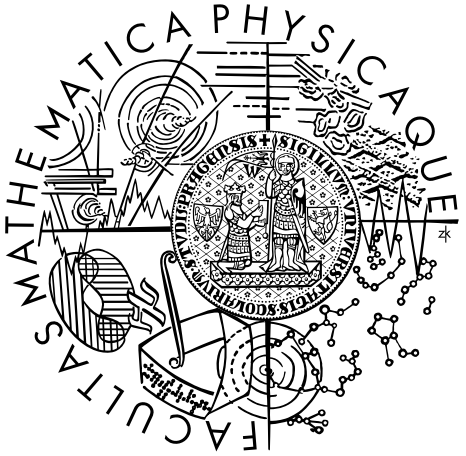
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Jmďž″no Pďž″dž″jmenďž″

Ndž″zev prdž″ce

Ndž″zev katedry nebo dž″stavu

Vedoucí bakalářské práce: Vedoucdž″ prdž″ce
Studijní program: studijnďž″ program
Studijní obor: studijnďž″ obor

Praha ROK

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Název práce: Nďž″zev prďž″ce

Autor: Jmďž″no Pďž″ďž″jmenďž″

Katedra: Nďž″zev katedry nebo ďž″stavu

Vedoucí bakalářské práce: Vedoucďž″ prďž″ce, katedra

Abstrakt: Abstrakt.

Klíčová slova: klďž″ďž″ovďž″ slova

Title: Name of thesis

Author: Jmďž″no Pďž″ďž″jmenďž″

Department: Name of the department

Supervisor: Vedoucďž″ prďž″ce, department

Abstract: Abstract.

Keywords: key words

Podďž″kovďž″nďž″.

Obsah

Úvod

Toto je implementačně-experimentální práce. Cílem mojí práce je vytvořit šifrovací algoritmus využívající celulární automaty.

1. Celulární automaty

1.1 Co jsou to celulární automaty

Celulární automat je diskrétní model, který se skládá z pravidelné mřížky buněk. Buňky se nacházejí v určitých stavech, přičemž množina stavů a pravidla pro přechod mezi stavy jsou společná pro celý automat. Celulární automat se vyvíjí diskrétně v čase (celý najednou).

Specialitou celulárních automatů je to, že i velmi jednoduchá sada pravidel může vést k velmi komplexnímu chování. Celulární automaty našly využití jako modely v biologii, chemii, fyzice, ale také třeba jako nástroj při procedurálním generování terénu pro počítačové hry.

1.2 Elementární celulární automaty

Wofram popisuje 256 elementárních celulárních automatů. Jedná se o binární 1D automaty, kde nový stav každé buňky závisí pouze na jejím stavu a stavu přímých sousedů. Formálně by se dal přechod zapsat jako:

$$x_i(t+1) = f(x_{i-1}(t), x_i(t), x_{i+1}(t))$$

, kde

$$f : \{0,1\}^3 \to \{0,1\}$$

, tudíž existuje $2^{2^3} = 2^8 = 256$ takových funkcí f.

1.3 Jiné 1D celulární automaty

- Nemusíme se omezovat jen na těsné sousedy.
- Můžeme povolit více než 2 stavy.
- Významným druhem celulárních automatů jsou totalistické automaty. Nehledě na to, jak velké se používá okolí a kolik stavů buňlky nabývají, v totalistických automatech se pouze číselně posčítají hodnoty hodnoty buňky s celým jejím okolím a podle součtu hodnot se přiřadí výsledná hodnota. Přechodová funkce pro totalistický automat s okolím velikosti o a počtem stavů s se tedy dá vyjádřit jako:

$$f : \{0,..,(s-1)(o+1)\} \to \{0,..,s-1\}$$

1.4 Dvourozměrné celulární automaty

V oblasti 2D celulárních automatů se používají hlavně totalistické automaty, protože vytváření jiných typů pravidel by bylo příliš složité. Typických příkladem totalistického 2D automatu je Game Of Life. To je dvourozměrný automat nad binární abecedou používající 8-okolí, který se řídí pravidlem, že živá buňka přežívá,

pokud má 2 až 3 zivé sousedy (jinak umírá), zatímco mrtvá buňka obžije, pokud má právě 3 živé sousedy. Game Of Life se stal zdrojem mnoha různých hříček. Nicméně pro účely šifrování se nezdá být moc užitečný, protože nejde dostatečně parametrizovat.

1.5 Implementace celulárních automatů

V teorii se typicky uvažují celulární automaty na nekonečném hřišti. V počátečním stavu mají se však všechny nenulové hodnoty vyskytují jen uvnitř nějaké konečné oblasti buněk. Nenulové hodnoty se pak ale mohou neomezeně rozrůstat do všech stran. Rychlost tohoto rozšiřování lze zhora omezit na základě velikosti okolí aplikovaného pravidla.

V praxi implementujeme celý automat na konečném hřišti. Možnosti jsou dvě. Buď celý automat zacyklíme a jeho vývoj v čase „pokresluje nekonečnou válcovou plochu“, nebo automat na stranách ohraničíme a při aplikaci pravidel na okraji hřiště čteme nulové hodnoty za jeho okrajem.

Výhodou je snadná implementace a nízké paměťové nároky. Nevýhodou je, že se vývoj celého automatu periodicky opakuje, pokud provedeme dostatečný počet kroků. Délku této periody lze bohužel odhadnout pouze zhora.

2. Šifrování

Věda zabývající se šifrováním se nazývá *kryptografie*. Lámáním šifer se zase zabývá *kryptoanalýza*. Úkolem šifrování je uchovat a předat tajnou zprávu tak, aby ji mohl přečíst ten, pro koho je určena, ale už nikdo jiný. Dále se někdy za cíl dává ověřitelnost autora zprávy.

Původní čitelná zpráva se nazývá *plaintext*. Data po zašifrování se nazývají *ciphertext*. Pro převod plaintextu na ciphertext je potřeba použít šifrovací algoritmus. Ten by měl zároveň být schopen převézt ciphertext zpět na plaintext (tzv. dešifrování). Protože fungování šifrovacího algoritmu se velmi snadno vyzradí, zásadní roli hraje *šifrovací klíč*. Pokud se jedná o *symetrickou kryptografii*, tak stejný klíč slouží i k dešifrování. V případě *asymetrické kryptografie* se používají dva různé klíče. Šifrovací resp. dešifrovací klíče v asymetrické kryptografii se s ohledem na jejich použití nazývají jako *veřejný* resp. *tajný* klíč.

Před zašifrováním zprávy se často provádí její *komprese*. To má za následek nejen úsporu přenosového pásma a množství práce (času) šifrovacího algoritmu, ale velkou výhodou je dosažení výrazně rovnoměrnějšího pravděpodobnostního rozdělení na prostoru plaintextů, což výrazně komplikuje kryptoanalýzu.

2.1 Několik ukázek

bla bla bla

V této práci se budeme věnovat vytvoření algoritmu na protahování klíčů (anglicky key stretching). Cílem je z krátkého klíče (který lze například v krátkém čase přenést pomocí RSA) vygenerovat dlouhý klíč (kterým lze přeXORovat celý soubor). Je to tedy podobný úkol jako naprogramovat *Key Stream Generator*, akorát my budeme dopředu vědět cílenou délku výsledného klíče.

Jako Key Stream Generator se dá použít například mnoho blokových šifer. U blokových šifer záleží na módu operace. Při *Cipher Block Chaining* (CBC, PCBC) či *Cipher Feedback* (CFB) módu zašifrování druhé části plaintextu záleží na výsledku zašifrování první části, tudíž to není Key Stream Generator. Ale při zapojení jako třeba *Counter* (CTR) vzniká proud bitů bez znalosti plaintextu, takže získáme Key Stream Generator. Dobrým příklalem je třeba AES-CTR.

2.2 Testy

Je příliš obtížné ukázat o šifrovacím algoritmu, že je doopravdy kvalitní. Jako záruka kvality se proto v praxi používá spíše jeho zveřejnění na několik let, aby ho měli šanci oponovat nejlepší odborníci. Pokud se ani po několika letech neukáže jeho slabina, je šifrovací algoritmus považován za dost dobrý. Naštěstí alespoň ty velmi špatně šifrovací algoritmy je možné rychle rozpoznat statistickými testy. Na to se zaměříme v této práci.

2.2.1 Testy na úrovni jednotlivých výstupů

Pro kryptografii je potřeba, aby dlouhé klíče měly vlastnosti pseudonáhodných posloupností. Pochopitelně zde není možné dosáhnout, aby všechny dlouhé klíče byly stejně pravděpodobné a dosáhli bychom tak „pravé náhodnosti“, ale můžeme alespoň otestovat konkrétní výstup, jestli se podobá náhodné posloupnosti.

Třída Crypto.RandomnessTesting obsahuje následující metody.

- EntropyTest(BitArray b, byte lengthLimit) : Testuje entropii bloků o velikosti od 1 po lengthLimit (používána hodnota 10). Jde o sledování frekvence jednotlivých bloků. V náhodné posloupnosti by měly být všechny bloky stejné délky přibližně stejně časté. Pro krátké posloupnosti (zde pod 10 tisíc bitů) samozřejmě není možné, aby se všechny bloky (zde délky 10) vyskytly, takže jsou všechny výsledky porovnány s maximálním možným výsledkem. Výstupem je vážený průměr, kde mají testy entropií všech délek výsledky v rozsahu 0 až 1. Optimální hodnota je 1.

- CompressionTest(BitArray b) : Zkusí data zkomprimovat pomocí programu gzip s optimální úrovní komprese a vrátí poměr mezi novou a původní velikostí. O posloupnostech, které lze zkomprimovat, je známo, že nejsou dokonale náhodné. Konkrétně velikost dat po kompresi je horním odhadem na Kolmogorovskou složitost. Optimální hodnota je 1.

2.2.2 Testy na úrovni celého zobrazení

Skutečnost, že výstupem algoritmu je pseudonáhodná posloupnost čísel, je jistě dobrá. Ale co když algoritmus všem vstupům přiřadí stejnou pseudonáhodnou posloupnost? Nebo jeden konkrétní bit na vstupu neovlivní výsledek? Takovou nekvalitu musí objevit druhá skupina testů.

Budeme zde zkoumat vlastnosti algoritmů jako vlastnosti celého zobrazení z krátkých klíčů do dlouhých klíčů. Protože si budeme často klást otázky typu „Jak moc se liší výstup A od výstupu B?“, tak by bylo vhodné zavést nějaké hodnocení, ideálně s vlastností metriky. Porovnávat budeme vždy jen výstupy stejné délky. Oblíbenými metrikami pro řetězce jsou *Hammingova vzdálenost* a *Levenshteinova vzdálenost*. Hamming měří počet pozic, na kterých se řetězce liší. Levenshtein měří minimální počet potřebných změn k tomu, abychom z jednoho řetězce dostali ten druhý. Jako změnu je možné provést záměnu znaku, smazání znaku, nebo dopsání znaku na libovolné místo.

Hammingova vzdálenost je triviálně horním odhadem na Levenshteinovu vzdálenost. V případě náhodných binárních řetězců je jejich hodnota často stejná, ale někdy můžu být Levenshtein výrazně nižší. Například když zrotujeme řetězec o jednu pozici, tak Levenshtein dává vzdálenost 2, zatímco Hamming může dát hodně vysoké číslo. Významná pro nás bude rychlost výpočtu. Hammingovu vzdálenost lze triviálně určit v lineárním čase, ale výpočet Levenshteinovy vzdálenost zabere čas kvadratický. Že to rychleji nejde, se nelze divit, protože Levenshtein vlastně spouští prohledávání prostoru editací. Díky dynamickému programování to lze provést alespoň v tom kvadratickém čase.

Třída Crypto.FunctionTesting obsahuje následující metody. Podle nastavení v konstruktoru mohou všechny testy používat buď Hammingovu, nebo Levensteinovu vzdálenost. Dále uváděno podle Hamminga.

- TestBitChange(IKeyExtender algorithm, int ratio) : Testuje, jak velká část bitů výstupu se změní při změně jednoho bitů vstupu. Metoda sampluje náhodné vstupy a pro každý z nich zkouší změnit zvlášť všechny bity. Optimální hodnota je 0,5.

- TestAverageDistance(IKeyExtender algorithm, int ratio) : Testuje průměrnou vzdálenost výstupů příslušející dvěma různým náhodně zvoleným vstupům. Optimální hodnota je 0,5.

- TestLargestBallExactly(IKeyExtender algorithm) : Testuje, jaká největší koule se dá vměstnat do prostoru výstupů tak, aby neobsahovala žádný vygenerovatelný dlouhý klíč. Zkouší úplně všechny vstupy na malém prostoru a ty natahuje na dvojnásobek. Motivací je, že pokud zobrazení nazaplňuje prostor dostatečně rovnoměrně, pak to rozpoznáme tak, že se do prostoru vejde velká koule.

- TestLargestBallApprox(IKeyExtender algorithm) : Testuje to samé, ale používá delší vstupy, které už nezvládá vyzkoušet všechny, takže je sampluje náhodně.

2.3 asdf

bla bla

3. Způsoby protahování klíčů

Bylo vytvořeno několik různých algoritmů na protahování klíčů využívajících celulární automaty. Je možné si zvlášť zvolit algoritmus (případně některé z nich lze parametrizovat) a zvlášť do něj vložit libovolný binární celulární automat.

Kromě opravdových algoritmů byly vytvořeny ještě dva falešné algoritmy pro účely testování. Jeden z nich jen kopíruje kratší klíč dokola. Druhý generuje pseudonáhodnout posloupnost bez ohledu na vstup.

Závěr

Seznam obrázků

Seznam tabulek

Seznam pouďž"itďž"ch zkratek

Pdž″dž″lohy