**Gymnázium Christiana Dopplera, Zborovská 45, Praha 5**

ROČNÍKOVÁ PRÁCE

# Jak funguje RSA a další šifrovací algoritmy

Vypracoval: Vojtěch Martínek
Třída: 4.C
Školní rok: 2024/2025
Seminář: Programovací seminář

Prohlašuji, že jsem svou ročníkovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím s využíváním práce na Gymnáziu Christiana Dopplera pro studijní účely.

V Praze dne Datum Vojtěch Martínek

# Obsah

# 1. Úvod

V této ročníkové práci se zaměřuji na šifrovací algoritmy a jak fungují, Konkrétně pak hlavně algoritmem RSA a jeho implementací v C#. Jelikož jsou šifry součásti téměř veškeré komunikace přes internet, je toto téma velmi důležité a pro mě velmi zajímavé. Obzvlášť pak v dnešní době, kdy kvantové počítače začínají ohrožovat nejpoužívanější způsoby kryptografie, zejména pak otázku bezpečnosti šifer jako je právě RSA. Z tohoto důvodu jsem se rozhodl se podrobně zabývat fungováním těchto algoritmů, jejich výhodami a nevýhodami, složitostí rozšifrování bez znalosti klíče a způsoby vytvoření klíčů na základě zašifrovaného textu.

# 2. Základní pojmy a principy

## 2.1 Vyhledávací tabulka (lookup table)

Vyhledávací tabulky jsou tabulky čtvercového tvaru které slouží na Záměnu znaků či čísel jinými. Tyto tabulky bývají používány v šifrovacích algoritmech na záměnu dat čímž zlepšují jejich bezpečnost.

## 2.2 Bitové operace

Bitové operace jsou jednoduché logické operace na úrovni bitů. Nejzákladnějšími z nich jsou tyto:

### 2.2.1 And

Operace AND bere jako vstup 2 stejně dlouhá čísla v binární soustavě a vrací jedno. Vrácené číslo bude mít jedničky pouze na místě kde obě vstupní čísla měli také jedna Příklad: při vstupu 1010 a 1100 bude výsledné číslo 1000 jelikož pouze na první pozici mají oba bity hodnotu 1.

### 2.2.2 OR

Operace OR bere stejně jako AND 2 vstupy a dává jeden výstup, na rozdíl od AND ji však stačí když alespoň jeden z bitů je jedna. Například z čísel 1010 a 1100 vznikne číslo 1110.

### 2.2.3 XOR

Operace XOR je speciální případ operace OR kdy aby byla vrácena jednička musí výt právě jeden z bitů jedna ale ne oba. Příklad: Při vstupu 1010 a 1100 nám XOR vrátí 0110.

### 2.2.4 Not

Not operace jednoduše změní všechny nuly na jedničky a všechny jedničky na nuly. Například z čísla 1010 by po operaci Not vzniklo 0101

## 2.3 Ciphertext

Ciphertext (česky zašifrovaný text) je jak už název napovídá text vzniklý zašifrováním pomocí algoritmu.

## 2.4 Public key

Public key (česky veřejný klíč) je druh klíče nejčastěji v Asymetrických šifrovacích algoritmech. V těchto algoritmech slouží public key jako veřejně známý klíč pomocí kterého může kdokoliv zašifrovat vlastní text a poslat ho konkrétnímu uživateli, který ho bude schopen rozšifrovat. V popisovaných algoritmech se vyskytuje vždy ve tvaru dvou čísel (a, b).

## 2.5 Private key

Private key (soukromý nebo také tajný klíč) je klíč sloužící v Asymetrických algoritmech k dešifrování zpráv poslaných ostatními uživateli. Stejně jako public key je i private key nejčastěji ve formátu dvou čísel (a, b)

## 2.6 Padding a Padding schemes

Padding je způsob zabezpečení textu přidáním zbytečného a často nesmyslného textu navíc k původní zprávě před šifrováním. Slouží k zlepšené bezpečnosti textu, ta vzniká jelikož kdokoliv kdo se pokusí rozluštit šifru nemá jak určit které části jsou po rozšifrování plnohodnotný text a které jsou pouze náhodné znaky.

# 3. Co je to šifrovací algoritmus

Šifrovací algoritmy nebo také šifry jsou algoritmy pomocí kterých můžeme bezpečně posílat text po nezabezpečené cestě. To dělají použitím symetrické či asymetrické změny textu pomocí klíče, díky kterým z užitečného či čitelného textu udělají nesmyslný či nepoužitelný text. tento text pak může odesilatel bez problému poslat příjemci, který za použití stejného či jiného klíče zprávu vrátí do původní čitelné nebo použitelné verze. Díky tomuto nemůže kdokoliv bez přístupu ke klíči zprávu přečíst, za předpokladu že nemůže klíč nějakým způsobem získat. Šifrovací algoritmy dělíme na 2 základní kategorie:

## 3.1 Druhy šifrovacích algoritmů

### 3.1.1 Symetrické Šifrovací algoritmy

Symetrické šifrovací algoritmy používají stejný klíč na zašifrovaní i rozšifrování zprávy. Výhoda symetrických algoritmů spočívá v jejich často velmi malých a bezpečných klíčích, což přispívá k jejich nízké složitosti a jednoduchosti. Jejich největší problém ale spočívá ve výměně klíče, která musí proběhnout po zabezpečené cestě nebo musí být sama zašifrována aby nedošlo k získání klíče nežádanou osobou. Pokud totiž nějaká třetí strana získá klíč, nedojde pouze k rozšifrování zpráv cizí osobou, tato osoba také může vytvářet zprávy nové u kterých nebude odlišitelné zda jsou od původního odesílatele či od třetí strany. Pokud však k tomuto nedojde jsou tyto algoritmy zpravidla velmi bezpečné.

Příklady Symetrický Šifrovacích algoritmů a jejich vysvětlení:

#### AES (Advanced Encryption standard - standard pokročilého šifrování)

AES je šifrovací algoritmus který je považován za jeden z bezpečnějších algoritmů protože byl vytvořen s cílem bezpečnosti vůči známým kryptoanalytickým útokům. Je rozdělen na tři standardy, které všechny používají stejný algoritmus pouze s jiným počtem kol. Tyto standardy jsou: AES-128 (používá 10 kol), AES-192 (používá 12 kol) a AES-256 (14 kol).

Nejdříve si tento algoritmus rozdělí původní klíč na základě AES key schedule, který za pomocí bitových operací a hexadecimální soustavy vytvoří tolik klíčů (tzv. Round keys) na tolik klíčů, kolik je potřeba pro každý standard (každý standard potřebuje stejný počet klíčů jako je počet kol která provádí +1).

Následně AES začne se šifrováním dat. Nejdříve přidá první Round key do dat, která jsou uspořádána v 4*4 "tabulce", pomocí bitové operace XOR a provede tolik kol kolik vyžaduje používaný standard. V každém kole provede postupně tyto korky: SubBytes, ShiftRows, MixCollums a AddRoundKey. Nejdříve pomocí lookup table (tzv. AES S-box) změní číslo v hexadecimální soustavě na jiné (SubBytes), posune bity podle toho v jakém řádku se nachází (první řádek se nemění, druhý se posune o jeden krok doleva, třetí o dva a čtvrtý o

tři) (ShiftRows) a pomocí lineární transformace zamíchá hodnoty v jednotlivých sloupcích dohromady (Mixcollums). Tento krok je jeden z nejdůležitějších, protože je velmi složité udělat jeho inverzní funkci bez znalosti klíče, což algoritmu velmi přidává na bezpečnosti. Nakonec kola algoritmus přidá pomocí bitové XOR funkce Round key (AddRoundKey) a pokračuje s dalším kolem. V posledním kole se však nemíchají hodnoty ve sloupcích a provádí se pouze ostatní kroky.

Při dešifrování se používají stejné kroky jako při šifrování, ale místo jednotlivých operací se používají jejich opaky (např. Inverzní posun řádků, který posune řádky doprava místo doleva. Kroky kola se odehrávají v následujícím pořadí: Inverzní ShiftRows, Inverzní SubBytes, AddRoundKey, a Inverzní MixCollums. Po dokončení všech kol dostaneme původní zprávu.

### DES (Data Encryption standard - standard šifrování dat)

DES je jeden ze starších šifrovacích algoritmů z 20. stol. V dnešní době je kvůli svému krátkému klíči o délce 64 bitů považován za nedostatečný, ale pro běžné použití je bezpečný. I Přes to že klíč je dlouhý 64 bitů, zkracuje se pomocí systému PC-1 na 56 bitů a zbylých 8 bitů je buď zahozeno nebo použito jako kontrola parity. Při každém kole se pak z 56 bitů vybírá systémem pc-2 pouze 48 bitů které jsou v něm použity jako klíč. Kvůli této délce byl DES nahrazen již zmíněným AES, které je oproti němu značně bezpečnější

DES je založen na Vyhledávacích tabulkách a bitových operacích. Nejdříve si rozdělí data na 64. bitové bloky které jsou dány do čtvercové tabulky. Tabulku si následně rozdělí na dvě 32. bitové poloviny se kterými pracuje odděleně. Pravou polovinu nejdříve Rozšíří na 48 bitů pomocí pevné lookup table (E-box), následně provede XOR operaci a pomocí další lookup table (S-box) zmenší zpět na 32 bitů. Tyto změny mezi počty bitů dělají algoritmus více nelineární, což zlepšuje jeho bezpečnost.

Čtvrtý krok kola je permutace, která přehází bity pomocí P-boxu. Nakonec se celá upravená pravá strana zkombinuje pomocí XOR s levou stranou, čímž nám vzniká nová pravá strana tabulky. Na levou stranu tabulky umístíme pravou stranu tabulky před provedením tohoto kola a proces opakujeme celkem 16krát. Nakonec provede DES poslední permutaci ($IP^{-1}$) a tím dostaneme finální ciphertext.

Při dešifrování DES používá stejné kroky jako při šifrování ale v opačném pořadí.

### One time pad (jednorázová tabulková šifra)

OTP je speciální šifrovací algoritmus který je nemožné prolomit za splnění jednoduchých podmínek: Klíč je stejně nebo delší než šifrovaný text, klíč není nikdy znovu použit a klíč je plně náhodný. Nevýhoda tohoto způsobu šifrování je hlavně sdílení klíčů, který je potřeba na každou zprávu nový. Jelikož klíče musí být vždy stejně velké nebo větší než posílaná zpráva a stejně jako zpráva musí být poslány bezpečnou cestou, je většinou zdaleka jednoduší použít bezpečnou cestu kterou bychom použili na předání klíče k předání zprávy. Kvůli tomu jsou OTP velmi málo používané a v dnešní době oproti ostatním algoritmům zastaralé.

### 3.1.2 Asymetrické šifrovací algoritmy

Asymetrické šifrovací se od symetrických liší několika způsoby. Jeden z největších je ale Rozdílnost šifrovacího a dešifrovacího klíče. Díky tomu mohou tyto mnohem lépe zajistit svou bezpečnost jelikož musí chránit pouze dešifrovací klíč (private key), zatímco šifrovací klíč (public key) mohou bez problému poslat do světa. Pokud někdo chce poslat zašifrovanou zprávu stačí si od příjemce vyžádat jejich public key a poté jim zprávu poslat zašifrovanou pomocí tohoto klíče. Příjemce si pak pomocí svého private key zprávu rozšifruje a získá tak původní text. Hlavní nevýhoda těchto algoritmů ale spočívá ve velikosti jejich klíčů a komplexitě algoritmů. Z tohoto důvodu se tyto algoritmy často používají na zašifrování klíče pro Symetrický algoritmus, který je pak používán pro zbytek zpráv.

Příklady Asymetrických algoritmů a jejich popis:

#### RSA (Rivest-Shamir-Adleman)

RSA je jeden z nejznámějších šifrovacích algoritmů, který je používán v dnešní době na zabezpečení komunikace mezi počítači a sdílení klíčů. Kvůli své převážně pomalé rychlosti se RSA příliš nepoužívá na šifrování samotných dat uživatelů. Jeho bezpečnost je postavená na prvočíselných základech klíčů, mocninách a funkci modulo. Hlavní nevýhoda RSA je že jelikož nepoužívá žádné náhodné prvky je možné prolomit veškeré jeho klíče, ale prolomení jednoho je i s nejvýkonnějšími počítači téměř nemožný úkol, a to obzvlášť když použijeme správný padding scheme. S nástupem kvantových počítačů se toto však stává znovu problémem jelikož je pro ně takovýto úkol zdaleka jednodušší než pro běžné počítače.

Na vytvoření klíčů v RSA je nejdříve potřeba náhodně vybrat 2 velká prvočísla p a q. Tento krok se nejčastěji řeší pomocí generování náhodných celých čísel a testovaní jejich prvočíselnosti. Po získaní dvou prvočísel získáme jejich vynásobením modulus, který je součástí jak public tak private key. Dále algoritmus vypočítá $\lambda$n, který se rovná nejmenšímu společnému násobku čísel p-1 a q-1 a zvolí si e (exponent při šifrování) tak aby platilo:

$1 < e < \lambda n$

Tím RSA získá druhou část public key, a musí pouze dodělat private key. Tomu chybí pouze exponent který získá pomocí vzorce:

$d * e \equiv 1 \pmod{\lambda n}$

Tento vzorec vyřeší pro d (exponent dešifrování) a získáme tím poslední část dešifrovacího klíče. Nyní může uživatel poslat svůj šifrovací klíč v podobě (n, e) komukoliv kdo mu chce poslat zprávu.

Šifrování v RSA je proces používající funkci modulo a mocniny, čímž se docílí zašifrování textu a kvůli funkci modulo je téměř nemožné zjistit zpětně původní zprávu a to i přes znalost klíče na zašifrování. Při šifrování si nejdříve rozdělí RSA původní text na menší bloky a převede na čísla tak aby vzniklé číslo bylo menší než n. K tomu přidá RSA padding, který zaručí bezpečnost textu před spoustou druhů útoků, a použije následující vzorec, kde c je ciphertext a m je blok nezašifrované zprávy, aby zprávu zašifrovalo:

$c \equiv m^e \pmod{n}$

Pro dešifrování RSA používá stejný vzorec pouze s jiným exponentem:

$m \equiv c^e \pmod{n}$

### Diffie-Hellman

Diffie-Hellman (DH) funguje podobně jako RSA na základě funkce modulo a mocnění. Slouží převážně k jednorázovému poslání klíče jiného šifrovacího algoritmu, ale může sám být použit na šifrování zpráv.

DH je založen na prvotní dohodě obou stran na public key, který následně upravují aby získaly stejný private key. Obě strany po dohodnutí na public key (p, g) vytvoří svůj náhodný tajný klíč (a), který kombinující s Public key pomocí následujícího vzorce:

$A = g^a \mod p$

Tímto vznikne meziklíč A, který DH pošle po veřejném kanále druhé straně. Po přijetí meziklíče od druhé strany provede DH s tímto meziklíčem stejnou operaci jako dříve, čímž se dostanou obě strany ke stejnému skrytému klíči.

### ELGamal signature scheme (podpisové schéma E)

ElGamal je

# 4. Praktická část - vlastní implementace RSA

## 4.1 Základní informace o mé implementaci

Na praktickou část mé seminární práce jsem se rozhodl vytvořit vlastní implementaci šifrovacího algoritmu RSA. Má implementace se liší zejména ve způsobu generování prvočísel. Standardní implementace hledání prvočísel používají náhodný výběr čísel a následného zkoušení prvočíselnosti. Má implementace RSA oproti tomu využívá před-generovaný textový dokument, ve kterém jsou vypsána veškerá prvočísla do určité hodnoty. Tato prvočísla si může uživatel sám vygenerovat pokud jich není předem vygenerováno dost a slouží k rychlejší generaci klíčů, jelikož nemusí náhodně generovat čísla a místo toho je pouze náhodně vybírá z před-generovaného souboru.

Má implementace algoritmu RSA je z velké části založená na komunikaci skrze textové soubory. K tomuto rozhodnutí jsem dospěl převážně kvůli mé z počátku nedostatečné znalosti způsobů komunikace mezi zařízeními. Tento způsob má samozřejmé nevýhody jako jsou nutnost cloudového úložiště pro komunikaci mezi více počítači a neefektivnost přenosu dat, slouží proto má implementace spíše jako koncept na implementaci která je rychlejší, ale více náročná na ukládání dat.

## 4.2 Postup tvorby programu

Má implementace byla rozdělena na dvě hlavní části. První část spočívala v generování prvočísel a zpracování prvočísel do klíče. Na generování prvočísel jsem nejdříve vytvořil oddělený program, který jsem později propojil s hlavním programem a stala se z něj tak třída GeneratePrimes. Na generování klíčů jsem se nejdříve pokoušel použít alternativní metodu, která jsem ale později zjistil že je matematicky chybná a proto jsem pře-implementoval generování klíčů do standardního postupu RSA vyjma vybírání náhodných prvočísel.

Druhá část spočívala v samotném šifrování a dešifrování. Tato část využívá víceméně standardní šifrování RSA ale místo posílání přímo druhé straně ukládá zašifrovaný text do souboru který si může druhý program přečíst. Díky způsobu mé implementace RSA může program bez problému běžet v několika instancích naráz a dokázal by s pouze minimálními úpravami běžet i přes cloud. Hlavní nevýhoda této implementace je však chybějící padding který jsem se rozhodl nepřidávat z důvodu výrazně zvýšené komplexity programu.

# 5. Závěr

Toto je závěr mé ročníkové práce.

# Literatura

[1] Birge J. R., Wets R. J.-B. (1987): Computing bounds for stochastic programing problems by means of a generalized moment problem. *Mathematics of Operations Research* **12**, 149-162.

# Přílohy