Střední průmyslová škola Třebíč

Maturitní práce

HASHTESTING

Profilová část maturitní zkoušky

Studijní obor: Informační technologie

Třída: ITA4

Školní rok: 2024/2025 Kamil Franek

Zadání práce

Střední průmyslová škola Třebíč
Manželů Curieových 734, 674 01 Třebíč

Zadání ročníkové práce

Obor studia: **18-20-M/01 Informační technologie**

Celé jméno studenta:	**Kamil Franek**		
Třída:	**ITA4**	Školní rok:	**2024/2025**
Číslo tématu:	**55**		
Název tématu:	**Hashovací algoritmy a jejich využití**		
Rozsah práce:	**15 - 25 stránek textu**		

Specifické úkoly, které tato práce řeší:

Teoreticky zpracujte problematiku hashování. S pomoci nejméně dvou hashovacích algoritmů vytvořte aplikaci na demonstraci procesu hashování. Vstupem bude textový ručně zadaný textový řetězec a soubor. Naprogramujte samotné hashovací funkce s možností volby stupně zabezpečení a délky výstupního hodnoty. Navrhněte způsob testovaní dané hashovací funkce a způsob demonstrace změny výstupní hodnoty na změně vstupní hodnoty a tyto funkčnosti naprogramujte. Aplikaci realizujte v prostředí VS, jazyk C#, sdílení a ukládání přes GitHub.

Termín odevzdání:	**28. března 2025, 23.00**
Vedoucí projektu:	**Ing. Ladislav Havlát**
Oponent:	**Ing. Drahomír Škárka**
Schválil:	**Ing. Petra Hrbáčková, ředitelka školy**

ABSTRAKT

Maturitní práce na téma hashování, jeho využití a typy hashovacích algoritmů. Zabývá se problematikou spojenou s hashováním a vysvětlení použití hashování v IT. Tento dokument popisuje použitou technologii, praktiky a vytváření samotného programu a všeho okolo. Výsledný program disponuje základními i rozšířenými funkcemi práce s hashem a soubory pro zvýšení efektivity práce. Dále disponuje okénkem pro informace, vysvětlení rozdílů mezi hashovacími algoritmy, rozdíl mezi použitím a nepoužitím soli a dalších informací za účelem zvýšení chápání daného téma pro uživatele.

KLÍČOVÁ SLOVA

Maturitní práce, Výuka, Hash, Kybernetický útok

ABSTRACT

abstrakt anglicky

KEYWORDS

klíčová slova anglicky

PODĚKOVÁNÍ

Děkuji Ing. Ladislavu Havlátu a oponentu Ing. Drahomíru Škárkovi za cenné připomínky a rady, které mi poskytli při vypracování maturitní práce.

V Třebíči dne 25. ledna 2025 podpis autora

PROHLÁŠENÍ

Prohlašuji, že jsem tuto práci vypracoval/a samostatně a uvedl/a v ní všechny prameny, literaturu a ostatní zdroje, které jsem použil/a.

V Třebíči dne 25. ledna 2025

podpis autora

Obsah

Úvod

Cílem této ročníkové práce a programu je zjednodušení práce s hashema, ukázku rozdílů mezi hashovacími algoritmy, používání soli a pepře a podrobné vysvětlení, postupné hashování pomocí mezikroků, silné a slabé stránky hashů, kde a proč se používají. Dále program obsahuje test prolomení hesla (útok hrubou silou) pomocí hrubého útoku. V dokumentu jsou popsány použité programy a technologie, jak jsou použité a proč jsou použité. V praktické části je popsána celá cesta dělání programu, hlavní problémy, trable a vysvětlení fungování celého programu s ukázkami samotného kódu, testování a různé obrázky z pracovního postupu. V závěru jsou popsány moje pocity z práce na projektu a spokojenost s finální verzí programu.

1 Teorie hashovaní

Hashování je matematický algoritmus pro převod dat do předem určitého dlouhého výstupu podle algoritmu tzv. hashovací funkce. Hashe mají několik výtečných vlastností: vstupní data můžou býti jakkoliv dlouhá, minimální změna v datech znamená velký rozdíl ve výstupech, s větší výstupní délkou se exponenciálně zmenšuje šance na stejnost výstupních hodnot při jiném vstupu a ta nejdůležitější, nedá se získat z výstupních dat vstupní data (bez použití kybernetických útoku), znamenaje, že proces je jednosměrný. Díky tomu se hash bere jako unikátní otisk vstupních dat. [1]

1.1 Používání Hashů

Hashe se používání k uschování důležitých informací (například hesel), kde pro bezpečnost nechceme dostat vstupní data zpátky, dělání kontroly a integrity dat (kontrolní součet), vytváření a ověřování elektronického podpisu (třeba pro bankovnictví nebo email ověření), hledání škodlivého malwaru antivirovým programem, k hledání úseků DNA sekvencí atd. [1]

1.2 Použité Hashe

Existuje spoustu hashovacích funkcí a každá má svoje výhody, nevýhody a využití pro jiné účely. Tady je informace pro hashovací funkce, které jsou použity v programu.

1.2.1 MD5

MD5 (Message-Digest Algorithm) pochází z rodiny „Message-Digest“ neboli algoritmus na strávení zprávy. Předchůdci MD5 jsou hashovací funkce MD2 a MD4, všechny tři vytvořeny a vydány Ronaldem Rivestem. MD2 byl vydán v roce 1989, MD4 jakožto pokračovatel v 1990 a MD5 jakožto vylepšená verze MD5 v roce 1991. MD5, na rozdíl od svých předchůdců, je docela složitý algoritmus na rozlousknutí . Používá 4 kola, místo 3 kol jako MD4, a pomocí matematických operací s maticema vypočítá výstup. Délka výstupu hashe je 128 bitů. I přes jeho používání v tehdější a dnešní době se v MD5 našla řada chyb, které by mohly být při ukládání hesel závažné. MD5 je totiž poměrně náchylný na takzvaný brute force attack, česky útok

hrubou silou. [1][2]

```
/* Round 1. */
/* Let [abcd k s i] denote the operation
     a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s). */
/* Do the following 16 operations. */
[ABCD  0  7  1]  [DABC  1 12  2]  [CDAB  2 17  3]  [BCDA  3 22  4]
[ABCD  4  7  5]  [DABC  5 12  6]  [CDAB  6 17  7]  [BCDA  7 22  8]
[ABCD  8  7  9]  [DABC  9 12 10]  [CDAB 10 17 11]  [BCDA 11 22 12]
[ABCD 12  7 13]  [DABC 13 12 14]  [CDAB 14 17 15]  [BCDA 15 22 16]
```

*Příklad 1. kola ze 4 u MD5

1.2.2 SHA1/SHA256/SHA512

SHA, Secure-Hash Algorithm neboli bezpečná hashovací funkce je další velice známý a používaná hashovací funkce. SHA se bere za nástupce MD5 s větší bezpečností a delším výstupem (SHA1 – 160 bitů. SHA256 – 256 bitů, SHA512 – 512 bitů). SHA1 byla první verze SHA vydaná v roce 1995, nepočítaje SHA-0, což byla rychle zapomenutá „před“ verze SHA-1. SHA je rodina vytvořena a zveřejněna americkým ústavem pro technologické standardy (National Institute of Standards and Technology [NIST]). [3][4]

V roce 2005 byl na SHA1 nalezen možný útok a proto v roce 2010 vyšla skupina SHA-2, což je skupina několika hashovacích funkcí, u kterých se mění délka výstupu (pro nás důležité SHA 256 a SHA512, dále se nachází SHA-224 a SHA-384. [5] Čísla na konci znamenají délku výstupu v bitech). Skupina SHA-2 se dodnes považují za bezpečné hashovací algorithmy pro integritu dat a ukládání hesel.[5]

Funkce SHA-256 se využívá ve virtuálních měnách jako třeba Bitcoin. Hlavní premise takzvaných těžiček je najít vstup zahashovaného textu pomocí SHA-256 funkce. [6]

„Google oznámil, že se mu podařilo prolomit bezpečnost hašovací funkce SHA-1. Od první publikované slabiny po úspěšný útok tak uběhlo deset let. Dva roky trvalo vědcům z CWI Institute in Amsterdam a společnosti Google, než dokončili práci na slabině a přinesli důkaz toho, že kolizní funkce existuje a významně urychluje útok.“ [7] Přikládám URL adresu projektu SHAttered: https://shattered.io/

1.2.3 RIPEMD160

RipeMD160 je hashovací funkce, která měla za účel nahradit MD4 a MD5, stejně jako SHA1. Hlavní rozdíl je, že RipeMD160 byla vyvinuta v EU jakožto součást projektu RIPE (RACE Integrity Primitives Evaluation, 1988-1992). RipeMD160 byla vytvořena Hansem Dobbertinem, Antoonem Bosselaersem a Bartem Preneelem. Spolu s RipeMD128, RipeMD256 a RipeMD320 byly vydány v roce 1996. Všechny tyto verze vstávají z originální RipeMD hashovací funkce, která byla vydána roku 1992. Dnes všechny tyto verze RipeMD nejsou doporučovány používat, i když v lepších verzích RipeMD (vydány v 1996) nebyly nalezeny žádné kolize. [8]

V originálním RipeMD z roku 1992 byla nalezena kolize v roce 2004. Pro maximální bezpečnost se může používat Hash160, což je kombinace SHA256 a RipeMD160, kde vstup prvně projde SHA256 a výsledek poté projde přes RipeMD160. Hash160 se používá u Bitcoinu jako identifikátor pro úschovu peněženky. [9] https://www.hash160.com/

1.2.4 CRC32

CRC32 je hashovací funkce z rodiny cyclic redundancy check neboli cyklická redundantní součet (CRC) je hashovací funkce zaměřená na kontrolní součet a integritu dat. Kvůli tomuto využití jsou všechny CRC velice rychlé. Rodina CRC byla zveřejněna W. Wesley Petersonem v 1961. Nejznámější a nejpoužívanější funkce z rodiny CRC jsou CRC32, která se používá v „ISO 3309 (HDLC), ANSI X3.66 (ADCCP), FIPS PUB 71, FED-STD-1003, ITU-T V.42, ISO/IEC/IEEE 802-3 (Ethernet), SATA, MPEG-2, PKZIP, Gzip, Bzip2, POSIX cksum, PNG, ZMODEM atd.“ [10], a CRC16, které se používá v Bluetooth, SD, X.25, v různých mobilních sítích, USB a mnoho dalších. [11]

Wesley Petersonovi byla za návrh CRC udělena cena „Japan Prize“ v roce 1999. [12] Co se týče hashování důležitých dat, klíčů a bezpečnostních prvků nemá CRC mnoho využití, ale jakožto funkce pro kontrolní součet jich má nespočet.

1.2.4.1 Výpočet CRC

Základ celého výpočtu CRC jsou polynomy. Každý jeden bit v určité délce může být zapsán do polynomu. Příklad (1101) může být zapsán jako $x^3 + x^2 + 1$. poslední bit reprezentuje 1, druhý bit x, třetí x^2 atd. CRC32 má až x^{31} a CRC16 až x^{15}. Tento

výpočet pro CRC32 je v mém kódu reprezentován druhým cyklem for (uint j = 8; j > 0; j--); Výpočet se prodeve tolikrát, kolik je výstupní délka v bitech (pro CRC32 to je 256, pro CRC16 to je rovno 128). Náš vytvořený polynom je poté porovnán operací XOR (Exclusive OR, které vrací 0, pokud jsou obě hodnoty stejné a 1, pokud jsou jiné).

```
string HashCRC32(string text) //text musí být ve formátu UTF-8
{
    uint[] crc32Table = new uint[256]; //Velikost (CRC32 == 32 bajtů = 256 bitů)
    const uint polynomial = 0xedb88320; //Polynom G (G jako Generátorový)

    for (uint i = 0; i < 256; i++) //předpočítání CRC tabulek
    {
        uint crc = i;
        for (uint j = 8; j > 0; j--)
        {
            if ((crc & 1) == 1) // & ==> bit AND, kontrola jestli je bit 1
            {
                crc = (crc >> 1) ^ polynomial; //^ ==> bit XOR
            }
            else
            {
                crc >>= 1; //>> ==> posun bitů (v tuhle chvíli o 1 bit do prava)
            }
        }
        crc32Table[i] = crc;
    }

    //Samotná kalkulace CRC32
    uint crcValue = 0xffffffff;
    byte[] inputBytes = System.Text.Encoding.UTF8.GetBytes(text); //Ze stringu na bajty

    foreach (byte b in inputBytes) //každý vstupní bajt projde přes CRC
    {
        byte tableIndex = (byte)((crcValue & 0xff) ^ b); //0xff ==> Hexadecimal number
        crcValue = (crcValue >> 8) ^ crc32Table[tableIndex]; //každý bajt vstupních dat projde přes CRC
    }

    crcValue = ~crcValue; //~ ==> bit NOT (negace bitů)
    return crcValue.ToString("x8"); //konvertuje na hexadecimální číslo používající malá písmenka
}
```

Další část je porovnávání vstupních bajtů s bajty v tabulce. Index tabulky je zjištěn pomocí předběžného výstupu CRC (pro první projití je do hodnoty CRC nastavena nejvyšší hodnota), operací AND (AND vrací log. 1 pouze pokud jsou oba bity log. 1) a pomocí operace XOR se vstupem. Dále se na CRC výstupu posune 8 bitů do prava, tímto se 8 bitů vpravo efektivně ztratí a provede se XOR operace s tabulkovou hodnotou na indexu předchozího výpočtu. Na závěr se celý výstup ještě neguje bit po bitu (operace NOT). [10][11]

1.3 Sůl, Pepř a jejich používání

Hashovací sůl a pepř jsou další vrstvou pro bezpečnost hashování. Sůl je náhodně vygenerována před generováním a dává se před samotnými daty. Může mít jakoukoliv

délku, záleží na správci, který bude hodnotu ukládat. Díky soli se chráníme před takzvanými rainbow tables útoky a útoky hrubou silou.

Pepř je podobný jako sůl, jenže je většinou krátký, dává se na konec dat místo před data a nikde se neukládá. To znamená, že musíme provést všechny možné kombinace hashů a porovnávat výsledné hashe abychom zjistili shodu. Jediná nevýhoda pepře je, že musíme hashovací a porovnávací proces dělat několikrát, což výrazně zvýší prodlevu. V praxi se používají sůl i pepř pro maximální ochranu hesel. [13]

Příklad: Máme heslo "TestingPassword123". Díky soli se před heslo vygeneruje sůl "a0_X", která je někde v tabulce uložena. Před heslo se vygeneruje pepř o délce jednoho ASCII znaku. To znamená že při každém pokusu o přihlášení se před heslo dá sůl a za heslo se postupně zkouší "000", poté "001", "010" a tak dale, dokud se nevyzkouší všechny kombinace pepře. Pokud žádný z těchto pepřových kombinací nevýjde, heslo je zadáno špatně.

Všechny hashe jsou šifrované v hashovací funkci MD5.
(bez soli) TestingPassword123 == 1d898af5dbe7c9e07fc473e248f623a1
(se solí) a0_XTestingPassword123 == 5a6c1e14762baf73406b7267a8afae88
(sůl I pepř) a0_XTestingPassword123C == d7d0d822ce9faea482a0c5ae372d0ed0

1.4 Kybernetické útoky na hashe

Hashe dokážou zpracovat jakékoliv množství dat a vrátit jenom určitou délku, to ovšem znamená menší problémy. Různé vstupní data mohou vracet stejnou hodnotu hashe, což v případě, že používáme hashe pro ukládání hesel znamená velký bezpečnostní problém. Šance kdy se něco takového může stát je závislá na délce výstupního hashe, proto se doporučuje používat delší a bezpečnější hashe pro ukládání důležitých dat (jako třeba hesel), jako třeba SHA-256 či SHA-512.

Pravděpodobnost si můžeme sami vypočítat pomocí jednoduchého zvorečku. Počet všech kombinací u hashovací funkce je rovna $2^{počet\ bitů\ na\ výstupu}$. To znamená že CRC32 má šanci 1 ku 2^{32}(skoro 4.3 miliardy kombinací) neboli 0.0000000233%. To se může zdát jako velice malá šance, jenomže tohle je šance jenom mezi 2 hashemy. Když započítáme šanci každého s každým (použití takzvaného narozeninového

paradox), když hledáme čistě jenom kolize, počet všech kombinací se nám sníží na $2^{\frac{počet\ počet\ bitů\ na\ výstupu}{2}}$.

„Narozeninový paradox nám říká, že pokud máme v místnosti 23 lidí, existuje přibližně padesátiprocentní šance, že se dva z nich narodili ve stejný den. Toto je velice důležitý fakt, který nám dává dolní ohraničení pro délku hashe produkovaného dobrou kryptografickou hashovací funkcí.“ „Pokud budeme mít například 40-bitovou zprávu, abychom našli kolizi s pravděpodobností 0.5, potřebujeme pouze 2^20 náhodných hashů.“ [14, strana 2]

Hashovací funkce	Velikost hashe (bit)	Počet možných kombinací (zaokrouhleno)	50% šance při hledání kolize (zaokrouhleno)
MD5	128	3,40e+38	18 446 744 073 709 551 616
SHA-1	160	1,46e+48	1 208 925 819 614 629 174 706 176
SHA256	256	1,15e+77	3,40e+38
SHA512	512	1,346e+154	1,15e+77
CRC32	32	4 294 967 296	65 536

1.4.1 Pravděpodobnost kolize (Narozeninový paradox)

https://www.youtube.com/watch?v=yQ1pGhMRLKI

Pokud ovšem hashujeme více než dva vstupy, například při hledání kolize, pravděpodobnost kolize vzrůstá. Při více projetí se totiž pravděpodobnost kolize řídí tzv. „narozeninovým paradoxem“. Pravděpodobnost, že mezi vygenerovanými hashy alespoň jednou nastane kolize lze vypočítat podle vzorce pro kombinatoriku:

$$p = 1 - \frac{N!}{(N-k)!\,N^k}$$

Tento vzorec je velice přesný, ovšem pro počítání skoro až nereálný, jelikož při N = 2^{160} jako to je u RipeMD-160 naprosto vyloučeno. Naštěstí si můžeme vzoreček poupravit do mnohem lepší formy.

$$p \geq 1 - e^{\frac{-k*(k-1)}{2N}}$$

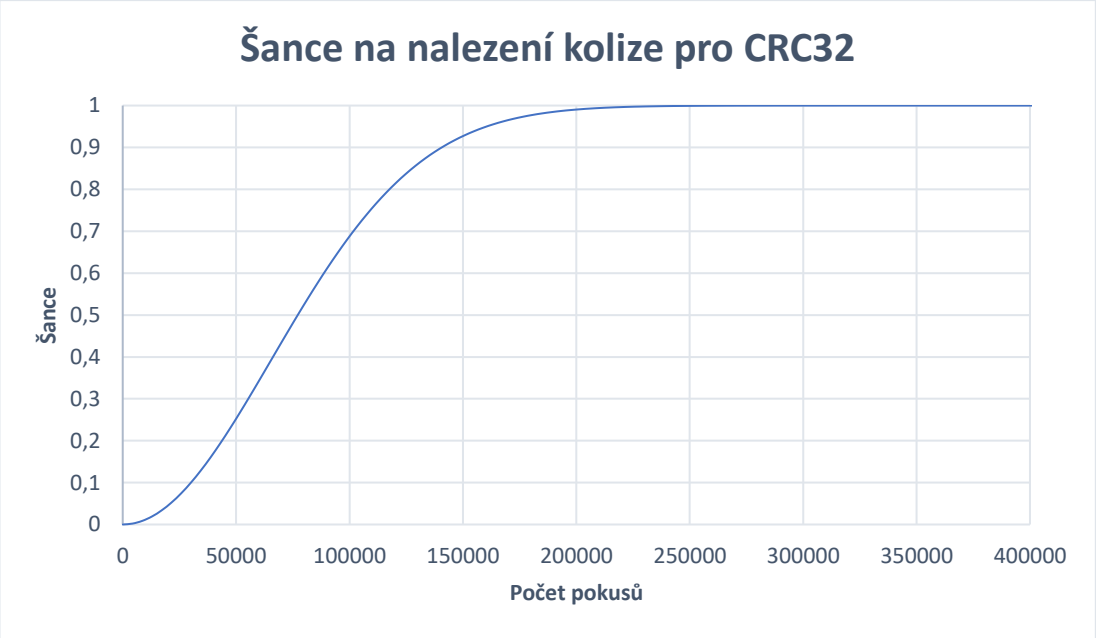
Tato upravená verze není až tak přesná, ale dá se mnohem snadněji vypočítat. [15, strana 11 a 12]

k = počet vygenerovaných hashů

e = eulerovo číslo (přibližně 2.718)

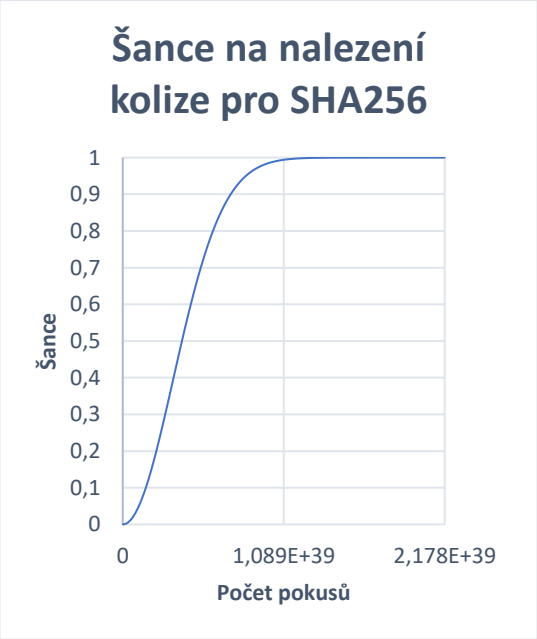
N = počet možných hashů ($2^{počet\ bitů\ na\ výstupu\ hashe}$)

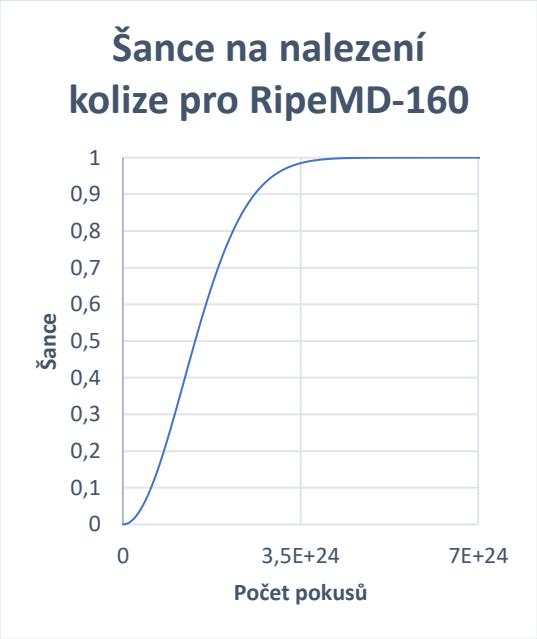
P ≥ zaokrouhlená šance na kolizi (mezi 0 a 1 / 0% a 100%)



Jak můžeme vidět, hledání kolizí je exponenciální. Díky narozeninovému paradoxu se nám drasticky zmenšuje počet projití, které je potřeba na najití jakékoliv jedné kolize. Pokud chceme najít pravděpodobnost kolize v určité šanci, třeba na 23 % (tedy p=0.23), můžeme si poupravit naši rovnici a získat vzorec $n = \sqrt{2N * \ln\frac{1}{1-P}}$, (ln je přirozený logaritmus neboli logaritmus o základu e) ze které po dosazení vznikne $n = \sqrt{2(2^{32}) * \ln\frac{1}{1-0.23}}$, n ≐ 47 382. Po projití 47 382 hashů máme 23% šanci na naleznutí kolize.

Tento způsob výpočtu má dva problémy. Problém jedna je, že $P \neq 1.00$ (100%), jelikož bychom se dostali k dělení 0 v jmenovateli, $\ln\frac{1}{1-1}$ ==> $\ln\frac{1}{0}$, proto musíme dosadit co nejbližší číslo menší než 1, což je 0.99 periodický. Další problém je zaokrouhlování, protože pomocí jedné metody ($2^{\frac{počet\ počet\ bitů\ na\ výstupu}{2}}$) nám vychází, že 50 % na najití kolize je zaokrouhleně 65 tisíc, jenže pomocí našeho upraveného vzorečku pro p nám vychází zaokrouhleně 74 tisíc (tento útvar jde vidět v grafu).





Tady dokazuji, že exponenciální rovnice je pro všechny hashovací funkce stejná. Konce grafů ovšem neukazují 100% úspěšnost, jelikož 100% úspěšnost lze získat pouze zkoušením všech možných kombinací.

1.5 Útoky

Jak už bylo zmíněno, na hashe a hlavně hesla existuje pár specifických útoků. Většinou jsou útoky zamýšlené na získání hesla či nějakého klíče či certifikátu, jako to používají cookies (sušenky) na webových prohlížečích. [16] Mezi nejrozšířenější jsou duhové tabulky a slovníkové útoky, útok hrubou silou a pass the hash.

1.5.1 Duhové tabulky a slovníkové útoky

Slovníkový útok a útok pomocí duhové tabulky můžou na první pohled vypadat jako jeden a ten samý útok, jenže tady je jeden hlavní rozdíl. Oba útoky jsou nějaká databáze potenciálních hesel, jediný rozdíl je, že duhová tabulka vyhledává už

zahešovaný výstup, nikoliv jeho vstup. Vstup si je schopná odvodit z nalezeného hashe. Naopak slovníkový útok se snaží najít počátek hashe.

Slovníkový útok se použije v případě, kdy nemáme přístup k databázi se zahešovanými hesly. Nádherný příklad je přihlášení. Zkoušíme všechna možná hesla ze seznamu, dokud nás jedno nedostane dovnitř. Pokud nenajdeme žádnou shodu, můžeme začít útok hrubou silou. Výhoda slovníkového útoku je, že nezáleží na použití soli, jediné, na čem záleží je velikost slovníku a čas. Jeden nádherný slovník má ve svém základě Kali Linux s názvem rockyou.txt, který obsahuje přes 12 milionů často používaných hesel.

Útok s duhovými tabulkami použijeme, když jsme se nějakým způsobem dostali k už zaheshovanému heslu. Díky délce hashe můžeme docela dobře zjistit použitou funkci. Každá funkce má svoji vlastní tabulku. Místo abychom jako u slovníkového útoku hledali počátek, hledáme radši shodu ve finálním hashy. Jakmile ho najdeme, tak s ním máme i originální vstup. Největší obtíže tomuto stylu útoku dělá používání soli a pepře při hashování vstupu. Jelikož skladování jak vstupu, tak výstupu může být u nějakých hashovacích funkcí náročné na místo (SHA512 má 512 bitů na heslo bez vstupu => pro 10 mil. hesel nám jenom hash zabírá 640 MB).

Vlastnosti	**Slovníkový útok**	**Útok přes duhové tabulky**
Rychlost hashování	Záleží na použité funkci	Předhashováno
Náročnost na uložiště	Malá (akorát vstup)	Velká (vstup i výstup)
Použitelnost	Platí na všechny funkce	Pro specifickou funkci
Rychlost	Pomalá (hashování)	Rychlá (hledání shody)
Efektivita	Záleží akorát na délce vstupu	Samostatně bezbranný proti solení/pepře

1.5.2 Útok hrubou silou

Útok hrubou silou (brute force) je nejzákladnější útok možný. Zkoušení hesel, dokud se nějakému nezadaří. Dal by se chápat, jako poslední možnost, když selžou všechny ostatní útoky. Fungování je jednoduché, začneme od nejmenšího možného znaku a postupně přidáváme, dokud se to jednoho dne nepovede. Záleží na délce hesla a na použitých znacích. Heslo může být 8 míst dlouhé, ale pokud víme, že používá pouze číslice, tak máme velikou výhodu. Vzoreček pro vypočítání všech možných kombinací je $Počet\ možných\ znaků^{Délka\ hesla}$. [17] V našem případě to je $10^8 = 100$ milionů možných kombinací. To možná zní jako hodně, ale když jedna grafická karta RTX 4090 dokáže zvládnout 200 tisíc hashů za sekundu BCryptu[1] a 300 milionů hashů NTLM [2] [17], tak 100 milionu najednou nezní tak krásně.

Při použití všech doporučených znaků pro heslo, což je malá písmena (26 znaků), velká písmena (26 znaků), číslice (10 znaků) a speciální znaky (33 znaků*), tak máme dohromady 95 možných použitelných znaků. Minimální doporučená délka je 7 znaků, takže počet kombinací je 95^7, což je skoro 70 bilión kombinací, což může jedné RTX 4090 trvat při použití MD5 necelých 5 dní. [19, řádek 41] Můžeme zvýšit velikost hesla, ale útočník může zvýšit počet grafik. Naštěstí 15 znaků dlouhé heslo pořád nezvládne ani 10 vysoce výkonných grafických karet zvládnout za skoro 300 miliard let. [17]

1.5.3 Pass the Hash

Hlavní důvod proč se hashování používá pro hesla je zabránit tomu, aby si někdo při komunikaci mohl jen tak vzít packet a přečíst si naše heslo. Když by někdo odposlouchával zaheshované heslo, tak musí pomocí dalšího útoku (například

[1] BCrypt je další hashovací funkce, která je ovšem značně pomalá. V roce vydání (1976) dokázala funkce zaheshovat skoro 4 hesla za sekundu. Vytvořena Nielsem Provosem a Davidem Mazièresem. [19]

[2] Windows New Technology LAN Manager (NTLM) je bezpečnostní protokol, který slouží k ověřování identity uživatelů a integritě dat. NTLM umožňuje jednotné přihlašování **(SSO).** NTLM Používá MD4 a DES šifrovací funkci. [20]

slovníkového) zjistit originální vstup. To, jak už určitě víme trvá spousty a spousty času a je díky tomu mnohem efektivnější dostat heslo jiným způsobem (třeba sociální inženýrství). Když ale na server přichází pouze zahashované heslo, a ne doopravdy heslo, proč bychom nemohli na daný server prostě poslat náš hash. O tomto je přesně pass the hash útok. Získat hash oběti je mnohem jednodušší než získat její heslo. Nejúčinnější je tento útok pro NTLM. Originální útok je od Paula Ashtona a byl zveřejněn roku 1997. Dokonce existuje i „toolkit“ pro zkoušení tohoto útoku přímo ve Windows či Kali Linuxu (Windows: https://github.com/byt3bl33d3r/pth-toolkit) (Kali: https://www.kali.org/tools/passing-the-hash/). [21] [22]

2 Použité programy a technologie

Každý správný projekt potřebuje použití několika programů či stránek pro zlepšení práce na projektu. Každý použitý program či stránka jsou popsané k čemu slouží, teorii, popřípadě vysvětlení jak fungují a proč jsem je použil.

2.1 C# a .NET

„Jazyk C# je multiplatformní jazyk pro obecné účely, který vývojářům umožňuje produktivní práci při psaní vysoce výkonného kódu. S miliony vývojářů je jazyk C# nejoblíbenějším jazykem .NET. Jazyk C# má širokou podporu v ekosystému a všech úlohách .NET. Na základě objektově orientovaných principů zahrnuje mnoho funkcí z jiných paradigmat, nikoli z nejméně funkčního programování. Funkce nízké úrovně podporují scénáře vysoké efektivity bez psaní nebezpečného kódu. Většina modulů runtime a knihoven .NET je napsaná v jazyce C# a pokroky v jazyce C# často využívají všechny vývojáře .NET.“ [23]

Programovací jazyk je základ všeho a jelikož jsem chtěl mít jistotu, tak jsem si vybral možnost, kterou velice dobře znám. Visual Studio, .NET framework pro Windows a s tím i spojený jazyk C#.

2.2 Visual Studio 2022

Hlavní důvod vybrání si Visual Studia a .NET frameworku je jednoduchost k přidání knihoven a manipulace s nimi. Hashování je provedeno přes „System.Security.Cryptography“ knihovnu, která je v základu s .NET frameworkem. Tuto knihovnu používám pro všechny hashe použité v programu, až na CRC32. Dále poskytuje spousty UI elementů a práci s formulářem, jednoduché přidání unit-testů, chybové hlášení, krokování programu, statistika využití CPU a RAM a mnoho dalšího.

2.3 Visual Code

Visual Code je univerzální editor kódu. Je nejlepší s použitím jazyka Python či HMTL a PHP, ale díky jeho univerzálnosti není žádný problém dělat v C#. Jediné co stačí je doinstalovat si rozšíření a to vyžaduje jedno zmáčknutí tlačítka. Při používání Visual Studia není potřeba používat Visual Code, ale líbí se mi jednoduchost programu.

Použil jsem program při vytváření dodatečných skriptů (jako například hasher.cs nebo settings.cs), které jsem poté naimportoval do Visual Studia.

2.4 Git/Github a Github Desktop

„Git je verzovací systém, pomocí kterého ukládáte své projekty a veškeré jejich verze. Je to distribuovaný systém správy verzí. To znamená, že k celému kódu i jeho historii se vývojář dostane z jakéhokoliv počítače.[3] Většina operací, které se s kódem provádějí, se dějí lokálně na disku pomocí příkazového řádku. Pokud ale chcete Git sdílet s kolegy, probíhá spolupráce téměř vždy přes centrální server nebo úložiště.“ [24]

Github je webová stránka, kde se dají procházet a ukládat různé Git projekty. Github nabízí spousty služeb, článků a dokonce nabízí i předplatné pro ještě více funkcí. Jedna z jejich nejnovějších zaměření je Copilot, což je umělá inteligence (AI), která má dělat programování o něco jednodušší. Nejdůležitější věc je, že většina hlavních funkcí je zadarmo, a to i s uložištěm (byť trochu malým).

„Díky tomuto systému je spolupráce na projektu bezpečnější a jednodušší. Možná i to je důvod, proč **Git používá více než 87 % vývojářů**.“ [24]
Github desktop je počítačová aplikace, která funguje stejně jako příkazový řádek v Git, akorát místo CLI [4]to je GUI [5]a dělá to Git uživatelsky přívětivé.

2.5 Unit-testy

Unit testing je testování nějaké části kódu či softwaru, jestli funguje tak, jak by měl. Patří sem nástroje pro unit-testing, různé metody a cokoliv spojené s ověřováním správného chodu. Kód se ověřuje po co možná nejmenších částech (units), aby se dalo jednoduše poznat, kde je problém. [25] V mém případě to znamená ověřování správného chodu metod jako třeba metoda „Hash“ v „hasher.cs“.

---

[3] Nejdůležitější funkce pro mě jako studenta, protože pracuji jak z domu, tak ze školy
[4] CLI je zkratka pro Command Line Interface (Rozhraní příkazového řádku)
[5] GUI je zkratka pro Graphical User Interface (Grafické uživatelské rozhraní)

Existuje i test-driven development (TDD) (Vývoj řízený testy v češtině), který spočívá na systému právě unit-testů. Před prací na programu si prvně připravíme unit-testy, které nám říkají, jak se má program chovat, a podle těchto testů pak píšeme program. [26]

```
1   using Microsoft.VisualStudio.TestTools.UnitTesting;
2   using HashTester;
3   using static HashTester.Hasher;
4   using static HashTester.Settings;
5   using System.IO;
6   using System;
7   using System.Runtime;
8   namespace UnitTest
9   {
10
11      [TestClass]
            0 references
12      public class ClassHasher
13      {
                24 references
14          Hasher hasher = new Hasher();
                6 references
15          string text01 = "HashTester12345";
                6 references
16          string text02 = "1234567890";
19          //MD5
20          [TestMethod]
                0 references
21          public void HashMD5Test01()
22          {
23              string test = hasher.Hash(text01, Hasher.HashingAlgorithm.MD5);
24              Assert.AreEqual("a51abed98e61011ef1162c93d7f8bd33", test);
25          }
26          [TestMethod]
                0 references
27          public void HashMD5Test02()
28          {
29              string test = hasher.Hash(text02, Hasher.HashingAlgorithm.MD5);
30              Assert.AreEqual("e807f1fcf82d132f9bb018ca6738a19f", test);
31          }
```

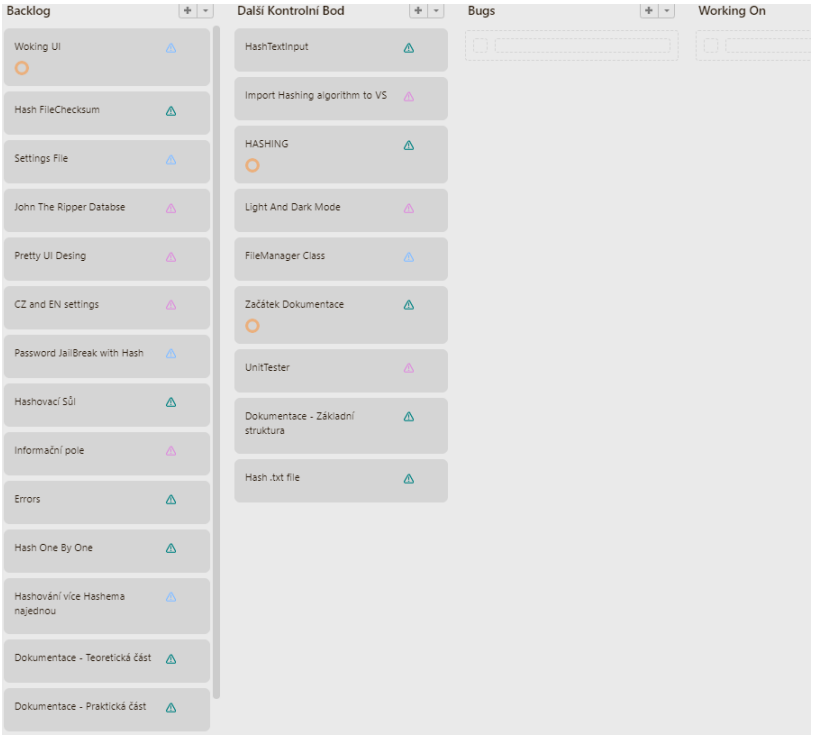
HashMD5Test01 < 1 ms
HashMD5Test02 < 1 ms

Příklad jedno z testů.

2.6 Freelo.io

„Freelo je online aplikace s cílem usnadnit řízení projektů a zvýšit efektivitu podnikatelů a firem. Řadí se mezi SaaS[6] aplikace a v červnu 2021 překročilo hranici 50 000 uživatelů.“ „Mezi hlavní funkce Freela patří Projekty, To-Do listy, Diskuse, Kalendář a Finance. Jejich cílem je usnadnění finanční správy projektu.“ [27]

Freelo je založeno na agilním projektovém řízení Kanban[7]. Kanban je založen na systému výroby Just In Time[8] (JIT), která byla vymyšlena v japonské automobilové firmě Toyota. Zakladatel je Taiičiho Óno. Kanban zjednodušuje práci několika lidí a její management. Díky Kanbanu jde krásně vidět co kdo dělá, co bude dělat a co se musí ještě udělat. V mém případě Freelo používám jako to-do list[9]. [29]



Příklad mého použití Freela na začátku projektu.

[6] SaaS je zkratka pro **software jako služba**
[7] Kanban (看板) z japonštiny znamená cedule či tabule
[8] Just In Time (právě v čas) „umožňuje podniku vyrábět výrobky v určeném množství a určeném čase dle požadavků zákazníka.“ [28]
[9] To-Do list znamená list věcí, které se ještě musí udělat

2.7 Microsoft Word

Microsoft Word je textový editor vyvíjen společností Microsoft. Celá písemná práce byla napsaná v tomto editoru. Word nabízí spousty různých funkcí pro editování textu, od vložení vzorců, referencí, generování obsahů, stylování, poznámky pod čarou, tabulky, odkazů, číslování řádků a mnoho dalšího. Program se může považovat za standardní textový editor v dnešní době. Word je součástí balíčku Microsoft Office, který má jak placený, tak zdarma plán. Word je zdarma pro studenty, online s účtem Microsoft na office.com, na mobilních zařízeních a ve zkušební verzi Office 365. [30]

2.8 Wayback Machine

Wayback Machine je digitální archiv, který je součástí Internet Archive. Internet Archive je nezisková organizace, původem z USA, která se snaží zachovat co nejvíce z internetu. Od webových stránek, po videa, audia, obrázky, programy a další. V dnešní době mají uloženo přes 960 miliard stránek v rozmezí 1996 až konce 2024. [31] Nejlepší je, že pro uživatele je všechno k podívání zadarmo bez potřeby účtu. U webových stránek stačí mít pouze URL[10] adresu. Záznamů může být několik a člověk si může vybrat. Odkaz na Internet Archive https://archive.org a Wayback Machine https://web.archive.org

Já osobně jsem používal Wayback Machine při dělání citací, protože většina internetových stránek nemají veřejný datum vzniku. Když je internetová stránka uložena v Internet Archive, znamená to, že v té době už existovala. Není to nejpřesnější datum, ale je to lepší než nic.

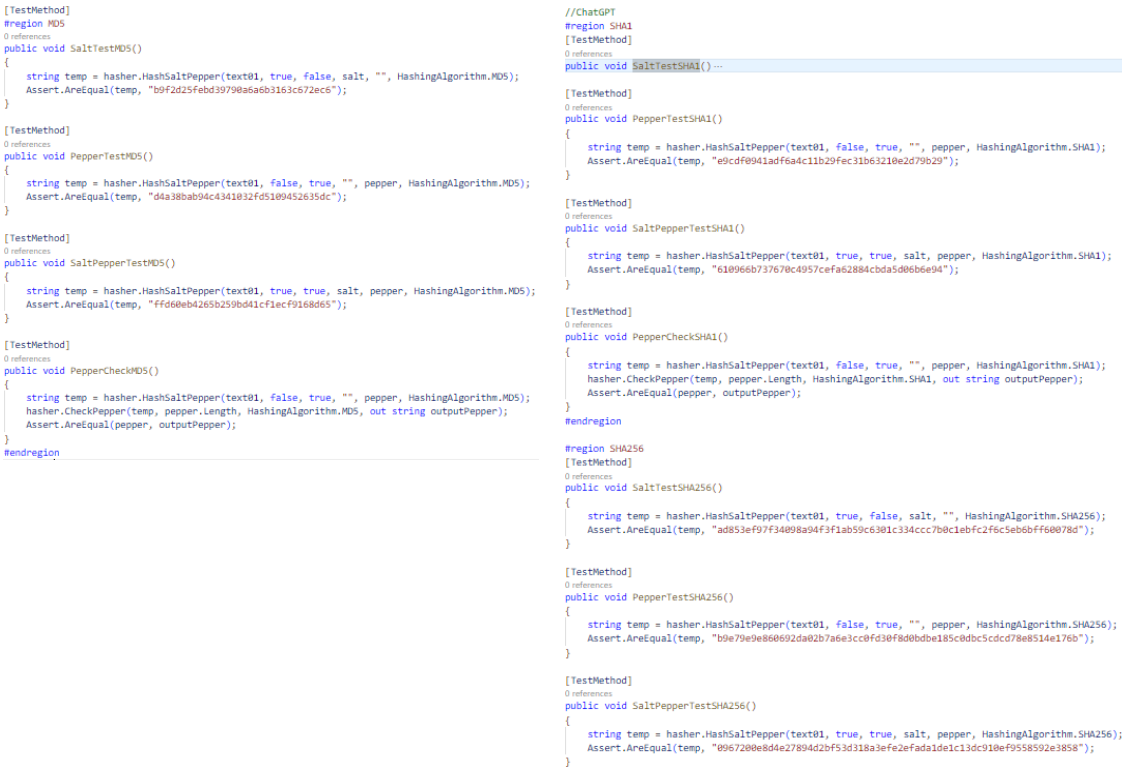
2.9 ChatGPT (AI)

„ChatGPT, což je zkratka pro "Chat Generative Pre-trained Transformer," je pokročilý model umělé inteligence vyvinutý společností OpenAI. Tento model je navržen tak, aby generoval lidsky srozumitelný a koherentní text na základě zadaných vstupů. Je

[10] Uniform Resource Locator (URL) znamená jednoznačné určení zdroje a používají to webové stránky jakožto doménovou adresu (příklad je seznam.cz)

postaven na architektuře GPT-4, která je jednou z nejmodernějších a nejpokročilejších v oblasti přirozeného jazyka. [32]

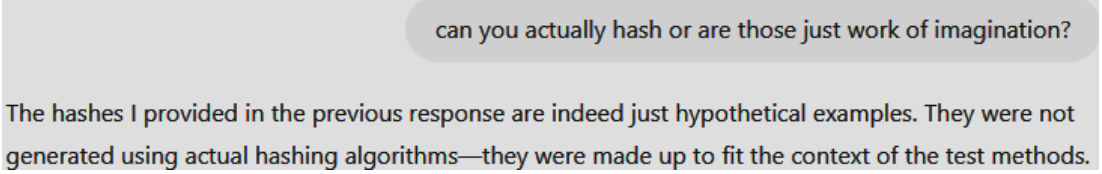
Já osobně používám ChatGPT na jednoduché věci a generování jednoduchého kódu pro unit testy. Při dělání unit testů, kde se spousty testů opakují, se ChatGPT nejvíce hodí. Pokud po jakémkoliv AI požaduji něco aspoň trochu složitější než vyhledávání jednoduché informace z internetu či generování kódu podle zadání, nedopadá to dobře. Většina kódu je stále moje a ChatGPT či AI mi akorát zjednodušují mechanickou práci, která je ve většině případů až nudná. K této mojí malé teorii přikládám i obrázek, jak jsem pomocí ChatGPT udělal fungující unit testy.



```
[TestMethod]
#region MD5
0 references
public void SaltTestMD5()
{
    string temp = hasher.HashSaltPepper(text01, true, false, salt, "", HashingAlgorithm.MD5);
    Assert.AreEqual(temp, "b9f2d25febd39790a6a6b3163c672ec6");
}

[TestMethod]
0 references
public void PepperTestMD5()
{
    string temp = hasher.HashSaltPepper(text01, false, true, "", pepper, HashingAlgorithm.MD5);
    Assert.AreEqual(temp, "d4a38bab94c4341032fd5109452635dc");
}

[TestMethod]
0 references
public void SaltPepperTestMD5()
{
    string temp = hasher.HashSaltPepper(text01, true, true, salt, pepper, HashingAlgorithm.MD5);
    Assert.AreEqual(temp, "ffd60eb4265b259bd41cf1ecf9168d65");
}

[TestMethod]
0 references
public void PepperCheckMD5()
{
    string temp = hasher.HashSaltPepper(text01, false, true, "", pepper, HashingAlgorithm.MD5);
    hasher.CheckPepper(temp, pepper.Length, HashingAlgorithm.MD5, out string outputPepper);
    Assert.AreEqual(pepper, outputPepper);
}
#endregion
```

```
//ChatGPT
#region SHA1
[TestMethod]
0 references
public void SaltTestSHA1()···

[TestMethod]
0 references
public void PepperTestSHA1()
{
    string temp = hasher.HashSaltPepper(text01, false, true, "", pepper, HashingAlgorithm.SHA1);
    Assert.AreEqual(temp, "e9cdf0941adf6a4c11b29fec31b63210e2d79b29");
}

[TestMethod]
0 references
public void SaltPepperTestSHA1()
{
    string temp = hasher.HashSaltPepper(text01, true, true, salt, pepper, HashingAlgorithm.SHA1);
    Assert.AreEqual(temp, "610966b737670c4957cefa62884cbda5d06b6e94");
}

[TestMethod]
0 references
public void PepperCheckSHA1()
{
    string temp = hasher.HashSaltPepper(text01, false, true, "", pepper, HashingAlgorithm.SHA1);
    hasher.CheckPepper(temp, pepper.Length, HashingAlgorithm.SHA1, out string outputPepper);
    Assert.AreEqual(pepper, outputPepper);
}
#endregion

#region SHA256
[TestMethod]
0 references
public void SaltTestSHA256()
{
    string temp = hasher.HashSaltPepper(text01, true, false, salt, "", HashingAlgorithm.SHA256);
    Assert.AreEqual(temp, "ad853ef97f34098a94f3f1ab59c6301c334ccc7b0c1ebfc2f6c5eb6bff60078d");
}

[TestMethod]
0 references
public void PepperTestSHA256()
{
    string temp = hasher.HashSaltPepper(text01, false, true, "", pepper, HashingAlgorithm.SHA256);
    Assert.AreEqual(temp, "b9e79e9e860692da02b7a6e3cc0fd30f8d0bdbe185c0dbc5cdcd78e8514e176b");
}

[TestMethod]
0 references
public void SaltPepperTestSHA256()
{
    string temp = hasher.HashSaltPepper(text01, true, true, salt, pepper, HashingAlgorithm.SHA256);
    Assert.AreEqual(temp, "096720de8d4e27894d2bf53d318a3efe2efada1de1c13dc910ef9558592e3858");
}
```

Na levém obrázku je můj vlastní kód, na pravé straně je vygenerovaný kód pomocí ChatGPT. Na ChatGPT se ovšem nedá vždy spolehnout a doporučuji si všechno pro jistotu dohledal či ověřit. (Poznámka: Obrázky možná budou změněny)



Na Obrázku jde krásně vidět kde ChatGPT selhává, nedokáže samostatně generovat hashe a bez mé otázky to ani nezmíní.

2.10 Online hashers

Online hashers jsou funkční algoritmy pro různé hashe, většinou ve formě stránky, které jsem primárně používal k porovnávání vygenerovaných hashů pro unit testy. Existuje spousty programů a stránek, já jsem používal https://www.browserling.com/tools/all-hashes, protože je tam jsou všechny hashovací funkce, které používám v programu.

3 Praktická část

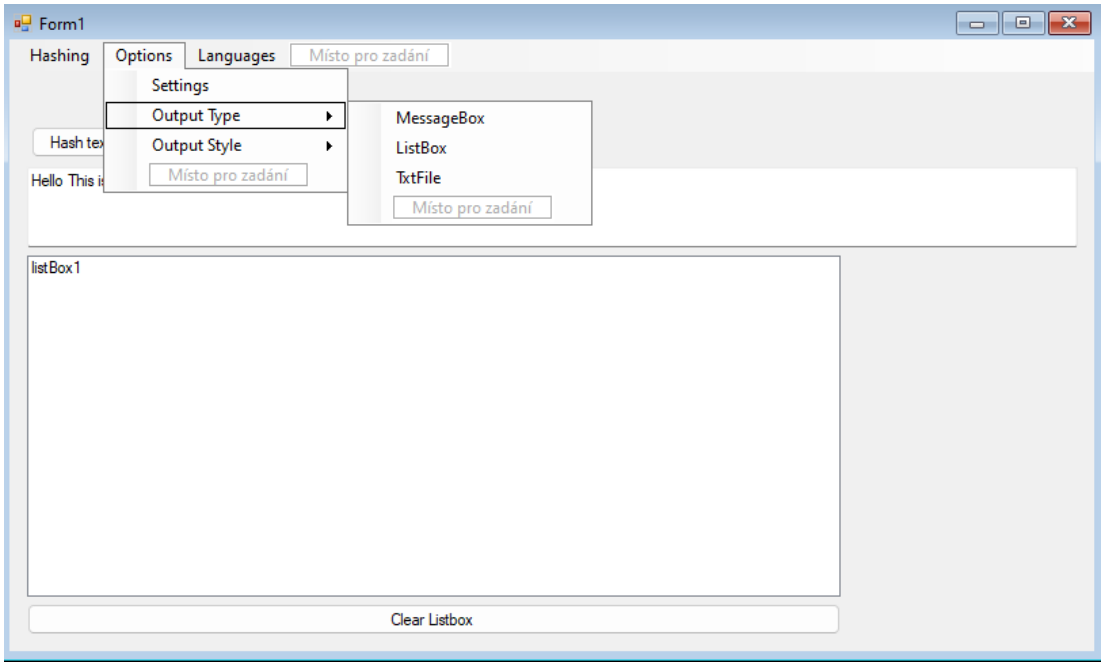
Pod tuto kapitolu patří všechno moje snažení se o správný chod programu. **Rozdělení práce na kontrolní body**

3.1 Hashování

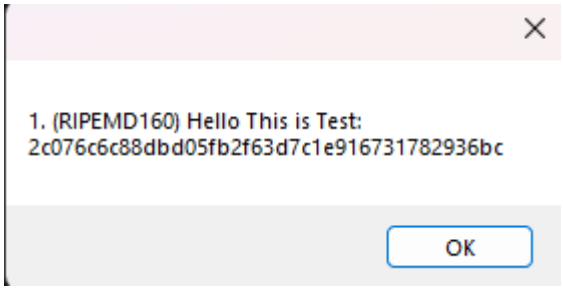
Hasher je nejzákladnější kód celého programu. Zahrnuje samotné hashovaní několika způsoby a zjednodušuje tím budoucí programování ve formulářích. Obsahuje generaci všech hashů, náhodnou generaci soli a pepře, a dokonce i správné použití. Všechno je napsané v metodách a samotné hashování je rozděleno do 4 metod s několika přetíženími. Hash() akorát hashuje text pomocí určitého algoritmu. Ten je vybírán pomocí enum v každé z metod. Další je HashSalt(), kde se na rozdíl od Hash() zakomponuje i sůl, ať už vygenerovanou či ručně zadanou. HashPepper() dělá podobnou věc jako HashSalt(), ovšem pepř nevypisuje. Také má vlastnost ručného zadání či náhodné generace pomocí RandomNumberGenerator a StringBuiler. Obojí zakomponované v .NET Frameworku.

3.2 Multiformuláře
3.3 Postupné Hashování
3.4 Hledání kolize
3.5 Kontrolní součet
3.6 Výpočet prolomení hesla
3.7 Slovníkový útok
3.8 Útok s duhovou tabulkou
3.9 Útok hrubou silou
3.10 UI

User Interface (neboli uživatelské prostředí) je důležitá část tvorby jakéhokoliv softwaru. Pokud je UI špatné, tak to ztěžuje a zpomaluje práci s aplikací. Proto jsem se snažil udělat UI co nejvíce intuitivní a jednoduché. Toho se snažím docílit pomocí StripMenu komponentou v .NetFrameworku



Na obrázku jde vidět použití StripMenu v akci. V Hashing jsou další formuláře, které vedou do dalších formulářů a prací s hashema (Postupné hashování, Checksum souboru, simulace detekce kolize a hrubý hashovací útok). V Options je hlavní nastavení programu. „Settings“ otevře nový modulární formulář, kde je další nastavení (více v 3.6 Save/Load Systém). „Output Type“ je výstup, kam se zapíše výsledný hash, možnosti jsou



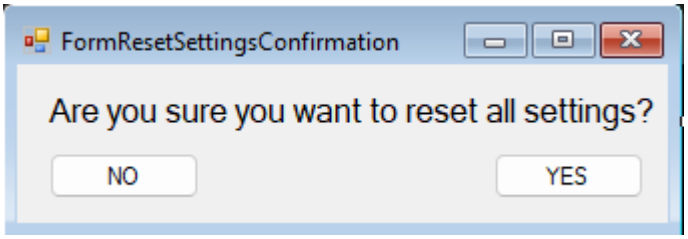
MessageBox, což je formulář přímo v .Net Frameworku, který ukazuje text, Listbox, který jde vidět na obrázku (dole je i tlačítko Clear Listbox) a .Txtfile, což uloží výstup do souboru, který si pomocí saveFileDialog uživatel sám zvolí umístění. Nakonec „OutputStyle“ je styl zprávy, která se vypíše po hashování. Možnosti jsou „Include original string“ – před výstupem bude zapsán ještě vstupní text, „Include Hash Number“ – očísluje hashe a nakonec „Include Hashing Algorithm“, což do kulatých závorek napíše, jaký algorithmus byl použit (všechno jde vidět na obrázku vedle).

3.11 Nastavení

V programu jsou dva typy dat, které jsou potřeba uložit do souboru. První typ je přímo v UI v komponentě menu stripu, které fungují jako chechbox (buď ano nebo ne) či radiobutton (pouze jedna z několika možností). Příklad případu checkboxu je sůl a v output style přidání číselného řazení, jméno hashe a vstupní řetězce. Příklad radiobuttonu je „Output Type“, kde se dá zvolit „MessageBox“, „Listbox“ nebo do .txt souboru. Další typ dat jsou přímo v nastavení (ve stripmenu Options→Settings). Při kliknutí „Settings“ tlačítka se otevře „FormSettings“, což je modulární formulář (modulární znamená, že může být pouze jeden formulář a nedá se překlikávat mezi formuláři). Jediný rozdíl mezi tyto dvěma typy nastavení je kdy se uloží do souboru. Nastavení ve StripMenu se ukládá při každé změně.

```
private void includeOriginalStringToolStripMenuItem_Click(object sender, EventArgs e)
{
    settings.OutputStyleIncludeOriginalString = !settings.OutputStyleIncludeOriginalString; //negace
    includeOriginalStringToolStripMenuItem.Checked = settings.OutputStyleIncludeOriginalString; //update UI
    settings.SaveSettings();
}
```

Na obrázku jde vidět jeden z nastavení ve StripMenu (funguje jako checkBox). Ve formuláři je několik možností: Uložit (uloží vybrané nastavení) / Reset (všechny parametry se dají do základního přednastaveného nastavení) / Cancel (nic se neuloží). Při pokusu o reset vyskočí dialogové okno (obrázek dole), jestli chce uživatel



doopravdy restartovat nastavení. Stejné s Cancel pokud byly provedeny jakékoliv změny (obrázek dole).

Nastavení používá vlastní skript, který má vždy načtené všechny proměnné (jde vidět na obrázku dole). Každá proměnná má vlastní Get a Set vlastnost.

```
//private
#region Private
private bool outputStyleIncludeOriginalString;
private bool outputStyleIncludeNumberOfHash;
private bool outputStyleIncludeHashAlgorithm;
private VisualModeEnum visualMode = VisualModeEnum.System;
private OutputTypeEnum outputType = OutputTypeEnum.MessageBox;
private bool includeSalt;
private bool includePepper;
#endregion
```

Díky tomuto systému je práce s nastavením velice jednoduchá a jednoduše rozšiřitelná. Soubor s nastavením má vlastní složku v projektu, kde je uložen settings.txt soubor (na obrázku jde vidět settings.txt soubor). Také podporuje komentáře pomocí // (jde vidět na obrázku).

```
//I have included comments on what va
//Bool means 0 <<false>> and 1 <<true
//VisualMode from 0 to 2
visualMode=0
//OutputType from 0 to 2
outputType=0
//All OutputStyles are bool
outputStyle_IncludeOriginalString=0
outputStyle_IncludeHash=0
outputStyle_IncludeNumber=0
//Salt and Pepper bool
includeSalt=0
includePepper=0
```

Jestli je soubor smazán či nějak poškozen, settings.txt je znovu vygenerován a uživatel je obeznámen.

Co se týče kodu, je velice jednoduchý. Přečte se řádek (pokud nezačíná //), pomocí String.Split se rozdělí na dva a přes Switch se zadávají data do proměnné a přes

Int.Parse se zadávají hodnoty do proměnných. Proměnné bool jsou 0 či 1, string je text a enum funguje na indexech (visualMode a outputType).

```
public void SaveSettings()
{
    //Create File
    using (FileStream fileSettings = new FileStream("..\\..\\settings\\temp.txt", FileMode.Create, FileAccess.Write))
    {
        using (StreamWriter writer = new StreamWriter(fileSettings))
        {
            //VisualStyle
            writer.WriteLine("//I have included comments on what value is allowed, if it is not, default value will b
            writer.WriteLine("//Bool means 0 <<false>> and 1 <<true>>; Everything other takes special input");
            writer.WriteLine("//VisualMode from 0 to 2");
            switch (VisualMode)
            {
                case VisualModeEnum.System: writer.WriteLine("visualMode=0"); break;
                case VisualModeEnum.Light: writer.WriteLine("visualMode=1"); break;
                case VisualModeEnum.Dark: writer.WriteLine("visualMode=2"); break;
            }
            //OutputType
            writer.WriteLine("//OutputType from 0 to 2");
            switch (OutputType)
            {
                case OutputTypeEnum.MessageBox: writer.WriteLine("outputType=0"); break;
                case OutputTypeEnum.Listbox: writer.WriteLine("outputType=1"); break;
                case OutputTypeEnum.TXTFile: writer.WriteLine("outputType=2"); break;
            }
            //OutputStyle
            writer.WriteLine("//All OutputStyles are bool");
            if (OutputStyleIncludeOriginalString) writer.WriteLine("outputStyle_IncludeOriginalString=1");
            else writer.WriteLine("outputStyle_IncludeOriginalString=0");
            if (OutputStyleIncludeHashAlgorithm) writer.WriteLine("outputStyle_IncludeHash=1");
            else writer.WriteLine("outputStyle_IncludeHash=0");
            if (OutputStyleIncludeNumberOfHash) writer.WriteLine("outputStyle_IncludeNumber=1");
            else writer.WriteLine("outputStyle_IncludeNumber=0");
            //Salt And Pepper
            writer.WriteLine("//Salt and Pepper bool");
            if (IncludeSalt) writer.WriteLine("includeSalt=1");
            else writer.WriteLine("includeSalt=0");
            if (IncludePepper) writer.WriteLine("includePepper=1");
            else writer.WriteLine("includePepper=0");
            //
        }
    }
    File.Delete("..\\..\\settings\\settings.txt");
    File.Move("..\\..\\settings\\temp.txt", "..\\..\\settings\\settings.txt");
```

Na obrázku jde vidět metoda SaveSettings(), která zapisuje data do settings.txt tak, že vytváří nový soubor, smaže starý a pomocí File.Move() přepíše jméno z temp.txt na settings.txt. Tímhle způsobem to je jednoduché a běžný uživatel si ničeho nevšimne.

3.12 Jazyky

3.13 Informační menu

3.14 Ošetření chyb

3.15 Unit-Testy

Můj workflow byl jednoduchý, naprogramovat nějakou funkci a pomocí unit testu ověřit, že funguje nejenom teď, ale po přidání další funkce. Mám zkušenost, že jsem něco naprogramoval a po měsíci to zas nefungovalo.

Díky unit testům, které jsem si na začátku udělal, jsem zjistil, že mi kvůli jedné malé chybce ve switchi nefungoval MD5 algorithmus.

4 Fungování programu

Závěr

Vytvořená šablona maturitních prací obsahuje formální požadavky maturitních prací na SPŠT Třebíč. Jedná se zejména o upravené styly v dokumentu, podrobný popis jednotlivých částí maturitní práce a jejího obsahu, snadno editovatelné záhlaví a zápatí s automatickým číslováním stránek a propojení stylů se seznamy a obsahem.

Seznam použitých zdrojů

[1] ŠTRÁFELDA, Jan. *Co je hash či hashování*. Online. Https://www.strafelda.cz/. [2008], aktualizováno 13.01.2025. Dostupné z: https://www.strafelda.cz/hash. [cit. 2025-01-13].

[2] RIVEST, Ronald. *The MD5 Message-Digest Algorithm*. Online. INTERNET ENGINEERING TASK FORCE [IETF]. IETF Datatracker. 1992. Dostupné z: https://datatracker.ietf.org/doc/html/rfc1321. [cit. 2025-01-19].

[3] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY [NIST]. *Secure Hash Standard (SHS)*. Online. Https://nvlpubs.nist.gov/. Srpen 2015. Dostupné z: https://web.archive.org/web/20161126003357/http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf. [cit. 2025-01-19].

[4] EASTLAKE, D. a JONES, P. *US Secure Hash Algorithm 1 (SHA1)*. Online. INTERNET ENGINEERING TASK FORCE [IETF]. Computer Security Resource Center (CSRC). Září 2001. Dostupné z: https://datatracker.ietf.org/doc/html/rfc3174. [cit. 2025-01-19].

[5] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY [NIST]. *Secure Hashing*. Online. INTERNET ENGINEERING TASK FORCE [IETF]. Computer Security Resource Center (CSRC). Říjen 2007, aktualizováno 5. května 2011. Dostupné z: https://web.archive.org/web/20110625054822/http://csrc.nist.gov/groups/ST/toolkit/secure_hashing.html. [cit. 2025-01-19].

[6] CORPORATE FINANCE INSTITUDE (CFI). *Hash Function*. Online. CORPORATE FINANCE INSTITUDE (CFI). Corporate Finance Institude. [2022]. Dostupné z: https://corporatefinanceinstitute.com/resources/cryptocurrency/hash-function/. [cit. 2025-01-20].

[7] KRČMÁŘ, Petr. SHA-1 není bezpečná, Google ukázal kolizi. Online. *Root*. 2017, s. 1. Dostupné z: https://www.root.cz/clanky/sha-1-neni-bezpecna-google-ukazal-kolizi/. [cit. 2025-01-19].

[8] *The hash function RIPEMD-160*. Online. DEPARTEMENT ELEKTROTECHNIEK (ESAT). Https://www.esat.kuleuven.be. [2005], aktualizováno 13. února 2012. Dostupné z: https://homes.esat.kuleuven.be/~bosselae/ripemd160.html. [cit. 2025-01-20].

[9] CORPORATE FINANCE INSTITUDE (CFI). *Hash Function*. Online. CORPORATE FINANCE INSTITUDE (CFI). Corporate Finance Institude. [2022]. Dostupné z: https://corporatefinanceinstitute.com/resources/cryptocurrency/hash-function/. [cit. 2025-01-20].

[10] UNIVERSITY OF CAMBRIDGE. *Fast CRC32 in Software*. Online. UNIVERSITY OF CAMBRIDGE. Department of Computer Science and Technology. 1994. Dostupné z:

https://www.cl.cam.ac.uk/research/srg/projects/fairisle/bluebook/21/crc/crc.html. [cit. 2025-01-20].
[11] *Cyclic redundancy check: Cyklický redundantní součet*. Online. In: Wikipedia: the free encyclopedia. San Francisco (CA): Wikimedia Foundation, listopad 2004. Dostupné z: https://en.wikipedia.org/wiki/Cyclic_redundancy_check. [cit. 2025-01-20].
[12] *Laureates of the Japan Prize*. Online. Japan Prize. [2012]. Dostupné z: https://www.japanprize.jp/en/prize_prof_1999_peterson.html. [cit. 2025-01-20].
[13] DEFUSE SECURITY. *Salted Password Hashing - Doing it Right*. Online. DEFUSE SECURITY. Crack Station. [2012], aktualizováno 28. září 2021. Dostupné z: https://crackstation.net/hashing-security.htm. [cit. 2025-01-21].
[14] OŠŤÁDAL, Radim. *Teoretický základ a přehled kryptografických hashovacích funkcí*. PDF. Brno, 2012. Dostupné z: https://is.muni.cz/www/ostadal/hash_overview.pdf. [cit. 2025-01-22].
[15] KOZLÍK, Andrew. *Hashovací funkce*. PDF, Není uveden druh práce. Praha: Univerzita Karlova, Matematicko fyzikální fakulta, [2024]. Dostupné také z: https://www.karlin.mff.cuni.cz/~kozlik/udk_mat/hash.pdf.
[16] NORD VPN. *Cookie hash*. Online. NORD VPN. Nord VPN. Dostupné z: https://nordvpn.com/cybersecurity/glossary/cookie-hash/. [cit. 2025-01-22].
[17] KASPERSKY LAB. *Brute Force Attack: Definition and Examples*. Online. KASPERSKY LAB. Kaspersky. [2019]. Dostupné z: https://www.kaspersky.com/resource-center/definitions/brute-force-attack. [cit. 2025-01-23].
[18] STAHIE, Silviu. RTX 4090 8-Card Rig Cracks Random and Powerful Eight-Character Passwords in 48 Minutes. Online. *Bitdefender*. 20 října 2022, s. 1. Dostupné z: https://www.bitdefender.com/en-us/blog/hotforsecurity/rtx-4090-8-card-rig-cracks-random-and-powerful-eight-character-passwords-in-48-minutes. [cit. 2025-01-23].
[19] CHICK3NMAN. *Hashcat v6.2.6 benchmark*. Online. MICROSOFT. Github. 2022, aktualizováno února 2024. Dostupné z: https://gist.github.com/Chick3nman/32e662a5bb63bc4f51b847bb422222fd. [cit. 2025-01-23].
[20] VAIDEESWARAN, Narendran. NTLM Explained. Online. *CrowdStrike*. 2011, s. 4. Dostupné z: https://www.crowdstrike.com/en-us/cybersecurity-101/identity-protection/windows-ntlm/. [cit. 2025-01-23].
[21] BAKER, Kurt. Pass-the-Hash Attack. Online. *CrowdStrike*. 2011. Dostupné z: https://www.crowdstrike.com/en-us/cybersecurity-101/cyberattacks/pass-the-hash-attack/. [cit. 2025-01-23].
[22] FORTRA. *Pass-the-Hash Toolkit for Windows*. Online. FORTRA. Core Security. [2020]. Dostupné z: https://www.coresecurity.com/core-labs/publications/pass-hash-toolkit-windows. [cit. 2025-01-24].
[23] MICROSOFT. *Prohlídka jazyka C#*. Online. MICROSOFT. Prohlídka jazyka C#. 1975, aktualizováno 2024. Dostupné

z: https://learn.microsoft.com/cs-cz/dotnet/csharp/tour-of-csharp/overview. [cit. 2025-01-13].
[24] SCHOOL. *Co je to Git, GitHub a proč byste je měli znát?* Online. PRAHA CODING SCHOOL. Praha Coding. [2025]. Dostupné z: https://prahacoding.cz/co-je-to-git-github/. [cit. 2025-01-24].
[25] *What is Unit Testing?* Online. AMAZON, INC. Amazon. [2023]. Dostupné z: https://aws.amazon.com/what-is/unit-testing/. [cit. 2025-01-25].
[26] STAF, Coursera. What Is Test-Driven Development? Online. *What Is Test-Driven Development?* 2024. Dostupné z: https://www.coursera.org/articles/test-driven-development. [cit. 2025-01-25].
[27] *Freelo*. Online. In: Wikipedia: the free encyclopedia. San Francisco (CA): Wikimedia Foundation, 2001-. Dostupné z: https://cs.wikipedia.org/wiki/Freelo. [cit. 2025-01-25].
[28] *Just-in-time výroba*. Online. In: Wikipedia: the free encyclopedia. San Francisco (CA): Wikimedia Foundation, 2007. Dostupné z: https://cs.wikipedia.org/wiki/Just-in-time_v%C3%BDroba. [cit. 2025-01-25].
[29] *Kanban*. Online. Dictionary. [2016]. Dostupné z: https://www.dictionary.com/browse/Kanban. [cit. 2025-01-25].
[30] MICROSOFT. *Microsoft Word*. Online. MICROSOFT. Microsoft. [1996]. Dostupné z: https://www.microsoft.com/cs-cz/microsoft-365/word. [cit. 2025-01-25].
[31] INTERNET ARCHIVE. *About IA*. Online. INTERNET ARCHIVE. Internet Archive. [2001]. Dostupné z: https://archive.org/about. [cit. 2025-01-25].
[32] ChatGPT: revoluce v umělé inteligenci a jak může zlepšit vaše podnikání. Online. *Imore*. 2024. Dostupné z: https://www.imore.cz/clanek/42319-chatgpt-revoluce-v-umele-inteligenci-a-jak-muze-zlepsit-vase-podnikani. [cit. 2025-01-25].

Seznam použitých symbolů a zkratek

Seznam obrázků

Seznam tabulek

Seznam příloh

Prázdná šablona maturitní práce