Nástroj pro identifikaci kryptografických a kódovacích algoritmů

Bc. Patrik Hajšo

Diplomová práce
2024

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2023/2024

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Patrik Hajšo**
Osobní číslo:	**A22351**
Studijní program:	**N0613A140022 Informační technologie**
Specializace:	**Kybernetická bezpečnost**
Forma studia:	**Prezenční**
Téma práce:	**Nástroj pro identifikaci kryptografických a kódovacích algoritmů**
Téma práce anglicky:	**Cryptographic and Encryption Algorithm Identification Tool**

Zásady pro vypracování

1. Spracujte rešerši v oblasti kryptografických a kódovacích algoritmů.
2. Proveďte průzkum a analýzu algoritmů, které bude nástroj schopen klasifikovat.
3. Navrhněte kritéria pro identifikaci těchto algoritmů.
4. Implementujte aplikaci na základě výsledků analýzy.
5. Testujte aplikaci a prezentujte výsledky.

Forma zpracování diplomové práce: **tištěná/elektronická**
Jazyk zpracování: **Slovenština**

Seznam doporučené literatury:

1. FIPS PUB 180-4: FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION – Secure Hash Standard (SHS). *National Institute of Standards and Technology* [online]. 2015 [cit. 2023-11-09]. Dostupné z: http://dx.doi.org/10.6028/NIST.FIPS.180-4
2. JOSEFSSON, S. RFC 4648: The Base16, Base32, and Base64 Data Encodings. *Browse RFC* [online]. 2006 [cit. 2023-11--09]. Dostupné z:https://dl.acm.org/doi/pdf/10.17487/RFC4648
3. NAGY, Zsolt. *Regex Quick Syntax Reference* [online]. Berlin: Springer, 2018 [cit. 2023-11-09]. ISBN 978-1-4842-3876-9. Dostupné z: doi:https://doi.org/10.1007/978-1-4842-3876-9
4. NABABAN, Erna. Analysis Performance BCRYPT Algorithm to Improve Password Security from Brute Force. *Journal of Physics Conference Series* [online]. 2021 [cit. 2023-11-09]. Dostupné z: https://www.researchgate.net/publication/350339179_Analysis_Performance_BCRYPT_Algorithm_to_Improve_Password_Security_from_Brute_Force
5. MACHARIA, Wahome. *Cryptographic Hash Functions* [online]. 2021 [cit. 2023-11-09]. Dostupné z: https://www.researchgate.net/publication/351837904_Cryptographic_Hash_Functions
6. BIRYUKOV, Alex, Daniel DINU a Dmitry KHORVATOVICH. *Argon2: the memory-hard function for password hashing and other applications* [online]. Luxembourg: University of Luxembourg, 2015 [cit. 2023-11-09]. Dostupné z: https://www.password-hashing.net/argon2-specs.pdf

Vedoucí diplomové práce: **Ing. David Malaník, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **5. listopadu 2023**
Termín odevzdání diplomové práce: **13. května 2024**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 5. ledna 2024

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne …….

podpis studenta

ABSTRAKT

Diplomová práca sa zameriava na tvorbu aplikácie pre analýzu a identifikáciu kryptografických a kódovacích prostriedkov. Cieľom práce je vytvorenie plnohodnotného nástroja, ktorý integruje jednotlivé návrhy a kritériá do funkčného celku za účelom detekcie použitých algoritmov pri šifrovaní, kódovaní alebo iných kryptografických činnostiach v rámci počítačových systémov. Teoretická časť práce sa venuje základným teoretickým pojmom z oblasti kryptografie a kódovania, ich deleniu na jednotlivé zložky aj popisu vybraných algoritmov. Praktická časť sa zameriava na výber vhodných technológií pre tvorbu nástroja, ďalej popis architektúry a princípy fungovania jednotlivých modulov. Na záver je v tejto časti demonštrovaná funkčnosť a účinnosť nástroja.

Klúčové slová: kryptografia, kódovanie, hash, identifikácia, analýza

ABSTRACT

This master thesis focuses on the development of an application for the analysis and identification of cryptographic and encoding mechanisms. The aim of this thesis is to create a comprehensive tool that integrates individual proposals and criteria into a functional unit for the purpose of detecting algorithms used in encryption, encoding, or other cryptographic activities within computer systems. The theoretical part of the thesis addresses basic theoretical concepts in the field of cryptography and encoding, their division into individual components and the description of selected algorithms. The practical part focuses on selecting suitable technologies for the development of the tool and further describing the architecture and operating principles of individual modules. Finally, this section demonstrates the functionality and effectiveness of this tool.

Keywords: cryptography, encoding, hash, identification, analysis

Ďakujem vedúcemu mojej diplomovej práce Ing. Davidovi Malaníkovi, Ph.D. za jeho ochotu, pomoc a cenné a odborné rady popri vedení tejto práce.

Ďalej chcem poďakovať rodine a všetkým priateľom, ktorý vo mňa verili a podporovali ma na ceste k dokončeniu tejto práce.

„Per Aspera Ad Astra“

OBSAH

ÚVOD

Dáta a informácie zohrávajú neoceniteľnú úlohu v našej digitálnej spoločnosti. Pre ich zabezpečenie sa používa kryptografia, ktorá slúži ako základný pilier bezpečnosti a ochrany informácií a dát. Pre prenášanie informácií a dát sa využíva kódovanie, ktoré zabezpečuje integritu, kompatibilitu a spoľahlivý prenos. Táto práca sa venuje problematike analýzy a efektívneho rozpoznania jednotlivých postupov používaných pri zabezpečení alebo prenose dát a informácií.

Cieľom práce je vytvoriť plnohodnotnú desktopovú aplikáciu, ktorá bude aplikovať navrhnuté kritériá do funkčného celku s cieľom poskytnúť užívateľom nástroj pre efektívnu analýzu kryptografických alebo kódovacích prostriedkov.

V teoretickej časti sa práca zaoberá základnými pojmami spojenými s kryptografiou a teoretickými poznatkami, ktoré slúžia návrhom kritérií vykonávaných v praktickej časti práce. Ďalej sú rozoberané druhy kryptografie a základné znalosti a princípy hashovania a kódovania. K jednotlivým oblastiam je v tejto časti práce vykonaný prieskum vybraných šifrovacích, hashovacích a kódovacích algoritmov.

Praktická časť práce sa v úvode zameriava na návrh kritérií, ktoré zohrávajú dôležitú úlohu pri implementácii celkovej logiky aplikácie. Následne sa práca venuje výberu vhodných technológií pre tvorbu nástroja. Dôležitú časť predstavuje popis samotnej implementácie a význam jednotlivých modulov a metód. V závere je vykonané testovanie funkčnosti vytvorenej aplikácie.

I. TEORETICKÁ ČASŤ

1 ZÁKLADNÉ POJMY

V tejto kapitole sú uvedené a vysvetlené základné pojmy spojené s kryptológiou a jej rozdelením na podoblasti kryptografie a kryptoanalýzy. V rámci kryptografie sa kapitola venuje základným princípom fungovania symetrickej a asymetrickej kryptografie, ktoré predstavujú základ bezpečnostných a šifrovacích mechanizmov. Následne sú popísané teoretické poznatky, ktoré sa týkajú merania entropie súborov a regulárnym výrazom.

1.1 Kryptológia

Kryptológia je veda, ktorá skúma komunikáciu a ukladanie informácií v zabezpečenej a tajnej forme. Tento termín je odvodený z gréckych pojmov *kryptós*, čo znamená „skrytý“, a *lógos*, znamenajúci „slovo“. [1]

Zahŕňa dva pojmy - kryptografiu, ktorá sa zaoberá vytváraním šifrovacích algoritmov a metód pre ochranu informácií a dát, a kryptoanalýzu, ktorá naopak sa snaží skúmať a prelomiť šifry. [1]

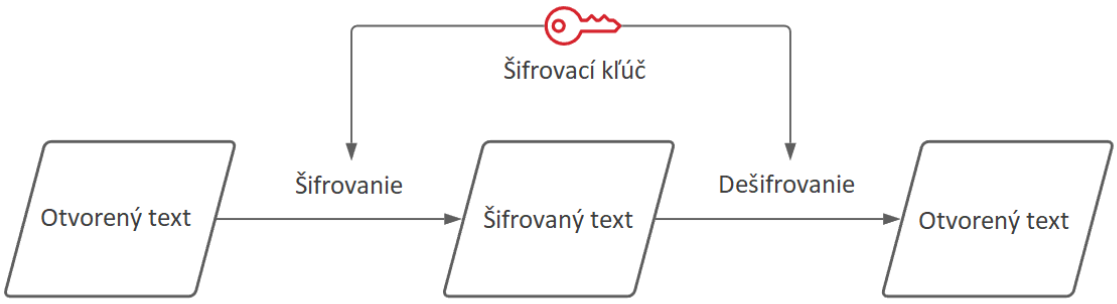
1.2 Kryptografia

Kryptografia je oblasť kryptológie, ktorá je zodpovedná za ochranu informácií a dát pred cudzím a neoprávneným prístupom. Pre dosiahnutie tohto cieľa používa šifrovanie, čo znamená proces prevodu originálnych a čitateľných údajov, odborne nazývaných aj čistý alebo otvorený text, do nečitateľnej, zašifrovanej podoby, inak nazývanej aj šifrovaný text, za pomoci určitého kryptografického algoritmu. To zaistí, že zašifrovanú správu budú môcť vidieť a čítať iba oprávnené osoby, ktoré majú potrebný kľúč na vykonanie dešifrovania. Kryptografia sa primárne delí na symetrickú a asymetrickú. [2]

1.2.1 Symetrická kryptografia

Symetrická kryptografia združuje šifrovacie algoritmy, ktoré pre proces šifrovania aj dešifrovania správy používajú jeden a ten istý kľúč. To znamená, že tento kľúč je potreba bezpečne zdieľať medzi všetkých účastníkov, ktorý majú oprávnenie správu dešifrovať. [2]

Je dôležité, aby šifrovací kľúč bol dostatočne silný a bezpečný, nakoľko bez správneho kľúča nie je možné vykonať proces dešifrovania. Na obrázku Obr. 1 je zobrazený základný princíp šifrovania a dešifrovania v symetrickej kryptografii. [2]

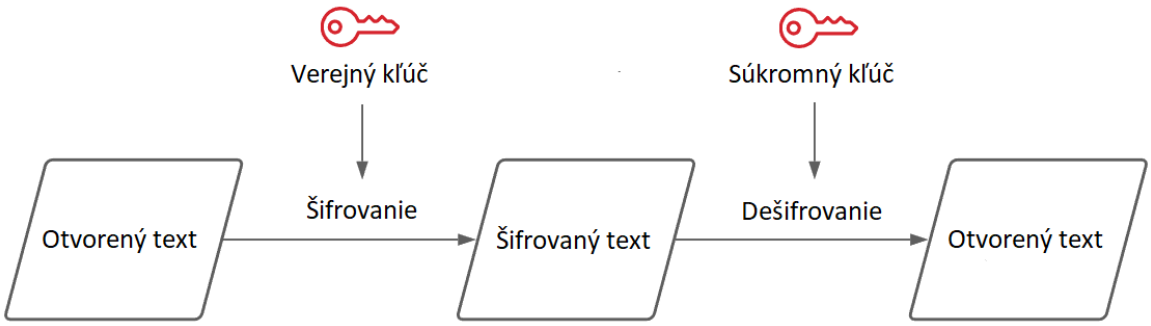


Obr. 1 Princíp fungovania symetrickej kryptografie [2]

1.2.2 Asymetrická kryptografia

Asymetrická kryptografia, na rozdiel od kryptografie symetrickej, používa kľúčový pár. Každý kľúč sa javí ako samostatný, no určitým spôsobom sú matematicky prepojené. [2]

Prvý kľúč sa označuje ako verejný kľúč a používa sa pre šifrovanie, pričom je možné ho zverejniť. Druhý kľúč sa nazýva súkromný kľúč a slúži pre dešifrovanie. Na rozdiel od verejného kľúča však musí byť bezpečne uložený a v žiadnom prípade by sa nemal zdieľať. V tomto prípade je dôležité, aby bol kľúč dostatočne silný a nemohlo dôjsť k výpočtu súkromného kľúča na základe kľúča verejného. Základný princíp šifrovania a dešifrovania v asymetrickej kryptografii je zobrazený na obrázku Obr. 2. [2]



Obr. 2 Princíp asymetrickej kryptografie [2]

1.3 Kryptoanalýza

Kryptoanalýza zahŕňa hľadanie nedostatkov a slabín v kryptografických algoritmoch s cieľom využiť ich k lúšteniu šifrovaného textu bez znalosti tajného kľúča. Tieto slabiny môžu byť v samotnom algoritme alebo často aj v jeho aplikácii. Ciele útočníka môžu byť rôzne, napríklad získanie tajného kľúča alebo získavanie informácií o otvorenom alebo šifrovanom texte. [3]

Úspech kryptoanalýzy závisí od kontextu a potrieb vykonávateľa tejto analýzy, pričom môže viesť k získaniu určitých informácií o otvorenom texte, a v prípade nedostatočne bezpečných šifrovacích postupov dokonca aj k získaniu úplných informácií. [3]

1.4 Shannonova entropia

V oblasti teórie informácií matematik Claude Shannon predstavil meranie informácie, ktorým meria rozloženie pravdepodobnosti jednotlivých symbolov a nazval to entropia. Dnes sa tejto metóde hovorí aj ako Shannonova entropia alebo Shannonovo meranie informácie. [4]

Všeobecne akceptovaný vzorec pre výpočet entropie súboru (H) sa počíta pomocou rovnice (1). [5]

$$H(X) = -\sum_{i=1}^{n} P(x_i) log_2 P(x_i) \quad (1)$$

Výsledná hodnota *H* sa uvádza v rozmedzí hodnôt od 0 do 8 a vyjadruje sa ako pomer počtu bitov na bajt, *n* predstavuje počet bajtov vo vzorku a $P(x_i)$ vyjadruje pravdepodobnosť výskytu bajtu *i* v danom vzorku. [5]

Tento princíp je možné použiť na vyjadrenie obsahu informácií v súbore a na základe toho určiť, či je súbor šifrovaný. Nešifrované a nekomprimované súbory, ako napríklad textové dokumenty, ktoré obsahujú iba bežné alfanumerické znaky, majú podstatne menšiu mieru náhodnosti a predvídateľnosti dát. Oproti tomu šifrované alebo komprimované súbory, ktorých obsah sa javí ako náhodná sekvencia znakov, vykazujú hodnoty entropie blízke k maximálnej hodnote, čo značí ich vysokú mieru náhodnosti a nepredvídateľnosti. [5]

1.5 Regulárny výraz

Regulárny výraz, často označovaný aj skrátene ako regex, je konečná postupnosť znakov reprezentujúca určitý vyhľadávací vzor. Základným princípom fungovania je hľadanie zhody medzi regulárnym výrazom a postupnosti znakov, čo napríklad často zjednodušuje a umožňuje získavanie relevantných údajov z veľkého množstva dát. Programovacie jazyky ako napríklad PHP, JavaScript, Java, C#, Python a mnoho ďalších obsahujú knižnice, pomocou ktorých je možné vykonávať rôzne operácie s regulárnymi výrazmi. [6]

Regulárny výraz sa často v praxi používa napríklad pre:

- Vyhľadávanie určitých znakov alebo ich postupností v ďalších reťazcoch.
- Hľadanie zhody postupnosti znakov a daného vzoru.
- Validáciu vstupov.
- Získavanie požadovaných informácií z rôznych súborov a dokumentov na základe daného vzoru.

  [6]

2 ALGORITMY SYMETRICKEJ KRYPTOGRAFIE

V tejto kapitole sú popísané základné informácie a princípy fungovania blokových a prúdových algoritmov, ktoré sú súčasťou symetrickej kryptografie, ktorej základné princípy sú popísané v podkapitole 1.2.1 „Symetrická kryptografia“. Ďalej sú v tejto časti popísaný zástupcovia z radov blokových symetrických algoritmov.

2.1 Prúdové symetrické algoritmy

V prúdových symetrických algoritmoch sa každý bit otvoreného textu šifruje samostatne a to kombináciou daného bitu s bitom generovaného šifrovacieho kľúča. [7]

Medzi predstaviteľov prúdových šifier patria napríklad RC4, Salsa20 alebo Grain-128. [8]

Pri šifrovaní a dešifrovaní pomocou prúdovej šifry pozostávajú otvorený text (x) , šifrovaný text (y) a kľúč (s) z jednotlivých bitov, teda $x_i, y_i, s_i \in \{0,1\}$. Proces šifrovania sa dá vyjadriť pomocou rovnice (2) a dešifrovanie pomocou rovnice (3) s použitím operácie modulo (mod). [7]

$$y_i = \mathrm{es}_i(x_i) \equiv x_i + s_i \bmod 2 \quad (2)$$

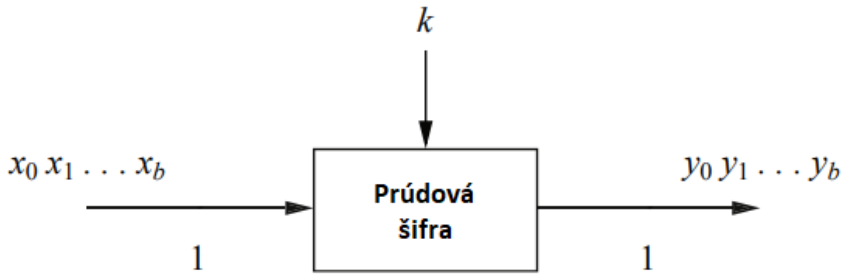
$$x_i = \mathrm{ds}_i(y_i) \equiv y_i + s_i \bmod 2 \quad (3)$$

Operácia sčítania binárnych čísel modulo 2 je ekvivalentná s logickou operáciou XOR, preto sa šifrovanie a dešifrovanie môže vyjadriť aj pomocou rovníc (4) a (5). [7]

$$y_i = \mathrm{es}_i(x_i) \equiv x_i \oplus s_i \quad (4)$$

$$x_i = \mathrm{ds}_i(y_i) \equiv y_i \oplus s_i \quad (5)$$

Na obrázku Obr. 3 je zobrazená jednoduchá schéma fungovania prúdovej symetrickej šifry, kde jednotlivé bity otvoreného textu $x_0, x_1 \ldots x_b$ sú samostatne šifrované pomocou kľúča k a výstupom sú bity šifrovaného textu $y_0, y_1 \ldots y_b$. [7]



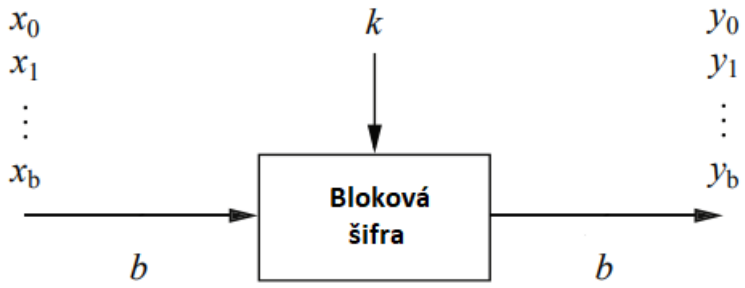
Obr. 3 Schéma fungovania prúdovej šifry [7]

2.2 Blokové symetrické algoritmy

Blokové algoritmy sú algoritmy symetrickej kryptografie, ktoré šifrujú skupiny bitov otvoreného textu, nazývaných bloky, za pomoci jedného šifrovacieho kľúča. Každý bit v bloku je ovplyvnený ostatnými bitmi daného bloku, to znamená že zmena jediného bitu môže vo výsledku spôsobiť výrazné zmeny v zašifrovanom bloku. Dĺžka bloku je vo väčšine prípadov blokových algoritmov nastavená na veľkosť 128 bitov, čo odpovedá 16 bajtom. [7]

Medzi predstaviteľov blokových symetrický šifier patria napríklad DES, 3DES alebo AES, a mnoho ďalších. [8]

Na obrázku Obr. 4 je zobrazená schéma fungovania blokovej symetrickej šifry, kde bloky otvoreného textu $x_0, x_1 \ldots x_b$ o veľkosti b bitov sú šifrované pomocou kľúča k a výstupom sú bloky šifrovaného textu $y_0, y_1 \ldots y_b$ taktiež o veľkosti b bitov. [7]



Obr. 4 Schéma fungovania blokovej šifry [7]

2.2.1 Advanced Encryption Standard (AES)

V roku 1977 vyhlásil Národný úrad pre štandardy a technológie v USA (NIST) verejnú iniciatívu s cieľom vytvoriť nový štandard šifrovania, ktorý bude niesť označenie AES. V roku 2000 bol za víťaza predstavený algoritmus Rijndael, ktorý sa následne stal štandardom s označením AES. Jedná sa o symetrickú blokovú šifru, ktorá môže používať kľúče o veľkosti 128, 192 a 256 bitov a pracuje s dĺžkou bloku o veľkosti 128 bitov. [9]

Algoritmus AES má niekoľko silných stránok:

- Je všeobecne možné implementovať AES v ktoromkoľvek softvérovom aj hardvérovom riešení, pričom však pri jeho realizácii záleží aj na istých špecifických faktoroch, ako napríklad používanej technológii, prostrediu alebo samotnej štruktúry aplikácie, aby bolo možné dosiahnuť požadovanú bezpečnosť, kompatibilitu a dostatočný výkon.
- AES je odporúčaný a dostupný nielen pre vládne agentúry a ochranu tajných údajov, ale aj nevládnym subjektom a organizáciám pre komerčné aj súkromné účely.

  [9]

AES je často využívaný v rôznych technológiách a pre mnohé účely:

- Šifrovanie v cloudových aplikáciách.
- Šifrovanie v mobilných aplikáciách.
- Šifrovanie súborov.
- Šifrovanie v protokoloch Wi-Fi (WPA-PSK, WPA2-PSK).
- Šifrovanie v sieťových protokoloch (FTPS, HTTPS...).

  [10]

2.2.2 International Data Encryption Algorithm (IDEA)

IDEA je symetrická bloková šifra, ktorá bola publikovaná v roku 1991 na univerzite v Zurichu a mala nahradiť už starnúcu šifru DES. Algoritmus bol neskôr začlenený do šifrovacieho systému Pretty Good Privacy (PGP), ktorý kombinuje používanie symetrických a asymetrických algoritmov. IDEA pracuje s blokmi o dĺžke 64 bitov a používa šifrovací kľúč o veľkosti 128 bitov. [11]

Algoritmus IDEA ponúka niekoľko výhod:

- Je možná efektívna implementácia algoritmu v rámci softvérových aj hardvérových riešení.
- Algoritmus pracuje rýchlo a nepotrebuje veľké množstvo pamäte alebo výkonu.
- Algoritmus je stále považovaný za bezpečný.

[11]

IDEA stále nachádza uplatnenia napríklad v nasledujúcich odvetviach:

- Šifrovanie sieťovej komunikácie.
- Finančné transakcie a bankovníctvo.
- Šifrovanie súborov.
- Šifrovací algoritmus v mechanizme PGP.

[11]

3 ALGORITMY ASYMETRICKEJ KRYPTOGRAFIE

V tejto kapitole sú popísané asymetrické šifrovacie algoritmy. Princíp asymetrickej kryptografie je popísaný v podkapitole 1.2.2 „Asymetrická kryptografia“, preto je v tejto kapitole dôraz kladený hlavne na popis základného využitia týchto algoritmov a ich rozdelenia podľa typu matematického problému, na ktorom sa zakladajú. Kapitola sa následne venuje aj bližšej špecifikácií konkrétnych predstaviteľov asymetrickej kryptografie.

3.1 Využitie a rozdelenie asymetrických algoritmov

Asymetrické algoritmy vďaka svojím charakteristickým znakom plnia prevažne nasledujúce funkcie:

- **Distribúcia kľúčov –** výmena a zdieľanie tajných kľúčov cez nezabezpečené kanály, napríklad pomocou algoritmu Diffie-Hellman.
- **Dátová integrita –** overovanie a zabezpečovanie integrity správ a dokumentov, napríklad pomocou algoritmu digitálneho podpisu (DSA).
- **Šifrovanie –** algoritmy asymetrickej kryptografie je možno použiť aj pre šifrovanie dát, napríklad pomocu RSA alebo Elgamal. Pre šifrovanie sa však tieto algoritmy často neuplatňujú kvôli ich nízkej efektivite a rýchlosti šifrovania v porovnaní s algoritmami symetrickej kryptografie.

  [7]

Tieto algoritmy sa dajú rozdeliť do skupín, nielen podľa ich možného využitia, ale aj podľa typu matematických problémov, na ktorých sú založené:

- **Faktorizácia –** tento problém je založený na zložitosti rozkladu veľkých čísel na súčin dvoch menších čísiel. Najznámejším predstaviteľom využívania tohoto typu problému je napríklad algoritmus RSA.
- **Diskrétny logaritmus** – problém je založený na zložitosti riešenia diskrétneho logaritmu. Do tejto skupiny spadajú algoritmy ako Diffie-Hellman, Elgamal alebo algoritmus digitálneho podpisu (DSA).

- **Eliptické krivky –** vychádzajú z problému diskrétneho logaritmu, ktorý je v tomto prípade adaptovaný na štruktúru eliptických kriviek. Významnými predstaviteľmi kryptografie eliptických kriviek sú algoritmy Diffie-Hellman založený na eliptických krivkách (ECDH) alebo algoritmus digitálneho podpisu na báze eliptických kriviek (ECDSA).

  [7]

3.1.1 RSA

RSA je v súčasnosti jedným z často používaných asymetrických algoritmov. Názov algoritmu pochádza z kombinácie iniciálok tvorcov tohoto algoritmu - Rivest-Shamir-Adleman, ktorý ho vytvorili v roku 1977. Algoritmus RSA bol patentovaný iba pre použitie v USA do roku 2000, následne sa sprístupnil pre neobmedzené používanie aj zvyšku sveta. [7]

Vzhľadom na veľkú výpočetnú náročnosť, ktorá je daná potrebou vykonania veľkého množstva operácií procesorovou jednotkou (CPU), je pomerne náročné a zdĺhavé používať algoritmus RSA pre šifrovanie väčšieho množstva dát. [7]

Vzhľadom k uvedeným skutočnostiam sa preto v praxi využíva hlavne pre tieto účely:

- Šifrovanie malého objemu dát – jedná sa hlavne o šifrovanie symetrických šifrovacích kľúčov, ktoré následne môžu byť bezpečne transportované, preto sa RSA často používa v kombinácií so symetrickou šifrou, ktorá sa použije na zašifrovanie veľkých objemov dát.
- Digitálne podpisy a digitálne certifikáty.

  [7]

3.1.2 Kryptografia eliptických kriviek (ECC)

Kryptografia eliptických kriviek, ďalej len ECC, je súčasťou asymetrickej kryptografie. Bola vytvorená v roku 1985 a jej tvorcami sú Victor Miller a Neil Koblitz, ktorých cieľom bolo vytvoriť iný spôsob systému implementácie asymetrickej kryptografie. [12]

Tieto algoritmy využívajú matematické operácie vykonávané na eliptických krivkách, z čoho následne vyplynie kľúčový pár používaný pri šifrovaní a dešifrovaní. [13]

Vzhľadom k tomu, že tento kryptografický systém bol vytvorený ako alternatíva k už existujúcim schémam asymetrickej kryptografie, ako napríklad RSA, zástupcovia ECC algoritmov disponujú niekoľkými výhodami:

- Používajú výrazne kratšie kľúče, napríklad o veľkosti 256 bitov. Algoritmy využívajúce eliptické krivky vďaka tomu pracujú efektívnejšie a rýchlejšie a takýto kľúč je rovnako menej náročný na spravovanie a ukladanie.
- Algoritmy z radov ECC poskytujú minimálne rovnakú alebo lepšiu bezpečnosť ako RSA. Napríklad algoritmus z radov eliptických kriviek, využívajúci kľúč o veľkosti 256 bitov, je približne rovnako silný ako algoritmus RSA, využívajúci kľúč o veľkosti 3072 bitov.

  [13]

Ako bolo spomenuté v kapitole 3.1 „Využitie a rozdelenie asymetrických algoritmov“, sú algoritmy ECC používané prevažne pri digitálnom podpisovaní, kde sa využíva *Elliptic-Curve Digital Signature Algorithm (ECDSA)* , alebo pri distribúcií tajných kľúčov, kde sa môže využiť algoritmus *Elliptic-Curve Diffie-Hellman (ECDH).*

4 HASHOVACIE ALGORITMY

Táto kapitola sa venuje hashovaniu a hashovacím algoritmom. Na začiatku je vysvetlený princíp vytvárania hashu a popísané požiadavky na hashovacie algoritmy, ktoré musia byť splnené pre zaistenie bezpečnosti a integrity dát. V kapitole sú popísané jedny z najznámejších a najpoužívanejších hashovacích algoritmov, ako napríklad rodiny SHA a MD, ktoré sú spomínané na webe SignMyCode [14], ktorý poskytuje certifikáty pre overovanie totožnosti vydavateľov softvéru. Spomenutý sú aj významný predstavitelia hashovacích algoritmov, ako Bcrypt, Scrypt a Argon2, odporúčaných komunitou OWASP [15] pre bezpečné ukladanie hesiel.

4.1 Hashovanie

Hashovanie je proces, pri ktorom sa vstupná správa transformuje na číselnú hodnotu s pevnou dĺžkou, to znamená že akokoľvek bude správa obsiahla, výstup bude vždy rovnako dlhý. Hodnota hashu je vytvorená pomocou operácie (6). [16]

$$h = H(m) \qquad (6)$$

V tomto prípade je m správa variabilnej dĺžky, H je hashovacia funkcia a h je hodnota hashu, teda výstup z hashovacej funkcie. Hodnota hashu sa väčšinou uvádza v hexadecimálnom formáte alebo v kódovanej forme za použitia algoritmu Base64. [16]

4.1.1 Hashovacie algoritmy a ich požiadavky

Hashovací algoritmus je kryptografická funkcia, ktorá zo vstupnej správy vytvára hodnotu s pevnou dĺžkou. Sú v nich používané matematické funkcie, ktoré sú bežne zakorenené v kryptografii. [16]

Tieto algoritmy by sa navonok mali chovať náhodne, no zároveň by mali byť deterministické a efektívne vypočítateľné. Algoritmy produkujúce hash sú jedným zo základných pilierov v oblasti kryptografie a používajú sa k vykonávaniu rôznych úkonov:

- Ukladanie hesiel do databáze.
- Digitálne podpisy.
- Unikátne identifikátory.
- Zabezpečenie dátovej integrity.
- Blockchain. [17]

Kvôli ich veľkému využitiu v oblasti bezpečnosti, musia hashovacie algoritmy **H** spĺňať určité požiadavky :

- **H** môže byť použitý pre informáciu ľubovoľnej veľkosti.
- **H** vytvára výstup s pevnou dĺžkou, ktorá závisí na základe použitého algoritmu.
- Výpočet **H(x)** je pomerne jednoduché vypočítať pre akékoľvek **x.**
- Pre akúkoľvek hodnotu **h** je nemožné nájsť vstup **x**, pre ktoré by platilo **H(x) = h.** Musí sa teda jednať o jednosmernú funkciu.
- Je výpočetne veľmi náročné, až nemožné, nájsť **x ≠ x'**, aby platilo **H(x) = H(x').** Jedná sa o takzvanú slabú odolnosť voči kolízií.
- Je výpočetne veľmi náročné, až nemožné, nájsť akýkoľvek pár (**x , x'**), aby platilo **H(x) = H(x').** Jedná sa o takzvanú silnú odolnosť voči kolízii.

  [16]

4.2 Hashovacie algoritmy z rodiny SHA-1 a SHA-2

Jedny z najznámejších hashovacích algoritmov pochádzajú z rodín Secure Hash Algorithm (SHA) 1 a 2. Boli vytvorené Národným inštitútom pre štandardy a technológie (NIST) so sídlom v USA. Jedná sa o konkrétne päť hashovacích funkcií – SHA-1, SHA-224, SHA-256, SHA-384 a SHA-512. [18]

Všetky zmienené funkcie dostávajú vstup rôznej dĺžky a produkujú pevne daný výstup, v závislosti na tom, o aký algoritmus sa jedná. V prípade SHA-1 je výstupná hodnota hashu o veľkosti 160 bitov, SHA-224 produkuje 224 bitový výstup, SHA-256 256 bitový výstup, SHA-384 produkuje hash o veľkosti 384 bitov a SHA-512 o veľkosti 512 bitov. [18]

Hashovacie algoritmy týchto rodín majú uplatnenie napríklad pri overovaní digitálnych podpisov, overovanie integrity správ a súborov alebo pri generovaní náhodných bitových postupností alebo čísel. [18]

Algoritmy z rodiny SHA-1 a SHA-2 generujú hodnoty hashu, ktoré sa uvádzajú v hexadecimálnom formáte:

- SHA-1: **2d7ff18c29ffa71ea839d401007f0656c34307ff**
- SHA-224: **960ee303e8a8b9ff54d502a9e1a3520a71cc0e926348048975c7f6c0**
- SHA-256: **886700af9b7c6a728439dc797d29ed3c6f71d860deaafdce1bb53cd54abc11ea**
- SHA-384: **b69b117004336dbeb89dbe81fa053b7650cf664b92bb7bf52508245c99045e2a9b3877e8ac6d4a86aaa36859e38f5463**
- SHA-512: **c48d83275010578dd3f649a1aae51c277d6c09a2c6bca59620f7f2f8e705bf88cdf9cbf614b09dac26024f5cbe08d62d474322efe03a746526c5e88ed1a91e15**

4.3 Hashovacie algoritmy z rodiny SHA-3

V roku 2015 vydal NIST publikáciu o nových hashovacích algoritmoch z rodiny SHA-3. Konkrétne sa jedná o štyri hashovacie algoritmy – SHA3-224, SHA3-256, SHA3-384 a SHA3-512 a dva algoritmy s rozšíriteľným výstupom – SHAKE128 a SHAKE256. [19]

Tieto algoritmy dopĺňajú rodiny SHA-1 a SHA-2, no líšia sa lepšou bezpečnosťou a hlavne konštrukciou s princípom hashovania založenom na permutáciách. Rodina SHA-3 obsahuje dva algoritmy SHAKE128 a SHAKE256, ktoré umožňujú produkovať hashe na základe požadovanej, nastaviteľnej dĺžky. [19]

Rovnako ako v prípade SHA-1 a SHA-2, sa algoritmy z rodiny SHA-3 uplatňujú v rôznych sférach, ako napríklad digitálne podpisy, generátory pseudonáhodných čísel a odvodenie kľúčov. [19]

Algoritmy z rodiny SHA-3 generujú hash, ktorý sa zapisuje v hexadecimálnom formáte:

- SHA3-224: **e2aecb1e0571092946e3f7893245a3fcbcbef615b9e7ea0e1842f832**
- SHA3-256: **86eb085fe86c8009762483ce7f59741da4cfbdaa3c0453672a6c60704c037d5e**

- SHA3-384:
  19a865e7099a1aa34836fb26a44c4743d3c4a6aca0d0df3e073d131944a131396bb667a56fd82ca3836b121d12449287
- SHA3-512:
  891107a79da6f9dc4119fc7907da5700eaa1f8ac0a74cd3cda31e3d8624ba7682582e3631f24b830f68511908e496d9b32e14125aecd5b7c7022df24a1087587
- SHAKE-128 - 256 bitový výstup :
  1f6dca37b9ec4423eda4576db05dc0e30c3a0cf025173dc4916e8c99eb4089dd
- SHAKE-256 – 384 bitový výstup :
  0c5c223b43b57df1b871c0263ff88b439baf8bcc67b1799283929673c14592b2cf48b96f1ed669ffbe3926532eb4aecb

4.4 Hashovacie algoritmy z rodiny MD

Hashovacie algoritmy z rodiny MD patria medzi ďalšie populárne hashovacie funkcie. Konkrétne sa jedná o algoritmy MD2, MD4, MD5 a MD6. V dnešnej dobe sa úplne neodporúčajú používať, nakoľko boli v algoritme MD5 už nájdené kolízie. Všetky spomenuté algoritmy z rodiny MD produkujú hodnotu hashu o veľkosti 128 bitov okrem najnovšieho algoritmu MD6, ktorý dokáže produkovať hash s dĺžkou 128, 256 alebo 512 bitov. [20]

Najznámejšou a najpoužívanejším algoritmom z rodiny MD je algoritmus MD5, ktorý je aj napriek spomínaným neodporúčaniam v súčasnoti stále využívaný k zaisteniu dátovej integrity súborov a dostupný v súborových serveroch. [20]

Algoritmy z rodiny MD generujú hash, ktorý sa zapisuje v hexadecimálnom formáte:

- MD2: **f0ac99970341bd1619da69b7b2d02c43**
- MD4: **df5308c6585aad1577089d4eb4800feb**
- MD5: **b7cf767f4462b7c2ec0e0be28d03f08e**
- MD6-128: **8002f6210dd26d948c0a575a29a4d288**
- MD6-256:
  c8e7e5b1f6c8d01cf277a7bc1449f5af33b866d4dedb83d66c05e911fe69d6b3

- MD6-512 :

  306274078bc92c030a92a88a607e515e0d47d5ba79cecc8bf4938df3b3e98cec757b79c7603c8d2517b2815f4e2934cba6499160be482bebafd61b232e359345

4.5 Bcrypt

Bcrypt je kryptografický hashovací algoritmus, ktorý bol primárne zostrojený pre bezpečné ukladanie hesiel a používa sa často v aplikáciách v oblasti backendu. Tento algoritmus bol vytvorený v roku 1999, pričom ako základ pre jeho konštrukciu slúži šifrovací algoritmus Blowfish. [21]

Na rozdiel od iných hashovacích algoritmov, Bcrypt pridáva k odtlačku aj náhodnú časť dát, ktorá sa v odbornej literatúre nazýva soľ (salt). Vďaka tomu zaručuje, že rovnaký vstupný text bude mať vždy odlišný hash, čo zvyšuje odolnosť voči útoku hrubou silou. [21]

Bcrypt takisto vyniká používaním takzvaného nákladového faktoru (cost factor), pomocou ktorého je možné určiť počet iterácií a kôl hashovania, vďaka čomu sa zvyšuje potrebný čas a výkon, ktorý je potrebný na výpočet hashu. [21]

Hash produkovaný algoritmom bcrypt, môže vyzerať nasledovne:

$2a$10$X/hqsTcxZm68xxOjVFm7nu MGrCNj8Rv6y3MCKdgyhDzI070b1seDy

Tento hash používa niekoľko charakteristických znakov, pomocou ktorých je možné ho jednoznačne identifikovať:

- **$2a$** udáva prefix algoritmu, ktorý jednoznačne identifikuje algoritmus Bcrypt. Ďalšími možnými prefixami sú **$2b$**, **$2y$** a **$2x$.**
- **10** udáva nákladový faktor, ktorý predstavuje číselnú reprezentáciu počtu iterácií algoritmu.
- Následný reťazec predstavuje hash, ktorý pozostáva zo soli **X/hqsTcxZm68xxOjVFm7nu** o veľkosti 22 znakov a hashu správy **MGrCNj8Rv6y3MCKdgyhDzI070b1seDy** o veľkosti 31 znakov, pričom obe časti sú uvádzané vo forme kódovania base64.

  [21]

4.6 Argon2

Argon2 je pamäťovo náročný, viacúčelový algoritmus, ktorý primárne slúži pre ukladanie hesiel, alebo ako algoritmus vhodný pre odvodenie kľúča. [22]

Existuje niekoľko typov tohto algoritmu , pričom jednotlivé typy algoritmu sa používajú na iné účely:

- **Argon2d** je jedným z dvoch variantov algoritmu Argon2. Je rýchlejší a primárne určený pre kryptomeny a aplikácie, u ktorých nehrozia útoky postrannými kanálmi.
- **Argon2i** je variant algoritmu, ktorý bol vytvorený pre zvýšenie odolnosti voči útokom typu tradeoff. Je pomalší ako predošlý typ a vhodný pre hashovanie hesiel alebo ako funkcia pre odvodenie kľúča.

  [22]

Argon 2 používa dva typy vstupov – primárne a sekundárne. Primárne vstupy tvorí vstupná správa *P* a soľ *S*. Vedľajšie vstupy majú nasledujúce parametre:

- Stupeň paralelizmu *p* určuje počet bežiacich výpočtových vlákien.
- Veľkosť pamäte *m* udáva hodnotu potrebnej pamäte pre vykonanie hashovania.
- Počet iterácií *t* udáva počet kôl hashovania. [22]

Hash produkovaný algoritmom Argon2 môže vyzerať nasledovne:

$argon2i$v=19$m=12,t=10,p=4$YXk4MGJjNHFkbTYwMDAwMA$
fYSTPOIWCGtzJon0ai4UxQ

Tento hash má niekoľko znakov, pomocou ktorých je možné ho jednoznačne identifikovať:

- **$argon2i$** značí použitý typ algoritmu.
- **$v=19$** predstavuje verziu algoritmu.
- **$m=12, t=10, p=4$** udáva pamäťovú náročnosť, počet iterácií a stupeň paralelizmu.
- **$YXk4MGJjNHFkbTYwMDAwMA$** udáva soľ v kódovaní base64.
- **fYSTPOIWCGtzJon0ai4UxQ** je výsledný hash v kódovaní base64.

  [22]

4.7 Scrypt

Scrypt je pamäťovo náročný hashovací algoritmus. Primárne je používaný v kryptomenách a v rôznych aplikáciách, ktoré vyžadujú bezpečné ukladanie hesiel vo forme hashu. [23]

Algoritmus je navrhnutý pre odolnosť útokov vysoko účinných hardvérov pre lámanie hesiel, ako napríklad výkonný grafický procesor (GPU), alebo technológie ASIC a FPGA. [23]

Algoritmus má niekoľko užívateľsky nastaviteľných vstupných parametrov:

- N – počet iterácií - predvolený počet je 16384.
- r – dĺžka bloku - predvolená dĺžka je 8 bajtov.
- p – stupeň paralelizmu – počet bežiacich vlákien - predvolený stupeň je 1.
- *správa* – vstupný reťazec znakov.
- *soľ* – náhodne generovaná hodnota - odporúčaná hodnota je minimálne 8 bajtov.
- *dĺžka výstupu* – odporúčaná dĺžka je 32 bajtov. [23]

Na základe zvolených parametrov sa dokáže vypočítať pamäť potrebná k vypočítaniu hashu pomocou vzorca (7). [23]

$$Pamäťová\ náročnosť = 128 * N * r * p \qquad (7)$$

Pre ukážku, zvolením parametrov $N = 16384$, $r = 8$ a $p = 1$, dostaneme výsledok 16 777 216 B, čo sa rovná 16 MB pamäte potrebnej pre výpočet hashu. [23]

Hash tvorený pomocou algoritmu Scrypt sa vo väčšine prípadov ukladá aj s danými parametrami, ktoré sú oddelené znakom „$“, ako aj v prípade iných algoritmov. Scrypt, narozdiel od algoritmu Bcrypt alebo Argon2, nemá jednoznačne predpísanú podobu, v ktorej sa má hash ukladať, preto jeden z možných vzorov hashu [23] produkovaného algoritmom Scrypt môže vyzerať nasledovne:

16384$8$1$kytG1MHY1KU=$afc338d494dc89be40e317788e3cd9166d066709db0e648 1f0801bd918710f46

Takýto hash má niekoľko znakov, pomocou ktorých je možné ho jednoznačne identifikovať:

- **16384** značí počet iterácií (N).
- **8** predstavuje dĺžku bloku (r) , s ktorým sa pracuje.
- **1** udáva stupeň paralelizmu (p) .
- **$kytG1MHY1KU=$** predstavuje soľ v kódovaní base64
- **afc338d494dc89be40e317788e3cd9166d066709db0e6481f0801bd918710f46** predstavuje hash v hexadecimálnom formáte.

Ďalším z možných vzorov pre algoritmus scrypt, ktorý je uvedený v dokumentácií nástroja pre lámanie hesiel s názvom hashcat [24], má nasledovnú podobu:

SCRYPT:1024:1:1:MDIwMzMwNTQwNDQyNQ==:5FW+zWivLxgCWj7qLiQbeC8zaNQ+qdO0NUinvqyFcfo=

Od predošlej podoby sa tento vzor líši v určitých aspektoch, ako napríklad uvedenie reťazca „SCRYPT“ pred samotný hash, používanie znaku „ : “ ako oddelovač jednotlivých parametrov alebo uvádzanie samotného hashu v kódovaní Base64 namiesto hexadecimálneho zápisu. Na základe týchto charakteristických znakov, je rovnako možné identifikovať aj takúto podobu hashu.

5 KÓDOVACIE ALGORITMY

Kapitola sa venuje kódovaniu a kódovacím algoritmom patriacim do rodiny Base. Sú tu uvedené základné informácie o známych zástupcov tohto typu, ako ich súbory znakov používaných v rámci kódovania a poskytnuté praktické príklady použitia týchto kódovacích algoritmov v oblasti informatiky a počítačových systémov.

5.1 Kódovanie

V rôznych formách komunikácie, vrátane výpočetnej techniky a prenosu dát, je často využívaný proces, ktorý sa nazýva kódovanie. Jedná sa o zmenenie pôvodnej formy dát do iného formátu, aby sa lepšie prenášali alebo ukladali. [25]

Pri kódovaní v počítačových systémoch dochádza k transformácii reťazcov znakov, ako sú písmená, čísla, interpunkčné znamienka a špeciálne symboly do kompaktnejšieho formátu vhodného na efektívnejší prenos alebo ukladanie. Dekódovanie je proces, ktorý zakódovaný reťazec prevádza späť do pôvodnej formy. [25]

Kódovanie by sa za žiadnu cenu nemalo zamieňať so šifrovaním, ktoré má za cieľ ochranu a utajenie dát. Je však možné spojiť šifrovanie s kódovaním dát, čo môže mať za následok zlepšenie kompatibility alebo zachovanie integrity. [25]

5.2 Kódovacie algoritmy typu Base

Kódovanie dát pomocou algoritmov Base sa používa pre ukladanie alebo prenos dát hlavne v prostrediach, ktoré sú obmedzené na štandard ASCII. Tieto algoritmy používajú pre kódovanie dát súbory znakov, ktoré sa označujú ako abeceda. [26]

5.2.1 Base64

Algoritmus Base64 je kódovací algoritmus, ktorý zahŕňa kódovanie dát pomocou malých a veľkých písmen, čísel a niekoľkých špeciálnych znakov. Používa abecedu o veľkosti 64 znakov, ktorá je určitou podmnožinou znakov používaných v ASCII. Abeceda, ktorá je v tomto algoritme využívaná, je zobrazená na obrázku Obr. 5. Dodatočným 65. znakom abecedy je znak „ = “, ktorý sa za určitých okolností používa ako takzvaná špeciálna operácia, napríklad vo forme dodatočného vyplnenia (padding), aby bola zabezpečená požadovaná dĺžka a bolo možné korektne dekódovať dáta. Na obrázku Obr. 6 je zobrazené kódovanie testovacieho reťazca. [26]

0 A	17 R	34 i	51 z
1 B	18 S	35 j	52 0
2 C	19 T	36 k	53 1
3 D	20 U	37 l	54 2
4 E	21 V	38 m	55 3
5 F	22 W	39 n	56 4
6 G	23 X	40 o	57 5
7 H	24 Y	41 p	58 6
8 I	25 Z	42 q	59 7
9 J	26 a	43 r	60 8
10 K	27 b	44 s	61 9
11 L	28 c	45 t	62 +
12 M	29 d	46 u	63 /
13 N	30 e	47 v	
14 O	31 f	48 w	(pad) =
15 P	32 g	49 x	
16 Q	33 h	50 y	

Obr. 5 Súbor znakov používaný v Base64 [26]

```
BASE64("") = ""

BASE64("f") = "Zg=="

BASE64("fo") = "Zm8="

BASE64("foo") = "Zm9v"

BASE64("foob") = "Zm9vYg=="

BASE64("fooba") = "Zm9vYmE="

BASE64("foobar") = "Zm9vYmFy"
```

Obr. 6 Testovanie kódovania znakov pomocou algoritmu Base64 [26]

Algoritmus Base64 sa používa napríklad pre nasledujúce účely:

- Prenos binárnych dát pomocou ich prevodu do textovej podoby, aby dokázali byť prenesené aj cez systémy a vrstvy, ktoré sú obmedzené na pracovanie s dátami iba v textovom formáte.
- Kódovanie a dekódovanie dát pre ukladanie informácií, ktoré by boli inak zložité pre uchovávanie v určitých formátoch, napríklad ukladanie dát do cookies alebo URL.
- Prenos šifrovaných dát, aby nedošlo k chybnému prenosu.

[27]

5.2.2 Base32

Base32 je kódovací algoritmus, ktorý ma rovnaké určenie ako Base64, ale jeho abeceda je redukovaná na 32 znakov ASCII a posledný 33. znak „=“ reprezentuje špeciálne operácie, rovnako ako v prípade Base64. Na obrázku Obr. 7 je zobrazená abeceda, ktorá je používaná pri kódovaní Base32 a na obrázku Obr. 8 je zobrazené testovanie kódovania reťazcov týmto algoritmom. [26]

0 A	9 J	18 S	27 3
1 B	10 K	19 T	28 4
2 C	11 L	20 U	29 5
3 D	12 M	21 V	30 6
4 E	13 N	22 W	31 7
5 F	14 O	23 X	
6 G	15 P	24 Y	(pad) =
7 H	16 Q	25 Z	
8 I	17 R	26 2	

Obr. 7 Súbor znakov používaný v Base32 [26]

```
BASE32("f") = "MY======"

BASE32("fo") = "MZXQ===="

BASE32("foo") = "MZXW6==="

BASE32("foob") = "MZXW6YQ="

BASE32("fooba") = "MZXW6YTB"

BASE32("foobar") = "MZXW6YTBOI======"
```

Obr. 8 Testovanie kódovania znakov pomocou algoritmu Base32 [26]

Algoritmus Base32 sa používa napríklad na nasledujúce účely:

- Kódovanie tajných kľúčov autentifikačných systémov typu Time-Based One-Time Password (TOTP).
- Používanie v rámci DNS záznamov pre kódovanie údajov a dát.
- Používanie v rôznych systémoch, ako napríklad InterPlanetary File System (IPFS), v rámci vhodnej formy dát pre URL.

[28]

5.2.3 Base16

Base16 je kódovací algoritmus, ktorý je známy aj ako hexadecimálne kódovanie. Používa redukovanú abecedu, ktorá obsahuje 16 znakov. Od algoritmov Base64 a Base32 sa líši tým, že neobsahuje znak " = " pre špeciálne operácie, nakoľko je pri tomto kódovaní vždy dostupné celé kódované slovo. Na obrázku Obr. 9 je zobrazená abeceda, ktorá je používaná pri kódovaní a na obrázku Obr. 10 je zobrazené kódovanie testovacích reťazcov. [26]

0 0	4 4	8 8	12 C
1 1	5 5	9 9	13 D
2 2	6 6	10 A	14 E
3 3	7 7	11 B	15 F

Obr. 9 Súbor znakov používaný pri kódovaní pomocou Base16 [26]

```
BASE16("") = ""

BASE16("f") = "66"

BASE16("fo") = "666F"

BASE16("foo") = "666F6F"

BASE16("foob") = "666F6F62"

BASE16("fooba") = "666F6F6261"

BASE16("foobar") = "666F6F626172"
```

Obr. 10 Testovanie kódovania znakov pomocou algoritmu Base16 [26]

Algoritmus Base16 sa často používa pre nasledujúce účely:

- Prevod binárnych dát do hexadecimálnej podoby pre lepšiu interpretáciu.
- Reprezentácia pamäte v počítači v hexadecimálnej podobe pre jednoduchšiu manipuláciu a užívateľskú čitateľnosť.
- Reprezentácia farieb v digitálnych systémoch pre jednoduchší prehľad a manipuláciu s farebným spektrom jednotlivých farieb.
- Formát adries IPv6 a MAC, čo umožňuje vytvárať obrovské množstvá jedinečných adries.

[29]

II. PRAKTICKÁ ČASŤ

6 KRITÉRIA PRE IDENTIFIKÁCIU KRYPTOGRAFICKÝCH A KÓDOVACÍCH ALGORITMOV

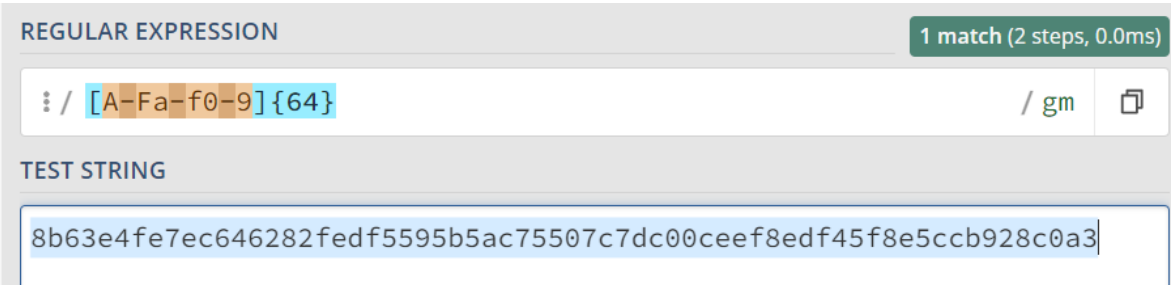
Táto kapitola sa zaoberá návrhom kritérií a metodík určených pre identifikáciu kryptografických a kódovacích algoritmov. Tieto postupy predstavujú základ pre vývoj a implementáciu aplikácie, ktorá je primárnym cieľom tejto práce. Návrhy týchto kritérií umožňujú vytvoriť nástroj, ktorý dokáže efektívne rozpoznávať a klasifikovať rôzne typy šifrovacích, hashovacích a kódovacích algoritmov.

6.1 Kritéria pre identifikáciu hashovacích algoritmov

Hashovacie algoritmy sa väčšinou vyznačujú jasným výstupným formátom, preto v rámci identifikácie hashovacích algoritmov je možné použiť regulárne výrazy, ktoré sú popísané v podkapitole 1.5 „Regulárne výrazy“. Tieto výrazy umožňujú definovať presné vzory, ktoré zodpovedajú výstupom jednotlivých hashovacích algoritmov a sú teda kľúčové pri ich identifikácii.

Známe hashovacie algoritmy, napríklad z rodiny SHA-2, generujú výsledný hash v hexadecimálnom formáte a v určenej bitovej dĺžke, ktorá závisí od konkrétneho algoritmu. Napríklad výstup produkovaný algoritmom SHA-256 je tvorený znakmi z hexadecimálnej abecedy a jeho počet znakov je vždy 64.

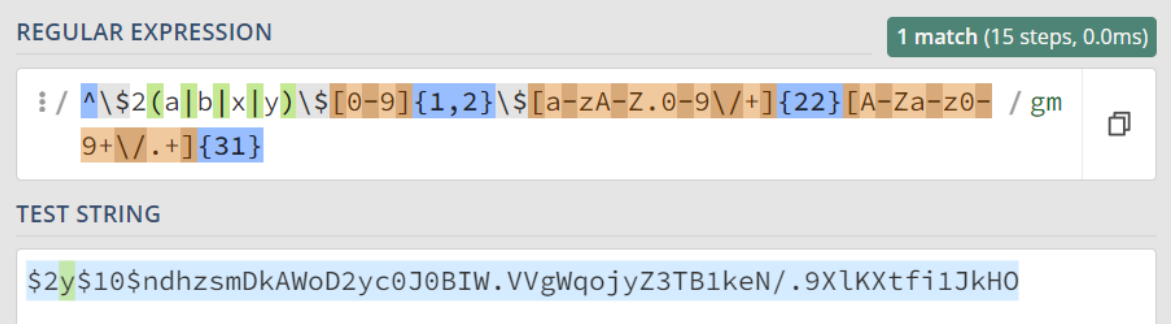
Na obrázku Obr. 11 je zobrazená podoba jednoduchého regulárneho výrazu pre hash produkovaný napríklad algoritmom SHA-256. Tento regulárny výraz je schopný nájsť zhodu s reťazcami znakov, ktoré používajú malé písmená a-f alebo veľké písmená A-F, čísla od 0 do 9 a majú dĺžku 64 znakov.



Obr. 11 Regulárny výraz pre hash produkovaný algoritmom SHA-256 [30]

Ostatné hashovacie algoritmy podobné k rodine SHA-2, ako napríklad algoritmy z rodiny MD, SHA-3, Whirlpool a ďalšie, sa líšia iba veľkosťou výstupného hashu, preto je nutné iba prispôsobiť daný regulárny výraz konkrétnemu počtu znakov.

Iné typy algoritmov, ako Bcrypt, Argon2 alebo Scrypt, sa generujú spolu s parametrami, ktoré bližšie špecifikujú ich nastavenia. Pomocou týchto dodatočných parametrov, ktoré sú pre jednotlivé algoritmy popísané v podkapitolách 4.5 „Bcrypt“, 4.6 „Argon2“ a 4.7 „Scrypt“, je možné vytvoriť regulárne výrazy, ktoré dokážu jednoznačne identifikovať aj tieto typy hashovacích algoritmov. Ako príklad je na obrázku Obr. 12 uvedený regulárny výraz [31] pre výstupy, ktoré produkuje algoritmus Bcrypt.

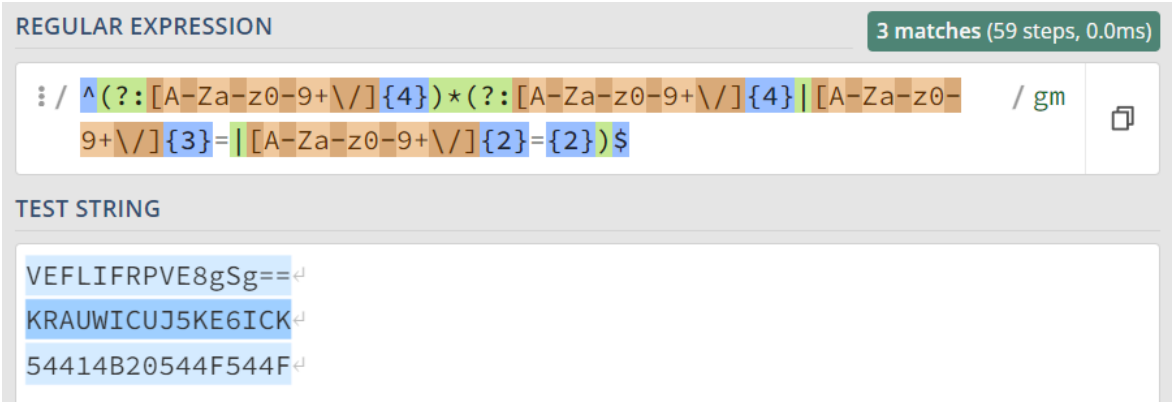


Obr. 12 Regulárny výraz pre hash produkovaný algoritmom Bcrypt [30]

6.2 Kritéria pre identifikáciu kódovacích algoritmov

Kódovacie algoritmy typu Base sa pri kódovaní správ opierajú o používanú abecedu, ktorá je vo väčšine prípadov tvorená malou a veľkou abecedou, číslami a prípadne dodatočnými špeciálnymi znakmi, ako je ukázané v podkapitolách 5.1.1 „Base64“, 5.1.2 „Base32“ a 5.1.3 „Base16“.

Pre identifikáciu konkrétnych algoritmov nie je vhodné používať regulárne výrazy, nakoľko nedokážu vždy jednoznačne určiť, ktorým kódovacím algoritmom bola správa zakódovaná. Dôkaz tohoto tvrdenia je možné vidieť na obrázku Obr. 13, kde je zobrazený regulárny výraz [32], určený pre identifikovanie správ kódovaných pomocou algoritmu Base64. Ten určil všetky tri kódované reťazce ako potencionálne kódovanie pomocou Base64, no v skutočnosti je správa v druhom riadku kódovaná pomocou algoritmu Base32 a správa v treťom riadku pomocou Base16.



Obr. 13 Testovanie regulárneho výrazu určeného pre kódovanie Base64 [30]

Pre presnejšiu a efektívnejšiu identifikáciu je vhodnejšie systematické testovanie dekódovania pomocou viacerých verzií algoritmov typu Base. V prípade že sa dekódovanie podarí, a niektorý algoritmus vráti dekódovaný reťazec znakov, ktorý je možné interpretovať vo formáte UTF-8, je takmer isté, že sa jedná o správne dekódovanie a správny kódovací algoritmus. V prípade, že po skúške dekódovania zo strany daných typov algoritmov Base nebude získané žiadne úspešné dekódovanie, čo môže byť spôsobené napríklad neplatnou vstupnou správou, alebo použitím iného neznámeho kódovania, príde oznámenie, že nie je možné danú správu jednoznačne dekódovať a určiť tak pôvodný algoritmus, ktorým bola správa kódovaná.

6.3 Kritéria pre identifikáciu šifrovania a kryptografických algoritmov

V tejto kapitole sú popísané návrhy analýzy šifrovania a možných identifikácií kryptografických algoritmov v komprimovaných archívoch, súboroch rôzneho formátu alebo digitálnych certifikátoch.

6.3.1 Analýza komprimovaných archívov formátu ZIP

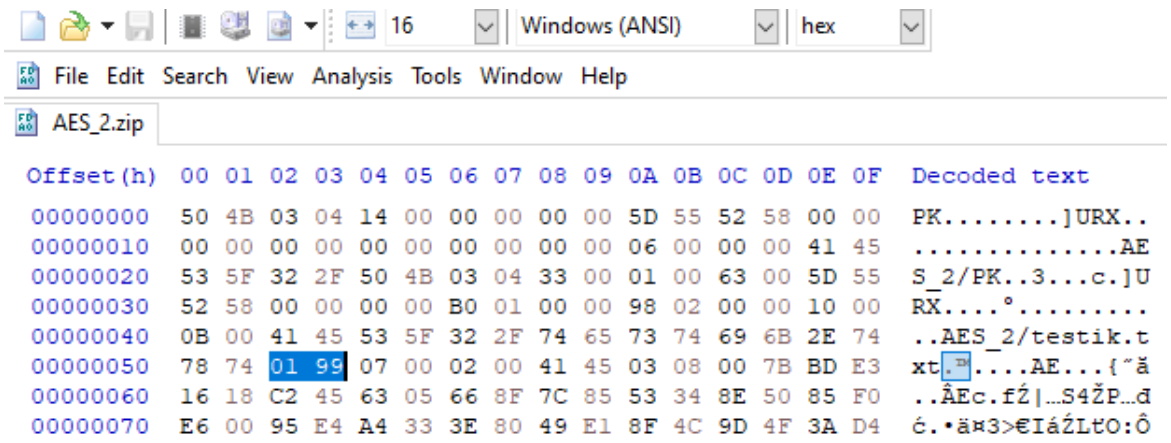
V rámci komprimovaných súborov typu ZIP je možné identifikovať nielen či je archív šifrovaný, ale aj konkrétny šifrovací algoritmus.

Prvým indikátorom toho, že archív ZIP je šifrovaný, udáva prvý bit, nachádzajúci sa v informáciách o atribútoch archívu, ktoré sa súhrnne označujú ako „general purpose bit flag“ (všeobecné bitové znaky). Ak tento bit je nastavený na hodnotu 1, znamená to že archív je šifrovaný. Naopak ak je hodnota nastavená na 0, archív nevyužíva žiadne šifrovanie. [33]

Komprimačné nástroje, ako napríklad 7-Zip alebo WinRar, ponúkajú ako hlavnú metódu šifrovania algoritmus AES, ktorého účely a možnosti využitia sú popísané v podkapitole

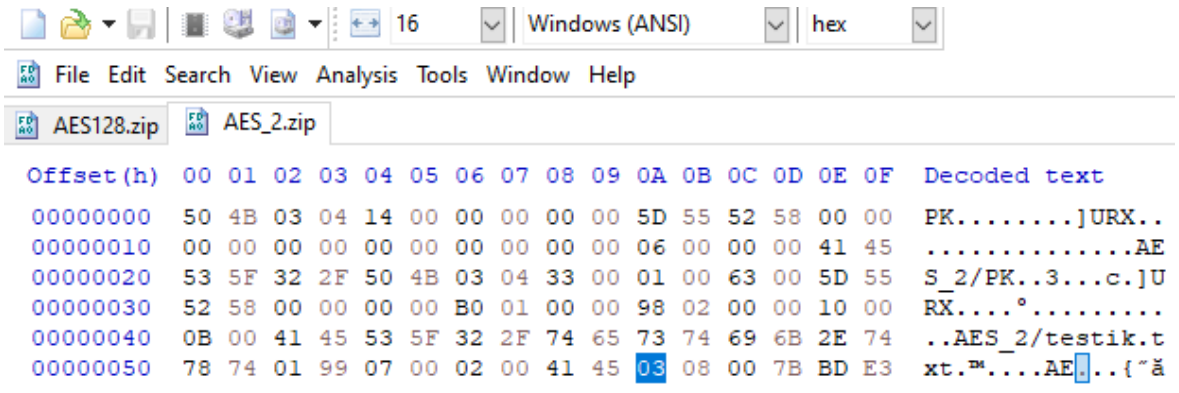
2.2.1 „Advanced Encryption Standard (AES)“. V prípade jeho použitia pre šifrovanie archívu, je algoritmus jednoznačne identifikovateľný, a to pridaním extra hodnoty 0x9901 do hlavičky súboru. [33]

Táto hodnota je uvádzaná vo formáte big-endian, ale obrázok Obr. 14, kde je šifrovaný archív otvorený v hexadecimálnom editore, poukazuje na použitie formátu little-endian, keďže sa tu nachádza prevrátená hodnota 0x0199. Nájdenie tejto hodnoty je jasným potvrdením, že pre šifrovanie bol použitý práve šifrovací štandard AES.



Obr. 14 Nájdená hodnota 0x0199 v hlavičke šifrovaného archívu ZIP

Ďalej je možné identifikovať aj verziu algoritmu AES, ktorej indikátor je vzdialený o 8 pozícii ďalej od identifikátora 0x0199, pričom hodnota 0x01 značí verziu AES-128, hodnota 0x02 značí verziu AES-192 a hodnota 0x03 značí verziu AES-256. Tento indikátor je zobrazený na obrázku Obr. 15, z ktorého vyplýva, že pre šifrovanie bola použitá práve verzia AES s dĺžkou kľúča o veľkosti 256 bitov. [33]

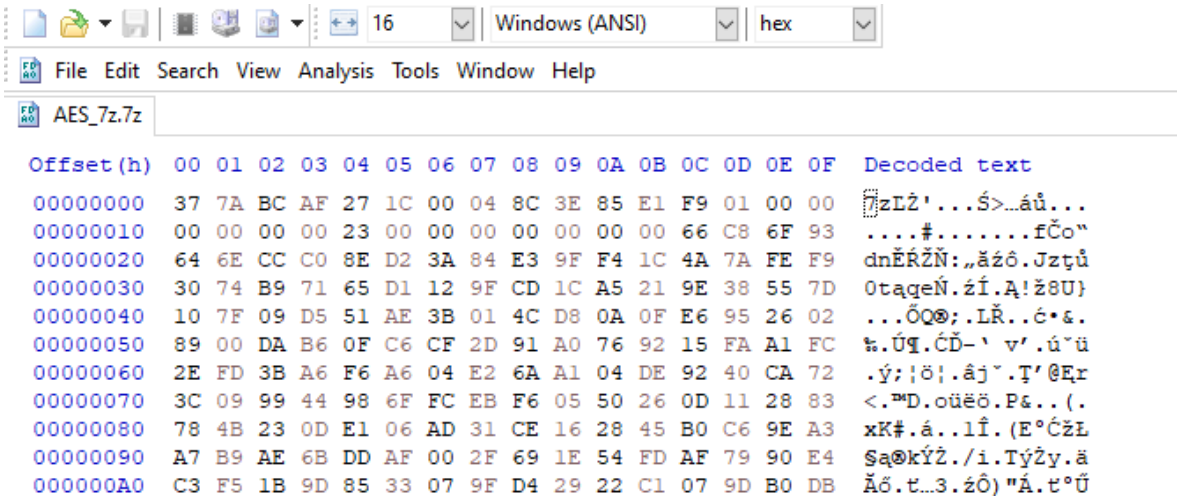


Obr. 15 Nájdená hodnota 0x03, ktorá indikuje typ algoritmu AES-256

6.3.2 Analýza komprimovaných archívov formátu 7z

Archívy formátu 7z neposkytujú, podľa dostupných informácií v oficiálnej dokumentácii tohto formátu [34], žiadne relevantné informácie o možných identifikátoroch šifrovania. V rámci identifikácie, či je daný archív šifrovaný, je preto možnosťou použitie jednoduchého overenia, či daný archív vyžaduje pre manipuláciu heslo, ktoré by značilo, že archív je šifrovaný. Na obrázku Obr. 16 je zobrazený šifrovaný archív formátu 7z, ktorý je analyzovaný v hexadecimálnom editore.

V tomto prípade nie je možné jednoznačné identifikovanie použitého šifrovacieho algoritmu, aj keď v dokumentácií je uvedené, že tento formát silne podporuje šifrovanie algoritmom AES, ktorý je popísaný v podkapitole 2.2.1 „Advanced Encryption Standard (AES)“, neexistuje žiadne jednoznačné potvrdenie, ktoré by podobne ako v prípade formátu ZIP, zaručovalo presne identifikovanie šifrovacieho algoritmu na základe špecifikovaných indikátorov vložených do štruktúry tohto formátu.



Obr. 16 Hlavička šifrovaného archívu formátu 7z

6.3.3 Meranie entropie a detekcia šifrovania súborov

Vhodnou metódou pre detekciu šifrovania súborov je použitie Shannonovej metódy pre meranie informácie, ktorá je uvedená a vysvetlená v podkapitole 1.4 „Shannonova entropia“.

Výsledkom výpočtu tejto metódy je hodnota, ktorá napovedá o miere náhodnosti dát a na jej základe je možné určiť, či je súbor šifrovaný. Je však potrebné rozlišovať medzi možnými komprimovanými súbormi, ako napríklad grafickými formátmi, ktoré aj keď nemusia byť šifrované, vykazujú vysoké hodnoty entropie práve na základe použitia určitej kompresie.

V prípade dokumentov PDF a MS Word, ktoré taktiež vykazujú vysoké hodnoty entropie, je však možné zistiť prítomnosť ochrany týchto dokumentov pomocou hesla, čo môže, ale aj nemusí naznačovať šifrovanie týchto súborov.

Kryptografický algoritmus, ktorý vykonal šifrovanie, je veľmi náročné identifikovať zo samotného šifrovaného textu. Je však možné pokúsiť sa rozdeliť šifrovaný text na bloky o veľkosti 128 bitov bez akéhokoľvek zvyšku. Ak je takéto rozdelenie úspešné, je možné predpokladať, že pre šifrovanie bol použitý symetrický blokový algoritmus, ako napríklad AES alebo IDEA, ktoré sú popísané v podkapitolách 2.2.1 „Advanced Encryption Standard (AES)“ a 2.2.2 „International Data Encryption Algorithm (IDEA)“. V opačnom prípade, ak rozdelenie na bloky nie je možné bez zvyšku, je isté, že nebol použitý symetrický blokový algoritmus.

6.3.4 Analýza digitálnych certifikátov

Digitálne certifikáty štandardu X.509 obsahujú detailné informácie o kryptografických prostriedkoch použitých pri tvorbe certifikátu. Tieto informácie zahŕňajú okrem iného aj algoritmus použitý pre generovanie kľúčov, algoritmus určený pre podpis certifikátu, dĺžku kľúčového páru alebo hashovací algoritmus. [35]

Pre identifikáciu použitých kryptografických prostriedkov je preto možné použiť modul X.509 v rámci knižnice *cryptography* [36], ktorý obsahuje potrebné funkcionality pre načítanie dát a analýzu digitálnych certifikátov.

7 IMPLEMENTÁCIA APLIKÁCIE

V tejto kapitole je popísaný vývoj aplikácie, ktorý vychádza z výsledkov analýzy a návrhu kritérií pre identifikáciu kryptografických prostriedkov. Podrobne rozoberá architektúru desktopovej aplikácie, ktorá je hlavným cieľom tejto práce, od výberu vhodných a optimálnych technológií až po prehľad jednotlivých naprogramovaných modulov.

7.1 Výber technológií pre vývoj aplikácie

Pri tvorbe aplikácie bolo cieľom vybrať technológie, ktoré by zabezpečili flexibilitu s možnosťou spustiť aplikáciu v rámci rôznych operačných systémov. Týmto požiadavkom vyhovovalo spojenie programovacieho jazyka Python, ktorý prináša jednoduchosť a rôznorodosť v rámci vývoja, a frameworku Qt, ktorý slúži pre tvorbu grafického užívateľského rozhrania, pomocou knižnice s názvom PyQt5. Toto spojenie vytvára silný základ pre efektívny vývoj aplikácie s dôrazom na užívateľskú prívetivosť.

Počas vývoja bolo rovnako dôležité pracovať aj s ďalšími knižnicami, ktoré ovplyvnili celkovú architektúru aplikácie v rámci efektivity a zjednodušenia implementácie. Vybrané knižnice sú uvedené a popísané v tabuľke Tab. 1.

Knižnica	Popis
cryptography	využívanie kryptografických prostriedkov
zipfile	práca s archívmi formátu ZIP
py7zr	práca s archívmi formátu 7z
base64	využívanie určitých algoritmov typu Base
base45	využívanie algoritmu Base45
re	operácie s regulárnymi výrazmi
os	využívanie funkcií operačného systému
math	využívanie matematických funkcií
fitz, msoffcrypto	práca s dokumentmi typu PDF a MS Word

Tab. 1 Súhrn vybraných knižníc používaných v rámci implementácie

7.2 Popis a architektúra aplikácie

V tejto podkapitole je prezentovaná finálna podoba aplikácie, vytvorená pomocou technológií popísaných v predchádzajúcej podkapitole. Štruktúra aplikácie sa opiera o analýzy a návrhy jednotlivých kritérií, ktoré sú popísané v kapitole 6 „Kritéria pre identifikáciu kryptografických a kódovacích algoritmov".

Aplikácia bola v rámci prehľadnosti rozdelená do štyroch modulov *CAIT_main.py, HashWindow.py, CodeWindow.py* a *EncryptionWindow.py,* pričom funkcie a účely jednotlivých modulov sú bližšie popísané v nasledujúcej časti. Pre prehľad sú, popri opise fungovania jednotlivých modulov, zobrazené aj vývojové diagramy približujúce tok fungovania jednotlivých metód v rámci aplikácie.

7.2.1 CAIT_main.py

Tento modul predstavuje úvodnú časť aplikácie. Zodpovedá za jej spustenie a zobrazenie úvodného okna, ktoré slúži ako hlavný navigačný uzol.

V rámci modulu *CAIT_main.py* sú definované tri metódy – *openHashWindow(), openCodeWindow()* a *openEncryptionWindow().* Tieto metódy slúžia pre zobrazenie ďalších grafických okien, ktoré užívateľovi umožňujú priamo interagovať s nástrojom.

Metóda *openHashWindow()* otvára okno, v ktorom užívateľ môže identifikovať rôzne hashovacie algoritmy. O funkcionalitu tohoto okna sa stará modul *HashWindow.py*, ktorý je vysvetlený a popísaný v podkapitole 7.2.2 „HashWindow.py".

Ďalšia metóda, *openCodeWindow()*, zobrazuje ďalšie grafické rozhranie, ktoré umožňuje identifikáciu kódovacích algoritmov na základe určitého kódovaného vstupu. Konkrétne sa o celkovú logiku v prípade využívania tohoto okna stará modul *CodeWindow.py,* ktorý je bližšie uvedený v podkapitole 7.2.3 „CodeWindow.py".

Posledná zo zmienených metód, *openEncryptionWindow()*, presmeruje užívateľa do okna aplikácie, ktoré dovoľuje užívateľovi identifikovanie šifrovania súborov, prípadne aj šifrovacích algoritmov. Funkcionalitou tohto grafického rozhrania sa zaoberá modul *EncryptionWindow.py*, ktorý je podrobnejšie vysvetlený v podkapitole 7.2.4 „EncryptionWindow.py".

7.2.2 HashWindow.py

Pre identifikovanie hashovacích algoritmov slúži modul *HashWindow.py*. Nachádzajú sa tu metódy, ktoré sú navrhnuté tak, aby nielen definovali logiku a procesy spojené s identifikovaním hashovacích algoritmov, ale aj zabezpečovali interakciu s užívateľom prostredníctvom grafického rozhrania.

Na začiatku modulu sú definované regulárne výrazy, pomocou ktorých je možné identifikovanie hashovacích algoritmov, ako to bolo spomínané v podkapitole 6.1 „Kritéria pre identifikáciu hashovacích algoritmov“. Sú definované v dátovom type *dictionary (slovník)*, ktorého štruktúra umožňuje na základe zhody regulárneho výrazu poskytnúť možné hashovacie algoritmy, ktoré daný hash mohli vyprodukovať. Na nasledujúcej stránke sú definované regexy v slovníku *hash_patterns* pre bežné hashovacie algoritmy, ako MD alebo SHA, ale aj menej známe, nespomínané algoritmy, ako napríklad TIGER alebo Whirlpool, ktoré všetky produkujú výstupy v hexadecimálnom formáte a líšia sa iba dĺžkou produkovaného hashu.

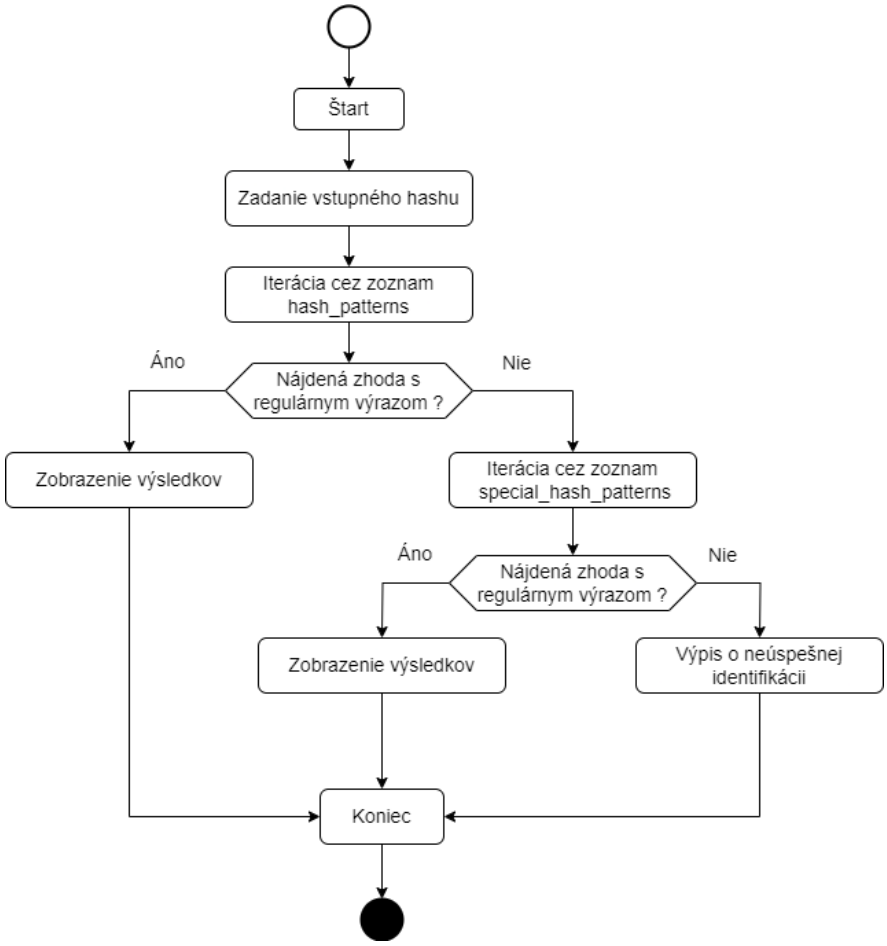
```
hash_patterns = {

     '128 bitov': {
         'regex': re.compile(r'^[a-fA-F0-9]{32}$'),
          'algorithms': ['MD2', 'MD4', 'MD5', 'MD6-128','RIPEMD- 128',
          'TIGER-128', 'NTLM']
      },
     '160 bitov': {
         'regex': re.compile(r'^[a-fA-F0-9]{40}$'),
         'algorithms': ['SHA-1', 'RIPEMD-160', 'TIGER-160']
      },
     '192 bitov': {
         'regex': re.compile(r'^[a-fA-F0-9]{48}$'),
         'algorithms': ['TIGER-192']
      },
     '224 bitov': {
         'regex': re.compile(r'^[a-fA-F0-9]{56}$'),
         'algorithms': ['SHA-224', 'SHA3-224']
      },
```

```
        '256 bitov': {
           'regex': re.compile(r'^[a-fA-F0-9]{64}$'),
           'algorithms': ['SHA-256', 'SHA3-256', 'MD6-256']
        },
        '320 bitov': {
            'regex': re.compile(r'^[a-fA-F0-9]{80}$'),
            'algorithms': ['RIPEMD-320']
        },
        '384 bitov': {
            'regex': re.compile(r'^[a-fA-F0-9]{96}$'),
            'algorithms': ['SHA-384', 'SHA3-384']
        },
       '512 bitov': {
            'regex': re.compile(r'^[a-fA-F0-9]{128}$'),
            'algorithms': ['SHA-512', 'SHA3-512', 'MD6-512', 'Whirlpool-
            512']
        }
```

Regulárne výrazy pre hashovacie algoritmy Bcrypt, Argon2 a Scrypt, sú definované v slovníku *special_hash_patterns.* Pre scrypt sú vytvorené dva výrazy pre možné formáty výstupu tohto algoritmu, ktoré sú popísané v podkapitole 4.7 „Scrypt“.

```
special_hash_patterns = {
      'bcrypt':re.compile(r'^\$(2[abyx])\$([09]{2})\$(.{22})([./A- Za-z0-
      9]{31})$'),
      'Argon2':re.compile(r'^\$argon2([id]{0,2})\$v=(\d+)\$m=(\d+),t=(\d
      +),p=(\d+)\$([A-Za-z0-9+/]+)\$([A-Za-z0-9+/]*=?=?)$'),
      'scrypt_1' : re.compile(r'^(\d+)\$(\d+)\$(\d+)\$([A-Za-z0-
      9+/=]+)\$([a-fA-F0-9]+)$'),
      'scrypt_2' : re.compile(r'^SCRYPT:(\d+):(\d+):(\d+):([A-Za-z0-
      9+/=]+):([A-Za-z0-9+/=]+)$')
}
```

Na obrázku Obr. 17 zobrazený princíp fungovania metódy *identify_hash()* vo forme vývojového diagramu pre lepšie pochopenie jej činnosti. Táto metóda predstavuje hlavnú funkcionalitu v rámci modulu *HashWindow.py.*



Obr. 17 Vývojový diagram metódy *identify_hash()*

Vstupným parametrom metódy *identify_hash()* je hash, ktorý zadá užívateľ do vstupného poľa alebo vyberie súbor s hashom. Iteratívne prechádza cez zoznamy regulárnych výrazov pomocou cyklu *for* a snaží sa nájsť zhodu medzi ním a daným vstupom pomocou metódy *search()*, ktorá je súčasťou knižnice *re*. Ak je zhoda nájdená, táto metóda zaistí, že sa užívateľovi zobrazia výsledné informácie o danom vstupe. Tie sa obsahovo líšia:

- V prípade že bola nájdená zhoda s regulárnym výrazom definovaným v *hash_patterns*, sú vypísané v rámci GUI nasledujúce informácie:
  - Bitová dĺžka hashu.
  - Počet znakov, ktoré tvoria daný hash.
  - Možné hashovacie algoritmy, ktoré dokážu produkovať daný hash.

- V prípade, že zhoda je s jedným z výrazov definovaných v *special_hash_patterns,* sú vypísané do GUI tieto informácie:
    - Bitová dĺžka hashu.
    - Počet znakov, ktoré tvoria daný hash.
    - Hashovací algoritmus, ktorému daný hash odpovedá.
    - Dodatočné špecifické informácie, ktoré sú charakteristické pre daný algoritmus.

Ďalšie metódy, nachádzajúce sa v module *HashWindow.py* plnia rôzne funkcionality spojené s analýzou vstupu alebo obsluhou grafického rozhrania :

- *choose_file()* – stará sa o možnosť načítania súboru s hashmi z dialogového okna a ich následnej identifikácie.
- *load_next_hash()* – zabezpečuje postupné načítavanie hashov zo súboru a používa sa v metóde *choose_file().*
- *identify_hash_input()* – získava vstupný hash z textového poľa GUI a následne volá metódu *identify_hash()*, pričom jej tento vstup predá ako argument.
- *base64_to_hex()* – parametrom je kódovaný reťazec Base64. Metóda vykoná v prípade potreby doplnenie (padding) a následne dekódovaný reťazec vráti v hexadecimálnom formáte. Používa sa pre prevod hashu z kódovania Base64 do hexadecimálnej podoby.
- *clear_results()* – vyčistí informácie a výsledky zobrazené v GUI a skryje vybrané elementy, čo nastaví okno do pôvodného stavu.
- *return_to_MainWindow()* – zabezpečuje navigáciu späť na hlavné okno.

7.2.3 CodeWindow.py

Modul *CodeWindow.py* sa stará o časť aplikácie, v ktorej sa vykonáva identifikovanie kódovacieho algoritmu na základe vstupnej kódovanej správy. Aplikácia dokáže rozpoznať niekoľko algoritmov typu Base, konkrétne Base94, Base85, Base64, Base62, Base58, Base56, Base45, Base36, Base32 a Base16.

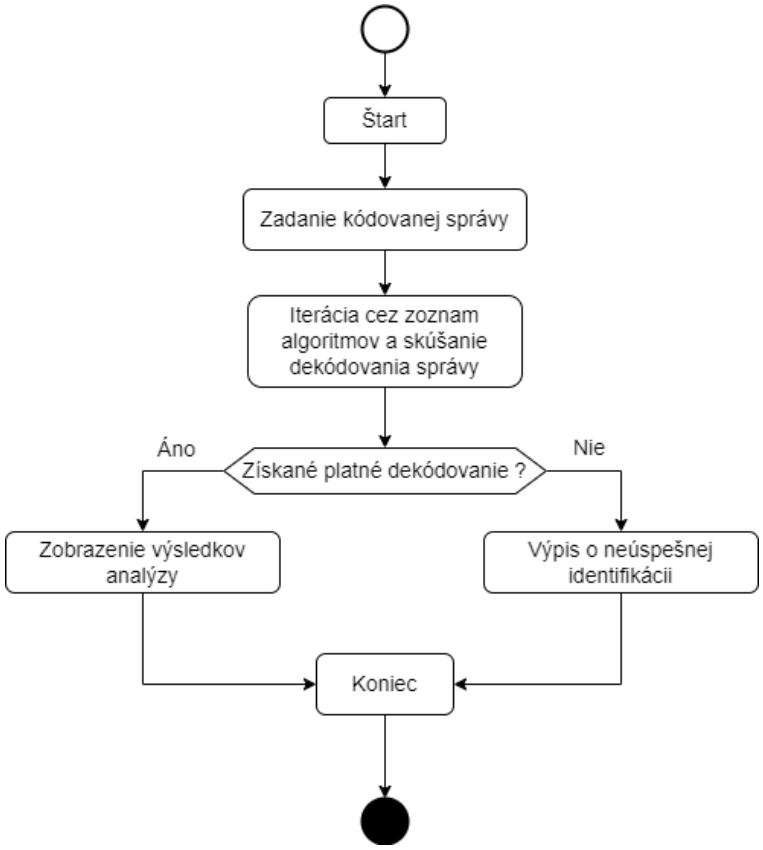
Na začiatku sú v module definované metódy dekódovania pre tieto algoritmy. Pre Base85, Base64, Base45, Base32 a Base16 nemusela byť použitá vlastná implementácia, pretože

je možné ich používať vďaka importovaniu knižníc *base64* a *base45*, ktoré poskytujú metódy pre vykonávanie kódovania a dekódovania dát pomocou týchto algoritmov. Naopak pre Base94, Base62, Base58, Base56 a Base36 boli vytvorené metódy, ktoré zabezpečujú dekódovanie na základe abecedy používanej v rámci jednotlivých algoritmov.

Na nasledujúcom kóde je pre ukážku zobrazená metóda *base36_decode()*. Jej vstupným parametrom je kódovaná správa v kódovaní base36. Následne prechádza každý znak tejto správy pomocou cyklu *for*. V rámci cyklu vypočíta číselnú hodnotu, ukladanú do premennej *num,* pričom vynásobí predošlú hodnotu premennej *num* číslom 36, ktorý je základom tohto systému, a pripočíta jeho pozíciu v rámci definovanej abecedy. Takto sa následne po prejdení všetkých znakov získa celková hodnota dekódovanej správy. Následne je pre výslednú hodnotu vypočítaná bitová dĺžka potrebná pre reprezentáciu tejto hodnoty v binárnej podobe, ktorá je potrebným argumentom metódy *to_bytes()*. Táto metóda prevádza konečnú hodnotu premennej *num* na reťazec bajtov. Ostatné metódy pre algoritmy Base, ktoré potrebujú vlastnú implementáciu, sa zakladajú na rovnakej logike a líšia sa iba používanou abecedou a číslom vo výpočte hodnoty, ktoré odpovedá danému systému.

```
def base36_decode(code_input):
        alphabet = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"
        num = 0
        for char in code_input:
            num = num * 36 + alphabet.index(char)
    byte_length = (num.bit_length() + 7) // 8
    return num.to_bytes(byte_length, 'big')
```

O funkcionalitu identifikácie kódovacích algoritmov sa stará metóda *identify_code_algorithm(),* ktorá zabezpečuje hlavnú funkcionalitu tohto modulu. Pracuje na základe návrhu z podkapitoly 6.2 „Kritéria pre identifikáciu kódovacích algoritmov“. Grafické znázornenie fungovania metódy je zobrazené na obrázku Obr. 18 vo forme vývojového diagramu.



Obr. 18 Vývojový diagram metódy *identify_code_algorithm()*

Vstupným argumentom metódy *identify_code_algorithm()* je kódovaná správa zadaná užívateľom. Metódy, ktoré sa pokúšajú dekódovať vstupnú správu jednotlivými algoritmami, sú umiestnené v dátovom type *zoznam (list)* s názvom *decode_functions*, cez ktorý metóda postupne iteruje pomocou cyklu *for*. Ak je nájdené nejaké úspešné dekódovanie, ktoré je možné interpretovať vo formáte kódovania *UTF-8,* vypíšu sa do grafického rozhrania informácie o úspešnom dekódovaní vstupnej správy, ktoré zahŕňajú:

- Zistený algoritmus, ktorým bola vstupná správa kódovaná.
- Abecedu, ktorú algoritmus používa v rámci kódovania a dekódovania.
- Dekódovanú správu.

Ďalšie metódy v module *CodeWindow.py* sú:

- *identify_code_input()* - získava vstupnú správu z textového poľa GUI a následne volá metódu *identify_code_algorhitm()*, pričom jej tento vstup predá ako argument. Ak nie je poskytnutý žiadny vstup, užívateľ je upozornený aby tento vstup poskytol do textového poľa, alebo vybral súbor.

- *copy_to_clipboard()* – umožňuje dekódovanú správu skopírovať do schránky pre pohodlnejší prenos výsledku dekódovania.
- *save()* – otvára dialogové okno a umožňuje uložiť kompletný výsledok analýzy do súboru. Na začiatok tohto súboru sa následne umiestni časové razítko, aby udávalo presný čas jeho vytvorenia.
- *clear_results()* – vyčistí informácie a výsledky zobrazené v GUI a skryje vybrané elementy, čo nastaví okno do pôvodného stavu.
- *return_to_MainWindow()* – zabezpečuje navigáciu späť na hlavné okno.
- Metódy pre dekódovanie správy za pomoci rôznych algoritmov ako napríklad *base36_decode(), base56_decode()* a ďalšie.
- Metódy pre skúšanie dekódovania, ktoré vracajú informácie o úspešnom alebo neúspešnom kódovaní - *try_decode_base36(), try_decode_base56()* a ďalšie, ktoré využívajú metódy pre dekódovanie spomínané vyššie a sú umiestnené v zozname *decode_functions*.

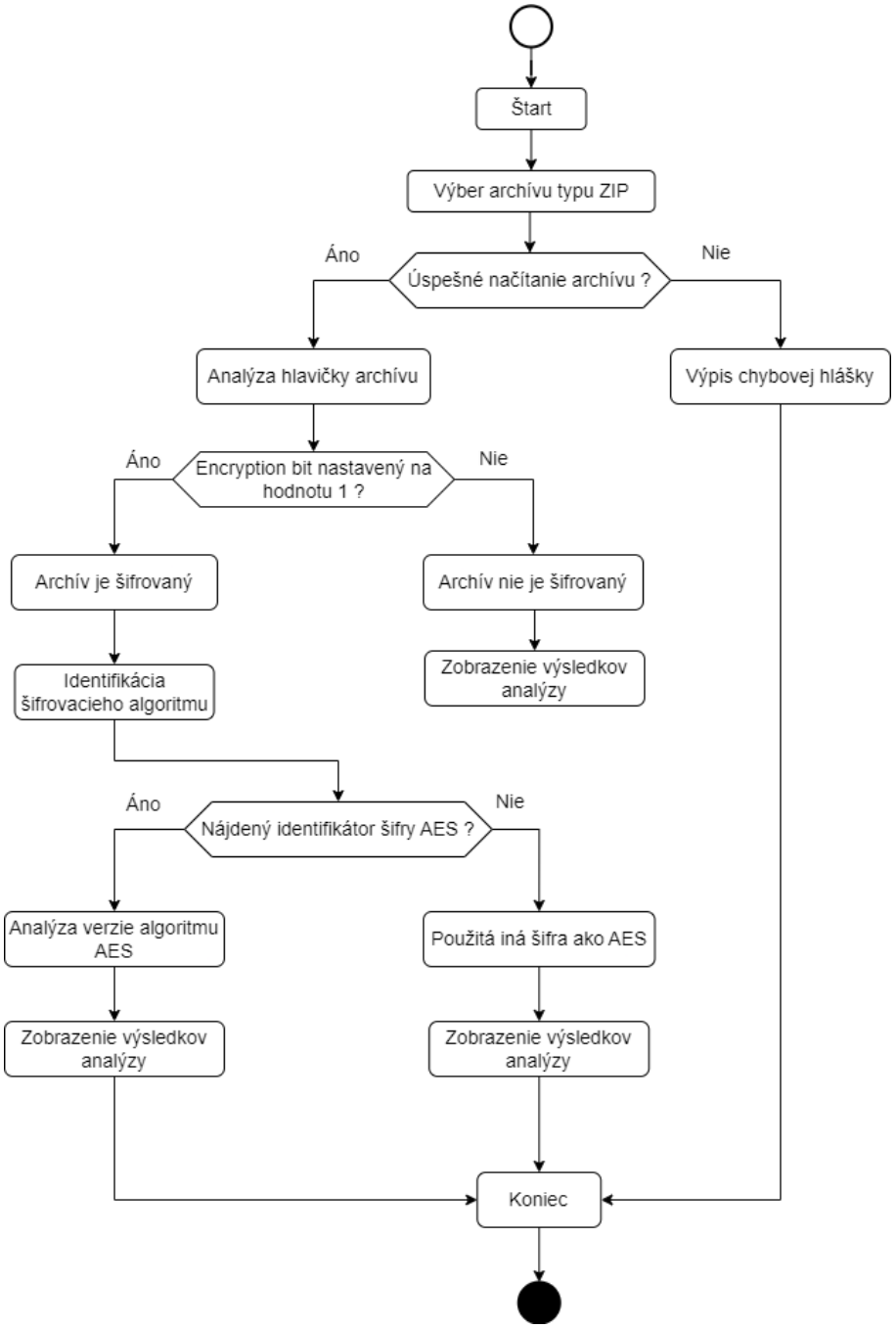
7.2.4 EncryptionWindow.py

V rámci modulu *EncryptionWindow.py* sú definované metódy, ktoré sú vytvorené na základe kritérií stanovených v podkapitole 6.3. „Kritéria pre identifikáciu šifrovania a šifrovacích algoritmov“. Modul zabezpečuje zisťovanie šifrovania v rámci archívov typu ZIP a 7z, šifrovanie súborov a identifikáciu kryptografických prostriedkov použitých v rámci digitálnych certifikátov.

check_zip_encryption()

Táto metóda je vytvorená na základe návrhov v podkapitole 6.3.1 „Analýza komprimovaných archívov formátu ZIP“.

Nižšie na obrázku Obr. 19 je zobrazený vývojový diagram metódy *check_zip_encryption()* pre grafické znázornenie fungovania a rozhodovania v rámci metódy.



Obr. 19 Vývojový diagram metódy *check_zip_encryption()*

Metóda *check_zip_encryption()* zabezpečuje identifikáciu šifrovania a šifrovacích algoritmov v rámci archívov tohto formátu. Vstupným argumentom metódy je archív typu ZIP a kľúčovú úlohu pri celkovej analýze zohráva knižnica *zipfile*. Po úspešnom načítaní sa zisťuje, či je prvý bit v rámci atribútov archívu v hlavičke nastavený na hodnotu 1 pomocou atribútu *flag_bits,* čo by znamenalo, že archív je šifrovaný. V takom prípade sa v ďalšom kroku snaží identifikovať použitý šifrovací algoritmus hľadaním identifikátora šifry AES, ktorým je hodnota 0x0199. To je možné pomocou atribútu *extra,* ktorý obsahuje dodatočné

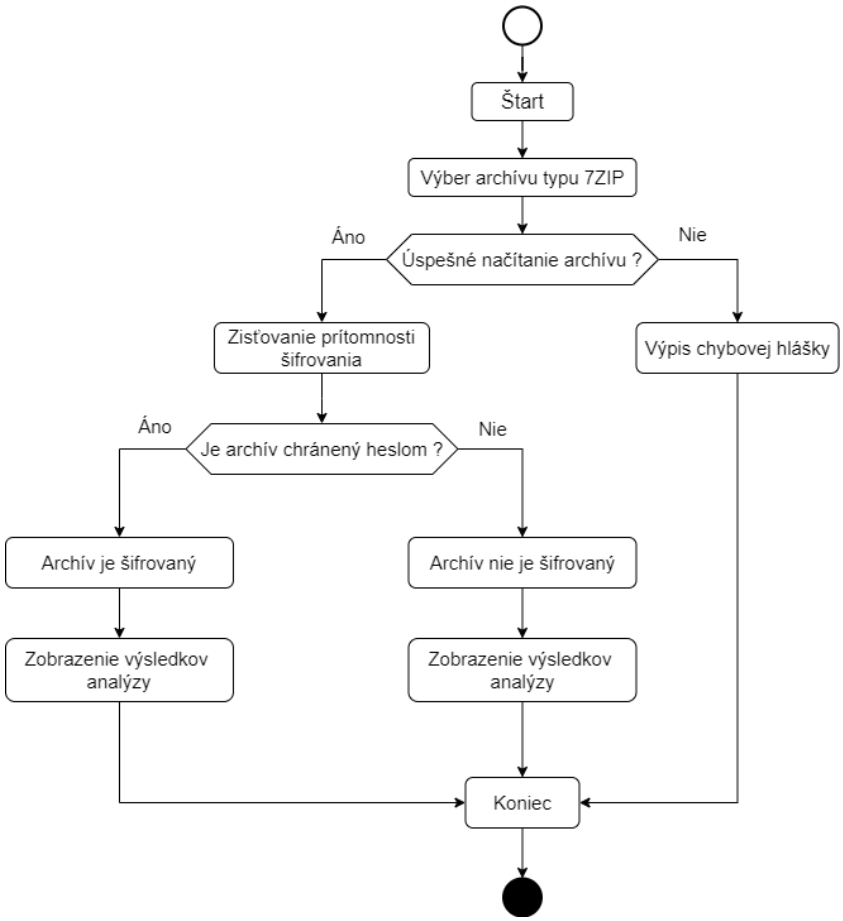
dáta spojené so štruktúrou archívu ZIP. Ak bola táto hodnota objavená, je následne zistený aj konkrétny typ AES, ktorého identifikátor sa nachádza o 8 pozícii od hodnoty 0x0199 a môže nadobúdať hodnoty 0x01 až 0x03. V prípade, že identifikátor AES nebol objavený, je usúdené, že archív používa šifrovanie pomocou iného algoritmu, napríklad ZipCrypto, ktorý je tiež ponúkanou možnosťou šifrovania.

Výstup metódy v rámci GUI poskytuje nasledujúce informácie:

- Zistená hodnota bitu, ktorá udáva či je súbor šifrovaný.
- Konštatovanie či archív je, alebo nie je šifrovaný.
- Zistený šifrovací algoritmus.

check_7z_encryption()

Metóda bola vytvorená na základe poznatkov uvedených v podkapitole 6.3.2 „Analýza komprimovaných archívov formátu 7z“. Zaisťuje identifikáciu šifrovania v rámci archívov typu 7z. Vývojový diagram *check_7z_encryption()* je zobrazený na obrázku Obr. 20.



Obr. 20 Vývojový diagram metódy *check_7z_encryption()*

Metóda *check_7z_encryption()* má jeden vstupný argument, ktorý predstavuje archív typu 7zip . Pre prácu s archívmi tohto formátu využíva knižnicu *py7zr,* ktorá poskytuje potrebné funkcionality v rámci analýzy týchto súborov. Po úspešnom načítaní archívu sa v rámci metódy zisťuje, či je potrebné heslo pre jeho otvorenie pomocou metódy *needs_password()*. Ak je heslo vyžadované (metóda vráti návratovú hodnotu *True),* znamená to že archív je šifrovaný. V tomto prípade je zložité zistiť konkrétny šifrovací algoritmus, preto vo výsledku nie je analýza použitého algoritmu vykonávaná.

Výstup tejto metódy poskytuje v grafickom rozhraní nasledujúce informácie:

- Archív vyžaduje alebo nevyžaduje heslo.
- Konštatovanie, čí archív je, alebo nie je šifrovaný.
- Konštatovanie, že sa pravdepodobne môže jednať o šifrovací algoritmus AES-256, ale nie je to možné jednoznačne potvrdiť.

Ďalšími metódami v rámci modulu sú *is_RSA_certificate()* a *is_ECC_certificate(),* ktoré vykonávajú identifikáciu použitých kryptografických prostriedkov a implementujú návrhy vykonané v podkapitole 6.3.4 „Analýza digitálnych certifikátov“. Ako bolo spomínané v danej podkapitole, v rámci týchto metód sú používané funkcionality knižnice *cryptography*. O načítavanie dát týchto certifikátov sa stará jednoduchá metóda *load_certificate()*, ktorá dokáže načítať dáta certifikátu, ktoré sú vo formáte .pem a .der (to zahŕňa aj certifikáty s koncovou príponou .crt).

is_RSA_certificate()

Na následujúcej ukážke je zobrazený kód metódy *is_RSA_certificate(),* ktorá má jeden vstupný parameter a tým je digitálny certifikát. Najprv sa vykoná zistenie typu verejného kľúča pomocou metódy *public_key()*. Následuje overenie kľúča, kedy sa zisťuje, či je tento kľúč inštanciou typu RSA pomocou metódy *isinstance()*. V prípade že inštanciou je, zisťuje sa dĺžka tohto kľúča pomocou atribútu *key_size.* Atribút *signature_algorithm_oid._name* zisťuje, aký algoritmus bol použitý pre podpis certifikátu, čo zahŕňa aj hashovací algoritmus. Metóda tak vracia trojicu hodnôt, a to informáciu, či je daný verejný kľúč typu RSA, dĺžku tohto kľúča v bitoch a názov podpisového a hashovacieho algoritmu.

```
def is_RSA_certificate(self,cert):

        public_key = cert.public_key()

        is_RSA = isinstance(public_key, rsa.RSAPublicKey)

        sign_algorithm = cert.signature_algorithm_oid._name

        if is_RSA:

                key_size = public_key.key_size
        else:

                key_size = None

        return is_RSA, key_size, sign_algorithm
```

is_ECC_certificate()

Nasledujúca ukážka zobrazuje kód metódy *is_ECC_certificate()*, ktorá pracuje takmer identicky ako predošlá metóda. Rozdiel spočíva v overovaní kľúča. V tomto prípade metóda zisťuje, či verejný kľúč je inštanciou eliptickej kryptografie (ECC). Navyše, okrem dĺžky tohto kľúča, analyzuje aký konkrétny typ eliptickej krivky bol v rámci certifikátu použitý s pomocou atribútu *curve.name.* Metóda tak vracia štvoricu hodnôt – či je verejný kľúč inštanciou ECC, dĺžku tohto kľúča v bitoch, názov použitej eliptickej krivky a názov podpisového a hashovacieho algoritmu.

```
def is_ECC_certificate(self, cert):
       public_key = cert.public_key()
       is_ECC = isinstance(public_key, ec.EllipticCurvePublicKey)
       sign_algorithm = cert.signature_algorithm_oid._name
       if is_ECC:
           curve_name = public_key.curve.name
           key_size = public_key.key_size
       else:
           curve_name = None
           key_size = None
       return is_ECC, curve_name, key_size, sign_algorithm
```

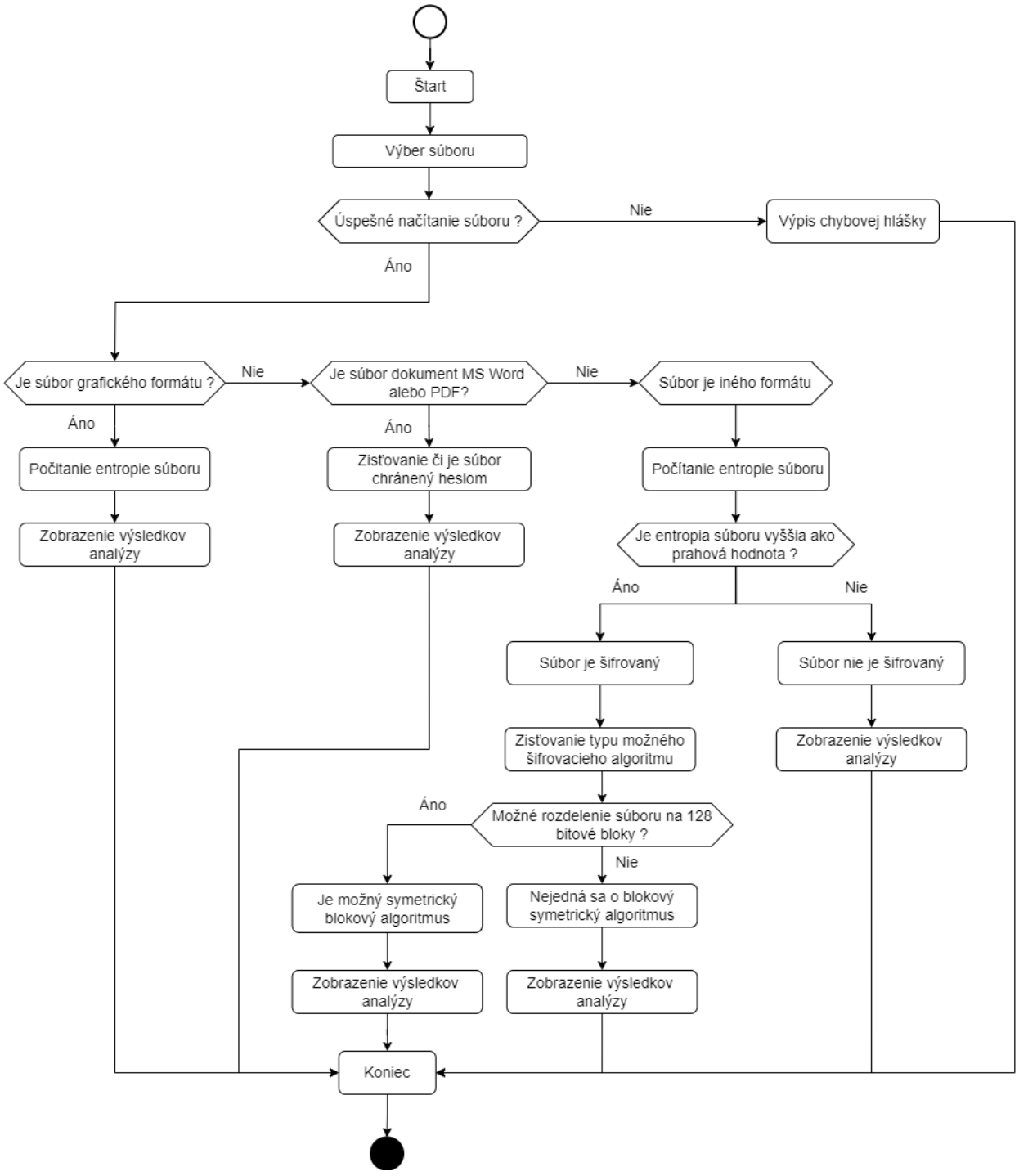
Výstupy týchto metód poskytujú v rámci grafického rozhrania nasledujúce informácie:

- Názov kryptografického algoritmu použitého pre generovanie kľúčov (RSA alebo algoritmus typu ECC).
- Dĺžka použitých kľúčov v bitoch.
- Podpisový a hashovací algoritmus použitý v rámci podpisovania certifikátu.
- V prípade, že sa jedná o ECC, aj konkrétny typ použitej eliptickej krivky.

check_if_encrypted()

Metóda *check_if_encrypted()* zabezpečuje zisťovanie šifrovania a ochrany súborov tak, ako je to navrhnuté v podkapitole 6.3.3 „Meranie entropie a detekcia šifrovania súborov“. Výpočet entropie súboru zabezpečuje metóda *calc_entropy(),* ktorá využíva matematické princípy popísané v podkapitole 1.4 „Shannonova entropia“.

Pre lepšiu orientáciu je v nasledujúcej časti na obrázku Obr. 21 zobrazené grafické znázornenie metódy *check_if_encrypted()* vo forme vývojového diagramu pre základný prehľad o jednotlivých krokoch v rámci tejto metódy.



Obr. 21 Vývojový diagram metódy *check_if_encrypted()*

V rámci *check_if_encrypted()* metódy je na začiatku vykonávané delenie súborov podľa formátov a na základe toho sú vykonávané následné akcie:

- Grafické súbory (.png, .jpg, .jpeg, .gif, .svg, .webp a .bmp) – V prípade súboru grafického formátu, je počítaná jeho entropia pomocou metódy *calc_entropy()*. V grafickom rozhraní sú následne zobrazené informácie o hodnote entropie tohto

súboru a informácia o tom, že hodnota entropie v tomto prípade nemusí znamenať šifrovanie, pretože za vysokú mieru entropie môže práve fakt, že tieto formáty bežne používajú kompresiu.

- Dokumenty MS Word alebo PDF (.docx, .doc a .pdf) – Ak sú vybrané súbory tohto typu, nevykonáva sa žiadne počítanie entropie, ale volanie metód *check_word_protected(),* respektíve *check_pdf_protected()*, pričom záleží či je daný súbor dokument MS Word alebo PDF. Obe metódy pracujú takmer identicky, len pre manipuláciu s daným formátom používajú iné knižnice. Ich úlohou je zistiť, či je súbor chránený heslom. K tomuto účelu využívajú atribút *is_encrypted*, ktorý zistí prítomnosť ochrany pomocou hesla. Následne je v rámci GUI zobrazený výsledok tejto analýzy a teda či súbor je, alebo nie je chránený heslom.
- Ostatné súbory (.txt, .locked ...) - Ak vybraný súbor nie je ani grafický súbor, ani dokument typu PDF alebo MS Word, je volaná metóda *calc_entropy(),* ktorá zistí mieru entropie daného súboru a táto hodnota sa zobrazí v grafickom rozhraní. Ak táto hodnota je väčšia ako prahová, ktorá bola po zvážení nastavená na hodnotu 7, je súbor považovaný za šifrovaný. V rámci šifrovaného súboru je následne volaná metóda *check_if_block_cipher(),* ktorá kontroluje, či je možné obsah súboru rozdeliť na presný počet blokov o veľkosti 128 bitov. V prípade že je takéto rozdelenie možné, je vypísaná informácia že šifrovanie bolo pravdepodobne vykonané pomocou symetrickej blokovej šifry, naopak v prípade ak rozdelenie na bloky nie je možné, je vypísaná hláška že sa nejedná o blokovú šifru, ale pravdepodobne zástupcu z radov symetrických prúdových alebo asymetrických algoritmov.

Ďalšie metódy v rámci modulu *EncryptionWindow.py* sú:

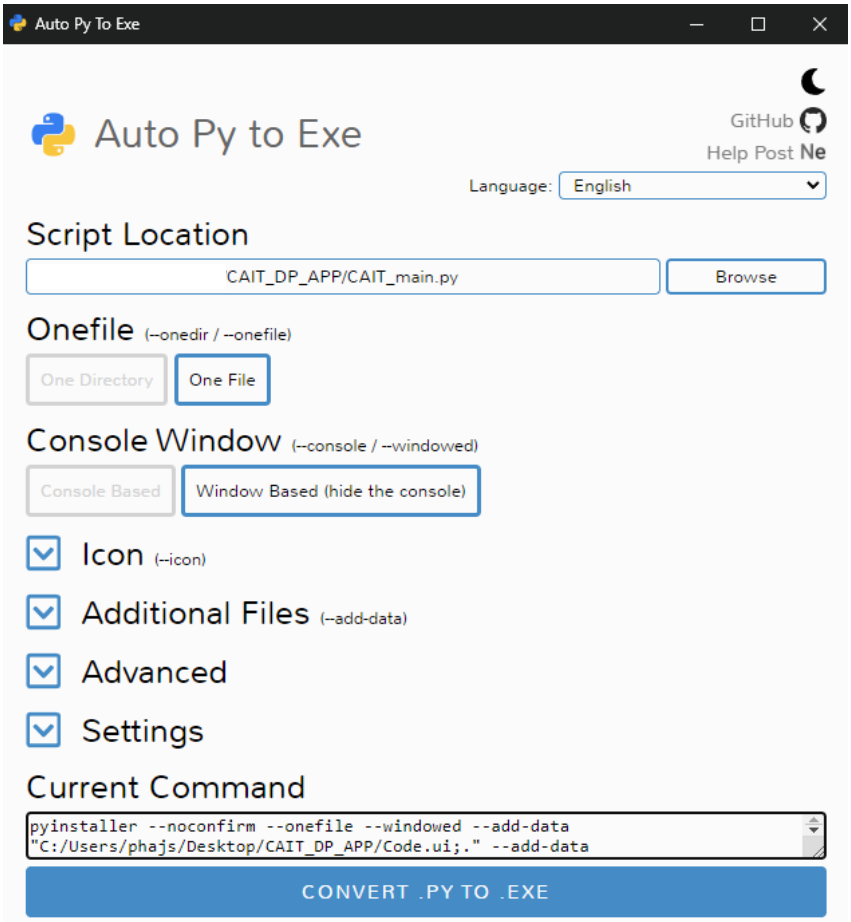
- *choose_button()* – Na základe vybraného rádiového tlačidla vykonáva načítavanie súborov a volanie príslušných metód pre analýzu a identifikáciu šifrovania a kryptografických prostriedkov.
- *return_to_MainWindow()* – zabezpečuje navigáciu späť na hlavné okno.
- *show_button_radius()* – Po tom čo je zvolené jedno z rádiových tlačidiel, táto metóda zobrazí tlačidlo, ktoré umožňuje vybrať daný súbor pre analýzu.

8 TESTOVANIE FUNKČNOSTI APLIKÁCIE

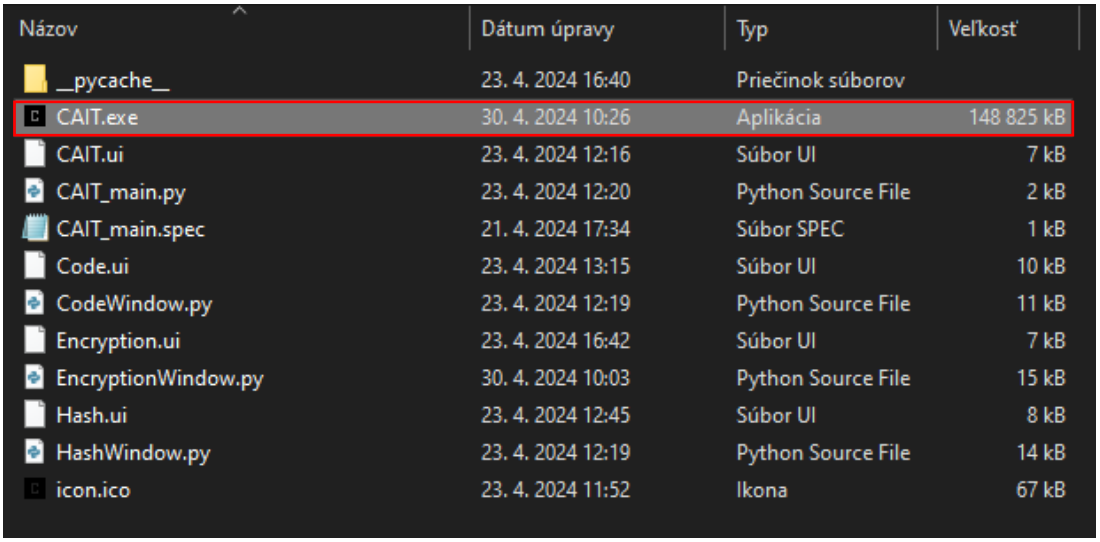
Táto kapitola sa zaoberá testovaním funkčnosti aplikácie, ktorej návrh a implementácia boli popísané v predchádzajúcej kapitole. Na začiatku je uvedený postup tvorby spustiteľného súboru na operačnom systéme Windows a následne je výsledná aplikácia testovaná a doložená vlastnými obrázkami, ktoré boli vytvorené pri overovaní jednotlivých funkcií. V rámci kapitoly je zobrazené aj testovanie multiplatformosti aplikácie, a to spustením a testovaním aplikácie aj na operačnom systéme Linux.

8.1 Vytvorenie spustiteľného súboru pre operačný systém Windows

Pre vytvorenie spustiteľného súboru bola v rámci operačného systému Windows použitá aplikácia „*Auto Py to Exe*“, ktorá je grafickým rozšírením nástroja „*Pyinstaller*“. Na obrázku Obr. 22 je zobrazené GUI, ktoré zjednodušuje použitie nástroja vybraním príslušných skriptov (súbory typu *.py)* a dodatočných súborov (GUI vo formáte *.ui* alebo napríklad ikonu aplikácie formátu *.ico*). Na obrázku Obr. 23 je zobrazený vytvorený spustiteľný súbor „*CAIT.exe*“.



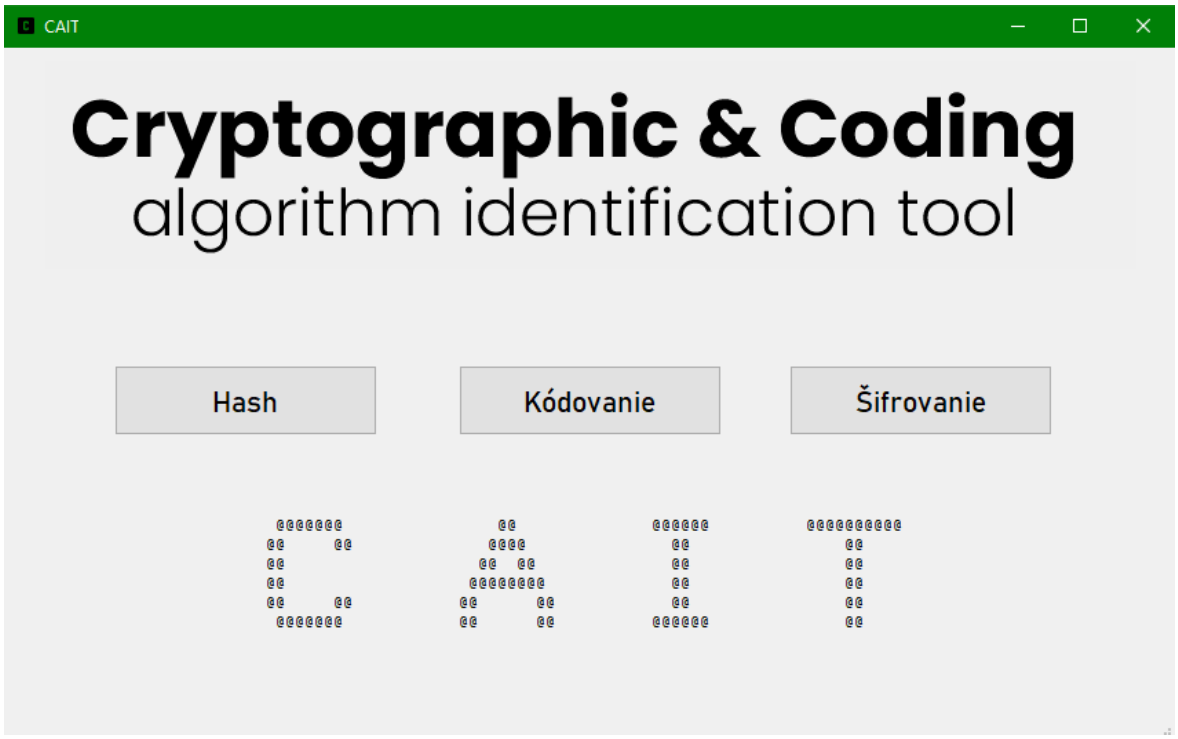
Obr. 22 Aplikácia Auto Py to Exe



Obr. 23 Spustiteľný súbor CAIT.exe

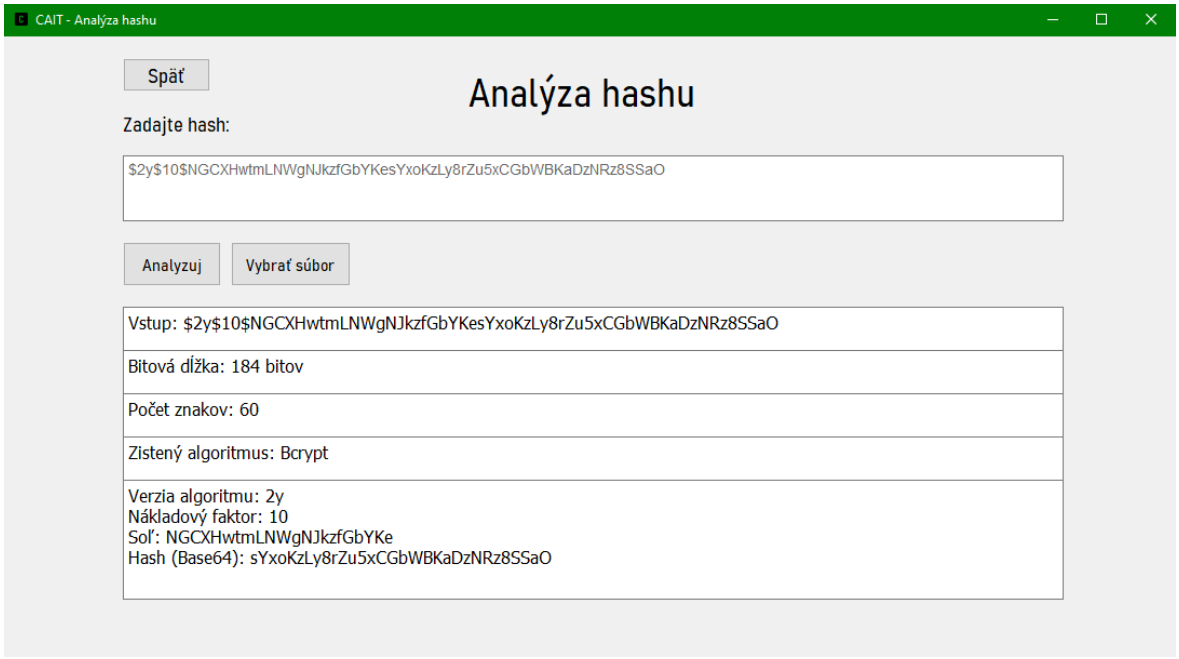
8.2 Testovanie aplikácie v operačnom systéme Windows

Po vytvorení spustiteľného súboru je aplikácia pripravená na spustenie. Na obrázku Obr. 24 je zobrazené hlavné okno, ktoré sa zobrazí ako prvé po spustení aplikácie. Následne užívateľ môže vybrať typ analýzy kliknutím na jedno z možných tlačidiel a tým otvoriť príslušné okno. Ďalšie obrázky zobrazujú jednotlivé analýzy a overujú funkčnosť aplikácie.

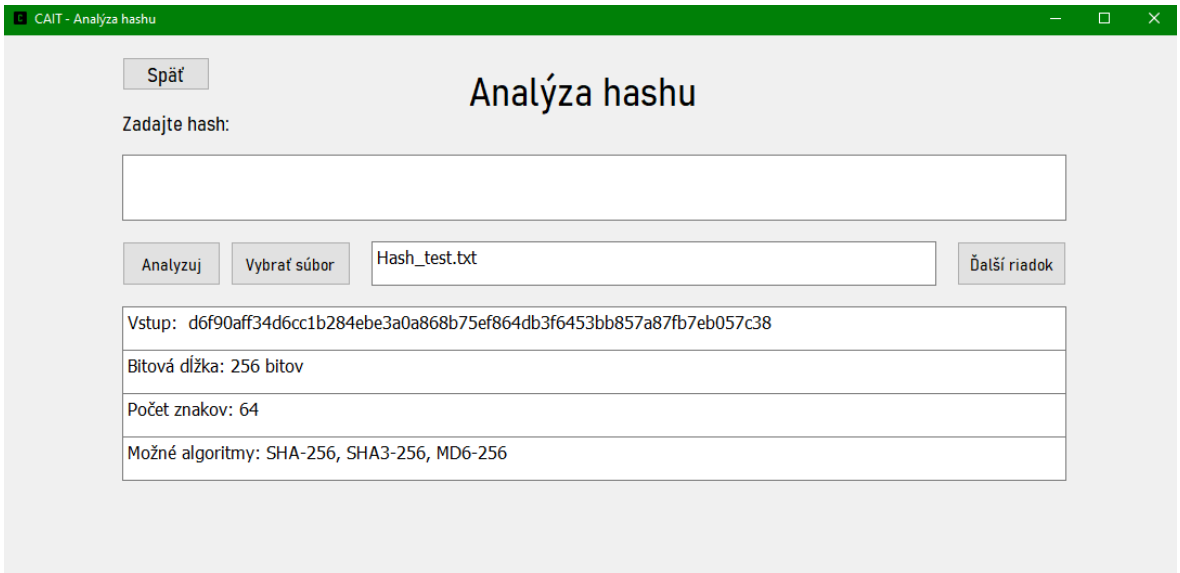


Obr. 24 Hlavné okno aplikácie

Na nasledujúcich obrázkoch je zobrazené okno, ktoré sa stará o anaýzu hashu a identifikovanie hashovacích algoritmov. Pre zadanie vstupného hashu je možné využiť dané textové pole (viď obrázok Obr. 25) alebo tlačítko *„Vybrať súbor“* pre vybratie súboru s jedným, alebo viacerými hashmi (viď obrázok Obr. 26). Po kliknutí na tlačítko *„Analyzuj“* je vykonaná analýza a zobrazené výsledky. V prípade ak súbor obsahuje viacero hashov, je možné načítavať a analyzovať ich postupne pomocou tlačítka *„Ďalší riadok“*.

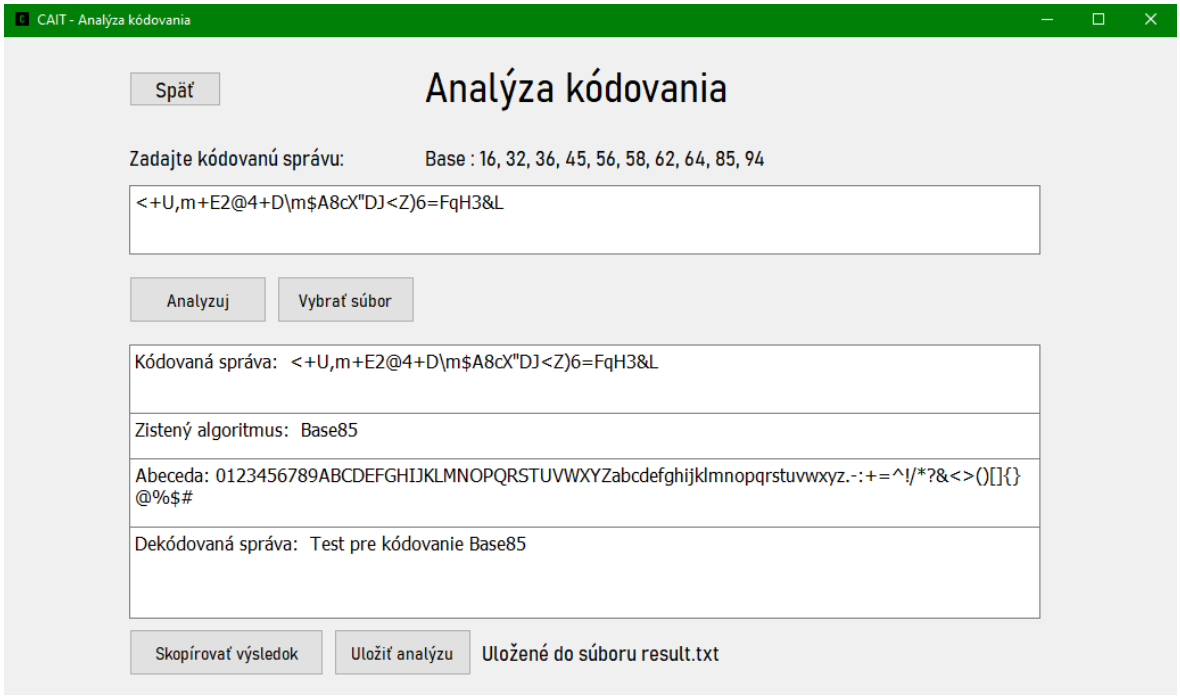


Obr. 25 Analýza hashu zadaním vstupu do textového poľa

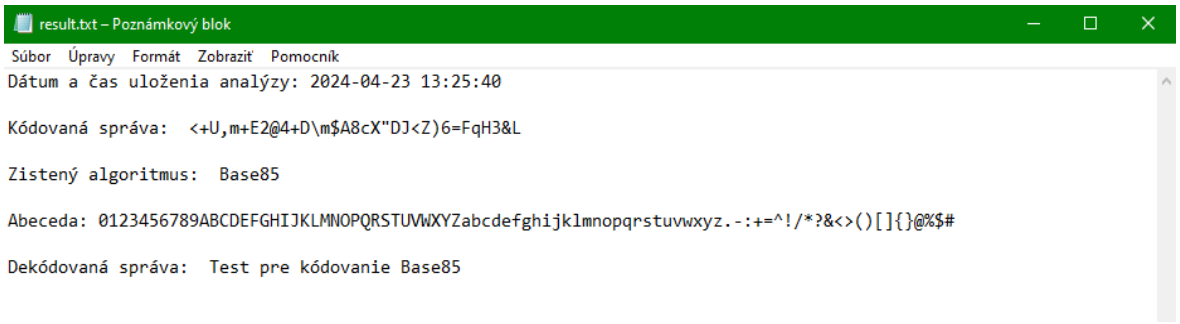


Obr. 26 Analýza hashu vybraním súboru Hash_test.txt

Na obrázku Obr. 27 je zobrazené okno, ktoré zabezpečuje analýzu kódovania a identifikáciu kódovacích algoritmov. Podobne ako v prípade analýzy hashu, je možné zadať vstup priamo do textového poľa alebo vybrať súbor. Ďalej je na obrázku Obr. 27 je zobrazená analýza kódovanej správy. Rovnako je vykonané uloženie analýzy do súboru pomocou tlačidla *„Uložiť analýzu“*. Na obrázku Obr. 28 je zobrazený obsah súboru *result.txt*, ktorý obsahuje informácie z vykonanej analýzy spolu s časovým razítkom vytvorenia súboru.

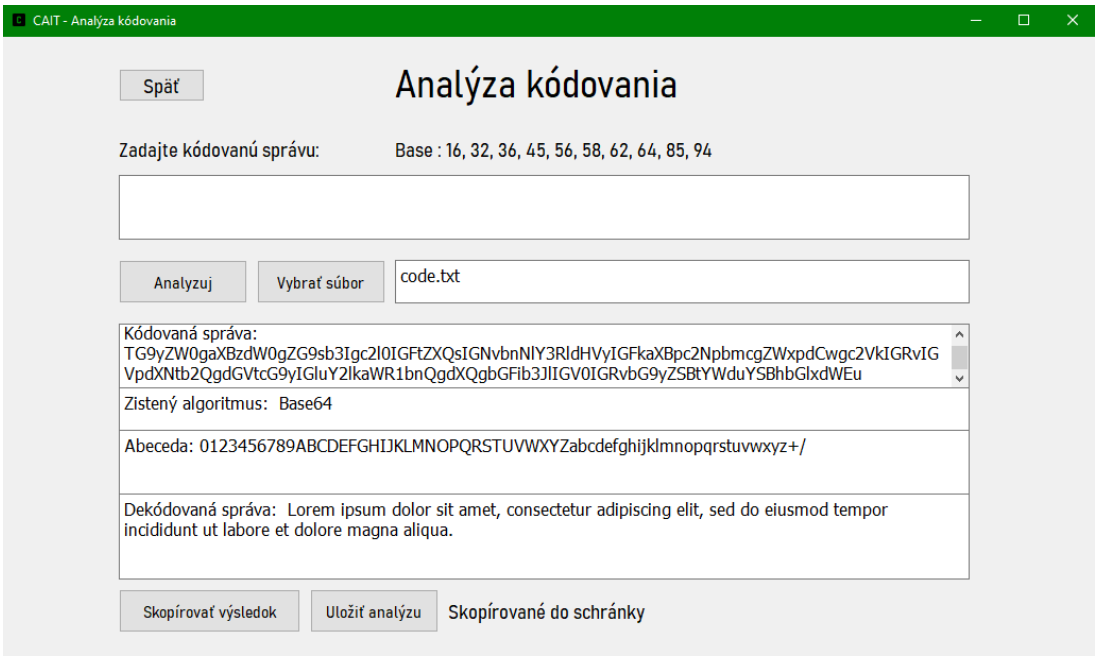


Obr. 27 Analýza kódovanej správy



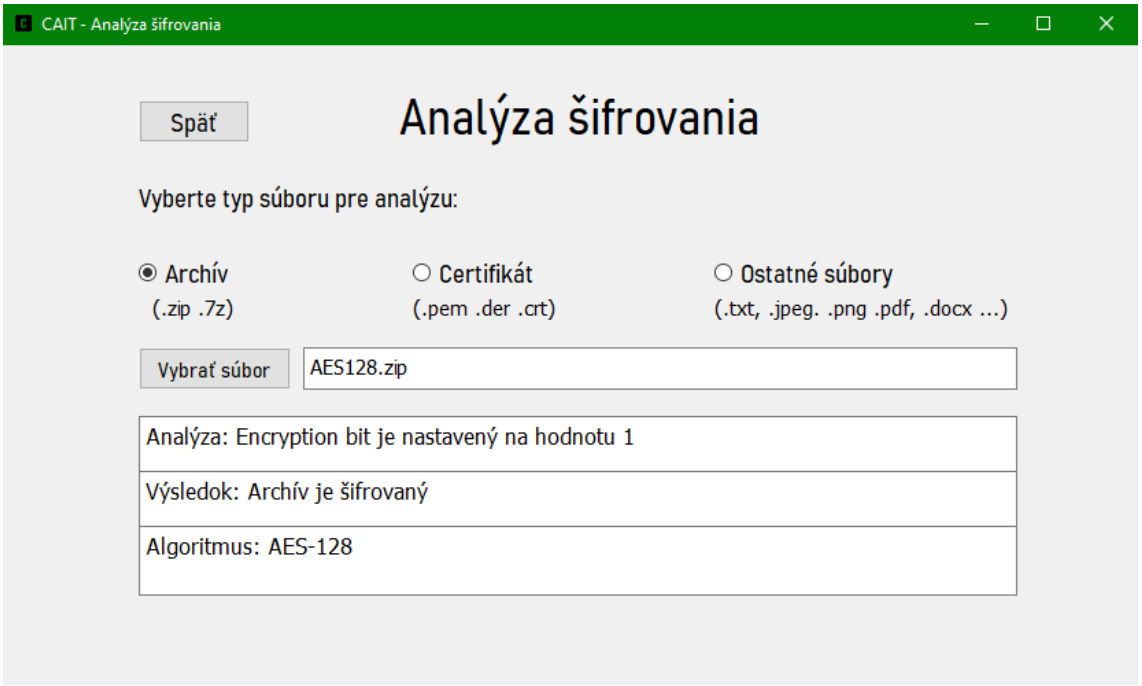
Obr. 28 Obsah súboru result.txt

Rovnako je možné aj vybrať súbor s kódovanou správou kliknutím na tlačidlo *„Vybrať súbor“*. Po vybraní súboru sa ihneď vykoná analýza a vypíšu sa jej výsledky. Výslednú dekódovanú správu je okrem uloženia možné aj skopírovať do schránky pomocou tlačidla *„Skopírovať výsledok“*. Analýza kódovanej správy zo súboru *„code.txt“* je zobrazená na obrázku Obr. 29.



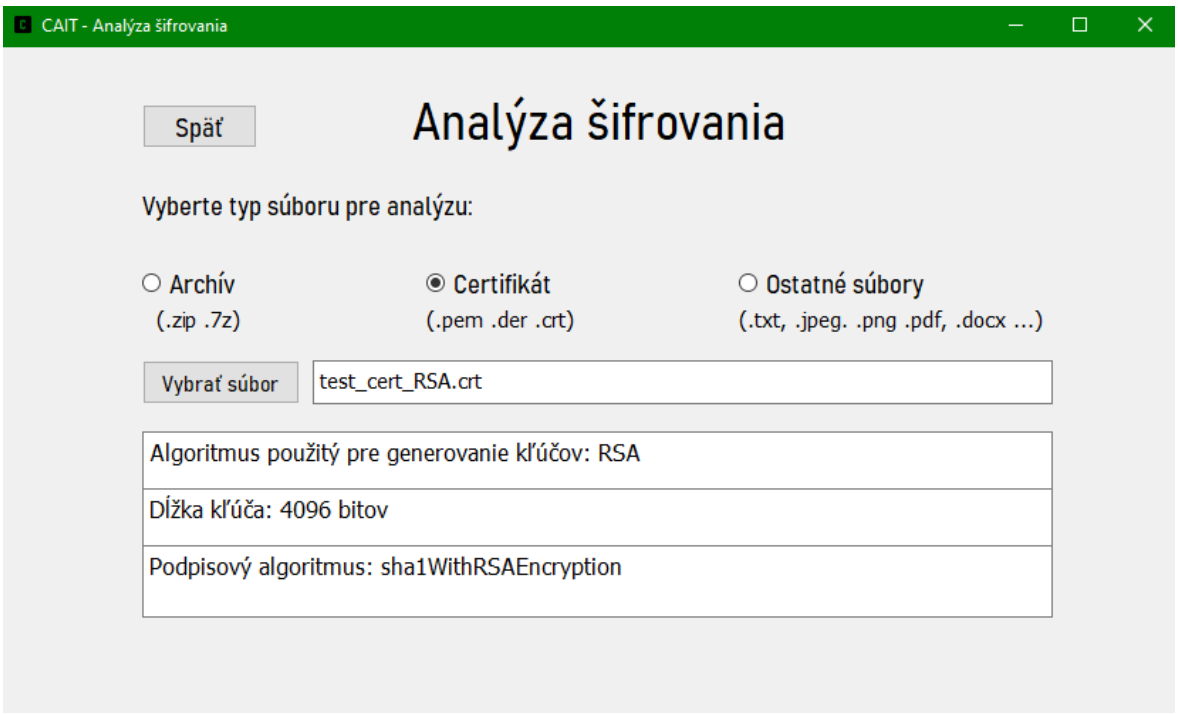
Obr. 29 Analýza kódovanej správy zo súboru code.txt

Posledné okno aplikácie sa stará o analýzu šifrovania a identifikácie kryptografických prostriedkov v rámci archívov, certifikátov alebo iných súborov. V tomto prípade je možné si vybrať druh analýzy zaškrtnutím jedného z troch rádiových tlačidiel. Po zaškrtnutí jedného z nich sa objaví tlačidlo *„Vybrať súbor“*, ktorý umožňuje vybrať konkrétny typ súboru pre danú analýzu (Napríklad v prípade zaškrtnutia tlačidla *„Archív“* dialogové okno umožní vybrať iba archívy formátu *.zip* alebo *.7z*). Na obrázku Obr. 30 je zobrazená analýza archívu *„AES128.zip“*.

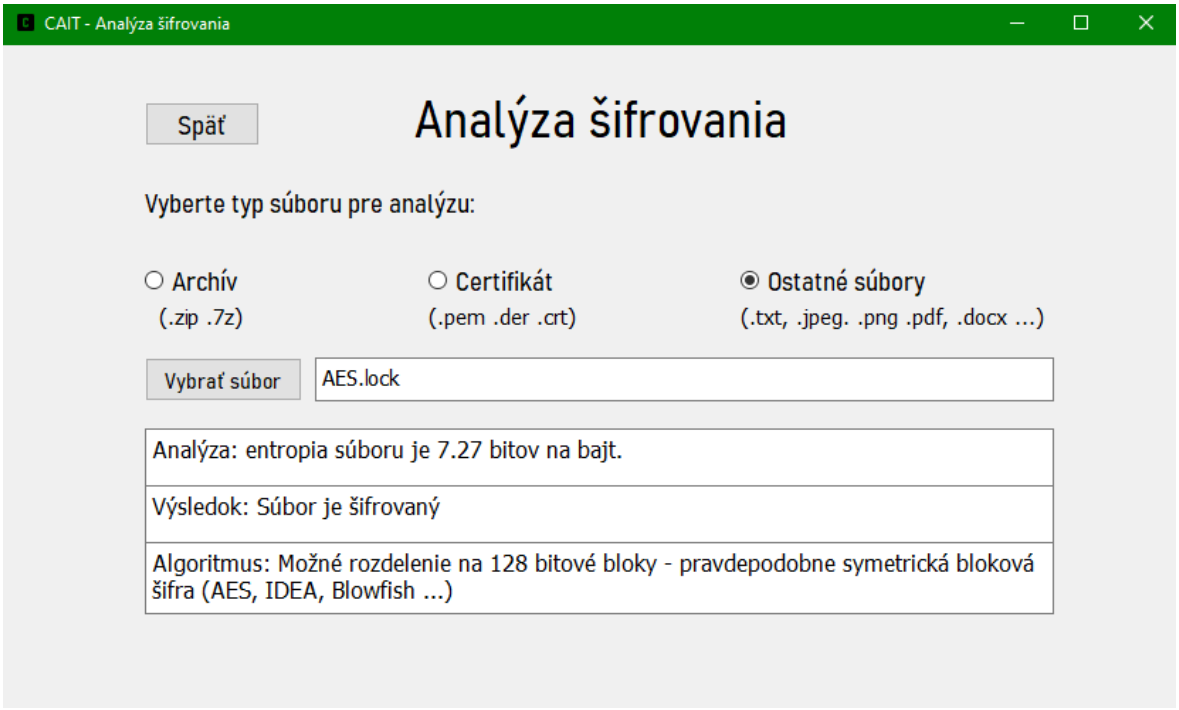


Obr. 30 Analýza šifrovania archívu AES-128.zip

Na záver je vykonané overenie funkčnosti aplikácie v prípade analýzy certifikátov a šifrovania ostatných súborov. Obrázok Obr. 31 zachytáva analýzu certifikátu „*test_cert_RSA.der*“ a obrázok Obr. 32 analýzu súboru „*AES.lock*“.



Obr. 31 Analýza certifikátu test_cert_RSA.der

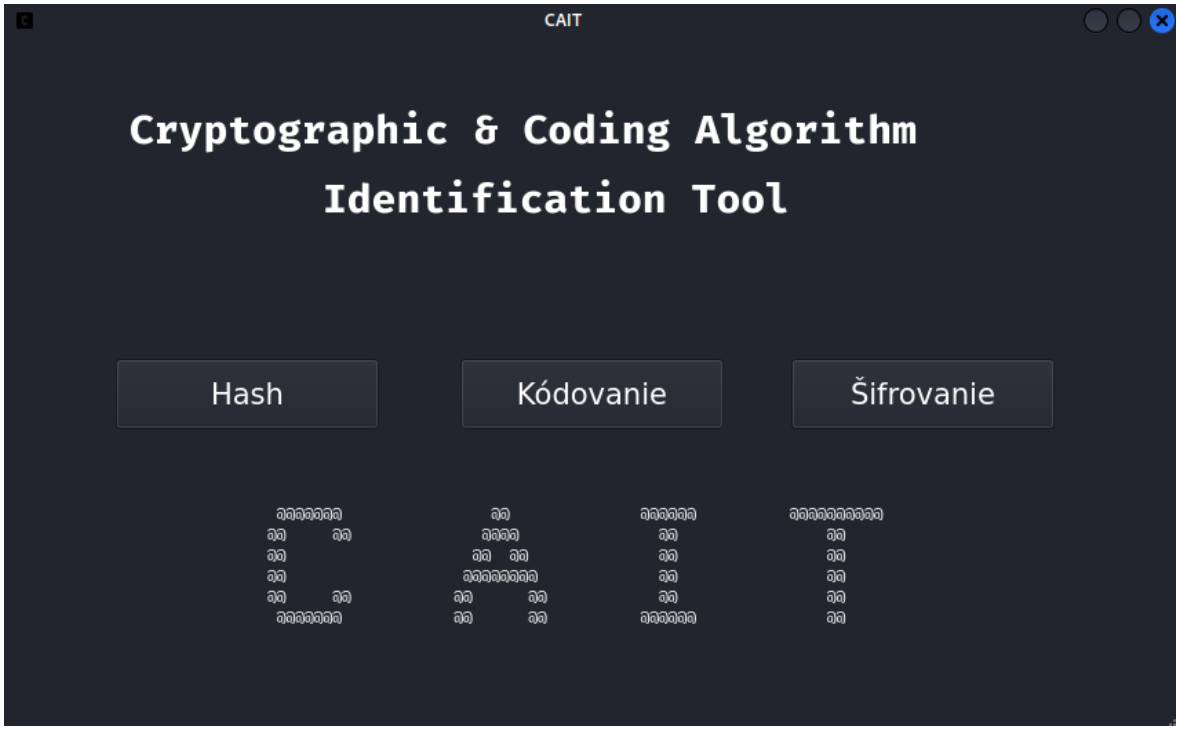


Obr. 32 Analýza šifrovania súboru AES.lock

8.3 Testovanie v operačnom systéme Linux

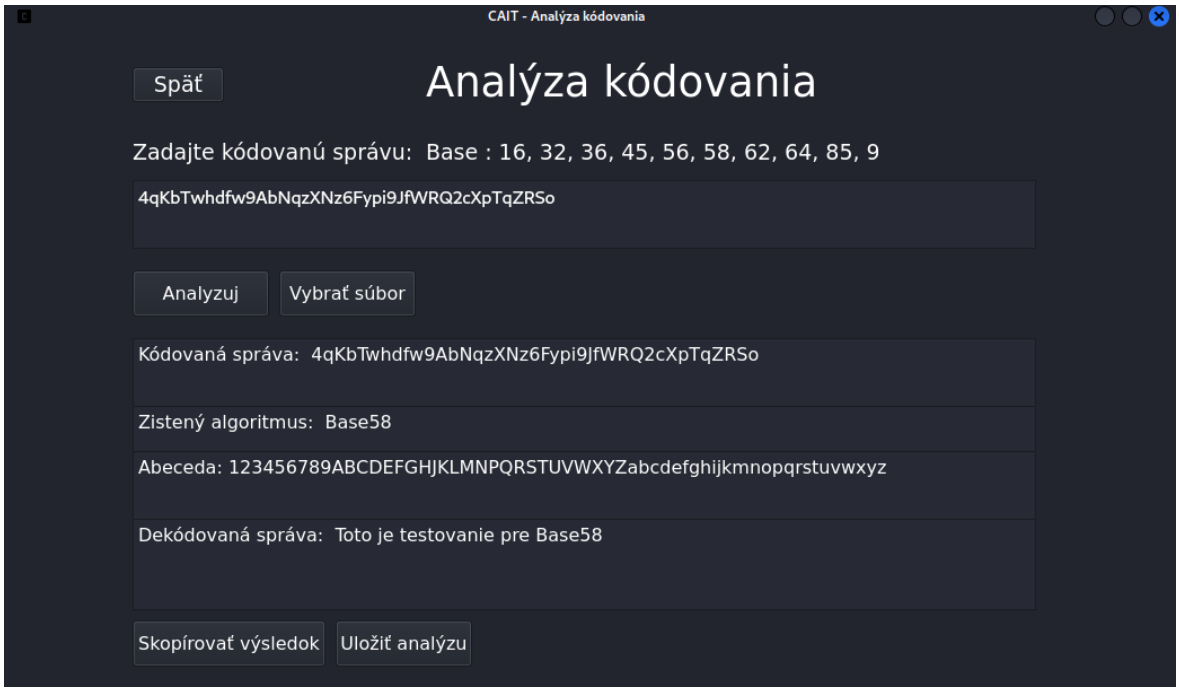
Pre overenie funkčnosti aplikácie aj mimo prostredia operačného systému Windows, bola aplikácia vyskúšaná aj na linuxovej distribúcii, konkrétne Kali Linux. Po nainštalovaní všetkých potrebných knižníc je možné spustiť aplikáciu pomocou daného príkazu, ktorý je zobrazený na obrázku Obr. 33. Po zadaní príkazu sa spustí hlavné okno aplikácie zobrazené na obrázku Obr. 34.

Obr. 33 Príkaz pre spustenie aplikácie v prostredí Kali Linux

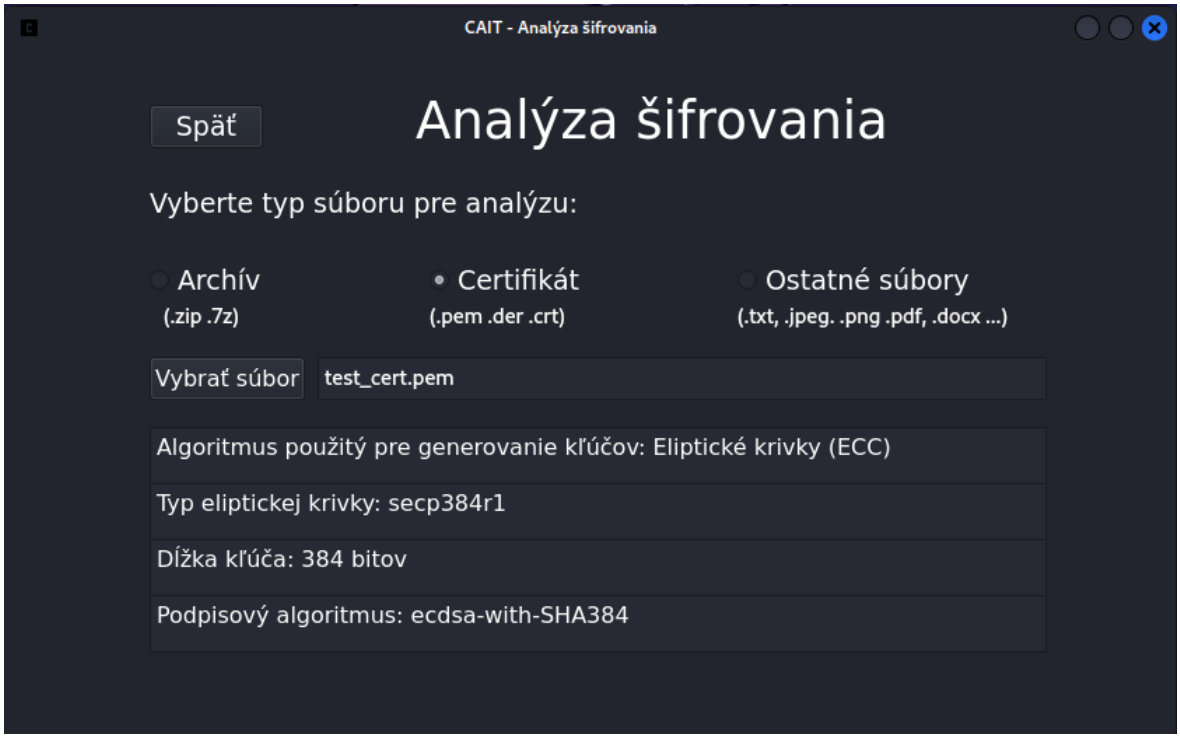


Obr. 34 Hlavné okno aplikácie v prostredí Kali Linux

Pre overenie funkčnosti aplikácie spustenej na operačnom systéme Linux, boli pre príklad otestované niektoré jej funkcionality. Analýza kódovanej správy je zobrazená na obrázku Obr. 35 a analýza certifikátu „*test_cert.crt*“ je zobrazená na obrázku Obr. 36.



Obr. 35 Analýza kódovanej správy v prostredí Kali Linux



Obr. 36 Analýza certifikátu test_cert.crt v prostredí Kali Linux

ZÁVER

Hlavným cieľom práce bolo navrhnúť a implementovať nástroj, ktorý by dokázal efektívne analyzovať dané vstupy a identifikovať použité kryptografické alebo kódovacie prostriedky.

V teoretickej časti práce boli popísané základné teoretické poznatky z oblasti kryptografie a kódovania. Ďalej bolo bližšie uvedené rozdelenie kryptografie na symetrickú a asymetrickú a vysvetlené ich princípy fungovania. Taktiež bol popísaný princíp hashovania, ktorý je súčasťou kryptografie, a spomenutý jeho význam a požiadavky. V oblasti kódovania boli vysvetlené základné pojmy spojené s technikami kódovania typu Base. Pre každú zo spomínaných oblastí bol v práci vykonaný prieskum a popis algoritmov, ktoré sú vhodné pre analýzu a identifikáciu výsledným nástrojom.

V praktickej časti boli predstavené technológie pre tvorbu nástroja, ktoré zabezpečujú funkčnosť a bezproblémový chod aplikácie na viacerých operačných systémoch, ako napríklad Windows a Linux. Ďalšie podkapitoly boli zamerané na samotnú implementáciu a rozdelenie aplikácie do jednotlivých modulov. Pre vybrané metódy boli vytvorené grafické diagramy, ktoré vizuálne znázorňujú ich fungovanie. Funkcionalita nástroja bola v závere spustená na spomínaných operačných systémoch, pričom na operačný systém Windows bol predstavený použitý postup pre tvorbu spustiteľného súboru aplikácie a na Linux príkaz, pomocou ktorého je možné jej spustenie.

V závere práce boli výsledky testovania aplikácie zaznamenané a vizualizované prostredníctvom snímkov obrazovky, ktoré dokazujú rôznorodosť, funkčnosť a schopnosť využitia tohto nástroja na rôzne analytické úkony v rámci kryptografie a kódovania.

ZOZNAM POUŽITEJ LITERATÚRY

[1] SIMMONS, Gustavus J. *Cryptology*. Online. Britannica. C2024. Dostupné z: https://www.britannica.com/topic/cryptology/. [cit. 2024-04-28].

[2] STEC, Albert. *Symmetric Cryptography vs Asymmetric Cryptography*. Online. Baeldung. 2023. Dostupné z: https://www.baeldung.com/cs/symmetric-vs-asymmetric-cryptography. [cit. 2024-04-28].

[3] *Cryptoanalysis*. Online. OWASP. C2024. Dostupné z: https://owasp.org/www-community/attacks/Cryptanalysis. [cit. 2024-04-28].

[4] LENT, Craig S. *Information and Entropy in Physical Systems*. Online. In: *Energy Limits in Computation*. Springer, 2019, s. 1-63. Dostupné z: https://doi.org/10.1007/978-3-319-93458-7_1. [cit. 2024-04-28].

[5] DAVIES, Simon R.; MACFARLANE, Richard a BUCHANAN, William J. *Comparison of Entropy Calculation Methods for Ransomware Encrypted File Identification*. Online. MDPI. 2022. Dostupné z: https://doi.org/10.3390/e24101503. [cit. 2024-04-28].

[6] NAGY, Zsolt. *Regex Quick Syntax Reference: Understanding and Using Regular Expressions*. Online. Springer, 2018. ISBN 978-1-4842-3876-9. Dostupné z: https://doi.org/10.1007/978-1-4842-3876-9. [cit. 2024-04-28].

[7] PAAR, Christof a PELZL, Jan. *Understanding Cryptography: A Textbook for Students and Practitioners*. Online. 2nd. Springer, 2010. ISBN 978-3-642-04101-3. Dostupné z: https://doi.org/10.1007/978-3-642-04101-3. [cit. 2024-04-28].

[8] GRIGUTYTÉ, Monika. *Block cipher vs stream cipher: What are the main differences?* Online. NordVPN. 2023. Dostupné z: https://nordvpn.com/blog/block-cipher-vs-stream-cipher/. [cit. 2024-04-28].

[9] *Advanced Encryption Standard (AES)*. Online. *FIPS* 197 : Federal Information Processing Standards Publication. National Institute of Standards and Technology. 2001. Dostupné z: https://doi.org/10.6028/NIST.FIPS.197-upd1. [cit. 2024-04-28].

[10] *What is the Advanced Encryption Standard (AES) ?* Online. Zenarmor. C2024. Dostupné z: https://www.zenarmor.com/docs/network-security-tutorials/what-is-advanced-encryption-standard-aes. [cit. 2024-04-28].

[11] *Simplified International Data Encryption Algorithm (IDEA)*. Online. GeeksforGeeks. 2023. Dostupné z: https://www.geeksforgeeks.org/simplified-international-data-encryption-algorithm-idea/. [cit. 2024-04-28].

[12] *Elliptic Curve Cryptography (ECC)*. Online. Certicom. C2016. Dostupné z: https://www.certicom.com/content/certicom/en/ecc.html. [cit. 2024-04-28].

[13] YAN, Yuhan. *The Overview of Elliptic Curve Cryptography* (ECC). Online. In: *Journal of Physics: Conference Series*. Volume 2386. 2022. ISSN 1742-6596. Dostupné z: https://doi.org/10.1088/1742-6596/2386/1/012019. [cit. 2024-04-28].

[14] *What is Hashing?: Best Hashing Algorithms*. Online. SignMyCode. C2024. Dostupné z: https://signmycode.com/resources/best-hashing-algorithms. [cit. 2024-04-28].

[15] *Password Storage Cheat Sheet*. Online. OWASP Cheat Sheet Series. C2024. Dostupné z: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html. [cit. 2024-04-28].

[16] *What are the requirement of hash function in Information Security?* Online. Tutorialspoint. C2024. Dostupné z: https://www.tutorialspoint.com/what-are-the-requirement-of-hash-function-in-information-security. [cit. 2024-04-28].

[17] *Applications of Hashing*. Online. GeeksforGeeks. 2023. Dostupné z: https://www.geeksforgeeks.org/applications-of-hashing/. [cit. 2024-04-28].

[18] *Secure Hash Standard (SHS)*. Online. *FIPS PUB 180-4 : Federal Information Processing Standards Publication*. National Institute of Standards and Technology. 2015. Dostupné z: http://dx.doi.org/10.6028/NIST.FIPS.180-4. [cit. 2024-04-28].

[19] *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. Online. FIPS PUB 202 : Federal Information Processing Standards Publication. National Institute of Standards and Technology. 2015. Dostupné z: https://dx.doi.org/10.6028/NIST.FIPS.202. [cit. 2024-04-28].

[20] *Cryptography Hash functions*. Online. Tutorialspoint. C2023. Dostupné z: https://www.tutorialspoint.com/cryptography/cryptography_hash_functions.htm. [cit. 2024-04-28]

[21] GRIGUTYTĖ, Monika. *What is bcrypt and how does it work?* Online. NordVPN. 2023. Dostupné z: https://nordvpn.com/blog/what-is-bcrypt/. [cit. 2024-04-28].

[22] BIRYUKOV, Alex; DINU, Daniel a KHORVATOVICH, Dmitry. *Argon2: the memory-hard function for password hashing and other applications*. Luxembourg: University of Luxembourg. Online. 2015. Dostupné z: https://www.password-hashing.net/argon2-specs.pdf. [cit. 2024-04-28].

[23] *Scrypt*. Online. Practical Cryptography for Developers. Dostupné z: https://cryptobook.nakov.com/mac-and-key-derivation/scrypt. [cit. 2024-04-28].

[24] *Example hashes*. Online. Hashcat. Dostupné z: https://hashcat.net/wiki/doku.php?id=example_hashes. [cit. 2024-04-28].

[25] ZOLA, Andrew. *Encoding and decoding*. Online. TechTarget. C2024. Dostupné z: https://www.techtarget.com/searchnetworking/definition/encoding-and-decoding. [cit. 2024-04-29].

[26] JOSEFSSON, S. *RFC 4648 : The Base16, Base32, and Base64 Data Encodings*. Online. 2006. Dostupné z: https://doi.org/10.17487/RFC4648. [cit. 2024-04-29].

[27] MOYAL, Shimon. *A Dive into Base64 and Its Significance in Web Development*. Online. 2023. Dostupné z: https://medium.com/@shimonmoyal/a-dive-into-base64-and-its-significance-in-web-development-b6bd4427f61d. [cit. 2024-04-29].

[28] *Base32 Encoder*. Online. RFC TOOLS. Dostupné z: https://www.rfctools.com/base32-encoder/. [cit. 2024-04-29].

[29] *Base 16: Cracking the Code: Base i and the World of Base 16*. Online. FasterCapital. C2024. Dostupné z: https://fastercapital.com/content/Base-16--Cracking-the-Code--Base-i-and-the-World-of-Base-16.html. [cit. 2024-04-29].

[30] *Regular Expression*. Online. Regular expressions 101. Dostupné z: https://regex101.com/. [cit. 2024-04-29].

[31] *Bcrypt_regex*. Online. GitHub Gist. C2024. Dostupné z: https://gist.github.com/b1ek/eaf427c8457de3d3ce37e2078e9d342e. [cit. 2024-04-29].

[32] *Base64 Regex*. Online. Regular expressions 101. Dostupné z: https://regex101.com/library/lXFWqM. [cit. 2024-04-29].

[33] *AES Encryption Information: Encryption Specification AE-1 and AE-2*. Online. WinZip. 2009. Dostupné z: https://www.winzip.com/en/support/aes-encryption/. [cit. 2024-04-29].

[34] *7z Format*. Online. 7-Zip. C2024. Dostupné z: https://www.7-zip.org/7z.html. [cit. 2024-04-29].

[35] *COMPONENTS OF A PKI, PART 1: DIGITAL CERTIFICATES*. Online. Ravenswood. C2024. Dostupné z: https://www.ravenswoodtechnology.com/components-of-a-pki-part-1/. [cit. 2024-04-29].

[36] *X.509 Reference*. Online. Cryptography. C2013-2024. Dostupné z: https://cryptography.io/en/latest/x509/reference/#loading-certificates. [cit. 2024-04-29].

ZOZNAM POUŽITÝCH SYMBOLOV A SKRATIEK

AES Advacned Encryption Standard

ASIC Application Specific Integrated Unit

CPU Central Processing Unit

DNS Domain Name System

ECC Elliptic Curve Cryptography

ECDH Elliptic Curve Diffie-Hellman

ECDSA Elliptic Curve Digital Signature Algorithm

FPGA Field Programmable Gate Array

GPU Graphics Processing Unit

IDEA International Data Encryption Algorithm

IPFS InterPlanetary File System

MAC Media Access Control

MD Message Digest

PGP Pretty Good Privacy

SHA Secure Hash Algorithm

TOTP Time-based One Time Password

URL Uniform Resource Locator

ZOZNAM OBRÁZKOV

ZOZNAM TABULIEK

ZOZNAM PRÍLOH

Příloha P I: Elektronická verzia diplomovej práce, zdrojové kódy aplikácie a spustiteľný súbor.

PRÍLOHA P I: ELEKTRONICKÁ VERZIA DIPLOMOVEJ PRÁCE, ZDROVÉ KÓDY APLIKÁCIE A SPUSTITEĽNÝ SÚBOR

Priložené súbory:

fulltext.pdf

/app

- CAIT.exe
- CAIT_main.py
- CAIT.ui
- CodeWindow.py
- Code.ui
- HashWindow.py
- Hash.ui
- EncryptionWindow.py
- Encryption.ui
- icon.ico