FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Bezpečnostní analýza šifry Solitaire
Student:	Vojtěch David
Vedoucí:	Ing. Josef Kokeš
Studijní program:	Informatika
Studijní obor:	Bezpečnost a informační technologie
Katedra:	Katedra počítačových systémů
Platnost zadání:	Do konce letního semestru 2019/20

Pokyny pro vypracování

Nastudujte principy manuální šifry Solitaire (https://www.schneier.com/academic/solitaire/).
Implementujte šifru ve zvoleném programovacím jazyku, ověřte funkčnost šifrování i dešifrování.
Analyzujte chování šifry v reálných situacích: Klíč vs. heslo, délka hesla, náhodnost hesla, závislosti ve vygenerovaném keystreamu.
Zhodnoťte dosažené výsledky z pohledu bezpečnosti šifry i její praktické uplatnitelnosti.
Naleznete-li zranitelnosti, diskutujte možnosti jejich omezení či eliminace.

Seznam odborné literatury

Dodá vedoucí práce.

prof. Ing. Pavel Tvrdík, CSc.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 13. prosince 2018

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Bezpečnostní analýza šifry Solitaire

Vojtěch David

Bezpečnost a informační technologie
Vedoucí práce: Ing. Josef Kokeš

6. května 2019

Poděkování

Velice děkuji svému vedoucímu práce za ochotu a za inspirující zodpovědnost a pracovitost, se kterou přistupuje ke své práci. Dále děkuji celé své rodinně za to, že mi vytvořila ideální prostředí ke studiu. V neposlední řadě děkuji své přítelkyni za všudypřítomnou podporu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 6. května 2019 .....

Odkaz na tuto práci

David, Vojtěch. *Bezpečnostní analýza šifry Solitaire*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

Ve své práci se zabývám bezpečnostní analýzou ruční šifry Solitaire od kryptologa Bruce Schneiera. V textu je detailně popsán její princip a vysvětleno, jak šifru bezpečně používat. Dále zkoumám možné slabiny šifry, konkrétně jsem se zaměřil na nenáhodnost ve výstupu, znovupoužití stejného klíče a náhodnost pouzic jokerů v balíčku. Ukazuji, jaký je rozdíl mezi použitím hesla a klíče a která z těchto variant je lépe využitelná v praxi. Dokazuji, že bezpečné heslo musí být minimálně 80 znaků dlouhé. Celá analýza je založená na mé vlastní inmplementaci Solitaire v jazyce C++, schopnou šifrovat a dešifrovat s použitím jak klíče, tak hesla.

Klíčová slova Solitaire, Bruce Schneier, ruční šifra, proudová šifra, NIST testy

Abstract

In my work I deal with the principles of the Solitaire pen-and-paper cipher by a cryptologist Bruce Schneier. The text describes in details its principles with a description of safe usage. Next, I examine the cipher vulnerabilities as bias in the stream key, reusing the same key and nonrandomness of the jokers possition in the deck. I also show the difference between using a password and using a key. I find out, that password must be at least 80 charaters long. Then I show which of these variants is better to use in practice. To do so, I use the Solitaire implementation written in C++, capable of encryption and decryption using either the key or the password.

Keywords Solitaire, Bruce Schneier, pen-and-paper cipher, stream cipher, NIST tests

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Počítače jsou v civilizovaném světě v různých formách všude kolem nás. Dnes by se již téměř žádná profese neobešla bez pomoci počítače. Lidé však nevyužívají jen rychlosti, které počítače dosahují při práci s čísly nebo daty. Obrovská výhoda počítačů je, že nikdy, na rozdíl od lidí, neztratí pozornost, nikdy se při dobře popsaném úkolu nespletou, nepotřebují spát, nevadí jim rutinní práce... Dalších takovýchto příkladů by se dalo najít mnoho.

Kryptografie není v tomto výjimkou, stejně jako lidstvo si prošla svým vývojem. Od primitivních metod šifrování jsme se posunuli k bezpečným a pro stroj rychle vykonatelným šifrám. Má tedy smysl se zabývat i nějakým jiným způsobem utajování zpráv, než je používání počítače s vhodným softwarem?

Práce se zaměřuje na ruční šifru od Bruce Schneiera zvanou Solitaire. Jedná se o šifru, ke které není potřeba nic jiného než papír, tužka a balíček karet. Šifra je být navržena tak, aby byly její kroky snadno proveditelné pro člověka bez použití počítače.

Hlavní motivací, proč jsem si toto téma vybral, bylo, že se mi líbí pohlížet na věci z různých úhlů a lákala mě myšlenka, že existuje šifra, kterou lze aplikovat pouze v ruce. Přitom má dostatečné vlastnosti, aby odolala nejmodernějším útokům. Také mě zaujalo, že je zapotřebí pouze balíčeku karet, který je naprosto běžnou věcí.

V první kapitole se zaměřím na argumentaci, proč je potřebné se zabývat studiem ručních šifer a definuji potřebné pojmy. V kapitole druhé popíši jednotlité šifrovací a dešifrovací kroky Solitaire. Ve třetí kapitole analyzuji, jak používat klíč a jak heslo. V kapitole čtvrté se podívám na fungování moderní a používané šifry ChaCha20 a porovnám její vlastnosti se Solitare. V kapitole páté píši o užití dvakrát stejného klíče a problémech, které to způsobuje. Následující kapitola je o nenáhodnostech v proudovém klíči a další kapitola analyzuje pozice jokerů (žolíků) v balíčku. Tato kapitola popisuje slabinu šifry, která ještě nikdy nebyla popsána a obsahuje kapitola také obsahuje důkaz této slabiny. V předposlední osmé kapitole podrobím výstup ze šifry

testům na náhodnost a poslední kapitola pohlíží na Solitaire z hlediska praktického využití.
Cílem této práce je zkoumat karetní ruční šifru Solitaire. Podrobně proberu, jaké jsou její principy a jaké jsou možnosti jejího využití. Solitaire porovnám s používanou moderní šifrou ChaCha a poukáži na podobnosti a rozdíly. Prozkoumám možné slabiny a zranitelnosti Solitaire, ať již popsané nebo mnou objevené, a ty otestuji na své implementaci v jazyce C++, jejíž správnost ověřuji na testovacích vektorech. Nalezené slabiny zanalyzuji a navrhnu řešení, pokud se slabina ukáže být závažnou. Poté vyzkouším, jak je obtížné a časově náročné zašifrovat nějaký text a doporučím, jak šifru co nejlépe a nejrychleji používat. Z těchto analýz ustanovím, za jakých podmínek, je bezpečné a praktické Solitaire používat.

Kapitola 1

Teoretický úvod

V této kapitole vysvětluji, proč je vhodné zabývat se tématem ručních šifer a za jakých okolností se může více vyplatit použít šifru ruční místo počítačový program. Dále definuji, co je vůbec možné považovat za ruční šifry a uvedu několik příkladů ručních šifer. Na závěr této kapitoly uvedu práce, které již na Solitaire vznikly a popíši, na které ve své práci navazuji.

1.1 Proč ruční šifry používat

Dovedu si představit, že zpravodajské služby musí řešit situace, kdy jejich zaměstnanci chtějí utajit nějaké písemnosti, ale poslat je s počítačem či mobilem, který má na to určený software, je příliš nápadné. Problém také může nastat při cestě do zahraničí s počítačem obsahující zašifrované soubory. Velká Británie uvádí v „Regulation of Investigatory Powers Act 2000“ v sekci 49, že britské úřady mají pravomoc požadovat převedení chráněných (zašifrovaných) dat do snadno čitelné podoby, pokud je podezření, že osoba na to má znalosti, tedy zná klíč nebo heslo. Odmítnutí či nesplnění může být potrestáno odnětím svobody ve výši až 2 roky. V případě podezření z narušení národní bezpečnosti či dětské pornografie může být trest až 5letý [1]. V praxi to znamená, že se např. na letišt musí odemknout počítač či mobil, a když tam celníci naleznou zašifrovaná data, tak i ta se musí odšifrovat. Na tento požadavek nemusejí mít autority povolení soudu. Hodně podobné zákony platí v USA nebo v Austrálii [1]. Může se tedy hodit nějaký nenápadný způsob šifrování, který by zůstal nepovšimnut. A právě šifrování bez použití počítače by mohlo protokolem řídící se úředníky ošálit.

Jiná validní možností je, že k dispozici nebude mít jedna z komunikujících stran počítač, ale i přesto je nutnost komunikovat šifrovaně. Dalším problémem může být, že elektronická zařízení potřebují zdroj energie, což v extrémních podmínkách může být obtížné zajistit. Ve všech těchto případech považuji za možné řešení použití ručních šifer.

Hlavně nanápadnost a neobvyklost považuji za přednosti, které Solitaire v praxi má. Vždy se dá velmi dobře najít důvod, proč vezu balíček karet, ale vysvětlovat, proč mám v počítači šifrovací software je podstatně složitější.

Solitaire potřebuje k šifrování symetrický klíč. V kapitole 3 popisuji, že lze klíč získat pomocí hesla. Solitaire by tedy šel použít k přenosu symetrického klíče, který by se dal použít i na počítačovou šifru.

1.2 Ruční šifra

Za ruční šifru považuji takovou metodu, která zajistí, aby zprávy, na které tuto metodu aplikuji, nebylo možné číst kýmkoliv, ale pouze osobami, které jsou k tomu určené, tedy vlastníky dešifrovacího klíče. Důležité je, že k šifrování ani dešifrování není potřeba žádné elektronické zařízení, ale pouze papír, tužka anebo mobilní, málo nápadná pomůcka. Čas zde hraje nemalou roli. Čistě teoreticky i bloková šifra AES (Advanced Encryption Standard) by se dala považovat za šifru proveditelnou v ruce, ale operace v ní jsou pro člověka tak časově náročné, že v praxi toto možné není.

Ruční šifry jsou tedy takové metody, které zašifrují požadované množství informací za přijatelný čas, a to jen s použitím papíru a tužky. Je také možné použít pomůcky, které neobsahují žádnou elektroniku, tak jako karty v případě Solitaire. Slovní spojení „za přijatelný čas“ je obecné z důvodu, že někomu stačí zašifrovat krátkou zprávu, ale silnou šifrou, která trvá déle, avšak někdo potřebuje jen slabší, rychlejší šifru na hodně dat.

1.3 Ruční šifry v historii

Lidé se již v dávné historii snažili utajit zprávy tak, aby je nemohl číst nikdo nepovolaný. První zaznamenané pokusy se datují 4 000 let př. n. l., kdy byly ve starověkém Egyptě vytesány neobvyklé hieroglyfy, které substituovaly skutečné a tím zatajovaly význam zprávy. Jedná se o stejnou metodu, kterou používá notoricky známá Césarova šifra [2].

Ve Spartě v Řecku 500 let př. n. l. se vyvinula metoda zvaná Scytale. Spočívala v tom, že se vzala tyč o určitém průměru a okolo ní se omotal spirálovitě pergamen. Poté se po délce tyče napsala zpráva. Ta je dešifrovatelná pouze s tyčí stejného průměru [2].

Šifrování však nebylo pouze o substituci či transformaci písmen. Peršan Histiaeus právě ve válce s Řeckem oholil hlavu poslovi a vytetoval mu na ni zprávu. Poté počkal, až vlasy dostatečně dorostou, a tímto způsobem ukryl zprávu [3]. Tomuto způsobu ukrytí zprávy se říká steganografie.

Steganografie nepoužívá pro ukrytí zprávy matematické principy, ale její síla spočívá v neobvyklém nebo nečekaném způsobu, jak se zpráva ukryla[2]. Solitaire obsahuje některé aspekty ze steganografie a to, že se šifruje za použití karet. Kdo by to očekával?

Princip šifer se do první světové války moc neměnil. Téměř vždy se jednalo o nějaký druh jednoduché substituce znaků nebo jejich transformace. Změna přišla při vzniku šifry zvané Enigma, která byla založena na složitějších substitucích, které vznikaly otáčením rotorů. Nejprve byla využívána k civilním účelům, ale na konci první a hlavně v průběhu druhé světové války byla používána upravená verze k armádním účelům. Sice byla na konci druhé světové války prolomena z důvodu chyb operátorů a šparného návrhu, jako například, že se písmeno nemohlo namapovat samo na sebe [2], ale jednalo se o elektromechanický stroj provádějící šifrování automaticky. Tímto vynálezem doba, kdy ruční šifry hrály prim na poli kryptografie, nadobro skončila.

1.4 Práce na téma Solitaire

Šifra Solitaire, ač pro zábavu vytvořená, má zajímavou vlastnost, že je schopná odolat útokům a výpočetní síle dnešních počítačů [4]. Analýzou vlastností šifry Solitaire se zabývá několik prací. Boris Pogorelov s Marina Pudovkina v [5] zkoumá vlastnosti proudového klíče, v případě jeho dělení na skupiny. V práci [6] na druhou stranu skupina autorů sleduje výskyty cyklů a s jakou pravděpodobností se objevují v proudovém klíči Solitaire. Paul Crowley se v [7] zabýval nenáhodnostmi v proudovém klíči a také poukázal na to, že k šifře nejde vždy udělat inverzní krok. Já ve své práci z části navazuji na práci P. Crowleyho a přidávám některá další pozorování, která se týkají stavů karet v balíčku. V kapitolách na začátku uvádím teoretický základ a poté předkládám výsledky ze své praktické práce.

Kapitola 2

Solitaire

Tuto kapitolu věnuji popisu šifry Solitaire. Celý tento popis výchází z článku [4], který autor šifry Bruce Schneier umístil na svůj web.

2.1 Úvod k Solitaire

Solitaire byla vytvořena, jak již bylo řečeno, americkým kryptologem Brucem Schneierem. Učinil tak na žádost Neala Stephensona pro jeho knihu Cryptonomicon. V této knize komunikují dvě postavy právě za použití Solitaire (v knize je šifra pojmenována Pontifex) a využívají její nenápadnosti a nepotřebnosti počítače.

Bruce Schneier píše ve svém popisu [4], že šifru navrhl tak, aby odolala i nejmodernějším počítačům a nejlepším kryptoanalytikům. Jedná se tedy o šifru, která má minimální technologické nároky, ale měla by odolat těm nejlepším technologiím. Její bezpečnost spočívá v tom, že balíček karet se specifickým typem míchání přemění na PRNG (Pseudo Random Number Generator). Pseudonáhodná čísla, která z něj odečítám, použiji k substituci OT (otevřený text), který chci zašifrovat.

2.2 Solitaire jako proudová šifra

Solitaire lze chápat jako symetrickou proudovou šifru. Proudová šifra zpracovává po znacích OT, na rozdíl od blokových šifer, kde se OT zpracovává po blocích. Další významný rozdíl je, že se nepoužívá stále stejná tranformace, jak tomu je u blokových šifer, ale pro každý znak se mění. Proudové šifry se skládají z generátoru, který pro každý klíč k vygeneruje pseudonáhodný proudový klíč $k_1k_2\ldots$ V tomto případě proud čísel od 1 do 52, u současných počítačových šifer proud bitů. Pro Solitaire je generátorem balíček karet s operacemi popsanými níže. Dále má proudová šifra dvě zobrazení E a D, která každému klíči k přiřadí transformaci E_k pro šifrování a D_k pro dešifrování [8].

Když OT je $m = m_1 m_2 \ldots$ a proudový klíč $k = k_1 k_2 \ldots$ poté transformace E_k pro Solitaire je definována následovně:

$$E_{k_1}(m_1) = |m_1 + k_1|_{26},\ E_{k_2}(m_2) = |m_2 + k_2|_{26}, \ldots$$

Vzhledem k tomu, že Solitaire bere OT jen jako proud velkých písmen anglické abecedy, můžeme o transformaci E_{k_1} mluvit jako o posun znaku m_1 o k_1 míst v rámci anglické abecedy. ŠT (šifrovaný text) je $c = c_1 c_2 \ldots$, transformace D_s pro Solitaire je definována velmi podobně, jen s inverzním posunem.

$$D_{k_1}(c_1) = |c_1 – k_1|_{26},\ D_{k_2}(c_2) = |c_2 – k_2|_{26}, \ldots$$

Tedy posun znaku c_1 o zápornou hodnotu k_1 v rámci anglické abecedy. Zašifrování OT $m = m_1 m_2 \ldots$ na ŠT (šifrovaný text) $c = c_1 c_2 \ldots$ probíhá následovně:

$$c_1 = E_{k_1}(m_1),\ c_2 = E_{k_2}(m_2), \ldots$$

Dešifrování ŠT c na OT m je podle následujícího vztahu:

$$m_1 = D_{k_1}(c_1),\ m_2 = D_{k_1}(c_2), \ldots$$

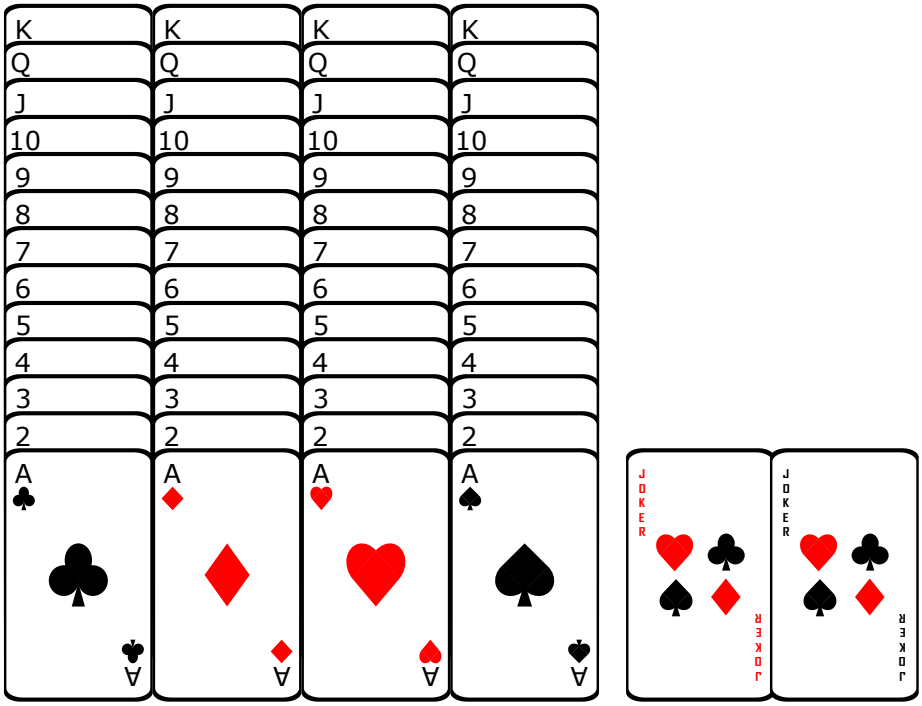
2.3 Požadavky Solitaire

Jak jsem již několikrát zmínil, proudový klíč se generuje pomocí balíčku karet. Tento balíček musí obsahovat 52 karet, tedy jde o balíček vhodný pro hry Poker nebo Bridge. Balíček se skládá z karet 13 různých hodnot: A, 2, 3, 4, 5, 6, 7, 8, 9, 10, kluk, dáma a král, a ze 4 barev: trefy, káry, srdce a piky. Velice důležité je, aby tento balíček obsahoval dva od sebe odlišitelné jokery.[1] Abych je od sebe dále v textu od sebe odlišil, používám stejné označení jako Bruce Schneier, a to joker A a joker B. V praxi jde např. o červeného a černého jokera. Obrázek 2.1. ukazuje, jak by mohl takový balíček vypadat.

Mám tedy balíček 54 karet, přičemž každá je od všech ostatních odlišitelná. Tyto karty si držím v balíčku, tedy všechny karty budou na sobě položené. V rámci balíčku rozlišuji vrchní kartu a spodní kartu. Vrchní karta je nejhornější karta, když jsou karty v balíčku položené lícem nahoru.[2] Analogicky je definovaná spodní karta. Balíček karet má předem stanovené pořadí karet před prvním krokem. To je velice důležitá část, protože toto pořadí karet budu nazývat klíčem a o něj je opřena velká část bezpečnosti Solitaire. Proudový klíč vychází přímo z něj. O tom, jak zvolit pořadí (dále již označováno jako klíč), a o tom, jestli je klíč dostatečně dlouhý, mluvím v následující kapitole.

[1] Používám slovo joker místo žolík, aby si znalci Kanasty nepletli žolíka s dvojkou.

[2] Když se šifruje, je potřeba hodnoty karet vidět. Proto je praktičtější horní kartu definovat jinak, než je přirozené při karetních hrách.



Obrázek 2.1: Požadavky Solitaire

2.4 Generování proudového klíče

Pro generování proudového klíče je potřeba stále dokola opakovat následujících 5 kroků.

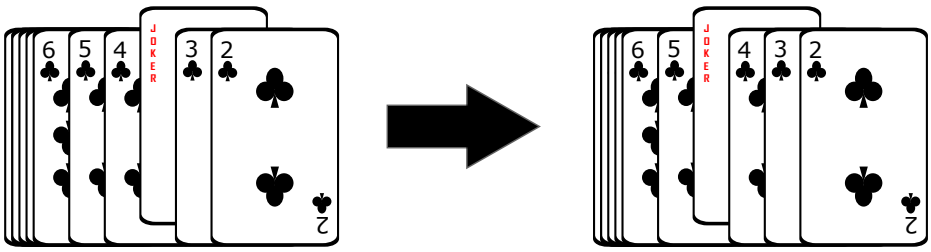
2.4.1 Nalezení jokera A

Pro vykonání prvního kroku musím najít jokera A a prohodit ho s kartou pod ním. Pokud je joker A spodní karta, přesune se pod vrchní kartu. Tedy můžu si balíček představit jako cyklus. Pro ilustraci ukáži příklad na zmenšeném balíčku. Další příklady jsou také prezentovány na zmenšenám balíčku. Na Obrázku 2.2 jsem tento krok zobrazil graficky, abych ujasnil pojmy, které se týkají balíčku. Dále budu používat následující symbolický symbolický zápis.

$$2\ 3\ jokerA\ 4\ 5\ 6 \Rightarrow 2\ 3\ 4\ jokerA\ 5\ 6$$

2.4.2 Nalezení jokera B

Druhý krok je velice podobný prvnímu kroku. Projdu balíček, naleznu jokera B a posunu ho o dvě karty dolů. Pokud byl joker B předposlední kartou,



Obrázek 2.2: Nalezení jokera A

umístím ho pod vrchní kartu, a pokud byl joker B spodní kartou umístím ho pod druhou kartu odshora. Příklad obsahuje první krajní případ.

$$2\ 3\ 4\ 5\ jokerB\ 6 \Rightarrow 2\ jokerB\ 3\ 4\ 5\ 6$$

2.4.3 Trojitý řez

K provedení třetího kroku se musí v balíčku nalézt jak jokera A, tak i jokera B. Tyto dvě karty balíček rozdělí na 3 části. První část je od začátku balíčku do jokera A, druhá je vymezena jokerem A a jokerem B, nakonec třetí je od jokera B do konce balíčku. Ani jeden z jokerů do žádné z těchto tří částí nepatří a je naprosto v pořádku, pokud jedna nebo více částí neobsahuje ani jednu kartu. Trojitý řez spočívá pouze v prohození první a třetí části, přičemž pořadí karet v rámci částí zůstává stejné.

$$\boxed{2\ 3}\ jokerA\ 4\ 5\ jokerB\ \boxed{6\ 7} \Rightarrow \boxed{6\ 7}\ jokerA\ 4\ 5\ jokerB\ \boxed{2\ 3}$$

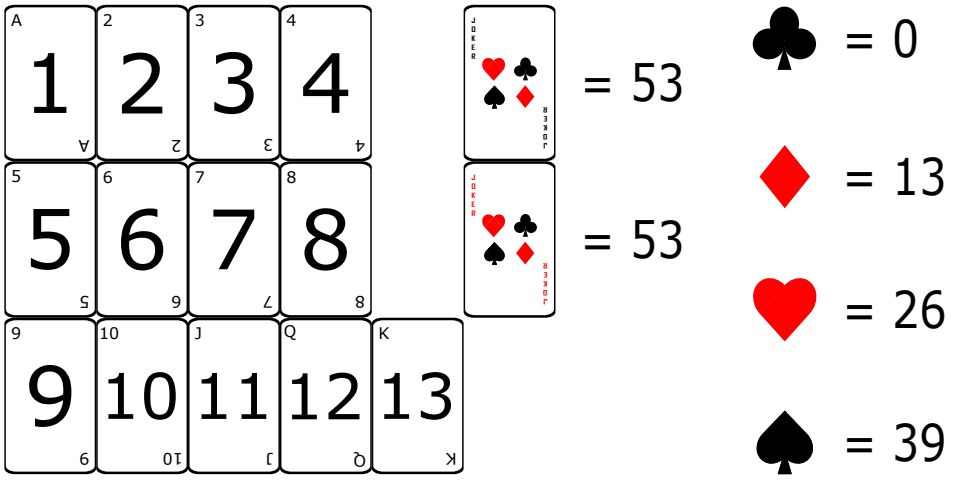
Jak je vidno z příkladu, oba jokeři zůstanou na místě spolu s druhou částí. Pokud je joker A horní kartou a zároveň joker B spodní, anebo obráceně, zůstane balíček po třetím kroku beze změny.

2.4.4 Řez s počítáním

K provedení čtvrtého kroku se podívám na spodní kartu a převedu ji na číslo. Abych svedl toho, ještě nemám dostatečné nástroje. Proto zde udělám odbočku a ukáži, jak převést kartu na číslo.

Balíček má od jedné barvy 13 druhů karet. Tyto karty se převedou na hodnoty zcela přirozeně. Áčko má v Solitaire nejnižší hodnotu, tedy 1, dvojka má hodnotu 2, a tak dále až do krále, který je ohodnocen číslem 13. Nyní se k této hodnotě přičítá váha za barvu. Za trefy se nepřičte nic, za káry 13, za srdce 26 a za piky 39. Tedy dvojka kárová má váhu 2 za hodnotu, plus 13 za barvu tedy v součtu 15. Král srdcový má hodnotu 39 a sedmička trefová pouze 7. Oba jokeři mají hodnotu danou, a to 53. Pro lepší pochopení hodnot karet

přidávám Obrázek 2.3. Dále v tomto textu budu slovo „hodnota“ používat ve významu popsaného převedení karty na číslo a ne ve významu druhu karty.



Obrázek 2.3: Hodnota karet

Nyní již znalosti pro převedení karty na hodnotu mám, tedy mohu snadno spodní kartu převést na hodnotu a tolik karet odpočítat zeshora. Všechny karty pod napočítanou kartou vezmu a ve stejném pořadí je umístím na vrchol balíčku. Pozor, důležité je, aby spodní karta zůstala na místě, a to z toho důvodu, aby bylo možné udělat ke každému kroku krok opačný (inverzní).

$$\boxed{4\ 5\ jokerB}\ 6\ 7\ 8\ 9\ 10\ \boxed{3}$$

Spodní karta má hodnotu 3. Tedy odpočítám 3 karty odshora a provedemu řez. Výsledek je:

$$6\ 7\ 8\ 9\ 10\ \boxed{4\ 5\ jokerB}\ \boxed{3}$$

Jak je vidět z příkladu, spodní karta zůstala na místě. Velké ušetření času je uvědomit si, že když je spodní karta joker, tak se balíček nijak nemění.

2.4.5 Výsledná karta

V pátém kroku se pořadí karet nijak nemění. Podívám se na horní kartu, převedu ji podle již známého postupu na hodnotu a opět přesně tolik karet odpočítám odshora. Kartu pod napočítaným balíčkem převedu na číslo, spočítám modulo 26 a výsledek si zapíši na papír. Toto číslo je součástí proudového klíče a jak jsem psal výše, můžeme ho brát jako posun v rámci anglické abecedy, jenž se pak aplikujeme na písmeno OT. Pokud jako výsledná karta vyšel joker, tak nic nezapisuji a přecházím zpět na krok 1.

Těmito operacemi jsem vždy schopen zašifrovat maximálně 1 znak OT, takže operace opakuji do té doby, než zašifruji celý požadovaný OT. Toto je přesně vlastnost, kterou od proudové šifry požaduji.

Kapitola 3

Klíč vs. heslo

V této kapitole se zabývám problematikou klíče a tím, jakým způsobem z hesla vytvořit klíč. Dále analyzuji, jak dlouhé heslo je vhodné používat. Podle popisu vytvářím implementaci v jazyce C++, jejíž správnost otestuji na testovacích vektorech.

Pro šifrování je potřeba nějakým způsobem balíček inicializovat, tedy zvolit, jak vypadá klíč. Na to jsou k dispozici dva různé způsoby.

3.1 Klíč

První a jednodušší způsob je si pořadí karet co možná najnáhodněji zvolit. Například vytažený balíček z krabičky důkladně zamíchat. Tento způsob se může zdát jednoduchý a přirozený, ale v praxi není dobře použitelný. K tomu, aby dešifrování fungovalo, si musím klíč pamatovat. Pamatování si pořadí 54 karet na dobu, než se bude muset dešifrovat, mi přijde hodně náchylné k chybám a tím nepraktické, protože stačí prohodit pouze dvě karty a proudový klíč je nepoužitelný. Napsání si pořadí karet na papír je asi stejně dobrý nápad, jako si dát napsaný PIN ke své platební kartě do peněženky.

3.1.1 Bezpečnost klíče

Klíč se skládá z libovolného pořadí 54 karet. Je tedy možné klíč napermutovat 54! způsoby.

$$54! \doteq 2,3 \cdot 10^{71}$$

To je ekvivalentní symetrickému klíči délky 237,06 bitů. Nikde se v organizacích zabývajících se standardy nepíše, jak dlouhý klíč by měl mít Solitaire. Ale vzhledem k tomu, že NIST (National Institute of Standards and Technology) akceptuje AES–128, jenž je také symetrickou šifrou, mající klíč dlouhý 128 bitů [9], je délka klíče Solitaire dostatečná.

3.2 Heslo

Schneier navrhuje ještě jinou alternativu, než si pamatovat přímo klíč. Ta spočívá v pamatování si hesla, které se skládá ze slov či vět. Pomocí tohoto hesla se zamíchá balíček s dobře zapamatovatelným původním pořadím, a tím se získá klíč. Já ve své implementaci používám výchozí seřazení podle hodnot $1 \ldots 52$ následováno jokerem A a jokerem B. Toto pořadí, budu nadále ve své práci nazývat *výchozím pořadím*.

Pro vznik klíče z hesla používám kroky 1–4 stejným způsobem, jakým byly popsány v sekci 2.4. Namísto kroku 5 však postupně používám písmena z hesla. Ta se převedou na hodnotu tak, že $A = 1,\ \ldots,\ Z = 26,\ a = 27,\ \ldots,\ z = 52$, a následně provedu ještě jednou obdobu kroku 4. Nedívám se však na spodní kartu a nepoužívám její hodnotu. Využiji příslušnou hodnotu písmene z hesla a provedu „řez s počítáním“. B. Schneier naznačuje[3] ve svém dokumentu použití pouze hesel s velkých písmen. Ale se zdá jako zbytečné omezení možností kroku 4, který je schopný pracovat s čísly od 1 do 52, a oslabení hesla. Proto jsem ve své implementaci dovolil hesla s malými i velkými písmeny.

Výhodou tohoto způsobu je snažší zapamatování si hesla, které může být v podobě věty což si člověk lépe zapamatuje. Nevýhodou je, že pro to abychom mohli šifrovat, musíme dělat hodně časově náročných operací navíc.

3.3 Délka hesla

Délka hesla by měla být z bezpečnostních důvodů minimálně 80 znaků aby se odstranila závislost klíče na výchozím pořadí. Tento závěr jsem učinil po vyhodnocení výsledku testu, který jsem učinil. Zmíněná závislost vadí v tom, že když existuje, tak se lépe odhaduje pořadí, v jakém karty jdou v klíči a tím se síla klíče hodně oslabí.

Klíč považuji za nezávislý na výchozím pořadí, jestliže se znalostí předchozích karet či karty nejsem schopen předpovědět následující kartu s pravděpodobností větší než jakou dává teorie pravděpodobnosti. Tedy aby byl klíč skutečně nezávislý, tak na první kartu, kterou hádám, by měla mít pravděpodobnost 1:54, pro druhou kartu by znalost první neměla nijak jinak pomoci, než že již vytaženou kartu nemá smysl hádat. Pravděpodobnost pro druhou kartu by tedy měla být 1:53. Takto postupuji dále, až pro poslední kartu je pravděpodobnost 1:1. Průměrný počet karet, které se tímto způsobem uhodnu, by podle pravděpodobnosti měl být:

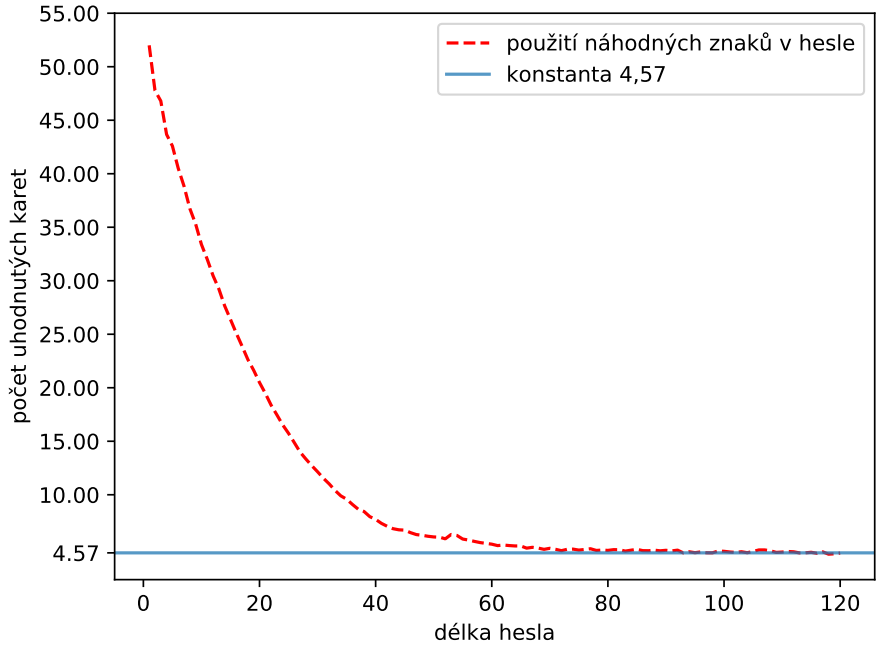
$$\sum_{k=1}^{54} \frac{1}{k} \doteq 4,5457$$

Pro každé heslo vznikl klíč a na tento klíč jsem začal zkoušet, jak je závislý na výchozím pořadí. Pro posuzovávání této závislosti jsem zvolil postup [10],

[3] Doslovně to tam neříká, ale když používá heslo, tak vždy tvořené velkými písmeny.

který hádá karty a sleduje počet uhodnutých karet. Postup je takový, že o první kartě neví nic, proto se konstantně zkouší karta s hodnotou 1. Všechny další karty se hádají vždy s hodnotou o jedna větší než předchozí uhodnutá karta. Pokud tato karta již byla uhodnuta, hledá se první nejbližší vyšší karta. Karta s hodnotou o jedna větší než předchozí karta se hádá z důvodu, že ve výchozím pořadí byly karty hodnotově seřazeny za sebou, takže pokud balíček není dostatečně zamíchán, karty s inktrementovanou hodnotou po sobě následují častěji.

Použil jsem svou implementaci Solitaire a zvolit jsem, že chci balíček inicializovat pomocí hesla. Tedy zvolilo se výchozí pořadí karet a podle hesla se balíček začal míchat, až vznikl klíč. Hesla byla dlouhá 1–120 znaků a znaky byly vygenerovány pseudonáhodně pomocí C++ funkce rand s proměnlivým semínkem. Počet vygenerovaných hesel stejné délky byl nastaven na 1 200. Na všechny vzniklé klíče jsem spustil metodu pro hádání karet. Obrázek 3.1 ukazuje, kolik karet v průměru algoritmus uhodl pro různě dlouhá hesla.



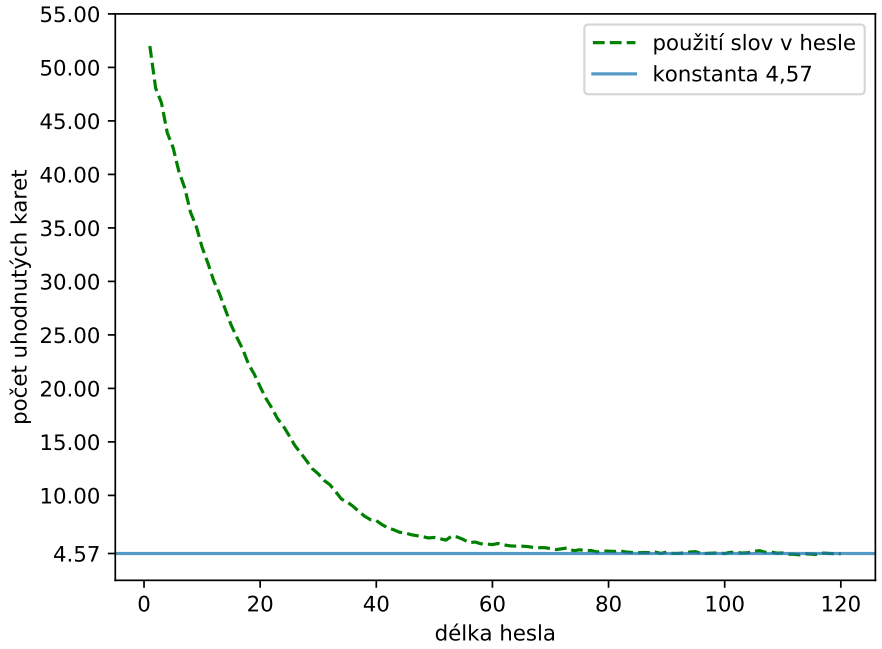
Obrázek 3.1: Náhodné heslo

Jak je vidět, počet uhodnutých karet, tedy závislost klíče na výchozím pořadí karet, velice rychle klesá s délkou hesla. Aby uhodnutí karty bylo dílem náhody, musí počet karet uhodnutých karet oscilovat okolo 4,57. Tento jev nastává od délky klíče 80. Tato hodnota souhlasí i s doporučením, které B. Schneier ke své šifře vydal [4].

V experimentu výše jsem používal heslo vygenerované pseudonáhodně. Jsou vlastnosti hesla horší, když se použijí místo náhodných řetězců slova? Stejný postup popsaný výše jsem aplikoval na klíče tvořené slovy[4]. Opět jsem začal na délce hesla 1 a pokračoval jsem do délky 120. Každé délce jsem vytvořil 1 200 hesel, stejně jako v předchozím experimentu.

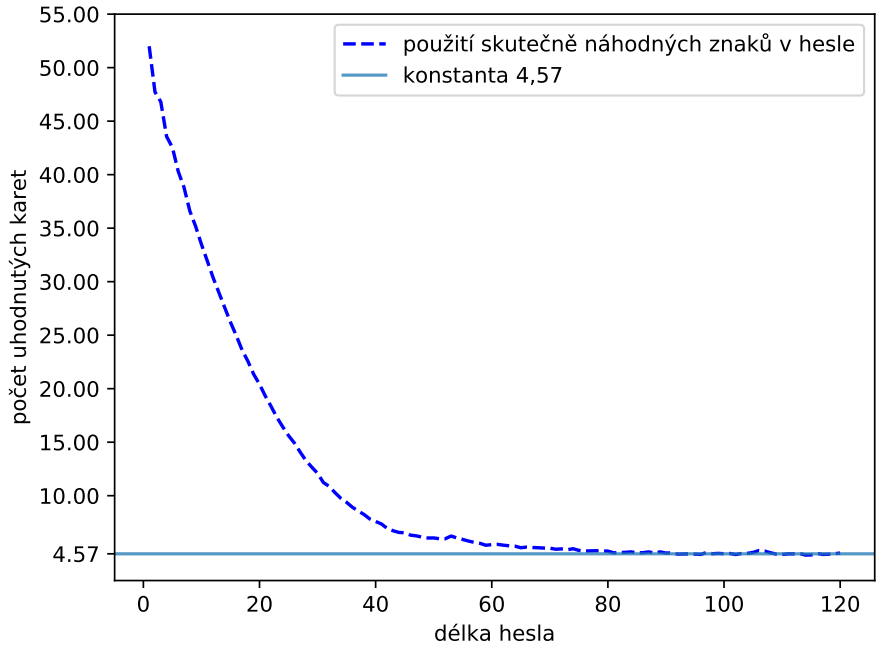
Z Obrázku 3.2 je patrné, že měřené vlastnosti jsou téměř totožné. Nyní udělám poslední část experimentu. Vezmu výstup ze šifry ChaCha20, která má dobré pseudonáhodné vlastnosti (narozdíl od výstupu z funkce rand) a použiji ho k vygenerování hesla. Obrázek 3.3 ukazuje, že i nyní nejsou rodíly téměř žádné. Z toho plyne závěr, že nezáleží na tom, co heslo obsahuje, ale jen jak je dlouhé. Je to z toho důvodu, že čím je heslo delší, tím se provede více míchacích operací, a to rozhoduje o tom, jestli se dostatečně promíchá balíček. Z jiného hlediska, je lepší používat náhodná hesla. Předejde se tím útokům za pomocí slovníkových hesel či odhadování hesla podle chrakteru či zálib osoby. Na druhou stranu, proč si pamatovat náhodné heslo, když už si můžu pamatovat náhodný klíč a ušetřím tím mnoho operací s balíčkem.

Pro porovnání jsem proložil v Obrázku 3.4 grafy do jednoho. Z grafu je dobře patrné, že rozdíly jsou minimální.

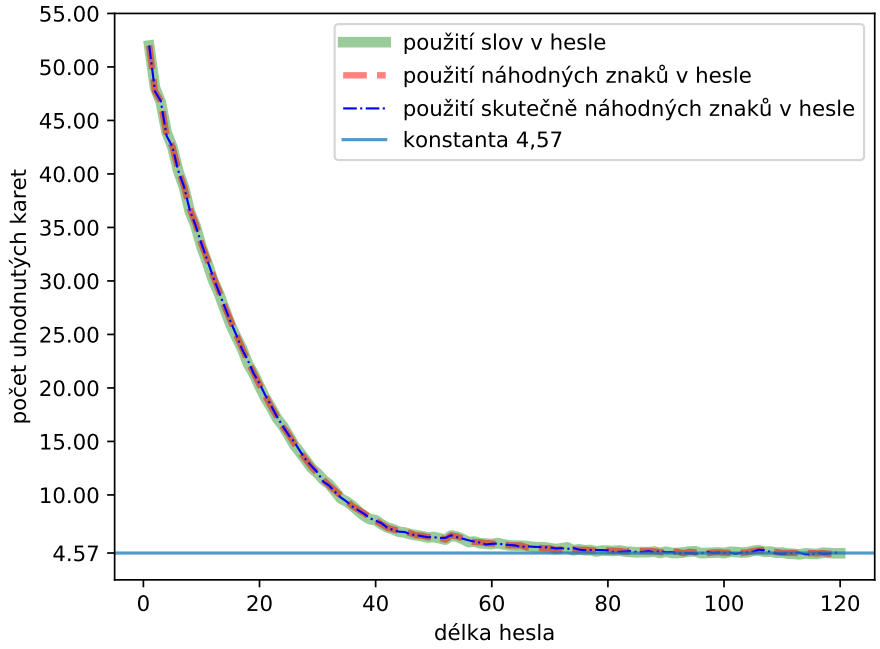


Obrázek 3.2: Heslo tvořené slovy

[4]Slova jsou použitá z `https://github.com/dwyl/english-words/blob/master/words.txt`



Obrázek 3.3: Skutečně náhodné heslo



Obrázek 3.4: Heslo tvořené slovy a náhodnými znaky

3.4 Implementace

Šifru jsem implementoval v jazyce C++, přičemž balíček je reprezentován jednosměrným spojovým seznamem a držím si ukazatel na konec a začátek balíčku. Spojový seznam namísto pole jsem zvolil z toho důvodu, že když se provádí operace s částí balíčku, tak stačí přepojit ukazatele a nemusí se složitě kopírovat prvky. Implementace umí zpracovávat jak klíč, který se programu předá v podobě textového souboru, tak i heslo, ze kterého klíč vznikne. Je možné šifrování i dešifrování.

Vysloveně testovací vektory k šifře neexistují, ale u popisu [4] je několik příkladů s očekávanými výsledky. Pro tyto příklady provedu zašifrování a porovnám výstup s očekávaným výsledkem. OT který má následující podobu:

$$AAAAA\ AAAAA\ AAAAA$$

Se zašifruje za pomocí klíče FOO na:

$$ITHZU\ JIWGR\ FARMW$$

Což je správný výsledek. Stejně úspěšně dopadne i další příklad uvedený v textu [4].

3.4.1 Jak pracovat s implementací

V přiloženém datovém nosiči je složka */Implementation*, která v sobě obsahuje zmiňovanou implemantaci. Ve složce je přiložen Makefile pro kompilaci (make) i pro spuštění (make run). Po spuštění si vyberu možnost, jak inicializovat balíček. Tento krok je stejný jak pro šifování, tak pro dešifrování. Inicializuji buď pomocí klíče (na vstup se zadá číslo 1), anebo pomocí hesla (na vstup se zadá 2).

Když vyberu první možnost, musím poté zadat název textového souboru, kde je uložen klíč. Pro lepší představu zde uvedu příklad obsahu takového souboru. Tento klíč je ekvivalentní s klíčem, který vznikne po zadání hesla FOO.

```
9D TD JD QD KD AH 2H 3H 4H 5H 6H 7H 8H 9H TH JH QH KH AS 2S 3S
4S 5S 6S 7S 8S 9S TS JS QS 3C 4C 5C 6C 7C AC TC JC QC KS RR 8C
9C RB KC AD 2D 3D 4D 5D 6D 7D 8D 2C
```

Jak je vidět, první část karty je její hodnota. Míto hodnoty 10 používám znak T, kluk je J, dáma Q, král K a eso A. Poté následuje barva karty. Pro trefy je použit znak C, káry D, srdce H a pro piky S. Jokeři jsou označení jako RR pro jokera A a RB pro jokera B. Mezery mezi karatmi nejsou povinné, ale zvyšují čitelnost.

Když vyberu druhou možnost, tedy inicializace pomocí hesla, je potřeba do vstupu zadat heslo. Heslo může být tvořeno malými a velkými písmeny anligcké abecedy.

Nyní je potřeba, abych si vybral, jestli chci šifrovat (na vstup se zadá 1) nebo dešifrovat (na vstup se zadá 2). Poté stačí, když napíši jméno souboru, který chci šifrovat respektive dešifrovat a výstup je soubor *CT.txt* respektive *OT_Decrypt.txt*, který obsahuje zašifrovaný vstupní soubor a nebo v případě dešifrování dešifrovaný vstupní soubor.

Kapitola 4

Solitaire vs. ChaCha20

V práci porovnávám ChaCha20 jako představitele moderních proudových šifer s ruční šifrou Solitaire z důvodu zajímavých kontrastů dvou zcela odlišných přístupů. A to šifry vykonavatelné strojem a šifry určené k lidskému užití.

4.1 Návrh šifry

ChaCha20 je proudová šifra vyvinutou Danielem J. Bernsteinem. Byla popsána v roce 2008. Může být dokonce považována za alternativu k AES, a to hlavně z důvodu, že ChaCha20 je v softwarové implementaci rychlejší. Pochopitelně toto neplatí, pokud procesor má přímo sadu instrukcí AES (AES-NI) [11]. Šifru ChaCha20 začala používat firma Google v protokolu TLS (Transport Layer Security) a SSL (Secure Sockets Layer) v roce 2014 pro symetrické šifrování z důvodu, že mobilní telefony často nemají AES-NI. ChaCha20 je tedy rychlejší, jak napsal Google na svém blogu[5]. Šifra je založena pouze na AXR operacích, tedy sčítání (addition), xor, rotace (rotation). Rotace je dále značena <<<.

Úvodním stavem ChaCha20 je vektor nebo spíše matice 4x4. Každý prvek (slovo) této matice má velikost 32 bitů, tedy dohromady 512 bitů.

$$\begin{matrix} c_0 & c_1 & c_2 & c_3 \\ k_4 & k_5 & k_6 & k_7 \\ k_8 & k_9 & k_{10} & k_{11} \\ b_{12} & b_{13} & n_{14} & n_{15} \end{matrix}$$

c – konstanty

k – klíč

b – blokový čítač

[5] https://security.googleblog.com/2014/04/speeding-up-and-strengthening-https.html

n – nonce

Verze ChaCha20, která je dnes používána v krypto knihovnách jako je například LibreSSL [12], je oproti originálnímu popisu [11] změněna v posledním řádku. Podle tohoto používaného popisu nadále postupuji i v této práci.

$$\begin{matrix} c_0 & c_1 & c_2 & c_3 \\ k_4 & k_5 & k_6 & k_7 \\ k_8 & k_9 & k_{10} & k_{11} \\ b_{12} & n_{13} & n_{14} & n_{15} \end{matrix}$$

Používá se tedy pouze 32bitový blokový čítač namísto 64bitového. Konstanty mají následující hodnoty:

$$c_0 = 0x61707865$$

$$c_1 = 0x3320646e$$

$$c_2 = 0x79622d32$$

$$c_3 = 0x6b206574$$

Nonce se využívá k tomu, aby se jeden klíč dal použít i vícekrát než jen jednou. V kapitole 5 vysvětlím, proč je znovu stejného klíče problém.

4.2 Rundy

Šifra ChaCha20 se skládá z 20 rund (z toho plyne číslovka v jejím názvu), z nichž je 10 sloupcových a 10 diagonálních. Každá z těchto rund je poskládána ze 4 čtvrtrund. Podoba rundy je následující.

Sloupcová runda

$$quaterRound(c_0,\ k_4,\ k_8,\ b_{12})$$

$$quaterRound(c_1,\ k_5,\ k_9,\ n_{13})$$

$$quaterRound(c_2,\ k_6,\ k_{10},\ n_{14})$$

$$quaterRound(c_3,\ k_7,\ k_{11},\ n_{15})$$

Diagonální runda

$$quaterRound(c_0,\ k_5,\ k_{10},\ n_{15})$$

$$quaterRound(c_1,\ k_6,\ k_{11},\ b_{12})$$

$$quaterRound(c_2,\ k_7,\ k_8,\ n_{13})$$

$$quaterRound(c_3,\ k_4,\ k_9,\ n_{14})$$

Čtvtrunda se skládá z jichž zmíněných operací AXR a dala by se v C-notaci napsat takto:

```
a += b; d ^= a; d <<<= 16;
c += d; b ^= c; b <<<= 12;
a += b; d ^= a; d <<<= 8;
c += d; b ^= c; b <<<= 7;
```

Kde a, b, c, d, jsou parametry, které čtvrtrundě předáme.

4.3 Šifrování

Počáteční stav vznikne vyplněním matice konstantami, zadaným klíčem s nonce a nakonec nastavením čítače na 0. Jak je zřejmé, klíč je dlouhý 256 bitů, konstanty 128 bitů, nonce 96 bitů a čítač 32 bitů. Na úvodní stav aplikuji 20 rund a finální výsledek klasickým sčítáním v modulo 2^{32} přičtu k původnímu stavu. Po přičtení vznikne 512 bitů dlouhý pseudonáhodný proudový klíč. Ten se operací xor aplikuje na data, která je potřeba zašifrovat. Následně se blokový čítač inkrementuje a vše začíná nanovo, dokud se nezašifrují celá data.

Čítač je velký jen 32 bitů, z toho vyplývá, že se nemůže inkrementovat do nekonečna. Maximální délka proudového klíče může být 256 GiB, což je dostačující, pokud se jedná o šifrování zpráv. ChaCha20 díky čítači silně připomíná blokové šifry v módu CTR (Counter mode).

4.4 Srovnání Solitaire s ChaCha20

ChaCha20 a Solitaire jsou každá vytvořená za jiným účelem. Proto se každá hodí v jiných situacích. Mají obě komunikující strany mají k dispozici počítač a bezpečný kanál pro komunikaci, ve všech směrech z porovnání vyjde lépe ChaCha20. Na druhou stranu, pokud počítač k dispozici není, tak operace, které počítač vykoná v okamžiku, jsou v ruce zdlouhavé a náchylné k chybám. V tomto případě je vhodnější šifra Solitaire. V Tabulkce 4.1 je srovnání vlastností a parametrů, které maji šifry ChaCha20 a Solitaire.

Tabulka 4.1: Porovnání Solitaire s ChaCha20

Vlastnost	**ChaCha20**	**Solitare**
délka klíče	256bitů	237bitů
nonce	ano	ne
rychlost provádění v ruce	pomalejší	rychlejší
rychlost provádení na PC	rychlejší	pomalejší
výstup	proud bitů	proud čísel 1–52
počet rund pro výstup	20	minimálně 5

Kapitola 5

Znovupoužití stejného klíče

V této kapitole se zaměřím na to, proč nepoužívat stejný klíč dvakrát. Problém demostruji na šifrování dvou obrázků stejným heslem. Tímto způsobem prezentace nebezpečí jsem se inspiroval na cvičení v předmětu BI-BEZ, který jsem absolvoval na Fakultě informační technologií v Praze.

5.1 Princip

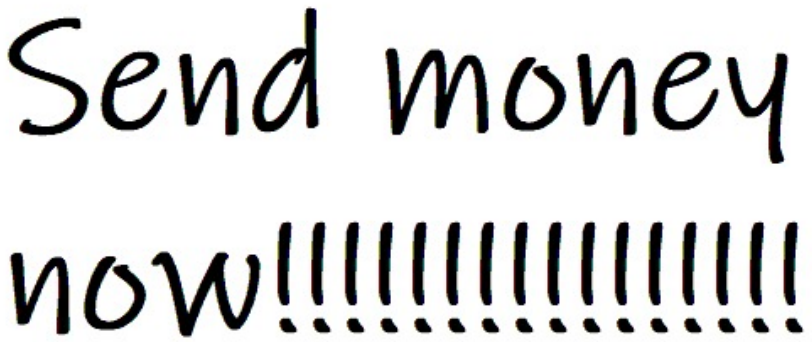
Každý uživatel Solitaire si musí dát pozor na to, aby jeden klíč, respektive jedno heslo nepoužil k šifrování dvakrát. Toto není slabina Solitaire je to vlastnost, kterou mají všechny šifry, které používají podobný princip zvaný jednorázová tabulková šifra (anglicky one-time pad). Problém je, že pro každý stejný klíč se vygeneruje stejný proudový klíč K. V případě dvou ŠT (ST_1 a ST_2) zašifrovaných stejným klíčem mohu ŠT od sebe pouze odečíst.

$$ST_1 - ST_2 = (A + K) - (B + K) = A–B + K–K = A–B$$

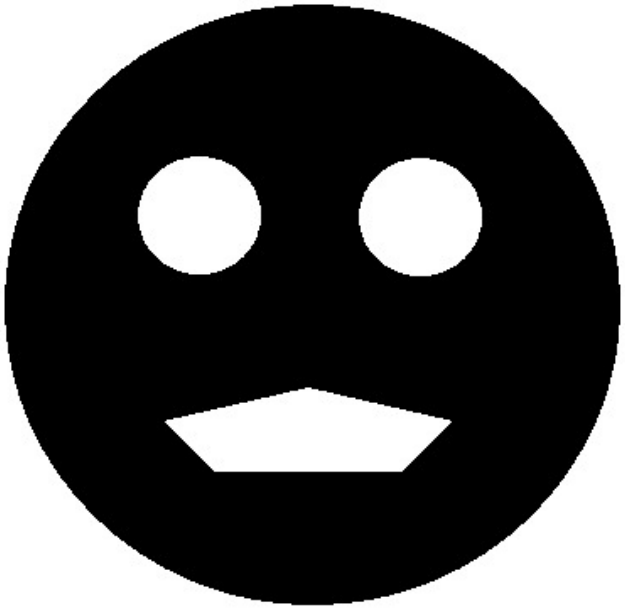
Jak je vidět, po odečtení se úplně zbavím klíče a zbyde rozdíl dvou OT. Toto mohu již snadno prolomit např. posuvným slovníkem.

5.2 Odstranění klíče

Toto nebezpečí je velmi dobře vidět na následujícím příkladu. Mám dva bitmapové obrázky A (Obrázek 5.1) a B (Obrázek 5.2). Tyto dva obrázky jsem pomocí Solitaire zašifroval. Abych tak mohl učinit, musel jsem vyřešit zásadní problém, a to, že šifrovací algoritmus negeneruje pseudonáhodný proud bitů, ale čísel od 1–52. Číslo 52 není mocnina dvou, takže kdybych to chtěl jednoduše převést do dvojkové soustavy a to využít, narazil bych na problém, že by počet nul silně převyšoval počet jedniček. Toto není vlastnost, kterou u pseudonáhodných čísel vítám.



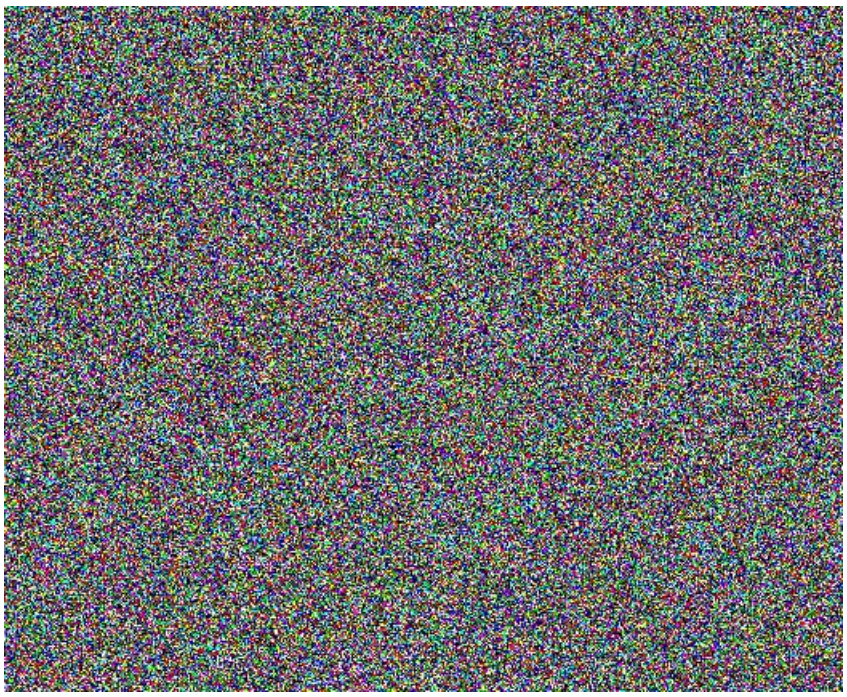
Obrázek 5.1: Obrázek A



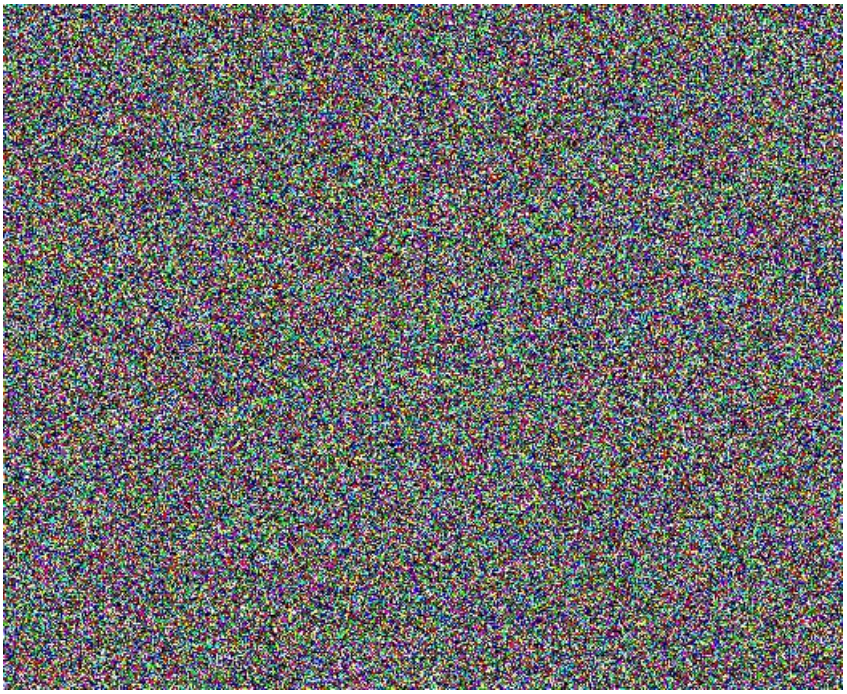
Obrázek 5.2: Obrázek B

K překonání této překážky jsem použil následující řešení. Vygenerovaný proudový klíč chápu jako jedno dlouhé číslo vyjádřené ve dvaapadesátkové soustavě. Tedy po každém úspěšném provedení kroků 1–5 výslednou číslici o jedna zmenším (aby se obsáhla i 0) a beru ji jako cifru ve dvaapadesátkové soustavě. Poté, abych proudový klíč mohl přičíst k 1 bajtovým pixelům a zašifroval je, převedu celý výsledný proudový klíč do hexadecimální soustavy. Výsledek beru po bajtech a přičítat je v modulo 256 k pixelům v obrázku.

Zašifrované obrázky A a B na první pohled vypadají nečitelně, jak uzkazují Obrázek 5.3 a Obrázek 5.4. Po vzájemném odečtení se projeví chyba použití stejného klíče. Jak je vidět na Obrázku 5.5, obrázky se vpily do sebe a rozhodně neukrývají žádnou informaci.



Obrázek 5.3: Zašifrovaný obrázek A pomocí Solitaire



Obrázek 5.4: Zašifrovaný obrázek B pomocí Solitaire

Obrázek 5.5: Odstranění klíče ze zašifrovaného obrázku A a B pomocí Solitaire

5.3 ChaCha20

Abych ukázal, že se skutečně nejedná jen o slabinu šifry Solitaire, tak ten samý příklad ukáži i pro ChaCha20. Vzheldem k tomu, že ChaCha20 používá nonce, tak jsem použil jak stejný klíč, tak stejný nonce. Opět jsem zašifroval obrázky A a B. Výsledek zobrazují Obrázek 5.6 a Obrázek 5.7. Obrázky se zašifrovaly do nečitelné podoby. Jak bylo řečeno v kapitole 4, ChaCha20 používá operaci xor ke spojení proudového klíče s OT. Tedy když mám dva ŠT (ST_1 a ST_2) a pro jejichž zašifování byl opužit stejný klíče K, tak pro zbavení klíče použiji následující postup:

$$ST_1 \oplus ST_2 = (A \oplus K) \oplus (B \oplus K) = A \oplus B \oplus K \oplus K = A \oplus B$$

V případe ChaCha20 není výsledek rozdíl dvou OT, ale jejich xor. Obrázek 5.8 ukazuje, jak situace vypadá, když použiji operaci xor na Obrázek 5.6 a Obrázek 5.7. Stejně jako v případě, když jsem šifroval pomocí Solitaire, se obrázky do sebe vpily.

Obrázek 5.6: Zašifrovaný obrázek A pomocí ChaCha20

Obrázek 5.7: Zašifrovaný obrázek B pomocí ChaCha20

Obrázek 5.8: Odstranění klíče ze zašifrovaného obrázku A a B pomocí ChaCha20

Kapitola 6

Nenáhodnost v Solitaire

V této kapitole ověřuji, zdali pozorování v práci P. Crowleyho [7] odpovídá mnou naměřeným hodnotám. Také navrhnu řešení z tohoto měření vycházejícího problému.

Ukazuje se, že výstup Solitaire není tak náhodný, jak by teoreticky být měl. Ve své práci [7] na to Crowley poukazuje a přidává důvod, proč tomu tak je.

Jak jsem již napsal v popisu šifry, po každém úspěšném provedení všech kroků získám číslo od 1-52, to by mělo být náhodné. Když je situace, že jsou dva po sobě úspěšné cykly (úspěšným cyklem se myslí provedení kroku 1-5, přičemž výstupní karta není joker, tedy kroky se nemusí opakovat). Pokud je v obou cyklech před 5. krokem stejná horní karta, tak pravděpodobnost i stejné výstupní karty je vyšší, než by měla. Teoreticky by tento stav měl nastávat v 1,92 % případů (neboli 1:52), ale Crowley říká, že tento stav nastane přibližně v 34 % případů.

6.1 Testování

Opět použiji svou implementaci a zjistím, jestli mohu potvrdit tuto nenáhodnost. Nejprve měřím jak často nastává stav, kdy po dvou úspěšných cyklech je před krokem 5 stejná horní karta.

Za účelem testování v této kapitole jsem vygeneroval 50 hesel délky 100. Pro každé toto heslo jsem provedl 5 000 000 cyklů a počítal jsem, kolik za sebou úspěšných cyklů má stejnou horní kartu před krokem 5. Podle výsledků testu nastává tento stav v 1,815 % případů. Výsledek tedy je přibližně odpovídající teoretické pravděpodobnosti, která je rovna 1,85 % případů. Každá karta by se měla na každé pozici vyskytovat se stejnou pravděpodobností.

Předpokládám, dvě stejné horní karty a určím pravděpodobnost, že se rovnají i výstupní karty. Teoretická pravděpodobnost shody dvou výstupních karet je 1:52 neboli jev nastane opět v 1,92 % případů. Naměřené hodnoty testu nabývají však hodnot mnohem vyšších, a to 34,99 %. To je mnohem hod-

nota, než by měla teoreticky být. Tedy zde se objevuje nenáhodnost v proudovém klíči. Důvod této nenáhodnosti přisuzuji nedostatečnému zamíchání karet během provedení kroků 1-5. A když nastane situace, že je po zamíchání horní karta stejná jako před zamícháním, je i dosti pravděpodobné, že stejné karty jsou i výstupní, protože výstupní karta se odvíjí právě od karty horní. Po dvou úspěšných cyklech je pravděpodobnost stejné horní karty 0.01815 a když toto nastane, tak pravděpodobnost i stejné výstupní karty je 0.3499. Když tato dvě čísla vynásobím, vyjde pravděpodonost, že výstupní karta byla ovlivněna tímto jevem. Výsledek je 0.0063, neboli v 0.63 % případů.

6.2 Dopad

Závažnost této nenáhodnosti je diskutabilní. Pokud by bylo známo, které znaky toto ovlivňuje, tak by šlo od sebe sousedící znaky odečíst a tím se zbavit klíče, ale i tak je stále k dispozici pouze rozdíl těchto znaků. A vzhledem k tomu, že závislost ovlivňuje jen 0,63 % dvojic a pozice těchto dvojic je neznámá a nepredikovatelná, tak frekvenční analýza není účinná.

Je však pravda, že k získání informací nepotřebuji prolomit celou zprávu. V určitých případech by mohl být klíčový jen jediný znak. Stále však není známo, které dva znaky jsou ovlivněny. Z tohoto důvodu si nedovedu představit, jakým způsobem by bylo možné této závislosti vyžít. Psal jsem kvůli tomuto problému B. Schneierovi, jestli vidí zneužitelnost této slabiny jako reálnou. V odpovědi jsem dostal, že neví, ale možné to je. A skutečně, i když zatím způsob není znám, tak to neznamená, že nebude v budoucnosti objeven a tím budou veškeré zprávy zašifrované šifrou Solitaire kompromitované.

6.3 Řešení

K řešení jsem zvolil přidání dalších operací po kroku 3, kdy ještě jednou opakuji kroky 1, 2, 3. Zlepšení je skutečně markantní. Nyní, když mám po dvou po sobě jdoucích cyklech stejné horní karty, tak pravděpodobnost, že se výstupní karty rovnají, klesla z 34.99 % k uspokojivým 2,91 %. Sice to pořád není teoretických 1,92 %, ale zlepšení je veliké. Toto nepovažuji za všespásné řešení, protože každý míchací krok vykonaný v ruce je časově drahý. Přidáním 3 kroků se šifrování i dešifrování prodlužuje. Také je možné, že se závislost po této změně přesunula jinam, a to, že jsem jí neobjevil, ještě neznamená, že tam není.

Kapitola 7

Pozice jokerů v balíčku

V této kapitole se zaměřuji na pozice jokerů během míchacích kroků a z popisu dokáži, proč není možné, aby se joker A nacházel naspodu balíčku. Joker B se sice tam objevit může, ale jen s malou pravděpodobností. Dále vyřeším, jaký to má dopad na bezpečnost šifry Solitaire.

7.1 Joker A

Princip spočívá v tom, že každý si může zvolit zcela libovolný klíč. Ale když se provede hned první cyklus, zjišťuji, že joker A se již nikdy nevyskytne na konci balíčku přímo po provedení alespoň 1 úspěšného cyklu. Důvod je v popisu šifry, jak ukáži v důkazu sporem.

Důkaz. Mám balíček karet s jokerem A naspodu balíčku po provedení alespoň jednoho celého cyklu. Balíček má následující podobu:

$$K_n,\ jokerB,\ K_m,\ jokerA$$

kde K_n a K_m reprezentují libovolný, i nulový, počet navzájem různých karet bez jokeru A a B z jednoho balíčku. Z popisu [4] B. Schneiera vím, šifra je inverzní, tedy každý krok je vratný. Provedu inverzní kroky 5-1, abych zjistil, z jakého pořadí karet může vzniknout balíček s jokerem A na konci. Krok 5 pořadí karet nemění, tedy balíček nechám tak jak je. Krok 4 se invertuje podle poslední karty v balíčku, která po provedení kroku 4 zůstane na místě, ale vzhledem k tomu, že spodní karta je joker A, tak se pořadí opět nemění. Tedy provedu inverzi ke kroku 3, ta spočívá v nalezení jokera B a jokera A a provedení „Trojitého řezu“. Výsledný balíček má následující podobu:

$$jokerB,\ K_m,\ jokerA,\ K_n$$

Teď je na řadě inverze kroku 2. A zde nastává spor. Není možné udělat inverzi kroku 2, pokud na vrcholu balíčku je joker B. Není to možné proto,

že když provedu krok 2 a joker B je poslední karta v balíčku, joker B se vkládá pod 2. kartu odshora. V případě, kdy je joker B předposlední karta, vkládá se pod horní kartu. A nakonec, pokud je joker B třetí karta odspodu, tak se přesune na konec balíčku. Tedy skutečně není možné, aby byl joker B na vrcholu balíčku po provedení kroku 2. Z toho všeho vyplývá, že joker A nemůže se nikdy od určení klíče objevit naspodu balíčku přímo před krokem 1.

□

7.2 Joker B

S pozicí jokera B to není tak jednoznačné, ale přesto je zde problém. Joker B se skutečně může na konci balíčku před krokem 1 objevit. Za jakých podmínek se tam objeví, to popisuje následující důkaz.

Důkaz. Mám balíček karet s jokerem B naspodu balíčku po provedení alespoň jednoho celého cyklu. Balíček má následující podobu:

$$K_n,\ jokerA,\ K_m,\ jokerB$$

kde K_n a K_m reprezentují libovolný, i nulový, počet navzájem různých karet bez jokeru A a B. Opět provedu inverzní kroky, abych zjistil, jak se joker B dostal naspod balíčku. Krok 5 pořadí nemění, tedy inverze také nic nezmění. Vzhledem k tomu, že krok 4 z důvodu jokera B na konci balíčku nic nezměnil, inverze také neprovede žádnou změnu. Po provedení inverzního kroku 3 bude mít balíček následující podobu:

$$jokerA,\ K_m,\ jokerB,\ K_n$$

K provedení opačného kroku ke kroku 2 potřebuji odpočítat 2 karty od pozice jokera B a přemístit jokera B za druhou odpočítanou kartu. A zde se důkaz dělí na tři větve, které musím samostatně vyřešit:

1. K_m reprezentuje více než jednu kartu, tedy joker A se neposune po inverzi před jokera B. Poté má výsledný balíček následující podobu:

   $$jokerA,\ K_{m-2},\ jokerB,\ K_{n+2}$$

   Docházím k závěru, že po provedení kroku 1 není možné, aby byl joker A na vrcholu balíčku, to plyne z popisu šifry, který říká, že pokud je joker A spodní karta, přesune se pod horní kartu a pokud je předposlední kartou, přesune se pod spodní kartu, tedy nikdy ne na vrchol balíčku. To je spor s předpokladem, že takový balíček může nastat.

2. K_m nereprezentuje žádnou kartu, tedy joker A sousedí s jokerem B. Po provedení inverzního kroku 2 balíček vypadá následovně:

$$jokerA,\ K_{n-1},\ jokerB,\ K_j$$

K_j reprezentuje jednu kartu, která předtím byla na posledním místě v K_n. A opět zde nastává spor s předpokladem, poněvadž joker A nemůže být po kroku 1 na vrcholu balíčku.

3. K_m reprezentuje právě jednu kartu, tady joker A je ob jednu kartu od jokera B. K provedení inverze však stojí velká překážka. Není definováno, kde se joker B nacházel předtím. Jsou totiž dva způsoby, jak dostat jokera B na druhou kartu odshora. Když je joker B poslední karta, tak se po kroku 2 přesune pod druhou kartu odshora. To samé platí i pro případ, kdy je první kartou v balíčku. Důkaz se zde znovu rozděluje na dva další podpřípady.

   a) Joker B se před vykonáním kroku 2 nacházel na konci balíčku. Tvar balíčku byl tedy v tomto případě následující:

   $$jokerA,\ K_m,\ K_n,\ jokerB$$

   A opět jsem v situaci, kdy nacházíme spor, není možné, aby po kroku 1 byl joker A na vrcholu balíčku.

   b) Joker B se nacházel na vrcholu balíčku. Balíček má následující tvar:

   $$jokerB,\ jokerA,\ K_m,\ K_n$$

   A nyní konečně je možné udělat inverzi ke kroku 1. Po provedení získáme jedinou možnou pozici jokerů, ze které můžeme po provedení cyklu dostat jokera B na konec balíčku.

□

Jak je vidět, na to, aby se mohl joker B dostal na konec balíčku, musí být oba jokeři na konkrétních místech. Tedy teoretická pravděpodobnost je:

$$\frac{1}{54} \cdot \frac{1}{53} = 0,00034$$

Tedy mnohem nižší než pro jakoukoliv jinou kartu. Toto číslo přibližně odpovídá hodnotě 0.00037, která mi vyšla při praktickém měření.

7.3 Závěr

To, že Solitaire není inverzní, jak tvrdí B. Schneier, již ve své práci zmínil P Crowley [7], ale nijak dále to nerozvádí. Problém je to proto, že se dva stavy slučují do jednoho. To znamená, že ze dvou různých klíčů může v průběhu šifrování vznikat stejný stav s větší pravděpodobností, než by odpovídalo plně náhodnému postupu.

Po úspěšném provedení prvního cyklu se joker A na konci balíčku neobjeví nikdy a joker B s mnohem menší pravděpodobností než ostatní karty. Útok by se tedy dal vést tak, že by se neútočilo na klíč samotný, ale na stav, který balíček nabývá po provedení prvního cyklu. Tam již balíček nemůže obsahovat 54! možností, ale jen $54! - 2 \cdot 53!$, což je jen nepatrné snížení, z 237,06 bitů na 237,009 bitů.

Závěr je takový, že i když se jedná o zajímavou vlastnost šifry a pravděpodobně jde o chybný návrh, v praktickém použití to šifru nijak neoslabí.

Kapitola 8

NIST testy

Výstupem Solitaire je proud pseudonáhodných čísel od 1-52, na čemž je postavena bezpečnost šifry. V této kapitole použiji baterii „Statistických testů pro náhodná a pseudonáhodná čísla“ od NIST (National Institute of Standards and Technology) pro ověření náhodnosti proudu dat. Jedná se o balíček 15 opensource testů které mají na vstupu binární soubor a na výstupu prohlásí, jestli jsou binární data v souboru náhodná a nebo ne. Popis testů je dostupný v [13].

8.1 Princip testů

Testy zkoumají jak náhodnost, tedy že pravděpodobnost bitu 0 je stejná, jako pravděpodobnost bitu 1, tak nepředvídatelnost, tedy, že na základě přechozích sekvencí nelze předpovědět další bit s pravděpodobností větší než 50 %.

Testy nepracují s celým souborem najednou, ale rozdělí ho na sekvence podle potřeb daného testu. Testuje se takzvaná nulová hypotéza H_0, která říká, že bitová sekvence je náhodná. Pokud se H_0 nepotvrdí, musíme přistoupit k hypotéze alternativní H_a, která má význam přesně opačný než H_0.

Během testů se spočítá pravděpodobnostní hodnota (P-hodnota) sekvence, která se testuje. Ta se porovnává s kritickou hodnotou. Pokud je hodnota menší než kritická hodnota, potvrdí se nulová hypotéza, v opačném případě se přijme H_a. Kritická hodnota se vypočítává z teoretického rozdělení pravděpodobnostních hodnost normálního rozdělení nebo chí-kvadrátu. Je nastavena tak, aby určité procento (v NIST testech je nastaveno 99 %) pravděpodobnostních hodnot daného rozdělení bylo menší než kritická hodnota. Pokud je testovaná sekvence skutečně náhodná, pravděpodobnost, že spočítaná hodnota je menší než kritická hodnota, je právě těch 99 %. Jak je tedy zřejmé, mohou nastávat chyby v hypotézách, neboli přijme se špatná hypotéza vůči datům. Jak ukazuje Tabulka 8.1, chyby se dělí na chyby prvního a druhého druhu. [13]

Doplněk ze zmíněného procenta, podle kterého se nastavuje kritická hodnota, říká, jaká je pravděpodobnost, že se objeví chyba prvního druhu. Označuje

Tabulka 8.1: Chyby prvního a druhého druhu

Data	**Vyhodnocení**	
	Přijetí H_0	**Přijetí H_a**
Náhodná data	Správně	Chyba prvního druhu
Nenáhodná data	Chyba druhého druhu	Správně

se α a v NIST testech je nastaven na hodnotu 0,01.

Pravděpodobnost, s jakou se objeví chyba druhého druhu, se označuje β. NIST testy se snaží minimalizovat pravděpodobnost chyb druhého druhu, k tomu je potřeba dostatečné množství testovacích dat a velká pestrost testů. Pro každý test je doporučena minimální délka sekvence a počet sekvencí, aby se dal test považovat za věrohodný.

Pravděpodobnostní hodnota, která se nad sekvencí vypočítá, se nazývá P-hodnota. Ta říká, s jakou pravděpodobností se data vygenerovaná teoreticky dokonalý generátor náhodných čísel by se jevila v testu jako méně náhodná, než testovaná data. Tedy pokud je P-hodnota rovna 1, data se jeví v testu jako dokonale náhodná, na druhou stranu, pokud je P-hodnota rovna 0, jedná se o naprosto nenáhodná data.

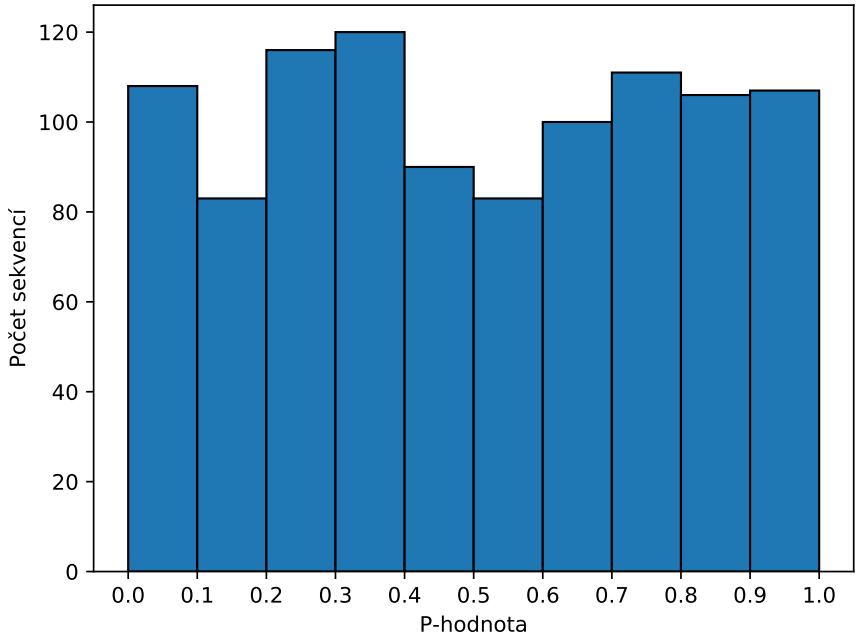
P-hodnoty se po vypočtení pro každou sekvenci zařadí do jednoho z intervalů podle hodnoty. Intervalů je v testech 10 a do každého by mělo spadat přesně 10 % všech P-hodnot daného rozdělení. Těchto 10 intervalů by mělo splňovat rovnoměrné rozdělení. Odchylka se spočítá tak, že se pro intervaly, do kterých se roztřídily P-hodnoty všech sekvencí, vypočte celková P-hodnota, ta se porovná s kritickou hodnotou a podle tohoto se přijme nebo zamítne celková H_0, jestli jsou data jako celek náhodná. Obrázek 8.1 ukazuje, jak vypadá rozdělení P-hodnot pro data ze šifry Solitaire pro Frekvenční monobitový test. Počet sekvencí byl 1 024. Toto ovšem není jediné rozhodovací kritérium, jestli vstupní data zamítnout. Další je, kolik sekvecí přijalo H_0 a kolik jich ji odmítlo.

8.2 Popis NIST testů

V této části popisuji jednotlivé testy pro náhodnost. Účelem nemá být zjištění, jak testy fungují, ale jak správně zadat parametry testu a korektně interpretovat výsledek. Pro zjištění, jak testy přesně fungují, si doporučuji přečíst jejich podrobný popis [13]. V popisu testu vždy uvedu v seznamu parametry testu a za čárkou minimální doporučené hodnoty.

8.2.1 Frekvenční monobitový test

Cílem testu je zjistit, jestli bity 0 a 1 mají přibližně stejnou pravděpodobnost. Pokud výsledné rozdělení P-hodnot má výrazně více hodnot blíže k 0 než k 1,



Obrázek 8.1: Rozdělení P-hodnot pro Frekvenční monobitový test

znamená to, že převládá v sekvenci výrazně počet bitů jedné hodnoty. Pokud naopak P-hodnoty převažují k hodnotě 1, znamená to až příliš stejný poměr mezi bity 1 a 0.

Parametry

- n – délka sekvence, $n >= 100$

8.2.2 Frekvenční test v blocích

Test se provádí za účelem zjištění, zda sedí poměr mezi bity 1 a 0, když se sekvence rozdělí do bloků. Pokud test nad sekvencí odmítne nulovou hypotézu, znamená to, že v blocích nejsou bitové hodnoty v rovnováze.

Parametry

- n – délka sekvence, $n >= 100$
- N – počet bloků, $N < 100$
- M – délka jednoho bloku, $M > 0,01 \cdot n$, $M >= 20$

Tabulka 8.2: Nastavení parametrů pro test „Nejdelší run v jednom bloku“

Minimální n	M
128	8
6 272	128
750 000	10 000

8.2.3 Run test

Slovem run se myslí nepřerušená posloupnost bitů stejné hodnoty. Test zkoumá délky a frekvenci jednotlivých runů. Během testu se spočítá hodnota V_n, která se dosadí do vzorce a z té se spočítá P-hodnota. Pokud je hodnota V_n příliš velká, značí to rychlé oscilování 1 a 0, v opačném případě pomalé.

Parametry

- n – délka sekvence, $n >= 100$

8.2.4 Nejdelší run v jednom bloku

Tento test se na rozdíl od předchozího nedívá na oscilaci runů, ale pouze na nejdelší v jednom bloku, a to bitů hodnoty 1. Za neúspěch se považuje, když je nejdelší run příliš krátký, ale i když je příliš dlouhý.

Parametry

- n – délka sekvence, $n >= 100$
- N – počet bloků, hodnota je vypočítána na základě délky bloku a sekvence
- M – délka jednotlivého bloku, hodnota je nastavena na základě Tabulky 8.2

8.2.5 Hodnost binární matice

Test spočítá hodnost matice určeného rozměru. Účelem je zjištění lineárních závislostí dat v jednotlivých maticích.

Parametry

- n – délka sekvence, $n >= 100$
- M – počet řádek, nastaven defaultně na 32
- N – počet sloupců, nastaven defaultně na 32

8.2.6 Diskrétní Fourierova transformace

Test se soustředí na krátké, opakující se vzorce v sekvenci. Opakující se vzorce se považují za vlastnost, kterou by pseudonáhodné generátory mít neměly.

Parametry

- n – délka sekvence, $n >= 100$

8.2.7 Nepřekrývaní při hledaní vzoru

Vybere se vzor a hledá se, kolikrát se objevil v sekvenci. Výsledek se porovná s předpokládanou hodnotou. Vzor se porovná se danou částí sekvence a vždy se o bit posune v případě, že vzor nesedí. Pokud sedí, přeskočí se na konec nalezeného vzoru v sekvenci. Neúspěch tohoto testu znamená, že některé vzory se opakují častěji, než by teoreticky měly.

Parametry

- n – délka sekvence, $n >= 100$
- m – vzdálenost bitů v testovaném vzoru, doporučeno 9 nebo 10
- M – délka podsekvence, $M > 0,01 \cdot n$

8.2.8 Překrývaní při hledaní vzoru

Cílem tohoto testu je zjistit, jestli se nějaké vzory neopakují častěji nebo naopak méně často, než by teoreticky měly. Jedná se tedy o test velmi podobný jako předchozí, jediným rozdílem je, že když je úspěšně nalezen vzor, tak se neskáče na konec tohoto vzoru, ale posune se na další bit v sekvenci.

Parametry

- n – délka sekvence, $n >= 100$
- m – vzdálenost bitů v testovaném vzoru, doporučeno 9 nebo 10
- M – délka podsekvence, $M > 0,01 \cdot n$

8.2.9 Maurerův univerzální statistický test

Test se soustředí na kompresi sekvence a následné porovnání velikostí se sekvencí původní. Velký rozdíl ve velikostech naznačuje nenáhodnost a je přijatá H_a. V parametrech neuvádím požadované minimální velikosti, při svém testování používám defaultní nastavení. Tabulka s požadovanými hodnotami je k nahlédnutí v dokumentu od NISTu [13].

Parametry

- n – délka sekvence
- L – délka každého bloku
- Q – počet bloků v sekvenci

8.2.10 Test lineární komplexnosti

Účel testu je zaměřit se na takové podsekvence, které mají vlastnost, že na základě znalosti předchozích bitů je algoritmus schopen dopočítat následující bity.

Parametry

- n – délka sekvence, $n >= 1\ 000\ 000$
- N – počet bloků, $N >= 200$
- M – počet bitů v bloku, $500 <= M <= 5\ 000$

8.2.11 Sekvenční test

Test vypočítává, kolikrát se objevují všechny možné m-bitové vzory. Výsledek se porovnává s teoretickou hodnotou. Pokud by délka vzoru nabývala hodnoty 1, test by pak byl obyčejným frekvenčním testem.

Parametry

- n – délka sekvence, velikost n je libovolná
- N – vzdálenost vzoru, $\lfloor \log_2(n) \rfloor - 2$

8.2.12 Test přibližné entropie

V principu podobný jako sekvenční test, rozdíl spočívá v tom, že se spočítají hodnoty pro vzory m a m+1, které se porovnají a zjistí se, jestli jejich rozdíl je přibližně rovnocenný s náhodnými daty. Neúspěch značí přílišnou pravidelnost ve vzorech.

Parametry

- n – délka sekvence, velikost n je libovolná
- N – délka vzoru, $\lfloor \log_2(n) \rfloor - 5$

8.2.13 Kumulativní test

Test je postaven tak, aby dokázal odhalit, jestli se v některých částích sekvence nekumuluje příliš mnoho bitů stejné hodnoty. Z celé sekvence nalezne místo, kde je nejvíce stejných bitů, a jejich počet porovná s očekávanou teoretickou hodnotou.

Parametry

- n – délka sekvence, $n >= 100$

8.2.14 Testovaní procházením sekvence

Účelem toho testu je zjistit, zda rozdíly v počtu bitů stejné hodnoty při průběhu sekvence osciluje kolem nuly tak často, jak se předpokládá. Test zkoumá nejenom kolikrát tento rozdíl v průběhu sekvence se rovnal 0, ale i jak dlouho se rozdíl pohyboval nad či pod nulou, než jí znovu dosáhl.

Parametry

- n – délka sekvence, $n >= 1\ 000\ 000$

8.2.15 Varianta k testovaní procházením sekvence

Na rozdíl od Testování procházením sekvence se tento test více zaměřuje na to, jak moc se rozdíl počtu bitů stejné hodnoty od nuly vzdálil. Opět se výsledek porovná s teoretickou hodnotou.

Parametry

- n – délka sekvence, $n >= 1\ 000\ 000$

8.3 Vstup pro testy

Testy pracují se souborem, který může být binární, ale je možné použít i textový dokument s bity v ASCII podobě. Pro vytvoření vstupních hodnot jsem použil stejný systém jako při šifrování obrázku. Tedy výstupní čísla šifry od 1–52 jsem zmenšil o jedna, aby reprezentovaly cifry v dvaapadesátkové soustavě, a bral jsem každé výstupní číslo jako cifru velice dlouhého čísla v dvaapadesátkové soustavě. To jsem posléze převedl do soustavy dvojkové a zapsal do binárního souboru.

Jak lze vidět z požadavků, minimální počet bitů jedné sekvence v některých testech je 1 000 000. Aby se daly testy považovat za průkazné, je podle specifikace potřeba otestovat alespoň 1 000 takovýchto sekvencí na jeden test. Abych tyto předpoklady splnil, tak testovací soubor obsahoval 1024 sekvencí,

každá délky 1 048 576 bitů. Aby byl vzorek co nejlepší, použil jsem 3 600 různých hesel délky 100, které vytvořily číslo v dvaapadesátkové soustavě o 53 248 cifrách. Toto jsem následně převedl do dvojkové soustavy a zapsal do testovacího souboru. Hesla byla vygenerována pomocí C++ funkce rand. Jak jsem uvedl dříve, není důvod používat lepší pseudonáhodnou funkci a s touto funcí se dobře pracuje.

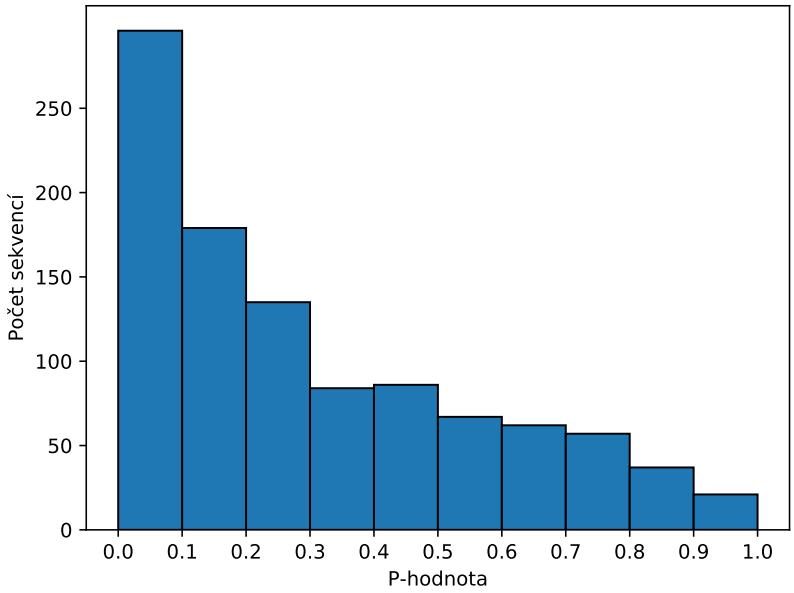
8.4 Výsledky NIST testů

Do testu jsem nastavil potřebné parametry a nyní interpretuji výsledky. Vzhledem k tomu, že výstupní soubor z testu, který obsahuje konečné výsledky, je velice dlouhý, neuvádím ho ani do přílohy. Je ale k nahlédnutí na přiloženém datovém nosiči. Zde uvedu jen krátkou ukázku na Obrázku 8.3.

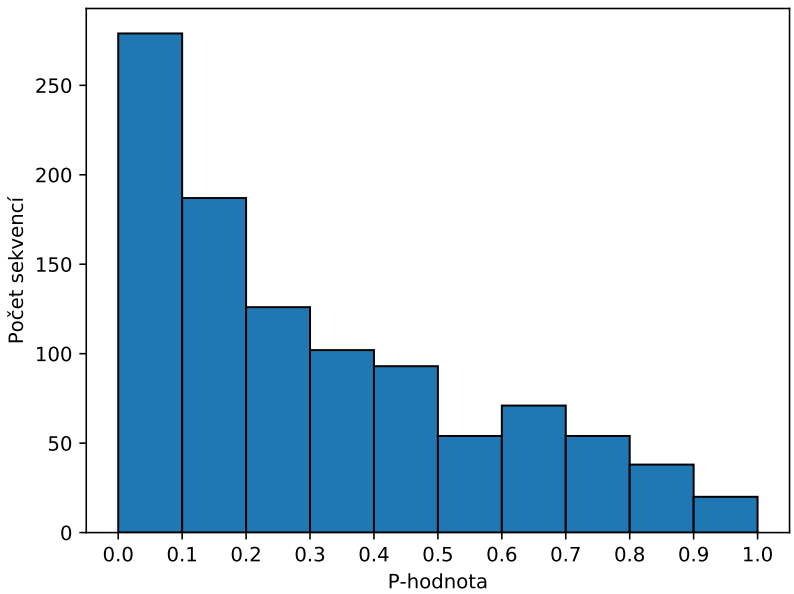
C1-10 jsou intervaly, do kterých se rozdělily P-hodnoty pro každou sekvenci. Proportion říká, kolik sekvencí z kolika přijato H_0. Testy, které neprošly jako celek, jsou označeny hvězdičkou. Jedná se o test Přibližné entropie.

Dále co není obsaženo ve výpisu, tak neprošly 3 ze 147 testů Nepřekrývání při hledání vzorů (pro každý vzor se dělá samostatný test). Ale vzhledem k tomu, že každý test se rozhoduje na základě kritické hodnoty a jak jsem již popsal, nastávají chyby prvního druhu, nepřijetí těchto testů nemusí nic znamenat a může se jednat o statistický jev.

Větší význam bych přikládal tomu, že signifikantně neprošel test Přibližné entropie. K tomu, abych mohl učinit závěr, potřebuju data na porovnání. Proto jsem podrobil NIST testům data vygenerovaná proudovou šifrou ChaCha20. Výsledek ukázal, že také neprošla tímto testem. Obrázek 8.2 zobrazuje porovnání P-hodnot šifer Solitaire a ChaCha20 v Testu přibližné entropie. A vzhledem k tomu, že ChaCha20 je používaná šifra považovaná za bezpečnou a má podobné výsledky se Solitaire, závěr činím takový, že vlastnosti Solitaire v generování pseudonáhodných čísel jsou uspokojivé. Jak moc podobné výsledky ukazuji na Obrázku 8.4. Graf porovnává, jak podobné výsledky mají šifry ChaCha20 a Solitaire v NIST testech.



(a) Solitaire



(b) ChaCha20

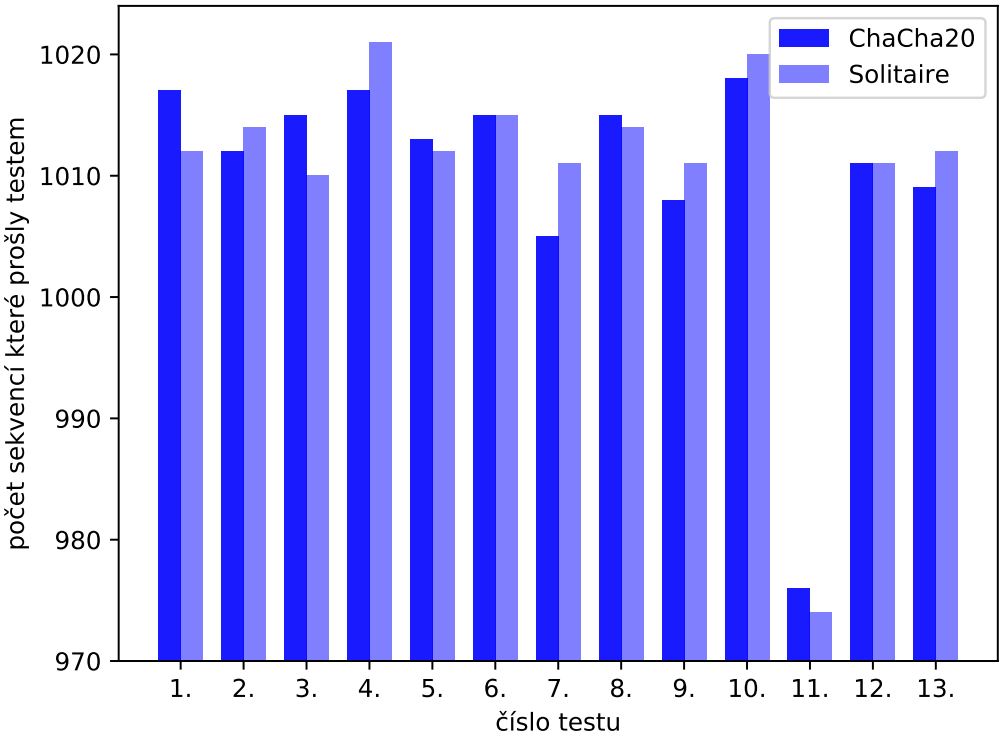
Obrázek 8.2: Rozdělení P-hodnot pro Test přibližné entropie, testovaná data vygenerovaná Solitaire vs. data vygenerovaná ChaCha20

```
C1  C2  C3  C4  C5  C6  C7  C8  C9 C10  P-VAL  PROPORT TEST

--------------------------------------------------------------

108  83 116 120  90  83 100 111 106 107  0.086201 1012/1024  Freq
112  98 109  99 114  81  82 112 126  91  0.023734  1014/1024  BlockFr
103  96 101 116  95 105  98  97 110 103  0.917746  1009/1024  CumSums
102 107  87 100 109 113 111  84 107 104  0.493241  1012/1024  CumuSums
90  96  98 132 103 100 100 104  87 114  0.105925  1021/1024  Runs
110 106  84 100 104 102 124  96  76 122  0.019950  1012/1024  LonRun
87 106  99 101 103 113 104  97 102 112  0.841091  1015/1024  Rank
119  95  96 110 108 100 102 102  82 110  0.415844  1011/1024  FFT
                            .
.                           .
                            .
296 179 135  84  86  67  62  57  37  21  0.00 *  974/1024 * AppEntr
                            .
                            .
                            .
96 108 109  94  98 102  90 100 116 111  0.726107 1009/1024  Serial
111 110 109 104  84 102 102 100  97 105  0.789657 1012/1024  Serial
108 100  99  95 103 102 106  98 102 111  0.990166 1012/1024  LinComp
```

Obrázek 8.3: Výpis NIST testů pro Solitaire



Obrázek 8.4: Počet přijatých sekvencí v jednolivých testech pro ChaCha20 a Solitaire

```
C1  C2  C3  C4  C5  C6  C7  C8  C9 C10  P-VAL  PROPORT TEST

--------------------------------------------------------------
119 107  86  92 112 102 112  88 100 106  0.30  1017/1024  Freq
102 104 106  97 121 103  90 105 101  95  0.74  1012/1024  BlockFr
111 113  86 100  99 127 107 102  86  93  0.11  1014/1024  CumSums
110 121  97  97  91  88 114 102 103 101  0.41  1016/1024  CumSums
 96 116  80 105 104 103 121 104 103  92  0.22  1017/1024  Runs
101 104 105  87  92 110 102 111 100 112  0.76  1013/1024  LonRun
106 114 110 104  93 109 116  88 105  79  0.18  1015/1024  Rank
120 107 105 113  93 101  92 100 109  84  0.33  1005/1024  FFT
                                .
                                .
                                .
279 187 126 102  93  54  71  54  38  20  0.0 *  976/1024 * AppEntr
                                .
                                .
                                .
102  77 108  98  99 113 103  96 108 120  0.224590 1009/1024  Serial
 90 112  85 109  87 109  93 111 108 120  0.140862 1013/1024  Serial
115  92 111 103  86 100 101  97 106 113  0.576114 1009/1024  LinComp
```

Obrázek 8.5: Výpis NIST testů pro ChaCha20

Kapitola 9

Používání Solitaire

V předchozích kapitolách jsem často upozorňoval jak šifru nepoužívat a jaké jsou její možné slabiny. Nyní se zaměřím na to, jak naopak šifru používat, aby se to dalo považovat za bezpečné. Mimo jiné vysvětlím, jak prakticky vyřešit problém, že se nemůže dvakrát použít stejný klíč. Měřím, jak je aplikování šifry v ruce časově náročné. Dále přidávám praktické tipy, které pomohou šifru rychleji vykonávat.

Šifra je navržena tak, aby uměla šifrovat jen velká písmena anglické abecedy. Cokoli jiného, jako jsou čísla, malá písmena či interpunkce je považováno za nesprávný vstup. OT se rozdělí na pětice a pokud počet znaků není dělitelný 5, tak se konec případně doplní písmemy X. Dále je již klasický postup, zvolím klíč nebo z hesla klíč získám a OT šifruji.

Solitaire má bohužel tu vlastnost, že když se při jakémkoliv kroku udělá chyba, tak se nenávratně pokazí zbytek sekvence a dešifrování je nemožné. Lidé často při mechanické práci ztratí pozornost a chyba je velmi častá, proto B. Schneier doporučuje, aby se zašifrování provádělo minimálně dvakrát a poté porovnaly výsledky, jestli se liší.

9.1 Praktické tipy

V krocích 1, 2, 3 se hledají v balíčku jokeři, to velice zdržuje. Proto ze zkušenosti mohu doporučit si jokery označit lepícím štítkem, aby na první pohled bylo jasné, kde se v balíčku nachází. Další praktické doporučení je nedělat si na stole hromádky: když se odpočítávají karty, způsobí to obrácení pořadí, a to je silně nežádoucí. Další dobrá rada je, nesnažit se hned výstupní kartu dělit modulo 26, rychlejší je vygenerovat celý proudový klíč a ten zmodulovat najednou. Zbytečné je taká každé dělení se zbytkem počítat, lepší je se podívat zpátky, jestli jsem již takové či blízké číslo nedělil, a hodnotu z toho odvodit. Nejvíce časově náročné operace jsou ty, kde se odpočítávají karty, hodně se mi osvědčilo karty počítat ne po jedné ale po čtyřech kusech.

Pro skutečně rychlé a bezchybné šifrování a dešifrování je potřeba určitý trénink, ale také perfektní znalost abecedy. Konkrétně kolikáté které písmeno je. Je hodně časově náročné pořád abecedu odpočítávat.

9.2 Bezpečné znovupoužití klíče

Když se v praxi šifruje více zpráv, je nemožné si pro každou pamatovat nový klíč. ChaCha20 používá z tohoto důvodu nonce. Solitaire nic takového v popisu [4] nemá, ale nyní popíši způsob, jak nonce používat i v Solitaire.

Prvním krok je nastavit balíček do podoby klíče, tedy buď pomocí hesla nebo zapamatováním si klíč. Poté je potřeba použít nonce. Nonce se používá jako heslo, tedy je to textový řetězec, podle kterého se balíček míchá. Po nastavení klíče se balíček podle nonce zamíchá a až poté je možné bezpečně šifrovat. Nonce může být veřejná informace, tady není problém pokaždé používat jiný, protože je bezpečené si ho někam zaznamenat. Veřejná informace to může být z důvodu, že není možné dešifrovat pouze se znalostí nonce a bez znalosti klíče.

Nonce by měl být tak dlouhý, aby balíček po zamíchání noncem byl nezávislý oproti klíči. V kapitole 3 jsem tento problém řešil a došel jsem z závěru, že je potřeba alespoň 80 znaků dlouhý řetězec. Tedy nonce by měl být alespoň 80 znaků dlouhý. Bohužel nonce přidává další kroky, které je potřeba vykonat, a tím se výrazně zvyšuje čas šifrování.

9.3 Po skončení šifrování

Když se úspěšně povede zašifrovat OT, je potřeba balíček zamíchat. Je to z toho důvodu, že šifra je pro většinu stavů inverzní a kdyby balíček padl do nesprávných rukou a nebyl zamíchán, je prolomení ŠT snadné. Pro dostatečné promíchání je potřeba provést 7krát techniku míchání zvanou „riffle shuffle“ aby nebyla pozorovatelná závislost na předchozí sekvenci karet. Jak se tato technika míchání provádí a důkaz, že skutečně 7krát zamíchat stačí, je možné si přečist v [10].

9.4 Naměřené hodnoty

Nyní uvedu přesné hodnoty, jak dlouho mi v průměru trvalo zašifrovat jeden znak OT. Jedná se o hodně subjektivní měření, nelze z něho vycházet pro odhadování časů k vykonání šifry pro jiné uživatele, ale pro orientační čas je to dostačující.

Pokud jsem používal základní techniku bez označených jokerů, čas strávený při zašifrování jednoho znaku byl v průměru 51 vteřin. Pokus jsem prováděl na 100 znacích, které jsem zašifroval za 1 hodinu a 28 minut.

Při použití označených jokerů se čas zrychlil přibližně o 10 vteřin na znak. Jedná se o docela podstatné zrychlení. Předchozí čas jsem při šifrování jiných 100 znaků zrychlil na čas 1 hodina a 8 minut.

9.5 K čemu Solitaire využívat

Jak jsem ukázal v kapitole 6, proudový klíč není zcela náhodný. Tato nenáhodnost není dramaticky pravděpodobná, ale pokud se šifrují dlouhé texty, jistě se to projeví. Proto je vhodné Solitaire používat jen pro skutečně krátké zprávy.

Krátkým zprávám nahrává i fakt, že se udělá méně operací a tím je menší pravděpodobnost chyby, která je pro dešifrování fatální. Totéž platí i z časového hlediska. Zkrátka je vhodné Solitaire používat k tomu, k čemu je určený, to znamená pro krátké zprávy, a to jen v případě, když není vhodné, či možné, aby šifrovací nebo dešifrovací strana použila počítač.

Závěr

Práce se zaměřovala na ruční šifru Solitaire. Cílem bylo podrobně popsat její principy a to, jak je možné šifru samotnou využít. Součástí cíle bylo vytvořit funkční implementaci schopnou šifrovat a dešifrovat. Dále probrat, jaký je rozdíl mezi použitím klíče a hesla a prozkoumat jejich vlastnosti při použití. Dalším cílem bylo probrat možné slabiny a když jsou nějaké nalezené, navrhnout možnosti řešení. Také jsem se měl podívat na možné závislosti nebo nenáhodnosti ve výstupním proudovém klíči.

Všechny body jsem v práci splnil a výsledkem je funkční implementace šifry Solitaire v jazyce C++. Slabiny v šifře jsem buď potvrdil anebo jako v případě pozicí jokerů i objevil a dokázal. Slabiny nepovažuji za fatální, ale snižují důvěru v používání šifry. Dále jsem potvrdil, že je potřeba používat klíč dlouhý alespoň 80 znaků, a z hlediska závislosti na původním balíčku je jedno, jestli je náhodný nebo nikoli. Výstup ze své implementace jsem podrobil testům na náhodnost a výsledek považuji, vzhledem k tomu, že podobný měla i ChaCha20, za velmi dobrý. Šifru jsem doporučil používat jenom v případě, že není možné či vhodné použít počítač, a to jen na krátké zprávy.

V budoucnu vidím možnost pokračovat ve zkoumání pozic karet v balíčku v závislosti na jokerech a tím usnadnit útok hrubou silou. Také bych se zaměřil na možnost cyklického pohybu jokerů v balíčku.

Literatura

[1] Global Partners Digital: *World map of encryption laws and policies [online].* [cit. 2019-02-26]. Dostupné z: https://www.gp-digital.org/world-map-of-encryption/

[2] McDonald, N. G.: *Past present and future methods oof chryptography and data encryption [online].* [cit. 2019-03-22]. Dostupné z: https://pubweb.eng.utah.edu/~nmcdonal/Tutorials/EncryptionResearchReview.pdf

[3] Roisman, J.: *Ancient Greece from Homer to Alexander: The Evidence.* Wiley-Blackwell, 2011, ISBN 978-1405127769, st. 197.

[4] Schneier, B.: *The Solitaire Encryption Algorithm [online].* 1999, [cit. 2019-03-26]. Dostupné z: https://www.schneier.com/academic/solitaire/

[5] Pogorelov, B.; Pudovkina, M.: *Properties of the Transformation Semigroup of the Solitaire Stream Cipher [online].* 2003, [cit. 2019-03-26]. Dostupné z: https://eprint.iacr.org/2003/169

[6] Tounsi, W.; Justus, B.; Boulahia, N. C.; aj.: *Probabilistic Cycle Detection for Schneier's Solitaire Keystream Algorithm [online].* 2014, [cit. 2019-03-26]. Dostupné z: https://ieeexplore.ieee.org/document/6901648?arnumber=6901648&tag=1

[7] Crowley, P.: *Problems with Bruce Schneier's Solitaire [online].* 2001, [cit. 2019-03-26]. Dostupné z: http://www.ciphergoth.org/crypto/solitaire/

[8] Klein, A.: *Stream Ciphers.* Springer, 2013, ISBN 978-1447150787.

[9] NIST: *Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths [online].* 2015, [cit. 2019-03-26], st. 11. Dostupné z: https://csrc.nist.gov/csrc/media/publications/sp/800-131a/rev-1/final/documents/sp800-131a_r1_draft.pdf

[10] Bayer, D.; Diaconis, P.: *Trailing The Dovetail Shuffle To Its Lair [online]*. 1992, [cit. 2019-03-28]. Dostupné z: https://projecteuclid.org/euclid.aoap/1177005705

[11] Bernstein, D. J.: *ChaCha, a variant of Salsa20 [online]*. [cit. 2019-03-28]. Dostupné z: http://cr.yp.to/chacha/chacha-20080128.pdf

[12] Heinrich, M.: *Útok postranním kanálem na šifru ChaCha [online]*. [cit. 2019-03-28], Praha, 2019. st. 13-17. Bakalářská práce. Vedoucí práce Ing. Jiří Buček.

[13] Rukhin, A.; Soto, J.: *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications [online]*. 2010, [cit. 2019-04-02]. Dostupné z: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-22r1a.pdf

Příloha A

Seznam použitých zkratek

AES Advanced encyption standard

AES-NI Advanced Encryption Standard New Instructions

NIST National Institute of Standards and Technology

PRNG Pseudo Random Number Generator

OT Otevřený text

ŠT Šifrovaný text

CTR Counter mode

RC4 Rivest Cipher 4

SSL Secure Sockets Layer

TLS Transport Layer Security

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu datového nosiče
- `src`
  - `Implementation` ..... zdrojové kódy implementace
  - `NonrandomnessInStreamkey` ..... testy pro měření nenáhodnosti v proudovém klíči
  - `PasswordLength` ..... test k minimální délce hesla
  - `SameKeyTwice` ..... použití stejného klíče pro šifrování dvou obrázků
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF