FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Aplikace homomorfního šifrování v praxi
Student:	Vojtěch Sedlák
Vedoucí:	Ing. Miroslav Pospíšek, CSc.
Studijní program:	Informatika
Obor / specializace:	Informační bezpečnost 2021
Katedra:	Katedra informační bezpečnosti
Platnost zadání:	do konce letního semestru 2024/2025

Pokyny pro vypracování

Homomorfní šifrování umožňuje provést operace se zašifrovanými daty tak, že výsledek je stejný,
jako by byly operace provedeny s původními otevřenými daty a výsledek následně zašifrován.
Je tedy možno pracovat pouze se zašifrovanými daty, bez nutnosti jejich dešifrování.
Tato myšlenka je teoreticky známa již více než třicet let, praktické použití se však potýká s řadou problémů.
V poslední době se tomuto tématu věnují renomované technologické společnosti i nově vznikající firmy (tzv. startupy).

1. Vysvětlete základní principy homomorfního šifrování.
2. Na základě dostupných informací vypracujte přehled o současném stavu.
3. Diskutujte možnosti praktické aplikace a kombinace s jinými kryptografickými metodami.
4. Demonstrujte praktickou ukázku s využitím open-source knihovny.
5. Zhodnoťte dosažené výsledky.

Elektronicky schválil/a prof. Ing. Róbert Lórencz, CSc. dne 26. ledna 2024 v Praze.

Bakalářská práce

APLIKACE HOMOMORFNÍHO ŠIFROVÁNÍ V PRAXI

Vojtěch Sedlák

Fakulta informačních technologií
Katedra informační bezpečnosti
Vedoucí: Ing. Miroslav Pospíšek, CSc.
15. května 2024

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2024 Vojtěch Sedlák. Všechna práva vyhrazena.
Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí a nad rámec oprávnění uvedených v Prohlášení, je nezbytný souhlas autora.

Odkaz na tuto práci: Sedlák Vojtěch. *Aplikace homomorfního šifrování v praxi.* Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2024.

Obsah

Seznam obrázků

Seznam tabulek

Rád bych poděkoval vedoucímu práce Ing. Miroslavu Pospíškovi, CSc za jeho trpělivost a odborné vedení bakalářské práce. Děkuji také všem svým přátelům a rodině za jejich podporu, kterou mi poskytovali během celého studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 15. května 2024

Abstrakt

Homomorfní šifrování je revoluční technika, která umožňuje provádět výpočty nad zašifrovanými daty bez nutnosti jejich předchozího dešifrování. Tato práce poskytuje přehled současného stavu a vysvětluje základní principy této šifrovací metody. Jsou zde představena tři základní homomorfní schémata: BFV, BGV a CKKS, včetně jejich vlastností, rozdílů a také výhod či nevýhod oproti klasickým šifrám. Dále se zde diskutuje o možnostech a vhodných oblastech využití této šifrovací techniky. Součástí práce je také praktická aplikace, která ukazuje použití homomorfního šifrování. K tomuto byla využita open source knihovna Microsoft SEAL. Na základě této aplikace byla provedena analýza, která se zaměřuje především na rychlost nebo velikost zašifrovaných dat.

Klíčová slova Homomorfní šifrování, praktické využití, BFV, BGV, CKKS, Microsoft SEAL

Abstract

Homomorphic encryption is a revolutionary technique that allows performing computations on encrypted data without the need for prior decryption. This paper provides an overview of the current state and explains the fundamental principles of this encryption method. It introduces three basic homomorphic schemes: BFV, BGV, and CKKS, along with their properties, differences, and advantages or disadvantages compared to traditional ciphers. Furthermore, the possibilities and suitable areas of application of this encryption technique are discussed. Additionally, the paper includes a practical application demonstrating the usage of homomorphic encryption. For this purpose, the open source Microsoft SEAL library was utilized. Based on this application, an analysis was conducted, focusing primarily on the speed and size of encrypted data.

Keywords Homomorphic encryption, practical usage, BFV, BGV, CKKS, Microsoft SEAL

Seznam zkratek

BFV	Brakerski-Fan-Vercauteren
BGV	Brakerski-Gentry-Vaikuntanathan
CKKS	Cheon-Kim-Kim-Song
FHE	Fully Homomorphic Encryption Plně Homomorfní Šifrování
GPU	Graphics Processing Unit
LSB	Least significant byte
LWE	Learning with Errors Učení s Chybami
MS	Microsoft
MSB	Most significant byte
PHE	Partially Homomorphic Encryption Částečně Homomorfní Šifrování
PIR	Private Information Retrieval Soukromé Vyhledávání Informací
RLWE	Ring-Learning with Errors Kruhové Učení s Chybami
SHE	Somewhat Homomorphic Encryption Poněkud Homomorfní Šifrování
ZKP	Zero-Knowledge Proof Důkaz s Nulovou Znalostí

Úvod

Ochrana dat a soukromí je v dnešní době velmi aktuálním tématem. S rostoucím množstvím digitálních informací a zvyšujícím se rizikem jejich zneužití se problematika šifrování stává nedílnou součástí každodenního života. Tradiční metody šifrování nabízejí efektivní způsob, jak zajistit důvěrnost dat, avšak tyto metody často přinášejí i určitá omezení. Jedním z těchto omezení je nemožnost provádět výpočty nad zašifrovanými daty bez předchozího dešifrování, což může být problematické zejména v případech, kde je třeba zachovat důvěrnost i při provádění výpočtů.

Homomorfní šifrování nabízí nový a inovativní přístup v zabezpečení dat. Tato metoda umožňuje provádět výpočty přímo nad zašifrovanými daty, což eliminuje potřebu jejich dešifrování a tím i potenciální riziko úniku citlivých informací. Tento koncept otevírá nové možnosti v oblastech jako je bezpečný outsourcing výpočtů, zabezpečení cloudových služeb nebo i kompletní anonymitu na internetu.

Téma jsem si zvolil díky jeho aktuálnosti a mému zájmu o nové technologie. Některé zdroje naznačují, že homomorfní šifrování je revoluční technika s potenciálem změnit celý technologický svět. V této práci se zaměřím na to, zda homomorfní šifrování skutečně může být efektivním nástrojem pro zajištění bezpečnosti dat, a také popíšu případné překážky, které by mohly bránit jeho širšímu uplatnění.

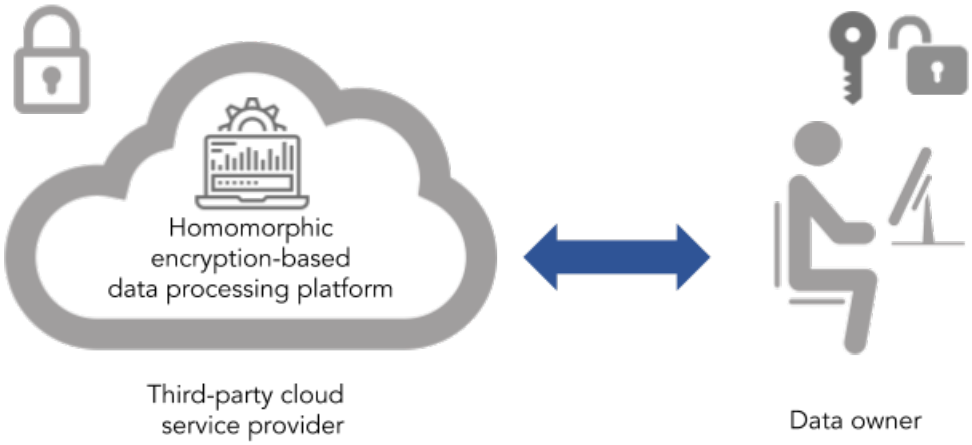
Cíl práce

Vzhledem k tomu, že homomorfní šifrování je v poslední době předmětem intenzivního výzkumu, cílem této práce je vytvořit přehled o současném stavu. Práce se zaměří na vysvětlení základních principů této šifrovací metody, představí existující schémata a zhodnotí výhody či nevýhody pro praktické využití. Součástí práce bude také diskuze, pro jaké programy by se homomorfní šifrování hodilo anebo s jakými dalšími kryptografickými metodami by bylo možné ho kombinovat.

Cílem praktické části je vytvořit jednoduchou aplikaci, která bude využívat principů homomorfního šifrování. K tomu bude využita některá z open source knihoven.

Kapitola 1
Základní přehled

Homomorfní šifrování umožňuje uživateli zašifrovat data a provádět na nich určité operace bez nutnosti odhalení původní zprávy. Typický příklad pro použití tohoto šifrování může být, kdy uživatel potřebuje zpracovat data v cloudu, ale z nějakého důvodu nedůvěřuje poskytovateli služby. Zde se nabízí možnost použít koncept homomorfního šifrování. Uživatel data zašifruje svým tajným klíčem a odešle je jako šifrovaný text na server. Server provede potřebné operace bez nutnosti dešifrování dat, a poté je odešle zpět uživateli. Uživatel je jediný, kdo zná tajný klíč, a tím pádem i jediný, kdo může data dešifrovat. Tímto způsobem je zajištěna bezpečnost dat po celou dobu zpracování v cloudu.



Obrázek 1.1 Příklad komunikace s nedůvěryhodným serverem. [1]

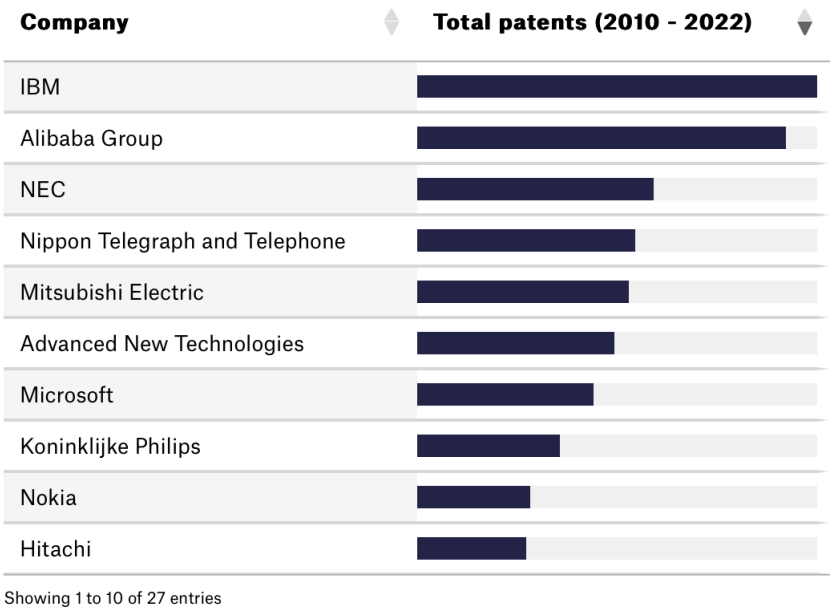
1.1 Historie

První zmínky o homomorfním šifrování se objevují už v roce 1978. Ronald L. Rivest, Len Adleman a Michael L. Dertouzos ve své práci [2] uvádějí koncept databanky, která by prováděla operace nad zašifrovanými daty. V této době se také začínají objevovat první částečně homomorfní šifry jako například RSA a ELGamal.

Až v roce 2009 Craig Gentry ve své disertační práci [3] popsal první plně homomorfní schéma, které umožňovalo provádět sčítání a násobení nad zašifrovanými daty a nebylo omezeno počtem operací. To se skládalo z několika kroků. Prvním bylo zkonstruování takzvaného poněkud homomorfního šifrovacího schématu. Operace nad zašifrovanými daty však přinášely určitý šum a tak Gentry přišel s konceptem bootstrappingu a dosáhl tak plně homomorfního schématu.

1.2 Současný vývoj

V posledních letech se homomorfní šifrování stává stále více populárnějším. Firmy si uvědomují jeho potenciál a investují velké množství času a peněz do výzkumu této technologie. S rostoucím zájmem roste také počet patentů, neboť společnosti se snaží chránit své inovace. Následující graf 1.2 zobrazuje společnosti s největším počtem udělených patentů v období od roku 2010 do roku 2022.



Obrázek 1.2 Poměr počtu udělených patentů v souvislosti s homomorfním šifrováním. Citovaný zdroj neuvádí konkrétní čísla [4]

Technologický gigant IBM drží v této oblasti dominantní postavení. Následovaný je společností Alibaba Group a japonskou firmou NEC. Nicméně homomorfnímu šifrování se věnují i další společnosti, které nejsou umístěny v žebříčku.

1.2.1 Zama

Za zmínku stojí společnost Zama. Tato firma se zaměřuje na výzkum TFHE schématu a vyvinula vlastní open source knihovnu. Jejich dlouhodobým cílem je zašifrování celého internetu pomocí homomorfního šifrování a zajistit tak kompletní anonymitu [5]. Kromě toho Zama navíc nabízí svůj „bounty and grant program“. Zde jsou vypsány různé výzvy, které mohou lidé plnit a přispět tak k vývoji homomorfních šifer.

1.3 Typy homomorfního šifrování

Homomorfní šifrování se dělí na tři hlavní typy podle toho, kolik a jaké operace je s nimi možné provádět [6].

1.3.1 Částečně homomorfní šifrování

Částečně homomorfní šifrování (PHE) je typ homomorfního šifrování, který umožňuje provádět pouze jednu konkrétní operaci nad zašifrovanými daty. Algoritmus může například podporovat sčítání nebo násobení, ale ne obě operace současně.

PHE algoritmy je relativně lehké zkonstruovat. Známá je například šifra RSA, která je multiplikativně homomorfní.

Důkaz. Pro šifrování se v RSA používá následující vztah:

$$C = m^e \mod n$$

Kde e je exponent, n je modul a m je zpráva kterou chceme zašifrovat. Pro násobení mocnin obecně platí následující vztah:

$$a^n \times b^n = (ab)^n$$

Pokud tedy vezmeme dva šifrované texty C_1 a C_2, které jsou šifrovány stejným klíčem, a vynásobíme je mezi sebou, dostaneme stejný výsledek jako kdybychom nejdříve vynásobili původní zprávy m_1, m_2 a následně je zašifrovali.

$$\begin{aligned} C_1 \times C_2 &= (m_1^e \mod n) \times (m_2^e \mod n) \\ &= (m_1^e \times m_2^e) \mod n \\ &= (m_1 \times m_2)^e \mod n \end{aligned}$$

□

Z tohoto důvodu je RSA multiplikativně homomorfní [6]. Existuje ale i několik dalších PHE šifer jako například El Gamal, Goldwasser–Micali cryptosystem, Benaloh cryptosystem a tak dále.

1.3.2 Poněkud homomorfní šifrování

Mezi poněkud homomorfní šifrování (SHE) se řadí již univerzálnější algoritmy. Oproti PHE umožňují provádět více než jednu určitou operaci, avšak jsou limitovány počtem provedení. Například algoritmus může podporovat desetkrát kombinaci sčítání nebo odčítání, ale po jedenácté již výsledek nebude dávat smysl. Zkonstruovat SHE algoritmus je značně jednodušší než zkonstruovat plně homomorfní algoritmus.

1.3.3 Plně homomorfní šifrování

Plně homomorfní šifrování (FHE) je nejsilnější typ homomorfního šifrování, které stejně jako SHE dokáže provádět více aritmetických operací nad zašifrovanými daty. Liší se v tom, že díky technice bootstrapping není limitován počtem provedení operací. První takovéto schéma zkonstruoval Craig Gentry a od té doby vzniklo několik dalších algoritmů, které se snaží tento původní algoritmus vylepšit.

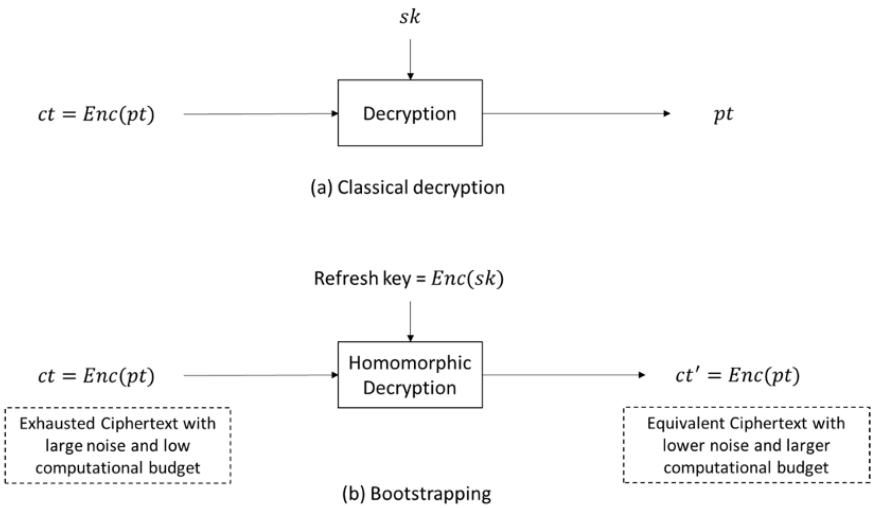
1.4 Noise budget

Noise (šum) je nepřesnost, která se hromadí během provádění homomorfních operací nad zašifrovanými daty. Každá homomorfní operace přidává určitý šum k zašifrovaným datům a pokud je tento šum příliš velký, může dojít k nemožnosti správného dešifrování dat [7].

Noise budget je tedy limitní hodnota, která určuje maximální množství šumu, který může být přítomen v zašifrovaných datech, aby bylo možné je úspěšně dešifrovat. Pokud úroveň šumu překročí daný noise budget, zašifrovaná data již nelze dešifrovat a jsou nepoužitelná.

1.5 Bootstrapping

Bootstrapping je technika, která umožňuje snížit úroveň šumu zašifrovaných dat, a tím umožnit provádění dalších homomorfních operací. Snížení šumu by se dalo vyřešit obyčejným dešifrováním a znovu zašifrováním dat. Nicméně tento přístup by znamenal poskytnout soukromý klíč druhé straně, nebo by musela tato strana posílat data s velkou úrovní šumu uživateli na přešifrování. Ani jedna z těchto variant není ideální, a proto se využívá homomorfního dešifrování. To se liší tím, že použijeme zašifrovaný soukromý klíč namísto obyčejného tajného klíče. Na výstupu pak nedostaneme plaintext, nýbrž stejný ciphertext, ale s menší úrovní šumu [8].



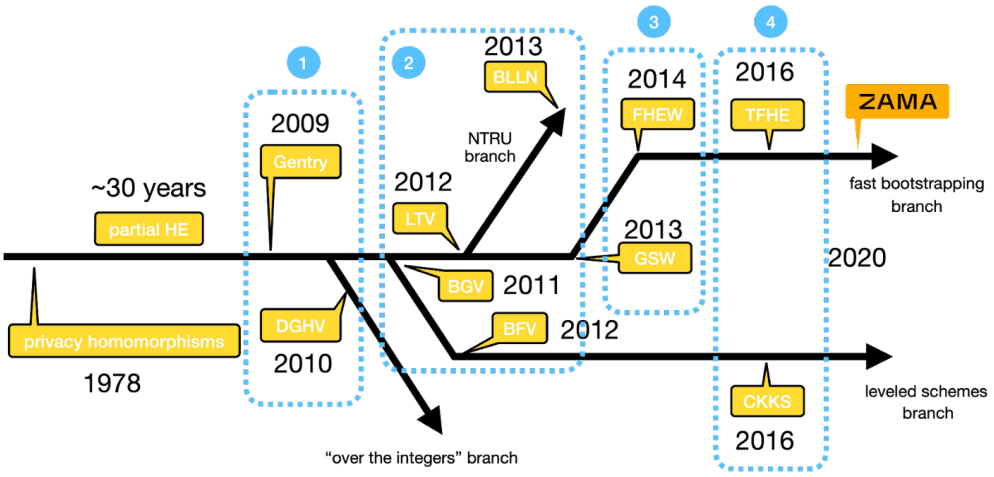
Obrázek 1.3 Znázornění procesu bootstrappingu. [9]

Bootstrapping je obvykle náročný proces, který vyžaduje značné množství výpočetních zdrojů. V Gentryho původním schématu trvala tato operace na jednoprocesorovém stroji až 30 minut [10]. Nicméně je to důležitá technika pro udržení funkčnosti homomorfního šifrování, protože umožňuje pracovat s daty i po provádění několika homomorfních operací a minimalizuje riziko ztráty dat kvůli akumulaci šumu.

Kapitola 2

Schémata

Po objevení plně homomorfního šifrování Craigem Gentrym vzniklo několik nových schémat, která se zaměřují na řešení problémů spojených s touto formou kryptografie. V dnešní době existují již 4 generace. Každá generace přináší nové techniky, optimalizace a inovace, které umožňují efektivnější a rychlejší provádění homomorfních operací. Na obrázku 2.1 je znázorněn dosavadní vývoj homomorfních schémat.



Obrázek 2.1 Vývoj homomorfních schémat. [11]

Jak lze vidět, vývoj se v poslední době dělí do dvou hlavních větví.

Fast bootstrapping branch: Tato větev má za cíl zrychlit operaci bootstrapping. Hlavními představiteli tohoto směru jsou FHEW a TFHE schémata. V porovnání s Gentryho schématem dosáhly razantního zrychlení, což je více přibližuje praktickému využití.

Leveled schemes branch: Tento směr se dívá na problematiku z druhého úhlu pohledu a zaměřuje se na příčinu hromadění šumu po provádění homomorfních operací. Schémata se snaží být efektivnější a bootstrapping se tudíž nemusí provádět tak často, nebo dokonce vůbec. Do tohoto směru se řadí BFV, BGV a CKKS schémata.

2.1 Základy bezpečnosti homomorfních schémat

Ještě než se podíváme na samotná schémata, je potřeba vysvětlit, na čem celá bezpečnost těchto schémat stojí. Většina je založena na problému Ring Learning With Errors (RLWE), což je specifický příklad Learning With Errors (LWE) problému. Oba tyto algoritmy jsou odolné i proti kvantovým počítačům a poskytují tak dobrý základ pro homomorfní schémata.

2.1.1 Learning With Errors

Pro lepší pochopení je dobré začít s Learning **without** errors metodou [12]. Stejně jako většina šifer je i tato založena na vlastnictví klíčů. Klíče jsou dva:

Soukromý klíč je vektor n čísel. Pro naši ukázku zvolíme $n = 4$ a vektor například 7, 35, 21, 19

Veřejný klíč je soubor několika rovnic o n neznámých. Pro naše $n = 4$ by veřejný klíč mohl vypadat například takto:

$$\begin{aligned} 53x + 12y + 42z + 9w &= 1844 \\ 34x + 60y + 3z + 15w &= 2686 \\ 8x + 31y + 72z + 4w &= 2729 \\ \vdots \quad & \quad \vdots \\ 36x + 22y + 25z + 41w &= 2326 \\ 2x + 76y + 29z + 43w &= 4100 \end{aligned}$$

Pokud bychom dosadili soukromý klíč do rovnic veřejného klíče, dostali bychom rovnost na obou stranách. Soukromý klíč je tedy řešením těchto rovnic. To ovšem nestačí, neboť takováto šifra není vůbec bezpečná. Stačily by nám pouze 4 rovnice a jednoduchým použitím lineární algebry bychom zjistili soukromý klíč.

Proto přichází na řadu metoda Learning **with** errors. Ta se liší v tom, že k pravé straně veřejného klíče je přičtena nějaká chyba.

$$\begin{aligned} 53x + 12y + 42z + 9w &= 1844 + 2 \\ 34x + 60y + 3z + 15w &= 2686 + -1 \\ 8x + 31y + 72z + 4w &= 2729 + 0 \\ &\vdots \quad \vdots \\ 36x + 22y + 25z + 41w &= 2326 + -3 \\ 2x + 76y + 29z + 43w &= 4100 + -2 \end{aligned}$$

To má za následek, že při dosazení soukromého klíče získáme pouze přibližný výsledek, nikoliv rovnost. Díky tomu nejsme schopni z veřejného klíče zjistit soukromý klíč, neboť do rovnic vstupují další proměnné a je tedy obtížné získat řešení jen na základě veřejného klíče. Dále se využívá modulární aritmetiky. V našem příkladu zvolíme například $\mod 91$. Po přičtení chyb a provedení modula získáme:

$$\begin{aligned} 53x + 12y + 42z + 9w &\approx 26 \\ 34x + 60y + 3z + 15w &\approx 46 \\ 8x + 31y + 72z + 4w &\approx 90 \\ &\vdots \quad \vdots \\ 36x + 22y + 25z + 41w &\approx 48 \\ 2x + 76y + 29z + 43w &\approx 3 \end{aligned}$$

2.1.1.1 Zašifrování bitu

Nyní se podíváme, jak pomocí LWE zašifrovat jeden bit, tedy 0 nebo 1. Představme si klasický scénář, kdy Bob bude chtít poslat Alici nějakou informaci. V první řadě Bob vybere z veřejného klíče několik rovnic a sečte je dohromady.

$$\begin{aligned} 34x + 60y + 3z + 15w &\equiv 46 \mod 91 \\ 2x + 76y + 29z + 43w &\equiv 3 \mod 91 \end{aligned}$$

$$36x + 45y + 32z + 58w \equiv 49 \mod 91$$

Pokud bude chtít Bob zašifrovat bit 0, pošle pouze tuto rovnici:

$$36x + 45y + 32z + 58w \equiv 49 \mod 91$$

Pokud bude chtít poslat bit 1, pošle rovnici:

$$36x + 45y + 32z + 58w \equiv 49 + 45 \mod 91$$

Číslo 45 se k pravé straně přičte proto, protože v modulu 91 je číslo 45 nejdále od čísla 0. Řekněme, že Bob poslal Alici bit 1 a tedy Alice obdržela rovnici:

$$36x + 45y + 32z + 58w \equiv 3 \mod 91$$

Nyní musí Alice ze zprávy vyčíst zašifrovaný bit. To provede tak, že do rovnice dosadí svůj soukromý klíč.

$$36 \times 7 + 45 \times 35 + 32 \times 21 + 58 \times 19 \equiv 52 \mod 91$$

Číslo 52 je tedy řešením této rovnice a to Alice odečte od rovnice Boba.

$$3 - 52 \equiv 42 \mod 91$$

Číslo 42 není ani bit 0 (0) ani 1 (45). To je způsobeno chybami, které byly přičteny k pravým stranám veřejného klíče. Výsledek proto osciluje někde okolo pravé hodnoty. To je důvod proč jsme zvolili kódování $0 \rightarrow 0$ a $1 \rightarrow 45$. Výsledek 42 je blízko pravé hodnoty 45 a proto Alice může bezpečně určit, že Bob zašifroval právě bit 1.

2.1.2 Ring Learning With Errors

Ring Learning With Errors neboli RLWE je specifický případ LWE [13]. Byl vymyšlen proto, aby se v zašifrovaném textu mohlo přenášet více informací. V základu funguje velmi podobně jako LWE, nicméně s tím rozdílem, že se zde používají polynomy. Rovnice vypadají takto:

$$\begin{aligned} a_0(x) \times s(x) + e_0(x) &= b_0(x) \\ a_1(x) \times s(x) + e_1(x) &= b_1(x) \\ \vdots \quad &\vdots \end{aligned}$$

Typický polynom $a(x)$ je vyjádřen jako:

$$a(x) = a_0 + a_1 x + a_2 x^2 + ... + a_{n-2} x^{n-2} + a_{n-1} x^{n-1}$$

$e(x)$ je neznámý polynom nízkého řádu. Stejně jako tomu bylo u LWE, tento polynom reprezentuje chybu a ztěžuje tak vyřešení rovnic. Polynom $s(x)$ je tajný polynom, kterým když vynásobíme $a(x)$ a přičteme chybu $e(x)$, tak dostaneme rovnosti s polynomy $b(x)$.

2.2 BFV

BFV (Brakerski-Fan-Vercauteren) je jedno z nejrozšířenějších a nejúčinnějších plně homomorfních šifrovacích schémat [14]. Umožňuje provádět operace sčítání a násobení nad zašifrovanými daty, přičemž násobení vyžaduje mnohem větší výpočetní výkon než operace sčítání. Schéma je založeno na RLWE problému a je implementováno nad dvěma kruhy:

- kruh s plaintexty,
- kruh se ciphertexty.

Homomorfní operace jsou možné díky funkci, která zachovává operace v prostoru ciphertextu C a prostoru plaintextu P.

BFV dokáže přesně pracovat nad celými čísly [15]. Zní to možná jako standard, ale některá homomorfní schémata dokáží určit výsledek jen přibližně. BFV ovšem dokáže zaručit přesný výsledek a hodí se tak pro aplikace, kde je kladem důraz na přesnost. Nejvíce se hodí pro práci s modulární aritmetikou, neboť BFV vrací výsledek po homomorfních operacích modulo p. Toto může být nevýhoda, protože ne vždy jsme schopni určit velikost p. Stejně tak může být nevýhoda i to, že BFV dokáže pracovat jen nad celými čísly. Pokud bychom potřebovali využít reálná čísla, bylo by potřeba zvolit jiné schéma.

2.2.1 Kryptografické funkce

BFV obsahuje následující kryptografické funkce [14]:

ParamGen(λ) $\rightarrow$ Params: Generátor parametrů vrací sadu šifrovacích parametrů používaných v BFV. Na vstupu dostane bezpečnostní parametr λ, který může nabývat hodnot 80, 128 a 256. Čím větší číslo, tím větší je i bezpečnostní úroveň BFV.

KeyGen(Params) $\rightarrow$ SK, PK, EK: Generátor klíčů dostává na vstupu sadu parametrů a vrací trojici klíčů. SK se používá pro dešifrování, PK se používá pro šifrování a EK se využívá k provádění homomorfních operací nad zašifrovanými daty. SK je tajný klíč, který by měl znát pouze vlastník dat, zatímco PK a EK jsou veřejné.

Encrypt(PK, M) $\rightarrow$ C: Šifrování dostává na vstupu PK klíč a plaintext M v prostoru P a vrací ciphertext v prostoru C.

Decrypt(SK, C) $\rightarrow$ M: Dešifrování naopak na vstupu dostává SK klíč a ciphertext v prostoru C a vrací původní plaintext v prostoru P.

EvalAdd(Params, EK, C_1, C_2) $\rightarrow$ C_3: Tato operace umožňuje provádět homomorfní sčítání. Na vstupu dostane Params, EK klíč a dva validní ciphertexty:

- C_1, který reprezentuje zašifrovanou zprávu M_1,
- C_2, který reprezentuje zašifrovanou zprávu M_2.

Na výstupu vrátí nový ciphertext C_3, který reprezentuje zašifrovaný součet zpráv $(M_1 + M_2)$.

EvalAddConst(Params, EK, C_1, M_2) $\rightarrow$ C_3: Tato operace je velmi podobná EvalAdd, ale s tím rozdílem, že na vstupu přijímá C_1 a plaintext M_2. Na výstupu opět vrací nový ciphertext C_3, který reprezentuje zašifrovaný součet zpráv $(M_1 + M_2)$.

EvalMult(Params, EK, C_1, C_2) $\rightarrow$ C_3: Tato operace umožňuje provádět homomorfní násobení. Na vstupu dostane Params, EK klíč a dva validní ciphertexty C_1, C_2. Na výstupu vrátí nový ciphertext C_3, který reprezentuje zašifrovaný součin zpráv $(M_1 \times M_2)$.

EvalAddMult(Params, EK, C_1, M_2) $\rightarrow$ C_3: Stejně jako u sčítání je tato operace variantou, která na vstupu přijímá ciphertext C_1 a plaintext M_2 a vrací nový ciphertext C_3, který reprezentuje zašifrovaný součin zpráv $(M_1 \times M_2)$.

Relinearize(Params, EK, C') $\rightarrow$ C: Relinearizace na vstupu očekává sadu parametrů, EK klíč a ciphertext C'. Po zpracování vrátí ciphertext C, který šifruje stejnou zprávu jako C', nicméně je v kompaktnějším formátu.

2.2.2 Relinearizace

BFV schéma má tu vlastnost, že provádění homomorfních operací (hlavně násobení) zvyšuje délku šifrovaného textu. To má negativní dopad na velikost dat a také na čas potřebný k provedení operací. Na začátku je délka šifrovaného textu 2 [16]. S tím, jak se postupně provádějí operace, velikost šifrovaného textu roste a přímo úměrně tomu se také zvyšuje paměťová i prostorová náročnost. Aby se předešlo tomuto růstu, BFV schéma podporuje operaci zvanou Relinearizace. Ta umožňuje zredukovat velikost šifrovaného textu zpět na velikost 2. Nicméně tato metoda má nevýhodu v tom, že její provedení vyžaduje podobný výpočetní výkon jako operace násobení. Existuje několik strategií, kdy relinearizaci provádět. Tento problém se nazývá „Relinearize problem“ a studie provedená Haem Chenem [17] ukázala, že se jedná o problém patřící do třídy NP-těžkých úloh.

2.3 BGV

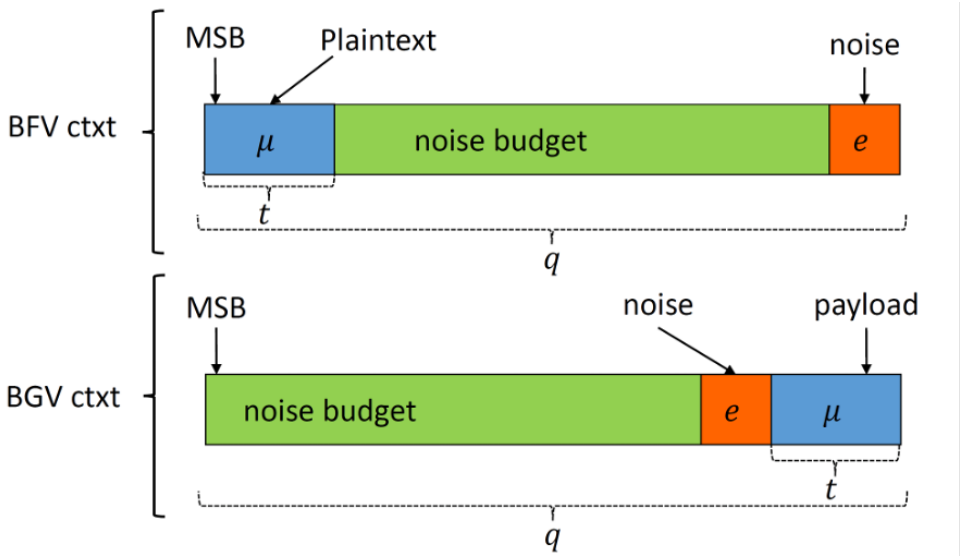
Brakerski-Gentry-Vaikuntanathan, neboli BGV, je další schéma, které patří do druhé generace plně homomorfních šifer. Stejně jako BFV umožňuje provádět operace sčítání a násobení nad zašifrovanými daty a jeho bezpečnost stojí na problému RLWE. Celkově jsou schémata hodně podobná, nicméně existuje několik rozdílů.

Největší rozdíl je v tom, že modulo šifrovaného textu zůstává v BFV schématu konstantní po celou dobu provádění homomorfních operací [18]. Naproti tomu v BGV schématu se toto modulo může měnit. Jak je vidět na následujícím obrázku 2.2, modulo šifrovaného textu se skládá z více malých modulů p_0–p_L a je spojeno s konkrétní úrovní. Nejčastější scénář je takový, že při zašifrování začíná na úrovni L a postupným prováděním operací se pohybuje směrem dolů. Existují ale i implementace, kde se může pohybovat nahoru i dolu.

$q_L = p_0 \cdot p_1 \cdot \ldots \ldots \ldots \ldots \cdot p_L$	**Encryption level** Level (L)
$q_{L-1} = p_0 \cdot p_1 \cdot \ldots \ldots \cdot p_{L-1}$	Level ($L-1$)
⋮	⋮
$q_l = p_0 \cdot p_1 \cdot \ldots \ldots \cdot p_l$	
⋮	⋮
$q_1 = p_0 \cdot p_1$	Level (1)
$q_0 = p_0$	**Decryption level** Level (0)

Obrázek 2.2 Modulo šifrovaného textu BGV. [18]

BFV a BGV schémata se také liší ve struktuře uložení šifrovaného textu. Zatímco BFV umisťuje zašifrovanou zprávu (payload) směrem k nejvýznamnějším bitům (MSB) šifrového textu, BGV ji umisťuje směrem k nejméně významným bitům (LSB). Rozdíl v ukládání obou schémat je znázorněn na obrázku 2.3.



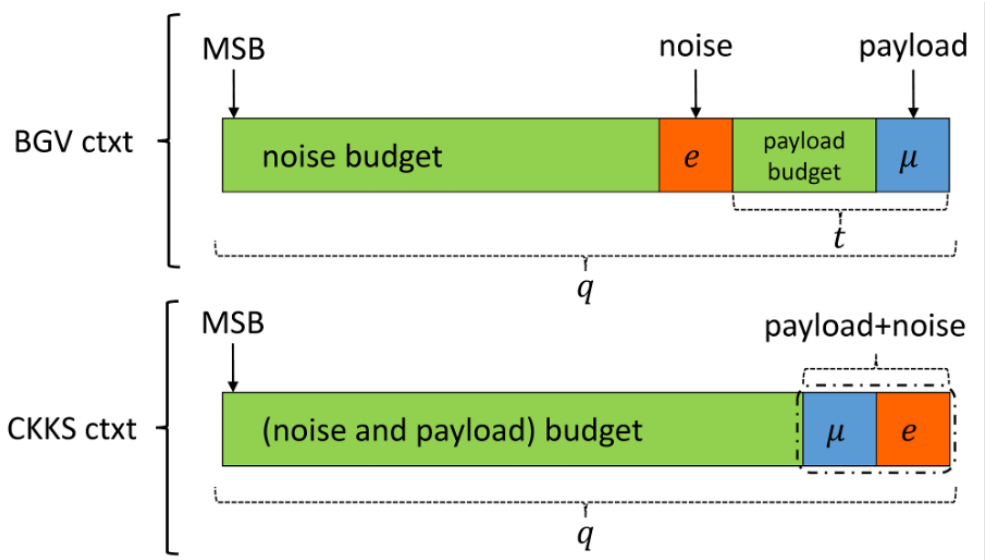
Obrázek 2.3 Struktura šifrovaného textu BFV a BGV. t je modulo plaintextu a q je modulo šifrovaného textu. [18]

Z obrázku je také vidět, jaký důsledek má zvětšování úrovně šumu (noise). Pokud provedeme homomorfní operace a šum vyčerpá všechno volné místo (noise budget), začne nám přetékat do zašifrované zprávy a dešifrování již nebude možné.

2.4 CKKS

CKKS neboli Cheon-Kim-Kim-Song je relativně nové plně homomorfní schéma, které se řadí do čtvrté generace FHE šifer. V porovnání s BFV nebo BGV schématy, které pracují pouze na celých číslech, přináší CKKS velkou výhodu, a tou je možnost provádět homomorfní operace nad reálnými čísly. Na druhou stranu je ale výsledek pouze přibližný.

Je to dáno jiným způsobem ukládání zašifrované zprávy (payload). Ta je spojena se šumem a následně jsou oba prvky zašifrovány společně [19]. Šifrovaný text tedy ztrácí přesnost díky šumu. V porovnání s daty je šum podstatně menší a chyba tudíž není tak velká. Principiálně je to stejné jako s ukládáním reálných čísel v současných systémech. Ty nejčastěji používají plovoucí desetinou čárku a přesnost reálného čísla je tak omezena na určitý počet míst.



Obrázek 2.4 Struktura šifrovaného textu BGV a CKKS. t je modulo plaintextu a q je modulo šifrovaného textu. [19]

Stejně jako u BFV nebo BGV schématu i zde platí, že operace sčítání je méně náročná než operace násobení. Zároveň nám násobení vnáší do výpočtů více nepřesnosti, což si můžeme ukázat na jednoduchém příkladu. Zvolme si dvě přirozená čísla například 2,37 a 2,59 a sečtěme je.

$$2,37 + 2,59 = 4,96$$

Vidíme, že se počet míst za desetinnou čárkou nezměnil. Součet bude mít tedy nejvíce tolik míst, kolik jich měl sčítanec. Nyní provedeme násobení.

$$2,37 \times 2,59 = 6,1383$$

Násobení nám zvětšilo dvojnásobně počet míst za desetinnou čárkou. Pokud bychom měli CKKS omezené pouze na dvě desetinná místa, museli bychom výsledek zaokrouhlit. Z tohoto důvodu nám operace násobení vnáší do výpočtů více nepřesnosti než operace sčítání. Nicméně i součet bude pouze přibližný díky již zmíněnému šumu, který je zakódován spolu se zprávou.

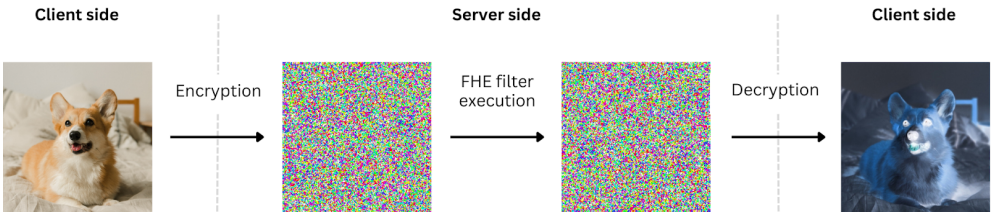
Kapitola 3

Využití homomorfního šifrování v praxi

Nyní už máme základní povědomí o tom, jak homomorfní šifrování funguje a jaké jsou typy schémat. V této kapitole se podíváme, jaké jsou možnosti praktického využití.

3.1 Úprava fotografií

Homomorfní šifrování by mohlo nalézt využití ve zpracování obrazu. Myšlenka spočívá v tom, že bychom mohli mít například internetovou stránku, kam bychom nahráli fotografii, zvolili požadovaný filtr (černobílý, inverze...) a stránka by nám vrátila upravenou verzi.



Obrázek 3.1 Zpracování obrázku v zašifrované podobě. [20]

Například černobílý filtr se může implementovat tak, že se z RGB složek každého pixelu vypočítá vážený průměr, který určuje stupeň šedi. Tyto operace můžeme bez problému provést s využitím homomorfního šifrování. Server tedy může obdržet zašifrované pixely, provést výpočet stupně šedi a vrátit černobílé pixely klientovi. Nevýhodou tohoto řešení je opět větší náročnost na výpočetní výkon a také na paměť. Společnost Zama vytvořila ukázkovou aplikaci, kde se tato úprava fotek dá vyzkoušet [20]. Testovací obrázek o rozměrech 100×100

pixelů měl v zašifrované podobě přes 180 Mb a aplikování filtru trvalo na osmi-procesorovém stroji téměř tři vteřiny.

3.2 Zero-Knowledge Proof

Zero-knowledge proof (ZKP) je kryptografický protokol, který umožňuje jedné straně (prover) prokázat druhé straně (verifier), že zná určitý fakt, aniž by při tom odhalil konkrétní informace o tomto tvrzení. ZKP je poměrně složitá věc a podrobné vysvětlení není cílem této práce. Nicméně rád bych zde nastínil, jak může proběhnout ověření znalosti nějakého polynomu [21] a jak se při tom dá využít homomorfního šifrování.

3.2.1 Prokázání znalosti polynomu

Představme si situaci, kdy prover bude chtít dokázat znalost nějakého polynomu. Každý polynom může být faktorizován ve tvaru:

$$(x - a_0) \times (x - a_1) \times ... \times (x - a_n)$$

Prover bude chtít například dokázat, že zná polynom $p(x)$, který má kořeny 1 a 2. To znamená že se jeho polynom dá zapsat ve tvaru:

$$(x - 1) \times (x - 2) \times ...$$

Zaveďme si polynom $t(x)$, který obsahuje pouze tyto dva kořeny.

$$t(x) = (x - 1) \times (x - 2)$$

Pokud bude chtít prover dokázat, že jeho polynom $p(x)$ má opravdu kořeny 1 a 2, pak se musí dát $p(x)$ vyjádřit jako

$$p(x) = t(x) \times h(x)$$

Pokud se proverovi nepodaří najít polynom $h(x)$, který by vyhovoval této rovnici, znamená to, že polynom $p(x)$ neobsahuje tyto kořeny.

Nyní přejdeme k samotnému důkazu. Postup je následující:

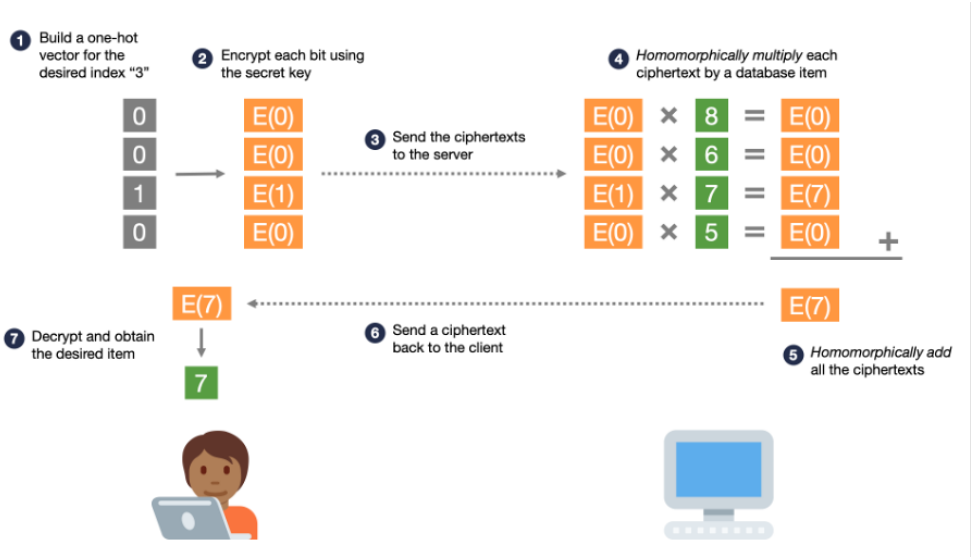
- Verifier si vymyslí náhodnou hodnotu r a spočítá $t = t(r)$. Hodnotu r předá Proverovi.
- Prover spočítá hodnoty $h = h(r)$ a $p = p(r)$ a pošle je zpět Verifierovi.
- Ten si ověří, že $p = t \times h$ a pokud ano, může proverovi věřit.

3.2.2 Využití homomorfního šifrování

Výše popsaný důkaz má několik nedostatků. Například prover vůbec nemusí znát polynom $p(x)$. Pokud si spočítá $t = t(r)$ a zvolí takové h, aby platila rovnost $p = t \times h$, verifier tento fakt přijme. Problém je v tom, že prover zná hodnotu r a $t(r)$. Potřebovali bychom zajistit, aby prover tyto hodnoty neznal, ale přesto byl schopný provádět operace s těmito hodnotami. Přesně k tomu nám slouží homomorfní šifrování.

3.3 Private Information Retrieval

Private Information Retrieval (PIR) je protokol, který umožňuje získat informace uložené v databázi serveru, aniž by provozovatelé databáze věděli, které konkrétní informace si uživatel vyžádal. Stejného cíle se dá dosáhnout i stažením celé databáze a následného vybrání prvků, které nás zajímají. To je ovšem pracné a časově náročné, neboť některé databáze mohou obsahovat desítky Gb [22]. Následující obrázek 3.2 ukazuje, jak je v tomto protokolu možné využít homomorfní šifrování.



Obrázek 3.2 Ilustrace funkcionality protokolu PIR. [23]

Příklad pracuje s databází, která obsahuje 4 prvky, a uživatel chce přistoupit k třetímu prvku. Uživatel tedy vytvoří pole, kde jsou samé nuly, a pouze na třetí pozici je hodnota 1. Toto pole homomorfně zašifruje a odešle serveru. Server každou položku pole homomorfně vynásobí s prvkem databáze a sečte všechny položky dohromady. Protože všechny kromě třetí položky měly hodnotu 0, součet bude obsahovat pouze třetí prvek databáze. Ten server odešle zpět klientovi který si ho může dešifrovat a přečíst.

Toto je pouze high-level pohled, ale ukazuje to, jak může být zajištěno anonymní prohlížení databáze na internetu. Webové stránky tedy nemusejí slibovat, že o uživateli nebudou ukládat data, ale mohou to kryptograficky zaručit.

3.4 Strojové učení

Strojové učení je oblast umělé inteligence, která umožňuje naučit stroje řešit nějaký problém bez nutnosti jejich naprogramování. Pracuje nad velkým množstvím dat, která analyzuje a vytvoří modely, které se pak dají použít k pozdějšímu rozhodování v podobných situacích. Počítače jsou díky tomu schopni analyzovat ručně psaný text, rozeznat různé druhy zvířat a tak podobně.

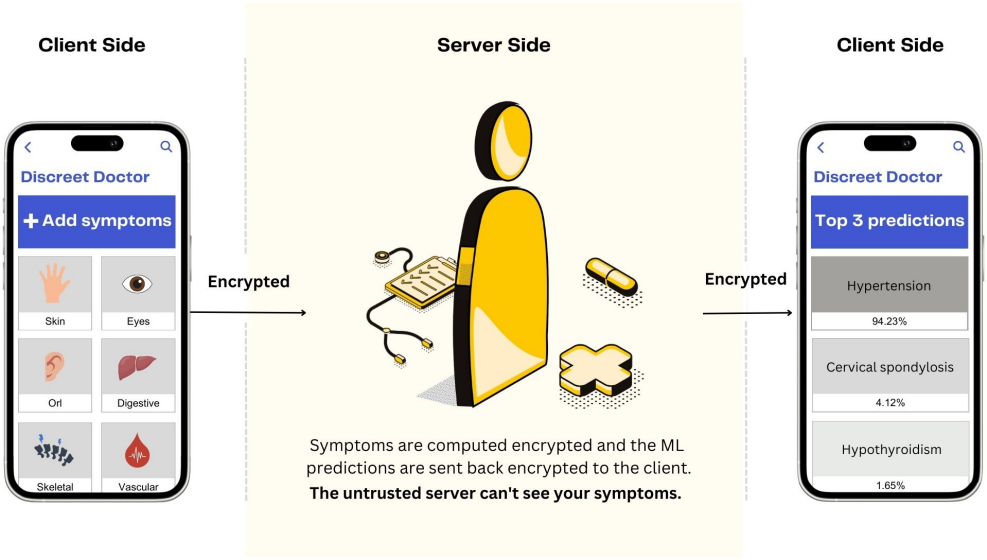
Problémem strojového učení je právě nutnost mít k dispozici velké množství dat, která často obsahují i citlivé informace. Tento problém může opět vyřešit homomorfní šifrování, neboť to umožňuje trénovat stroje na zašifrovaných datech. Data tak zůstanou v bezpečí, a přesto bude možné provádět strojové učení i na citlivých datech jako například v oblasti financí nebo zdravotnictví.

V roce 2018 se společnost IBM rozhodla otestovat tuto metodu a aplikovala strojové učení nad zašifrovanými daty v jedné brazilské bance [24]. Nejdříve homomorfně zašifrovali data s modelem a potvrdili, že je možné provádět strojové učení se stejnou přesností jako bez šifrování. Následně pak model učili a dokázali analyzovat data jako například kolik a za co klient utrácí své úspory.

3.5 Zdravotnictví

Data ve zdravotnictví patří k těm nejcitlivějším datům vůbec. Pacienti svěřují poskytovatelům zdravotní péče intimní informace o svém zdraví a jakékoliv narušení této důvěry může znamenat velký problém. Pokud pacienti uvidí, že záznamy o jejich zdraví nejsou dostatečně chráněny, mohou přestat doktorům důvěřovat a přestat s nimi sdílet své zdravotní problémy. To pak bude mít negativní vliv na kvalitu celého zdravotního systému. Dalším důvodem mohou být i peníze. Pokuty za úniky takto citlivých informací jsou obrovské a mohou tak zatížit zdravotnické organizace.

Zdravotnictví je tedy dostatečně motivováno k ochraně dat svých pacientů. I když je možné citlivé záznamy zašifrovat pomocí běžných šifer, mnoho nemocnic a institucí spoléhá na třetí strany, které s daty pracují.



Obrázek 3.3 Aplikace od společnosti ZAMA, která homomorfně zpracovává zdravotnická data. [25]

Tyto externí společnosti často vyžadují přístup k nezašifrovaným datům, aby je mohly zpracovat. Homomorfní šifrování však umožňuje zpracování dat i v zašifrované podobě, což snižuje riziko úniku informací o pacientech.

3.6 Elektronické volby

Volby jsou základním pilířem demokratického systému. Voliči prostřednictvím hlasů určují, kdo je bude zastupovat a jakým směrem se bude jejich země ubírat. Součástí demokratických voleb je nejenom právo občanů volit, ale také zaručení jejich soukromí při výkonu tohoto práva. Každý volič by měl mít jistotu, že jeho hlas je utajen a že nikdo nemá možnost zjistit, pro koho nebo pro co hlasoval.

Kromě tradičního fyzického hlasování ve volebních místnostech se stále častěji objevují i moderní formy elektronického hlasování, které umožňují voličům hlasovat online z pohodlí svého domova. Pro zajištění bezpečnosti těchto voleb je klíčové, aby volební systémy byly odolné vůči kybernetickým útokům a zaručovaly důvěrnost a integritu hlasovacích dat. Tradiční šifrovací metody sice umí účinně ochránit data, avšak pro sečtení hlasů je potřeba data nejprve dešifrovat. Homomorfní šifrování by tento problém mohlo vyřešit, neboť sčítání hlasů by mohlo probíhat i nad zašifrovanými daty.

V únoru 2020 společnost Microsoft otestovala homomorfní šifrování v rámci projektu Microsoft ElectionGuard [26]. Cílem bylo dokázat voličům, že jejich hlasy jsou zabezpečené a že byly započítány do celkového počtu hlasů. Po volbách voliči obdrželi jedinečný kód, který jim umožnil přístup k platformě pro ověření výsledků.

Kapitola 4
Praktická ukázka

Tato kapitola se zaměřuje na demonstraci praktické ukázky homomorfního šifrování. Cílem bylo využít již existující open source knihovnu a naprogramovat aplikaci, která homomorfní šifrování využívá.

4.1 Open source knihovny

Důležitým krokem pro naprogramování praktické aplikace je volba open source knihovny. Proto zde uvedu stručný přehled několika známých a volně dostupných knihoven.

4.1.1 OpenFHE

OpenFHE je multiplatformní open source knihovna napsaná v jazyce C++, která podporuje operační systémy Windows, Linux a MacOS. Knihovna je poměrně bohatá a implementuje pět typů homomorfních schémat: BFV, BGV, CKKS, FHEW, TFHE [27]. Ke knihovně je dostupná dokumentace, kde je přehledně popsána funkčnost knihovny i návod k instalaci. Na webových stránkách této knihovny jsou dokonce pořádány webináře, kde se uživatele mohou dozvědět nejen o nových funkcích OpenFHE knihovny, ale také nové informace ze světa homomorfního šifrování.

4.1.2 Microsoft SEAL

Microsoft SEAL (dále jen MS SEAL) je open source knihovna vyvinutá výzkumnou skupinou kryptografie a ochrany soukromí ve společnosti Microsoft. Je napsaná v jazyce C++ a je kompatibilní s operačními systémy Windows, Linux a MacOS. Narozdíl od OpenFHE podporuje pouze tři homomorfní schémata: BFV, BGV a CKKS [28]. MS SEAL sice nemá dostupnou dokumentaci, nicméně praktické ukázky použití této knihovny obsahují bohaté komentáře,

které dokáží dostatečně vysvětlit její fungování. Na webových stránkách Microsoftu můžeme najít i přednášku [29], kde jeden z vývojářů MS SEAL, Kim Laine, vysvětluje homomorfní šifrování a popisuje funkcionalitu této knihovny.

4.1.3 TFHE-rs

TFHE-rs je knihovna od společnosti ZAMA, která je napsána v jazyce Rust a opět je kompatibilní s operačními systémy Windows, Linux a MacOS. Jak už název vypovídá, jedná se o implementaci TFHE schématu a žádné jiné schéma knihovna nepodporuje. Na této knihovně je zajímavé to, že nová verze poskytuje podporu pro GPU (Graphics Processing Unit) s CUDA implementací [30]. GPU může pomoci urychlit výpočty prováděné nad zašifrovanými daty, což je výhodné, neboť rychlost výpočtů je jedním z největších problémů homomorfního šifrování.

4.1.4 HElib

HElib je další open source knihovna vyvinutá společností IBM, je napsána v jazyce C++ a podporuje operační systémy Linux a MacOS. Knihovna podporuje pouze BGV a CKKS schémata a soustředí se na jejich optimalizaci [31]. Zajímavá může být podpora „jazyka symbolických adres pro HE“, která poskytuje nízkoúrovňové příkazy jako set, add, shift...

4.2 Volba knihovny

Při volbě knihovny pro svou aplikaci jsem měl požadavek, aby byla knihovna kompatibilní s MacOS a poskytovala C++ API. Zároveň jsem chtěl implementovat BFV schéma a tudíž pro mě TFHE-rs od ZAMY nebyla dobrou volbou. Pro mé účely nejvíce vyhovovaly knihovny OpenFHE a MS SEAL. Bohužel, při instalaci OpenFHE jsem narazil na problémy a tudíž jsem zvolil MS SEAL. S touto knihovnou se jednoduše pracuje, instalace proběhla bez problémů a celkově poskytuje dostatečné nástroje, které pro svou jednoduchou ukázku homomorfního šifrování potřebuji.

4.3 Srovnání Microsoft SEAL s MerMer

V průběhu psaní bakalářské práce se mi dostalo srovnání s kanadskou firmou PrivID. Tato firma se zaměřuje na zabezpečení dat a mimo jiné vyvíjí systém MerMer [32], který kombinuje techniky zero-knowlage proof a homomorfního šifrování. MerMer sice není open source, ale PrivID mi poskytl jednoduchý benchmark test, abych mohl provést porovnání se svou volbou knihovny.

První část testu obsahovala zašifrování 30 čísel (0-29). Dále se provedly jednoduché operace sčítání dle následujícího vzorečku:

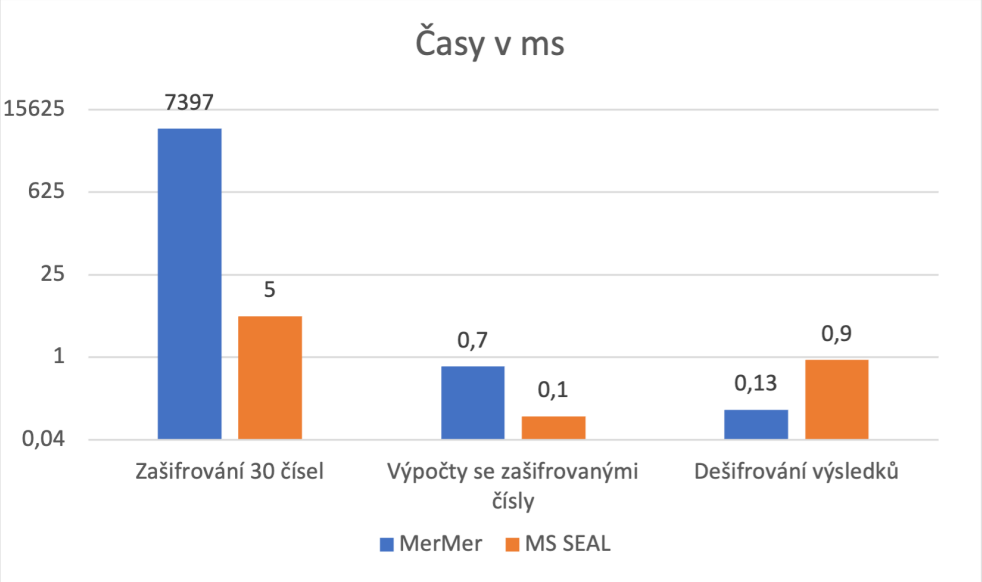
$$x_i = n_i + n_{i+1}$$

Posledním krokem bylo opět dešifrování výsledných hodnot. Bohužel, testy obou knihoven probíhaly na rozdílných zařízeních. Test MS SEAL probíhal na počítači MacBook Air M1 8 Gb RAM a kolegové z PrivID provedli test na AMD Ryzen 5 5600X 6-Core Processor, 32GB RAM.

Operation	**M1**	**AMD**
Integer math	33.7 GOps/sec	72.3 GOps/sec
Floating point math	35.9 GOps/sec	39.5 GOps/sec
Find prime numbers	142M Primes/sec	130M Primes/sec
Random string sorting	21.4M Strings/sec	26.5M Strings/sec
Data encryption	8.6 GBytes/sec	15.7 GBytes/sec
Data compression	170.6 MBytes/sec	252.5 MBytes/sec
Physics	1388 Frames/sec	1325 Frames/sec
Extended instructions	6.7B Matrices/sec	16.9B Matrices/sec

Tabulka 4.1 Tabulka srovnání procesorů, data jsou přebrány z [33], [34]

V tabulce 4.1 je vidět porovnání výkonosti obou procesorů, přičemž AMD je v porovnání s M1 o něco výkonnější. Výsledné časy po provedení příkladu na obou knihovnách tedy musejí být brány pouze přibližně.



Obrázek 4.1 Výsledné časy porovnání obou knihoven

Jak je vidět z grafu 4.1, největší rozdíl je v čase potřebným pro zašifrování dat. Přestože test MerMer byl proveden na výkonnějším procesoru, potřeboval k zašifrování dat téměř 1500x více času. PrivID tento fakt okomentoval s tím, že používají TFHE a ZKP a ne jenom FHE. Stejně tak výpočty nad zašifrovanými daty jsou ve prospěch MS SEAL, které k tomu potřebovalo zhruba 7x méně času. Na druhou stranu MerMer má lépe zvládnuté dešifrování, které je v porovnání se zašifrováním a výpočty poměrně levnou záležitostí.

Celkově mi toto srovnání potvrdilo, že MS SEAL je poměrně kvalitní knihovna a že jsem ji pro svou aplikaci zvolil dobře.

4.4 Ukázková aplikace

Jedním z možných využití homomorfního šifrování může být provádění cloudových výpočtů nad zašifrovanými daty. Tímto jsem se nechal inspirovat, a proto jsem naprogramoval jednoduchou cloudovou kalkulačku.

4.4.1 Popis aplikace

Praktická ukázka se skládá ze dvou klíčových částí: klienta a serveru. Obě tyto části jsou napsané v jazyce C++ a pro komunikaci mezi sebou používají protokol TCP, který zajišťuje spolehlivý přenos dat.

Obrázek 4.2 Ukázkový běh aplikace

Aplikaci je možno zkompilovat v několika variantách. Základní kompilace obsahuje pouze funkce potřebné k provedení homomorfního šifrování a dialog, který uživatele provede aplikací. Druhá možnost je překlad DEBUG. Ten vypisuje navíc několik dalších informací o programu. Serverová část pod tímto překladem vypisuje informaci, jak dlouho která operace trvala vyhodnotit, a klientská část vypisuje například informaci o stavu noise budgetu nebo o zadaných parametrech. Poslední možností je překlad RELINEARIZE. Ten zajistí, že po každé operaci násobení nebo mocnění se provede relinearizace a šifrovaný text pak nebude růst na velikosti.

4.4.2 Server

Serverová část aplikace je zodpovědná za provádění samostatných výpočtů. Díky homomorfnímu šifrování server pracuje jen se zašifrovanými daty a nemá tak přístup k původním hodnotám.

Po základním síťovém nastavení server čeká na připojení klienta. V implementaci jsou použita vlákna, aby měl server možnost obsluhovat více klientů současně a nedošlo tak denial of service po připojení prvního uživatele. Po připojení klienta server očekává zašifrované první číslo. Následuje operace, kterou chce klient provést, přičemž jsou podporované 4 typy:

- sčítání,
- odčítání,
- násobení,
- mocnění.

Poté, co klient zvolí operaci, server opět očekává zašifrovaná data, provede požadovanou operaci a zašle výsledek zpět uživateli.

4.4.3 Klient

Klientská část aplikace slouží k interakci s uživatelem a odesílání požadavků na server. Po vygenerování potřebných šifrovacích klíčů a připojení k serveru je uživatel vyzván k zadání čísel a výběru operace, kterou chce provést. Data se v zašifrované podobě odešlou na server, který je zpracuje a pošle zpátky výsledek. Proces je možné opakovat nicméně jen do té doby, než klesne noise budget na hodnotu 0. Poté již dešifrovaná data nemusejí dávat smysl.

Kapitola 5
Měření

V této kapitole jsem svou aplikaci podrobil několika testům a zkoumal, jak výkonné homomorfní šifrování vlastně je. Všechny testy byly provedeny na počítači Macbook Air M1 8 Gb RAM.

5.1 Parametry Microsoft SEAL

V knihovně MS SEAL je možné nastavit 3 základní parametry:

- Polynomial modulus,
- Coefficient modulus,
- Plaintext modulus.

V této sekci jsem se podrobněji zaměřil na jejich nastavení a popsal, jaký vliv mají na chod programu.

5.1.1 Polynomial modulus

Stupeň polynomial modulu (dále jen poly modulu) je parametr v knihovně MS SEAL, který má zásadní vliv na zašifrovaná data. Musí to být kladná mocnina dvou, přičemž Microsoft doporučuje zvolit číslo v rozsahu 1024–32768 [28]. Čím větší stupeň zvolíme, tím budou zašifrovaná data větší a operace pomalejší. Na druhou stranu budeme mít k dispozici větší noise budget a tím i možnost provádět složitější operace.

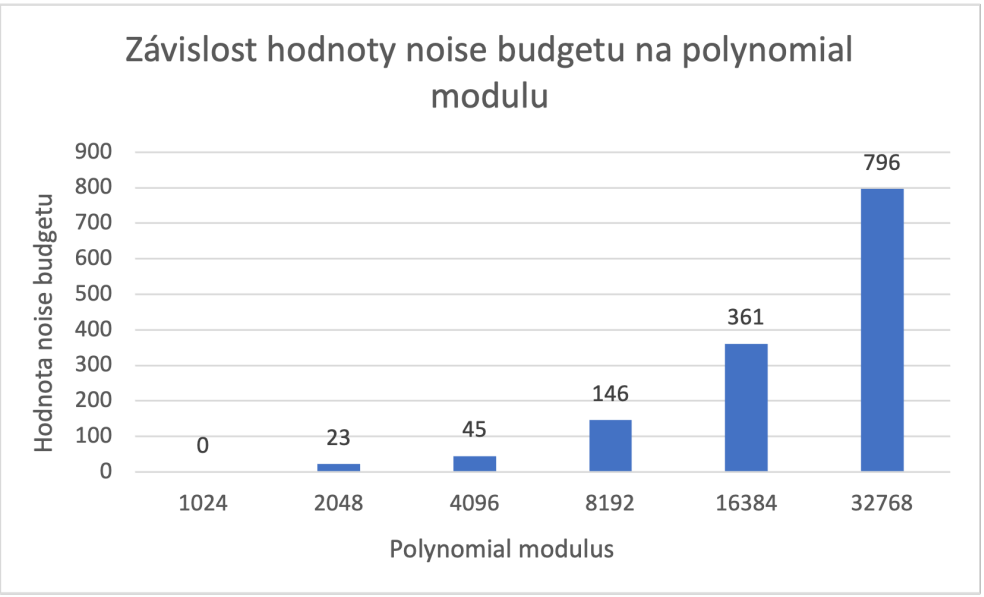
Na svém programu jsem provedl řadu testů a zjišťoval, jak se mění určitá data při zvětšování či zmenšování stupně poly modulu a výsledky jsem zaznamenal do následující tabulky 5.1. Plaintext modulus byl nastaven na hodnotu 1048576.

Stupeň polynomial modulu	Noise budget	Rychlost operace násobení (ms)	Velikost zašifrovaných dat (byte)	Velikost relinearizačního klíče (byte)
1024	0	1,6	8639	0
2048	23	2,4	30898	0
4096	45	5,8	88556	276949
8192	146	16	432357	2168059
16384	361	32	1826329	16456114
32768	796	149	7405121	118506575

Tabulka 5.1 Výsledky měření pro různé hodnoty polynomial modulu

5.1.1.1 Noise budget

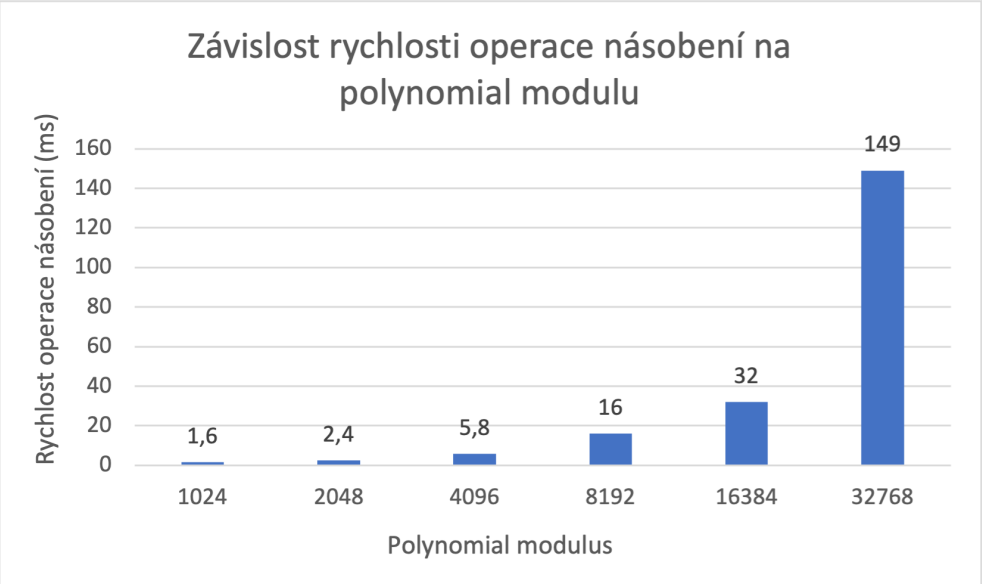
Hlavním důvodem zvětšení stupně poly modulu je právě noise budget, který určuje počet operací, které můžeme provést. Například je zajímavé, že při zadaném plaintext modulu a stupni poly modulu 1024 nemůžeme provést ani jednu jedinou operaci. Už jen samotné zašifrování a následné dešifrování nám nemusí dát stejný výsledek. Naopak při maximální velikosti stupně poly modulu již máme dostatečné množství noise budgetu, které nám vystačí i na poměrně složité výpočty.



Obrázek 5.1 Graf noise budgetu

5.1.1.2 Rychlost operace násobení

Násobení trvá oproti operacím sčítání/odčítání značně déle, a proto jsem se zaměřil na jeho časovou náročnost při změně stupně poly modulu. Již při nejmenším možném stupni trvá jedna operace násobení 1,6 ms, což je poměrně dlouhá doba. Při dvojnásobném zvětšení stupně poly modulu se přibližně dvakrát zvětší i potřebný čas k provedení operace. Nicméně výrazný nárůst nastává při změně stupně z 16384 na 32768, kdy se doba násobení téměř pětinásobně zvýší.

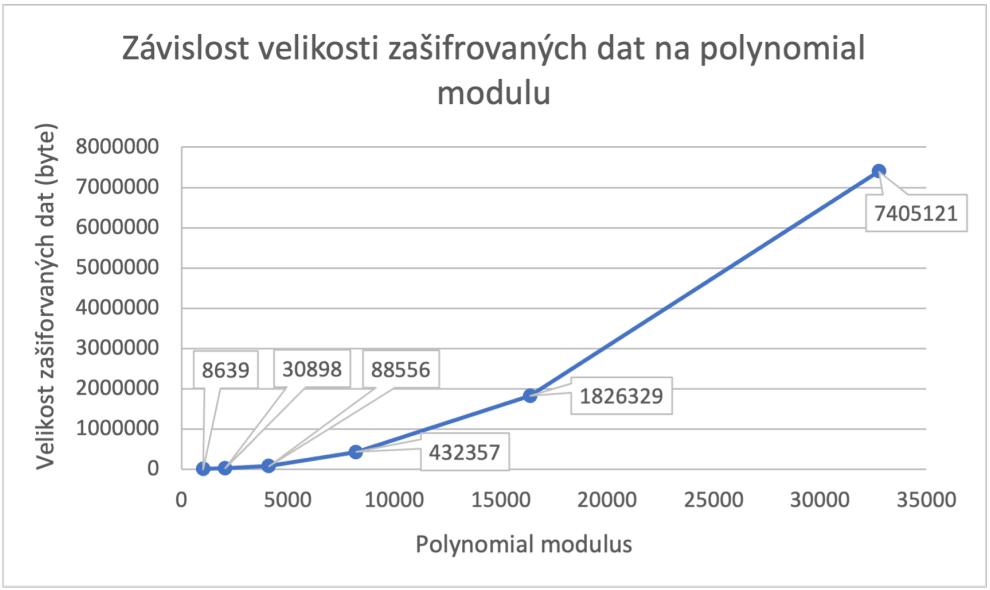


Obrázek 5.2 Graf rychlosti operace násobení

5.1.1.3 Velikost zašifrovaných dat

Velikost zašifrovaných dat je další velký problém homomorfního šifrování. Pokud bychom to porovnali s některými standardními šiframi, jako například se šifrou AES, uvidíme, že provádění homomorfních operací vyžaduje určitou daň. Šifra AES zachovává velikost dat. To znamená, že pokud na vstupu dostaneme n bytů, tak po provedení šifrování bude výstup obsahovat také n bytů. Případně může dojít k přidání paddingu pro zarovnání do bloků.

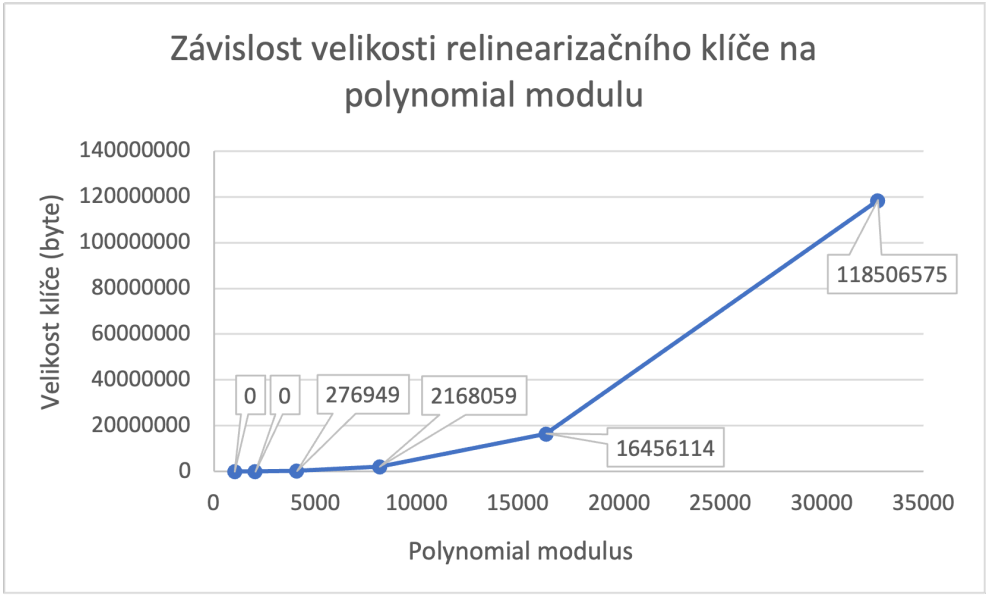
BFV schéma však velikost dat nezachovává. Už při nejmenším možném stupni poly modulu je vidět výrazný růst velikosti. Na vstupu jsem programu dal číslo 2, a po zašifrování měl výstup 8639 bytů. Jak již víme, tento stupeň ovšem nestačí ani k provádění základních operací. Pokud zvolíme stupeň, který nám poskytne více možností, například 8192, výsledný šifrovaný text bude mít 422 kB.



Obrázek 5.3 Graf velikosti zašifrovaných dat

5.1.1.4 Velikost relinearizačního klíče

Relinearizační klíč využívá MS SEAL k provádění relinearizace nad zašifrovanými daty. Tento proces je vykonáván serverem po dokončení operace, nicméně klíče generuje klient. Ten je tedy musí serveru přenést přes síť. Velikost klíče se tedy neprojevuje jen na velikosti potřebné paměti, ale i na výkonosti aplikace kvůli síťovému přenosu. Při stupni poly modulu 1024 a 2048 není možné relinearizační klíč vůbec zkonstruovat. Generování klíče vyhodí výjimku „keyswitching is not supported by the context“. U vyšších stupňů poly modulu je vidět, kolik bytů klíč zabírá. Při maximálním stupni je to dokonce více než 113 MB. To se negativně projeví na výkonosti aplikace, protože odeslat 113 MB na server určitý čas zabere.



Obrázek 5.4 Graf velikosti relinearizačního klíče

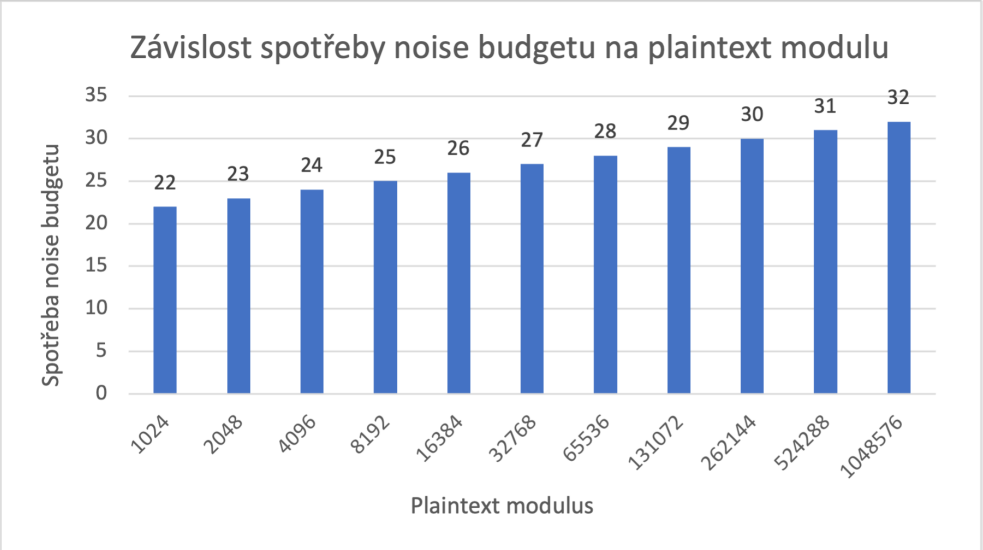
5.1.2 Coefficient modulus

V knihovně MS SEAL je třeba nastavit coefficient modulus. Jedná se o číslo, které je výsledkem součinu různých prvočísel, přičemž každé z těchto prvočísel má maximálně 60 bitů [28]. Tento parametr má vliv na velikost noise budgetu, kde větší hodnota coefficient modulu znamená rozsáhlejší noise budget. Nicméně parametr je shora omezen stupněm poly modulu a lze ho tedy pouze snižovat. Pro nastavení tohoto parametru Microsoft poskytuje pomocnou funkci, která nastaví vždy optimální hodnotu coefficient modulu. Z tohoto důvodu jsem se rozhodl změny parametru netestovat.

5.1.3 Plaintext modulus

Plaintext modulus je posledním parametrem BFV schématu, který je u MS SEAL potřeba nastavit. BFV počítá s čísly modulo nějakou hodnotou, a tu udává právě tento parametr. Z toho plyne, že čím větší hodnotu zvolíme, tím větší číslo můžeme získat. Opět platí, že pokud zvýšíme hodnotu parametru, projeví se to negativně na jiných vlastnostech. Zvětšení plaintext modulu povede k větší konzumaci noise budgetu.

Operace sčítání/odčítání snižuje hodnotu noise budgetu pouze minimálně. Rozhodl jsem se tedy provést test s operací násobení, kdy jsem zkoumal, jak moc noise budgetu spotřebuje výpočet příkladu 2×2 při různých hodnotách plaintext modulu. Stupeň poly modulu byl nastaven na 8192.



Obrázek 5.5 Graf spotřeby noise budgetu

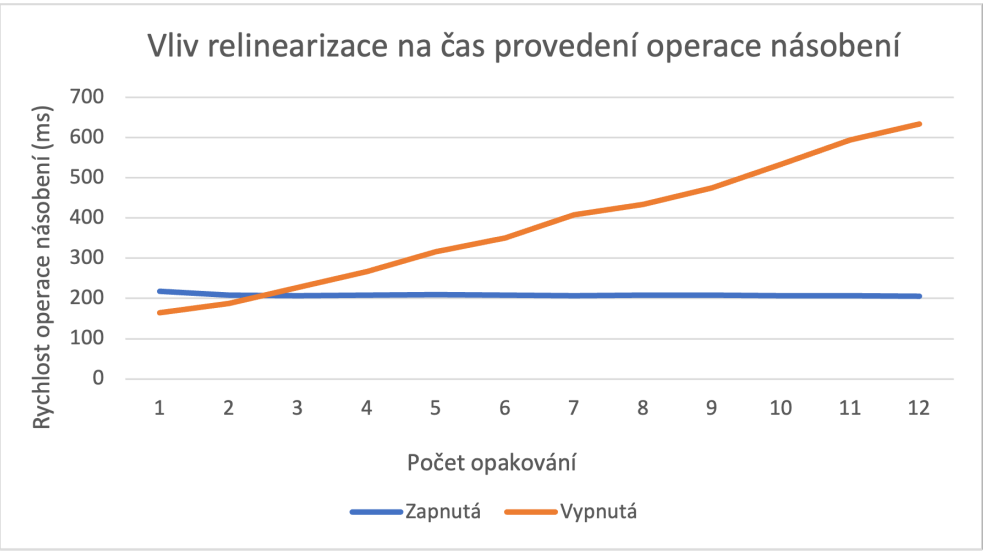
Z grafu 5.5 je vidět lineární závislost. Při dvojnásobném zvětšení plaintext modulu se vždy spotřeba noise budgetu zvýší o jedna. To ovšem není jediný důsledek. Zvětšení plaintext modulu se projeví i na celkové hodnotě noise budgetu, kdy při každém zvětšení modulu klesne i počáteční stav noise budgetu.

Pro příklad vezměme hodnotu 4096. Před provedením operace násobení byla hodnota noise budgetu 154 a po vynásobení 130. Pokud se plaintext modulus zvětší na 8192, je vidět pokles počáteční hodnoty. Před vynásobením byla hodnota 153 a po provedení operace 128.

Obecně tedy platí, že by se měla držet hodnota plaintext modulu co nejníže. Pokud víme něco o datech a jsme si jisti, že hodnota výpočtu nepřekročí nějakou mez, můžeme podle toho upravit tento parametr a docílit tak nižšího spotřebovávání noise budgetu. Na druhou stranu, pokud budeme potřebovat jen operace sčítání/odčítání, můžeme parametr nastavit na poměrně velkou hodnotu, protože tyto dvě operace spotřebovávají noise budget opravdu málo.

5.2 Relinearizace

Jak již bylo popsáno v sekci 2.2.2, relinearizace dokáže snížit velikost zašifrovaných dat po provedení operace násobení a zamezit tak zvětšování prostorové a časové náročnosti. Ve svém programu jsem spočítal jednoduchý příklad, kdy jsem dvanáctkrát za sebou vynásobil číslo 2 a zkoumal, jaký vliv má relinearizace na čas potřebný k provedení každé operace. Poly modulus byl nastavený na hodnotu 32768 a plaintext modulus na 1048576.



Obrázek 5.6 Vliv relinearizace

Z grafu 5.6 je vidět, že samotná relinearizace vyžaduje také několik milisekund k provedení. První vynásobení čísel bez zapnuté relinearizace trvalo 164 ms, zatímco se zapnutou relinearizací trvalo násobení 218 ms. Na začátku se tedy může zdát, že relinearizace spotřebovává poměrně hodně času. Ovšem zlom nastane při třetím vynásobení. Při zapnuté relinearizaci program potřebuje k vynásobení dvou čísel stále stejný konstantní čas okolo 200 ms. Bez zapnuté relinearizace ovšem potřebný čas lineárně roste a již při jedenáctém vynásobení program potřeboval třikrát delší čas.

Z toho plyne, že pokud plánujeme používat více operací násobení, měli bychom zvážit implementování relinearizace. Naopak pokud víme, že budeme potřebovat pouze operace sčítání/odčítání nebo méně než tři operace násobení, využití relinearizace nemá smysl.

Závěr

Cílem teoretické části práce bylo poskytnout přehled o současném stavu homomorfního šifrování. V této části byla nastíněna historie a také současný vývoj této šifrovací metody. Dále byla představena omezující vlastnost homomorfního šifrování, což je počet operací, které lze s tímto typem šifrování provádět. Tento problém se ovšem dá vyřešit pomocí operace zvané bootstraping. Dále byly představeny tři nejznámější homomorfní schémata. Vzhledem k matematické složitosti těchto metod bylo pouze povrchově naznačeno, jak tyto metody fungují. Byly popsány vlastnosti těchto schémat a také to, čím se od sebe liší. V závěru teoretické části byly diskutovány různé způsoby využití homomorfního šifrování, ať už ve specifických aplikacích nebo v komplexnějších oblastech, jako je například machine learning.

V praktické části práce byla naprogramována aplikace, která pro svou funkcionalitu využívá homomorfní šifrování. Konkrétně se jedná se o cloudovou kalkulačku, která dokáže provádět výpočty nad zašifrovanými daty. Pro implementaci jsem zvolil open source knihovnu MS SEAL a provedl srovnání s knihovnou MerMer od firmy PrivID. Dále jsem aplikaci podrobil několika testům, ve kterých jsem zkoumal například velikost zašifrovaných dat nebo rychlost provedení operací. Spousta zdrojů toto uvádí jako největší nedostatek homomorfního šifrování, což mohu ze svých výsledků potvrdit.

Závěrem bych chtěl říci, že homomorfní šifrování má opravdu velký potenciál. Toto šifrování by mohlo ještě zvýšit úroveň anonymity a ochrany soukromí uživatelů. Zároveň je ale třeba zmínit, že ačkoliv bylo dosaženo značného zlepšení, homomorfní šifrování stále čelí určitým výzvám. Hlavním problémem je velikost zašifrovaných dat, které jsou mnohonásobně větší než při použití klasických šifer, jako například AES. Další nevýhodou je velká náročnost na výpočetní výkon, neboť homomorfní operace jsou poměrně náročné a trvají dlouho. Masové nasazení homomorfního šifrování by proto v dnešní době bylo drahou záležitostí. Avšak s pokračujícím vývojem a investicemi do této technologie věřím, že se v blízké budoucnosti dočkáme většího využití než je tomu dnes.

Bibliografie

1. THAINE, Patricia. *Homomorphic Encryption for Beginners: A Practical Guide (Part 1)* [online]. 2018. Dostupné také z: `https://medium.com/privacy-preserving-natural-language-processing/homomorphic-encryption-for-beginners-a-practical-guide-part-1-b8f26d03a98a`. [Accessed 26-02-2024].
2. RIVEST, Ronald L.; ADLEMAN, Len; DERTOUZOS, Michael L. *On Data Banks And Privacy Homomorphisms* [online]. 1978. Dostupné také z: `https://luca-giuzzi.unibs.it/corsi/Support/papers-cryptography/RAD78.pdf`. [Accessed 26-02-2024].
3. GENTRY, Craig. *A fully homomorphic encryption scheme*. Stanford, CA, USA: Stanford University, 2009. ISBN 9781109444506. Dis. pr. AAI3382729.
4. VARA, Vasanthi. *Cybersecurity: who are the leaders in homomorphic encryption for the technology industry?* [Online]. 2023. Dostupné také z: `https://www.verdict.co.uk/innovators-cybersecurity-homomorphic-encryption-technology/?cf-view`. [Accessed 2-04-2024].
5. HINDI, Rand. *The Zama FHE Master Plan* [online]. 2024. Dostupné také z: `https://www.zama.ai/post/zama-fhe-master-plan`. [Accessed 26-03-2024].
6. PAINE, Kirsty. *Homomorphic Encryption: How It Works* [online]. 2024. Dostupné také z: `https://www.splunk.com/en_us/blog/learn/homomorphic-encryption.html`. [Accessed 27-02-2024].
7. COSTACHE, Anamaria; LAINE, Kim; PLAYER, Rachel. *Evaluating the effectiveness of heuristic worst-case noise analysis in FHE* [Cryptology ePrint Archive, Paper 2019/493]. 2019. Dostupné také z: `https://eprint.iacr.org/2019/493`.
8. BADAWI, Ahmad Al; POLYAKOV, Yuriy. *Demystifying Bootstrapping in Fully Homomorphic Encryption* [Cryptology ePrint Archive, Paper 2023/149]. 2023. Dostupné také z: `https://eprint.iacr.org/2023/149`.

9. POLYAKOV, Yuriy; BADAWI, Ahmad Al. *Bootstrapping in Fully Homomorphic Encryption (FHE)* [online]. 2023. Dostupné také z: `https://dualitytech.com/blog/bootstrapping-in-fully-homomorphic-encryption-fhe/`. [Accessed 01-03-2024].
10. GENTRY, Craig; HALEVI, Shai. *Implementing Gentry's Fully-Homomorphic Encryption Scheme* [Cryptology ePrint Archive, Paper 2010/520]. 2010. Dostupné také z: `https://eprint.iacr.org/2010/520`.
11. JOYE, Marc. *Homomorphic Encryption 101* [online]. 2021. Dostupné také z: `https://www.zama.ai/post/homomorphic-encryption-101`. [Accessed 15-03-2024].
12. MIRANTI, Andrew; VERMA, Tanya. *Encryption from Learning with Errors* [online]. 2019. Dostupné také z: `https://courses.grainger.illinois.edu/cs598dk/fa2019/Files/lecture10.pdf`. [Accessed 23-03-2024].
13. CHEN, Hao. *A Survey on Ring-LWE Cryptography* [online]. 2016. Dostupné také z: `https://www.microsoft.com/en-us/research/video/a-survey-on-ring-lwe-cryptography`. [Accessed 23-03-2024].
14. INFERATI. *Introduction to the BFV FHE Scheme* [online]. 2021. Dostupné také z: `https://inferati.azureedge.net/docs/inferati-fhe-bfv.pdf`. [Accessed 14-03-2024].
15. GEELEN, Robin; VERCAUTEREN, Frederik. *Bootstrapping for BGV and BFV Revisited* [Cryptology ePrint Archive, Paper 2022/1363]. 2022. Dostupné z DOI: `10.1007/s00145-023-09454-6`. `https://eprint.iacr.org/2022/1363`.
16. SATO, Hiroki. *A Study on Relinearize Problem in Fully Homomorphic Encryption* [online]. 2019. Dostupné také z: `https://core.ac.uk/download/pdf/286963324.pdf`. [Accessed 26-02-2024].
17. CHEN, Hao. Optimizing relinearization in circuits for homomorphic encryption. *CoRR*. 2017, roč. abs/1711.06319. Dostupné z arXiv: `1711.06319`.
18. INFERATI. *Introduction to the BGV FHE Scheme* [online]. 2022. Dostupné také z: `https://inferati.azureedge.net/docs/inferati-fhe-bgv.pdf`. [Accessed 15-03-2024].
19. INFERATI. *Introduction to the CKKS encryption scheme* [online]. 2022. Dostupné také z: `https://inferati.azureedge.net/docs/inferati-fhe-ckks.pdf`. [Accessed 15-03-2024].
20. ZAMA. *Encrypted Image Filtering Using Homomorphic Encryption* [online]. 2023. Dostupné také z: `https://www.zama.ai/post/encrypted-image-filtering-using-homomorphic-encryption`. [Accessed 24-03-2024].

21. PETKUS, Maksym. Why and How zk-SNARK Works. *CoRR*. 2019, roč. - abs/1906.07221. Dostupné z arXiv: 1906.07221.

22. AGUILAR-MELCHOR, Carlos; KILLIJIAN, Marc-Olivier; BARRIER, Joris; FOUSSE, Laurent. *XPIR: Private Information Retrieval for Everyone* [Cryptology ePrint Archive, Paper 2014/1025]. 2014. Dostupné také z: https://eprint.iacr.org/2014/1025.

23. MENON, Samir. *Private information retrieval using homomorphic encryption (explained from scratch)* [online]. 2022. Dostupné také z: https://blintzbase.com/posts/pir-and-fhe-from-scratch. [Accessed 24-03-2024].

24. MOSKVITCH, Katia. *Top Brazilian Bank Pilots Privacy Encryption Quantum Computers Can't Break* [online]. 2020. Dostupné také z: https://ibm-research.medium.com/top-brazilian-bank-pilots-privacy-encryption-quantum-computers-cant-break-92ed2695bf14. [Accessed 26-03-2024].

25. ZAMA. *Health Prediction On Encrypted Data Using Fully Homomorphic Encryption* [online]. 2023. Dostupné také z: https://huggingface.co/spaces/zama-fhe/encrypted_health_prediction. [Accessed 10-04-2024].

26. FLEMING, Seán. *What is homomorphic encryption and how can it help in elections?* [Online]. 2020. Dostupné také z: https://news.microsoft.com/on-the-issues/2020/04/13/what-is-homomorphic-encryption-and-how-can-it-help-in-elections. [Accessed 27-03-2024].

27. BADAWI, Ahmad Al; BATES, Jack; BERGAMASCHI, Flavio; GENISE, Nicholas; COUSINS, David Bruce; ERABELLI, Saroja; HALEVI, Shai; HUNT, Hamish; KIM, Andrey; LEE, Yongwoo; LIU, Zeyu; MICCIANCIO, Daniele; QUAH, Ian; POLYAKOV, Yuriy; R.V., Saraswathy; ROHLOFF, Kurt; SAYLOR, Jonathan; SUPONITSKY, Dmitriy; ZUCCA, Vincent; TRIPLETT, Matthew; VAIKUNTANATHAN, Vinod. *OpenFHE: Open-Source Fully Homomorphic Encryption Library* [Cryptology ePrint Archive, Paper 2022/915]. 2022. Dostupné také z: https://eprint.iacr.org/2022/915. https://eprint.iacr.org/2022/915.

28. *Microsoft SEAL (release 4.1)* [https://github.com/Microsoft/SEAL]. 2023. Microsoft Research, Redmond, WA.

29. LAINE, Kim. *Homomorphic Encryption with Microsoft SEAL - Microsoft Research* [online]. 2019. Dostupné také z: https://www.microsoft.com/en-us/research/video/homomorphic-encryption-with-microsoft-seal/. [Accessed 03-03-2024].

30. ZAMA. *TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data*. 2022. https://github.com/zama-ai/tfhe-rs.

31. HALEVI, Shai; SHOUP, Victor. *Design and implementation of HElib: a homomorphic encryption library* [Cryptology ePrint Archive, Paper 2020/1481]. 2020. Dostupné také z: `https://eprint.iacr.org/2020/1481`.

32. FIALA, Jiri. *Revolutionising Cloud Security with MerMer* [online]. 2023. Dostupné také z: `https://www.privid.co/privid-media/revolutionising-cloud-security-with-mermer`. [Accessed 22-03-2024].

33. PASSMARK. *Apple M1 8 Core 3200 MHz* [online]. 2024. Dostupné také z: `https://www.cpubenchmark.net/cpu.php?cpu=Apple+M1+8+Core+3200+MHz`. [Accessed 22-03-2024].

34. PASSMARK. *AMD Ryzen 5 5600X* [online]. 2024. Dostupné také z: `https://www.cpubenchmark.net/cpu.php?cpu=AMD+Ryzen+5+5600X`. [Accessed 22-03-2024].

Obsah příloh

- `readme.txt` ..... stručný popis obsahu média
- `exe` ..... adresář se spustitelnou formou implementace
- `src` ..... zdrojové kódy implementace
- `thesis`
  - `source` ..... zdrojová forma práce ve formátu LaTeX
  - `thesis.pdf` ..... text práce ve formátu PDF