících se všemi aspekty informační bezpečnosti. Tyto aspekty zahrnují, avšak nejsou omezeny na, důvěrnost dat, integritu dat, autentizaci entity, autentizaci původu dat, nepopiratelnost a mnoho dalších. Tato definice je velmi široká a je v ní zahrnuto vše, co přímo a nepřímo souvisí s informační bezpečností. (Martin, 2020)

3.1.2 Kryptoanalýza

Kryptoanalýza je proces zničení kryptografické ochrany, nebo obecněji na studium bezpečnostních vlastností a možností prolomení kryptografických technik a systémů. Dle Oppligera (2021) je kryptoanalytik antagonista pro kryptografa, jeho úkolem je prolomit, či obejít ochranu, jenž kryptograf navrhl a implementoval. Zcela přirozeně je mezi těmito skupinami rivalita, i přesto, že jednotlivci mohou disponovat znalostmi z obou oblastí. (Oppliger, 2021)

3.1.3 Náhodnost

U každého šifrovacího algoritmu, s výjimkou hašovacích funkcí, je využívána náhodnost. Bez té by nebyla kryptografie možná, všechny operace by byly předvídatelné, a tudíž by nebyly bezpečné. (Aumasson, 2018) V případě bezpečnosti a kryptografie musí být tedy tato náhodnost nepředvídatelná. V kryptografii je extrahována náhodnost z pozorování těžko předvídatelných fyzikálních jevů. (C. van Oorschot, 2020)

Například je těžké předpovědět výsledek hodu kostkou, i přestože se jedná o deterministický proces, a kdyby byly známy všechny podmínky na počátku, mělo by být možné předpovědět výsledek. Stačí ale malá nepřesnost ve znalosti těchto podmínek a výsledek může být jiný. (Wong, 2021)

Náhodnost nemůže být generována pouze samotným počítačovým algoritmem. V dnešní době má čím dál tím více zařízení přístup k dalším sensorům a hardwarovým pomůckám, které poskytují lepší zdroj entropie. Tyto generátory jsou označovány jako hardwarové generátory náhodných čísel, využívají vnější nepředvídatelné fyzikální jevy jako například tepelný šum pro extrakci náhodnosti. (Aumasson, 2018) Tento šum však nemusí přinášet dostatek entropie, díky tomu musí extraktory náhodnosti vyčistit a shromáždit několik zdrojů šumu dohromady, než jej bude možné použít pro aplikaci v kryptografii. Toho může být dosaženo například použitím hašovacích funkcí na různé zdroje a použitím operandů XOR pro zkombinování dohromady. Tato metoda se však může jevit jako pomalá a pro nějaké aplikace, které by potřebovali hodně náhodných čísel rychleji, by to znamenalo jejich zpomalení. (Wong, 2021)

3.1.4 Hardwarové generátory náhodných čísel a generátory pseudonáhodných čísel

Hardwarový generátor náhodných čísel je zdrojem nejistoty, zdrojem entropie. Může je získat z běžícího operačního systému čerpáním z připojených senzorů, vstupních a výstupních zařízení, aktivity sítě nebo disku, systémových logů, běžících procesů a aktivity uživatele, jako je stisknutí kláves či pohyb myši. Takovéto činnosti generované systémem a lidmi mohou být dobrým zdrojem entropie, avšak mohou být manipulovatelné útočníkem. Tato nejistota je využívána generátorem pseudonáhodných čísel. Používání těchto dvou prvků je klíčové k praktické a bezpečné kryptografii. (Wong, 2021)

Generátor pseudonáhodných čísel spolehlivě vytváří mnoho umělých náhodných bitů z několika skutečných náhodných bitů. Například hardwarový generátor náhodných čísel, který využívá pohyby myši k vytvoření náhodných bitů by přestal fungovat, kdyby se myš vůbec nehýbala, avšak generátor pseudonáhodných čísel by vracel pseudonáhodné bity pořád. (Aumasson, 2018)

Generátor pseudonáhodných čísel spoléhá na hardwarovém generátoru náhodných čísel, ale chová se jinak. Hardwarový generátor náhodných čísel vytváří skutečné náhodné bity relativně pomalu z analogových zdrojů nedeterministickým způsobem a žádnou garancí vysoké entropie. Oproti tomu generátor pseudonáhodných čísel rychle generuje náhodně vypadající bity z digitálních zdrojů deterministickým způsobem a s maximální entropií. (Aumasson, 2018)

3.1.5 Distribuce klíčů

Většina moderních kryptografických systémů má pesimistický předpoklad, že útočník o daném systému ví úplně vše s výjimkou soukromého klíče. Bezpečné získání soukromých klíčů obou stran komunikace je tedy kritickým problémem každého kryptografického systému. (Burda, 2019)

Symetrické kryptosystémy využívají jeden klíč k zabezpečení komunikace pro obě strany. Tento klíč lze distribuovat více způsoby, prvním způsobem je transport klíče kurýrní službou, jde tedy o fyzickou přepravu klíče druhé straně. Tato přeprava je realizována pomocí speciálního paměťového uložiště zvaného „key loader“. Klíč je do tohoto uložiště nejdříve zapsán, toto úložiště je poté kurýrem dopraveno druhé straně. Ta je nakonec daným způsobem autentizována, například heslem, a je jí umožněno získat klíč z paměti uložiště. (Burda, 2019)

Další možností je šifrované odeslání klíče pomocí veřejného kanálu, to je možné použitím symetrického i asymetrického kryptosystému. V případě symetrického kryptosystému mají obě strany transportní klíč, který je využíván výhradně pro transport klíče. Tento klíč má dlouhodobou platnost a běžně je doručován kurýrní službou. Distribuce probíhá tak, že stranou A je vygenerován soukromý klíč, tento klíč je navíc zašifrován pomocí transportního klíče do podoby kryptogramu a ten je odeslán přes veřejný kanál straně B. Ta tento kryptogram rozšifruje pomocí transportního klíče, tím je získán soukromý klíč, který je poté používán ke komunikaci se stranou A. (C. van Oorschot, 2020)

3.1.6 Digitální podpis

Dvě strany, které si vyměňují zprávy jsou chráněny autentizací zpráv od jakékoli třetí strany. Nejsou však chráněny proti sobě. Mezi oběma stranami je možných několik forem sporu. Například v případě, kdy strana A odešle ověřenou zprávu straně B, je možné že nastanou následující spory: (Stallings, 2020)

1. Strana B může zfalšovat jinou zprávu a tvrdit, že pochází od strany A. Strana B by jednoduše mohla vytvořit zprávu a připojit ověřovací kód pomocí klíče, který jí strana A sdílela.
2. Strana A může odeslání zprávy popřít. Protože je možné, aby strana B zprávu zfalšovala. Neexistuje způsob, jak dokázat, že strana A opravu zprávu poslala. (Stallings, 2020)

V situacích, kdy mezi odesílatelem a příjemcem není úplná důvěra, je potřeba něco víc než autentizace. Nejlákavějším řešením tohoto problém je digitální podpis, ten musí mít následující vlastnosti: (Stallings, 2020)

- Musí být schopen ověřit autora, datum a čas podpisu
- Musí ověřit obsah v době podpisu
- Je třeba, aby v případě sporu byl ověřitelný třetími stranami (Stallings, 2020)

Digitální podpis tedy zahrnuje funkci ověřování. Na základě jeho vlastností a možných útoků lze formulovat následující požadavky na digitální podpis: (Stallings, 2020)

- Podpis musí být bitový vzorek, který závisí na zprávě určené k podpisu.
- Podpis je povinen obsahovat některé informace, jež zná pouze odesílatel, aby se zabránilo padělání a popření
- Digitální podpis musí být relativně jednoduché vytvořit
- Digitální podpis musí být poměrně snadné rozpoznat a ověřit
- Padělání digitálního podpisu musí být výpočetně neproveditelné, nelze to provést ani vytvořením nové zprávy pro existující digitální podpis či vytvořením podvodného digitálního podpisu pro danou zprávu
- Musí být jednoduché uchovávat kopii digitálního podpisu v úložišti (Stallings, 2020)

Bezpečná hašovací funkce, jež je vložena do schématu poskytuje základ pro splnění těchto požadavků. (Stallings, 2020)

3.1.7 Hašovací funkce

Hašovací funkce při stejném vstupu vždy reprodukuje stejný řetězec bitů. Tento jednoduchý jev je extrémně užitečný pro vytváření mnoha dalších konstrukcí v kryptografii. Hašovací funkce může v různých případech poskytnout záruku integrity dat. (Stinson, 2019)

Vstup hašovací funkce může mít libovolnou velikost, může být i prázdný. Výstup této funkce je pokaždé stejně dlouhý a deterministický: vždy produkuje stejný výsledek, pokud je dán stejný vstup. Pro příklad funkce SHA-256 vždy poskytuje výstup 256 bitů, který je vždy zakódován jako 64 alfanumerických znaků v šestnáctkové soustavě. Jednou z hlavních vlastností hašovací funkce je, že by nemělo být možné určit vstup pouze z výstupu. (Wong, 2021)

3.1.8 Prokazatelné a heuristické zabezpečení

Prokazatelné zabezpečení je skvělým nástrojem k získání důvěry vůči kryptografickému schématu, ale nevztahuje se na všechny druhy algoritmů. Jako prokazatelné je označován jakýkoli typ nebo úroveň zabezpečení, jež je možné dokázat. Obvykle je toho dosaženo matematickými důkazy. Ve skutečnosti většina symetrických šifer nemá bezpečnostní důkaz. Každý den se například spoléháme na Advanced Encryption Standard (AES), abychom mohli bezpečně komunikovat pomocí našich zařízení, ale AES není prokazatelně bezpečný standard. Neexistuje žádný důkaz, že je těžké ho prolomit jako nějaký známý problém. AES nesouvisí s žádným matematickým problémem nebo jiným algoritmem, protože to je obtížný problém sám o sobě. V případech, kdy prokazatelné zabezpečení neplatí je jediným důvodem, proč šifře věřit, to že se ji mnoho zkušených lidí pokusilo prolomit a neuspěli. To se někdy nazývá jako heuristické zabezpečení. (Aumasson, 2018)

Nikdy není stoprocentně jisté, že je šifra bezpečná, ale můžeme si být docela jisti, že algoritmus nebude narušen. A to ani v případě, kdy stovky zkušených kryptoanalytiků strávili stovky hodin pokusy o její prolomení a zveřejnili svá zjištění, která obvykle obsahovala pokusy o útoky na zjednodušené verze šifry. Při analýze nové šifry se

kryptoanalytici nejprve snaží prolomit jedno kolo, poté dvě, tři a nakonec co nejvíce dokážou. Rozdíl mezi celkovým počtem kol a počtem úspěšně napadených kol je bezpečnostní marže. Když je po letech studia tato marže stále vysoká, získáváme větší jistotu, že je šifra pravděpodobně bezpečná. (Aumasson, 2018)

3.2 Současná kryptografie – Symetrická

V symetrické kryptografii je využíván pouze jeden klíč, kterým se zprávy šifrují i dešifrují. (Buell, 2021) Pokud M je prostor pro prostý text, C prostor pro šifrovaný text a K prostor pro klíč, pak symetrický šifrovací systém nebo šifra je pár (E, D) rodin efektivně vyčíslitelných funkcí, které jsou definovány takto: (Oppliger, 2021)

- $D : K \times M \to C$ označuje rodinu $\{E_k : k \in K\}$ šifrovacích funkcí $E_k : M \to C$
- $D : K \times C \to M$ označuje rodinu $\{D_k : k \in K\}$ příslušných dešifrovacích funkcí $D_k : C \to M$. (Oppliger, 2021)

Pro každou zprávu $m \in M$ a klíč $k \in K$ musí být funkce E_k a D_k vzájemně inverzně, to znamená že platí $D_k\big(E_k(m)\big) = m$. V opačném případě nemusí být šifrovaný text možné dešifrovat, a proto šifrovací systém nemusí být v první řadě příliš užitečný. (Oppliger, 2021)

V některých symetrických šifrovacích systémech nezáleží, jestli je vstup nejdříve šifrován a poté dešifrován nebo naopak nejdříve dešifrován a poté šifrován. To znamená, že platí: (Oppliger, 2021)

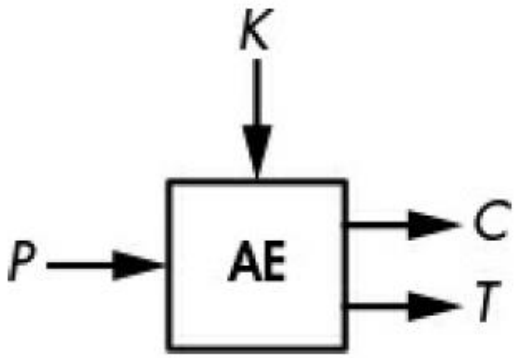
$$D_k\big(E_k(m)\big) = E_k\big(D_k(m)\big) = m \tag{1}$$

V případě posunutí této myšlenky ještě o krok dále, symetrický systém je komutativní, pokud lze vícekrát zašifrovanou zprávu dešifrovat v libovolném pořadí. Pokud je zpráva m zašifrována pomocí klíčů k_1 a k_2 jako $c = E_{k_2}(E_{k_1}(m))$ pak v komutativním systému platí: (Oppliger, 2021)

$$D_{k_2}\left(D_{k_1}(c)\right) = D_{k_1}\left(D_{k_2}(c)\right) = m \tag{2}$$

3.2.1 Ověřené šifrování

Ověřené šifrování je typ symetrického šifrování, které vrací autentizační značku kromě samotného šifrovaného textu. Liší se tou vlastností, že z šifrovacího algoritmu vrátí navíc s šifrovaným textem i autentizační značku. Jedná se o krátký řetězec, který je nemožné uhodnout bez klíče, je vracen jako výstup šifry současně s šifrovaným textem. Na obrázku číslo 1 je tento řetězec označen písmenem T.(Aumasson, 2018)



Obrázek 1 Schéma ověřeného šifrování Zdroj: (Aumasson, 2018)

Autentizační značka zajišťuje integritu zprávy a je používána jako důkaz, že přijatý šifrovaný text je totožný s tím, který odeslala legitimní strana, která zná klíč K. Pokud je K sdílen pouze s jednou další stranou, značka také zaručuje, že zpráva byla odeslána touto stranou. To znamená, že implicitně ověřuje očekávaného odesílatele jako skutečného tvůrce zprávy. (Aumasson, 2018)

3.2.2 Proudové šifry

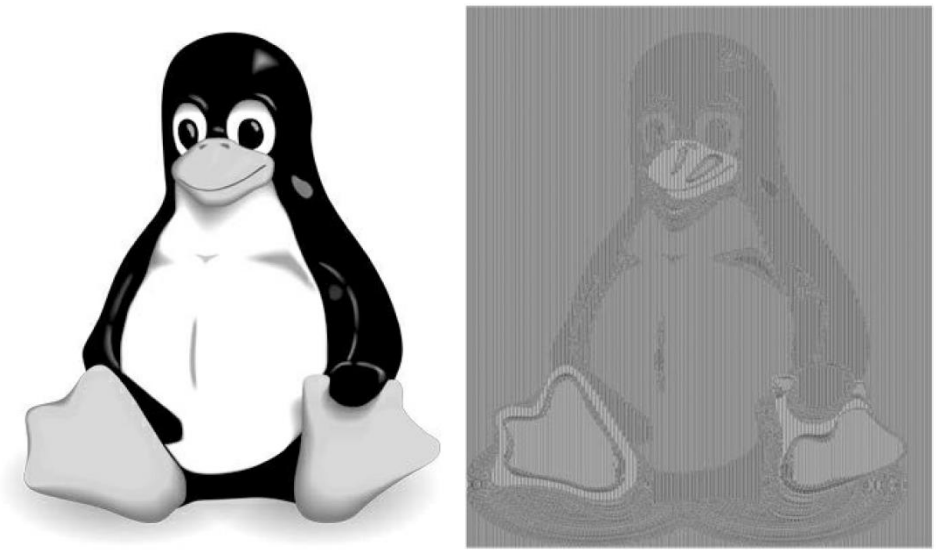
V proudových šifrách je šifrován tok dat bit po bitu nebo bajt po bajtu. Při každém šifrování je transformován jeden a týž bit nebo bajt veřejného textu do různých bitů nebo bajtů textu šifrovaného. Příklady těchto proudových šifer jsou například autoklíčová Vignèrova šifra či Vernamova šifra, ta používá jednorázový proud bitů, který je stejně dlouhý jako nešifrovaný text. Pokud je tento proud bitů dostatečně náhodný, tak je tato šifra prakticky nerozluštitelná bez získání samotného klíče. Avšak tento klíč musí být poskytnut oběma stranám komunikace předem, dříve, než začnou komunikaci, přes nezávislý a bezpečný kanál. To přináší velké, až nepřekonatelné, logistické problémy, pokud je zamýšlený datový provoz velmi velký. (Stallings, 2020)

V souladu s tím musí být generátor bitového toku implementován tak, aby mohli kryptografický bitový tok vytvářet oba uživatelé. V tomto přístupu je generátor bitového toku algoritmu řízený klíčem a musí produkovat bitový tok, který je kryptograficky silný. To znamená, že musí být prakticky výpočetně nemožné předpovídat budoucí části bitového toku na základě částí předchozích. Oba uživatelé tedy musí sdílet takzvaný generující klíč a každý z nich může produkovat proud bitů. (Stallings, 2020)

3.2.3 Blokové šifry

Blokovými šiframi se rozumí šifry s algoritmem, který pracuje s blokem bitů najednou, při tom je vytvářen blok šifrovaného textu z textu prostého. Pro příklad AES pracující s 128bitovými bloky vyprodukuje 128 bitů šifrovaného textu ze 128 bitů textu prostého. Pokud se stane, že je dat více, jsou rozděleny do bloků právě tak, aby s nimi algoritmus dokázal pracovat. (Buell, 2021)

Díky tomu pak mají tyto šifry různé provozní režimy, které definují, jak algoritmy zachází s těmito bloky. První a nejvíce jednoduchý z těchto režimů je ECB, šifra pracující s tímto režimem vezme libovolně dlouhou zprávu, ta je rozdělena do n-bitových bloků, kde n představuje délku bloku blokové šifry a každý tento blok je individuálně zašifrován či rozšifrován. Pokud není délka zprávy násobkem n bitů, tak zpráva musí být nejdříve doplněna o takzvanou výplň, aby se přizpůsobila délce bloku. Zároveň ale v tomto režimu se prostý text šifruje vždy stejně a tím pádem je i stejný šifrovaný text. (Oppliger, 2021) Tento režim díky tomu nemusí být vždy bezpečný, například u obrázku, který je z velké části tvořen stejným pozadím. Toto pozadí bude i stejně zašifrováno, tím pádem budou obrysy obrázku lehce identifikovatelné. To lze vidět i na obrázku číslo 2. (Buell, 2021)

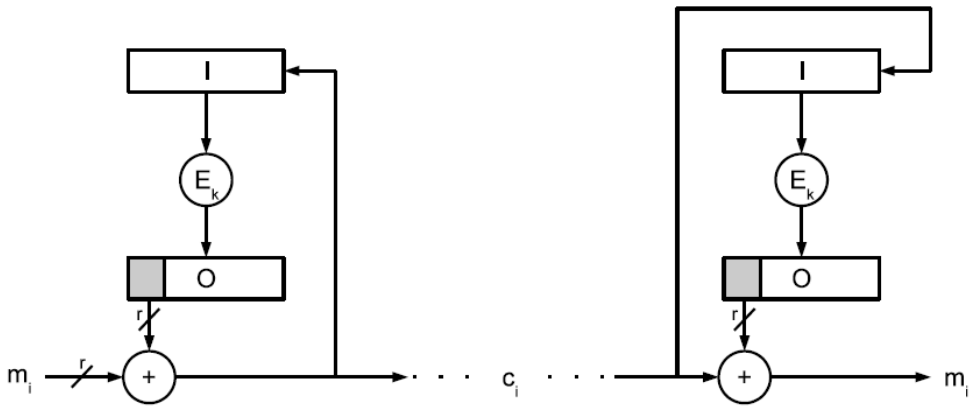


Obrázek 2 Originální obrázek(vlevo) a obrázek šifrovaný módem ECB(vpravo) Zdroj: (Aumasson, 2018)

Dalším režimem je režim CBC, který je často využíván k překonání největší nevýhody ECB. Toho je dosahováno tím, že šifrování nezávisí jen na prostém textu a klíči, ale i na předchozím bloku zašifrovaného textu, až na prvního blok, u kterého je využit takzvaný inicializační vektor, kterým může být například náhodné číslo. To znamená, že bloky šifrovaného textu jsou kryptograficky zřetězeny a tedy, že jeden konkrétní blok šifrovaného textu závisí na všech předchozích blocích. Za předpokladu, že inicializační vektor a všechny předchozí bloky nejsou stejné. (Oppliger, 2021)

Dále existuje režim CFB, ten prakticky mění blokovou šifry v proudovou. Přesněji využívá blokovou šifru k vygenerování sekvence pseudonáhodných bitů a poté přidá modulo 2 těchto bitů do prostého texu k vygenerování šifrovaného textu.

Na obrázku 3 je vlevo vyobrazen proces šifrování a vpravo proces dešifrování. Oba procesy mají k dispozici dva registry, registr I pro vstup a registr O pro výstup. Na obou stranách jsou registry inicializovány pomocí inicializačního vektoru před zahájením procesu šifrování a dešifrování. Délka obou registrů a také inicializačního vektoru odpovídá délce bloku používané blokové šifry. (Oppliger, 2021)



Obrázek 3 Schéma zašifrování a rozšifrování módu CFB Zdroj: (Oppliger, 2021)

3.2.4 DES

DES algoritmus byl z počátku považován za silný algoritmus, ale nárůst množství dat a krátká délka jeho klíče omezuje jeho použití. Umí šifrovat a dešifrovat data v 64bitových blocích za použití 64bitového klíče. Protože vždy pracuje na blocích stejné velikosti a používá jak permutace, tak substituce, DES je tedy bloková i složená šifra. (Egerton, 2018)

Algoritmus DES byl optimalizován spíše pro hardware než software. Když vláda Spojených států amerických standardizovala DES, většina cílových aplikací byla implementace na hardware. Není tedy žádné překvapení, že S-boxy v DES jsou malé a rychlé na výpočet, když jsou implementovány jako logický obvod, ale neefektivní v implementaci softwaru. (Aumasson, 2018)

DES je také hlavním představitelem Feistelovy šifry. To je bloková šifra s charakteristickou strukturou. Abeceda je definována $\sum = \mathbb{Z}_2 = \{0, 1\}$ a délka bloku je $2t$ pro přiměřeně velké $t \in \mathbb{N}^+$. Feistelova šifra běží $r \in \mathbb{N}^+$ rund, kde r rundovních klíčů je generováno z $k \in \mathcal{K}$ a používají se na bázi každého kola. (Oppliger, 2021)

Šifrovací funkce E_k nejdříve rozdělí blok zprávy prostého textu m na dvě poloviny, každou velikou t bitů. Nechť L_0 je levá polovina a R_0 je pravá polovina m, to znamená že $m = L_0 \parallel R_0 = (L_0, R_0)$. Pro $i = 1, \dots, r$, se pak rekurzivně vypočítá sekvence párů (L_i, R_i) následovně: (Oppliger, 2021)

$$(L_i, R_i) = \left(R_{i-1}, L_{i-1} \oplus f_{k_i}(R_{i-1})\right) \tag{3}$$

Pár (L_r, R_r) v opačném pořadí pak představuje blok šifrovaného textu. Proto šifrování zprávy prostého textu m pomocí klíče k lze vyjádřit následovně: (Oppliger, 2021)

$$E_k(m) = E_k(L_0, R_0) = (R_r, L_r) \tag{4}$$

Rekurzivní vzorec číslo 3 lze také zapsat jako: (Oppliger, 2021)

$$(L_{i-1}, R_{i-1}) = (R_i) \tag{5}$$

To znamená, že je možné opakovaně vypočítat L_{i-1} a R_{i-1} z L_i, R_i a k_i a k určení (L_0, R_0) z (R_r, L_r) pomocí rundovních klíčů v opačném pořadí (to je $k_r, \dots, k_1$). V důsledku toho lze Feistelovu šifru vždy dešifrovat pomocí stejného šifrovacího algoritmu a použitím rundovních klíčů v opačném pořadí. To značně zjednodušuje implementaci. (Oppliger, 2021)

DES je Feistelova šifra, kde $t = 32$ a $r = 16$. To znamená, že délka bloku DES je 64 bitů, tudíž $M = C = \{0,1\}^{64}$, a že DES šifrující a dešifrující algoritmus operuje v 16 rundách. Kromě toho jsou klíče DES 64bitové řetězce s takovou vlastností, že poslední bit každého bajtu má lichou paritu. To znamená, že součet modulo 2 všech bitů v bajtu musí být lichý a že paritní bit je nastaven odpovídajícím způsobem. To lze vyjádřit jako: (Oppliger, 2021)

$$\mathcal{K} = \left\{(k_1, \dots, k_{64}) \in \{0,1\}^{64} \mid \sum_{i=1}^{8} k_{8j+1} \equiv 1 (mod\ 2)\ pro\ j = 0, \dots, 7\right\} \tag{6}$$

Například klíč „F1DFBC9B79573413“ je validní DES klíč. Jeho lichou paritu je možné ověřit pomocí následující tabulky číslo 1: (Oppliger, 2021)

Tabulka 1 Ověření liché parity klíče

F1	1	1	1	1	0	0	0	1
DF	1	1	0	1	1	1	1	1
BC	1	0	1	1	1	1	0	0
9B	1	0	0	1	1	0	1	1
79	0	1	1	1	1	0	0	1
57	0	1	0	1	0	1	1	1
34	0	0	1	1	0	1	0	0
13	0	0	0	1	0	0	1	1

Zdroj: Vlastní zpracování dle (Oppliger, 2021)

Bit zadaný v posledním sloupci každého řádku představuje paritní bit, který zajišťuje, že každý řádek má lichý počet čísel jedna. V prvním řádku jsou například čtyři ze sedmi bitů rovny jedné, což znamená, že paritní bit musí být také nastaven na hodnotu jedna V důsledku toho prvních sedm bitů bajtu klíče DES určuje poslední bit a proto je počet validních klíčů pro tento algoritmus jen 2^{56} (místo 2^{64}). (Oppliger, 2021)

K zašifrování 64bitového bloku zpráv prostého textu m pomocí 64bitového klíče k funguje algoritmus ve třech krocích: (Oppliger, 2021)

1. Počáteční permutace IP je aplikována na m. Pokud $m = m_1 m_2 m_3 \dots m_{64} \in M = (0,1)^{64}$, pak $IP(m) = m_{58} m_{50} m_{42} \dots m_7 \in M$. To znamená, že 58. bit v m se stane prvním bitem v $IP(m)$, 50. bit se stane druhým bitem $IP(m)$ a tak dále. Výsledné m je poté rozděleno na dvě části: L_0 odkazující na 32 bitů m nejvíce vlevo (označeno $m|^{32}$) a R_0 odkazující na 32 nejvíce pravých bitů z m (označeno $m|_{32}$).
2. Na L_0 a R_0 se pak aplikuje 16rundová Feistelova šifra.
3. Inverzní počáteční permutace IP^{-1} je aplikována na (R_{16}, L_{16}) pro vygenerování bloku šifrovaného textu, to je $c = IP^{-1}(R_{16}, L_{16})$ (Oppliger, 2021)

3.2.5 AES

Z velké části kvůli obavám o bezpečnost předchozího algoritmu DES byl nezbytný nový šifrovací standard, kterým se stal AES. Při vytváření tohoto standardu byl velký nátlak na rychlost a možnou implementaci na zařízeních s nízkou výpočetní kapacitou. Algoritmus Rijndael/AES je silně bajtově orientovaný, díky tomu je efektivní ve vyšších programovacích jazycích, ale také relativně přímočarý k implementaci na procesorech s minimální kapacitou. (Buell, 2021)

AES je bloková šifra se střídáním klíčů, prostý text zpracovává v blocích po 128 bitech pomocí tajného klíče o velikosti 128, 192 nebo 256 bitů, podle délky klíče je poté algoritmus označován jako AES-128, AES-192 a AES-256. (Buell, 2021) Nejběžněji používaným klíčem je klíč délky 128 bitů, protože šifrování mírně urychluje a protože rozdíl mezi 128 a 256bitovým zabezpečením je pro většinu aplikací bezvýznamný. (Aumasson, 2018) AES je iterovaná bloková šifra, protože fixní šifrovací proces, obvykle nazýván runda, je na blok bitů aplikován několikrát. Střídáním klíčů se rozumí, že šifrovací klíč je XORován střádavě s aplikací rundové transformace. (Katz, 2020)

Vstup do šifrovacího či dešifrovacího algoritmu je jediný 128bitový blok. Ten je vyobrazen jako čtvercová matice bajtů o rozměrech 4 x 4. Podobně je klíč znázorněn jako čtvercová matice bajtů. Klíč je poté rozšířen do pole slov takzvaných rundovních klíčů. Každé slovo má čtyři bajty a celý rundovní klíč má 44 slov, pokud je velikost klíče 128 bitů. 16 vstupních bajtů $in_0, \dots, in_{15}$ je nejdříve zkopírováno do stavového pole s před tím, než je něj aplikována počáteční transformace AddRoundKey(). Po závěrečné fázi je zkopírováno stavové pole do výstupní matice. Algoritmus poté vstoupí do smyčky, která iteruje $N_r - 1$ krát, kde N_r je 10, 12 nebo 14 v závislosti na délce použitého klíče. V každé iteraci je použita rundovní funkce, tu lze vidět na obrázku číslo 4 a skládá se z následujících čtyř transformací: (Oppliger, 2021)

1. Při transformaci SubBytes() jsou bajty stavového pole nahrazeny podle dobře definované substituční tabulky
2. V transformaci ShiftRows() jsou řádky stavového pole podrobeny cyklickému posunu (nebo rotaci) o počet bajtů specifický pro každý řádek.
3. Při transformaci MixColumns() jsou sloupce stavového pole podrobeny lineární transformaci.

4. Transformace AddRoundKey() ke stavovému poli přidá rundovní klíč. V zápisu algoritmu $w[i]$ odkazuje na i-té slovo v poli slov rundovních klíčů w a $w[i,j]$ odkazuje na $j-i+1$ slova mezi w_i a w_j v poli slov rundovních klíčů. (Oppliger, 2021)

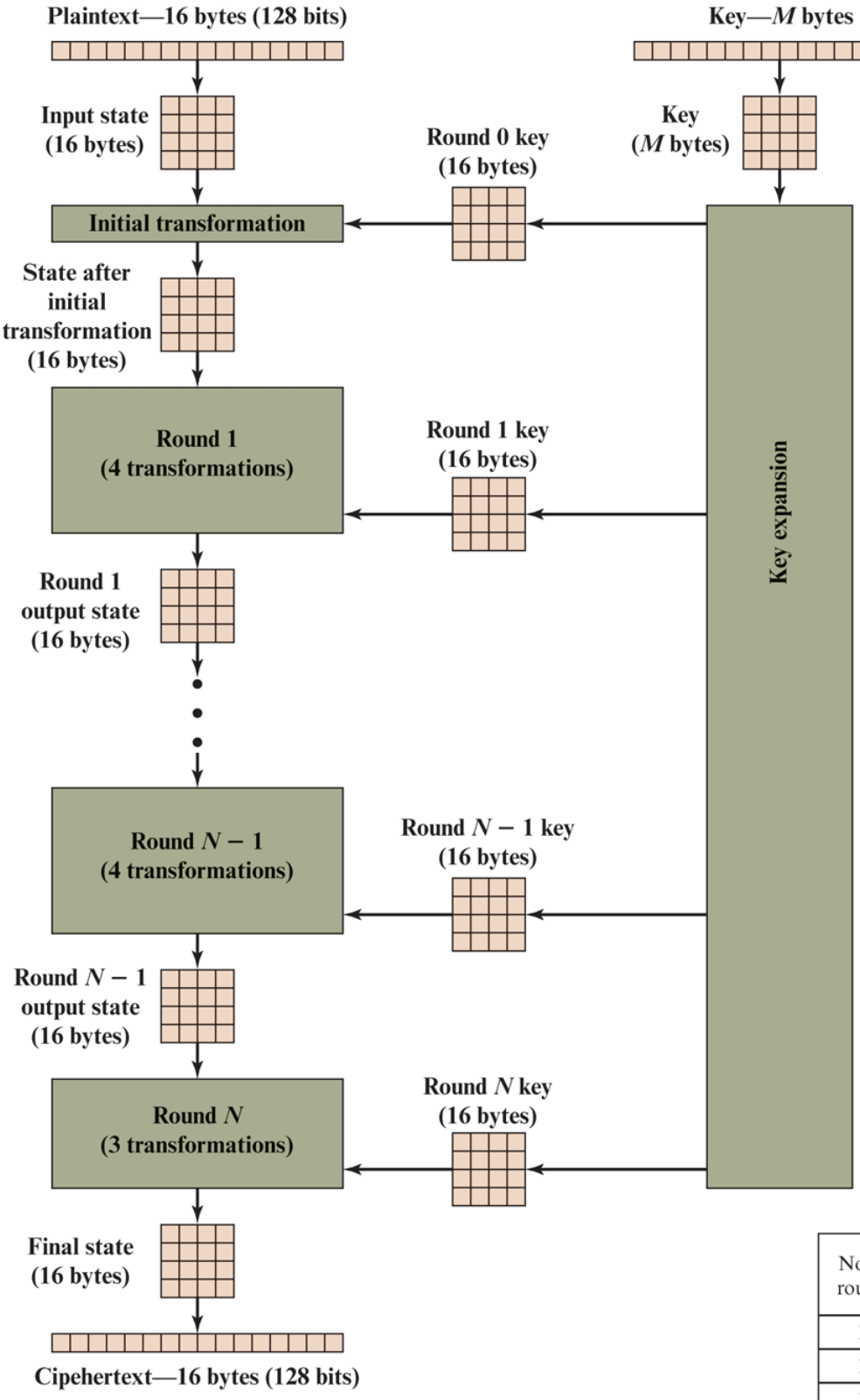
(in)

$s = in$
$s =$ AddRoundKey$(s, w[0, N_b - 1])$
for $r = 1$ **to** $(N_r - 1)$ **do**
  $s =$ SubBytes(s)
  $s =$ ShiftRows(s)
  $s =$ MixColumns(s)
  $s =$ AddRoundKey$(s, w[rN_b, (r+1)N_b - 1])$
$s =$ SubBytes(s)
$s =$ ShiftRows(s)
$s =$ AddRoundKey$(s, w[N_rN_b, (N_r+1)N_b - 1])$
$out = s$

(out)

Obrázek 4 Šifrovací algoritmus AES Zdroj: (Oppliger, 2021)

Transformace SubBytes() a ShiftRows() jsou komutativní, což znamená, že transformace SubBytes() bezprostředně následovaná transformací ShiftRows() je ekvivalentní transformaci ShiftRows() po níž nastává transformace SubBytes(). U všech ostatních transformací tomu tak v tomto případě není. V algoritmu 9.5 je smyčka ukončena po $N_r - 1$ rundách. Pak následuje poslední kolo, které je mírně odlišné, protože neobsahuje transformaci MixColumns(). Nakonec obsah stavového pole představuje výstup šifrování AES a je zkopírován do výstupního pole. Celé schéma algoritmu AES lze vidět na obrázku číslo 5. (Oppliger, 2021)



No. of rounds	Key Length (bytes)
10	16
12	24
14	32

Obrázek 5 Schéma algoritmu AES Zdroj: (Stallings, 2020)

3.3 Současná kryptografie – Asymetrická

V symetrické kryptografii je využíván pouze jeden klíč, který sdílí obě strany. V asymetrické kryptografii je tomu však jinak, zde jsou klíče dva. Jeden se nazývá veřejný klíč, ten je používaný pro zašifrování zprávy, tudíž ho používá strana, která zprávy odesílá. Druhý klíč se nazývá soukromý, ten se používá pouze pro dešifrování. (Aumasson, 2018)

Veřejný klíč je možné spočítat z klíče soukromého, ale naopak to není tak jednoduché. Jinými slovy je jednoduché spočítat klíč jedním směrem, ale tím druhým již ne, odtud tedy název „Asymetrická kryptografie“. (Katz, 2020)

Bezpečnost asymetrické kryptografie je závislá na principu prvočísel. Prvočísla jsou celá čísla, která mají v principu jednu velmi jednoduchou aritmetickou vlastnost. Číslo je prvočíslo právě tehdy, když je beze zbytku dělitelné jen číslem 1 a prvočíslem samotným. Nejmenším prvočíslem je číslo 2, které je jediným sudým číslem a zároveň prvočíslem. Prvočísla jsou v jistém smyslu základními stavebními kameny pro všechna celá čísla, protože každé celé číslo lze vytvořit vynásobením některých prvočísel dohromady. Například číslo 100 je možné získat pouze vynásobením prvočísel 2, 2, 5 a 5, žádná jiná skupina prvočísel násobením nezíská výsledek 100. Toto jedinečné spojení mezi číslem a jeho prvočíselným rozkladem se nazývá faktorizace a tvoří základ jednoho z nejslavnějších asymetrických šifrovacích algoritmů, RSA. (Martin, 2020)

3.3.1 RSA

RSA je od doby svého vzniku považována jako nejrozšířenější a nejvíce implementovaný přístup k asymetrickému šifrování s univerzálním přístupem. Podporuje šifrování a digitální podpisy. (Stallings, 2020) Schéma RSA je šifra, ve které jsou prostým textem a šifrovaným textem celá čísla mezi 0 a n-1 pro nějaké n. Typická velikost pro n je 1024 bitů nebo 309 dekadických číslic. To znamená, že n je menší než 2^{1024}. (Egerton, 2018)

Prostý text je zašifrován v blocích, přičemž každý blok má binární hodnotu menší než nějaké číslo n. To znamená, že velikost bloku musí být menší nebo rovna $log_2(n) + 1$ v praxi velikost bloku je i bitů, kde $2^i < n \leq 2^{i+1}$. Šifrování a dešifrování mají pro bloky prostého textu M a bloky šifrovaného textu C následující formu. (Stallings, 2020)

$$C = M^e \bmod n \tag{7}$$

$$M = C^d \bmod n = (M^e)^d \bmod n = M^{ed} \bmod n \tag{8}$$

Odesílatel i příjemce musí znát hodnotu n. Odesílatel zná hodnotu e a pouze příjemce zná hodnotu d. Jedná se tedy o šifrovací algoritmus s veřejným klíčem $PU = \{e, n\}$ a soukromým klíčem $PR = \{d, n\}$. Aby byl tento algoritmus pro šifrování veřejným klíčem uspokojivý, musí být splněny následující požadavky. (Stallings, 2020)

1. Je možné najít hodnoty e, d a n takové, že $M^{ed} \bmod n = M$ pro všechna $M < n$
2. Je relativně snadné vypočítat $M^e \bmod n$ a $C^d \bmod n$ pro všechny hodnoty $M < n$
3. Je nemožné určit d pro dané e a n (Stallings, 2020)

Vztah $M^{ed} \bmod n = M$ platí, pokud e a d jsou modulární inverze $\phi(n)$, kde $\phi(n)$ je Eulerova totientová funkce. Pro prvočísla p a q platí $\phi(pq) = (p-1)(q-1)$. Vztah mezi e a d lze vyjádřit jako: (Stallings, 2020)

$$ed \bmod \varphi(n) = 1 \tag{9}$$

Z toho vyplývá

$$ed \equiv 1 \bmod \varphi(n) \tag{10}$$

$$d \equiv e^{-1} \bmod \varphi(n) \tag{11}$$

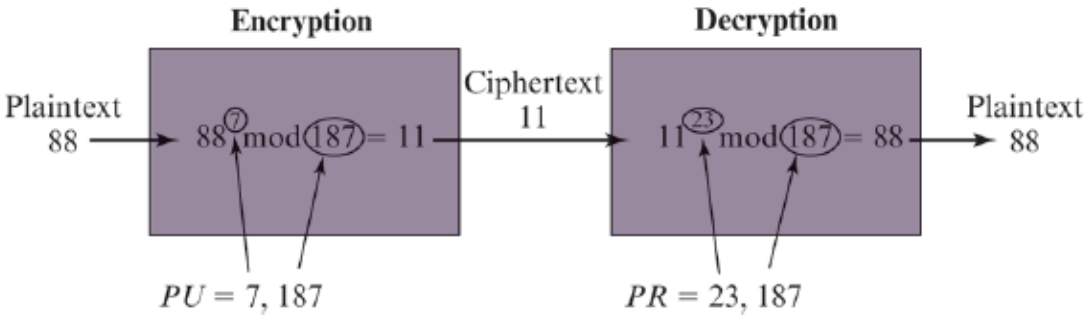
To znamená, že e a d jsou modulární inverzí $\bmod \phi(n)$. Podle pravidel modulární aritmetiky to platí pouze v případě, že d (a tedy i e) je nesoudělné číslo k $\phi(n)$. Stejně platí $\gcd(\phi(n), d) = 1$. (Stallings, 2020)

Složky pro schéma RSA jsou následující:

- dvě prvočísla p a q, která jsou soukromá a vybraná
- číslo n, které je násobkem prvočísel p a q, to je veřejné a vypočtené
- e, pro které platí $gcd(\varphi(n), e) = 1$ a $1 < e < \varphi(n)$, je veřejné a vybrané
- $d \equiv e^{-1} (\bmod \varphi(n))$, které je soukromé a vypočtené (Stallings, 2020)

Soukromý klíč se skládá z $\{d, n\}$ a veřejný klíč se skládá z $\{e, n\}$. Předpokládejme, že uživatel A zveřejnil svůj veřejný klíč a že uživatel B si přeje poslat zašifrovanou zprávu M uživateli A. Poté uživatel B vypočítá $C = M^e \bmod n$ a odešle C. Po přijetí tohoto šifrovaného textu ho uživatel A dešifruje výpočtem $M = C^d \bmod n$. (Stallings, 2020)

Příklad RSA algoritmu lze vyjádřit tímto obrázkem: (Stallings, 2020)



Obrázek 6 Příklad algoritmu RSA Zdroj: (Stallings, 2020)

Pro tento příklad byly klíče vygenerovány následovně: (Stallings, 2020)

1. Byly vybrány dvě prvočísla, $p = 17$ a $q = 11$
2. Vypočetlo se $n = pq = 17 \times 11 = 187$
3. Vypočetlo se $\phi(n) = (p - 1) \times (q - 1) = 16 \times 10 = 160$
4. Vybralo se e takové, že je nesoudělné s $\phi(n) = 160$ a je menší než $\phi(n)$, v tomto příkladu bylo zvoleno $e = 7$.
5. Určilo se d takové, že $de \equiv 1\ (mod\ 160)$ a $d < 160$. Správná hodnota je $d = 23$, protože $23 \times 7 = 161 = (1 \times 160) + 1$. d je možné vypočítat například pomocí rozšířeného Euklidova algoritmu. (Stallings, 2020)

Výsledkem je poté veřejný klíč $PU = \{7, 187\}$ a soukromý klíč $PR = \{23, 187\}$. Na obrázku číslo 6 je vyobrazeno využití těchto klíčů pro vstupní nezašifrovaný text $M = 88$. Pro zašifrování je potřeba vypočítat $C = 88^7\ mod\ 187$. Využitím vlastností modulární aritmetiky je to možné následovně. (Stallings, 2020)

$$88^7\ mod\ 187 = [(88^4\ mod\ 187) \times (88^2\ mod\ 187) \times (88^1\ mod\ 187)]\ mod\ 187$$

$$88^1\ mod\ 187 = 88$$

$$88^2\ mod\ 187 = 7744\ mod\ 187 = 77$$

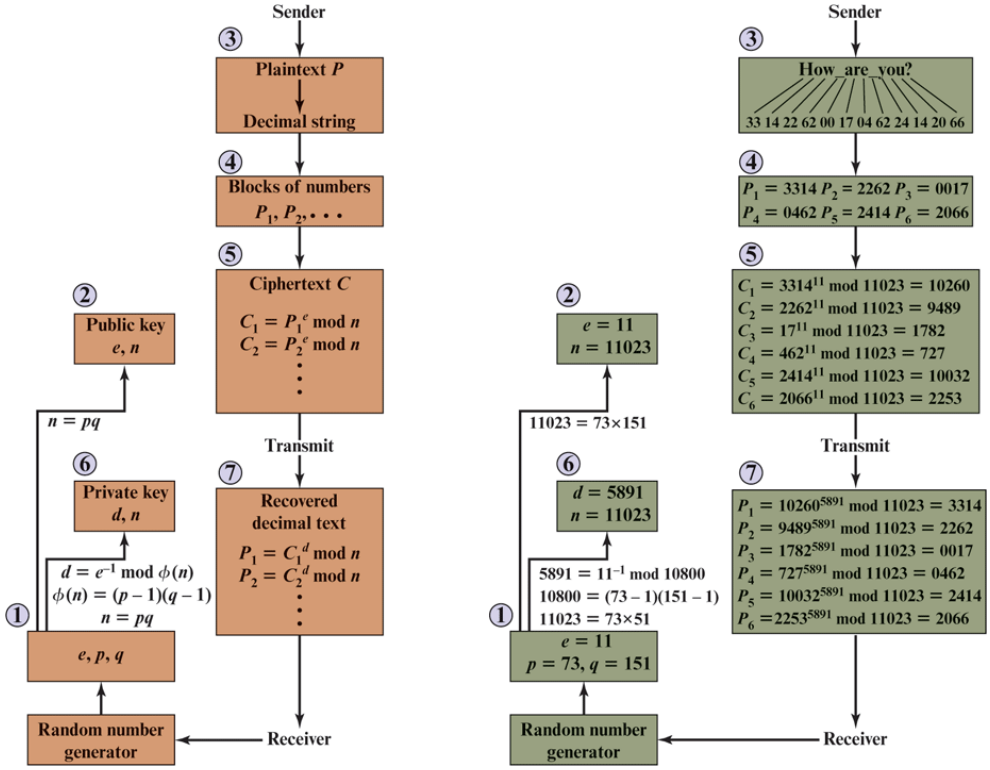
$$88^4\ mod\ 187 = 59\,969\,536\ mod\ 187 = 132$$

$$88^7\ mod\ 187 = (88 \times 77 \times 132)\ mod\ 187 = \ 894\,432\ mod\ 187 = 11$$

Pro dešifrování je vypočítáno $M = 11^{23} mod\ 187$: (Stallings, 2020)

$$11^{23}\ mod\ 187$$
$$= [(11^8\ mod\ 187) \times (11^8\ mod\ 187) \times (11^4\ mod\ 187)$$
$$\times (11^2\ mod\ 187) \times (11^1\ mod\ 187)]\ mod\ 187$$
$$11^1\ mod\ 187 = 11$$
$$11^2\ mod\ 187 = 121$$
$$11^4\ mod\ 187 = 14641\ mod\ 187 = 55$$
$$11^8\ mod\ 187 = 214358881\ mod\ 187 = 33$$
$$11^{23}\ mod\ 187 = (33 \times 33 \times 55 \times 121 \times 11)\ mod\ 187 = 79\,720\,245\ mod\ 187 = 88$$

Tento příklad znázorňuje zašifrování a dešifrování jednoho bloku, avšak v reálném použití je skoro vždy potřeba využití algoritmu na větší objem dat. Tuto variantu znázorňuje obrázek číslo 7. Čísla v kroužcích znázorňují, v jakém pořadí jsou kroky prováděny, část (a) zobrazuje obecný postup, část (b) znázorňuje příklad s textem „How are you?“: (Stallings, 2020)



Obrázek 7 Schéma a příklad algoritmu RSA Zdroj: (Stallings, 2020)

3.3.2 Hybridní kryptografie

Asymetrické šifrování je využíváno hlavně k šifrování menších objemů dat, to je důsledkem toho, že mezi jeho nevýhody se řadí nižší rychlost. Asymetrické šifrování však využívá i technika zvaná hybridní kryptografie, ta v podstatě spojuje výhody asymetrického a symetrického šifrování dohromady. (Stinson, 2019)

Pro příklad: „Alice chce Bobovi poslat dlouhou šifrovanou zprávu, ale ještě si nesdělili soukromý klíč. Alice si tedy zvolí náhodný soukromý klíč a rychlejší metodou zašifruje svou zprávu. Poté Alice zašifruje použitý soukromý klíč Bobovým veřejným klíčem. Alice následně pošle zašifrovanou zprávu a zašifrovaný klíč Bobovi, ten nejdříve svým soukromým klíčem rozšifruje klíč, který následně použije k rozšifrování zprávy, kterou mu Alice chtěla původně poslat.“ (Stinson, 2019)

Pomalejší asymetrické šifrování tedy bylo využito pouze pro zašifrování krátkého soukromého klíče a jeho bezpečné doručení k Bobovi. Poté je tento klíč použit rychlejším algoritmem symetrického šifrování k rozšifrování dlouhé zprávy. Hybridní kryptografie tedy skoro dosahuje účinnosti asymetrické kryptografie. (Stinson, 2019)

3.4 Postkvantová kryptografie

Oblast postkvantové kryptografie je o navrhování asymetrických algoritmů, které nelze prolomit kvantovým počítačem a byly by schopné nahradit v budoucnu RSA a algoritmy založené na eliptických křivkách. (Stănică, 2021)

Tyto algoritmy by se neměly spoléhat na obtížný problém, o kterém je známo, že je efektivně řešitelný pomocí Shorova algoritmu, který eliminuje obtížnost faktorizace a diskrétního logaritmu. Symetrické algoritmy jako jsou blokové šifry a hašovací funkce by tváří v tvář kvantovému počítači ztratily pouze polovinu své teoretické bezpečnosti, ale nebyly by prolomeny tak nebezpečně jako RSA. Mohly by tedy tvořit základ pro postkvantové schéma. (Aumasson, 2018)

Postkvantová kryptografie může být zásadně silnější než RSA nebo kryptografie nad eliptickými křivkami, ale není neomylná ani všemocná. (Aumasson, 2018)

Existuje několik problémů, u kterých se zatím nepodařilo nalézt algoritmus, který by běžel výrazně rychleji na kvantových počítačích oproti počítačům klasickým, nekvantovým. Tudíž se předpokládá, že by příchod kvantových počítačů nerozbil bezpečnost algoritmů založených na těchto problémech. (Bernstein, 2009)

3.4.1 Princip kvantových počítačů

Kvantové počítače jsou počítače, využívající jevů z kvantové fyziky ke spouštění různých druhů algoritmů, než na jaké jsme zvyklí. Kvantové počítače zatím neexistují a jejich sestavení vypadá velmi obtížně, ale pokud jednoho dne budou existovat, budou mít potenciál prolomit šifry jako RSA, Diffie-Hellman a kryptografii nad eliptickými křivkami, tedy veškerou asymetrickou kryptografii. (Aumasson, 2018) Již také existují útoky na současnou kryptografii za použití kvantových počítačů. Avšak kvantová kryptografie neposkytuje úplnou náhradu za asymetrickou kryptografii. Například ji není možné použít k implementaci digitálního podpisu, v důsledku toho je to nanejvýš záložní technologie pro některé části kryptografie s veřejným klíčem. (Wong, 2021)

Tradiční počítač používá dlouhé řetězce bitů, které mohou nabývat hodnot nula nebo jedna. Kvantový počítač na rozdíl od toho používá kvantové bity nebo qubity. (Oppliger, 2021) Qubit je kvantový systém, který kóduje nulu a jedničku do dvou rozlišitelných kvantových stavů. Protože se ale qubity chovají kvantově, můžeme využít jevů superpozice a provázání. Superpozice je v podstatě schopnost kvantového systému být ve více stavech současně, takže něco může být „tady“ a „tam“ nebo „nahoře“ a „dole“ zároveň. Provázání je extrémně silná korelace, která existuje mezi kvantovými částicemi, ve skutečnosti tak silná, že dvě nebo více kvantových částic může být nerozlučně spojeno, i když jsou odděleny na velké vzdálenosti. Díky těmto dvěma jevům dokáže kvantový počítač zpracovat obrovské množství výpočtů současně. (Stinson, 2019)

3.4.2 Algoritmy založené na kódu

Algoritmy založené na kódu jsou založeny na kódech opravující chyby, což jsou techniky navržené pro přenos bitů přes šumový kanál. Základní teorie pro opravu chyb pochází z 50. let 20. století. Toto šifrovací schéma lze použít jak pro šifrování, tak pro digitální podpisy. Jeho hlavním omezením je velikost veřejného klíče, která se pohybuje v řádu sta kilobajtů. (Aumasson, 2018)

Kód opravující chyby se označuje jako (n, k) ECC. Šifrování lze popsat jako násobení k-bitového datového vektoru m s $k \times n$ maticí G, čímž se získá n-bitový vektor kódového slova c: (Stallings, 2020)

$$c = mG \tag{12}$$

Pro každou generující matici existuje $(n-k) \times k$ paritní kontrolní matice H, která má řady kolmé k řádkům matice G, to znamená $GH^T = 0$. Kódové slovo, ať už uložené nebo přenášené, podléhá poškození, které může v bloku způsobit jednu nebo více bitových chyb. V cíli může přijaté kódové slovo obsahovat chyby. Tento blok prochází dekodérem ECC s jedním ze čtyř možných výsledků: (Stallings, 2020)

- Žádné chyby: Pokud nejsou nalezeny žádné bitové chyby, vstup do dekodéru ECC je identický se originálním kódovým slovem a dekodér vytváří původní datový blok jako výstup.
- Detekovatelné, opravitelné chyby: U určitých vzorců chyb je možné, aby dekodér tyto chyby detekoval a opravoval. I když se příchozí datový blok liší od přenášeného kódového slova, dekodér je schopen mapovat tento blok do původního datového bloku.
- Detekovatelné, neopravitelné chyby: U určitých vzorců chyb může dekodér chyby detekovat, ale neopravovat je. V tomto případě dekodér hlásí neopravitelnou chybu.
- Nedetekovatelné chyby: U určitých, obvykle vzácných, vzorců chyb dekodér nezjistí, že se jedná o chybu a namapuje příchozí n-bitový blok dat na k-bitový blok, který se však liší od původního k-bitového bloku. (Stallings, 2020)

Korekce chyb funguje přidáním dostatečné redundance do datového bloku. Redundance umožňuje přijímači odvodit jaký byl původní blok, a to i při určité úrovni chybovosti. (Stallings, 2020)

3.4.3 Algoritmy založené na mřížkách

Algoritmy založené na mřížkách jsou velkým příslibem pro postkvantovou kryptografii, protože mají velmi silné bezpečnostní důkazy založené na složitosti nejhoršího případu, relativně efektivní implementaci a také jsou velmi jednoduché. (Bernstein, 2009)

m-rozměrná mřížka úrovně n je množina vektorů, které lze vyjádřit jako součet celočíselných násobků konkrétní množiny n vektorů, souhrnně nazývaných základ mřížky. Formálně lze mřížku definovat jako: (Stallings, 2020)

$$L = \left\{ \sum_{i=1}^{n} x_i b_i | n_i \in \mathbb{Z}, b_i \in \mathbb{R}^m \right\} \tag{13}$$

Kde b_i jsou lineárně nezávislé vektory délky m nad reálnými čísly a x_i jsou celá čísla. Množina vektorů b_i se nazývá základ mřížky. Základ mřížky může být reprezentován maticí B, kde i-tý sloupec matice je b_i. Označujeme m jako rozměr mřížky a n hodnost mřížky. Mřížka může být zobrazena jako n bodů definovaných základem v m-rozměrném prostoru. To znamená, že každý bod je koncovým bodem jednoho z vektorů základu. O mřížce se dá říct, že je plnohodnotná, když $n = m$. Existuje nekonečně mnoho mřížek stejných rozměrů. (Stallings, 2020)

Základ pro danou mřížku není unikátní. Existence více základů pro stejnou mřížku je důležitá pro vývoj kryptografických algoritmů, protože s některými základy se manipuluje snadněji než s jinými. (Stănică, 2021)

Kryptografické konstrukce založené na mřížkách jsou založeny na předpokládané složitosti mřížkových problémů, z nichž nejzákladnější problém je problém s nejkratším vektorem. Při tomto problému je jako vstup dána mřížka reprezentovaná libovolným základem a cílem je v ní vyhledat nejkratší nenulový vektor. Ve skutečnosti se typicky uvažuje o aproximační variantě problému s nejkratším vektorem, kde je cílem vyhledat vektor, jehož délka je nanejvýš nějaký aproximační faktor $\gamma(n)$ krát délka nejkratšího nenulového vektoru, kde n je rozměr mřížky. (Bernstein, 2009)

3.4.4 Algoritmy založené na vícerozměrných polynomech

Vícerozměrná schémata jsou založena na obtížnosti řešení systémů vícerozměrných kvadratických polynomů nad konečnými poli. Termín vícerozměrný polynom se odkazuje na polynom ve více než jedné proměnné a termín kvadratický polynom se týká polynomu stupně 2. (Bernstein, 2009) Obecně lze tato schémata popsat následovně. Veřejný klíč se skládá ze sady m polynomů: (Stallings, 2020)

$$P(x_1, \dots, x_n) = (p_1(x_1, \dots, x_n), p_2(x_1, \dots, x_n), \dots, p_m(x_1, \dots, x_n)) \tag{14}$$

Který lze rozšířit na následující: (Stallings, 2020)

$$p_1(x_1, \dots, x_n) = \sum_{i=1}^{n} \sum_{j=1}^{n} p_{1,ij} x_i x_j + \sum_{i=1}^{n} p_{1,i} x_i + p_{1,0} \tag{15}$$

$$p_2(x_1, \dots, x_n) = \sum_{i=1}^{n} \sum_{j=1}^{n} p_{2,ij} x_i x_j + \sum_{i=1}^{n} p_{2,i} x_i + p_{2,0} \tag{16}$$

$$p_m(x_1, \dots, x_n) = \sum_{i=1}^{n} \sum_{j=1}^{n} p_{m,ij} x_i x_j + \sum_{i=1}^{n} p_{m,i} x_i + p_{m,0} \tag{17}$$

Přičemž m se rovná počtu rovnic a n se rovná počtu proměnných. Obecně se šifrování pomocí veřejného klíče provádí následovně. Vzhledem k danému prostému textu $m = (y_1, \dots, y_n)$ je šifrovaný text: (Stallings, 2020)

$$P(m) = \big(p_1(y_1, \dots, y_n), p_2(y_1, \dots, y_n), \dots, p_m(y_1, \dots, y_n)\big) = (c_1, \dots, c_m) \tag{18}$$

Soukromý klíč je mapování P^{-1} a poskytuje prostý text: (Stallings, 2020)

$$(y_1, \dots, y_n) = P^{-1}(c_1, \dots, c_m) \tag{19}$$

Předpokládá se, že vzhledem k P je obtížné najít P^{-1}, ale naopak ne. Přesněji řečeno, bezpečnost schématu závisí na obtížnosti následujícího problému. Pro dané $P(x_1, \dots, x_n)$ najít vektor $(z_1, \dots, z_n)$ takový, že $P(z_1, .., z_n) = 0$. (Stallings, 2020)

3.4.5 Algoritmy pro digitální podpisy založené na bázi hašů

Předpokládejme hašovací funkci, která vytváří b-bitovou hašovací hodnotu. Tedy pro SHA-256 platí $b = 256$. V Lamportově schématu je veřejný a soukromý klíč použit pro danou zprávu m pouze jednou. Zapojené kroky jsou následující: (Stallings, 2020)

1. Vypočte se b-bitová hašovací hodnota $H(m)$.
2. Vygenerují se $2b$ tajné bitové řetězce, dva pro každou bitovou pozici k v $H(m)$, $S_{0,k}$ a $S_{1,k}$. Množina tajných hodnot tvoří soukromý klíč.
3. Veřejný klíč se skládá z hašovaných hodnot každé tajné hodnoty: $H\big(S_{0,k}\big), H\big(S_{1,k}\big), k = 1, \dots, b$.
4. Digitální podpis se skládá z poloviny vypočtených hašovaných hodnot v kroku 3. Pro blok m je podpis generován následovně. Pokud k-tý bit $H(m)$ je 0, pak k-tý prvek

podpisu je $S_{0,k}$. Pokud k-tý bit $H(m)$ je 1, pak k-tý prvek podpisu je $S_{1,k}$. Podpis tedy odhalí polovinu soukromého klíče.

5. Ověření podpisu zahrnuje následující: Ověřovatel vypočítá $H(m)$. Poté se bity $H(m)$ použijí k výběru odpovídajících prvků veřejného klíče. Pokud je tedy k-tý bit 1, vybere se $H(S_{1,k})$. Poté jsou hašovací hodnoty b v podpisu porovnány s hašovacími hodnotami b vybranými z veřejného klíče. Pokud se všechny shodují, podpis je ověřen. (Stallings, 2020)

Toto schéma má řadu nevýhod. Podpis zprávy odhalí polovinu soukromého klíče. To nestačí k tomu, aby útočník mohl podepsat další zprávy s různými hašovanými zprávami, ale nebylo by bezpečné použít tento pár klíčů více než jednou. Dále mají veřejný i soukromý klíč značnou délku. (Stallings, 2020)

3.4.6 XSynd

XSynd je upravenou efektivní variantou proudové šifry Synd. Tato varianta řeší problémy, které byly známé u varianty předchozí a vyžaduje poměrně malou kapacitu úložiště, díky tomu je velmi atraktivní pro praktickou implementaci. (Meziani, 2020)

Původní Synd šifra je vylepšenou variantou Fisher-Sternova generátoru pseudonáhodných čísel. Vylepšením jsou použití částečně cyklických kódů, které snižuje potřebnou kapacitu úložiště a použití regulárních slov, což zrychluje generování klíčového proudu. Tento generátor pseudonáhodných čísel lze považovat za konečný automat, S, určený sadou vnitřních stavů s délkami od 256 do 1024 bitů. Šifra Synd přijímá klíče délky od 128 do 512 bitů a v každém kole vytváří dvakrát větší proud bitů než je velikost klíče. (Meziani, 2020)

Generování proudu bitů funguje ve třech krocích za použití tří odlišných funkcí nazvaných Ini, Upd a Out. Funkce Ini vezme tajný klíč K spojený s počátečním vektorem IV, oba o délce $r/2$ bitů a vrací počáteční stav $e_0 = Ini(K|IV)$, který spustí proces generování proudu bitů. Funkce Ini je trojitá Feistelova transformace založena na funkcích Upd a Out a je dána: (Meziani, 2020)

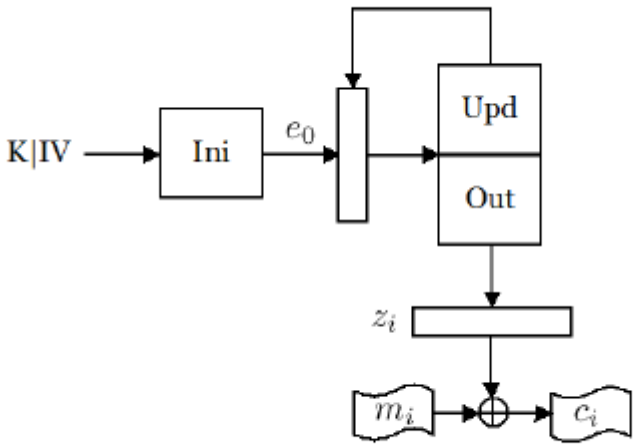
$$Ini(x) = y \oplus Out(x \oplus Upd(y));\ y = x \oplus Upd(y);\ \forall x\ = (K, IV) \in\ \mathbb{F}_2^{r/2} \times\ \mathbb{F}_2^{r/2} \qquad (20)$$

Kde funkce Upd a Out jsou definovány jako: (Meziani, 2020)

$$Upd(x) = A \cdot \theta(x); Out(x) = B \cdot \theta(x), \forall x \in \mathbb{F}_2^r \tag{21}$$

Zde jsou A a B náhodné binární matice, které popisují stejný skoro cyklický kód délky n opravující až w chyb. Mapování $x \rightarrow \theta(x)$ je kódovací algoritmus, který transformuje r-bitový řetězec na regulární slovo o délce n a hmotnosti w. Počínaje e_0 v každé časové jednotce i, vydává S klíčový bit $z_i = Out(e_i)$ a mění vnitřní stav následovně: $e_{i+1} = Upd(e_i)$. (Meziani, 2020)

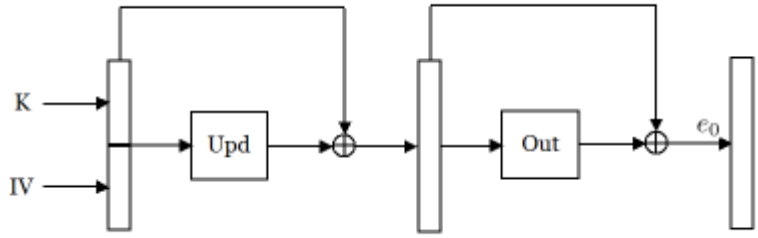
Po vygenerování klíčového proudu bitů $z_0, z_1, \ldots$ je proud bitů prostého textu $m_0, m_1, \ldots$ zašifrován do proudu šifrovaného textu $c_0, c_1, \ldots$ pomocí bitového operátoru XOR jako $c_i = z_i \oplus m_i$. Při znalosti tajného stavu e_0 může příjemce generovat klíčový proud bitů a proto může obnovit proud bitů prostého textu pomocí $m_i = z_i \oplus c_i$. (Meziani, 2020)



Obrázek 8 Schéma proudové šifry Synd Zdroj: (Meziani, 2020)

Šifra XSynd pak upravuje funkci Ini tak, že vyžaduje pouze dvě vyhodnocení funkcí, oproti třem, bez ztráty bepečnosti. Nová funkce je označena jako $XIni$. Tato modifikace neovlivňuje získání tajného K nebo počátečního vektoru IV. Nová funkce $XIni$ je definována takto: (Meziani, 2020)

$$XIni(x) = y \oplus Out(y);\ y = x \oplus Upd(x), \forall x\ = (K, IV) \in\ \mathbb{F}_2^{r/2} \times\ \mathbb{F}_2^{r/2} \qquad (22)$$



Obrázek 9 Schéma upravené funkce XIni šifry XSynd Zdroj: (Meziani, 2020)

Druhou modifikací XSynd je vyhnutí se běžnému kódování $x \rightarrow \theta(x)$ ve funkcích Upd a Out pomocí paradigmatu znáhodnění a poté sloučení. Přesněji řečeno, je-li dán vstup x skládající se z w bloků, přičemž každý blok je velký b bitů, kde b je zvoleno libovolně, nejprve každý blok projde náhodnou funkcí f, čímž se získá výstup y_i. Hodnoty $y_1, y_2, \dots, y_w$ se zkombinují pomocí bitového operátoru XOR pro vytvoření konečného výstupu. (Meziani, 2020)

V XSynd je použita následující funkce f: nechť H je náhodná binární matice o velikosti $wb \times w \cdot 2^b$, skládající se z w podmatic o velikosti $wb\ \times\ 2^b$. Pokud se zápis pro tyto podmatice změní jako $H_i = (h_i^{(0)}, h_i^{(1)}, \dots, h_i^{(2^b-1)})$, kde $h_i^{(j)} \in \mathbb{F}^{wb}$ pro $j \in \{0, 1, \dots, 2^b - 1\}$, pak lze definovat f pomocí $y_i = h_i^{(j)}$ tehdy a pouze tehdy pokud desetinná hodnota x_i je rovna j. Existuje 2^b možných hodnot pro každé y_i, v závislosti na desetinné hodnotě bloku x_i. Tímto způsobem jsou funkce Upd a Out předefinovány jako: (Meziani, 2020)

$$Upd(x) = a_1^{(x_1)}\ a_2^{(x_2)}\ \dots\ a_w^{(x_w)}; Out(x) = b_1^{(x_1)}\ b_2^{(x_2)}\ \dots\ b_w^{(x_w)};\ \forall x \in \mathbb{F}_2^{wb} \qquad (23)$$

3.4.7 ChaCha20

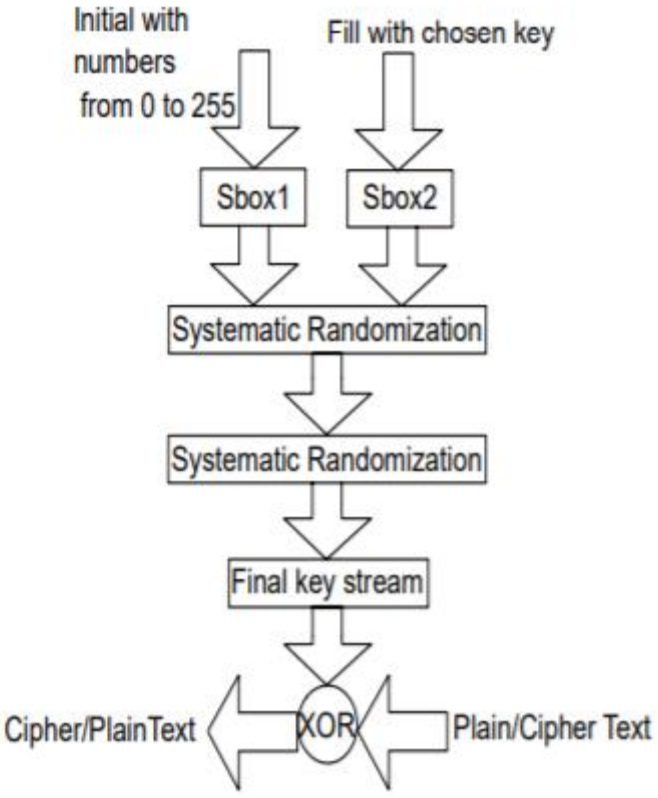
ChaCha20 je proudová šifra, která generuje proud bitů aplikací blokové funkce na klíč, nonce a čítač bloků. Prostý text je poté zašifrován pomocí tohoto proudu bitů, přičemž blok i prostého textu a výstup blokové funkce ChaCha20, vyhodnocený použitím blokového čítače i, se zkombinují pomocí bitového operátoru XOR. (Yadav, 2016)

Bloková funkce ChaCha20 bere jako vstup klíč o velikosti 32 bajtů, 4bajtové blokové číslo a nonce o velikosti 12 bajtů, Výstupem je 64 pseudonáhodných bajtů. (Yadav, 2016)

3.4.8 Arcfour

Arcfour je proudová šifra se symetrickým klíčem. Stejný algoritmus je použit pro šifrování i rozšifrování, protože vstupní datový je jednoduše XORován s vygenerovanou sekvencí klíčů. Proud bitů je zcela nezávislý na použitém prostém textu. K inicializaci 256bitové stavové tabulky se používá klíč s proměnlivou délkou od 1 do 256 bitů. Stavová tabulka se používá pro následné generování pseudonáhodných bitů a poté pro generování pseudonáhodného proudu, který je pomocí bitového operátoru XOR zkombinován s prostým textem, aby se získal šifrovaný text. (Goutam, 2019)

Algoritmu lze rozdělit do dvou fází, a to inicializace a provoz. V inicializační fázi je naplněna 256bitová stavová tabulka *S* pomocí klíče *K* jako takzvané semínko. To je vstupem do generátoru pseudonáhodných čísel a zajišťuje vrácení jiné sekvence pseudonáhodných dat oproti jiným semínkům. Jakmile je tabulka stavů nastavena, pokračuje se v jejím pravidelném upravování, společně s tím jak jsou data šifrována. (Goutam, 2019)



Obrázek 10 Šifrovací algoritmus Arcfour Zdroj: (Goutam, 2019)

Kroky algoritmu Arcfour lze popsat následovně: (Goutam, 2019)

1. Získání vstupních dat a vybraného klíče.
2. Vytvoření dvou polí řetězců.
3. Inicializace jednoho pole s čísly od 0 do 255.
4. Vyplnění druhého pole vybraným klíčem.
5. Náhodné rozdělení prvního pole v závoslosti na poli s klíčem.
6. Náhodné rozdělení prvního pole uvnitř sebe, aby se vygeneroval proud bitů.
7. Sloučení proudu bitů pomocí bitového operátoru XOR s daty, která se mají zašifrovat nebo rozšifrovat. (Goutam, 2019)

4 Vlastní práce

Vlastní práce je zaměřena na popis použitého hardwaru, softwaru a souborů použitých při měření délky běhu kryptografických metod. Je popsáno i ovládání a výpis těchto programů. Následně jsou sepsány výsledky měření, které jsou přepočítány na rychlost jednotlivých metod. Podle této rychlosti jsou metody dále porovnány.

4.1 Popis použitého hardwaru a softwaru

Pro testování výkonnosti kryptografických metod je užíván autorův osobní počítač. V případě postkvantových metod je použit software dostupný zdarma stažený z internetu a pro moderní metody je využit autorem napsaný program.

4.1.1 Detail hardwaru

Rychlost kryptografických algoritmů je velice závislá na hardwaru, na kterém je algoritmus spouštěn. Z tohoto důvodu jsou všechny algoritmy spouštěny na právě jednom zařízení, aby tento vliv neměl dopad na výsledky.

V tabulce 2 je popsán hardware, jenž je využit při spouštění programů využitých ve vlastní práci.

Tabulka 2 Hardware komponenty systému, který spouštěl programy

Komponenty	Parametry
Procesor	Intel Core i7-8550U
Běžný takt procesoru	1,8 GHz
Takt procesoru v režimu Boost	4 GHz
Počet jader	4
Operační paměť	8 GB

Zdroj: (Vlastní zpracování)

4.1.2 Popis softwaru Codecrypt

Program Codecrypt využívá mnoho tradičních, avšak kvantově bezpečných kryptografických základů. Volby autorů programu právě těchto základů byly založeny na poskytované bezpečnosti, jednoduchosti a auditovatelnosti designu. Program Codecrypt je

volně dostupný na platformě GitHub pro platformy Gentoo, Debian, Arch linux a také Windows.

Implementace postkvantových šifrovacích metod je vyřešena programem Codecrypt ve verzi 1.8 portable pro Windows. Tato verze je ovládána převážně přes příkazovou řádku a v jejím modulu jsou obsaženy šifrovací metody Chacha20, XSynd a Arcfour.

4.1.3 Popis autorova softwaru

Autorem napsané programy jednotlivě implementují moderní šifrovací metody AES, DES a RSA. Programy jsou napsány v jazyce Python, v době psaní byla aktuální verze 3.10.2, s využitím balíčků zlehčujících implementaci kryptografie pro tento jazyk. Pro realizaci metod AES a DES je využit balíček PyCryptodome verze 3.14.1, jenž je nainstalován pomocí správce balíčků pip pomocí příkazu „pip install pycryptodome“. Pro tyto metody je také využita knihovna secrets, která je součástí jazyka Python. Tato knihovna je využita ke generování klíčů.

Pro metodu RSA je použit balíček Python-RSA nainstalovaný také podle správce balíčků pip pomocí příkazu „pip install rsa“. Všechny metody dále používají knihovny sys a datetime, které jsou součástí jazyka Python. Knihovna sys umožňuje programům přijímat argumenty. V tomto případě první a jediný argument, který program přijímá je název souboru, na který má program aplikovat šifrovací metodu. Díky knihovně datetime je možné sledovat délku běhu šifrovacího algoritmu.

U programů pro metody RSA a DES je nutné rozdělit text na více částí, jinak ho balíček nezvládne zpracovat. Zdrojový kód pro jednotlivé programy lze nalézt v přílohách číslo 1, 2 a 3.

4.1.4 Popis souborů použitých k šifrování

Soubory, jež byly použity během šifrování byly náhodně vygenerovány webovou stránkou onlinefiletools.com. V souborech jsou obsažena pouze velká a malá písmena latinské abecedy a pro testování doby běhu algoritmů souborů jsou použity soubory velikostí 1500 bajtů, 10 kilobajtů, 100 kilobajtů, 500 kilobajtů, 1 megabajt, 10 megabajtů, 100 megabajtů a 500 megabajtů.

4.2 Spuštění a běh softwaru

Spuštění a běh softwaru je popsán pouze na platformě Windows a funguje v příkazové řádce Command prompt i Windows PowerShell.

4.2.1 Spuštění a ovládání softwaru Codecrypt

Ke spuštění programu Codecrypt není třeba nic instalovat, ani samotný program, jelikož je ve verzi portable, což znamená, že je možné ho spustit i z přenosného zařízení, jako je například USB flash disk nebo CD.

Před samotným šifrováním je nutno vygenerovat klíč, respektive klíče. K tomuto je využit u tohoto programu přepínač „-g“, který má i svůj pomocný příkaz „help“. Celý příkaz pro výpis pomoci k tomuto přepínači tedy vypadá „.\ccr.exe -g help“ a jeho výpis je zobrazen na následujícím obrázku číslo 11.

```
PS C:\codecrypt-portable> .\ccr.exe -g help
available algorithms: ([S]ig., [E]nc., sym. [C]ipher, [H]ash)
 S      FMTSEQ128C-CUBE256-CUBE128
 S      FMTSEQ128C-SHA256-RIPEMD128
 S      FMTSEQ128H20C-CUBE256-CUBE128
 S      FMTSEQ128H20C-SHA256-RIPEMD128
 S      FMTSEQ192C-CUBE384-CUBE192
 S      FMTSEQ192C-SHA384-TIGER192
 S      FMTSEQ192H20C-CUBE384-CUBE192
 S      FMTSEQ192H20C-SHA384-TIGER192
 S      FMTSEQ256C-CUBE512-CUBE256
 S      FMTSEQ256C-SHA512-SHA256
 S      FMTSEQ256H20C-CUBE512-CUBE256
 S      FMTSEQ256H20C-SHA512-SHA256
 E      MCEQCMDPC128FO-CUBE256-ARCFOUR
 E      MCEQCMDPC128FO-CUBE256-CHACHA20
 E      MCEQCMDPC128FO-CUBE256-XSYND
 E      MCEQCMDPC128FO-SHA256-ARCFOUR
 E      MCEQCMDPC128FO-SHA256-CHACHA20
 E      MCEQCMDPC128FO-SHA256-XSYND
 E      MCEQCMDPC256FO-CUBE512-ARCFOUR
 E      MCEQCMDPC256FO-CUBE512-CHACHA20
 E      MCEQCMDPC256FO-CUBE512-XSYND
 E      MCEQCMDPC256FO-SHA512-ARCFOUR
 E      MCEQCMDPC256FO-SHA512-CHACHA20
 E      MCEQCMDPC256FO-SHA512-XSYND
 C      ARCFOUR
 C      CHACHA20
 C      XSYND
 H      CUBE512
 H      RIPEMD128
 H      SHA256
 H      SHA512
 H      TIGER192
```

Obrázek 11 Výpis příkazu „.\ccr.exe -g help“ programu Codecrypt Zdroj: Vlastní zpracování

Dostupné šifrovací algoritmy lze vidět na řádcích s písmenem C. Avšak pro klíč k šifrování je potřeba využít jeden z řádků s písmenem E. Příkaz pro vygenerování klíče pro šifru Arcfour s použitím hašovací funkce SHA512 je „.\ccr.exe -g MCEQCMDPC256FO-SHA512-ARCFOUR –name ‘sharc4’“. Výpis tohoto příkazu lze vidět na obrázku číslo 12.

```
PS C:\codecrypt-portable> .\ccr.exe -g MCEQCMDPC256FO-SHA512-ARCFOUR --name 'sharc4'
Gathering random seed bits from kernel...
If nothing happens, move mouse, type random stuff on keyboard,
or just wait longer.
Seeding done, generating the key...
```

Obrázek 12 Výpis při generování klíče programem Codecrypt Zdroj: Vlastní zpracování

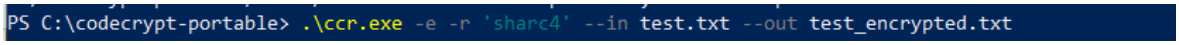
Hodnota za přepínačem „-g“ určí jaký šifrovací klíč se má vygenerovat a hodnota za přepínačem „--name“ určuje název klíče, v běžném využití by se tato hodnota mohla nastavit například na jméno adresáta šifrovaných dat. V tomto případě je klíč pojmenován autorem jako „sharc4“, aby byla jednoduše rozlišena metoda a hašovací funkce, jenž je daným klíčem využívána.

Vygenerované klíče lze zobrazit příkazem „.\ccr.exe -k“, pomocí přepínače “-k” se do konzole ve výpisu zobrazí vygenerované klíče jako je zobrazeno na obrázku 13.

```
PS C:\codecrypt-portable> .\ccr.exe -k
pubkey  MCEQCMDPC256FO-SHA512-CHACHA20  @67fa76d38d...  shachacha
pubkey  MCEQCMDPC256FO-CUBE512-CHACHA20 @6a17dbf5aa...  cubechacha
pubkey  MCEQCMDPC256FO-SHA512-XSYND     @6cfbe457c8...  shaxsynd
pubkey  MCEQCMDPC256FO-SHA512-ARCFOUR   @97df25849c...  sharc4
pubkey  MCEQCMDPC256FO-CUBE512-ARCFOUR  @ec839b6387...  cuberc4
pubkey  MCEQCMDPC256FO-CUBE512-XSYND    @fce25035e6...  cubexsynd
```

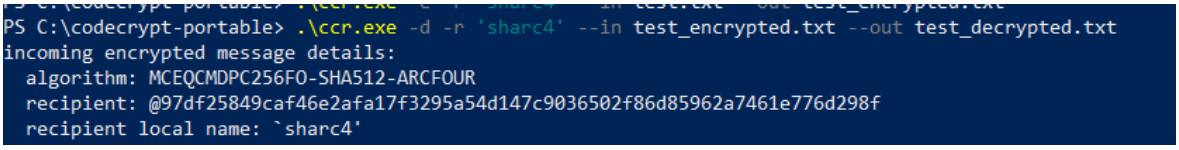
Obrázek 13 Výpis uložených klíčů programu Codecrypt Zdroj: Vlastní zpracování

Příkazem „.\ccr.exe -e -r ‘sharc4’ --in test.txt --out test_encrypted.txt“ je pomocí právě vygenerovaného klíče s názvem ‘sharc4‘ zašifrován soubor test.txt a výstup je uložen do souboru test_encrypted.txt. Přepínačem „-e“ je určeno, že program má šifrovat data. Za přepínačem „-r“ následuje název klíče použitého při šifrování. Pomocí přepínače „--in“ je určen soubor se vstupními daty pro šifrování, stejně tak je přepínačem „--out“ nastaven výstupní soubor šifrování. V případě, že tento soubor neexistuje, je vytvořen, pokud soubor již existuje, je jeho obsah přepsán. Příkaz nemá žádný výpis do konzole a je vidět na obrázku číslo 14.



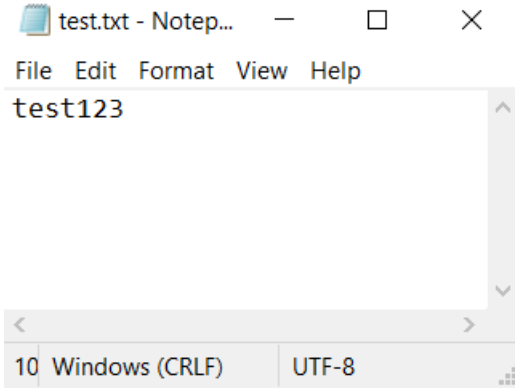
Obrázek 14 Příkaz k zašifrování souboru pomocí programu Codecrypt Zdroj: Vlastní zpracování

Podobným příkazem je soubor i rozšifrován, v příkazu stačí přepínač „-e“ na přepínač „-d“ a změnit názvy vstupního a výstupního souboru podle potřeby. Příkaz by tedy vypadal „.\ccr.exe -d -r ‘sharc4’ --in test_encrypted.txt --out test_decrypted.txt“. Jeho výpis je zobrazen na obrázku číslo 15.

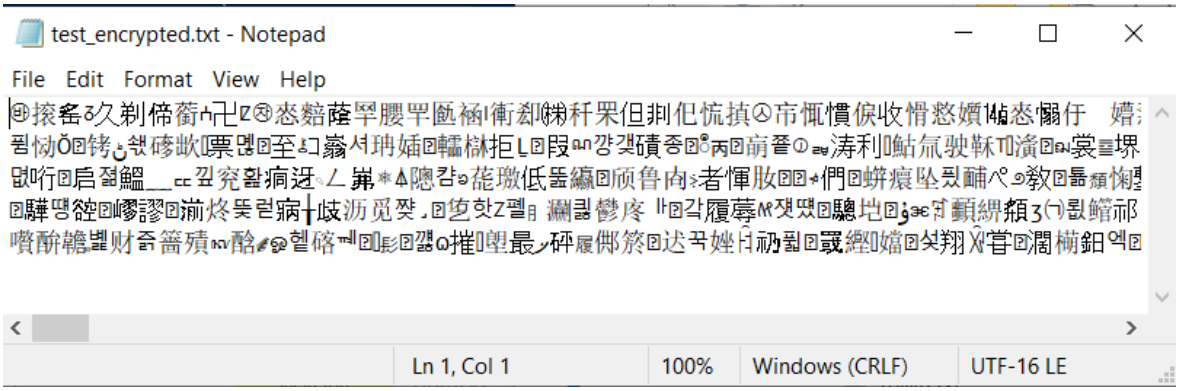


Obrázek 15 Příkaz pro rozšifrování souboru programem Codecrypt a jeho výpis Zdroj: Vlastní zpracování

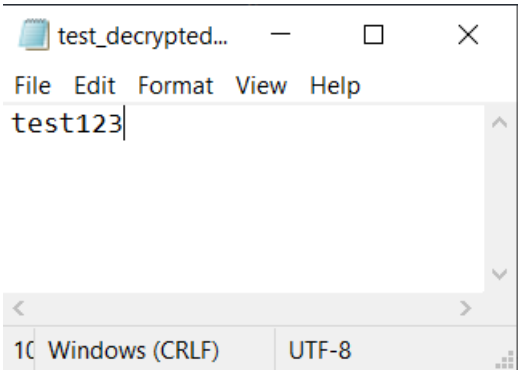
Vstupní a výstupní data těchto příkazů jsou zobrazeny na obrázcích číslo 16, 17 a 18.



Obrázek 16 Data vstupního souboru test.txt Zdroj: Vlastní zpracování



Obrázek 17 Data výstupního souboru šifrování test_encrypted.txt Zdroj: Vlastní zpracování



Obrázek 18 Data výstupního souboru rozšifrování test_decrypted.txt Zdroj: Vlastní zpracování

Měření délky běhu těchto příkazů je provedeno pomocí příkazu Measure-Command, ten je součástí Windows PowerShell. Syntaxe tohoto příkazu je „Measure-Command {}“, kde je ve složených závorkách napsán příkaz, který si přeje uživatel měřit. Jeho výstup je zobrazen na obrázku číslo 19:

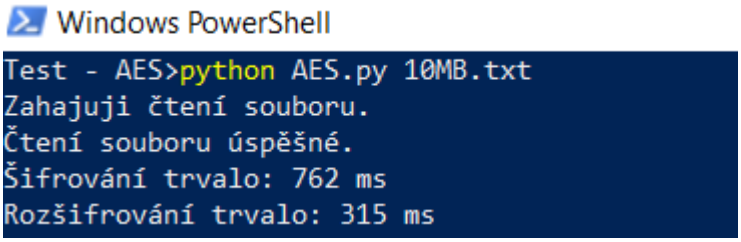
```
PS C:\codecrypt-portable> Measure-Command {.\ccr.exe -e -r 'sharc4' --in test.txt --out test_encrypted.txt}


Days              : 0
Hours             : 0
Minutes           : 0
Seconds           : 0
Milliseconds      : 163
Ticks             : 1634654
TotalDays         : 1.89196064814815E-06
TotalHours        : 4.54070555555556E-05
TotalMinutes      : 0.00272442333333333
TotalSeconds      : 0.1634654
TotalMilliseconds : 163.4654
```

Obrázek 19 Výpis příkazu Measure-Command Zdroj: Vlastní zpracování

Pro porovnání je využita hodnota TotalMiliseconds zaokrouhlená na celé číslo, což je celková doba běhu programu v milisekundách.

4.2.2 Spuštění a ovládání autorova programů

Ke spuštění programů napsaných autorem je potřeba mít nainstalovaný Python. V případě, kdy je na zařízení Python nainstalovaný, lze programy spustit pomocí příkazu „python AES.py 10MB.txt“, v této situaci by se spustil program pro šifrovací metodu AES a bylo by zahájeno šifrování a dešifrování souboru s názvem „10MB.txt“.



Obrázek 20 Spuštění a výpis autorova programu AES.py Zdroj: Vlastní zpracování

V programu je po ukončení běhu vypsán čas, jak dlouho trvalo šifrování a rozšifrování souboru. Tento čas je v následující kapitole využit pro porovnání. Výpis zobrazen na obrázku 20 mají všechny programy, jež byly napsány autorem, totožný.

4.3 Výsledky měření kryptografických metod

Výsledky měření jsou uvedeny v milisekundách. U tohoto měření platí čím menší hodnota, tím lepší. U metody RSA není použit soubor o velikosti 100 MB, jelikož by program běžel moc dlouhou dobu.

4.3.1 Délka běhu zašifrování jednotlivých metod

V tabulce číslo 3 je zobrazena doba běhu zašifrování souborů o různých velikostech jednotlivými metodami. V této tabulce lze vidět, že i přestože při 1500 bajtech je nejpomalejší metoda XSynd s použitím hašovací funkce SHA, tak při větším objemu dat se nejpomalejší metodou stává RSA, která již při 10 megabajtech měla dobu běhu delší než 12 minut.

Tabulka 3 Doby běhu zašifrování jednotlivých algoritmů

	1500 B	10 KB	100 KB	500 KB	1 MB	10 MB	100 MB
AES	1	2	2	31	78	734	8015
DES	1	1	15	62	124	1250	12252
RSA	105	734	6913	36071	70963	747669	-
Chacha20 (SHA)	298	303	325	411	469	1943	16591
Chacha20 (Cube)	309	320	395	741	1172	8918	86961
XSynd (SHA)	319	322	281	642	975	7110	68019
XSynd (Cube)	311	318	445	978	1701	14093	138308
Arcfour (SHA)	312	324	331	408	495	2115	18168
Arcfour (Cube)	310	330	397	736	1224	9078	88864

Zdroj: Vlastní zpracování dle výstupu z programů

4.3.2 Délka běhu rozšifrování jednotlivých metod

V tabulce číslo 4 je zobrazena doba běhu souborů o různých velikostech jednotlivými metodami při jejich rozšifrování. V této tabulce je zřejmé, že všechny postkvantové metody mají u souborů menších objemů delší dobu běhu v porovnání se zašifrováním. Oproti tomu metody současné mají tuto dobu nižší.

Tabulka 4 Doby běhu rozšifrování jednotlivých algoritmů

	1500 B	**10 KB**	**100 KB**	**500 KB**	**1 MB**	**10 MB**	**100 MB**
AES	1	1	1	15	31	296	3112
DES	1	1	15	46	93	767	7732
RSA	45	272	2623	13194	26228	264217	-
Chacha20 (SHA)	1257	1259	1282	1386	1447	2921	17664
Chacha20 (Cube)	1276	1302	1349	1681	2123	9863	87686
XSynd (SHA)	1269	1274	1375	1594	2012	8044	68513
XSynd (Cube)	1273	1275	1394	1936	2634	15279	138985
Arcfour (SHA)	1289	1301	1312	1356	1451	3018	18889
Arcfour (Cube)	1273	1297	1371	1682	2211	10041	89467

Zdroj: Vlastní zpracování dle výstupu z programů

4.3.3 Délka běhu celého procesu šifrování jednotlivých metod

V tabulce číslo 5 je zobrazena doba běhu celého procesu šifrování, to znamená zašifrování a rozšifrování souborů o různých velikostech jednotlivými metodami. V této tabulce je znázorněno, že nejrychlejší je současná metoda AES. Naopak metoda RSA běžela nejdéle, i přes ten fakt, že nebyla použita na souboru o velikosti 100 megabajtů.

Tabulka 5 Doby běhu celého procesu šifrování jednotlivých algoritmů

	1500 B	10 KB	100 KB	500 KB	1 MB	10 MB	100 MB
AES	2	3	3	46	109	1030	11127
DES	2	2	30	108	217	2017	19984
RSA	150	1006	9536	49265	97191	1011886	-
Chacha20 (SHA)	1555	1562	1607	1797	1916	4864	34255
Chacha20 (Cube)	1585	1622	1744	2422	3295	18781	174647
XSynd (SHA)	1588	1594	1756	2236	2987	15154	136532
XSynd (Cube)	1584	1593	1839	2914	4335	29372	277293
Arcfour (SHA)	1601	1625	1643	1764	1946	5133	37057
Arcfour (Cube)	1583	1627	1768	2418	3435	19119	178331

Zdroj: Vlastní zpracování dle výstupu z programů

4.4 Zhodnocení výsledků jednotlivých kryptografických metod

Výsledky měření jsou přepočítány na rychlost jednotlivých kryptografických metod a následně jsou tyto hodnoty porovnány. U těchto hodnot platí čím vyšší, tím lepší.

4.4.1 Interpretace měření jako rychlost v jednotkách MB/s

Pro interpretaci výsledků jako rychlost v jednotkách megabajty za sekundu jsou jednotlivé velikosti souborů přepočítány do velikosti v megabajtech a délky běhu programů jsou přepočítány na sekundy. Poté je velikost souboru v megabajtech vydělena délkou běhu v sekundách, výsledkem je rychlost v jednotkách MB/s a ta je zaokrouhlena na dvě desetinná místa. Z těchto výsledků je následně vypočten aritmetický průměr této rychlosti a jednotlivé metody jsou porovnány podle tohoto aritmetického průměru.

4.4.2 Výsledky rychlosti zašifrování v MB/s

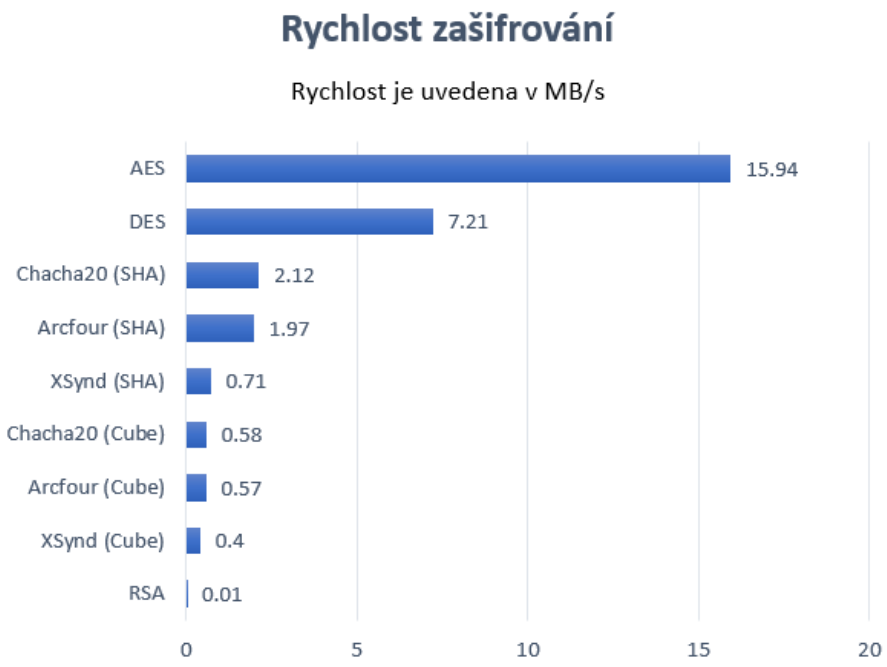
Výsledky výpočtů rychlosti zašifrování jsou znázorněny v tabulce číslo 6. V tabulce je zobrazeno, že nejvyšší rychlost zašifrování mají současné symetrické metody. U metod postkvantových je taktéž vidět, že na jejich rychlost má vliv použitá hašovací funkce.

Tabulka 6 Výsledky výpočtu rychlosti zašifrování jednotlivých metod

	Rychlost
AES	15,94
DES	7,21
Chacha20 (SHA)	2,12
Arcfour (SHA)	1,97
XSynd (SHA)	0,71
Chacha20 (Cube)	0,58
Arcfour (Cube)	0,57
XSynd (Cube)	0,40
RSA	0,01

Zdroj: Vlastní zpracování

Tyto výsledky jsou rovněž znázorněny v grafu, jenž lze vidět na obrázku číslo 21.



Obrázek 21 Graf rychlosti zašifrování jednotlivých algoritmů Zdroj: Vlastní zpracování

4.4.3 Výsledky rychlosti rozšifrování v MB/s

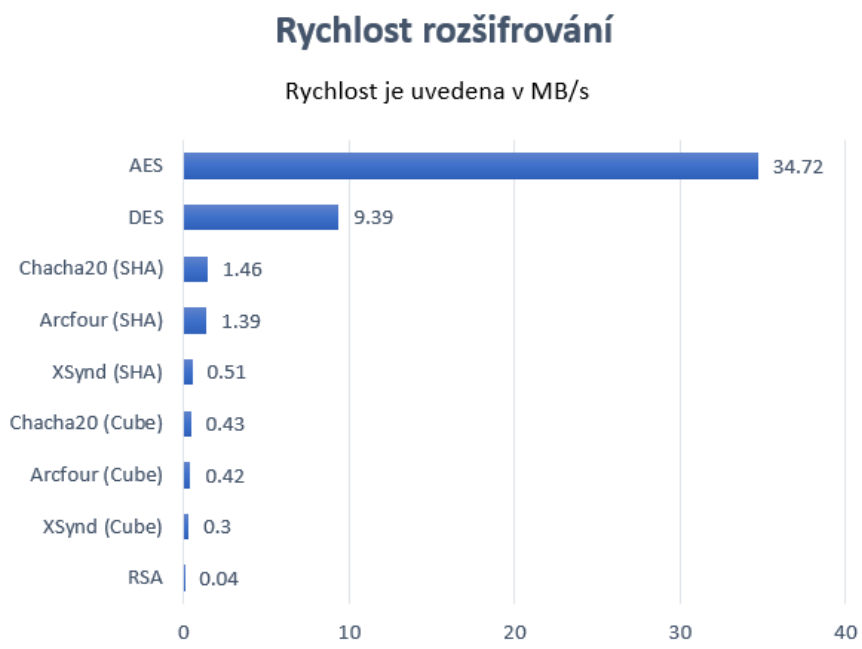
Výsledky výpočtů rychlosti rozšifrování jsou znázorněny v následující tabulce číslo 7. Pořadí je v této tabulce stejné jako u rychlosti zašifrování, tudíž je zřejmé, že na pořadí metod nemá vliv proces využití těchto metod. Jednoznačně nejrychlejší je dle výsledků metoda AES, která je svou rychlostí skoro čtyřikrát rychlejší než metoda umístěná na druhém místě a oproti metodám ostatním je tento náskok ještě výraznější.

Tabulka 7 Výsledky výpočtu rychlosti zašifrování jednotlivých metod

	Rychlost
AES	34,72
DES	9,39
Chacha20 (SHA)	1,46
Arcfour (SHA)	1,39
XSynd (SHA)	0,51
Chacha20 (Cube)	0,43
Arcfour (Cube)	0,42
XSynd (Cube)	0,30
RSA	0,04

Zdroj: Vlastní zpracování

Na obrázku číslo 22 jsou výsledky znázorněny i v grafu.



Obrázek 22 Graf rychlosti rozšifrování jednotlivých algoritmů Zdroj: Vlastní zpracování

4.4.4 Hodnocení porovnání

Pořadí výsledků je stejné u procesu šifrování i rozšifrování. Metoda AES se podle rychlosti umístila na prvním místě, na druhém místě se umístila metoda DES. Tyto dvě metody jsou výrazněji rychlejší než metody poskvantové i asymetrická metoda RSA, která se umístila na místě posledním. U metod postkvantových je podle měření zřejmé, že na jejich výkonnost má vliv i použitá hašovací metoda.

5 Závěr

Cílem práce bylo porovnání současných a postkvantových šifrovacích metod pomocí jejich časové složitosti. V teoretické části jsou na základě studia odborné a vědecké literatury popsány jednotlivé šifrovací metody a další témata spojena s problematikou kryptografie.

V praktické části byl k realizaci a měření současných metod využit autorem napsaný program v jazyce Python s využitím knihoven volně přístupných na internetu. Pro realizaci postkvantových metod byl použit již hotový program s názvem Codecrypt. Pro měření výkonnosti tohoto programu byl využit příkaz zabudovaný ve Windows PowerShell, přes který byl i program Codecrypt spouštěn. Pro měření bylo využito více souborů s různými velikostmi.

Výsledkem měření byla původně doba běhu programů měřená v milisekundách. Toto měření bylo provedeno pro každou metodu zvlášť a zároveň i jednotlivě pro samotné procesy zašifrování a rozšifrování souborů. Pro porovnání byly výsledky měření přepočteny na rychlost šifrování v jednotkách MB/s.

Metoda RSA v porovnání skončila výrazně nejpomaleji, pro tuto metodu je však předností její bezpečnost, kterou lze velmi jednoduše zachovat za použití hybridní kryptografie. V kombinaci se symetrickou metodou AES by bezpečnost neměla klesnout a zároveň by se rychlost šifrování měla výrazně zvýšit.

I přestože byl DES charakterizován jako neefektivní v implementaci softwaru, byl druhý v pořadí podle rychlosti. Dle teoretických východisek je předností metody AES rychlost. Toto tvrzení bylo v praktické části potvrzeno.

Výsledky porovnání ukázaly, že na základě provedených měření mají současně využívané symetrické kryptografické metody lepší optimalizaci, nevyžadují ke svému běhu tedy tak vysoký výpočetní výkon. Respektive stejně velké soubory zvládnou zpracovat rychleji než metody postkvantové. V případě potřebného přechodu na tyto postkvantové metody, v nezměněné formě, by tedy bylo nutné brát v potaz výpočetní výkonnost zařízení, na kterém by tyto metody byly využívány.

6 Seznam použitých zdrojů

AUMASSON, Jean-Philippe, 2018. Serious cryptography: A Practical Introduction to Modern Encryption. San Francisco: No Starch Press. ISBN 978-159-3278-267.

BERNSTEIN, Daniel J., Johannes BUCHMANN a Erik DAHMEN, 2009. Post-Quantum Cryptography. Springer. ISBN 978-354-0887-027.

BUELL, Duncan A., 2021. Fundamentals of cryptography: introducing mathematical and algorithmic foundations. Cham, Switzerland: Springer. Undergraduate topics in computer science. ISBN 978-303-0734-923.

BURDA, Karel, 2019. Kryptografie okolo nás. Praha: CZ.NIC, z.s. p.o. CZ.NIC. ISBN 978-808-8168-492.

C. VAN OORSCHOT, Paul, 2020. Computer Security and the Internet: Tools and Jewels. Springer. ISBN 978-303-0336-486.

EGERTON, Taylor Onate a Victor EMMAH, 2018. Comparative Analysis of Cryptographic Algorithms in Securing Data. ResearchGate. 2018, 5. ISSN 2231-5381.

GOUTAM, Paul a Maitra SUBHAMOY, 2019. RC4 Stream Cipher and Its Variants. 2. Taylor & Francis Group. ISBN 0367382164.

KATZ, Jonathan a Yehuda LINDELL, 2020. Introduction to Modern Cryptography. 3rd. Chapman and Hall/CRC. ISBN 978-081-5354-369.

MARTIN, Keith, 2020. Cryptography: The Key to Digital Security, How It Works, and Why It Matters. W. W. Norton & Company. ISBN 978-132-4004-295.

MEZIANI, Mohammed a Gerhard HOFFMANN, 2020. Improving the Performance of the SYND Stream Cipher. HAL – Open Science [online]. 2012, 19 [cit. 2022-03-05]. Dostupné z: https://hal-ujm.archives-ouvertes.fr/ujm-00865533/en

OPPLIGER, Rolf, 2021. Cryptography 101: From Theory to Practice. Artech House. ISBN 978-163-0818-463.

STALLINGS, William, 2020. Cryptography and Network Security: Principles and Practice. 8th Edition. Pearson. ISBN 978-013-5764-251.

STĂNICĂ, Pantelimon, Sugata GANGOPADHYAY a Sumit Kumar DEBNATH, 2021. Security and Privacy: Select Proceedings of ICSP 2020. Springer. ISBN 978-981-3367-807.

STINSON, Douglas Robert a Maura B. PATERSON, 2019. Cryptography: Theory and practice. Fourth edition. CRC Press. ISBN 978-113-8197-015.

WONG, David, 2021. Real-World Cryptography. Manning Publications. ISBN 978-161-7296-710.

YADAV, Prateek, Indivar GUPTA a Somanchi Krishna MURTHY, 2016. Study and analysis of eSTREAM cipher Salsa and ChaCha. 2016 IEEE International Conference on Engineering and Technology (ICETECH) [online]. 2016, 94 [cit. 2022-03-05]. Dostupné z: doi:10.1109/ICETECH.2016.7569218

7 Přílohy

Příloha 1 Zdrojový kód programu pro algoritmus AES

```
from Crypto.Cipher import AES
from secrets import token_bytes
import sys
import datetime

def encrypt(msg):
    cipher = AES.new(key, AES.MODE_EAX)
    nonce = cipher.nonce
    ciphertext, tag = cipher.encrypt_and_digest(msg.encode('ascii'))
    return nonce, ciphertext, tag

def decrypt(nonce, ciphertext, tag):
    cipher = AES.new(key, AES.MODE_EAX, nonce=nonce)
    plaintext = cipher.decrypt(ciphertext)
    try:
        cipher.verify(tag)
        return plaintext.decode('ascii')
    except:
        return False

key = token_bytes(16)

print('Zahajuji čtení souboru.')
start_time = datetime.datetime.now()
filename = str(sys.argv[1])
file = open(filename, 'r')
lines = file.readlines()
file.close()
text = ''
for line in lines:
    text += str(line)
filereaddelta = datetime.datetime.now() - start_time
print('Čtení souboru úspěšné.')


start_time = datetime.datetime.now()
nonce, ciphertext, tag = encrypt(text)
filename = filename.replace(".", "_encrypted.")
file = open(filename, 'w')
file.write(str(ciphertext))
file.close()
delta = datetime.datetime.now() - start_time + filereaddelta
print('Šifrování trvalo: %s ms' % int(delta.total_seconds() * 1000))

start_time = datetime.datetime.now()
file = open(filename, 'r')
lines = file.readlines()
file.close()
text = ''
for line in lines:
    text += str(line)
plaintext = decrypt(nonce, ciphertext, tag)
delta = datetime.datetime.now() - start_time
print('Rozšifrování trvalo: %s ms' % int(delta.total_seconds() * 1000))
```

Příloha 2 Zdrojový kód programu pro algoritmus DES

```
from Crypto.Cipher import DES
from secrets import token_bytes
import sys
import datetime

def encrypt(message):
    cipher = DES.new(key, DES.MODE_EAX)
    nonce = cipher.nonce
    ciphertext, tag = cipher.encrypt_and_digest(message.encode('ascii'))
    return nonce, ciphertext, tag

def decrypt(nonce, ciphertext, tag):
    cipher = DES.new(key, DES.MODE_EAX, nonce=nonce)
    plaintext = cipher.decrypt(ciphertext)
    try:
        cipher.verify(tag)
        return plaintext.decoce('ascii')
    except:
        return False


key=token_bytes(8)

print('Zahajuji čtení souboru.')
start_time = datetime.datetime.now()
filename = str(sys.argv[1])
file = open(filename)
lines = file.readlines()
file.close()
text = ''
for line in lines:
    text += str(line)
filereaddelta = datetime.datetime.now() - start_time
print('Čtení souboru úspěšné.')

n = 5000000
chunks = [text[i:i+n] for i in range(0, len(text), n)]
ciphertext = bytes(''.encode('ascii'))
encryptdelta = int(filereaddelta.total_seconds() * 1000)
decryptdelta = 0
filename = filename.replace(".", "_encrypted.")
filewrite = open(filename, 'w')
fileread = open(filename, 'r')

for chunk in chunks:
    start_time = datetime.datetime.now()
    nonce, ciphertext, tag = encrypt(chunk)
    filewrite.write(str(ciphertext))
    delta = datetime.datetime.now() - start_time
    encryptdelta += int(delta.total_seconds() * 1000)
    start_time = datetime.datetime.now()
    text = fileread.readlines()
    plaintext = decrypt(nonce, ciphertext, tag)
    delta = datetime.datetime.now() - start_time
    decryptdelta += int(delta.total_seconds() * 1000)


filewrite.close()
fileread.close()
print('Šifrování trvalo: %s ms' % encryptdelta)
print('Rozšifrování trvalo: %s ms' % decryptdelta)
```

Příloha 3 Zdrojový kód programu pro algoritmus RSA

```
import rsa
import sys
import datetime

def generate_keys():
    (pubKey, privKey) = rsa.newkeys(1024)
    with open('rsa-keys/publickey.pem', 'wb') as f:
        f.write(pubKey.save_pkcs1('PEM'))

    with open('rsa-keys/privatekey.pem', 'wb') as f:
        f.write(privKey.save_pkcs1('PEM'))

def load_keys():
    with open('rsa-keys/publickey.pem', 'rb') as f:
        pubKey = rsa.PublicKey.load_pkcs1(f.read())

    with open('rsa-keys/privatekey.pem', 'rb') as f:
        privKey = rsa.PrivateKey.load_pkcs1(f.read())

    return pubKey, privKey

def encrypt(msg, key):
    return rsa.encrypt(msg.encode('ascii'), key)

def decrypt(ciphertext, key):
    try:
        return rsa.decrypt(ciphertext, key).decode('ascii')
    except:
        return False

def sign_sha1(msg, key):
    return rsa.sign(msg.encode('ascii'), key, 'SHA-1')

def verify_sha1(msg, signature, key):
    try:
        return rsa.verify(msg.encode('ascii'), signature, key) == 'SHA-1'
    except:
        return False

generate_keys()
pubKey, privKey = load_keys()

print('Zahajuji čtení souboru.')
start_time = datetime.datetime.now()
filename = str(sys.argv[1])
file = open(filename)
lines = file.readlines()
file.close()
text = ''
for line in lines:
    text += str(line)
filereaddelta = datetime.datetime.now() - start_time
print('Čtení souboru úspěšné.')

n = 120
chunks = [text[i:i+n] for i in range(0, len(text), n)]
ciphertext = bytes(''.encode('ascii'))
signature = bytes(''.encode('ascii'))
encryptdelta = int(filereaddelta.total_seconds() * 1000)
decryptdelta = 0
filename = filename.replace(".", "_encrypted.")
filewrite = open(filename, 'w')
fileread = open(filename, 'r')
file.close()
for chunk in chunks:
    start_time = datetime.datetime.now()
    ciphertext = encrypt(chunk, pubKey)
    signature = sign_sha1(chunk, privKey)
    filewrite.write(str(ciphertext))
    delta = datetime.datetime.now() - start_time
    encryptdelta += int(delta.total_seconds() * 1000)
    start_time = datetime.datetime.now()
    text = fileread.readline()
    plaintext = decrypt(ciphertext, privKey)
    delta = datetime.datetime.now() - start_time
    decryptdelta += int(delta.total_seconds() * 1000)

filewrite.close()
fileread.close()
print('Šifrování trvalo: %s ms' % encryptdelta)
print('Rozšifrování trvalo: %s ms' % decryptdelta)
```