Tvorba přístupových hesel pro specifické prostředí

Martin Pfeffer

Bakalářská práce
2018

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2017/2018

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Martin Pfeffer**
Osobní číslo:	**A14285**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Bezpečnostní technologie, systémy a management**
Forma studia:	**kombinovaná**
Téma práce:	**Tvorba přístupových hesel pro specifické prostředí**
Téma anglicky:	**Creating Passwords For Specific Environments**

Zásady pro vypracování:

1. **Proveďte literární rešerši na téma tvorby přístupových hesel.**
2. **Popište způsoby tvorby ve vazbě na specifické prostředí.**
3. **Navrhněte způsob tvorby hesel.**
4. **Aplikujte vlastní návrh řešení.**
5. **Vyhodnoťte zvolený způsob tvorby a jeho reálnou aplikovatelnost.**

Rozsah bakalářské práce:

Rozsah příloh:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **DOUCEK, Petr, Luděk NOVÁK, Lea NEDOMOVÁ a Vlasta SVATÁ. Řízení bezpečnosti informací: 2. rozšířené vydání o BCM. 2., přeprac. vyd. Praha: Professional Publishing, 2011, 286 s. ISBN 978-80-7431-050-8.**
2. **JAŠEK, Roman a Milan OULEHLA. Moderní kryptografie: Průvodce světem šifrování. 1. Praha: IFP Publishing, 2017. ISBN 978-80-87383-67-4.**
3. **Knopová Martina. Bezpečnost dat v informačních systémech [online]. Dostupné z WWW http://ikaros.cz/bezpecnost-dat-v-informacnich-systemech.**
4. **Miroslav Ludvík; Bohumír Štědroň. Teorie bezpečnosti počítačových sítí. Computer Media. 2008. 98 s. ISBN 978-80-8668-635-6.**
5. **Vyhláška č. 523/2005 Sb., o bezpečnosti informačních a komunikačních systémů a dalších elektronických zařízení nakládajících s utajovanými informacemi a o certifikaci stínicích komor, ve znění vyhlášky č. 453/2011 Sb.**

Vedoucí bakalářské práce: **prof. Mgr. Roman Jašek, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **8. prosince 2017**

Termín odevzdání bakalářské práce: **24. května 2018**

Ve Zlíně dne 12. prosince 2017

doc. Mgr. Milan Adámek, Ph.D.
děkan

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

Ing. Jan Valouch, Ph.D.
ředitel ústavu

Jméno, příjmení: Martin Pfeffer

Název bakalářské práce: Tvorba přístupových hesel pro specifické prostředí

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské 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 bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou 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/bakalářskou 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 bakalářské 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 bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské 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 bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne

.....
podpis diplomanta

ABSTRAKT

Tématem této práce je tvorba silných a bezpečných hesel za použití hašovacích funkcí. Toto téma bylo vybráno z důvodu častého používání slabých a nedostatečných hesel uživateli, která jsou rizikem nejen pro uživatele, ale i jeho zaměstnavatele nebo firmu. Teoretická část práce je zaměřena na seznámení se s heslem, jeho vlastnostmi a způsoby jakými jsou hesla překonávána. Dále na seznámení s hašovacími funkcemi, jejich vlastnostmi a využitím. Jako součást této práce byla vytvořena aplikace „Oraculum“, která za využití hašovacích funkcí vytváří hesla. V praktické části jsou vysvětleny její funkce, zdrojový kód a nakonec jsou vyhodnoceny aplikací vytvořená hesla. Tato aplikace by mohla usnadnit uživatelům tvorbu silných a bezpečných hesel.

Klíčová slova: heslo, autentizace, hašovací funkce, haš

ABSTRACT

The theme of this thesis is the creation of strong and secure passwords using hash functions. This topic has been chosen because of the frequent use of weak and insufficient passwords by the user, which are a risk not only for the user, but also for his employer or business. The theoretical part of the thesis is focused on getting acquainted with the password, its properties and the ways in which the passwords are overcome. Next on getting acquainted with hash functions, their properties and utilization. As part of this thesis, an application "Oraculum" has been created, which creates passwords using hash functions. The practical part of thesis explains functions of aplication, its source code and finaly passwords created by aplication „Oraculum“ are evaluated. This application could make it easier for users to create strong and secure passwords.

Keywords: password, authentication, hash funkcion, hash

Chtěl bych poděkovat vedoucímu mé bakalářské práce panu prof. Mgr. Romanu Jaškovi, Ph. D. za cenné rady a odborný dohled při zpracování této práce.

Prohlašuji, že odevzdaná verze bakalářské/diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD **7**
I TEORETICKÁ ČÁST **8**
1 HESLO **9**
1.1 BEZPEČNOST HESLA 10
1.1.1 Ochrana poskytovaná uživatelem 10
1.1.2 Ochrana poskytovaná provozovatelem služby 11
1.2 SILNÉ A BEZPEČNÉ HESLO 12
1.2.1 Způsoby útoku 12
1.2.1.1 Slovníkový útok 12
1.2.1.2 Recyklace hesel 13
1.2.1.3 Sociální inženýrství 13
1.2.1.4 Keyloggery 15
1.2.1.5 Zjištění hesla od uživatele 15
1.2.1.6 Využití přednastavených přístupů 15
1.2.1.7 Útok hrubou silou 16
1.2.2 Tvorba silného a bezpečného hesla 16
1.2.2.1 Složitost hesla 16
1.2.2.2 Délka hesla 17
1.2.2.3 Entropie hesla 18
1.2.2.4 Vztah mezi vlastnostmi hesla 20
1.2.2.5 Posouzení hesla vzhledem k způsobům útoků 22
1.2.2.6 Vytvoření bezpečného hesla 22
2 HAŠOVACÍ FUNKCE **25**
2.1 VLASTNOSTI HAŠOVACÍ FUNKCE 25
2.1.1 Konstantní velikost haše 25
2.1.2 Reagování na změny na vstupu (lavinový efekt) 26
2.1.3 Jednosměrnost 26
2.1.4 Bezkoliznost 26
2.2 VYUŽITÍ HAŠOVACÍCH FUNKCÍ 27
2.2.1 Ověřování pravosti dat 27
2.2.2 Porovnávání dat 28
2.2.3 Vyhledávání dat (indexace) 28
2.2.4 Ukládání uživatelských hesel 29
2.3 BEZPEČNOST HAŠOVACÍCH FUNKCÍ 29
2.4 NEJBĚŽNĚJŠÍ DRUHY HAŠOVACÍCH FUNKCÍ 30
2.4.1 Message-digest algorithm 30
2.4.2 Secure hash algorithm 31
2.5 VYUŽITÍ HAŠOVACÍCH FUNKCÍ K TVORBĚ HESLA 31
II PRAKTICKÁ ČÁST **33**
3 VYUŽITÍ HAŠOVACÍCH FUNKCÍ PŘI TVORBĚ HESLA **34**

3.1 PRAKTICKÉ VÝHODY A NEVÝHODY VYUŽITÍ HAŠOVACÍCH FUNKCÍ PŘI TVORBĚ HESLA 34
3.1.1 Výhody 34
3.1.2 Nevýhody 34
3.1.3 Jiný úhel pohledu 35
4 APLIKACE „ORACULUM“ 36
4.1 UŽIVATELSKÉ ROZHRANÍ APLIKACE ORACULUM 36
4.1.1 Okno aplikace „Oraculum“ 36
4.1.1.1 Panel Vstup 38
4.1.1.2 Panel Parametry tvorby 39
4.1.1.3 Tlačítko Vytvoř 39
4.1.1.4 Panel Výstup 41
4.2 ZDROJOVÝ KÓD APLIKACE „ORACULUM“ 42
4.2.1 Struktura aplikace „Oraculum“ 42
4.2.2 Hash.cs 43
4.2.2.1 Veřejná textová funkce „GeneratorMD5“ 43
4.2.2.2 Další hašovací funkce v třídě Hash.cs 45
4.2.3 Uprava.cs 46
4.2.3.1 Veřejná číselná funkce „Prevod“ 47
4.2.3.2 Veřejná číselná funkce „CH“ 48
4.2.3.3 Veřejná textová funkce „uprava“ 50
4.2.4 FormMain.cs 56
4.2.4.1 Soukromá funkce „FormMain_Load“ 56
4.2.4.2 Soukromá funkce „bOK_Click“ 57
4.2.4.3 Soukromá funkce „bZobrazit_Click“ 57
4.2.4.4 Soukromá funkce „bVytvor_Click“ 58
4.2.4.5 Soukromá funkce „cbViditelnyVystup_CheckedChanged“ 62
4.2.4.6 Soukromá funkce „cbViditelnyVstup_CheckedChanged“ 63
4.2.4.7 Soukromá funkce „bKopirovat_Click“ 64
4.3 POSTUP TVORBY HESLA 64
5 VYHODNOCENÍ APLIKACE „ORACULUM“ 67
ZÁVĚR 69
SEZNAM POUŽITÉ LITERATURY 71
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK 73
SEZNAM OBRÁZKŮ 74
SEZNAM TABULEK 75
SEZNAM PŘÍLOH 76

ÚVOD

Je 21. století. Digitální věk. Světu vládnou jedničky a nuly. Výpočetní technika už pronikla do všech odvětví a oborů lidské činnosti. Internet je něco nehmatatelného a neviditelného. Něco, co ale dokázalo propojit skoro všechny lidi na naší planetě. Nemá hranic a úplně neskutečně rozšiřuje možnosti běžného člověka. Pokud víte jak a kde máte hledat, najdete na něm úplně vše. Je to až ironické, že ta samá věc zároveň dost zmenšila naši planetu. Z České republiky do Austrálie se díky Internetu dostanete prakticky okamžitě. V jednu chvíli můžete pozorovat lvy v Africe, za okamžik západ slunce nad severním pólem. Létat v oblacích nebo se potápět uprostřed oceánu. Můžete si povídat s lidmi, se kterými byste se bez Internetu neměli nikdy šanci potkat. V Internetu neexistuje pojem daleko. Vše je dostupné okamžitě jen díky několika kliknutí myší.

Lidé si zvykli na možnosti a pohodlí, které jim nabízí Internet. Od sociálních sítí po stránky s odbornými texty a materiály. Každý zde může najít něco, co ho zajímá. Často si stačí jen založit účet a už máte přístup k vámi hledaným službám. Účet je zde vlastně taková virtuální identita.

Na rozdíl od reálného světa, ve kterém si většina z nich uvědomuje rizika spojená s kriminální činností některých jedinců, ve virtuálním světě internetu jim často bezpečnostní hrozby vůbec nic neříkají. V reálném světě si uživatel nechá nainstalovat mříže na okna, pořídí bezpečnostní skla, bezpečnostní dveře, zámek atd. jelikož si uvědomuje, že pokud si domov nezabezpečí, hrozí mu reálná šance na materiální nebo psychickou újmu. Ve virtuálním světě internetu však velké množství uživatelů používá u všech účtů stejné heslo, a pokud to poskytovatel služby dovolí tak co nejjednodušší, jelikož čím složitější, tím náročnější na zapamatování a také na napsání. Marná je pak snaha poskytovatelů služeb zajistit uživatelům co největší míru ochrany za pomoci mnohdy velmi nákladných bezpečnostních opatření. Když prakticky vzato, uživatel sám nechává útočníkovi dveře dokořán.

A přitom by stačilo se jen na chvilku zamyslet a vytvořit silné a bezpečné heslo, které uživateli zajistí, že dveře k jeho virtuálnímu světu zůstanou pro ostatní uživatele zavřeny.

I. TEORETICKÁ ČÁST

1 HESLO

Co je to vlastně heslo? Výkladový slovník kybernetické bezpečnosti uvádí:

„Heslo - Znakový řetězec používaný jako součást autentizační informace. Obecný prostředek k autentizaci uživatele pro přihlášení k počítači, k přístupu k souborům, programům a službám.“ [1]

„Autentizace - Poskytnutí záruky, že prohlašovaná charakteristika je správná.“[1]

Když se zeptáte laika, odpověď bude asi taková: „To je takové slovo, které musím napsat, když se chci přihlásit.“ I s tímto výrokem by se s trochou dobré vůle dalo souhlasit.

Bohužel si ale dost lidí myslí, že heslo jim chrání počítač, data, programy atd. Což už není tak pravdivá informace, jak by se na první pohled mohlo zdát. Pravda je taková, že počítač, data, programy či informace chrání různé druhy bezpečnostních opatření, jako jsou například omezení přístupu, použití různých druhů šifrování, antivirová ochrana nebo utajování existence. Přihlašovací údaje (login a heslo) bývají využívány při ověřování identity uživatele. Ověřování je formou porovnávání údajů zadaných uživatelem s údaji v databázi platných identit. Důvodem je zjištění, zda se jedná o osobu, za kterou se přihlašující se jedinec vydává. To, jestli má daná osoba přístup k něčemu dalšímu je pak ovlivněno oprávněním daného uživatele. *[2]*

Když se nad tím člověk zamyslí, tak bezpečnostní opatření chránící data, jsou schránkou a heslo je klíč, který nám tuto schránku otevírá. Pokud máte správný klíč, máte přístup k datům uvnitř schránky. *[2]*

Přihlášení je prakticky jednostupňové ověření identity, při kterém prokazujeme znalost. Pokud nemáme znalost správného hesla, je nám přístup zamítnut. Heslo jako způsob ověření se využívalo už v dávné minulosti, kdy členové určité skupiny (armáda, různé spolky) za pomocí předem domluveného hesla ověřovali, jestli daná osoba skutečně patří k nim, nebo je to někdo cizí. *[2]*

Heslo je dnes nejpoužívanější prostředek k ověření oprávněnosti přístupu k uživatelskému účtu. Většina z nás ho několikrát za den využije. V současné době díky různým funkcím prohlížečů internetu či správců hesel si spousta uživatelů ani neuvědomuje, že nějaké heslo používá. Jednoduše jen „klikne“ na odkaz nebo spustí program a jiný program sám vyplní jeho přihlašovací údaje. *[2, 3]*

Jedná se o uživatelsky velmi přívětivý systém. Na druhou stranu tento systém má určité bezpečnostní nedostatky. U ukládání hesel do internetových prohlížečů je riziko jednoznačné. Kdokoliv, kdo se dostane do profilu daného uživatele, má pak automaticky kompletní přístup k jeho účtům, u kterých má v prohlížeči uložené heslo. U většiny správců hesel pak jsou všechny spravované účty opatřeny silným a bezpečným heslem. Nicméně, správce jako takový je opatřen přístupovým heslem, které vytvořil uživatel. Takže prakticky jsou při přístupu ke správci hesel, všechny uživatelovi účty opatřeny jedním heslem, které vytvořil uživatel a které nemusí být silné ani bezpečné. Jako další nedostatek některých správců hesel by mohla být nutnost instalování různých doplňků. To by nemuselo být povoleno buď administrátorem systému, nebo samotným uživatelem, v jehož profilu se chcete přihlásit ke svému online účtu například na různých sociálních sítích. *[3]*

Předtím nežli se začneme věnovat pojmu bezpečné heslo, bylo by dobré si uvědomit, že kromě pojmu bezpečné heslo je zde i pojem bezpečnost hesla. Ačkoliv by se mohlo zdát, že se jedná o ten samý pojem, není tomu tak. Bezpečné heslo je heslo splňující určité parametry, bezpečnost hesla je prakticky ochrana samotného hesla před vnějšími hrozbami. *[2]*

1.1 Bezpečnost hesla

Bezpečnost hesla je ochrana samotného hesla před vnějšími hrozbami. Tuto ochranu můžeme rozdělit do dvou základních skupin v závislosti na straně, která ochranu poskytuje. *[4]*

1.1.1 Ochrana poskytovaná uživatelem

Do ochrany poskytované uživatelem patří veškerá opatření, která brání zjištění hesla jinou osobou. Prakticky se jedná o dodržování několika základních pravidel: *[4, 5]*

- své heslo nesdílet s žádným jiným uživatelem. *[4, 5]*
- pokud je to trochu možné, nemít heslo nikde napsané. V případě, že si uživatel heslo nepamatuje a musí ho mít zapsané, držet místo zapsání bezpečně uschované před zneužitím jiným uživatelem. *[4, 5]*
- s žádným uživatelem se o svém hesle nebavit, neposkytovat žádné údaje, které by mohli jinému uživateli usnadnit uhodnutí nebo překonání hesla. (mezi tyto údaje patří např. délka hesla („Já mám heslo o 10-ti znacích.“), formát hesla („Já mám nejdříve číslo, pak normální znaky a na konci velké písmeno.“), nebo obsah hesla

(„Já si jako heslo nastavil datum narození ženy.“ nebo „Jako heslo jsem měl nastavené jméno dítěte, ale jelikož to mělo jen 7 znaků a oni jich chtějí 8, tak jsem za jméno napsal ještě číslo a je to.“)) *[4, 5]*

- svá hesla pravidelně měnit. Toto pravidlo má své opodstatnění, jelikož síla hesla spočívá mimo jiné v schopnosti odolávat dostatečně dlouho různým způsobům útoku. Díky pravidelné změně hesla je šance, že ani úspěšné prolomení hesla nemusí útočníkovi být nic platné, pokud doba nutná pro prolomení hesla stačila na to, aby si uživatel heslo změnil v rámci povinné změny po určitém čase. Na druhou stranu, toto pravidlo může v některých případech být až kontraproduktivní. Příkladem tohoto může být například stav, kdy z důvodu častých obměn a díky pravidlu nepoužívat již použitá hesla, začne uživatel svá hesla zjednodušovat a oslabovat. Doba potřebná k prolomení takového hesla pak bude kratší nežli obměňovací cyklus. Tím dojde k ohrožení bezpečnosti uživatelského účtu. *[4, 5]*

1.1.2 Ochrana poskytovaná provozovatelem služby

Do ochrany poskytované provozovatelem služby patří veškerá opatření, kterými chrání provozovatel služby (např. sociální sítě), u které má uživatel vytvořený účet, uživatelovo heslo před manipulací či zneužitím neoprávněnou osobou. *[4]*

Mezi tato opatření patří například:

- antivirová ochrana serveru s databází uživatelských profilů
- ochrana serveru s databází uživatelských profilů před neoprávněnou manipulací.
- omezení neúspěšných pokusů o přihlášení (při *x* neúspěšných pokusech o přihlášení dojde k dočasné nebo úplné blokaci účtu.)
- zabezpečení veškerých aplikací s přístupem k databázi uživatelských účtů před zneužitím k získání neoprávněného přístupu do databáze uživatelských profilů.
- nastavení přístupových oprávnění k databázi uživatelských profilů pouze nezbytně nutným osobám, potřebným k administraci a správě databáze.
- způsob ukládání hesel. Ukládání hesel v podobě, v jaké je nebezpečné. Pokud by došlo k prolomení ochrany databáze uživatelských profilů, zjistil by útočník reálná hesla uživatelů. Z tohoto důvodu se hesla při vytváření účtu ukládají v pozměněné podobě. (Nejčastěji se k tomuto využívají různé hašovací funkce. Díky vlastnosti hašovacích funkcí, kterou je jednosměrnost, jsou jednou z nejvhodnějších metod.) Při následném přihlašování uživatele se jím aktuálně zadané heslo

zahašuje (stejnou metodou jako původně zadané heslo). Takto vzniklý haš, se následně porovná s dříve uloženým výstupem z hašovací funkce, patřícím k přihlašovanému účtu. V případě že výstupy jsou totožné, dojde k ověření identity a k přihlášení ke službě. *[4]*

1.2 Silné a bezpečné heslo

V předchozím textu bylo několikrát užito slovní spojení „silné a bezpečné“ heslo. Co si ale představit pod pojmem silné a bezpečné heslo. Silné a bezpečné heslo by mělo splňovat několik parametrů, které co nejvíce snižují útočníkovu šanci na prolomení nebo uhádnutí hesla. *[6]*

Aby byl uživatel schopen vytvořit silné a bezpečné heslo, je dobré se nejdříve seznámit se způsoby a metodami, pomocí kterých bude útočník uživatelem vytvořené heslo zkoušet překonat. Až když si uživatel uvědomí, jakých nedostatků ve skladbě hesla útočník využívá, je schopen vytvořit heslo, které bude odolné proti těmto útokům. *[6]*

1.2.1 Způsoby útoku

Jelikož je v dnešní době při přístupu k spoustě důležitých věcí potřeba znát patřičné heslo, stalo se překonávání („prolamování“) hesel jednou z důležitých činností při získávání neoprávněného přístupu. Na druhou stranu znalost způsobu překonávání hesla může být důležitou pomůckou při tvorbě silného a bezpečného hesla. Mezi nejpoužívanější útoky na heslo patří bezesporu „slovníkový útok“, „útok hrubou silou“, „recyklace hesel“, „sociální inženýrství“, „keylogery“, „zjištění hesla od uživatele“, „využití přednastavených přístupů“. *[6]*

1.2.1.1 Slovníkový útok

Jedná se o způsob útoku, při níž útočník využívá seznam často používaných, běžných, v minulosti uniklých nebo v minulosti používaných slov a hesel. Tomuto seznamu se říká slovník. Útočníkem zvolený program pak následně zkouší při přihlašování použít jedno slovo ze slovníku za druhým v naději, že napadený uživatel použil heslo, které je ve slovníku obsaženo. Při úspěšném přihlášení pak program vypíše název účtu a heslo, které umožnilo přihlášení. *[4, 6]*

Jako slovník se dá použít i takzvaná „rainbow table“ neboli „duhová tabulka“. Prakticky se jedná o tabulku o dvou sloupcích. První sloupec obsahuje generátorem vygenerované řady znaků, druhý sloupec obsahuje haše těchto řad. První sloupec se dá použít jako slovník při slovníkovém útoku. Druhý sloupec pak slouží útočníkovi ke zjištění původního slova k haši, který se nachází například v databázi uživatelských účtů poskytovatele nějaké služby. *[7]*

Jelikož lidé při tvorbě hesel dost často používají slova nebo jména běžného použití, bývá tento způsob útoku rychlý a úspěšný. *[4]*

Nastavením dočasného nebo úplného blokování účtu při určitém množství po sobě jdoucích neúspěšných pokusech o přihlášení, může poskytovatel služby částečně chránit uživatele před tímto způsobem útoku. *[6]*

1.2.1.2 Recyklace hesel

Při tomto způsobu útočník využívá „dříve prolomených“ hesel napadeného. To je umožněno nejčastěji neznalostí nebo leností napadeného uživatele. Tito uživatelé používají pro ověření identity u různých poskytovatelů služeb stejné jméno a heslo. V případě, že útočník zjistí jméno a heslo k jednomu z těchto účtů, získá zároveň přístup i k jeho ostatním účtům. Útočník si pak toto jméno a heslo uloží do slovníku účtů a hesel pro pozdější útok u jiných poskytovatelů služeb, případně hned vyzkouší různé jiné poskytovatele, zda u nich napadený uživatel nemá vytvořenou stejnou identitu se stejným heslem. *[6]*

V případě útoku cíleného na určitou osobu se při dalších útocích zkouší i různě upravené varianty jednou prolomeného jména a hesla. *[6]*

1.2.1.3 Sociální inženýrství

Tato metoda je nejčastěji využívána při cíleném útoku na určitý subjekt. Prvním krokem při tomto útoku je zjišťování informací o cíli útoku a jeho okolí. Tyto informace se následně využívají, při snaze uhádnou heslo, které cíl používá. *[8]*

Tato metoda vyžaduje určitý čas na získání informací o cíli, její úspěšnost však následně bývá vysoká, jelikož většina lidí nevytváří složité a špatně zapamatovatelné heslo. Daleko pohodlnější je pro ně vytvořit heslo z něčeho, co je jim blízké. Mezi tyto věci patří například datum narození, jména rodinných příslušníků nebo mazlíčků, jména nebo názvy

věcí či událostí, které jsou předmětem jeho zájmu. Čím útočník cíl lépe pozná, tím snadnější je pro něj heslo uhodnout. *[8]*

Sociálního inženýrství lze využít ale i u ostatních způsobů útoku. A nemusí se přitom jednat pouze o útok cílený na jeden subjekt. Díky různým průzkumům je známo, že většina uživatelů donucených použít kromě malých písmen i písmena velká, případně čísla. Umístí velké písmeno na první pozici a číslo dá na konec hesla. Další nápověda, kterou nám sociální inženýrství může poskytnout je záměna písmen a čísel, kterou lidé nejčastěji používají. Mezi tyto záměny patří například psaní „0“ místo „o“, „3“ místo „e“ či „4“ místo „a“. *[8]*

V sociálním inženýrství nemusíme nezbytně nutně zjišťovat informace o subjektu a jeho okolí. Útočník se může zaměřit na oklamání subjektu a následném vylákání informací a hesel, aniž by si subjekt uvědomil, že tyto údaje někomu poskytl. Mezi tyto metody patří například Phishing. *[8]*

1.2.1.3.1 Phishing (rybaření)

Výkladový slovník kybernetické bezpečnosti uvádí, že: *„Phishing je podvodná metoda, usilující o zcizování digitální identity uživatele, jeho přihlašovacích jmen, hesel, čísel bankovních karet a účtu apod. za účelem jejich následného zneužití (výběr hotovosti z konta, neoprávněný přístup k datům atd.). Vytvoření podvodné zprávy, šířené většinou elektronickou poštou, jež se snaží zmíněné údaje z uživatele vylákat. Zprávy mohou být maskovány tak, aby co nejvíce imitovaly důvěryhodného odesílatele. Může jít například o padělaný dotaz banky, jejíchž služeb uživatel využívá, se žádostí o zaslání čísla účtu a PIN pro kontrolu (použití dialogového okna, předstírajícího, že je oknem banky – tzv. spoofing). Tímto způsobem se snaží přistupující osoby přesvědčit, že jsou na známé adrese, jejímuž zabezpečení důvěřují (stránky elektronických obchodů atd.). Tak bývají rovněž velice často zcizována například čísla kreditních karet a jejich PIN.“ [1]*

Odvětvím phishingu je Spear phishing (rybaření oštěpem). Rozdíl mezi phishingem a spear phishingem je v zaměření. Zatím co phishing není cílen na určitý subjekt (cílí na kohokoliv, kdo není dost pozorný a znalý, aby se tomuto útoku ubránil), spear phishing je zaměřen na určitý subjekt nebo určitou skupinu subjektů (např. zaměstnance určité firmy). *[8]*

Před tímto způsobem útoku uživatele silné a bezpečné heslo neochrání. V některých případech mohou pomoci i různé nástroje (používání softwaru, které umožňují detekci těchto útoků nebo například bankou vydaný software pro bezpečné připojení k jejím službám). *[8]*

Ve většině případů je však jako základní prvek obrany před phishingovým útokem uživatelovo povědomí o možné hrozbě. V případě menších kolektivů, jako jsou např. zaměstnanci určité firmy, lze toto povědomí získat cestou různých školení a přednášek. V případě potřeby informování široké veřejnosti lze využít různých médií (televize, rádio, internet). Je zároveň nutné, aby informace, kterými se snažíme zvýšit povědomí lidí o těchto hrozbách, byly podávány formou, v které jsou lidé schopni je pochopit. Ve většině případů lze předpokládat, že osoby, které informujeme, nejsou IT odborníci, nýbrž lidé kteří příliš informačním technologiím nerozumí. *[8]*

U uživatelů, kteří mají povědomí o možné hrozbě, už záleží jen na jejich ochotě, pozornosti a opatrnosti. *[8]*

1.2.1.4 Keyloggery

Keylogery jsou počítačové programy, které snímají stisky kláves, ukládají je a v některých případech následně uložené soubory odesílají na určené úložiště případně emailové adresy. Prakticky se jedná o skryté sledování činnosti klávesnice. Tyto programy jsou ve většině případů antivirovými programy vyhodnocovány jako viry. *[6]*

Ani před tímto způsobem útoku uživatele sebesilnější heslo neochrání. Proto se tomuto způsobu útoku nebudeme více věnovat. *[6]*

1.2.1.5 Zjištění hesla od uživatele

Tato metoda sice nemá moc společného s kybernetickými způsoby překonávání, nicméně nikdy by člověk neměl podceňovat účinnost hrubého násilí. Tento způsob by mohl být pojmenovaný různě. Prakticky se jedná o způsob, při kterém uživatel sdělí útočníkovi své heslo v důsledku projevu fyzického násilí případně jeho pohrůžkou vůči osobě uživatele nebo osobám jemu blízkým. Samozřejmě, že v určitých případech psychický nátlak funguje stejně dobře. Před tímto útokem uživatele nezachrání sebebezpečnější heslo a proto se jím dále již nebudeme zabývat. *[8]*

1.2.1.6 Využití přednastavených přístupů

Tento způsob útoku využívá výrobcem přednastavené administrátorské účty. Výrobce vytváří tyto účty za účelem jednoduššího prvotního přihlášení uživatele, při kterém by si uživatel měl změnit v optimálním případě jméno účtu a heslo, v horším případě alespoň heslo. Útočníci pak využívají skutečnosti, že jsou uživatelé, kteří tyto defaultně nastavené hodnoty nepřenastaví a nechávají je tak, jak jsou. Jelikož jsou defaultně nastavené hodnoty zveřejňovány na stránkách výrobce, je jejich zjištění útočníkem velmi snadné. Mezi tato zařízení patří například síťové routery. *[8]*

1.2.1.7 Útok hrubou silou

Na rozdíl od předchozího útoku, zde nepůjde o fyzický útok na osobu uživatele, i když k tomu název zavádí. Tento způsob útoku využívá generátoru hesel. Útočníkův program generuje hesla jako kombinaci možných variant písmen, čísel a znaků dle určitého systému a ty pak zkouší použít při přihlášení. *[6]*

Útoky hrubou silou jsou ve většině případů dost nešikovné (z důvodu jejich časové a výpočetní náročnosti), se sporadickou mírou úspěšnosti. Jejich úspěch závisí na mnoha faktorech. Důležitým faktorem je například výpočetní výkon útočníkova počítače (či sítě počítačů), délka a složitost hesla, nebo alespoň částečná znalost struktury hesla (jeho délka, použité znakové sady, apod.). *[6]*

Například pokud útočník ví, že heslo má 6 míst a je složeno pouze z čísel, je použití této metody rychlé a úspěšné. Pokud tyto znalosti nemá a nemá k dispozici nějaký superpočítač či velkou síť počítačů, bývá útok hrubou silou použit až jako poslední možnost v naději že uživatelovo heslo je krátké a jednoduché. Doba překonávání hesla pomocí tohoto způsobu je závislá hlavně na délce hesla a množství použitých znakových sad. Čím je heslo delší a složitější, tím je doba potřebná na jeho překonání hrubou silou větší. *[6]*

Nutno podotknouti, že proti tomuto způsobu může uživatele částečně chránit i poskytovatel služby. Například nastavením dočasného nebo úplného zablokování uživatelského účtu při určitém počtu po sobě jdoucích, neúspěšných pokusech o přihlášení nebo použitím přihlašovacích scryptů. *[6]*

1.2.2 Tvorba silného a bezpečného hesla

Při tvorbě hesla je nutno si uvědomit základní vlastnosti hesla. Mezi vlastnosti hesla patří složitost, délka a entropie hesla. Tyto vlastnosti pak vytvářejí sílu hesla. *[6]*

1.2.2.1 Složitost hesla

V každém jazyce existují tzv. znakové sady. Je důležité si uvědomit, jaký je rozdíl mezi jazykovou sadou a znakovou sadou. Jazyková sada je sada slov a výrazů používaných v daném jazyce. Znakové sady jsou skupiny znaků. Každý jazyk může mít jiné jazykové sady. Znaky využívané českým jazykem bychom mohli rozdělit do několika skupin. viz. Tabulka 1. Čím více znakových sad je v hesle zastoupeno, tím je heslo složitější. *[4]*

Tabulka 1: Znakové sady. *Zdroj: vlastní*

název	znaky v sadě	počet znaků
a-z	a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z	26
A-Z	A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z	26
0-9	0, 1, 2, 3, 4, 5, 6, 7, 8, 9	10
a-z s diakritikou	á, č, ď, é, í, ň, ó, ř, š, ť, ú, ů, ý, ž	14
A-Z s diakritikou	Á, Č, Ď, É, Í, Ň, Ó, Ř, Š, Ť, Ú, Ů, Ý, Ž	14
speciální znaky	. , : ; - ? ! @ # $ & ^ _ * ( ) [ ] ~	19

Je potřeba upozornit, že speciálních znaků je daleko víc, ale ostatní nejsou dost často v heslech podporovány. Stejně tak se nedoporučuje používat znaky s diakritikou, jelikož nemusí být poskytovatelem služby při tvorbě hesla podporovány. (např. při zakládání emailového účtu na www.seznam.cz nejsou v heslech podporovány znaky s diakritikou nebo „§“) *[9]*

Z výše uvedené tabulky je patrné, že i při vynechání znaků s interpunkcí nám zůstává více jak 80 možných znaků, které můžeme při tvorbě hesla použít. Množství zastoupených znakových sad v hesle pak následně ovlivňuje sílu hesla. *[9]*

Když budeme mít heslo o pěti znacích, které budou náležet pouze k sadě 0-9, je zde možných 10^5 kombinací. To je 100 000 variant hesla. Pokud v tom samém hesle použijeme sady 0-9, a-z, A-Z a speciální znaky, bude počet možných kombinací 81^5. To je celkem 3 486 784 401 variant hesel. *[9]*

1.2.2.2 Délka hesla

Délka hesla je prakticky počet znaků, které heslo má. Délka hesla je velice důležitým faktorem ovlivňujícím sílu hesla. Čím je heslo delší, tím lépe. Nicméně jak dlouhou řadu znaků je schopen si běžný uživatel zapamatovat? *[4]*

Například když budeme mít heslo o pěti znacích a použijeme znaky ze znakových sad „0-9“ a „a-z“, vznikne 36^5 možných kombinací. To je 60 466 176 variant hesel. Pokud bychom měli heslo o deseti znacích s použitím stejných jazykových sad, vznikne 36^{10} možných kombinací, což je 3 656 158 440 062 976 variant hesel. Porovnání možných kombinací nám ukazuje, že délka hesla je jedním z faktorů, který ovlivňuje kvalitu a sílu hesla. *[4]*

Je důležité si ale v souvislosti s délkou hesla uvědomit několik podstatných skutečností: *[4]*

- Dlouhé heslo nemusí vždy znamenat to samé jako bezpečné heslo, i když je to jeden z předpokladů pro něj. Pokud budeme mít heslo, které sice bude dlouhé, ale zároveň to bude běžné slovo objevující se v slovnících používaných pro slovníkové útoky, nemůže být takové heslo bráno jako bezpečné. Dobrým příkladem je třeba dvanáctimístné slovo „Popocatepetl“. 12 míst by v současné době mohlo být označeno za dostatečnou délku pro bezpečné heslo. Nicméně při slovníkovém útoku by toto heslo moc dlouho neobstálo. Proto není možné toto heslo označit za bezpečné, i když má dostatečnou délku. *[4]*
- Při dodržování všech bezpečnostních pokynů na tvorbu a používání hesel může být délka hesla problémem pro paměť běžného uživatele. Pamatovat si jedno dlouhé heslo nemusí být nic těžkého. Když si ale uvědomíme, že uživatel má několik různých účtů u různých poskytovatelů, přičemž by u každého účtu měl mít jiné heslo. Každé z těch hesel by mělo být dost dlouhé a dost složité. Každé by mělo být v pravidelných intervalech měněno. Zároveň by žádné z těch hesel nemělo být uživatelem v minulosti použité. Je snadné si představit, že běžný uživatel může mít problém si to všechno pamatovat. Při snaze vytvořit a pamatovat si dlouhá hesla, tak může docházet ke snižování kvality z důvodu snadnějšího zapamatování. *[4]*

1.2.2.3 Entropie hesla

Entropie je pojem, který se využívá v mnoha odvětvích, například ve fyzice, matematice nebo informatice a je úzce spojena s pravděpodobností výskytu určitého jevu. Nejdříve bychom si měli vysvětlit co to vlastně entropie je. Entropie je míra neurčitosti prvků určitého systému. Čím je vyšší pravděpodobnost výskytu prvku v systému, tím je nižší entropie systému. V informatice platí, že čím vyšší entropie zprávy, tím je nižší informační hodnota zprávy. *[9, 10]*

Nás bude zajímat především informační entropie. C. E. Shannon ji vyjádřil vzorcem:

$$H = -K \sum P_i \, ln \, P_i \qquad (1)$$

kde *H* je Shannonova entropie, *K* je integrační konstanta, *P* je pravděpodobnost výskytu a *i* je určitý prvek. V informatice je integrační konstanta vyjádřena jako: *[9, 10]*

$$K = \frac{1}{\ln 2} \qquad (2)$$

Po dosazení do předchozí Shannonovi rovnice nám vyjde rovnice: *[10]*

$$H = \left(-\frac{1}{\ln 2}\right) \sum P_i \ln P_i = -\sum P_i \left(\frac{\ln P_i}{\ln 2}\right) = -\sum P_i \, log_2 \, P_i \qquad (3)$$

Jednotkou entropie je 1 bit. Je to zároveň i nejmenší jednotka informace. Je to například odpověď „Ano“ „Ne“. Dva možné stavy, které mají stejnou pravděpodobnost výskytu. 1 bit je definován jako: *[9, 10]*

$$log_2 \, 2 = 1 \, bit \qquad (4)$$

Nejvyšší míra entropie systému je v okamžiku, kdy všechny prvky systému mají stejnou pravděpodobnost výskytu. To je prakticky i případ hesla. Pokud mají všechny prvky stejnou pravděpodobnost výskytu, je entropie soustavy vyjádřena rovnicí: *[9, 10]*

$$H = log_2 \, s \qquad (5)$$

Kde *H* je entropie, *s* je množství prvků, které se mohou vyskytnout. Musíme si ale uvědomit, že v případě hesla je soustavou myšlena jedna pozice z textového řetězce. Pokud tedy budeme chtít vyjádřit hodnotu entropie celého hesla, musíme rovnici pro výpočet upravit do tohoto tvaru: *[9, 10]*

$$H = n \, log_2 \, s \qquad (6)$$

Kde *H* je entropie, *n* je počet pozic (délka hesla), *s* je počet znaků, které se mohou na každé pozici vyskytnout. *[9, 10]*

Za použití výše uvedené tabulky znakových sad a výše uvedeného vzorce není problém zjistit entropii hesla. Čím vyšší bude hodnota entropie hesla, tím silnější by heslo mělo

být. Je zde opět nutné podotknout, že ani entropie není jednoznačným ukazatelem bezpečnosti hesla. I když nám může dost napovědět. Důvodem pro to je skutečnost, že entropie počítá s délkou a rozložením znaků, nicméně neposuzuje hesla v rámci odolnosti vůči různým slovníkovým útokům. Jelikož heslo s vysokou entropií, ale běžně se vyskytující v slovnících používaných při slovníkovém útoku, nemůže být považováno za bezpečné heslo. *[9]*

1.2.2.4 Vztah mezi vlastnostmi hesla

Po seznámení se se základními vlastnostmi hesla se můžeme věnovat vztahem mezi jednotlivými vlastnostmi. Asi nejdůležitějším vztahem je závislost entropie hesla na délce a složitosti hesla. Jelikož entropie nám dává vodítko pro posouzení síly hesla. *[9]*

Jsou si však délka hesla a složitost hesla rovnými partnery? Můžeme to zkusit zjistit na příkladu. Pro zachování rovnosti vztahů, potřebné pro posouzení míry vlivu jednotlivých vlastností, použijeme jako základ 26 znaků a 5 míst. První heslo bude mít délku 5 míst a použité dvě znakové sady o 52 možných znacích. Druhé heslo bude mít 10 míst a použitou znakovou sadu o 26 možných znacích. Pro jednodušší znázornění, použijeme slova „Heslo“ a „hesloheslo“. V praxi by ani jedno z těchto hesel nemohlo být považováno za bezpečné. V našem příkladu však nejde o slovo jako takové, ale pouze o řadu znaků požadované délky a složitosti. Pro porovnání vlivu délky a složitosti hesla na jeho bezpečnost, nám tato slova budou stačit. *[9]*

Nejdříve porovnáme počet možných variant, které lze vytvořit v závislosti na parametrech hesel. *[9]*

První heslo má 5 míst a 52 možných znaků. Počet variant hesla je:

$$52 \; x \; 52 \; x \; 52 \; x \; 52 \; x \; 52 = \; 52^5 = 380\,204\,032$$

Druhé heslo má 10 míst a 26 možných znaků. Počet variant hesla je:

$$26 \; x \; 26 \; x \; 26 \; x \; 26 \; x \; 26 \; x \; 26 \; x \; 26 \; x \; 26 \; x \; 26 \; x \; 26 = \; 26^{10} = 141\,168\,095\,653\,376$$

Zatím co použití parametrů prvního hesla umožňují vytvoření více jak 380 milionů variant hesla. Parametry druhého hesla (dvojnásobná délka a poloviční složitost) umožňují vytvoření více jak 141 bilionů variant hesla. *[9]*

Výpočtem entropie prvního a druhého hesla pak můžeme získat další indicie pro posouzení vlivu těchto vlastností. *[9]*

Entropie hesla „Heslo“:

$$H = n\ log_2\ s = 5\ x\ log_2\ 52 = 5\ x\ 5{,}7004 = 28{,}502\ bit$$

Entropie hesla „hesloheslo“:

$$H = n\ log_2\ s = 10\ x\ log_2\ 26 = 10\ x\ 4{,}7004 = 47{,}004\ bit$$

Porovnání výsledků výpočtu entropie a možných variant výše uvedených hesel nás vede k závěrům, že délka hesla ovlivňuje entropii hesla a tím i jeho sílu z větší míry nežli složitost hesla. Zároveň ale není pochyb o tom, že i složitost hesla se podílí na míře entropie a síle hesla. *[9]*

Tabulka 2: Entropie jedné pozice hesla v závislosti na použitých znakových sadách. *Zdroj: vlastní*

použité znakové sady	**počet možných znaků**	**entropie jedné pozice hesla**
0-9	10	3,3219 bit
a-z	26	4,7004 bit
a-z, 0-9	36	5,1699 bit
a-z, A-Z, 0-9	62	5,9542 bit
a-z, A-Z, 0-9, spec. znaky	81	6,3399 bit

Při prozkoumání výše tabulky č. 2 zjistíme, že při použití znakových sad „a-z“, „A-Z“, „0-9“ a spec. znaky (což nám umožní použití 81 možných znaků) nedosáhneme ani dvojnásobné entropie v porovnání s nejslabší znakovou sadou „0-9“ (pouhých 10 možných znaků). *[9]*

Při hlubším zamyšlení nad tímto vztahem, může se někdo pozastavit nad faktem, že pokud je vše výše uvedené pravda, jak můžou banky věřit čtyřmístnému pinu u platebních karet a označovat ho za dostatečný způsob ověření identity. Pravdou je, že i přes skutečnost, že čtyřmístný pin má pouze 10 000 možných variant (šance na úspěch se třemi pokusy je 0,03 %), je toto dostatečně bezpečný způsob ověření uživatele. Má to prostý důvod. Banka má možnost při třech chybných pokusech kartu zablokovat. To útočníkovi, který nemá žádné povědomí o číselném sestavení vašeho pinu, dává minimální šanci na úspěch. Samozřejmě v prostředí, kdy by nebyl počet pokusů o autentizaci nijak limitován, by pin s entropií 13,2876 bitu byl naprosto nedostatečným. *[9]*

1.2.2.5 Posouzení hesla vzhledem k způsobům útoků

Slovníkový útok cílí na jednoduchá hesla, která jsou složena pouze jedním slovem, běžným v jazyce uživatele. Mezi takováto hesla patří například jména, příjmení, věci nebo názvy. (*Adam, Novák, heslo, Audi, apod.*) Upravené slovníky mohou obsahovat i pro hesla často používané kombinace. (*Heslo123, H3sl0 apod.*) *[6, 8]*

Útok hrubou silou cílí na krátká a jednoduchá hesla. Čím kratší a jednodušší heslo, tím kratší čas bude generátoru trvat, nežli se k němu dopracuje. Pokud budeme mít heslo „Heslo“, které má 5 pozic a byly použity znakové sady a-z a A-Z s maximálním počtem znaků 52 potřebuje útočník vygenerovat 52^5 hesel (což je 380 204 032 hesel). *[6]*

Útok za pomocí recyklace hesel. Zde je potřebné si uvědomit, že se jedná o útok, který využívá uživatelova hesla, která byla překonána buď v minulosti, nebo v současnosti u jiného provozovatele služby, nežli na kterou útočník útočí. K ochraně před tímto útokem stačí mít pro každou službu nastavené jiné přístupové heslo. A nevyužívat heslo použité u jiné služby, i když ji už uživatel delší dobu nevyužívá. *[6]*

Při tvorbě hesla si musí uživatel uvědomit, že použitím slova či slovní spojení, které je pro něj typické, vznikne heslo, které se sice dobře pamatuje, zároveň je velmi snadno překonatelné pomocí sociálního inženýrství. Stejný problém je například při použití jmen členů rodiny, případně dat narození, svátků, svatby či dalších významných událostí členů rodiny. *[8]*

1.2.2.6 Vytvoření bezpečného hesla

Díky výše uvedeným informacím by měl být uživatel schopen vytvořit dostatečně bezpečné heslo. Shrneme si ještě jednou, co je k tomu zapotřebí. *[4, 9]*

- Je nutné, aby heslo bylo dostatečně dlouhé. Termín dostatečně dlouhé je dosti nejednoznačný. Většina poskytovatelů má nastavený minimální požadavek na délku hesla 8 znaků. Tento požadavek je takto nastaven už skoro 10 let. V době, kdy s tímto minimálním limitem poskytovatelé přišli, s ohledem na výpočetní výkony počítačů, mohl být tento limit dostačující. Osobně se domnívám, že bezpečné heslo by nemělo mít méně jak 12 znaků. Samozřejmostí je čím delší tím lepší. A to i s přihlédnutím k faktu, že čím delší heslo, tím větší problémy by mohl mít člověk si ho zapamatovat. *[4, 9]*

Jako drobná pomůcka pro vytvoření a zapamatování si dlouhého a bezpečného hesla může být tvorba hesla za pomoci několika slov oblíbené písničky či citátu nebo několika významově nesouvisejících slov (např. spojením slov „motor“, „veverka“, „zelený“, „oběd“ můžeme vytvořit heslo „motorveverkazelenyobed“, které i za použití pouze jedné znakové sady bude mít entropii 103,4088 bit a díky naprosto šílenému spojení různých slov by vznikla hloupost, která by se uživateli i dobře pamatovala. Takto vytvořené heslo v bezpečnosti dalece překoná jakékoliv dvanáctimístné heslo, které bychom vytvořili za použití všech 4 znakových sad. Toto heslo by mělo entropii jen 76,0788 bit.) *[9]*

- K zvýšení síly a bezpečnosti hesla můžeme využít kombinace různých znakových sad. Důležité je si uvědomit, že čím složitější heslo, tím hůře se může pamatovat a zadávat. Pro snadnější zapamatování si složitého hesla je dobré si vytvořit systém skladby hesla. V tomto systému je pak dobré vyhnout se oblíbeným zvyklostem většiny uživatelů a to např. první písmeno psát velké nebo čísla psát nakonec hesla. *[4, 9]*
- Při tvorbě hesla za pomoci jednoho slova, sady čísel či slovního výrazu, snažit se nepoužívat jako vzor věci, které jsou pro nás typické, nebo k nim máme blízko. *[9]*
- Heslo by nemělo být stejné, nebo lehce pozměněné, jako heslo již používané při přihlašování k jinému účtu. (např. u jednoho poskytovatele služby mít heslo „Borivoj1“ a k dalšímu „Borivoj2“) *[4]*

Při dodržení těchto jednoduchých pravidel by uživatel měl být schopen sám vytvořit silné heslo. Takto vytvořit bezpečné heslo je celkem jednoduché a není to ani moc namáhavé. Jediný, možná ne zcela, viditelný nedostatek je, do kdy budou tato pravidla platit. Kdy takto vytvořené heslo bude ještě bezpečné a kdy už jím být přestane. Vývoj výpočetní techniky jde tak rychle dopředu, že obyčejný člověk, který denně nesleduje nejnovější trendy v IT, nemá šanci zjistit, že jeho hesla už nejsou dost bezpečná. Proč tedy pro tvorbu hesla nepoužít nástroj nebo program, který by vytvořil bezpečné heslo, jenž by i podle předpovědi expertů mělo být bezpečné ještě několik dalších let. *[4, 9]*

Jako jednu z možných metod, které by nám mohli s tímto úkolem pomoci, by bylo využití hašovacích funkcí. Jako jeden a nikoliv jediný z důvodů, proč se hašovací funkce k tvorbě hesla nevyužívá, leží v naprosté nemožnosti běžného uživatele si zapamatovat celý textový řetězec, který hašovací funkce vytvoří. Když se člověk koukne na problém z druhé

strany, potřeba mít při přihlášení patřičný program, by mohla povýšit ověřování identity z jednostupňového (prokázat znalost) na dvojstupňový (vlastnictví potřebného předmětu, prokázání znalosti). Dalším důvodem proč nepoužívat hašovací funkci při tvorbě hesla, může být fakt, že při pouhém použití hašovací funkce by útočník se znalostí způsobu tvorby našeho hesla nemusel mít moc problémů naše heslo prolomit. Na druhou stranu pokud by heslo nebylo tvořeno čistým hašem, ale haš by sloužil jako základ pro tvorbu hesla, mohla by metoda tvorby hesla založena na použití hašovací funkce být účinnou metodou. *[4]*

Osobně se domnívám, že tato metoda stojí za bližší prozkoumání. Nejdříve si však přiblížíme, co to vlastně hašovací funkce je, jak funguje, jaké má vlastnosti atd.

2 HAŠOVACÍ FUNKCE

Hašovací funkce je kryptografická metoda. Prakticky se jedná o matematické algoritmy, jejichž pomocí se vytváří „otisk“ určité informace. Hašování je převod vstupních dat neomezené velikosti na relativně malé číslo konstantní velikosti. Tento otisk bývá označován také jako hash (česky haš nebo heš). *[4, 11]*

Hašovacích funkcí máme několik druhů a verzí. Mezi nejznámější funkce patří MD5, SHA1, SHA2 a SHA 3. *[4]*

2.1 Vlastnosti hašovací funkce

Kvalitní a bezpečná hašovací funkce musí mít určité vlastnosti

- Konstantní velikost haše
- Reagování na změny na vstupu (lavinový efekt …)
- Jednosměrnost
- Bezkoliznost *[4, 11]*

2.1.1 Konstantní velikost haše

Hašovací funkce by měla ze vstupních dat jakékoliv délky vytvořit haš konstantní velikosti. Například MD5 vytváří haš o velikosti 128 bitů. SHA-1 vytváří haš o velikosti 160 bitů. Další skupinou jsou funkce označovány SHA-2, které každá vytváří haš o jiné velikosti. Do skupiny SHA-2 patří SHA-228, SHA-256, SHA-338, SHA-512. Velikost haše těchto funkcí je udána v názvu funkce. Do skupiny SHA-3 patří čtyři kryptografické hašovací funkce SHA3-224, SHA3-256, SHA3-384 and SHA3-512. *[4]*

Například pokud budeme hašovat funkcí MD5 písmeno:

„a“ → ***„0cc175b9c0f1b6a831c399e269772661“***

Pokud bychom hašovali text:

„Hašovací funkce by měla ze vstupních dat jakékoliv délky vytvořit haš konstantní velikosti.“ → ***„e82c9cf85de2e947e5c3fba5fd8b0e53“***

Zde je vidět, že je jedno, zda se hašuje pouze jedno písmeno nebo dlouhý text. Na výstupu je vždy haš o stejné velikosti. *[4, 12]*

2.1.2 Reagování na změny na vstupu (lavinový efekt)

I nepatrná změna na vstupu musí vyvolat velkou změnu na výstupu. Toto je jedno ze základních pravidel hašovacích funkcí. *[12]*

Například když budeme hašovat funkcí MD5 slovo:

„**heslo123**“ → „***6a284155906c26cbca20c53376bc63ac***“.

Pokud bychom ale hašovali slovo:

„**Heslo123**“ → „***29d847ffce86b63c39a756a25b198751***“.

Zde je vidět, že i změna jednoho písmena z malého na velké vyvolala zásadní změnu v haši. *[12]*

2.1.3 Jednosměrnost

Hašování je jednosměrná kryptografická metoda. V praxi to znamená, že je jednoduché zahašovat původní data a vytvořit z nich haš. Na druhou stranu vytvořit z haše původní data je výpočetně prakticky nemožné. *[4, 12]*

V současné době lze na internetu najít velké množství stránek, které umožňují „dekryptaci“ hašů různých hašovacích funkcí. Princip těchto stránek, ale nespočívá v dekryptaci haše (rekonstrukci vstupních dat z haše). Tyto stránky mají přístup k rozsáhlým databázím slov a jejich hašů nebo můžou používat např. rainbow tables. V těchto databázích nebo tabulkách pak následně program vyhledává shodu. V případě nálezu pak vrátí uživateli původní výraz, ze kterého haš vznikl. *[12]*

Jako příklad těchto stránek mohu uvést například md5hashing.net, které umožňují „dekryptaci“ hašů více jak 15ti druhů hašovacích funkcí. Tyto stránky zároveň umožnují i online hašování textových řetězců. Další stránky jsou například hashkiller.co.uk, které umožňují „dekryptaci“ 3 druhů hašovacích funkcí, nebo md5online.org, které se zaměřují pouze na funkci MD5. *[4]*

2.1.4 Bezkoliznost

Kolize je stav, kdy hašovací funkce vytvoří ze dvou různých datových vstupů stejný haš. Jelikož z neomezeně velkých vstupních dat vytváříme pomocí hašovacích funkcí menší množství dat přesně dané velikosti, je logické, že takovýchto kolizí je velké množství. *[11]*

K tomuto závěru nás vede i tzv. „Narozeninový paradox“. *[11]*

Tento paradox nám vysvětluje, že k tomu abychom měli 50% šanci na nalezení dvou různých lidí, kteří mají ve stejný den narozeniny, nám stačí 23 osob. Prakticky nám narozeninový paradox říká, že na nalezení kolize u *n* bitové hašovací funkce nám stačí pouze $2^{\frac{n}{2}}$ vstupních zpráv. Z toho vyplývá například, že k nalezení náhodné kolize s 50% šancí u hašovací funkce MD5 (128 bitů) nám stačí $2^{\frac{128}{2}} = 2^{64}$ zpráv. Což činí více jak 18,446 744 trilionu zpráv ($18{,}446\ 744\ x\ 10^{18}$). *[11]*

Pro bezpečnost hašovací funkce je ale důležitá tzv. bezkoliznost:

- Jako bezkoliznost 1. řádu někteří označují stav, kdy je výpočetně nemožné najít dvě různé zprávy *Z* a *Z´* kde by platilo: *haš(Z) = haš(Z´)*.
- Jako bezkoliznost 2. řádu pak označují stav, kdy je výpočetně nemožné najít k zprávě *X* zprávu *Y*, kde by platilo $Y \neq X$ a zároveň *haš(X) = haš(Y).* *[11]*

U bezkoliznosti 2. řádu tedy nehledáme jakoukoliv kolizi, kde nám teoreticky na cca 50% šanci na nalezení kolize stačí zahašovat $2^{\frac{n}{2}}$ zpráv, ale hledáme kolizi k určitému haši, kde už potřebujeme pro stejnou šanci 2^n zpráv (což u MD činí cca $3{,}402\ 823\ x\ 10^{38}$ zpráv). *[11]*

2.2 Využití hašovacích funkcí

Hašovací funkce jsou v současné době využívány různými způsoby. Můžou být využívány pro ověřování pravosti, porovnávání nebo pro vyhledávání dat. Velmi často jsou užívány i pro zabezpečování uživatelských hesel. *[11]*

2.2.1 Ověřování pravosti dat

Hašovací funkce jsou výrazným pomocníkem, při ověřování pravosti dat. Například při stahování instalačního softwaru od výrobce, dojde zároveň i ke stažení jeho haše. Po stažení se instalační software zahašuje, následně dojde k porovnání obou hašů. Pokud haše souhlasí, software je pravý a nikdo neoprávněný s ním nemanipuloval. Pokud haše nesouhlasí, bylo se softwarem manipulováno a může být bezpečnostní hrozbou pro náš systém a naše data. *[11]*

K ověření pravosti můžou být hašovací funkce použity například i při šifrované komunikaci s využitím digitálního podpisu. Při tomto jsou využívány metody symetrického a asymetrického šifrování a hašovací funkce. Symetrické šifrování využívá k šifrování a dešifrování jeden klíč. Je to rychlá metoda, jejíž nedostatkem je složitější klíčová správa. Oproti tomu asymetrické šifrování používá dvou klíčů (veřejný klíč a soukromý klíč). Je to pomalejší metoda s jednoduchou klíčovou správou. Při výše uvedené komunikaci je vytvořen haš původní zprávy. Následně je pak text zprávy zašifrován symetrickou šifrou, symetrický klíč je zašifrován veřejným klíčem příjemce a haš je zašifrován soukromým klíčem odesílatele. Následně dojde k odeslání těchto tří věcí (zašifrovaný haš, zašifrovaný symetrický klíč, zašifrovaný text zprávy). Příjemce svým soukromým klíčem rozšifruje symetrický klíč, kterým pak rozšifruje text zprávy. Veřejným klíčem odesilatele (ověření, že odesílatel je ten za koho se vydává) dešifruje haš. Tento haš se porovná s hašem, který vznikne po zahašování rozšifrovaného textu. Pokud jsou oba haše shodné, tak zpráva nebyla pozměněna v průběhu přenosu mezi odesilatelem a příjemcem. *[4, 13]*

2.2.2 Porovnávání dat

Každý s počítačem pracující člověk určitě zažil situaci, kdy po zkopírování dat z jednoho místa na druhé si nebyl zcela jist, zda se zkopírovalo opravdu vše. V podmínkách běžného uživatele, který většinou kopíruje přijatelné množství dat, je zběžná kontrola otázkou pár vteřin a kliknutí. *[11]*

Jelikož základní vlastností hašovacích funkcí je změna jakkoliv velkých objemů dat na relativně malý haš. Zároveň s tím jakákoliv nepatrná změna ve vstupních datech vyvolá velkou změnu v haši. Můžeme místo porovnávání původních a zkopírovaných dat, porovnávat jejich haše a případné drobné změny budou snadno patrné. *[11]*

Porovnávání hašů nemusí sloužit pouze pro kontrolu celistvosti dat. Může být využita například i pro vyhledávání duplicit. *[11]*

2.2.3 Vyhledávání dat (indexace)

Hašování může být využito jako rychlý způsob vyhledávání a přístupu k datům. Jednotlivá data se zahašujou a haše se umístí do tabulky spolu s údaji o umístění původních dat. Při vyhledávání se parametry dotazu zahašují a následně porovnávají s haši v tabulce. Při shodě pak vyhledávač přejde díky indexům uvedeným u haše na místo původních dat. *[11]*

Podobného principu využívají například antivirové programy, které ukládají do databáze haše potenciálně nebezpečných programů. Při příchodu nových dat, dříve nežli dovolí antivirový program počítači s daty jakkoliv pracovat, tato data zahašuje a porovná s databází. V případě, že najde shodu, antivir tato data nebo program zablokuje, aby zamezil potenciální hrozbě. *[11]*

2.2.4 Ukládání uživatelských hesel

V rámci správy hesel uživatelů se využívá hašovacích funkcí za účelem ochrany uživatelských hesel před potencionálním útočníkem. Při prvním zadávání, když uživatel zadává své heslo, je toto heslo zahašováno. Výsledný haš je pak uložen do databáze pro pozdější ověřování uživatele. Při dalším přihlašování uživatele je pak opět, jím aktuálně zadávané, heslo zahašováno. Tento haš je pak porovnán s hašem uloženým v databázi u daného uživatele. Při shodě dojde k ověření uživatele a k jeho přihlášení. *[11]*

V případě že by se útočníkovi podařilo dostat do databáze s hesly, nevidí zde hesla zadávaná uživateli, ale pouze jejich haše. To je výhoda, které může poskytovat heslům vysokou ochranu: *[11]*

Při zadání získaného haše při přihlašování by došlo k vygenerování haše, který by neodpovídal haši uloženému v záznamu uživatele, a proto by nedošlo k přihlášení. Jelikož jednou z podmínek hašovacích funkcí je jednosměrnost, získat původní, uživatelem zadané heslo, je prakticky nemožné. Tedy pokud pomineme existenci databází hašů, případně rainbow tabulek, apod., kde se dají v případě slabších hesel dohledat některé původní textové řetězce k zde uloženým hašům. *[11]*

2.3 Bezpečnost hašovacích funkcí

Bezpečnost hašovacích funkcí je to nejdůležitější, co nás u hašovacích funkcí zajímá. Ale co to vlastně ta bezpečnost hašovací funkce je.

Jednou z podmínek bezpečné hašovací funkce je její dostatečná odolnost vůči nalezení kolizí 1. řádu. Přeloženo do normální řeči: Za bezpečnou se považuje ta funkce, u které je výpočetně nezvládnutelné nalezení náhodné kolize z dvou různých vstupů. V okamžiku, kdy je nalezen způsob, jakým v reálném čase jednoduše vytvořit kolizi, přestává být hašovací funkce bezpečná. *[12]*

Další podmínkou pro bezpečnou hašovací funkci je její jednosměrnost. Bezpečná hašovací funkce musí vytvářet takové haše, ze kterých je výpočetně nezvládnutelné získat vstupní data. V okamžiku, kdy je nalezen způsob, jakým v reálném čase jednoduše získat z haše původní data, přestává být hašovací funkce bezpečná. *[4, 12]*

Tyto dvě podmínky tvoří základ pro posuzování bezpečnosti hašovacích funkcí. Například u hašovacích funkcí MD-2, MD-4, MD-5, SHA-0, SHA-1 již byla prolomena první podmínka. U těchto funkcí byl vyvinut způsob jednoduché generace kolizí 1. nebo i 2. řádu. *[4, 12]*

Nutno ještě podotknout, že i když je hašovací funkce překonaná, dá se stále využívat například k vyhledávání.

2.4 Nejběžnější druhy hašovacích funkcí

Od vzniku prvních hašovacích funkcí (1953), prošli hašovací funkce určitým vývojem. Docházelo k prolamování funkcí (byl nalezen způsob jak v reálném čase vytvořit kolizi k určitému haši), čímž dané funkce přestaly být bezpečné. Zároveň vznikaly funkce nové nebo modifikované verze předchozích funkcí. V současné době je mnoho hašovacích funkcí. Mezi nejznámější hašovací funkce patří MD5 (Message-digest algorithm) a SHA-X (Secure hash algorithm). Další funkce můžou být například Whirpool, Ripemd, Tiger, Ghost, Adler, Snefru, Cro. *[11]*

V této práci bych chtěl přiblížit vývoj dvou nejznámějších MD5 a SHA-X.

2.4.1 Message-digest algorithm

Hašovací funkce MD5 vytvořil L. R. Rivest a publikoval v roce 1992. Měla nahradit MD4, která už nebyla bezpečná. MD5 vytvoří 128-bitový haš, který se výstupu zobrazí jako 32-místný hexadecimální řetězec. V roce 2005 byl představen algoritmus, jakým lze vytvořit dva certifikáty X.509 se stejným hašem na běžném počítači. V roce 2008 byl pomocí kolizí vytvořen falešný certifikát SSL, ten umožňuje následně vytvořit další certifikáty. V současné době je algoritmus MD5 překonaný a tak není jeho používání doporučeno. *[12]*

2.4.2 Secure hash algorithm

Algoritmus SHA byl navrhnut NSA („National Security Agency“ česky „Národní bezpečnostní agentura“ v USA) a vydala ho jako bezpečnostní hašovací standard NIST („National Institute of Standards and Technology“ česky „Národní institut pro standardy a technologie“ v USA) jako náhradu za MD5. Pod hašovací funkcí SHA jsou zahrnuty funkce SHA-0, SHA-1, SHA-224, SHA-256, SHA-386, SHA-512. Poslední čtyři funkce jsou označovány jako SHA-2. Od roku 2015 do rodiny SHA patří i hašovací funkce SHA-3. *[14, 15]*

SHA-0 a SHA-1 vytvářejí 160-bitové haše (o velikosti 160 bitů), které se na výstupu zobrazují jako 40ti místný hexadecimální řetězec. Funkce z rodiny SHA-2 pak vytvářejí haše o délce odpovídající číslu v jejich označení (224 b, 256 b, 386 b, 512 b) a zobrazují se jako hexadecimální řetězce o velikosti (56, 64, 96 a 128) míst. K zvětšení haše u SHA-2 došlo z důvodu poskytnutí větší bezpečnosti těchto hašovacích funkcí. Zvětšením haše totiž došlo ke zvýšení počtu možných výstupů z hašovací funkce a tím ke snížení šance na nalezení kolize 1. nebo 2. stupně. *[15]*

V roce 2015 NIST vydala jako bezpečnostní hašovací standart skupinu hašovacích funkcí pod označením SHA 3. Na rozdíl od předchozích hašovacích funkcí SHA byla SHA 3 vytvořena v rámci veřejné soutěže. Tuto soutěž vyhrála hašovací funkce přihlášená pod názvem KECCAK. Skupina SHA 3 se skládá z hašovacích funkcí SHA3-224, SHA3-256, SHA3-384 a SHA3-512 a dvou funkcí prodlužujících výstup SHAKE128 a SHAKE256. *[14]*

V současné době jsou jako hašovací funkce z rodiny SHA využívány převážně funkce SHA-2 a SHA-3, jelikož funkce SHA-0 a SHA-1 již byly překonány a není doporučeno jejich využívání. *[14]*

2.5 Využití hašovacích funkcí k tvorbě hesla

S přihlédnutím k vlastnostem, díky kterým je hašovací funkce schopna i z jednoho písmene vytvořit textový řetězec o konstantní velikosti, jeví se hašovací funkce jako ideální nástroj, který může pomoci při tvorbě silných a bezpečných hesel. *[4]*

Jako vstupní hodnotu nepotřebujete žádné složité heslo. Prakticky vzato by to mohlo být i jen jediný znak. Hašovací funkce pak vytvoří textový řetězec, který bude mít za každé situace stejnou délku. *[4]*

Navíc zde odpadá, v porovnání s ostatními druhy použití hašovací funkce, nutnost pravidelně sledovat, zda již nebyla použitá hašovací funkce překonána. Důvodem je podmínka pro vytvoření kolize v hašovací funkci. Tou podmínkou je znalost výsledného haše, ke kterému se hledá kolize. A jelikož není předpoklad, že by se útočník dostal k haši, který bude použit jako základ při tvorbě hesla, je bezpečné použít k tvorbě hesla i funkci, která již byla prolomena a je tak pro jiná použití nevyhovující. *[4]*

II. PRAKTICKÁ ČÁST

3 VYUŽITÍ HAŠOVACÍCH FUNKCÍ PŘI TVORBĚ HESLA

Cílem praktické části této bakalářské práce je návrh prakticky použitelné aplikace, umožňující tvorbu hesel do informačních systémů ve specifickém prostředí s využitím hašovacích funkcí.

3.1 Praktické výhody a nevýhody využití hašovacích funkcí při tvorbě hesla

3.1.1 Výhody

Jednou z největších výhod využití hašovací funkce při tvorbě hesla, vyplývá ze samotných vlastností hašovací funkce. Hašovací funkce vytváří z libovolně dlouhého textového řetězce řetězec konstantní délky a zároveň i sebemenší rozdíl na vstupních hodnotách vytváří velký rozdíl na výstupním řetězci.

Z toho vyplývá, že i když jako vstupní hodnota bude pouze jedno písmeno, hašovací funkce stejně vytvoří řetězec o předem určeném počtu znaků. Například hašovací funkce MD5 vytvoří textový řetězec o 32 znacích. Hašovací funkce SHA 512 pak i z jednoho písmene vytvoří textový řetězec o 128 znacích.

3.1.2 Nevýhody

Největší nevýhodou při vytváření hesla hašovací funkcí, je paradoxně to, co je největší výhodou tohoto využití. Při vytvoření hesla pomocí hašovací funkce vznikne dlouhé a bezpečné heslo, jehož zapamatování si je ale daleko za hranicí schopností paměti běžného člověka. Obzvlášť vezmeme-li v potaz bezpečnostní zásadu, podle které by neměl uživatel používat jedno heslo k dvěma různým účtům. Pokud má běžný uživatel několik emailových účtů, internetový bankovní účet a několik účtů na sociálních sítích, stává se zapamatování si tolik různých a dlouhých hesel nadlidským výkonem. Z tohoto důvodu by mohlo využívání hašovacích funkcí být dost nepraktickým způsobem tvorby reálných hesel.

V okamžiku kdy útočník zná algoritmus tvorby hesla, není pro něj problém takové heslo prolomit.

Jako další nevýhoda při tvorbě hesel hašovacími funkcemi by mohla být skutečnost, že někteří poskytovatelé služeb stále ještě používají při ukládání hesel do databáze hašovací

funkci MD5 apod. V okamžiku kdy heslo vytvořené např. hašovací funkcí SHA-512 zahašujeme funkcí MD5, dojde k tzv. přetečení a díky tomu ke zvyšení šance na vznik kolize. Při velké smůle pak hašovací funkcí SHA-512 vytvořené heslo bude mít stejný MD5 haš jako například slovo „heslo“. Což může ohrozit bezpečnost uživatelského účtu.

3.1.3 Jiný úhel pohledu

Obtížnost nebo spíše nemožnost pro běžného člověka zapamatovat si i jen jediné heslo o délce například 128 znaků je bohužel fakt, se kterým se toho moc dělat nedá. A aby si běžný uživatel z hlavy, bez použití nějakého programu, správně vypočítal haš z textového řetězce, je v současné době za hranicí zázraku.

Při hlubším zamyšlení pak vyvstává otázka, když na vytvoření hesla potřebuje uživatel určitý program, musí si uživatel pamatovat hašovací funkcí vytvořený řetězec, anebo mu stačí pamatovat si původní vstupní text a mít u sebe ten určitý program, který mu opětovně vytvoří to samé heslo?

Díky využívání programu, uživateli stačí pouze mít program a pamatovat si vstupní text pro hašovací funkci. Tento vstupní text nemusí být pro uživatelovu paměť nijak náročný a může umožnit uživateli i pravidelnou změnu hesla.

Pokud by program výstupní haš ještě upravil, tak že by pro útočníka bylo obtížné i při znalosti postupu tvorby, tento postup bez znalosti uživatelem zadávaných vstupních hodnot replikovat. A zároveň by zkrátil výstupní řetězec na délku, která by umožňovala nepřetečení jednoho z nejběžněji používaného hašovacího algoritmu MD5, čímž by se vyhnul výše uvedeným nevýhodám, mohly by být hašovací funkce užitečným pomocníkem při tvorbě bezpečných hesel.

4 APLIKACE „ORACULUM“

Jako specifické prostředí pro použití aplikace, byl zvolen operační systém Windows 10 od firmy Microsoft. Při tvorbě samotné aplikace bylo použito vývojové prostředí Microsoft Visual Studio Community 2015. Zdrojový kód samotné aplikace je pak napsán v programovacím jazyce C#.

Aplikace byla pojmenována „Oraculum“. Tento název byl inspirován přednáškou „*Hašovací funkce: MD5,SHA-x, HMAC*„ doc. Ing. Roberta LÓRENCZ, CSC., který v ní uvádí:

Hašovací funkce jako náhodné orákulum

Z hlediska bezpečnosti požadujeme, aby se hašovací funkce chovala jako náhodné orákulum. Odtud se odvozují bezpečnostní vlastnosti.

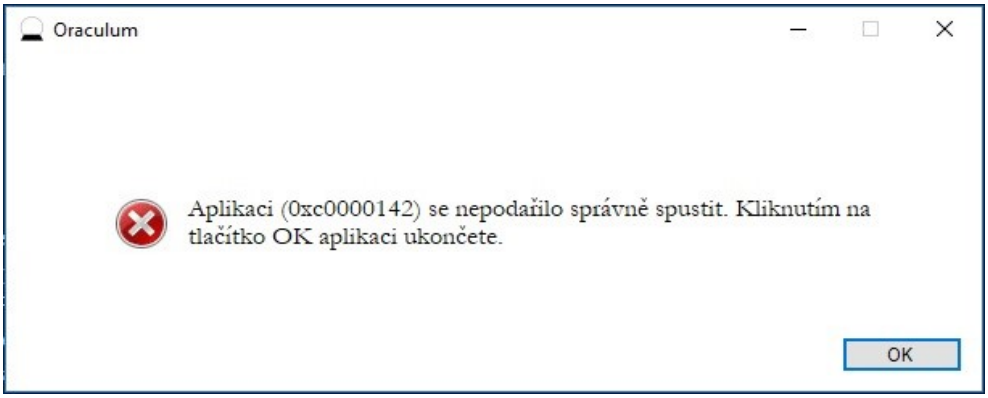
- *Orákulum nazýváme libovolný stroj ("podivuhodných vlastností"), který na základě vstupu odpovídá nějakým výstupem. Má pouze vlastnost, že na tentýž vstup odpovídá stejným výstupem.*
- *Náhodné orákulum je orákulum, které na nový vstup odpovídá náhodným výběrem výstupu z množiny výstupů. [3] [11]*

Z důvodu snadného použití i na počítačích, na kterých nejste jako stálý uživatel, je aplikace sestavena v tzv. „portablové“ verzi, která pro své spuštění nepotřebuje instalaci. Jednoduše může uživatel mít aplikaci uloženou například na flash disku, který stačí připojit k počítači a následně z něj spustit aplikaci Oraculum.

4.1 Uživatelské rozhraní aplikace Oraculum

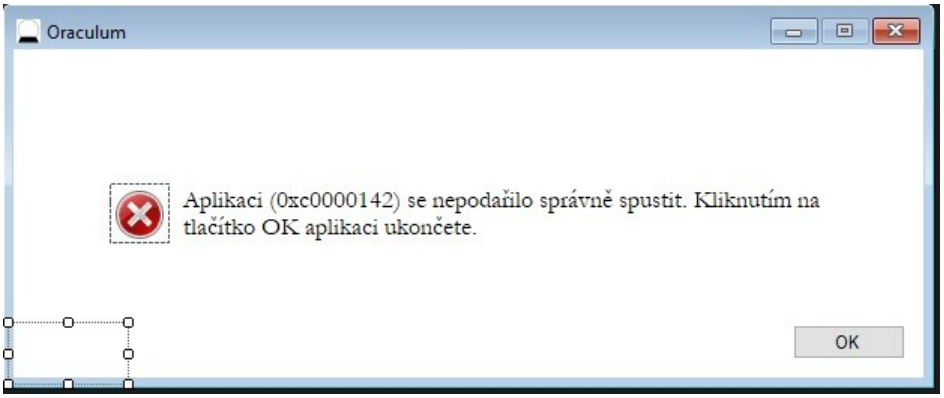
4.1.1 Okno aplikace „Oraculum“

Při spuštění aplikace se otevře okno, které obsahuje jedno viditelné tlačítko s nápisem OK, umístěné v pravém dolním rohu, jedno neviditelné tlačítko, umístěné v levém dolním rohu a na pozadí okna aplikace je zobrazeno běžné chybové hlášení Windows: „Aplikaci (0xc0000142) se nepodařilo správně spustit. Kliknutím na tlačítko OK aplikaci ukončete.“ oznamující uživateli, že aplikace Oraculum se nespustila správně a tudíž nebude fungovat. viz. Obrázek 1

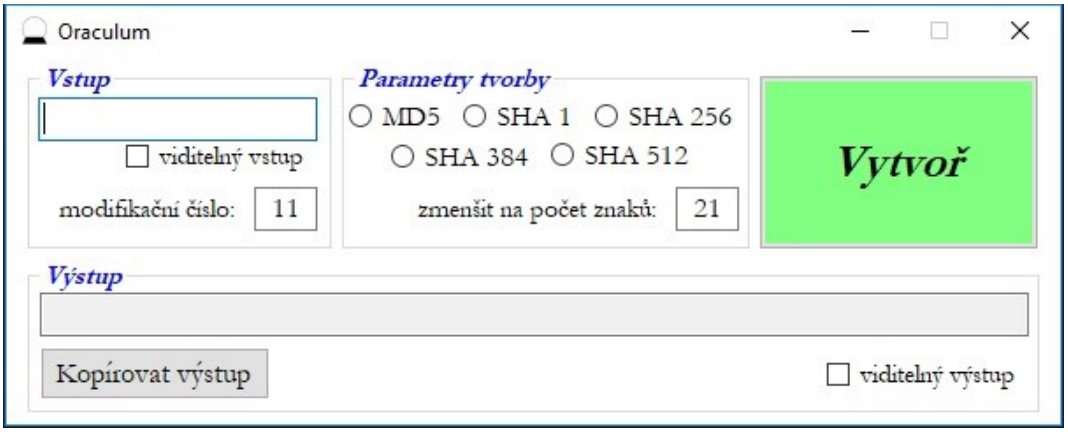


Obrázek 1: Chybové hlášení Windows. *Zdroj: vlastní*

Toto hlášení má zabránit nezasvěcenému uživateli v odhalení funkce tohoto programu. Prakticky se jedná o snahu ochránit tuto aplikaci před nezasvěceným uživatelem za pomoci Staganografie, což je metoda při které není nutné využít šifrování, místo toho informaci schováváme. Zasvěcený uživatel bude vědět, že v levém dolním rohu aplikace je skryté tlačítko, které zpřístupní uživateli textová pole a další komponenty programu potřebné k vytvoření hesla. viz. Obrázek 2 a 3.



Obrázek 2: Zvýrazněné tlačítko pro zpřístupnění komponent. *Zdroj: vlastní*



Obrázek 3: Aplikace se zpřístupněnými komponentami. *Zdroj: vlastní*

Okno aplikace pak obsahuje tři viditelné panely. Jeden panel pro vstup, druhý pro nastavení parametrů tvorby a třetí pro výstup. Při zachování komponent nacházejících se uvnitř panelů, by tyto panely v aplikaci vůbec nemuseli být. Byly použity pouze pro grafické oddělení vstupních hodnot od nastavovacích parametrů a výstupních hodnot. Tím došlo k zpřehlednění uživatelského rozhraní aplikace.

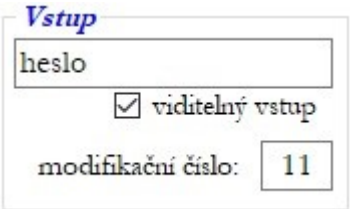
4.1.1.1 Panel Vstup

Tento panel obsahuje tři komponenty. První z nich je textové pole, do kterého uživatel zapíše jím zvolený textový řetězec, který následně bude sloužit jako vstupní data pro tvorbu hesla. V rámci zvýšení bezpečnosti vstupního textového řetězce, je textové pole nastaveno tak, aby zobrazovalo místo uživatelem zadaného textu jen tečky. viz Obrázek 4.

Obrázek 4: Skrytý vstupní text. *Zdroj: vlastní*

Druhou komponentou je textové pole pro zadávání celého čísla, které je následně použité při tvorbě hesla. Toto číslo umožňuje uživateli ovlivnit tvorbu hesla.

Třetí komponentou je zaškrtávací políčko, které má sloužit pro případné zobrazení a opětovnému skrytí uživatelem zadaného textového řetězce ve „vstupním“ textovém poli viz. Obrázek 5.



Obrázek 5: Viditelný vstupní text. *Zdroj: vlastní*

4.1.1.2 Panel Parametry tvorby

Tento panel obsahuje pět „radio buttonů“ a jedno textové pole. Sada „radio buttonů“ dovoluje uživateli vybrat pouze jednu z možných pěti variant. U každé z variant je napsána hašovací funkce, která bude použita při tvorbě hesla. To umožňuje uživateli volbu, pomocí jaké hašovací funkce své heslo vytvoří. Zároveň to umožní za použití jednoho vstupního řetězce, i bez použití modifikačního čísla, vytvořit 5 různých hesel.

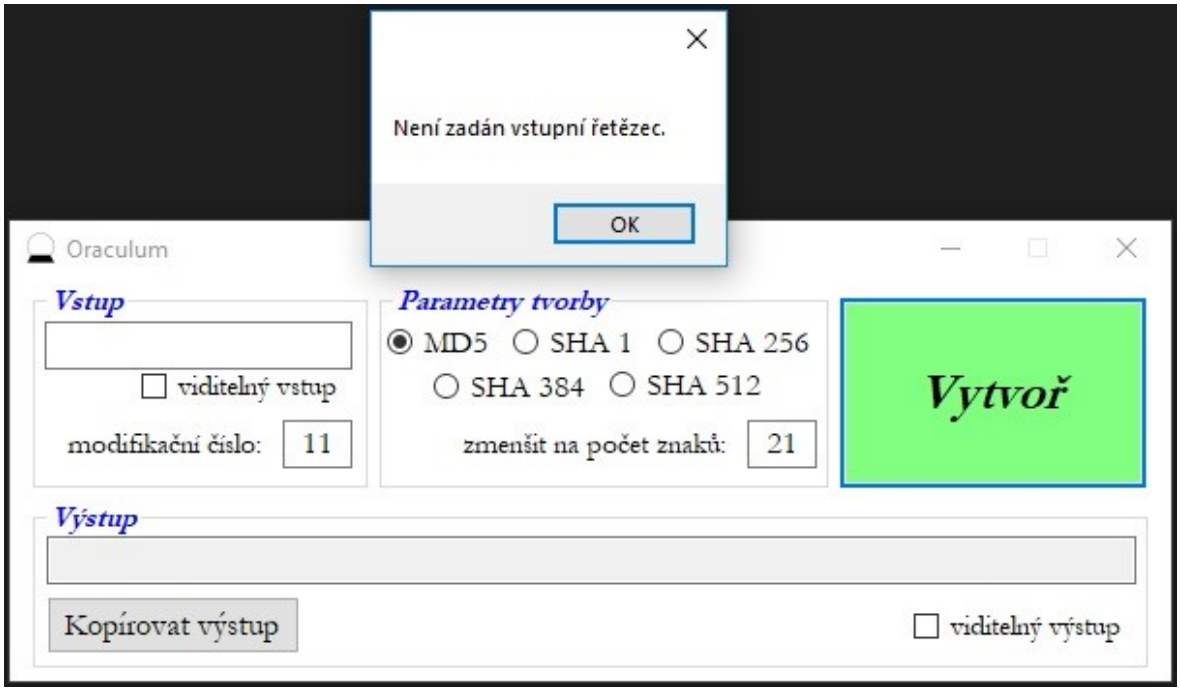
Textové pole v tomto panelu dává možnost uživateli nastavit velikost vytvořeného hesla na jinou nežli přednastavenou velikost. Zkrácení vzniklého hesla oslabuje bezpečnost hesla.

4.1.1.3 Tlačítko Vytvoř

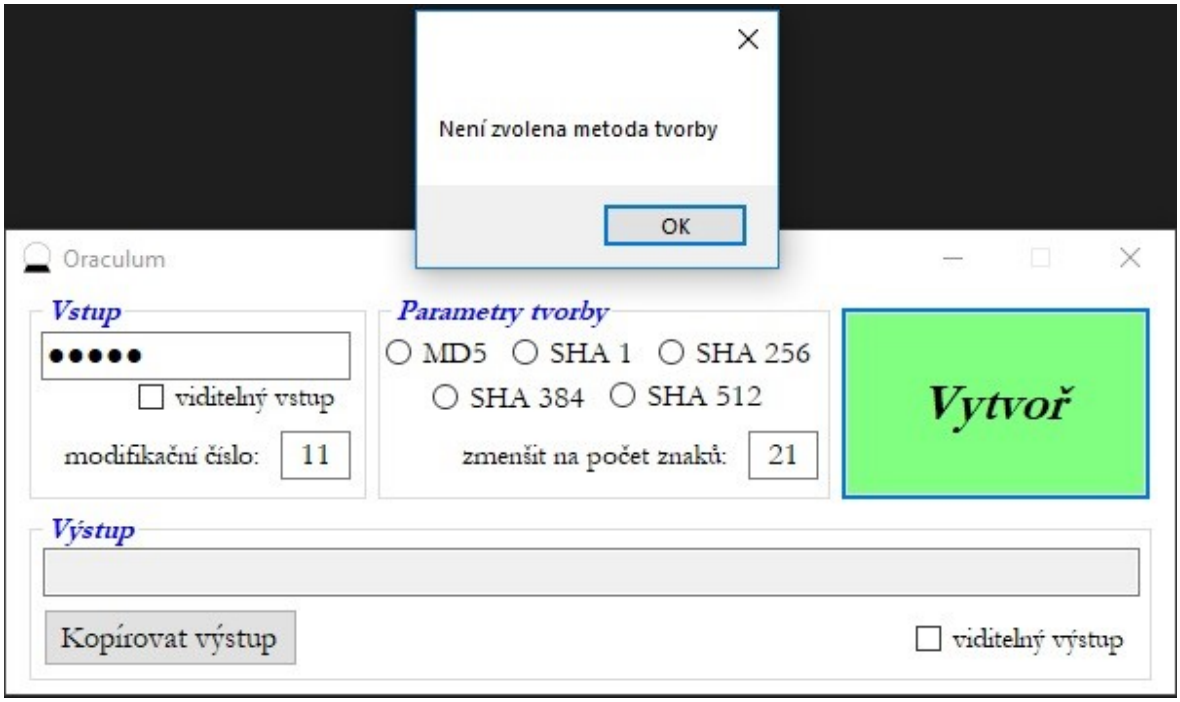
Při kliknutí na tlačítko „Vytvoř“, se nejdříve zkontroluje vyplnění všech údajů nezbytných pro vytvoření hesla. Těmito parametry jsou:

1. textový řetězec ve vstupním textovém poli
2. označena jedna z 5-ti variant v panelu „Parametry tvorby“

Pokud nejsou oba požadavky splněny, program zobrazí okno s upozorněním na zjištěný nedostatek. Viz Obrázek 6 a 7.



Obrázek 6: Chybové hlášení při chybějícím vstupním řetězci. *Zdroj: vlastní*



Obrázek 7: Chybové hlášení při chybějícím nastavení parametru tvorby. *Zdroj: vlastní*

V případě, že jsou splněny oba požadavky nezbytné pro vytvoření hesla, dojde následně k vytvoření patřičného haše z textového vstupu.

Zde by mohl nastat drobný problém, jelikož standardní zobrazení haše vytvořeného hašovací funkcí, je ve formátu, který obsahuje znaky pouze dvou znakových sad (0-9 a a-z). Někteří poskytovatelé mají totiž nastaveno bezpečnostní minimum pro jimi uznatelné heslo, bez jehož splnění uživateli nedovolí heslo použít. Jak už bylo uvedeno v teoretické části, toto minimum je ve většině případů nastaveno na minimální počet 8 znaků a použití min. tří znakových sad.

Délka hesla pro nás nebude problém, ale kvůli splnění podmínky na počet použitých znakových sad, upraví aplikace Oraculum tento haš tak, aby výsledný výstup obsahoval i prvky ze znakových sad (A-Z a speciální znaky). Tím dojde k splnění podmínek nastavených poskytovateli a zároveň k zvýšení síly hesla. Znakovou sadu „a-z“ s diakritikou a „A-Z“ s diakritikou aplikace Oraculum nepoužívá, z důvodu možné nekompatibility se znakovými sadami podporovanými systémem poskytovatele služby.

Na panelu „Parametry tvorby“ je umístěno textové pole, kam může uživatel zadat číselnou hodnotu, určující požadovaný počet míst vytvářeného hesla. V případě že je zadaná

správná číselná hodnota, dojde k úpravě hesla na požadovanou velikost. V případě že není zadána správná číselná hodnota, je velikost hesla nastavena na 21 míst.

Po skončení všech těchto činností je následně vytvořené heslo zapsáno do textového pole v panelu Výstup. Viz. Obrázek 8.

Obrázek 8: „Zobrazeno“ heslo ve výstupním textovém poli. *Zdroj: vlastní*

4.1.1.4 Panel Výstup

Na panelu Výstup je umístěno jedno textové pole, jedno zaškrtávací tlačítko s nápisem „viditelný výstup“ a jedno tlačítko s nápisem „Kopírovat výstup“.

Do textového pole zapisuje program vygenerované heslo. Editace textového řetězce v tomto textovém poli je znemožněna z důvodu ochrany před nechtěným pozměněním textového řetězce uživatelem. Pole je nastaveno tak, aby textová hodnota zapsaná do tohoto pole nebyla čitelná. Pole zobrazuje textový řetězec, jako řadu teček viz Obrázek 8. Toto je z důvodu ochrany hesla před neoprávněným pozorovatelem. Je pravda, že zapamatovat si textový řetězec o délce 21 míst složený ze čtyř znakových sad, by pro normálního člověka mohlo být trochu problematické, jsou však jedinci, kterým by to nedělalo žádný problém. Navíc nikdy nemůžete vyloučit použití některého ze záznamových zařízení (např. fotoaparát či kamera).

Zaškrtávací tlačítko slouží pro zobrazení čitelných znaků a jejich opětovnému „schování“ viz Obrázek 9. Tato komponenta je zde pro případ, že bychom potřebovali vidět textovou podobu vytvořeného hesla.

Obrázek 9: Viditelný výstupní řetězec. *Zdroj: vlastní*

Nastavením výstupního textového pole tak, aby nezobrazovalo čitelnou textovou hodnotu, dochází zároveň i k zablokování možnosti kopírovat obsah textového pole za pomocí

klávesové zkratky „Ctrl+c“ nebo vybráním možnosti „kopírovat“ z nabídky zobrazené po kliknutí pravým tlačítkem myši. Pro odblokování těchto možností by uživatel nejdříve musel zviditelnit textový výstup. To však ohrožuje bezpečnost vytvořeného hesla. Proto bylo na panel „Výstup“ přidáno tlačítko s nápisem „Kopírovat výstup“. Toto tlačítko umožňuje zkopírovat do schránky (clipboardu) Windows hodnotu výstupního textového pole bez ohledu na to, zda je výstupní textové pole nastaveno na zobrazování či nezobrazování čitelné textové hodnoty pole.

4.2 Zdrojový kód aplikace „Oraculum“

4.2.1 Struktura aplikace „Oraculum“

Jak bylo výše uvedeno, program Oraculum byl vytvářen ve vývojovém prostředí Microsoft Visual Studio Community 2015. Zde byl vytvořen nový projekt s nastavenou vlastností projektu „Windows Forms Applications“. Při tvorbě nového projektu pak vývojové prostředí automaticky vytvoří funkční program, který po spuštění zobrazí prázdné okno. Části tohoto programu, které jsou rovněž automaticky vygenerovány, jsou *Properties*, *References*, *Form1.cs* a *Program.cs*

Properties umožňují uživateli aplikace číst, zapisovat a měnit, některé vlastnosti aplikace. References obsahují knihovny, které vytvořená aplikace potřebuje pro svůj bezproblémový chod. Form1.cs je formulář aplikace. Prakticky se jedná o okno, které se zobrazí při spuštění programu. Program.cs je třída, která nastavuje základní vlastnosti programu. Protože při tvorbě aplikace „Oraculum“ byl z výše jmenovaných částí editován pouze obsah formuláře Form1.cs, nebudeme se ostatním výše jmenovaným částem dále věnovat.

Jako první věc po vygenerování aplikace vývojovým prostředím, byl *Form1.cs* přejmenován na *FormMain.cs*. Nenechávat formuláře pojmenované *Form1*, *Form2*, *Form3* atd. je dobrým zvykem z důvodu přehlednosti programu. V případě, že by se jednalo o větší aplikaci, bude orientace ve formulářích *Form1 – n* dost nepřehledná. Na druhou stranu správné pojmenování formulářů (např. *FormUzivatel*, *FormVozidlo*, *FormAutor*, *FormKniha*) může orientaci ve zdrojových kódech programu dost usnadnit.

Dále pak byly vytvořeny dvě nové třídy a pojmenovány *Hash.cs* a *Uprava.cs*. Z důvodu jednoduššího pochopení běhu programu, bude nejdříve vysvětlen zdrojový kód těchto dvou tříd a následně pak zdrojový kód formuláře *FormMain*.

4.2.2 Hash.cs

Třída *Hash.cs* nám bude sloužit jako místo, kde bude docházet k samotnému hašování uživatelem zadaného textového řetězce. Z tohoto důvodu jsme do třídy *Hash.cs* nejdříve do části pro překladač doplnili:

```
1)  „using System.Security.Cryptography;“
```

Překladač pak při sestavování programu, použije i námi uvedenou část knihovny. Ta nám následně umožní používání hašovacích algoritmů v této části programu.

Jako další byly na začátku těla třídy *Hash* vytvořeny dvě textové proměnné. První z nich s názvem *hash16* je veřejná a je jí přiřazena prázdná hodnota. Druhá textové proměnná s názvem *hash64* je lokální (neveřejná) a je jí také přiřazena prázdná hodnota. Rozdíl mezi veřejnou a lokální proměnnou, je v možnosti přístupu k ní. Zatím co k lokální proměnné jsme schopni přistupovat pouze z těla funkce, třídy či formuláře, ve kterém byla tato proměnná deklarována, k veřejné proměnné můžeme přistupovat za určitých podmínek z jakéhokoliv místa v programu. Samotná deklarace těchto proměnných pak vypadá následovně:

```
1)  public string hash16 = "";
2)  string hash64 = "";
```

4.2.2.1 Veřejná textová funkce „GeneratorMD5“

Jako další byla v třídě *Hash* vytvořena veřejná textová funkce, která byla pojmenována *GeneratorMD5* a jako vstupní parametr vyžaduje textovou hodnotu. Veřejná funkce znamená, že tato funkce je dostupná ze všech částí programu. Textová funkce znamená, že návratová hodnota této funkce bude v textovém formátu. Celá funkce pak vypadá následovně:

```
1)   public string GeneratorMD5(string str)
2)   {
3)       hash16 = "";
4)       byte[] bytovaHodnota = Encoding.ASCII.GetBytes(str);
5)       MD5CryptoServiceProvider md5 = new MD5CryptoServiceProvider();
6)       bytovaHodnota = md5.ComputeHash(bytovaHodnota);
7)       foreach (byte b in bytovaHodnota)
8)       {
9)           hash16 += b.ToString("x2");
10))      }
```

```
11)         hash64 = Convert.ToBase64String(bytovaHodnota);
12)         return (hash64);
13)     }
```

Při deklaraci této funkce zároveň deklarujeme textovou proměnnou s názvem *str*. Jako hodnota této proměnné pak bude nastavena textová hodnota, která bude vyžadována při volání této funkce jako vstupní parametr.

Na třetím řádku výpisu je proměnné *hash16* přiřazena prázdná hodnota. Opětovné přiřazení prázdné hodnoty této proměnné je proto, že se jedná o veřejnou proměnnou, která může být pozměněna při předchozí tvorbě hesla a tato změněná hodnota se v ní stále uchovává. Tím že opětovně přiřadíme proměnné prázdnou hodnotu na začátku funkce, získáme jistotu, že v okamžiku kdy s touto proměnnou začne tato funkce pracovat, nebude mít proměnná hodnotu nastavenou předchozím hašováním.

Na čtvrtém řádku výpisu dochází nejdříve ke změně formátu hodnoty obsažené v proměnné *str* z textové na bytovou. Následně je deklarováno pole typu „byte“ s názvem *bytovaHodnota*, které je naplněno bytovou hodnotou vzniklou převodem hodnoty proměnné *str*.

Na pátém řádku výpisu pak deklarujeme novou instanci MD5CryptoServiceProvider, která následně zajistí hašování algoritmem MD5. Tato instance byla pojmenována *md5*. *[16]*

Na šestém řádku výpisu, je použita výše uvedená instance „MD5CryptoServiceProvider“, která zahašuje veškeré hodnoty v bytovém poli *bytovaHodnota* a následně hodnoty v tomto poli přepíše nově vzniklými hodnotami. *[16]*

Na sedmém řádku výpisu funkce je pak umístěn cyklus „foreach“. V parametru cyklu pak deklarujeme proměnou *b* typu „byte“ v bytovém poli *bytovaHodnota*. Cyklus „foreach“ pak bude mít tolik opakování, kolik je v poli *bytovaHodnota* neprázdných míst. Při nálezu neprázdného místa v poli (v našem případě místo s bytovou hodnotou), jsou provedeny příkazy uvedené v těle cyklu „foreach“. V okamžiku, kdy dojde k projití všech těchto míst, dojde k ukončení cyklu a program následně pokračuje na prvním řádku za koncem těla cyklu „foreach“.

Devátý řádek výpisu obsahuje vlastní tělo cyklu „foreach“. Zde dochází k změně formátu z bytové hodnoty proměnné *b* na textovou hodnotu, která je díky parametru „x2“ for-

mátována do hexadecimální podoby. Takto zformátovaná hodnota je pak přiřazena k proměnné *hash16*. Díky „+=“ ale nedochází k přepisování hodnoty proměnné hodnotou novou, ale k dopisování nové hodnoty za poslední znak hodnoty původní. *[17]*

Na jedenáctém řádku jsou pak hodnoty bytového pole *bytovaHodota* převedeny do textového formátu „Base64“ a takto vzniklý textový řetězec je pak přiřazen jako hodnota textové proměnné *hash64*.

Na dvanáctém řádku těla funkce je splněna podmínka návratové hodnoty funkce v textovém formátu. Tento řádek vrací jako výstup z funkce *GeneratorMD5* hodnotu textové proměnné *hash64*.

Důvod, proč jako návratová hodnota je použit výstup z hašovací funkce převedený do formátu „Base64“ je prostý. Výstup převedený do hexadecimální podoby obsahuje znaky pouze dvou znakových sad („0-9 a „a-z“). V případě, že bychom šli úplně do detailů, tak výstup v hexadecimální podobě může nabývat pouze znaků v rozsahu „0-9“ a „a-f“, což je prakticky 16 možných znaků na každé pozici. Oproti tomu formát „Base64“ umožňuje na každé pozici nabývat znaků v rozsahu „0-9“, „a-z“, „A-Z“, „+“ a „/“. Což je 64 možných variant na každé pozici. V závislosti na délce formátovaného řetězce se může na konci zformátovaného výstupu objevit ještě znak „=“. Jelikož však tento znak se může objevit pouze na posledních dvou pozicích výstupu, není tento znak považován v „Base64“ jako další varianta. *[18]*

Jelikož více odpovídá bezpečnostním parametrům, požadovaným poskytovateli služeb, jak již bylo v teoretické části popsáno, zdá se výhodnější, použít formát „Base64“ nežli hexadecimální formát.

4.2.2.2 Další hašovací funkce v třídě Hash.cs

Po vytvoření veřejné textové funkce *GeneratorMD5* byly stejným způsobem vytvořeny i další veřejné textové funkce *GeneratorSHA1*, *GeneratorSHA256*, *GeneratorSHA384* a *GeneratorSHA512*.

Jedním ze dvou rozdílů v těchto funkcích je na pátém řádku výpisu. Zde na rozdíl od výše probrané funkce není deklarována instance „MD5CryptoServiceProvider“, která by byla pojmenována *md5*, ale jsou deklarovány instance „SHA1CryptoServiceProvider“, pojmenovaná *sha1*, „SHA256CryptoServiceProvider“ pojmenovaná *sha256*, „SHA384-

CryptoServiceProvider“ pojmenovaná *sha384* a „SHA512CryptoServiceProvider“ pojmenovaná *sha512*. V každé funkci je deklarována instance v závislosti na hašovacím algoritmu, který má být v dané funkci použitý.

Druhý rozdíl je pak na šestém řádku výpisu, kde při hašování není použita instance s názvem *md5*, ale jsou zde použity instance v závislosti na předchozí deklaraci v té dané funkci.

Jelikož toto jsou jediné rozdíly mezi funkcí *GeneratorMD5*, jejíž zdrojový kód byl výše probrán, nebudou zde zdrojové kódy těchto dalších funkcí, vytvořených ve třídě *Hash.cs* dále probírány.

4.2.3 Uprava.cs

Jak bylo již výše zmíněno, výstup z veřejných funkcí v třídě *Hash.cs* je ve formátu „Base64“, který obsahuje znaky ze znakových sad „0-9“, „a-z“, „A-Z“ a prakticky i některé „speciální znaky“. Z důvodu většího počtu možností na každé pozici jsou řetězce formátované v Base64 kratší nežli řetězce v hexadecimálním formátu. U MD5 má výstupní řetězec ve formátu Base64 pouze 24 pozic oproti 32 pozicím hexadecimálního řetězce. U SHA 1 je to 28 pozic ve formátu Base64, u SHA 256 je to 44 pozic, u SHA384 je to 64 pozic a u SHA 512 je to 88 pozic.

Takto vytvořené heslo by samo o sobě mělo vysokou entropii. Například ze vstupního textového řetězce „heslo“ bude hašovací funkcí MD5 vytvořený textový řetězec ve formátu „Base64“ mít 24 míst a bude vypadat takto:

„*lV2wuB7xmJtKTf6ugGGppg==*“

Takové heslo by mělo více jak 5,44 x 10^{39} variant $(64^{22} + 1^2)$ a entropie tohoto hesla by byla 132 bitu $((22*\log_2 64) + (2 * \log_2 1) = 22*6 + 2* 0)$. Takto vytvořené heslo má vysokou entropii a jeví se jako silné a bezpečné. Ohrožením tohoto hesla může být okamžik, kdy útočník bude znát postup tvorby hesla. Pak pro útočníka nemusí být zjištění původní vstupní hodnoty žádný velký problém. Aby bylo vytvořené heslo co nejbezpečnější, současný výstup z hašovací funkce bude ještě upraven.

Z tohoto důvodu byla ve třídě *Uprava.cs* vytvořena veřejná textová funkce *uprava*, která požaduje jako vstupní parametry dvě textové hodnoty a čtyři číselné hodnoty. V této třídě pak byly vytvořeny i veřejné číselné funkce *Prevod* a *CH*, které obě požadují jako

vstupní parametr textovou proměnnou. Jelikož ve funkci *uprava* využíváme i funkce *Prevod* a *CH*, bude pro lepší pochopení nejdříve probrána funkce *Prevod* a *CH*.

4.2.3.1 Veřejná číselná funkce „Prevod“

Zdrojový kód funkce *Prevod* vypadá následovně:

```
1)  public int Prevod(string hodnota)
2)  {
3)      int cislo = -1;
4)      try
5)      {
6)          cislo = Convert.ToInt32(hodnota);
7)          return cislo;
8)      }
9)      catch
10)     {
11)         return cislo;
12)     }
13) }
```

Tato funkce nám slouží k ověření, zda textová hodnota, kterou ověřujeme je celé číslo v textovém formátu. Proto vstupní hodnota je požadována v textovém formátu. Návratová hodnota je pak ve formátu celého čísla.

Na třetím řádku výpisu funkce je deklarována celočíselná proměnná *číslo*, které je přiřazená hodnota -1.

Na čtvrtém řádku je příkaz „try“, který patří mezi příkazy pro zpracování výjimek. Prakticky se jedná o ochranný příkaz. V případě, že by v bloku zdrojového kódu umístěného v těle příkazu „try“ došlo k chybě, která by normálně způsobila pád programu, díky příkazu try, dojde k výjimce, při které jsou veškeré příkazy vykonané v těle příkazu „try“ zrušeny, tělo příkazu „try“ se neprovede, ale běh programu je přesunut do těla záchytného příkazu „catch“, v jehož těle je zdrojový kód, který se má provést v případě chyby v těle příkazu „try“. Pokud by při běhu programu nedošlo v těle příkazu „try“ k chybě, je tento zdrojový kód vykonán jako by tam příkaz „try“ nebyl a záchytný příkaz „catch“ je pak přeskočen a nevykonává se.

V našem případě je v těle příkazu „try“ (na šestém řádku) umístěn převod hodnoty textové proměnné *hodnota* do celočíselného formátu, a následné přiřazení této číselné hodnoty do výše deklarované proměnné *číslo*. V případě, že převáděná textová hodnota je celé číslo v textovém formátu, převod proběhne v pořádku a na sedmém řádku výpisu je hodnota proměnné *číslo* vrácena jako návratová hodnota funkce *Prevod*. Pokud by převáděná hodnota nebylo celé číslo v textovém formátu, dojde k chybě, která zruší vykonávání zdrojového kódu v těle příkazu „try“ a přesune běh programu do těla záchytného příkazu „catch“.

V těle příkazu „catch“ na jedenáctém řádku výpisu je jako návratová hodnota funkce vrácena hodnota proměnné *číslo*, která byla na třetím řádku nastavena na hodnotu -1.

4.2.3.2 Veřejná číselná funkce „CH“

Zdrojový kód funkce *CH* vypadá následovně:

```
1)  public int CH(string text)
2)  {
3)      string mala = "aábcčdďeéfghiíjklmnňoópqrřsštťuúůvwxyýzž";
4)      string velka ="AÁBCČDĎEÉFGHIÍJKLMNŇOÓPQRŘSŠTŤUÚŮVWXYÝZŽ";
5)      string cisla = "0123456789";
6)      if (mala.Contains(text))
7)          return 1;
8)      else if (velka.Contains(text))
9)          return 2;
10)     else if (cisla.Contains(text))
11)         return 3;
12)     else
13)         return 0;
14) }
```

Jak bylo výše uvedeno, funkce *CH* je veřejná celočíselná funkce, jejíž vstupní parametr je textová proměnná *text*.

Na třetím, čtvrtém a pátém řádku výpisu této funkce jsou deklarovány textové proměnné *mala*, *velka*, *cisla*. Těmto proměnným jsou pak následně přiřazeny hodnoty, které jsou uvedeny mezi uvozovkami. Na dalších řádcích zdrojového kódu této funkce pak bude s těmito proměnnými pracováno jako s textovými poli znaků („0-9“, „a–ž“, „A-Ž“).

Na šestém řádku je podmínka „if“, jejíž parametr, který musí být splněn, je nastaven: „*hodnota vstupní textové proměnné „text“ je obsažena v proměnné „mala“*“. Použitím příkazu „.Contains()“ dojde k rozdělení hodnoty v proměnné „mala“ na jednotlivé znaky. Ty jsou následně jeden po druhém porovnány s hodnotou vstupní proměnné „text“. V případě, že bude parametr této podmínky splněn a hodnota proměnné „text“ bude obsažena v hodnotě proměnné *mala*, dojde k vykonání příkazů v těle podmínky „if“. Tělo této podmínky obsahuje jediný řádek zdrojového kódu. Na sedmém řádku je příkazem „return 1“ jako návratová hodnota odesláno číslo „1“.

V případě, že není splněn parametr podmínky „if“, je tělo této podmínky ignorováno a běh programu pokračuje na osmém řádku výpisu.

Na osmém řádku výpisu funkce je podmínka „else if“. K prověřování, zda je splněn parametr této podmínky „if“, dochází jen, pokud není splněna předchozí podmínka „if“. Pokud by předchozí podmínka byla splněna, jsou tato podmínka a její tělo ignorovány. Parametr této podmínky je „*hodnota vstupní textové proměnné „text“ je obsažena v proměnné „velka“*“. Tak jako v předchozí podmínce je i zde použitím příkazu „.Contains“ porovnávána hodnota proměnné *text*. Zde je na rozdíl od předchozí podmínky porovnávána se znaky v proměnné *velka*. Pokud je parametr podmínky splněn, dojde k vykonání těla této podmínky, které obsahuje jediný řádek zdrojového kódu. Na devátém řádku je příkazem „return 2“ jako návratová hodnota odesláno číslo „2“.

Na desátém řádku těla funkce je další podmínka „else if“. K prověřování splnění parametru této podmínky dojde pouze, když nebyla splněna první podmínka „if“ na řádku 6 a následně nebyl splněn ani parametr podmínky „else if“ na osmém řádku výpisu. Jako parametr této podmínky je „*hodnota vstupní textové proměnné „text“ je obsažena v proměnné „cisla“*“. Podobně jako v předchozích dvou podmínkách dochází i zde, díky příkazu „.Contains“, k porovnávání hodnoty *text* se znaky v určité proměnné. V tomto případě se jedná o proměnnou *cisla*. Při splnění parametru, bude vykonáno tělo této podmínky. Na jedenáctém řádku je pak příkazem „return 3“ jako návratová hodnota odesláno číslo „3“.

Na dvanáctém řádku je podmínka „else“, která je vykonána pouze v případě, že nebyla splněna ani jedna z předchozích podmínek v této funkci. Prakticky dojde k provedení těla této podmínky pouze, pokud hodnota textové proměnné *text* není obsažena ani v jedné tex-

tové proměnné, které byly deklarovány v této funkci (*mala*, *velka*, *cisla*). V těle této podmínky je jediný řádek zdrojového kódu. Na tomto třináctém řádku je příkazem „return 0“ jako návratová hodnota funkce odesláno číslo „0“.

Prakticky pomocí této funkce vyhodnocujeme, zda hodnota proměnné *text* (bude vždy obsahovat pouze jeden jediný znak) patří do skupiny malých písmen, velkých písmen, čísel nebo jiných znaků. Podle toho do jaké skupiny znak v textové proměnné *text* patří, bude navrácena celočíselná hodnota „0“, „1“, „2“, „3“.

4.2.3.3 Veřejná textová funkce „uprava“

Zdrojový kód funkce *uprava* vypadá následovně:

```
1)  public string uprava(string hash, int posledni, int posun,
                         int mnozstvi,string vstup, int delka)
2)  {
3)      string vystup = "";
4)      int index = 7;
5)      int dv = vstup.Length;
6)      int dh = hash.Length;
7)      int pIndex = 0;
8)      if (posun < 0)
9)          posun = (posun * (-1)) % dh : ;
10)     if (posun == 0)
11)         posun += 3;
12)     for (int i = 0; i <= delka; i = i + mnozstvi)
13)     {
14)         switch (CH(vstup[i % dv].ToString()))
15)         {
16)             case 0:
17)                 index = (index + posun + i) % dh;
18)                 break;
19)             case 1:
20)                 index = (index * posun + i) % dh;
21)                 break;
22)             case 2:
23)                 index = ((index – posun) + dh + i) % dh;
24)                 break;
25)             case 3:
26)                 index = ((index * index) + i) % dh;
```

```
27)                     break;
28)           }
29)           if (pIndex == index)
30)                index  += 7;
31)           for (int m = 0; m < mnozstvi; m++)
32)           {
33)                vystup += hash[(index + m) % dh];
34)           }
35)           pIndex = index;
36)      }
37)      // spec. znaky
38)      for(int i = 2; i <= 4; i++)
39)      {
40)           string znak = "-*^";
41)           if (posledni == -1)
42)           {
43)                vystup = vystup.Insert(delka / i,            znak[i -
                                2].ToString());
44)                vystup = vystup.Remove((delka / i) + 1, 1);
45)           }
46)           else
47)           {
48)                vystup = vystup.Insert(delka / i,
                                znak[i - 2].ToString());
49)           }
50)           if (i == 3)
51)           {
52)                vystup = vystup.Insert((delka / i) * 2,       znak[i
                                - 2].ToString());
53)                vystup = vystup.Remove(((delka / i) * 2) + 1, 1);
54)           }
55)      }
56)      vystup = vystup.Remove(delka, vystup.Length-delka);
57)      return vystup;
58) }
```

První řádek zdrojového kódu nám deklaruje veřejnou textovou funkci *uprava*, jejíž vstupními parametry jsou dvě textové proměnné a čtyři celočíselné proměnné. Pro textové proměnné jsou zde deklarovány dvě proměnné *hash* a *vstup*. Pro celočíselné proměnné jsou

zde deklarovány čtyři proměnné *posledni*, *posun*, *mnozstvi* a *delka*. Výstupní hodnota této funkce je v textovém formátu.

Na řádcích 3 až 7 jsou deklarovány textová proměnná *vystup*, které je přiřazena prázdná hodnota a čtyři celočíselné proměnné *index*, *dv*, *dh*, *pIndex*. Proměnné *index* je přiřazena hodnota „1“, proměnné *dv* je přiřazena příkazem „.Length“ hodnota odpovídající počtu znaků v proměnné *vstup*, proměnné *dh* je stejným příkazem přiřazena hodnota odpovídající počtu znaků proměnné *hash* a proměnné *pIndex* je přiřazena hodnota “0“.

Na osmém řádku výpisu funkce je podmínka „if“ s parametrem „*hodnota proměnné „posun“ je menší nežli „0“*“. Při splnění tohoto parametru, je v těle této podmínky hodnota proměnné *posun* vynásobena hodnotou „-1“. Výsledek je pak vydělen, metodou modulo, hodnotou proměnné *dh*. Jelikož nás zajímá zbytek po tomto dělení, nebyla použita klasická metoda dělení, ale metoda modulo. Tato podmínka má za účel ošetřit možnou chybu programu, v okamžiku, kdy by uživatel zadal jako modifikační číslo něco jiného nežli číslo z oboru přirozených čísel s nulou. V případě zadání záporné hodnoty nebo v případě, že návratová hodnota funkce *Prevod* bude „-1“, odstraní vynásobení hodnoty proměnné *posun* hodnotou „-1“ případné záporné znaménko. Pro případ, kdy by uživatel zadal celé číslo, které přesahuje délku textového řetězce proměnné *hash* a docházelo by tak při tvorbě hesla k zbytečnému opakovanému přetékání, byla použita metoda modulo. Tím je zajištěno, že hodnota této proměnné bude v rozsahu velikosti hesla a tak dojde maximálně jednomu přetečení. Výsledná hodnota je pak přiřazena proměnné *posun*.

Na desátém řádku těla funkce je podmínka „if“ jejímž parametrem je „*hodnota proměnné „posun“ je rovna „0“*“. Při splnění tohoto parametru, dojde k vykonání těla této podmínky. Zde je pouze jeden řádek, na kterém je k hodnotě proměnné „posun“ přičtena hodnota „3“.

Na dvanáctém řádku těla této funkce je cyklus „for“, který má jako první vstupní parametr deklarovanou celočíselnou proměnnou *i*, které je zároveň přiřazena hodnota “0“. Druhý vstupní parametr je „*hodnota proměnné „i“ je menší nebo rovna hodnotě proměnné „delka“*“. Ve třetím vstupním parametru, vykonávaném po ukončení každého kola cyklu „for“, je proměnné *i* přiřazena hodnota rovna součtu hodnot proměnných *i* a *množství*. Prakticky na začátku každého kola tohoto cyklu „for“ dojde k ověření, zda je hodnota proměnné *i* nižší nebo rovna hodnotě proměnné *delka* a na konci každého kola tohoto cyklu „for“ k nahrazení současné hodnoty proměnné *i* součtem proměnných *i* a *množství*. Pokud při ověření

je parametr splněn, tělo cyklu „for“ je opět vykonáno, v případě že není, pokračuje program na třicátém osmém řádku výpisu.

V těle tohoto cyklu „for“ se pak na čtrnáctém řádku nachází podmínka „switch“. Tato podmínka jako svůj vstupní parametr přijímá určitou hodnotu. Následně v těle této podmínky jsou nastaveny možné hodnoty vstupního parametru, se kterými je vstupní hodnota porovnávána. Každá z takto nastavených hodnot má pak vlastní zdrojový kód, který bude vykonán v případě, že vstupní hodnota odpovídá té dané nastavené hodnotě. Každý tento kód musí být ukončen příkazem „break“. Toto je také jedním z rozdílů mezi podmínkou „if“, u které je vstupní parametr vyhodnocován před vykonáváním těla této podmínky a podmínkou „switch“, u které je vstupní parametr vyhodnocován uvnitř těla této podmínky.

V našem případě je vstupním parametrem cyklu „switch“ návratová hodnota z výše popsané veřejné funkce *CH*. Jako vstupní parametr při volání této funkce je jeden znak textové proměnné *vstup*, který se nachází na pozici určené výpočtem hodnota proměnné *i* modulo hodnota proměnné *dv*. Tímto výpočtem je zajištěno, že hodnota určující pozici znaku v textové proměnné *vstup* nebude větší, nežli je hodnota indexu posledního znaku této proměnné, a to ani v případě, že hodnota proměnné *i* bude větší nežli délka textového řetězce proměnné *vstup*.

Jak bylo popsáno v předchozí kapitole, může návratová hodnota veřejné funkce *CH* nabývat pouze hodnot „0“, „1“, „2“ a „3“. Z tohoto důvodu jsou v těle podmínky „switch“ příkazy „case“ na řádcích 16, 19, 22 a 25 nastaveny čtyři hodnoty, se kterými je vstupní parametr porovnáván.

V případě, že vstupní parametr podmínky má hodnotu „0“ (v tomto případě, vstupní znak funkce *CH* nepatřil ani do jedné ze skupin znaků v proměnných uvedených ve funkci *CH*), dojde na řádku 17 k sečtení hodnot proměnných *index*, *posun* a *i*. Po součtu je výsledek vydělen, metodou modulo, hodnotou proměnné *dh*. Výsledek je následně přiřazen jako hodnota proměnné *index*.

V případě, že vstupní parametr podmínky má hodnotu „1“ (v tomto případě, vstupní znak funkce *CH* se nacházel mezi znaky proměnné *mala* ve funkci *CH*), dojde na řádku 20 k vynásobení hodnoty proměnné *index* proměnnou *posun*. K výsledku násobení je přičtena hodnota proměnné *i* a celé je to následně vyděleno metodou modulo hodnotou proměnné *dh*. Výsledek je pak přiřazen jako hodnota proměnné *index*.

V případě, že vstupní parametr podmínky má hodnotu „2“ (v tomto případě, vstupní znak funkce *CH* se nacházel mezi znaky proměnné *velka* ve funkci *CH*), dojde na řádku 23 k odečtení hodnoty proměnné *posun* od hodnoty proměnné *index*. K výsledku je přičtena hodnota proměnné *dh* a proměnné *i*. Výsledek tohoto součtu je pak vydělen, metodou modulo, hodnotou proměnné *dh*. Výsledek je pak následně přiřazen jako hodnota proměnné *index*.

V případě, že vstupní parametr podmínky má hodnotu „3“ (vstupní znak funkce *CH* se nacházel mezi znaky proměnné *cisla* ve funkci *CH*), dojde na řádku 26 k vynásobení hodnoty proměnné *index* tou samou hodnotou. K výsledku je přičtena hodnota proměnné *i*. Výsledek tohoto součtu je vydělen, metodou modulo, hodnotou proměnné *dh*. Výsledek je pak přiřazen jako hodnota proměnné *index*.

Po ukončení podmínky „switch“ na dvacátém osmém řádku výpisu, je řádku 29 podmínka „if“, jejíž parametr je „*hodnota proměnné „pIndex“ je roven hodnotě proměnné „index“*“. Při splnění parametru této podmínky dojde k vykonání těla této podmínky. Tělo podmínky je pouze na řádku 30, na kterém je k hodnotě proměnné *index* přičtena hodnota „7“.

Na třicátém prvním řádku výpisu je cyklus „for“. V jeho prvním vstupním parametru je deklarována celočíselná proměnná *m* a je jí přiřazena hodnota „0“, druhým parametrem je „*hodnota proměnné „m“ je menší než hodnota proměnné „množství“*“, třetím parametrem je zvětšení hodnoty proměnné *m* o hodnotu „1“.

V těle tohoto cyklu na řádku 33 dochází k připsání znaku na určité pozici v proměnné *hash* na konec hodnoty textové proměnné *vystup*. Pozice připisovaného znaku je určena výpočtem „součet hodnot proměnných *index* a *m*, který je vydělen, metodou modulo, hodnotou proměnné *dh*.

Řádek 35 je zároveň posledním řádkem nadřazeného cyklu „for“. Na tomto řádku je proměnné *pIndex* přiřazena hodnota proměnné *index*. Tento cyklus nám prakticky vytvořil z haše ve formátu „Base64“ heslo o minimální velikosti, která byla buď nastavena uživatelem, nebo přednastavena v tomto programu na 21 míst.

Aby bylo zajištěno, že takto vytvořené heslo bude obsahovat speciální znak, bude provedena ještě jedna úprava.

Na řádku 38 je cyklus „for“. V jeho prvním parametru je deklarována celočíselná proměnná *i*, které je přiřazena hodnota „2“. Druhý argument je „*hodnota proměnné „i“ je menší nebo rovna „4“*“. Třetí parametr je nastaven na zvětšení hodnoty proměnné *i* o „1“.

Na řádku 40 je deklarována textová proměnná *znak*, které je zároveň přiřazena hodnota „-*^”.

Na řádku 41 je podmínka „if“, jejíž parametr je „*hodnota proměnné „poslední“ je rovna „-1“*“. V případě splnění parametru, dojde k vykonání těla této podmínky. Zde jsou dva řádky zdrojového kódu. Na řádku 43 je do proměnné *vystup* na pozici určenou výpočtem (*hodnota proměnné „delka“ děleno hodnota proměnné „i“*) vložen příkazem „.Insert()“ znak z proměnné *znak*. Pozice znaku, který bude vložen, je určena výpočtem (*hodnota po odečtení hodnoty „2“ od hodnoty proměnné „i“*). Takto upravenou hodnotou proměnné *vystup* je nahrazena její původní hodnota.

Na řádku 44 této podmínky je příkazem „.Remove()“ z hodnoty proměnné *vystup* odstraněn jeden znak. Jeho pozice je určena výpočtem (*hodnota proměnné „delka“ je vydělena hodnotou proměnné „i“ a k výsledku je přičtena hodnota „1“*). Poté je původní hodnota proměnné nahrazena touto upravenou hodnotou.

V případě, že není splněn parametr podmínky „if“, dojde k vykonání příkazu „else“, který je umístěn na řádku 46 za koncem těla podmínky „if“. V těle příkazu „else“ na řádku 48 je do hodnoty textové proměnné *vystup* vložen příkazem „.Insert()“ znak z proměnné *znak*. Pozice kam je znak vkládán, je určena výpočtem (*„hodnota proměnné „delka“ děleno hodnota proměnné „i“*). Pozice znaku, který je vkládán, je určena výpočtem (*„odečtení hodnoty „2“ od hodnoty proměnné „i“*).

Na řádku 50 po konci těla příkazu „else“ je další podmínka „if“, jejímž parametrem je „*hodnota proměnné „i“ je rovna „3“*“. Při splnění parametru dojde k vykonání těla této podmínky. Na řádku 52 je do proměnné *vystup* na pozici určenou výpočtem (*hodnota proměnné „delka“ děleno hodnota proměnné „i“. Výsledek tohoto dělení je vynásoben hodnotou „2“*) vložen příkazem „.Insert()“ znak z proměnné *znak*. Pozice znaku, který bude vložen, je určena výpočtem (*hodnota po odečtení hodnoty „2“ od hodnoty proměnné „i“*). Takto upravenou hodnotou proměnné *vystup* je nahrazena její původní hodnota. Na řádku 53 je příkazem „.Remove()“ z hodnoty proměnné *vystup* odstraněn jeden znak. Jeho pozice

je určena výpočtem (*hodnota proměnné „delka“ je vydělena hodnotou proměnné „i“, výsledek tohoto dělení je vynásoben hodnotou „2“ a k výsledku je přičtena hodnota „1“*). Následně je původní hodnota proměnné *vystup* nahrazena touto upravenou hodnotou.

Na předposledním padesátém šestém řádku těla funkce *uprava*, je příkazem „.Remove()“ odstraněn určitý počet znaků z proměnné *vystup*. Počáteční pozice, na které začne odstraňování, je rovna hodnotě proměnné *delka*. Počet znaků k odstranění je pak určen výpočtem (*odečtení hodnoty proměnné „delka“ od délky textového řetězce proměnné „vystup“získané příkazem „.Length“*).

Na posledním padesátém sedmém řádku těla funkce *uprava* je jako návratová hodnota této funkce příkazem „return“ vracena hodnota proměnné *vystup*.

4.2.4 FormMain.cs

Jako základní formulář této aplikace byl vytvořen Form1.cs, který byl následně přejmenován na *FormMain.cs*. Prakticky se jedná o úvodní okno, které se zobrazí po spuštění aplikace. Zdrojový kód tohoto formuláře byl rozšířen o obslužné metody prvků, které byly popsány v kapitole 5.1 Okno aplikace „Oraculum“. Tyto obslužné metody jsou řešené díky soukromým funkcím *FormMain_Load*, *bOK_Click*, *bZobrazit_Click*, *bVytvor_Click*, *cbViditelnyVstup_CheckedChanged*, *cbViditelnyVystup_CheckedChanged*, „*bKopirovat_Click*“.

4.2.4.1 Soukromá funkce „FormMain_Load“

FormMain_Load je funkce, která je volána při načítání okna *FormMain.cs*. Jedná se o soukromou funkci, bez návratové hodnoty. Tato funkce je soukromá a proto ji lze volat pouze z formuláře *FormMain*. Její zdrojový kód vypadá následovně:

```
1)  private void FormMain_Load(object sender, EventArgs e)
2)  {
3)      bOK.Visible = true;
4)  }
```

Na třetím výpisu této funkce dochází k nastavení viditelnosti tlačítka s názvem *bOK* (v uživatelském rozhraní je to tlačítko vpravo dole s textem „OK“) na hodnotu pravda. To znamená, aby bylo viditelné.

4.2.4.2 Soukromá funkce „bOK_Click“

Tato funkce zajišťuje obslužnou metodu vlastnosti „Click“ tlačítka s názvem *bOK* (v okně aplikace se zobrazuje po spuštění v pravém dolním rohu). Její zdrojový kód vypadá následnovně:

```
1)  private void bOK_Click(object sender, EventArgs e)
2)  {
3)      this.Close();
4)  }
```

Opět se jedná o soukromou funkci. Její tělo obsahuje jediný řádek zdrojového kódu. Výraz „this“ odkazuje na formulář, ve kterém se tato funkce nachází a příkaz „.Close()“ pak slouží k ukončení. Prakticky kliknutím na tlačítko s nápisem „OK“ ukončí aplikaci „Oraculum“.

4.2.4.3 Soukromá funkce „bZobrazit_Click“

Tato soukromá funkce zajišťuje obslužnou metodu pro vlastnost „Click“ tlačítka *bZobrazit* (v okně aplikace je toto tlačítko neviditelné a umístěné v levém dolním rohu). Zdrojový kód této funkce vypadá následovně:

```
1)  private void bZobrazit_Click(object sender, EventArgs e)
2)  {
3)      gbUprava.Visible = true;
4)      gbVstup.Visible = true;
5)      gbVystup.Visible = true;
6)      bVytvor.Visible = true;

7)      label2.Visible = false;
8)      label3.Visible = false;
9)      pChyba.Visible = false;
10)     bOK.Visible = false;

11)     bOK.Enabled = false;
12)     bZobrazit.Enabled = false;
13) }
```

Prakticky by se tělo této funkce dalo rozdělit do tří částí. První část, na řádcích 3 až 6, zajišťuje zobrazení komponent potřebných pro tvorbu hesla. Druhá část, na řádcích 7 až 10, zajišťuje zneviditelnění komponent, které se zobrazili při spuštění aplikace a třetí část, na řádcích 11 a 12, nastaví tlačítka *bOK* a *bZobrazit* jako nedostupná. Nastavované vlastnosti uvedených komponent jsou „.Visible“(viditelnost, která je nastavovaná buď na hodnotu „true“ pro viditelnost komponenty a „false“ pro neviditelnost komponenty) a „.Enebled“(dostupnost, která je nastavovaná buď na hodnotu „true“ pro dostupnost komponenty nebo „false“ pro nedostupnost komponenty).

Komponenty, u kterých je viditelnost nastavována na hodnotu „true“ jsou panely *gbUprava* (panel s textem „Parametry tvorby“), *gbVstup* (panel s textem „Vstup“), *gbVystup* (panel s textem „Výstup“) a tlačítko *bVytvor* (tlačítko s textem „Vytvoř“ v pravém horním rohu aplikace). Nastavením viditelnosti u panelu, dojde prakticky i k zobrazení veškerých komponent, které jsou na daném panelu umístěny.

Komponenty, u kterých je viditelnost nastavována na hodnotu „false“ jsou popisky *label2* a *label3* (které při spuštění aplikace zobrazují chybové hlášení), panel *pChyba* (panel, který je zobrazován při spuštění aplikace a jako pozadí má nastaven obrázek červeného kruhu s bílým „x“ uprostřed) a tlačítko *bOK* (tlačítko s nápisem „OK“ v pravém dolním rohu aplikace).

Komponenty, u kterých je dostupnost nastavována na hodnotu „false“ jsou dvě tlačítka *bOK* (tlačítko v pravém dolním rohu aplikace s textem „OK“) a *bZobrazit* (neviditelné tlačítko v levém dolním rohu aplikace).

4.2.4.4 Soukromá funkce „bVytvor_Click“

Tato funkce zajišťuje obslužnou metodu pro vlastnost „Click“ tlačítka *bVytvor*, které má zobrazovaný text „Vytvoř“ a je umístěno v pravém horním rohu aplikace. Zdrojový kód této funkce vypadá následovně:

```
1)  private void bVytvor_Click(object sender, EventArgs e)
2)  {
3)      string vysledek = "";
4)      if (tbVstup.Text == "")
5)      {
6)          MessageBox.Show("Není zadán vstupní řetězec.");
7)          return;
```

```
8)      }
9)      Hash has = new Oraculum.Hash();
10)     if (rbMD5.Checked == true)
11)     {
12)          vysledek = has.GeneratorMD5(tbVstup.Text);
13)     }
14)     else if (rbSha1.Checked == true)
15)     {
16)          vysledek = has.GeneratorSHA1(tbVstup.Text);
17)     }
18)     else if (rbSha256.Checked == true)
19)     {
20)          vysledek = has.GeneratorSHA256(tbVstup.Text);
21)     }
22)     else if (rbSha384.Checked == true)
23)     {
24)          vysledek = has.GeneratorSHA384(tbVstup.Text);
25)     }
26)     else if (rbSha512.Checked == true)
27)     {
28)          vysledek = has.GeneratorSHA512(tbVstup.Text);
29)     }
30)     else
31)     {
32)          MessageBox.Show("Není zvolena metoda tvorby");
33)          return;
34)     }
35)     //odeslani k uprave
36)     Uprava cu = new Uprava();
37)     int pozadDelka = cu.Prevod(tbPocetZnaku.Text);
38)     if (pozadDelka == -1)
39)          pozadDelka = 21;
40)     int posledni = cu.Prevod(has.hash16[has.hash16.Length -
                                1].ToString());
41)     int mc = cu.Prevod(tbMCislo.Text);
42)     if (mc == -1)
43)          mc = 11;
44)     int  cp;
45)     cp = ((cu.CH(tbVstup.Text[0].ToString()) + tbVstup.Text.Len-
            gth)%4)+1;
```

```
46)      vysledek = cu.uprava(vysledek, posledni, mc,
                              cp,tbVstup.Text,pozadDelka);
47)      tbVystup.Text = výsledek;
48)  }
```

Na třetím řádku výpisu funkce je deklarovaná textová proměnná *vysledek* a je jí přiřazena prázdná hodnota.

Na řádku 4 je podmínka „if“ jejíž parametr je „*hodnota vlastnosti „Text“ textového pole „tbVstup“ je rovna prázdné hodnotě*“. V okamžiku, kdy bude dané textové pole prázdné, bude proveden zdrojový kód v těle této podmínky. Tělo této podmínky pak obsahuje dva řádky. Na řádku 6 je vyvoláno a zobrazeno, příkazem „MessageBox.Show()“, oznamovací okno, které vypíše textovou hodnotu uvedenou uvnitř závorek. Může se jednat buď o textovou proměnnou, nebo jako v tomto případě pouze o text mezi uvozovkami. Na řádku 7 je příkaz „return“, který ukončí vykonávání této funkce a vrátí běh programu zpět do okna aplikace, kde bude program čekat na činnost uživatele. Tato podmínka je zde, aby nedošlo k pokusu o vytvoření hesla bez vstupního textu, čímž by došlo k chybě programu a pádu aplikace.

Na řádku 9 je deklarována třída *Hash*. Díky tomu bude moci tento formulář využít veřejných funkcí v třídě *Hash.cs*.

Na řádku 10 je podmínka „if“ s parametrem „*označení „rbMD5“ je pravdivé*“. Pokud je *rbMD5* („radiobutton“ s textem „MD5“) vybrán (uprostřed jeho kolečka je tečka), dojde k provedení zdrojového kódu těla této podmínky. Na řádku 12 nejdříve pomocí dříve deklarované třídy *Hash* využije funkci *GenerátorMD5*, kde jako vstupní parametr se použije hodnota vlastnosti „Text“ textového pole *tbVstup*. Po proběhnutí funkce *GeneratorMD5* tak jak byla popsána v kapitole 4.2.2.1 Veřejná textová funkce *GeneratorMD5* je návratová hodnota této funkce přiřazena textové proměnné *vysledek*, deklarované výše a běh programu se přesune na řádek 36 výpisu za tělo příkazu „else“.

Pokud by parametr této podmínky „if“ nebyl splněn, dojde k postupnému ověřování označení ostatních „radiobuttonů“ (s texty „SHA 1“, „SHA 256“, „SHA 384“ a „SHA 512“) za pomoci podmínek „else if“ na řádcích 14, 18, 22 a 26. V případě, že dojde k splnění podmínky u jednoho z těchto „radiobuttonů“, provede se tělo podmínky prakticky stejně, jak bylo popsáno v předchozím odstavci. Jediný rozdíl je v použité veřejné hašovací funkci.

V případě, že by ani jeden z „radiobuttonů“ nebyl označen, dostal by se běh programu do těla příkazu „else“ na řádku 30, který obsahuje dva řádky kódu. V řádku 32 opět vyvoláme a zobrazíme, příkazem „MessageBox.Show()“, oznamovací okno, které nám vypíše text, který je umístěný uvnitř závorek a uvozovek. Řádek 33 zajišťuje příkazem „return“ přerušení vykonávání této funkce. Prakticky toto okno informuje uživatele, že při snaze vytvořit heslo, zapomněl označit metodu, která se má použít při tvorbě.

Na řádku 36 je deklarována třída *Uprava*, která tomuto formuláři následně umožní přistupovat k veřejným funkcím třídy *Uprava*.

Na řádku 37 je volána veřejná funkce *Prevod* z třídy *Uprava*, kde jako vstupní parametr je hodnota vlastnosti „Text“ textového pole *tbPocetZnaku*. Následně je zde deklarována celočíselná proměnná *pozadDelka*. Návratová hodnota volané funkce *Prevod* je pak přiřazena proměnné *pozadDelka*.

Na řádku 38 je podmínka „if“ jejíž parametr je „*hodnota proměnné „pozadDelka“ je rovna „-1“*“. Při splnění tohoto parametru je v těle této podmínky na řádku 39 přiřazena hodnotě proměnné *pozadDelka* hodnota „21“.

Na řádku 40 po konci těla podmínky „if“ je deklarována celočíselná proměnná *posledni*. Zároveň na tomto řádku dochází k volání veřejné funkce *Prevod* v třídě *Uprava*. Vstupní parametr pro funkci *Prevod* je volána veřejná proměnná *hash16* ve třídě *Hash*. Z této veřejné proměnné je pak pro vstupní parametr funkce *Prevod* použit pouze jeden znak. Pozice tohoto znaku v proměnné *hash16* je určena výpočtem (*odečtení hodnoty „1“ od délka textového řetězce proměnné „hash16“ ve třídě „Hash“ získané příkazem „.Length“*). Prakticky je použit poslední znak. Návratová hodnota funkce *Prevod* je pak přiřazena proměnné *posledni*.

Na řádku 41 výpisu těla funkce je deklarována číselná proměnná *mc*. Také je zde volána veřejná funkce *Prevod* z třídy *Uprava*. Jako vstupní parametr pro tuto funkci je použita hodnota vlastnosti „Text“ textového pole *tbMCislo*. Návratová hodnota funkce je přiřazena proměnné *mc*.

Na řádku 42 je podmínka „if“ jejíž parametr je „*hodnota proměnné „mc“ je rovna „-1“*“. Při splnění parametru této podmínky dojde v těle podmínky na řádku 43 k přiřazení hodnoty „11“ proměnné *mc*.

Na řádku 44 je deklarována celočíselná proměnná *cp*. Na řádku 45 je volána veřejná funkce *CH* z třídy *Uprava*. Jako vstupní parametr je použit první znak hodnoty vlastnosti „Text“ textového pole *tbVstup*. K návratové hodnotě této funkce je přičtena hodnota délky textového řetězce ve vlastnosti „Text“ textového pole *tbVstup*, získaná příkazem „.Length“. Výsledek tohoto sčítání je vydělen, metodou modulo, hodnotou „4“. K výsledku dělení je následně přičtena hodnota „1“. Celkový výsledek je pak přiřazen jako hodnota proměnné *cp*.

Na řádku 46 je volána funkce *uprava* z třídy *Uprava*. Jako vstupní parametry jsou použity proměnné *vysledek*, *poslední*, *mc*, *cp*, *pozadDelka* a hodnota vlastnosti „Text“ textového pole *tbVstup*. Návratovou hodnotou volané funkce je pak nahrazena hodnota proměnné *vysledek*.

Na řádku 47 je vlastnosti „Text“ textového pole *tbVystup* přiřazena hodnota proměnné *vysledek*.

4.2.4.5 Soukromá funkce „cbViditelnyVystup_CheckedChanged“

Tato funkce tvoří obslužnou metodu pro vlastnost „CheckedChanged“ (změna zaškrtnutní) zaškrtávacího políčka *cbViditelnyVystup* (umístěné v panelu „Výstup“). Její zdrojový kód vypadá následovně:

```
1)  private void cbViditelnyVystup_CheckedChanged(object sender,
                                                  EventArgs e)
2)  {
3)      if (cbViditelnyVystup.Checked == true)
4)      {
5)          tbVystup.UseSystemPasswordChar = false;
6)      }
7)      else
8)      {
9)          tbVystup.UseSystemPasswordChar = true;
10)     }
11) }
```

Tělo této funkce je rozděleno podmínkou „if - else“ na dvě části, přičemž při vykonávání této funkce se provádí buď jedna, nebo druhá část. Parametrem podmínky „if“ určující, která část se bude vykonávat je „*označení zaškrtávacího políčka „cbViditelnyVystup“ je*

pravdivé“. Pokud je podmínka splněna, zaškrtávací políčko je zaškrtnuté, dojde k provedení těla podmínky „if“.

V těle této podmínky na řádku 5 je vlastnost „UseSystemPasswordChar“ textového pole *tbVystup* nastavena na hodnotu „false“. Prakticky to znamená, že hodnota vlastnosti „Text“ bude zobrazena jako čitelný text.

V případě, že by parametr podmínky „if“ nebyl splněn, dojde k přesunutí běhu programu do těla příkazu „else“. V těle tohoto příkazu na řádku 9 je vlastnost „UseSystemPasswordChar“ textového pole *tbVystup* nastavena na hodnotu „true“. Prakticky to znamená, že hodnota vlastnosti „Text“ nebude zobrazena jako čitelný text, ale jako řada teček.

4.2.4.6 Soukromá funkce „cbViditelnyVstup_CheckedChanged“

Tato funkce zajišťuje obslužnou metodu vlastnosti „.CheckedChanged“ zaškrtávacího políčka *cbViditelnyVstup* (umístněného v panelu „Vstup“). Její zdrojový kód vypadá následovně:

```
1)  private void cbViditelnyVstup_CheckedChanged(object sender,    Even-
                                                          tArgs e)
2)  {
3)      if (cbViditelnyVstup.Checked == true)
4)          tbVstup.UseSystemPasswordChar = false;
5)      else
6)          tbVstup.UseSystemPasswordChar = true;
7)  }
```

Tělo této funkce je stejně jako předchozí funkce rozděleno podmínkou „if – else“ na dvě části, přičemž při volání této funkce dojde vždy k vykonání pouze jedné z těchto částí. Část, která se vykoná, je určena parametrem podmínky „if“, který je *„označení zaškrtávacího tlačítka „cbViditelnyVstup““ je pravdivé.* Pokud je parametr splněn, zaškrtávací políčko *cbViditelnyVstup* je zaškrtnuté, dojde k vykonání těla podmínky „if“. Zde na řádku 4 je nastavována vlastnost „.UseSystemPasswordChar“ textového pole *tbVstup* na hodnotu „false“ (zobrazí viditelný text). V případě, že parametr podmínky není splněn, přesune se běh programu do těla příkazu „else“, kde je jeden řádek kódu. Zde je na řádku 6 výše uvedená vlastnost textového pole nastavena na hodnotu „true“ (skryje viditelný text a zobrazí tečky).

4.2.4.7 Soukromá funkce „bKopirovat_Click“

Tato funkce zajišťuje obslužnou metodu vlastnosti „.Click“ tlačítka *bKopirovat*. Její zdrojový kód vypadá následovně:

```
1)  private void bKopirovat_Click(object sender, EventArgs e)
2)  {
3)      Clipboard.SetText(tbVystup.Text);
4)  }
```

Nastavením vlastnosti „.UseSystemPasswordChar“ textového pole na hodnotu „true“ dojde automaticky k zablokování možnosti kopírovat hodnotu vlastnosti „.Text“ tohoto pole, pomocí klávesových zkratek „Ctrl+C“ nebo kopírováním přes nabídku zobrazující se při kliknutí pravého tlačítka myši do textového pole. Proto bylo vytvořeno tlačítko *bKopirovat* (umístěné v panelu „Výstup“ vlevo dole). Do těla obslužné metody vlastnosti „.Click“ byl na řádku 3 umístěn příkaz „Clipboard.SetText()“, který umožní uložení určité hodnoty do schránky („clipboardu“) Windows. Parametrem tohoto příkazu je pak hodnota vlastnosti „.Text“ textového pole *tbVystup*.

4.3 Postup tvorby hesla

Po spuštění programu se zobrazí chybové hlášení systému Windows, upozorňující na chybné spuštění aplikace a vyzývající uživatele ke kliknutí na tlačítko s nápisem „OK“ umístěné v pravém dolním rohu. Při kliknutí na toto tlačítko se aplikace ukončí.

V pravém dolním rohu okna aplikace je skryté tlačítko. Při kliknutí na toto tlačítko, dojde ke skrytí zobrazované chybové zprávy a tlačítka s nápisem „OK“. Zároveň dojde k zobrazení komponent programu, potřebných k vytvoření hesla.

Zobrazované komponenty můžeme rozdělit do dvou skupin. První skupina jsou vstupní komponenty a druhá jsou výstupní komponenty. Vstupní komponenty umožňují uživateli nastavit způsob tvorby, délku vzniklého hesla, vstupní řetězec, ze kterého je heslo vytvářeno a modifikační číslo ovlivňující tvorbu hesla. Vstupní komponenty můžeme dále rozdělit na povinné a nepovinné. Povinné komponenty slouží k zadávání vstupního řetězce a k volbě hašovací funkce, která bude při tvorbě použita. Nepovinné komponenty slouží k zadávání modifikačního čísla a k nastavení délky vytvořeného hesla. Výstupní komponenty slouží k zobrazení vzniklého hesla případně jeho kopírování.

Samotný proces tvorby hesla začíná po kliknutí na tlačítko „Vytvoř“. Nejdříve dojde k vyhodnocení, zda všechny vstupní povinné komponenty, mají požadované hodnoty. V případě, že nemají, je uživatel upozorněn vyskakujícím oknem o vzniklém nedostatku. V případě, že vstupní povinné komponenty mají požadované hodnoty, dojde k ověření, zda byly zadány patřičné hodnoty i do nepovinných komponent. V případě že nikoliv, dojde k použití přednastavených hodnot.

Po tomto ověření, program odesílá vstupní textový řetězec do uživatelem zvoleného hašovacího algoritmu. Bitový výstup z hašovací funkce je pak převeden do hexadecimální soustavy. Poslední znak z takto zformátovaného haše je pak uložen do proměnné pro pozdější použití. Bitový výstup z hašovací funkce je pak převeden i do formátu „Base64“ a odeslán k následné úpravě. Úprava takto zformátovaného výstupu následně probíhá v cyklech. Pro samotnou úpravu jsou důležité různé modifikační údaje.

Jedním z údajů je první znak vstupního textového řetězce a délka tohoto řetězce. Z těchto údajů je vypočteno číslo, které určuje množství znaků používaných při tvorbě hesla v jednom cyklu.

Dalším údajem je hodnota modifikačního čísla, které zadal uživatel. Tato hodnota ovlivňuje míru posunu po jednotlivých pozicích vzniklého haše. Při výpočtu posunu při každém cyklu jsou ještě zohledňovány znaky na jednotlivých pozicích vstupního textového řetězce zadaného uživatelem. Program při každém cyklu úpravy cyklicky prochází i tento vstupní řetězec. V závislosti na druhu znaku na dané pozici dochází k použití jedné ze čtyř výpočtových metod.

Samotný cyklus pak probíhá tak, že nejdříve dojde k určení metody výpočtu pozice znaku, který bude použit při tvorbě hesla. Následně za použití modifikačního čísla zadaného uživatelem, dojde k samotnému výpočtu pozice. Počínaje touto pozicí je vybrán počet znaků odpovídající hodnotě vypočtené z prvního znaku vstupního řetězce a délky tohoto řetězce. (například pokud počet znaků je vypočten na 3 a pozice znaku byla vypočtena na 7, dojde k použití znaků na pozici 7, 8 a 9). Tyto znaky jsou pak připsány k znakům, vybraným v předchozích cyklech. Takovéto cykly se opakují stále dokola, až do okamžiku, kdy sestavované heslo nemá minimálně přednastavenou, nebo uživatelem požadovanou velikost.

V okamžiku kdy textový řetězec vytvořený těmito cykly má danou velikost, dojde k úpravě, která zajistí, že každé vytvořené heslo bude obsahovat i prvky ze znakové sady

speciálních znaků. Zde dochází k použití posledního znaku hexadecimálního výstupu hašovací funkce. V závislosti na druhu znaku jsou znaky v textovém řetězci na pozicích, kam se budou speciální znaky vkládat, buď posunuty o pozici dozadu, nebo přepsány. V praxi to znamená, že buď dojde k odstranění původního znaku na pozici, nebo jeho posunutí o pozici dozadu. Tím dojde ke zvětšení textového řetězce. Při závěrečném zkrácení tohoto řetězce na požadovanou velikost pak dojde k odstranění posledního znaku místo znaku na vkládané pozici.

Na závěr je výstupní textový řetězec zkrácen na požadovanou délku a „zobrazen“ ve výstupním textovém poli.

5 VYHODNOCENÍ APLIKACE „ORACULUM“

Při základním nastavení a použití pouze nezbytně nutných údajů pro tvorbu hesla, vytváří tato aplikace 21-místné heslo, které je složeno ze znaků čtyř znakových sad. („a-z“, „A-Z“, „0-9“ a „speciálních znaků“). Jako příklad, můžeme vytvořit hesla z textového řetězce „heslo“. Délka hesla zůstane nastavena na přednastavených 21 míst. Změna bude pouze v modifikačním čísle. Pro tento příklad, použijeme přednastavené modifikační číslo 11 a jako druhé číslo pro porovnávání, zvolíme modifikační číslo 5.

Tabulka 3 Použití dvou modifikačních čísel při stejném textovém řetězci. *Zdroj: vlastní*

vstup „heslo“	modifikační číslo - 11	modifikační číslo - 5
MD5	Tf6wu^BG*ppJ-tK*V2ugG	Gppug^Gl*V2J-tK*ppwuB
SHA 1	bgF7V^6*bK-BIK*=bgT4I	6fb7V^2*o1-fT4*A2ObXE
SHA 256	oAU=V^rE*8oO-oL*TP4q1	U=VAy^2o*AU9-ah*Kqhdw
SHA 384	PMmk1^TG*Ge5-iV*AJKw9	GGe1Y^5Y*5i2-iJ*VXLVK
SHA 512	mA=ON^nJ*Qmn-P9*BDHfd	8ON08^Nf*o8j-sU*i6U+u

V tabulce 3 jsou vidět vytvořená hesla za pomoci jednotlivých hašovacích funkcí, při stejném vstupním řetězci „heslo“ a při dvou různých modifikačních číslech (11 a 5). Všechna vytvořená hesla v této tabulce jsou rozdílné textové řetězce, všechny obsahují znaky čtyř znakových sad a všechny mají délku 21 míst. Prakticky jen díky změně hašovací funkce a modifikačního čísla jsme z jednoho vstupu vytvořili 10 různých hesel.

Takto by se sice mohlo zdát, že s jedním vstupním textem jsme schopni za pomoci modifikačního čísla a jedné hašovací funkce vytvořit nekonečné množství hesel. Bohužel tomu tak není. Prakticky jsme schopni vytvořit množství hesel v závislosti na délce výstupu z hašovací funkce ve formátu „Base64“. Důvodem je cyklické procházení tohoto řetězce. V okamžiku překročení jeho délky dojde díky operacím modulo k přetečení zpět na čísla menší nežli je délka hesla. Z tohoto důvodu budou mít například, při stejném vstupu a použití funkce MD5, modifikační čísla 11 a 35 stejný výstup (11 % 24 = 11 a 35 % 24 = 11). To znamená, že u funkce MD5 je to 24 možných hesel z jednoho vstupu. U funkce SHA 1 je to 28 možností, u SHA 256 je to 44 možností, u SHA 384 je to 64 možností a u SHA 512 je to 88 možností. Z toho vyplývá, že při jednom vstupním řetězci a za použití všech hašovacích

funkcí a všech možných modifikačních čísel, by měla být aplikace „Oraculum“ schopna vytvořit 248 různých hesel.

Při pohledu na výstupy v tabulce 3 zjistíme, že všechny hesla v tabulce obsahují znaky čtyř znakových sad, a mají délku 21 míst. Při neznalosti způsobu tvorby hesla by toto nastavení mělo $1{,}19 * 10^{40}$ možností (81^{21}) a v souladu s tabulkou 2 byla entropie takového hesla byla rovna 133,1379 bitu ($21 * \log_2 81$). Při znalosti způsobu tvorby hesla by bylo nutné tyto výpočty lehce upravit. Důvodem je vědomí, že formát „Base64“ nemá 81 variant znaků, ale pouze 64. Díky algoritmu tvorby hesla lze jako další znak připočíst i „=“ a následně jsou přidány ještě další 3 znaky, které nejsou obsaženy v možnostech formátu „Base64“. Proto je nutné upravit výpočty na $3{,}03 * 10^{38}$ možných variant (68^{21}) a entropii rovnou 127,8375 bitu ($21 * \log_2 68 = 21 * 6{,}0875$). Z tohoto pohledu by takovéto heslo mohlo být označeno za silné heslo.

Další úhel pohledu, kterým je potřeba se na takováto hesla podívat, je v rámci odolnosti takovýchto hesel vůči jednotlivým útokům na hesla. Při pohledu na skladbu textového řetězce hesla, by takovéto heslo mělo být odolné vůči různým druhům slovníkových útoků. Díky 248 možnostem výstupu při použití jednoho vstupního textového řetězce, by tato aplikace měla poskytovat uživateli dostatečný prostor při tvorbě hesla bez nutnosti pamatovat si spoustu složitých textových řetězců. Tím by takováto hesla měla být odolná i před útoky za pomoci recyklace hesel. V rámci útoků hrubou silou by tato hesla měl být dost dlouhá a složitá, aby nedošlo k jejich prolomení v pro útočníka využitelném čase (řády hodin či dnů).

Z výše uvedených důvodů lze hesla vytvořená aplikací „Oraculum“ označit za silná a bezpečná.

ZÁVĚR

Tato bakalářská práce měla za úkol prověřit a otestovat, zda lze za použití hašovacích funkcí vytvářet silná a bezpečná hesla. Obsah předchozích kapitol mne přesvědčil, že využití hašovacích funkcí při tvorbě přístupových hesel může být jednou z cest, která umožní vytvářet silná a bezpečná hesla. Zároveň je ale důležité si uvědomit, že základní vlastnosti hašovacích funkcí nabízející silné argumenty pro vytváření bezpečných hesel tímto způsobem, mají i svá omezení a záludnosti, které mohou být zároveň i silnými argumenty proti využívání tohoto způsobu tvorby.

Jednou z těchto záludností je skutečnost, že hašovací funkce mají standardně výstup v hexadecimálním formátu. Z toho vyplývá, že heslo není tvořeno 62 možnými znaky na každé pozici („a-z“ 26 možností, „A-Z“ 26 možností, „0-9“ 10 možností) ale pouze 16 možnými znaky na každé pozici („0-F“ hexadecimální soustavy). Tím dojde k razantnímu snížení možných variant hesel.

S výše uvedeným souvisí i případné přetékání. V okamžiku, kdy uživatel vytvoří heslo za pomoci hašovací funkce SHA 512 (heslo o délce 128 míst a $1,3 * 10^{154}$ možných variant). Následně toto heslo použije u poskytovatele, který toto heslo zahašuje hašovací funkcí MD5, kde vzniklý haš, bude mít pouze 32 míst a $3,4 * 10^{38}$ možných variant. V takovém případě dojde při hašování MD5 k několika násobnému přetečení a může tak vzniknout kolize například s hašem písmene „a“, což může oslabit bezpečnost uživatelova hesla.

Při tvorbě aplikace Oraculum jsem se snažil takovýmto nedostatkům vyhnout, nebo jim zabránit. Myslím si, že aplikace Oraculum by mohla najít využití u jednotlivců, nebo organizací, kde jsou lidé nuceni přihlašovat se k různým aplikacím za pomoci mnoha hesel a u kterých jsou díky bezpečnostní politice organizace, nebo poskytovatele služby nuceni k pravidelným obměnám těchto hesel.

Aplikace Oraculum využívá při tvorbě hesla jako vstupní hodnotu pro hašovací funkci textový řetězec. Pro ověření, zda tato metoda tvorby je použitelná či nikoliv, je takováto vstupní hodnota dostatečná. Je však nutné si uvědomit, že hašovací funkce nevytváří haše pouze z textových řetězců. Tyto funkce jsou schopné vytvářet haše z jakýchkoliv vstupních dat. Jako vstupní hodnota by tak mohl být použit například obrázek, video, složka, soubor, databáze a jiné. Přitom například u obrázku stačí, aby uživatel změnil pouze jeden jediný pixel. Získal by tím jedinečný obrázek, který může sloužit jako vstupní hodnota pro hašovací

funkci. Toto by mohla být cesta jak dále vyvíjet tuto aplikaci. Pak už by uživatel nebyl odkázán pouze na textové řetězce, ale mohl by se přihlašovat prakticky za pomocí čehokoliv.

SEZNAM POUŽITÉ LITERATURY

[1] JIRÁSEK, Petr, Luděk NOVÁK a Josef POŽÁR. *Výkladový slovník kybernetické bezpečnosti*. Třetí doplněné a upravené vydání. Praha: Policejní akademie České republiky v Praze, 2015. ISBN ISBN 978-80-7251-436-6.

[2] KNOPOVÁ, Martina. Bezpečnost dat v informačních systémech. Ikaros [online]. 2011, ročník 15, číslo 6 [cit. 2018-03-05]. ISSN 1212-5075. Dostupné z: https://ikaros.cz/bezpecnost-dat-v-informacnich-systemech

[3] *Software zdarma: S dobrým správcem bude jedno heslo vládnout všem* [online]. Technet.cz, 2016 [cit. 2018-03-05]. Dostupné z: https://technet.idnes.cz/spravce-hesel-0di-/software.aspx?c=A160114_112011_software_dvr

[4] JAŠEK, doc. Mgr. Roman, Ph.D. a MALANÍK, Ing. David, Ph.D. *Bezpečnost informačních systémů*. Zlín, 2013. ISBN 978 - 80 - 7454 - 312 - 8.

[5] *Ve čtvrtek je Světový den hesel. Jsou ta vaše bezpečná?* [online]. SecurityWorld, 2017 [cit. 2018-03-05]. Dostupné z: https://computerworld.cz/securityworld/ve-ctvrtek-je-svetovy-den-hesel-jsou-ta-vase-bezpecna-53853?utm_source=rss&utm_medium=web&utm_campaign=rss

[6] *Lámání hesel* [online]. clever and smart, 2012 [cit. 2018-03-05]. Dostupné z: http://www.cleverandsmart.cz/lamani-hesel/

[7] *Lámání hesel: rainbow tables* [online]. clever and smart, 2012 [cit. 2018-03-05]. Dostupné z: http://www.cleverandsmart.cz/lamani-hesel-rainbow-tables/

[8] KOLOUCH, JUDr. Jan, Ph.D. *CyberCrime*. Praha: nakladatelství Milan Hodek, 2016. ISBN 978 - 80 - 88168-15-7.

[9] *Správné měření síly hesla* [online]. Cleverweb.cz, 2015 [cit. 2018-03-05]. Dostupné z: https://www.cleverweb.cz/spravne-mereni-sily-hesla/

[10] *Teorie informace* [online]. [cit. 2018-03-05]. Dostupné z: biofyzika.upol.cz/userfiles/file/biokybernetika_6_informace.doc

[11] LÓRENCZ, CSC., doc. Ing. Róbert. *Hašovací funkce: MD5,SHA-x, HMAC* [online]. ČVUT FIT, 2011 [cit. 2017-03-27]. Dostupné z: https://edux.fit.cvut.cz/oppa/BI-BEZ/prednasky/bez5.pdf

[12] MACHÁČ, David. *MD5 hešovací funkce a krtyptografické hešovací funkce obecně* [online]. Praha: FJFI ČVUT v Praze, 2010 [cit. 2018-03-05]. Dostupné z: https://kmlinux.fjfi.cvut.cz/~balkolub/Vyuka/Machac.pdf

[13] JAŠEK, prof. Mgr. Roman, Ph.D. a OULEHLA, Ing. Milan. *Moderní Kryptografie: Průvodce světem šifrování*. 1. Praha: IFP Publishing, 2017. ISBN 978-80-87383-67-4

[14] *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions* [online]. NIST, 2015 [cit. 2018-03-05]. Dostupné z: https://www.nist.gov/publications/sha-3-standard-permutation-based-hash-and-extendable-output-functions?pub_id=919061

[15] *Secure Hash Standard (SHS)* [online]. 2012 [cit. 2018-05-02]. Dostupné z: https://csrc.nist.gov/CSRC/media/Publications/fips/180/4/archive/2012-03-06/documents/Draft-FIPS180-4_Feb2011.pdf

[16] *HashAlgorithm Class* [online]. Microsoft [cit. 2018-03-05]. Dostupné z: https://msdn.microsoft.com/en-us/library/system.security.cryptography.hashalgorithm(v=vs.110).aspx

[17] *Standard Numeric Format Strings* [online]. Microsoft, 2017 [cit. 2018-03-05]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/standard/base-types/standard-numeric-format-strings

[18] *What is base 64 encoding used for?* [online]. Stackoverflow.com, 2017 [cit. 2018-03-05]. Dostupné z: https://stackoverflow.com/questions/201479/what-is-base-64-encoding-used-for

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

b	bit (binary digit – číslice dvojkové soustavy), základní jednotka informace
IT	informační technologie
MD5	Message-digest algorithm
NIST	National Institute of Standards and Technology (Národní institut pro standardy a technologie v USA)
NSA	National Security Agency (Národní bezpečnostní agentura v USA)
PIN	osobní identifikační číslo z anglického personal identifacation number
SHA-X	Secure hash algorithm

SEZNAM OBRÁZKŮ

Obrázek 1: Chybové hlášení Windows. *Zdroj: vlastní*.....37
Obrázek 2: Zvýrazněné tlačítko pro zpřístupnění komponent. *Zdroj: vlastní*.....37
Obrázek 3: Aplikace se zpřístupněnými komponentami. *Zdroj: vlastní*.....37
Obrázek 4: Skrytý vstupní text. *Zdroj: vlastní*.....38
Obrázek 5: Viditelný vstupní text. *Zdroj: vlastní*.....38
Obrázek 6: Chybové hlášení při chybějícím vstupním řetězci. *Zdroj: vlastní*.....39
Obrázek 7: Chybové hlášení při chybějícím nastavení parametru tvorby. *Zdroj: vlastní*.....40
Obrázek 8: „Zobrazeno“ heslo ve výstupním textovém poli. *Zdroj: vlastní*.....41
Obrázek 9: Viditelný výstupní řetězec. *Zdroj: vlastní*.....41

SEZNAM TABULEK

Tabulka 1: Znakové sady. *Zdroj: vlastní* 17

Tabulka 2: Entropie jedné pozice hesla v závislosti na použitých znakových sadách. *Zdroj: vlastní* 21

Tabulka 3 Použití dvou modifikačních čísel při stejném textovém řetězci. *Zdroj: vlastní* 67

SEZNAM PŘÍLOH

[1] aplikace Oraculum

[2] zdrojový kód aplikace Oraculum