Univerzita Komenského, Bratislava
Fakulta matematiky, fyziky a informatiky

Šifrovanie zachovávajúce formát

Diplomová práca

2015 Bc. Ladislav Bačo

Univerzita Komenského, Bratislava
Fakulta matematiky, fyziky a informatiky

Šifrovanie zachovávajúce formát

Diplomová práca

Študijný program:	Informatika
Študijný odbor:	2508 informatika
Katedra:	Katedra informatiky
Vedúci:	doc. RNDr. Martin Stanek, PhD.

Bratislava, 2015 Bc. Ladislav Bačo

98071198

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Ladislav Bačo
Študijný program: informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: 9.2.1. informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Šifrovanie zachovávajúce formát
Format preserving encryption

Cieľ: Analýza existujúcich schém pre šifrovania zachovávajúce formát, preskúmanie súvislostí týchto schém s miešaniami kariet a implementácia vhodnej schémy s podporou rôznych formátov v jazyku Python.

Vedúci: doc. RNDr. Martin Stanek, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: doc. RNDr. Daniel Olejár, PhD.

Dátum zadania: 20.11.2013

Dátum schválenia: 26.11.2013 prof. RNDr. Branislav Rovan, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie

Ďakujem môjmu vedúcemu doc. RNDr. Martinovi Stanekovi, PhD. za jeho neúnavnú pomoc, cenné rady, inšpiratívne konzultácie a podporu. Taktiež ďakujem rodičom a najmä priateľke za podporu počas písania práce.

Abstrakt

Šifrovania zachovávajúce formát (FPE) umožňujú šifrovať údaje tak, aby výsledné šifrové texty mali rovnaký formát ako vstupné texty. FPE predstavujú pomerne mladú oblasť kryptológie. V práci sme predstavili koncept FPE a ich súvis s miešaniami kariet.

Následne sme navrhli nový model bezpečnosti RUL (kartová ruleta), ktorý zodpovedá Thorpovej analýze pomocou matice pravdepodobností a odvodili jeho vzťahy s doteraz známymi modelmi používanými pri analýze FPE. Následne sme model RUL využili pri hľadaní ideálneho miešania kariet spĺňajúceho nami definované kritériá.

Implementovali sme knižnicu pyFNR, v súčasnosti najvýkonnejšiu knižnicu s podporou FPE v jazyku Python, ktorá zároveň podporuje najširšiu sadu rôznych formátov spomedzi dostupných knižníc.

Kľúčové slová: FPE, šifrovania zachovávajúce formát, miešanie kariet, Thorpovo miešanie, matica pravdepodobností, RUL, FNR

Abstract

Format preserving encryption (FPE) allows us to encrypt data in the way that the ciphertext format is the same as the plaintext format. FPE is relatively new field of cryptology. In this thesis we introduce the FPE and its relation to card shuffles.

We have proposed a new security notion, RUL (card roulette) game, corresponding to the Thorp's analysis utilizing the probability matrix of a shuffle. We have investigated relations between RUL game and other security notions of FPE. Then, we have used the RUL security notion in finding the ideal shuffle that meets our conditions.

We have also implemented library pyFNR, the fastest FPE library in Python. Our library also supports the largest number of formats in comparison to existing libraries.

Keywords: FPE, format preserving encryption, card shuffle, Thorp shuffle, probability matrix, RUL, FNR

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam výpisov programov

Úvod

V dnešnej dobe nadobúda potreba šifrovania oveľa väčšiu dôležitosť ako kedykoľvek predtým. Vďaka moderným technológiám je prístup k informáciám jednoduchší. Služby sú dostupnejšie. Avšak pri používaní rôznych online služieb o sebe častokrát prezrádzame aj osobné informácie. Robíme to vedome, napr. pri nákupoch v internetových obchodoch alebo pri registráciách do rôznych služieb. Naše osobné údaje sú tiež uložené v informačných systémoch verejnej správy alebo súkromných firiem.

Prístup k citlivým údajom (osobné údaje, utajované skutočnosti) by mal byť riadený tak, aby sa k nim nemohol dostať ktokoľvek. Navyše, v prípade bezpečnostných incidentov ak aj niekto získa uložené údaje, nemal by byť schopný zistiť z nich nič viac, ako keby tieto údaje nemal. To možno dosiahnuť vhodným šifrovaním údajov. Aj napriek zjavnej dôležitosti sú častokrát citlivé údaje uložené v databázach nešifrované, prípadne zabezpečené len slabou a ľahko prelomiteľnou šifrou. Za všetky spomeňme napríklad útok na hráčsku sieť Sony Playstation Network z apríla 2011 [40], pri ktorom sa útočníkom podarilo získať prístup k citlivým informáciám 77 miliónov používateľov.

Vo väčšine krajín býva potreba zabezpečenia citlivých prípadne utajovaných údajov zakotvená aj v legislatíve, u nás napríklad v Zákone o ochrane utajovaných skutočností [34]. V niektorých prípadoch sú teda prevádzkovatelia informačných systémov povinní zabezpečiť svoje systémy, a nie je to iba otázka ich dobrej vôle a proaktívnej ochrany spracovávaných údajov.

Informačné systémy zvyčajne obsahujú nejakú formu kontroly údajov, či už priamo v aplikačnej časti, databázovej časti, alebo v oboch častiach. Použitím bežných spôsobov šifrovania na dané údaje dostaneme ako výsledok údaje v úplne inom formáte, zvyčajne bitové reťazce namiesto napríklad rodných čísel. Teda nie je možné tieto údaje iba zašifrovať a uložiť do databázy, pretože kontrola formátu zlyhá.

Jednoduchšie riešenie by bolo použiť šifrovanie, ktoré by na výstupe vrátilo šifrový text v rovnakom formáte, v akom bol vstupný otvorený text. Takéto šifrovanie by bolo možné vložiť napríklad nad databázovú časť systému, čím by sme dosiahli, že všetky citlivé údaje v databáze by boli zabezpečené a zároveň by sme mohli použiť existujúce databázové riešenie bez potreby čokoľvek upravovať, lebo z pohľadu databázy by sa do

nej ukladali údaje v rovnakom formáte ako predtým.

Výhodu takéhoto šifrovania si uvedomili kryptológovia a posledných rokoch sa takýmto šifrovaniam intenzívne venuje niekoľko prác, napr. [1, 4, 29, 45]. Bellare, Rogaway a Spies v roku 2010 poslali NISTu (National Institute of Standards and Technology) návrh na štandardizáciu algoritmu nazvaného FFX [3], ktorý je v súčasnosti spolu s ďalšími návrhmi publikovaný NISTom ako draft [38].

Ciele práce

Táto práca sa zaoberá súčasnými technikami používanými pri šifrovaní zachovávajúcom formát, ich bezpečnosťou a efektívnosťou. Preskúmame tiež úzky súvis niektorých techník s inými, zdanlivo nesúvisiacimi oblasťami (miešanie kariet), zhodnotíme aktuálny stav pripravenosti týchto šifrovaní na použitie v praxi a implementujeme knižnicu v jazyku Python s podporou pre šifrovania zachovávajúce formát.

Štruktúra práce

Kapitola 1 obsahuje úvod do problematiky šifrovaní zachovávajúcich formát, základné pojmy a definície, zoznámenie sa s kryptoanalytickými modelmi bezpečnosti. Taktiež uvádzame základné princípy používané pri konštrukcii schém pre šifrovania zachovávajúce formát.

V kapitole 2 sa zaoberáme súvisom miešaní kariet a šifrovaní zachovávajúcich formát. Aplikujeme metódu využitú pri analýze vplyvu miešania na úspech v kartových hrách na analýzu šifrovacích schém.

Kapitola 3 sa venuje aktuálnym návrhom schém, ktoré sú v čase písania práce v štandardizačnom procese NISTu. Taktiež sa v tejto kapitole venujeme optimalizáciám šifrovacích schém.

Kapitola 4 sa primárne zameriava na použitie šifrovaní zachovávajúcich formát v praxi. V prvej časti uvádzame aktuálne dostupné implementácie, v druhej časti navrhneme vlastnú knižnicu v jazyku Python a v tretej časti porovnáme existujúce riešenia vrátane našej knižnice.

Vlastný prínos a výsledky

V práci sme predstavili koncept šifrovaní zachovávajúcich formát, poskytli sme vlastný dôkaz vzťahu dvoch známych kryptoanalytických modelov bezpečnosti a vlastný dôkaz korektnosti jednej metódy používanej pri konštrukcii šifrovaní zachovávajúcich formát. Túto metódu sme neskôr využili pri implementácii vlastnej knižnice pre jazyk Python.

Následne sme definovali nový model bezpečnosti, ktorý zodpovedá Thorpovej analýze pomocou matice pravdepodobností a odvodili jeho vzťahy s doteraz známymi modelmi používanými pri analýze šifrovaní zachovávajúcich formát. Následne sme tento model využili pri hľadaní ideálneho miešania kariet spĺňajúceho nami definované kritériá.

Ďalej sme sa zaoberali súčasným stavom šifrovaní zachovávajúcich formát a ich použitím v praxi. Na základe našich zistení sme implementovali knižnicu s podporou takéhoto šifrovania v jazyku Python, ktorá je v čase písania práce najrýchlejšou knižnicou pre tento jazyk a obsahuje najširšiu podporu rôznych formátov spomedzi nám dostupných súčasných knižníc.

Kapitola 1

Základné pojmy a konštrukcie

V úvode práce bola naznačená potreba šifrovania, ktoré má rovnaký formát vstupného (otvoreného) a výstupného (šifrového) textu. Takéto šifrovania zachovávajúce formát (format-preserving encryption) sú vhodné napríklad na šifrovanie osobných údajov s pevne daným formátom[1], znakových reťazcov konkrétnej dĺžky, 512-bytových diskových sektorov, a pod. Výhodou takýchto šifrovaní je ľahké nasadenie bez potreby veľkých zmien v existujúcich informačných systémoch, ktoré kontrolujú formát údajov. V tejto kapitole stručne predstavíme historický vývoj v danej oblasti, uvedieme formálne definície a modely súvisiace so šifrovaniami zachovávajúcich formát a následne uvedieme základné princípy využívané pri konštrukcii takýchto šifrovaní.

1.1 História

Informácie o historickom vývoji pochádzajú z [43] a z vlastného prieskumu.

V roku 1981, americký NBS (National Bureau of Standards, predchodca NISTu) publikoval v dokumente FIPS 74 (sekcia 8) [32] postup pre šifrovanie znakových reťazcov na znakové reťazce rovnakej dĺžky. Metóda bola založená na šifrovaní k-bitových zápisov jednotlivých znakov pripočítaním k-bitového náhodného kľúča (modulo 2^k), pričom náhodný kľúč bol generovaný použitím algoritmu DES v CFB (cipher feedback) móde[2]. Z dnešného pohľadu táto šifra nie je vôbec bezpečná; ak o dvoch šifrových textoch vieme, že boli zašifrované pomocou rovnakého kľúča, stačí nám tieto dva šifrové texty sčítať po znakoch (modulo 2^k), čím získame súčet po znakoch (modulo 2^k) otvo-

[1] napr. rodné čísla (RČ), evidenčné čísla vozidiel (ECV), telefónne čísla, čísla kreditných kariet (CCN), bankových účtov (PAN) a sociálnych poistiek (SSN), občianskych preukazov, poštové adresy a smerovacie čísla (PSČ)

[2] bitová reprezentácia znakov sa zašifrovala pomocou CFB módu a výsledok bol použitý ako náhodný kľúč pre zašifrovanie jednotlivých znakov

rených textov. A z tohto súčtu sa už ľahko dajú zrekonštruovať oba pôvodné texty za predpokladu, že nie sú príliš krátke alebo náhodné.

V roku 1997 Brightwell a Smith ako prví jasne opísali šifrovanie zachovávajúce formát pod názvom datatype-preserving encryption [8]. Termín format-preserving encryption ako prvý použil Spies v roku 2008 [45]. No už 6 rokov predtým bola známa práca Blacka a Rogawaya zaoberajúca sa (z dnešného pohľadu) celočíselným FPE [4].

V súčasnosti sa šifrovaniam zachovávajúcim formát venuje značná pozornosť. Bellare, Ristenpart, Rogaway a Stegers formalizovali problém FPE a popísali niektoré všeobecné riešenia [1] a Bellare, Rogaway a Spies poslali v roku 2010 NISTu návrh na štandardizáciu FPE algoritmu pod názvom FFX. Tento návrh obsahuje popis všeobecného algoritmu spolu s dvoma konkrétnymi módmi (FFX-A2 pre binárne reťazce a FFX-A10 pre dekadické reťazce). Tomuto algoritmu aj o ďalším návrhom na štandardizáciu sa viac venujeme v časti 3.1.

V priebehu posledných rokov sa objavili práce dávajúce do súvisu miešanie kariet a FPE a konštruujúce algoritmy pre FPE na základe týchto súvislostí [18, 20, 28, 29, 41]. Podrobnejšie sa touto témou zaoberáme v kapitole 2, kde uvádzame popis niektorých známych FPE algoritmov využívajúcich miešania kariet a tiež zaujímavý prístup k analýze bezpečnosti takýchto šifrovacích schém.

Ďalším rozvíjajúcim sa trendom sú snahy o vylepšenie bezpečnostných vlastností a efektivity FPE. Pre dosiahnutie týchto cieľov sa využívajú rôzne vylepšenia v generovaní pseudonáhodných čísel [18] a permutácií [12], prípadne aj možnosti dnešného hardvéru s podporou inštrukcií vhodných pre kryptografické účely [46]. Ukazuje sa, že tieto snahy sú viac či menej vhodné pre praktické použitie. Bližšie je táto problematika popísaná v časti 3.2.

1.2 FPE: šifrovania zachovávajúce formát

1.2.1 Definície a syntax

FPE je deterministické symetrické šifrovanie. Vo všeobecnej definícii (podľa [1]) je FPE na rozdiel od zvyčajných šifrovacích schém asociované so súborom domén $\{\mathcal{X}_N\}, N \in \mathcal{N}$, kde $\mathcal{X}_N$ je diel domény (v orig. slice), množina $\mathcal{N}$ je priestor formátov a celková doména FPE schémy je $\mathcal{X} = \bigcup_N \mathcal{X}_N$. Diely domény $\mathcal{X}_N$ nemusia byť disjunktné a formát N nemusí byť jednoznačne odvoditeľný zo samotného otvoreného/šifrového textu.

Pre ilustráciu na konkrétnom príklade si môžeme predstaviť, že pomocou FPE chceme šifrovať čísla, ktoré majú podobné vlastnosti ako naše rodné čísla (prvá časť z dátumu

narodenia, druhá časť je koncovka, celé číslo je deliteľné 11) [33], avšak povolíme koncovky rôznych dĺžok, konkrétne $3, 4, 5$. Celé rodné číslo teda bude mať dĺžku 9 až 11 číslic. Táto dĺžka určuje konkrétny formát rodného čísla, pričom množina formátov je $\mathcal{N} = \{9, 10, 11\}$, jednotlivé diely domény $\mathcal{X}_9, \mathcal{X}_{10}, \mathcal{X}_{11}$ predstavujú množinu rodných čísel príslušnej dĺžky a ich zjednotenie je celková doména $\mathcal{X}$ všetkých rodných čísel, ktoré môžeme pomocou tejto FPE schémy šifrovať a dešifrovať.

Ďalší príklad predstavuje šifrovanie prirodzených čísel, ktoré spĺňajú nejakú konkrétnu vlastnosť, napríklad deliteľnosť prvočíslom. Nech $\mathcal{X}_p$ je diel domény, ktorý obsahuje prirodzené čísla deliteľné prvočíslom p. Potom napr. číslo 42 patrí do troch dielov domény: $\mathcal{X}_2, \mathcal{X}_3, \mathcal{X}_7$. Pri šifrované čísla 42 nie je preto jasné, v ktorom dieli domény ho chceme šifrovať, preto je potrebné uviesť aj formát (deliteľnosť prvočíslom p), ktorým jednoznačne určíme množinu prípustných otvorených aj šifrových textov.

Vylepšenia (tweaks). Na rozdiel od väčšiny bežných šifrovacích schém používajú FPE schémy pri šifrovaní a dešifrovaní ďalší parameter (okrem vstupného textu a kľúča). Tento parameter nazývame vylepšenie (v orig. tweak), pričom tieto vylepšenia v istom zmysle pomenúvajú nezávislé šifry. Teda ak použijeme dve šifry s rôznymi vylepšeniami a ostatnými parametrami rovnakými, nemala by byť medzi ich výstupmi žiadna korelácia.

Použitie vylepšení ilustrujeme na nasledovnom príklade. Opäť použijeme rodné čísla, tentoraz so 6 miestnou časťou odvodenou z dátumu narodenia a 4 miestnou koncovkou. Uvažujme systém, v ktorom sú uložené rodné čísla v otvorenom tvare a chceli by sme ich zašifrovať použitím FPE. Tento systém sa však môže spoliehať na to, že z nešifrovaných rodných čísel vie určiť pohlavie aj dátum narodenia osôb. Ak by sme rodné čísla zašifrovali, tieto údaje by systém určoval chybne.

Ako riešenie by sme mohli šifrovanie aplikovať iba na štvormiestnu koncovku RČ[3], ale tým by sme rapídne zmenšili priestor správ[4]. Ak by sme však mali veľkú databázu RČ, mnohé koncovky by sa v nej viackrát opakovali a vďaka jednoznačnému šifrovaniu by útočník mohol pomocou malého počtu známych dvojíc rodných čísel a ich zašifrovanej podoby spraviť jednoznačné priradenie štvorciferných otvorených a šifrových koncoviek RČ a pomocou neho dešifrovať veľkú časť celej databázy.

Na zabránenie takémuto útoku je potrebné, aby dve RČ s rovnakými koncovkami mohli mať po zašifrovaní koncovky rôzne. Presnejšie povedané, ak vieme, ako sa zašifrovalo jedno RČ, nevieme na základe toho povedať nič o zašifrovaní iného RČ (hoci aj

[3]so zachovaním zvyšku po delení jedenástimi

[4]diely domény by boli štvorciferné čísla s rovnakými zvyškami po delení jedenástimi, teda v jednom diely domény by bolo približne $10^4/11 \approx 909$ prvkov, v celej doméne 10^4 prvkov

s rovnakou koncovkou). Toto sa dá dosiahnuť pridaním ďalšieho parametra (spomínaného vylepšenia) do FPE.

Vylepšenie môže byť verejne známy parameter, napríklad vo vyššie uvedenom príklade s koncovkami RČ by vylepšenie mohol byť dátum narodenia. Zároveň požadujeme, aby medzi vylepšeniami a výsledkami šifrovania/dešifrovania neboli žiadne korelácie, resp. aby útočník, ktorý má k dispozícii dvojice otvorených a šifrových textov pre FPE s vylepšením T_1, nevedel na základe nich zistiť žiadnu informáciu o podobe šifrových textov pri použití vylepšenia T_2.

Ako vidno na základe tejto úvahy, zavedením vylepšení môžeme zvýšiť bezpečnosť šifrovaní zachovávajúcich formát.

Definícia 1.2.1 (FPE šifrovanie)**.** *Nech $\mathcal{N}$ je priestor formátov, $\mathcal{K}$ je priestor kľúčov, $\mathcal{T}$ je priestor vylepšení, $\mathcal{X} = \bigcup_N \mathcal{X}_N$ je celková doména a $\perp \notin \mathcal{X}$ je prvok označujúci neplatný text. Šifrovanie zachovávajúce formát (FPE šifrovanie) je zobrazenie*

$$E : \mathcal{N} \times \mathcal{T} \times \mathcal{K} \times \mathcal{X} \to \mathcal{X} \cup \{\perp\}.$$

FPE šifrovanie E správy X vo formáte N s kľúčom K a vylepšením T označujeme $E(K, N, T, X) = E_K^{N,T}(X)$, pričom požadujeme, aby to, či $E_K^{N,T}(X) = \perp$ alebo nie, záviselo iba na parametroch N, X a nie na K, T a aby zobrazenia $E_K^{N,T} : \mathcal{X}_N \to \mathcal{X}_N$ boli bijektívne pre ľubovoľné N, T, K.

Definícia 1.2.2 (FPE dešifrovanie)**.** *Nech $E_K^{N,T}$ je FPE šifrovanie, potom inverzné zobrazenie $D : \mathcal{N} \times \mathcal{T} \times \mathcal{K} \times \mathcal{X} \to \mathcal{X} \cup \{\perp\}$ definované predpisom $D_K^{N,T}(Y) = D(K, N, T, Y) = X \Leftrightarrow E_K^{N,T}(X) = Y$ nazveme dešifrovanie.*

V týchto definíciách pre každé $N \in \mathcal{N}, T \in \mathcal{T}, K \in \mathcal{K}$ FPE šifrovanie $E_K^{N,T}$ vlastne označuje inú permutáciu množiny $\mathcal{X}_N$ a FPE dešifrovanie $D_K^{N,T}$ inverznú permutáciu k $E_K^{N,T}$.

Definícia 1.2.3 (FPE schéma)**.** *FPE schéma je usporiadaná množina*

$$\mathcal{E} = (E, D, \mathcal{N}, \mathcal{T}, \mathcal{K}, \mathcal{X}, \perp),$$

kde E je FPE šifrovanie, D je FPE dešifrovanie, $\mathcal{N}$ je priestor formátov, $\mathcal{T}$ je priestor vylepšení, $\mathcal{K}$ je priestor kľúčov, $\mathcal{X}$ je celková doména a $\perp \notin \mathcal{X}$ je prvok označujúci neplatný text.

Definícia 1.2.4 (Diel domény)**.** *N-tý diel domény nazveme množinu $\mathcal{X}_N$ definovanú ako*

$$\mathcal{X}_N = \left\{ X \in \mathcal{X}; \forall (K, T) \in \mathcal{K} \times \mathcal{T} : E_K^{N,T}(X) \neq \perp \right\}.$$

Navyše chceme, aby každý prvok z $\mathcal{X}$ bol zahrnutý v nejakom dieli domény a aby každý diel domény bola konečná množina.

1.2.2 Bezpečnostné pojmy a definície

Hry. Základný model pre definovanie bezpečnosti rôznych kryptografických konštrukcií využíva hry, ktoré hrá útočník proti algoritmu hry samotnej. Najprv vo všeobecnosti uveďme, ako taká hra vyzerá, potom sa pozrieme na konkrétne príklady hier, ktoré sa používajú pri definovaní a dokazovaní bezpečnosti FPE schém.

Na začiatku sa samotná hra G inicializuje, teda nastaví počiatočné hodnoty svojich premenných, vygeneruje kľúče, permutácie a ďalšie parametre potrebné počas behu samotnej hry potrebovať. Ak nie je povedané inak, neinicializované premenné majú implicitnú hodnotu ekvivalentnú 0 alebo False, teda booleovské premenné sú False, číselné premenné 0, reťazce sú prázdne, atď.

Po tejto inicializácii sa spustí útočníkov algoritmus $\mathcal{A}$, ktorý môže na vstupe dostať nejaké hodnoty z inicializačnej časti hry G (napríklad dvojice šifrového a otvoreného textu). Útočník $\mathcal{A}$ môže volať jednotlivé procedúry a funkcie hry (okrem inicializácie a finalizácie, o ktorej bude reč neskôr), a po skončení svojho behu vypíše niečo na výstup. Ak hra obsahuje ešte finalizáciu, tá spracuje tento útočníkov výstup a výsledok je výstup hry; ak hra finalizáciu nemá, výstup hry je priamo útočníkov výstup. Zároveň jednotlivé procedúry a funkcie hry môžu volať nejaké útočníkove metódy, syntax je $\mathcal{A}$(`fcia[, premenne]`). Metódy útočníka sú volané vždy na novej inštancii, ktoré medzi sebou nezdieľajú žiadne stavové informácie. Udalosť, pri ktorej hra G s útočníkom $\mathcal{A}$ vráti y, budeme značiť $G^{\mathcal{A}} \Rightarrow y$. Volanie útočníkových metód sa využíva napr. v hrách na str. 11 a 12, v ktorých útočník poskytuje hrám pravdepodobnostnú distribúciu pre voľbu parametrov.

Bezpečnosť. V práci [1] autori rozšírili štandardnú PRP (pseudorandom permutation) hru pre použitie s nami zadefinovanými FPE, a okrem toho použili ešte aj niekoľko slabších verzií hier (SPI, MP, MR), ktoré sú opísané nižšie spolu so základnými výsledkami a vzťahmi medzi nimi. V ďalšom texte $\mathcal{E}$ predstavuje FPE schému $(E, D, \mathcal{N}, \mathcal{T}, \mathcal{K}, \mathcal{X}, \perp)$, E predstavuje FPE šifrovanie $E : \mathcal{N} \times \mathcal{T} \times \mathcal{K} \times \mathcal{X} \to \mathcal{X} \cup \perp$, pričom predpokladáme korektnosť dopytov, teda že pre každý dopyt (N, T, K, X) platí $N \in \mathcal{N}, T \in \mathcal{T}, K \in \mathcal{K}, \mathcal{X} \in \mathcal{X}_N$. Pokiaľ nie je uvedené inak, náhodným výberom myslíme uniformný výber (označujeme ho symbolom $\xleftarrow{\$}$) a ľubovoľným útočníkom ľubovoľného PPT[5] útočníka s parametrom $\log |\mathcal{K}|$. Funkcia $\mathrm{negl} : \mathbb{N}_0 \to \mathbb{R}_0^+$ predstavuje (asymptoticky) zanedbateľnú funkciu, teda funkciu, ktorá je od určitého miesta $n_0 \in \mathbb{N}_0$ menšia ako $1/p(n)$ pre všetky $n > n_0$, kde p je ľubovoľný kladný polynóm.

[5] pravdepodobnostný polynomiálny čas

PRP bezpečnosť. V hre $\mathrm{PRP}_{\mathcal{E}}$ ide o schopnosť útočníka rozlíšiť FPE šifrovanie E od náhodnej permutácie množiny $\mathcal{X}_N$. Hra prebieha nasledovne:

1. **Inicializácia.** Zvolíme náhodne jeden bit b z množiny $\{0,1\}$ a kľúč K z množiny $\mathcal{K}$. Pre každú dvojicu $(N,T) \in \mathcal{N} \times \mathcal{T}$ náhodne zvolíme permutáciu $\pi_{N,T}$ množiny $\mathcal{X}_N$.

2. **Hra.** Útočník môže volať ľubovoľne veľa krát s ľubovoľným korektným vstupom funkciu $Enc(N,T,X)$ definovanú nasledovne:

$$Enc(N,T,X) = \begin{cases} E_K^{N,T}(X), & \text{ak } b = 1, \\ \pi_{N,T}(X), & \text{ak } b = 0. \end{cases}$$

3. **Finalizácia.** Po poslednom volaní $Enc(N,T,X)$ sa útočník pokúsi uhádnuť hodnotu bitu b a hádanú hodnotu vypíše na výstup ako b'. Finalizácia vráti na výstupe booleovskú hodnotu $(b = b')$.

Pre posúdenie útočníkovej úspešnosti sa používa pojem výhody definovaný pre $\mathrm{PRP}_{\mathcal{E}}$ hru nasledovne:

$$\mathbf{Adv}_{\mathcal{E}}^{\mathrm{PRP}}(\mathcal{A}) = 2 \cdot \Pr\left[\mathrm{PRP}_{\mathcal{E}}^{\mathcal{A}} \Rightarrow \mathrm{True}\right] - 1.$$

V ideálnom prípade útočník nebude schopný rozlíšiť, či funkcia $Enc(N,T,X)$ vracia hodnoty zašifrované pomocou E alebo je to náhodná permutácia a teda mu nezostane nič iné, ako náhodne zvoliť b' na výstupe. Vtedy v $\mathrm{PRP}_{\mathcal{E}}$ hre uspeje s pravdepodobnosťou $1/2$ a teda jeho výhoda bude nulová.

Definícia 1.2.5 (PRP bezpečnosť.)**.** *Nech $\mathcal{E}$ je FPE schéma a $\mathcal{A}$ je ľubovoľný PRP útočník. Hovoríme, že schéma $\mathcal{E}$ je bezpečná vzhľadom na hru $PRP_{\mathcal{E}}$ (resp. schéma $\mathcal{E}$ je PRP-bezpečná) práve vtedy, keď platí:*

$$\mathbf{Adv}_{\mathcal{E}}^{PRP}(\mathcal{A}) = \mathrm{negl}(\log|\mathcal{K}|)$$

SPI bezpečnosť. V hre $\mathrm{SPI}_{\mathcal{E}}$ (single-point indistinguishability) sa zameriavame na útočníkovu schopnosť rozlišovať medzi jednou správou zašifrovanou pomocou E a náhodne zvolenou správou z príslušného dielu domény. Hra modeluje adaptívny útok, teda útočník má po obdržaní testovanej správy prístup k šifrovaciemu orákulu. Priebeh hry je nasledovný:

1. **Inicializácia.** Zvolíme náhodne jeden bit b z množiny $\{0,1\}$ a kľúč K z množiny $\mathcal{K}$ a množinu položených dopytov S inicializujeme na prázdnu množinu.

2. **Hra.** Útočník môže raz v priebehu hry zavolať funkciu $Test(N,T,X)$ a ľubovoľne veľakrát volať funkciu $Enc(N,T,X)$, pričom

$$Test(N,T,X) = \begin{cases} E_K^{N,T}(X), & \text{ak } (N,T,X) \notin S \wedge b = 1; \quad S \leftarrow S \cup (N,T,X), \\ X_r, X_r \xleftarrow{\$} \mathcal{X}_N & \text{ak } (N,T,X) \notin S \wedge b = 0; \quad S \leftarrow S \cup (N,T,X), \\ \perp, & \text{ak } (N,T,X) \in S, \end{cases}$$

$$Enc(N,T,X) = \begin{cases} E_K^{N,T}(X), & \text{ak } (N,T,X) \notin S; \quad S \leftarrow S \cup (N,T,X), \\ \perp, & \text{ak } (N,T,X) \in S, \end{cases}$$

   kde X_r predstavuje náhodne vybratú premennú z množiny $\mathcal{X}_N$.

3. **Finalizácia.** Po poslednom volaní $Enc(N,T,X)$ sa útočník pokúsi uhádnuť hodnotu bitu b a hádanú hodnotu vypíše na výstup ako b'. Finalizácia vráti na výstupe booleovskú hodnotu $(b = b')$.

Útočníkova výhoda pre hru $\mathrm{SPI}_{\mathcal{E}}$ je

$$\mathbf{Adv}_{\mathcal{E}}^{\mathrm{SPI}}(\mathcal{A}) = 2 \cdot \Pr\left[\mathrm{SPI}_{\mathcal{E}}^{\mathcal{A}} \Rightarrow \mathrm{True}\right] - 1.$$

Nech $\mathrm{SPI1}_{\mathcal{E}}$ je $\mathrm{SPI}_{\mathcal{E}}$ hra, v ktorej sa v inicializácii zvolilo $b = 1$ a nech $\mathrm{SPI0}_{\mathcal{E}}$ je $\mathrm{SPI}_{\mathcal{E}}$ hra, v ktorej sa v inicializácii zvolilo $b = 0$. Pre výhodu útočníka potom možno odvodiť:

$$\mathbf{Adv}_{\mathcal{E}}^{\mathrm{SPI}}(\mathcal{A}) = \Pr\left[\mathrm{SPI1}_{\mathcal{E}}^{\mathcal{A}} \Rightarrow \mathrm{True}\right] - \Pr\left[\mathrm{SPI0}_{\mathcal{E}}^{\mathcal{A}} \Rightarrow \mathrm{False}\right].$$

Pri hre $\mathrm{SPI}_{\mathcal{E}}$ existuje dolné ohraničenie útočníkovej výhody závisiace od počtu volaní $Enc(N,T,X)$: nech je $b = 0$ a útočník urobí q volaní $Enc(N,T,X)$ s rovnakými N,T. Potom pravdepodobnosť, že niektoré z týchto volaní vráti tú istú správu ako $Test(N,T,X)$, je $q/|\mathcal{X}_N|$. Ak útočník zvolí také N, pre ktoré je $|\mathcal{X}_N|$ minimálna (označme túto veľkosť M), tak pravdepodobnosť úspechu je aspoň q/M. Z tohto dôvodu nezavádzame definíciu SPI bezpečnosti.

V práci [1] sú uvedené vzťahy medzi hrami $\mathrm{PRP}_{\mathcal{E}}$ a $SPI_{\mathcal{E}}$. Tieto modely sa navzájom implikujú, avšak ani v jednom prípade nie je redukcia tesná. Pri redukcii jedného modelu na druhý vieme ohraničiť výhodu útočníka v jednej hre pomocou výhody útočníka v hre druhej a nejakého aditívneho faktora, čím strácame istú mieru bezpečnosti. Výhoda PRP útočníka implikuje výhodu SPI útočníka s aditívnou stratou q_{SPI}/M, kde q_{SPI} je počet volaní $Enc(N,T,X)$ SPI útočníkom a $M = min_{N \in \mathcal{N}}|\mathcal{X}_N|$. Výhoda SPI útočníka podobne implikuje výhodu PRP útočníka, pričom $\mathbf{Adv}_{\mathcal{E}}^{\mathrm{PRP}}(\mathcal{A}) \leq q_{\mathrm{PRP}} \cdot \mathbf{Adv}_{\mathcal{E}}^{\mathrm{SPI}}(\mathcal{B}) + q_{\mathrm{PRP}}^2/M$, kde q_{PRP} je počet dotazov na $Enc(N,T,X)$ PRP útočníka $\mathcal{A}$ a skonštruovaný SPI útočník $\mathcal{B}$ vykoná $q_{\mathrm{PRP}} - 1$ dotazov na $Enc(N,T,X)$.

MR bezpečnosť. V hre $\mathrm{MR}_{\mathcal{E}}$ (message recovery) ide o schopnosť útočníka odhaliť pôvodnú správu zo šifrového textu, pričom má k dispozícii šifrovacie orákulum a distribúciu správ, formátov a vylepšení si môže zvoliť hocijakú. Hrá sa podľa nasledujúcich pravidiel:

1. **Inicializácia.** Zvolíme náhodne kľúč K z množiny $\mathcal{K}$ a trojicu (N_0, T_0, X_0) z $\mathcal{N} \times \mathcal{T} \times \mathcal{X}$ s ohľadom na pravdepodobnostnú distribúciu $\mathcal{A}(dist)$ poskytnutú útočníkom. Zašifrujeme danú správu, teda $Y_0 = E_K^{N_0,T_0}(X_0)$ a útočníkovi vrátime dvojicu (N_0, T_0), aby poznal všetky parametre okrem správy a kľúča.

2. **Hra.** Útočník na začiatku hry raz zavolá funkciu $Test()$, ktorá vráti Y_0 a následne môže ľubovoľne veľakrát volať funkciu $Enc(N, T, X) = E_K^{N,T}(X)$.

3. **Finalizácia.** Po poslednom volaní $Enc(N, T, X)$ sa útočník pokúsi uhádnuť pôvodnú správu X_0 a hádanú hodnotu vypíše na výstup ako X'. Finalizácia vráti na výstupe booleovskú hodnotu $(X' = X_0)$.

S prístupom k $Enc(N, T, X)$ však môže útočník jednoducho skúšať postupne šifrovať jednotlivé otvorené texty a porovnávať, kedy je zašifrovaný výsledok rovnaký ako šifrový text Y_0. Toto sa mu určite vždy podarí, keďže šifrovanie je deterministické. Otázkou však ostáva, či existuje aj nejaký lepší postup pre útočníka, preto zavedieme formalizmus, v ktorom bude útok úplným preberaním všetkých otvorených textov najlepší z možných útokov.

Hra $\mathrm{MR}_{\mathcal{E}}$ bude poskytovať ešte jednu funkciu $Eq(X) = (X = X_0)$, ktorú však útočník nebude môcť používať. Predstavme si, že namiesto útočníka $\mathcal{A}$ použijeme simulátor $\mathcal{S}$, pričom simulátor nebude môcť používať funkcie $Test()$ a $Enc(N, T, X)$, bude môcť použiť iba funkciu $Eq(x)$ najviac toľkokrát, koľkokrát pôvodný útočník $\mathcal{A}$ použil funkciu $Enc(N, T, X)$, $\mathcal{S}(dist) = \mathcal{A}(dist)$ a inicializácia simulátora $\mathcal{S}$ bude prebiehať rovnako ako inicializácia útočníka $\mathcal{A}$. Potom výhoda útočníka bude

$$\mathbf{Adv}_{\mathcal{E}}^{\mathrm{MR}}(\mathcal{A}) = \Pr\left[\mathrm{MR}_{\mathcal{E}}^{\mathcal{A}} \Rightarrow \mathrm{True}\right] - p_{\mathcal{A}},$$

kde $p_{\mathcal{A}} = \max_S \Pr\left[\mathrm{MR}_{\mathcal{E}}^{\mathcal{S}} \Rightarrow \mathrm{True}\right]$ je maximum pravdepodobností úspechu cez všetky možné simulátory $\mathcal{S}$. Inak povedané, zaujíma nás, o koľko lepší môže byť útočník, ako keby sa iba pýtal na jednotlivé správy, či to nie je pôvodná správa.

Je zjavné, aký simulátor má najväčšiu šancu na úspech. Keďže pravdepodobnostnú distribúciu volí útočník/simulátor a navyše poznajú dvojicu (N_0, T_0), tak sa bude postupne pýtať na hodnoty X_i zostupne zoradené podľa pravdepodobnosti (uvažujúc dvojicu (N_0, T_0)). Nech q je počet volaní $Enc(N, T, X)$ útočníkom $\mathcal{A}$, potom sa simulátor $\mathcal{S}$ spýta na prvých q hodnôt X_i. Ak ani jedna z týchto hodnôt nie je pôvodná

tajná správa, tak simulátor vypíše na výstup X_{q+1} a skončí. Toto je najlepší simulátor a výhoda útočníka je rozdiel, o koľko môže byť útočník lepší ako takýto simulátor.

Definícia 1.2.6 (MR bezpečnosť.)**.** *Nech $\mathcal{E}$ je FPE schéma a $\mathcal{A}$ je ľubovoľný MR útočník. Hovoríme, že schéma $\mathcal{E}$ je bezpečná vzhľadom na hru $MR_{\mathcal{E}}$ (resp. schéma $\mathcal{E}$ je MR-bezpečná) práve vtedy, keď platí:*

$$\mathbf{Adv}_{\mathcal{E}}^{MR}(\mathcal{A}) = \mathrm{negl}(\log|\mathcal{K}|)$$

MP bezpečnosť. V hre $\mathrm{MP}_{\mathcal{E}}$ (message privacy) sa zameriavame na útočníkovu schopnosť odhaliť nejakú informáciu o otvorenom texte pomocou šifrového textu. Formálne, o schopnosť útočníka vypočítať zo šifrového textu hodnotu nejakej funkcie nad otvoreným textom.

Hra $\mathrm{MP}_{\mathcal{E}}$ je veľmi podobná hre $\mathrm{MR}_{\mathcal{E}}$, akurát tentoraz útočník navyše poskytuje nekonštantnú funkciu $f(X)$ počítanú nad otvoreným textom. Pravidlá hry sú rovnaké ako pri $\mathrm{MP}_{\mathcal{E}}$, jediné dva rozdiely sú na konci útočníkovho algoritmu a vo finalizácii. Útočník vypíše nejakú hodnotu Z, ktorú spočítal pomocou šifrového textu a $Enc(N,T,X)$ dopytov, finalizácia použije útočníkom poskytnutú funkciu $f(X)$, vypočíta pomocou nej hodnotu $Z_0 = f(X_0)$ a na výstupe vráti booleovskú hodnotu $(Z = Z_0)$.

Obdobne ako pri $\mathrm{MR}_{\mathcal{E}}$ porovnávame útočníka $\mathcal{A}$ so simulátorom $\mathcal{S}$, ktorý nesmie použiť $Test()$ ani $Enc(N,T,X)$, má rovnakú distribúciu (N,T,X) aj funkciu $f(X)$ ako $\mathcal{A}$ a použije najviac toľko volaní $Eq(X)$, koľko použil útočník volaní $Enc(N,T,X)$. Potom výhoda útočníka bude

$$\mathbf{Adv}_{\mathcal{E}}^{\mathrm{MP}}(\mathcal{A}) = \Pr\left[\mathrm{MP}_{\mathcal{E}}^{\mathcal{A}} \Rightarrow \mathrm{True}\right] - p_{\mathcal{A}},$$

kde $p_{\mathcal{A}} = \max_S \Pr\left[\mathrm{MP}_{\mathcal{E}}^{\mathcal{S}} \Rightarrow \mathrm{True}\right]$ je maximum pravdepodobností úspechu cez všetky možné simulátory $\mathcal{S}$.

Definícia 1.2.7 (MP bezpečnosť.)**.** *Nech $\mathcal{E}$ je FPE schéma a $\mathcal{A}$ je ľubovoľný MP útočník. Hovoríme, že schéma $\mathcal{E}$ je bezpečná vzhľadom na hru $MP_{\mathcal{E}}$ (resp. schéma $\mathcal{E}$ je MP-bezpečná) práve vtedy, keď platí:*

$$\mathbf{Adv}_{\mathcal{E}}^{MP}(\mathcal{A}) = \mathrm{negl}(\log|\mathcal{K}|)$$

Vzťahy medzi modelmi. Stojí za povšimnutie, že $\mathrm{MR}_{\mathcal{E}}$ je iba špeciálny prípad $\mathrm{MP}_{\mathcal{E}}$, v ktorom je funkcia $f(X) = X$, teda identita. Zjavne teda výhoda $\mathrm{MP}_{\mathcal{E}}$ útočníka pre nejakú FPE schému zhora ohraničuje výhodu $\mathrm{MR}_{\mathcal{E}}$ útočníka pre túto schému. V [1] je dokázané, že výhoda $\mathrm{SPI}_{\mathcal{E}}$ útočníka zhora ohraničuje výhodu $\mathrm{MP}_{\mathcal{E}}$ útočníka. Naopak to neplatí, teda výhoda $\mathrm{MP}_{\mathcal{E}}$ útočníka neohraničuje zhora výhodu $\mathrm{SPI}_{\mathcal{E}}$ útočníka a ani $\mathrm{MR}_{\mathcal{E}}$ neimplikuje $\mathrm{MP}_{\mathcal{E}}$. Hlavné myšlienky dôkazov posledných dvoch tvrdení sú uvedené v práci [1], vlastný dôkaz posledného tvrdenia uvádzame nižšie.

Veta 1.2.1. *$MR_{\mathcal{E}} \not\Rightarrow MP_{\mathcal{E}}$ Existuje schéma, ktorá je MR-bezpečná, ale nie je MP-bezpečná.*

Dôkaz. Nech $\mathcal{E} = (E, D, \{n\}, \mathcal{T}, \mathcal{K}, \{0,1\}^n, \perp)$ je MR-bezpečná FPE schéma, teda schéma, pre ktorú platí: $\forall \mathcal{A} : \mathbf{Adv}^{\mathrm{MR}}_{\mathcal{E}}(\mathcal{A}) = \mathrm{negl}(\log|\mathcal{K}|)$. Skonštruujme pomocou nej FPE schému $\mathcal{E}' = (E', D', \{n+1\}, \mathcal{T}, \mathcal{K}, \{0,1\}^{n+1}, \perp)$, ktorá bude prezrádzať informáciu o jednom bite nasledovne:

$$\begin{aligned} E'^{N,T}_K(x_0 x_1 \dots x_n) &= E^{N,T}_K(x_0 x_1 \dots x_{n-1}) \,||\, x_n \\ D'^{N,T}_K(y_0 y_1 \dots y_n) &= D^{N,T}_K(y_0 y_1 \dots y_{n-1}) \,||\, y_n \end{aligned}$$

Je očividné, že pre takúto schému existuje útočník $\mathcal{B}$, ktorý v hre $\mathrm{MP}_{\mathcal{E}'}$ vždy uspeje. Stačí, aby jeho funkcia $\mathcal{B}(f, x_0 \dots x_n) = x_n$, a hneď po zavolaní funkcie $Test()$ vypísal na výstup hodnotu $Z = f(Y) = f(y_0 \dots y_n) = y_n = x_n$ a skončil. Zároveň najlepší simulátor $\mathcal{S}$ nesmie ani raz použiť funkciu $Eq(X)$ a musí hneď vypísať hodnotu 0 alebo 1 a skončiť. Simulátor $\mathcal{S}$ teda uspeje iba s pravdepodobnosťou $1/2$ a výhoda najlepšieho simulátora je $\mathbf{Adv}^{\mathrm{MP}}_{\mathcal{E}'}(\mathcal{B}) = 1/2$.

Ešte potrebujeme ukázať, že schéma $\mathcal{E}'$ je bezpečná v hre MR. Predpokladajme, že nie je, teda že existuje taký útočník $\mathcal{C}$, pre ktorého je výhoda $\mathbf{Adv}^{MR}_{\mathcal{E}'}(\mathcal{C}) > \mathrm{negl}(\log|\mathcal{K}|)$ pre ľubovoľnú zanedbateľnú funkciu negl. Potom by pomocou neho dokázal útočník $\mathcal{A}$ uspieť v hre $\mathrm{MR}_{\mathcal{E}}$ tým, že by simuloval hru $\mathrm{MR}_{\mathcal{E}'}$ s útočníkom $\mathcal{C}$.

Všetky volania funkcie $Enc(N, T, x_0 \dots x_n)$ útočníkom $\mathcal{C}$ v hre $\mathrm{MR}_{\mathcal{E}'}$ pretransformuje[6] útočník $\mathcal{A}$ na volania funkcie $Enc(N, T, x_0 \dots x_{n-1})$ v hre $\mathrm{MR}_{\mathcal{E}}$. Pri volaní funkcie $Test()$ vráti $\mathcal{A}$ útočníkovi $\mathcal{C}$ náhodne zvolené hodnotu $Y||0$ alebo $Y||1$. Pri finalizácii, keď $\mathcal{C}$ skončí a vypíše $x'_0 \dots x'_n$, skončí aj $\mathcal{A}$ a vypíše $x'_0 \dots x'_{n-1}$. Keď $\mathcal{C}$ uspeje, uspeje aj $\mathcal{A}$, pretože $E'^{N,T}_K(x'_0 \dots x'_n) = E^{N,T}_K(x'_0 \dots x'_{n-1})||x'_n = Y||x'_n$.

To by bol spor s MR bezpečnosťou schémy $\mathcal{E}$, preto takýto útočník $\mathcal{C}$ neexistuje a schéma $\mathcal{E}'$ je v hre $\mathrm{MR}_{\mathcal{E}'}$ bezpečná, no zároveň nie je bezpečná v hre $\mathrm{MP}_{\mathcal{E}'}$ a teda MR bezpečnosť neimplikuje MP bezpečnosť. □

Pri $\mathrm{SPI}_{\mathcal{E}}$ hre boli uvedené jej vzťahy s $\mathrm{PRP}_{\mathcal{E}}$ hrou, ktorých výsledkom je, že tieto dva bezpečnostné modely sa implikujú navzájom, avšak táto redukcia nie je tesná. Graficky sa vzťahy medzi jednotlivými modelmi dajú znázorniť nasledovne, pričom plné šípky predstavujú tesné redukcie[7] a čiarkované šípky predstavujú redukcie, ktoré nie sú tesné[8]:

[6] odstráni posledný bit x_n, zašifruje a pripojí posledný bit x_n

[7] horné ohraničenie výhody útočníka v jednom modeli pomocou výhody útočníka v druhom modeli

[8] horné ohraničenie výhody útočníka v jednom modeli pomocou výhody útočníka v druhom modeli a aditívneho člena

$$\mathrm{PRP}_{\mathcal{E}} \rightleftarrows \mathrm{SPI}_{\mathcal{E}} \rightleftarrows \mathrm{MP}_{\mathcal{E}} \rightleftarrows \mathrm{MR}_{\mathcal{E}}$$

Variácie bezpečnostných modelov. V jednotlivých hrách tak, ako boli prezentované vyššie, skúmame CPA (choosen plaintext attack) bezpečnosť FPE schém, teda útočník $\mathcal{A}$ má prístup k šifrovaciemu orákulu, avšak nie k dešifrovaciemu. Ak by sme chceli zosilniť bezpečnostné požiadavky o úroveň vyššie na odolnosť voči CCA útokom (choosen ciphertext attack), pridali by sme do hier dešifrovaciu funkciu $Dec(N, T, Y)$, avšak s obmedzením, že po použití na Y_0 (pri $\mathrm{MR}_{\mathcal{E}}$ a $\mathrm{MP}_{\mathcal{E}}$) vráti $\perp$. Takto modifikované hry budeme označovať prefixom „d“, teda dPRP, dSPI, dMP a dMR. Poznamenajme, že takto upravené CCA modely sú ekvivalentné pôvodným CPA modelom, nakoľko útočníkovi dovoľujeme vykonať ľubovoľné množstvo dopytov na funkciu $Enc(N, T, X)$, môže si útočník zistiť zobrazenie každého prvku a následne si môže funkciu $Dec(N, T, Y)$ vypočítať sám.

Hry možno použiť aj pre neadaptívneho útočníka, ktorý by sa v tomto prípade na začiatku behu rozhodol, aké metódy hry a v akom počte zavolá a podľa toho by sa správal. Takéto hry budeme označovať prefixom „n“, teda nPRP, nSPI, nMP a nMR. Vzťahy medzi modelmi v tomto prípade ostávajú rovnaké ako v prípade adaptívneho útočníka [1].

V niektorých prípadoch je užitočné skúmať aj bezpečnosť schém s obmedzeným prístupom útočníka k funkcii $Enc(N, T, X)$. Pri väčšine praktických aplikácií skúmaných šifrovacích schém bude mať útočník prístup k limitovanému počtu dvojíc otvoreného a šifrového textu. Zavedieme preto ešte jednu variáciu uvedených modelov, v ktorých obmedzíme počet útočníkových dopytov na funkciu $Enc(N, T, X)$.

Definícia 1.2.8. *Nech $\mathcal{E}$ je FPE schéma, G je niektorá z hier PRP, SPI, MR, MP a $\mathcal{A}$ je ľubovoľný útočník, ktorý vykoná v hre G najviac q dopytov na funkciu $Enc(N, T, X)$. Potom výhodu útočníka $\mathcal{A}$ v hre G budeme označovať $\mathbf{Adv}^G_{\mathcal{E}}(\mathcal{A}, q)$ a schému $\mathcal{E}$ nazveme G-bezpečnou s použitím najviac q dopytov, práve vtedy, keď platí:*

$$\mathbf{Adv}^G_{\mathcal{E}}(\mathcal{A}, q) = \mathrm{negl}(\log |\mathcal{K}|)$$

Na tomto mieste poznamenajme, že ak je schéma $\mathcal{E}$ G-bezpečná s použitím najviac q dopytov a $q \geq |X|$, tak schéma $\mathcal{E}$ je zároveň aj G-bezpečná pre $G \in \{\mathrm{PRP}, \mathrm{MR}, \mathrm{MP}\}$.

1.2.3 Kategorizácia FPE

Nech $S = |\mathcal{X}|$ je veľkosť domény, teda priestoru správ. Vhodnosť rôznych metód na konštrukciu FPE závisí od očakávanej veľkosti domény, pre ktorú chceme dané

FPE používať. Napríklad pre malé domény vieme dopredu určiť náhodnú permutáciu celej domény a potom pri šifrovaní/dešifrovaní už len vypísať príslušný obraz/vzor v predpočítanej náhodnej permutácii. Tento prístup si však vyžaduje čas $O(S)$ na inicializáciu FPE, čo pre veľké S už je príliš veľa, napr. pre rodné čísla je $S \approx 365 * 100 * 10000 \approx 2^{32}$, pre 16-ciferné čísla v desiatkovej sústave je $N = 10^{16} \approx 2^{53}$, atď.

FPE schémy môžeme rozdeliť na tri druhy v závislosti od veľkosti domény a priestoru správ, ktoré môžeme pomocou nich šifrovať [43]. Toto rozdelenie je uvedené v tabuľke 1.1.

Druh	**Veľkosť**	**Priestor správ**	**Použitie**
tiny-space FPE	$S \leq 2^{10}$	$[S]$	žiadne praktické
small-space FPE	$S \leq 2^{128}$	Σ^n	PAN, SSN, CCN
large-space FPE	$S \geq 2^{128}$	$\{0,1\}^n$	diskové sektory

Tabuľka 1.1: Rozdelenie FPE podľa veľkosti priestoru správ [43]

Tiny-space FPE. Pre Tiny-space FPE je priestor správ natoľko malý, že je efektívne stráviť aj $O(S)$ času na inicializáciu, prípadne samotné šifrovanie/dešifrovanie jednotlivých správ. V praktických aplikáciách sa však FPE tejto kategórie nepoužívajú, hoci vieme pre ne vymyslieť potenciálne aplikačné použitie typu šifrovanie dní v roku ($S = 365$, resp. 366), šifrovanie posledného štvorčíslia RČ a podobne. Pre túto kategóriu sú použiteľné aj algoritmy pre small-space FPE. Príklady schém vhodných pre kategóriu Tiny-space FPE sú napríklad Knuthova permutácia alebo Prefixová šifra, ktoré sú PRP-bezpečné [4, 13, 26].

Knuthova permutácia [13, 26] vznikne zamiešaním usporiadanej množiny obsahujúcej prvky $[S]$ náhodným výberom prvku i spomedzi prvých j prvkov a výmenou prvkov na pozíciách i, j pre $j \in \{S-1, \ldots, 1\}$, ako je uvedené vo výpise 1.1

V Prefixovej šifre sa permutácia množiny $[S]$ určí pomocou vhodnej blokovej šifry E a kľúča K. Najprv sa vypočíta usporiadaná množina $I = (E_K(0), E_K(1), \ldots, E_K(S-1))$. Prvky tejto množiny následne zobrazíme na prvky množiny $[S]$ podľa ich veľkosti[9] Takto upravená usporiadaná množina I určuje permutáciu množiny $[S]$ a teda aj samotné šifrovanie. Ak je použitá bloková šifra PRP-bezpečná, potom je aj Prefixová šifra PRP-bezpečná [4].

[9] poradie v utriedenej postupnosti $\{E_K(i)\}_{i=0}^{S-1}$

```
def knuth(c): # c je zoznam prvkov
  for j in range(len(c)-1,0,-1)
    i = random(0,j)
    c[i], c[j] = c[j], c[i]
```

Výpis 1.1: Knuthova permutácia

Small-space FPE. FPE pre túto kategóriu je väčšinou FPE schéma postavená nad vhodnou blokovou šifrou, pričom hranica pre veľkosť domény je 2^w, kde w je veľkosť bloku blokovej šifry. Pre tieto veľkosti domény obsahuje priestor správ najviac toľko prvkov, koľko je možných blokov v danej blokovej šifre. V dnešnej dobe sa najčastejšie používa ako bloková šifra NISTom štandardizovaný AES [37]. Pre AES je $w = 128$, odkiaľ je odvodená hranica pre Small-space FPE.

Techniky pre Small-space FPE sa dajú použiť aj pre Tiny-space FPE. Na rozdiel od schém určených výlučne pre Tiny-space FPE sú síce efektívnejšie, no nie vždy majú tak silnú bezpečnosť (napr. PRP bezpečnosť). Väčšinou sú schémy pre Small-space FPE PRP-bezpečné s použitím najviac q dopytov pre $q < |\mathcal{X}|$, prípadne sú SPI-bezpečné. Práve vyššia miera bezpečnosti je dôvod, prečo sa pri tejto kategorizácii uvádza aj kategória pre Tiny-space FPE.

Väčšina algoritmov pre túto kategóriu využíva klasické, nevyvážené alebo alternujúce Feistelovské siete (FFX [3], FFSEM [44]) a miešania kariet ([20], [28]), aj keď nájdu sa aj príklady iných typov konštrukcií (FIPS74 [32]). Prehľad najznámejších (aj historických) schém kategórie Small-space FPE je uvedený v [43].

Large-space FPE. Pre túto kategóriu predpokladáme, že priestor správ tvoria binárne reťazce aspoň tak dlhé ako je jeden blok blokovej šifry, teda FPE schémy pre túto kategóriu sú vlastne schémy pre šifrovanie veľkých blokov binárnych reťazcov. Zvyčajne algoritmy pre túto kategóriu využívajú kombináciu hešovania a šifrovania, napríklad HEH (hash-encrypt-hash), prípadne šifrovania a miešania, napríklad EME (encrypt-mix-encrypt). Prehľad najznámejších schém kategórie Large-space FPE je uvedený v [43].

V tejto práci sa nebudeme zaoberať kategóriou Large-space FPE, keďže je to vlastne iný druh šifrovania. Obvyklé použitie FPE schém patrí do kategórie Small-space FPE, preto sa na túto kategóriu budeme zameriavať primárne.

1.3 Konštrukcia FPE

V tejto časti sú uvedené dva základné prístupy používané pri konštrukcii FPE schém nad ľubovoľnými doménami pomocou FPE schém nad nejakými bežnými doménami, napr. prirodzenými číslami [S] alebo binárnymi reťazcami konkrétnej dĺžky, napr. 128 bitov.

1.3.1 Cyklická prechádzka

Predpokladajme, že máme FPE schému pre šifrovanie 20-bitových prirodzených čísel (rozsah $0 \dots 1048575$) a chceli by sme pomocou nej zašifrovať nejaké 6-ciferné číslo x v desiatkovej sústave (rozsah $0 \dots 999999$). Triviálne riešenie – zapíšeme x v dvojkovej sústave a zašifrujeme ako 20-bitové binárne číslo – nemusí vždy fungovať, ako výsledok môžeme dostať binárne číslo, na zápis ktorého potrebujeme v desiatkovej sústave 7 číslic.

Prístup cyklickej prechádzky (cycle walking) má ambíciu riešiť tento problém opakovaným šifrovaním nevyhovujúceho výsledku dovtedy, kým nedostaneme výsledok vyhovujúci. Formálne cyklickú prechádzku a jej korektnosť zavedieme v nasledujúcej vete, ku ktorej poskytneme vlastný dôkaz.

Veta 1.3.1 (Cyklická prechádzka)**.** *Nech $\mathcal{E} = (E, D, \mathcal{N}, \mathcal{T}, \mathcal{K}, \mathcal{X}, \perp)$ je FPE schéma nad doménou $\mathcal{X} = \bigcup_N \mathcal{X}_N$. Nech $\mathcal{X}'_N \subseteq \mathcal{X}_N$ pre všetky $N \in \mathcal{N}$ a nech $\mathcal{X}' = \bigcup_N \mathcal{X}'_N$. Nech $\mathcal{E}'$ je FPE schéma $(E', D', \mathcal{N}, \mathcal{T}, \mathcal{K}, \mathcal{X}', \perp)$, kde ${E'}_K^{N,T}$ je definované ako*

$$ {E'}_K^{N,T}(X) = \begin{cases} E_K^{N,T}(X), & \text{ak } E_K^{N,T}(X) \in \mathcal{X}', \\ {E'}_K^{N,T}(E_K^{N,T}(X)), & \text{ak } E_K^{N,T}(X) \notin \mathcal{X}'. \end{cases} $$

Potom je $\mathcal{E}'$ FPE schéma nad doménou $\mathcal{X}'$, pričom pre každé $X \in \mathcal{X}'$ hodnota ${E'}_K^{N,T}(X) \in \mathcal{X}'$ a je vypočítaná v konečnom počte krokov.

Dôkaz. Z definície ${E'}_K^{N,T}(X)$ vidíme, že sa bude opakovane volať dovtedy, kým $E_K^{N,T}(X)$ nebude z domény $\mathcal{X}'$ a vtedy túto hodnotu vráti. Takže ak ${E'}_K^{N,T}$ nejaké hodnotu vráti, bude určite z domény $\mathcal{X}'$.

Ešte je potrebné ukázať, že ${E'}_K^{N,T}$ vždy vráti nejakú hodnotu. Pre spor predpokladajme, že existuje také $X_0 \in \mathcal{X}'$, na ktorom sa bude ${E'}_K^{N,T}$ cykliť donekonečna. Nech $\{X_i\}_{i \geq 0}$ je postupnosť argumentov ${E'}_K^{N,T}$ počas tohto nekonečného cyklenia. Pretože členy tejto postupnosti sú z domény $\mathcal{X}$ (lebo vznikli použitím E_K na predchádzajúci prvok) a doména $\mathcal{X}$ je konečná, tak táto postupnosť musí byť od nejakého miesta periodická s nejakou (hoci aj prázdnou) predperiódou. Zároveň hodnota každého prvku závisí iba od hodnoty prvku pred ním, keďže $X_i = E_K^{N,T}(X_{i-1})$. Označme X_j prvý prvok,

v ktorom sa začala perióda (teda prvý prvok, ktorý sa v tejto postupnosti zopakoval) a n nech je dĺžka periódy. Potom platí[10] $E_K^{N,T}(X_{j-1}) = X_j = X_{j+n} = E_K^{N,T}(X_{j+n-1})$.

Keďže je $E_K^{N,T}$ FPE šifrovanie, existuje k nemu aj FPE dešifrovanie $D_K^{N,T}$, v ktorom $X_{j-1} = D_K^{N,T}(X_j) = D_K^{N,T}(X_{j+n}) = X_{j+n-1}$. To ale znamená, že prvok X_{j-1} sa tiež zopakoval v danej postupnosti, čo je spor s tým, že prvý takýto prvok je až X_j. Preto funkcia $E'^{N,T}_K$ vždy vráti nejakú hodnotu a táto hodnota je z domény $\mathcal{X}'$, takže $\mathcal{E}'$ je FPE schéma nad doménou $\mathcal{X}'$. □

Efektívnosť. Aj keď šifrovanie $E'^{N,T}_K$ skončí po konečnom počte opakovaní, bolo by vhodné, ak by bol počet aplikácií $E_K^{N,T}$ malý. V prípade cyklickej prechádzky a dobrého (pseudonáhodného) šifrovania $E_K^{N,T}$ je očakávaný počet opakovaní $|\mathcal{X}_N|/|\mathcal{X}'_N|$. Takže ak cieľová množina $\mathcal{X}'_N$ nie je príliš malá v porovnaní s pôvodnou množinou $\mathcal{X}_N$, očakávaný počet šifrovaní bude malý. V príklade so 6-cifernými číslami spomínanom v úvode je očakávaný počet opakovaní približne 1,05. Ak by sme však chceli šifrovať napríklad rodné čísla pomocou cyklickej prechádzky a AES, tak očakávaný počet opakovaní by bol približne $2^{128}/2^{32} = 2^{96}$, čo je už neúnosne veľa.

Na tomto mieste je nutné poznamenať, že efektívnosť metódy cyklickej prechádzky závisí aj od efektívnosti testovania, či prvok X patrí alebo nepatrí do množiny $\mathcal{X}'$. Ak by neexistoval efektívny test na príslušnosť prvku do množiny $\mathcal{X}'$, tak táto metóda nebude efektívna.

Bezpečnosť. Použitie cyklickej prechádzky otvára postranný kanál informácie pre útočníka, a to čas šifrovania. Ak každé použitie šifrovania $E_K^{N,T}$ trvá približne rovnako dlho, tak čas šifrovacej transformácie $E'^{N,T}_K$ už trvá rôzne dlho v závislosti od počtu volaní $E_K^{N,T}$. Tento čas môže útočník zmerať a získať tak informáciu o počte použitých šifrovacích transformácií a teda aj o počte prvkov postupnosti $\{X_i\}_{i\geq 0}$ a ich vzťahu k množinám $\mathcal{X}'$, resp. $\mathcal{X}$. Našťastie, v článku [1] je dokázané, že prezradenie tejto informácie útočníkovi neovplyvní PRP bezpečnosť FPE schémy $\mathcal{E}'$.

1.3.2 Ohodnoť a šifruj

Predstavme si, že máme FPE schému pre doménu $\mathcal{X} = [S] = \{0, \ldots, N-1\}$. Takúto FPE schému nazvime celočíselná FPE schéma. Ak by sme teraz chceli skonštruovať FPE schému nad ľubovoľnou doménou $\mathcal{X}'$, pričom $|\mathcal{X}'| = S$, tak by nám stačilo nájsť nejakú bijekciu medzi $\mathcal{X}$ a $\mathcal{X}'$. Potom by sme mohli správy z $\mathcal{X}'$ previesť pomocou bijekcie na správy z $\mathcal{X}$, tie zašifrovať pomocou celočíselného FPE a výsledok opäť previesť bijektívnym zobrazením späť do $\mathcal{X}'$.

[10] predpokladáme $j \geq 1$, inak by $X_j = X_0 \in \mathcal{X}'$

Prístup ohodnoť a šifruj (rank-then-encipher) je konkrétny spôsob realizácie vyššie uvedenej myšlienky. Formálne ho popíšeme nasledovne:

Definícia 1.3.1 (Ohodnoť a šifruj)**.** *Nech rank je zobrazenie* $\mathcal{N} \times \mathcal{X} \to \mathbb{N} \cup \{\perp\}$*, pre ktoré je* $rank_N(X) = rank(N, X)$ *bijekcia medzi* $\mathcal{X}_N$*, pričom* $rank_N(X) = \perp \Leftrightarrow N \notin \mathcal{N} \vee X \notin \mathcal{X}_N$*. Nech unrank je zobrazenie* $\mathcal{N} \times \mathbb{N} \to \mathcal{X} \cup \perp$*, pre ktoré je* $unrank_N(n) = unrank(N, n)$ *bijekcia medzi* $[|\mathcal{X}_N|]$ *a* $\mathcal{X}_N$*, pričom* $unrank_N(n) = \perp \Leftrightarrow n \notin [|\mathcal{X}_N|]$*. Potom prístup „usporiadaj a šifruj“ predstavuje FPE šifrovanie* ${E'}_K^{N,T}$ *definované ako*

$$ {E'}_K^{N,T}(X) = unrank_N\left(E_K^{|\mathcal{X}_N|,T}\left(rank_N(X)\right)\right), $$

kde $E_K^{|\mathcal{X}_N|,T}$ *je celočíselné šifrovanie na doméne* $[|\mathcal{X}_N|]$*.*

Bezpečnosť a efektívnosť. Je zrejmé, že ak je bezpečné šifrovanie E vzhľadom na PRP_E, SPI_E, MP_E a MR_E, tak je rovnako bezpečné aj šifrovanie E'. Funkcie *rank* a *unrank* nie sú totižto nič iné ako obyčajné premenovanie prvkov domény daného FPE. Efektívnosť (časová náročnosť) šifrovania E' je zjavne súčtom časových náročností pre šifrovanie E a funkcie *rank* a *unrank*.

Veľká výhoda tohto prístupu je to, že ak pre nejakú doménu vieme ľahko skonštruovať efektívne *rank* a *unrank* funkcie, tak vďaka nim môžeme použiť ľubovoľnú celočíselnú FPE schému na konštrukciu FPE schémy pre žiadanú doménu. Zároveň má nová schéma rovnaké bezpečnostné vlastnosti ako pôvodná celočíselná FPE schéma.

FPE schémy pre regulárne jazyky. Použitie prístupu usporiadaj a šifruj ilustrujeme na príklade regulárnych jazykov, v ktorom skonštruujeme pomocou celočíselnej FPE schémy E FPE schému E' pre ľubovoľný regulárny jazyk L nad konečnou abecedou Σ.

Nech $\mathcal{X} = L, \mathcal{N} = \mathbb{N}, \mathcal{X}_N = L \cap \Sigma^N$. Teda N-tý diel domény budú tvoriť všetky slová jazyka L dĺžky N. Majme DKA (deterministický konečný automat) $M = (Q, \Sigma, \delta, q_0, F)$, kde Q konečná množina stavov, Σ je konečná abeceda vstupných symbolov, δ je prechodová funkcia, q_0 je počiatočný stav a F je množina akceptačných stavov. Nech je M taký DKA, že $L_M = L$. Ďalej majme nejaké usporiadanie znakov abecedy $a_1 \prec a_2 \prec \cdots \prec a_{|\Sigma|}$ a funkciu $ord(a_i) = i$. Potom pre jednotlivé slová dĺžky n vieme prirodzene usporiadať pomocou usporiadania znakov abecedy. Teraz už len chceme spočítať pre dané slovo jeho poradie v usporiadaní a tiež pre dané poradie v usporiadaní skonštruovať príslušné slovo (teda chceme funkcie *rank* a *unrank*).

Tieto dve funkcie vieme vypočítať pomocou dynamického programovania. Algoritmy sú uvedené napr. v [1], pre prehľadnosť ich uvedieme aj tu spolu s vysvetlením.

Vo výpise 1.2 funkcia `buildTable(`N`)` spočíta, koľko suffixov slov DKA M akceptuje, keď je v stave q a na vstupe mu ostáva i znakov. Pomocou tejto tabuľky T vieme určiť poradie slova X vo funkcii `rank(`X`)`. Pre každý znak, ktorý je v abecednom usporiadaní pred aktuálnym znakom na vstupe, sčítame počet slov (suffixov), ktoré automat akceptuje (uložené v tabuľke T) a posunieme sa na ďalší znak na vstupe. Pri počítaní funkcie `unrank(`c`)` postupujeme opačne. Sčítavaním počtu suffixov pre danú pozíciu hlavy a jednotlivé znaky na páske v abecednom poradí určíme znak, ktorý musí byť na aktuálnej pozícii hlavy tak, aby počet akceptovaných slov nepresahoval hodnotu c. Po určení znaku na aktuálnej pozícii hlavy sa presunieme na nasledujúce políčko a pokračujeme.

Časová náročnosť funkcií `rank` a `unrank` je $O(|\Sigma| \cdot N)$, keďže pre každú pozíciu hlavy na páske prejdeme v najhoršom prípade tabuľku T pre všetky znaky abecedy. Čas potrebný pre vypočítanie tabuľky T je $O(|Q| \cdot |\Sigma| \cdot N)$. Pri optimalizácii sa preto oplatí konštruovať DKA M tak, aby mal čo najmenej stavov.

```
def buildTable(N):
  for q in Q:
    if q in F:
      T[q,0] = 1
  for i in range(1,N):
    for q in Q:
      for a in Σ:
        T[q,i] += T[δ(q,a),i-1]

def rank(X):
  q = q0; c = 0; n = |X|
  for i in range(1,N):
    for j in range(1,ord(X[i])-1):
      c += T[δ(q, aj),N-i]
    q = δ(q,X[i])
  return c

def unrank(c):
  X = ε; q = q0; j = 1
  for i in range(1,N):
    while c >= T[δ(q,aj),N-i]:
      c -= T[δ(q,aj),N-i]; j += 1
    X[i] = aj; q = δ(q,X[i]); j = 1
  return X
```

Výpis 1.2: Usporiadanie slov regulárneho jazyka [1]

1.4 Celočíselné FPE

V predchádzajúcej časti sme viackrát spomínali celočíselné FPE schémy a pomocou nich vytvárali schémy pre všeobecnejšie domény. V tejto časti priblížime dve základné metódy konštrukcie celočíselných FPE schém, teda schém, ktorých doména $\mathcal{X}$ je podmnožina celých, resp. prirodzených čísel.

1.4.1 Feistelovské siete

V kryptografii veľmi známy a používaný spôsob konštrukcie blokových šifier tvoria Feistelovské siete, resp. Feistelovské šifry. Ide o pomerne starý spôsob zo začiatku 70. rokov 20. storočia, názov konštrukcie je odvodený od jej autora Horsta Feistela, ktorý ju použil v návrhu šifry Lucifer [17] v roku 1971. Neskôr z tejto šifry vychádzal známy šifrovací algoritmus DES, ktorý bol štandardizovaný NBS (predchodca NISTu) koncom 70. rokov [31].

V súčasnosti sú Feistelovské siete veľmi dobre preskúmanou kryptografickou konštrukciou so známymi bezpečnostnými hranicami, vďaka čomu sa často využívajú pri konštrukcii blokových šifier, napr. DES, Triple DES, Camellia, Blowfish.

Feistelovská sieť je viackolová šifra na doménach, ktoré môžu byť tvaru Σ^n, najčastejšie však $\{0,1\}^n$, $\{0,\ldots,9\}^n$ alebo $\{0,\ldots,9,A,\ldots,F\}^n$. V každom kole sa vstupný text rozdelí na dve časti (L,R), ktoré sa transformujú na dve časti výstupného textu (L',R') nasledovne:

$$\begin{aligned} L' &= R \\ R' &= L \boxplus F_i(R,K_i), \end{aligned}$$

kde $\boxplus$ predstavuje sčítanie po znakoch modulo $|\Sigma|$, prípadne sčítanie po blokoch modulo $|\Sigma|^l$, F_i je transformačná funkcia a K_i kľúč asociovaný s i-tym kolom šifrovania. Definičný obor funkcií F_i je Σ^r, kde r je dĺžka časti R, ich obor hodnôt je Σ^l, kde l je dĺžka časti L. Na rozdiel od substitučno-permutačných sietí[11] nepožadujeme, aby boli funkcie F_i invertibilné.

Ak $r = l$, hovoríme, že Feistelovská sieť je klasická, ilustrácia klasickej trojkolovej Feistelovskej siete pre 6-miestne čísla v desiatkovej sústave je zobrazená na obrázku 1.1(a). Poznamenajme, že klasické Feistelovské siete je možné použiť iba na doménach tvaru Σ^{2l}, takže v prípade, že by sme napr. chceli pomocou siete zobrazenej na obrázku

[11] viackolová konštrukcia blokových šifier pozostávajúca zo substitúcií menších bitových blokov, permutácií všetkých bitov a „pripočítania“ kľúča, zvyčajne pomocou operácie XOR. Príkladom takejto blokovej šifry je napr. AES

1.1(a) šifrovať 5-miestne čísla v desiatkovej sústave, museli by sme použiť metódu cyklickej prechádzky, teda doplniť čísla na vstupe na 6-miestne pridaním nuly na ich začiatok a opakovane ich šifrovať pomocou Feistelovskej siete pre 6-miestne čísla, kým by sme nedostali čísla, ktoré budú mať na prvej pozícii nulu. Avšak kvôli veľkosti domén by bol očakávaný počet šifrovaní značne vysoký a tento prístup neefektívny.

Výhodou klasických Feistelovských sietí je jednoduché dešifrovanie. Pri šifrovaní stačí upraviť posledné kolo nasledovne:

$$\begin{aligned} L' &= L \boxplus F_i(R, K_i) \\ R' &= R. \end{aligned}$$

Potom pre dešifrovanie môžeme použiť rovnakú schému (resp. hardvérový obvod), akurát s obráteným poradím funkcií F_i a kľúčov K_i.

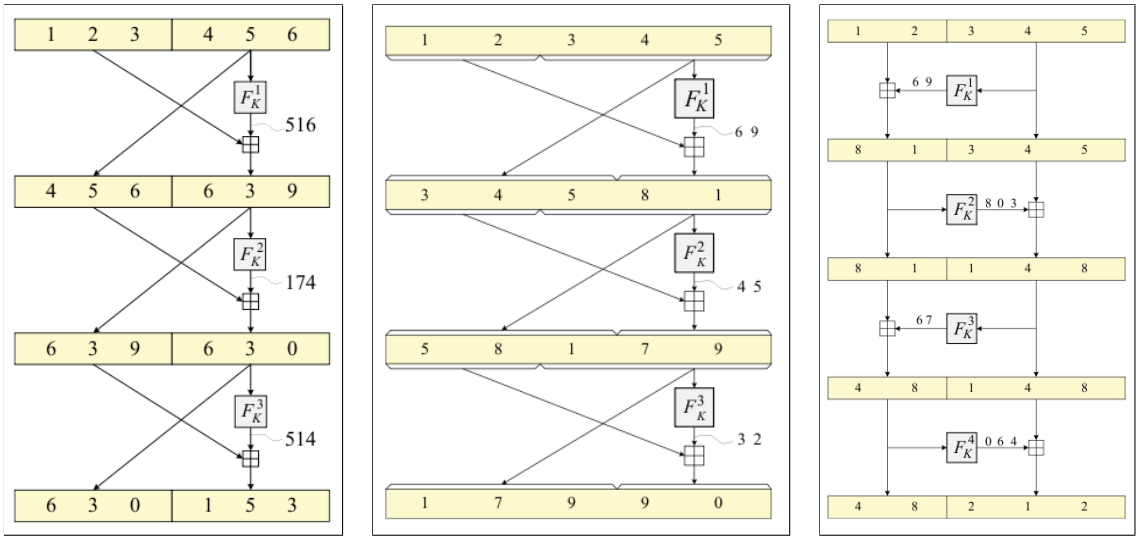
Riešením problému šifrovania blokov nepárnej dĺžky pomocou Feistelovských sietí je použitie nevyváženej alebo alternujúcej Feistelovskej siete, pri ktorých $r \neq l$.

Nevyvážené Feistelovské siete sa niekedy nazývajú aj zovšeobecnené Feistelovské siete, a rozlišujú sa dva typy podľa vzťahu r a l: source-heavy s kontrahujúcimi transformačnými funkciami F_i, ak $l < r$; a target-heavy s expandujúcimi transformačnými funkciami F_i, ak $l > r$. Príklad nevyváženej Feistelovskej siete typu source-heavy je na obrázku 1.1(b).

Bezpečnostné vlastnosti source-heavy sietí sa zlepšujú so narastajúcou mierou nevyváženosti (s klesajúcim l), avšak s primeraným zvýšením počtu kôl. Bezpečnostné vlastnosti target-heavy sietí s veľkou dĺžkou ľavého bloku l nie sú príliš dobré je odporúčané takéto siete nepoužívať [43].

V predchádzajúcom odstavci sme uviedli, že s klesajúcim l je potrebné zvýšenie počtu kôl Feistelovskej siete. Ako extrémny príklad môžeme uviesť maximálne nevyváženú Feistelovskú sieť s $l = 1$. V tomto prípade sa vlastne časť R tvorená $n - 1$ znakmi po každom kole posunie o jeden znak doľava a na posledné miesto vpravo sa doplní znak, ktorý je odvodený zo všetkých znakov v pôvodnom bloku. Je evidentné, že potrebujeme aspoň n kôl šifrovania, aby každý znak výsledného textu bol aspoň raz ovplyvnený transformačnou funkciou a kľúčom. Takýto počet kôl sa niekedy zvykne označovať pojmom „prechod“ (z angl. pass). S takouto Feistelovskou sieťou sa ešte stretneme v časti 2.1, kde sa budeme bližšie zaoberať jej bezpečnosťou.

Nevýhoda nevyvážených Feistelovských sietí spočíva v opakovanom prerábaní a zmene veľkosti častí L a R. Pri hardvérových implementáciách je táto nevýhoda iba myšlienková, pri reálnom zapojení („zadrôtovaní“) vstupov a výstupov nie je potrebná žiadna réžia navyše, avšak pri softvérových implementáciách je potrebných niekoľko



(a) klasické (b) nevyvážené (c) alternujúce

Obr. 1.1: Feistelovské siete

Zdroj obrázkov: [43]

aritmetických operácií (binárne posuny, resp. násobenie a delenie, sčítanie) a malý pamäťový priestor navyše. Túto nevýhodu môžeme eliminovať použitím alternujúcich Feistelovských sietí. Príklad takejto siete je na obrázku 1.1(c).

Princíp alternujúcej Feistelovskej siete spočíva v striedaní kontrahujúcich a expandujúcich transformačných funkcií, vďaka čomu je možné použiť výstupné časti L' a R' jedného kola ako vstupné časti L a R nasledujúceho kola bez dodatočných úprav a zmien veľkostí.

Luby a Rackoff [27] dokázali, že Feistelovské siete predstavujú efektívny spôsob, ako vytvoriť pomocou pseudonáhodných funkcií pseudonáhodné permutácie. Ak sú transformačné funkcie F_i (resp. jedna transformačná funkcia $F : \forall i F_i = F$) pseudonáhodné, tak tri kolá v klasickej Feistelovskej sieti stačia na to, aby táto sieť bola pseudonáhodnou permutáciou, resp. 4 kolá stačia na silnú pseudonáhodnú permutáciu (CCA útok) s použitím najviac $S^{\frac{1}{4}}$ dopytov.

1.4.2 Permutácie a miešania kariet

Celočíselné FPE schémy predstavujú pseudonáhodné permutácie danej domény, pre ktorú sú určené. Môžeme povedať, že takéto FPE schémy realizujú pseudonáhodné permutácie S-prvkovej množiny. Ak očíslujeme prvky takejto množiny číslami $0, \ldots, S-1$, dostávame celočíselnú FPE schému na doméne $[S]$.

Alternatívny pohľad na množinu $[S]$ predstavuje balíček S kariet očíslovaných číslami $0, \ldots, S-1$. Potom každé zamiešanie tohto balíčka kariet môžeme interpretovať ako permutáciu množiny $[S]$. Takáto úvaha vedie k súvisu miešaní kariet a celočíselných FPE schém a myšlienke vytvoriť celočíselné FPE schémy pomocou miešaní kariet.

Problematike miešaní kariet a ich aplikácii v oblasti celočíselných FPE schém sa podrobne venujeme v kapitole 2 a niektoré známe FPE schémy založené na miešaniach kariet uvádzame v časti 2.2.

Kapitola 2

FPE a miešania kariet

V časti 1.4.2 sme uviedli súvis permutácií a miešaní kariet s celočíselnými FPE schémami. Celočíselné schémy na doméne $[S]$ predstavujú permutácie množiny $[S]$ rovnako ako miešanie balíčka očíslovaných kariet s číslami $0, 1 \ldots, S-1$.

Pri miešaní kariet použijeme kľúč K spôsobom, ktorý ovplyvní výslednú pozíciu jednotlivých kariet, napr. K použijeme ako parameter pri nastavení počiatočného stavu generátora pseudonáhodných čísel, ktorý bude predstavovať zdroj náhodnosti v danom miešaní kariet. Podobným spôsobom môžeme použiť aj vylepšenie. Pokiaľ nebude uvedené inak, budeme predpokladať, že generované pseudonáhodné čísla majú uniformnú pravdepodobnostnú distribúciu.

Nepamätlivé miešania. Pri väčšine miešaní kariet závisí pozícia konkrétnej karty od pozícií mnohých iných kariet a na mnohých výstupoch generátora pseudonáhodných čísel. Ako príklad uveďme Knuthovu permutáciu predstavenú v časti 1.2.3 na str. 15. Pri výmene kariet na pozíciách i a j môže byť na pozícii i, resp. j akákoľvek karta s číslom väčším-rovným i, resp. j. Závisí to od toho, či v predchádzajúcich výmenách dvojíc kariet boli tieto karty vymenené, teda či boli vygenerované pseudonáhodné čísla i, j alebo nie.

Závislosť pozície konkrétnej karty od pozícií ostatných kariet znamená, že na určenie pozície jednej karty potrebujeme určiť pozíciu $O(S)$ iných kariet a zvyčajne urobiť $O(S)$ dopytov na generátor pseudonáhodných čísel. To nie je pre Small-space techniky FPE vhodné vzhľadom na ohraničenie $S \leq 2^{128}$.

Pri vytváraní FPE schém pomocou miešaní kariet sa používajú miešania, v ktorých vieme sledovať pohyb karty („trajektóriu“) a na určenie jej konečnej pozície potrebujeme určiť pozície iba malého počtu iných kariet. To znamená, že čas miešania závisí najmä na počte kôl miešania a nie na veľkosti balíčka kariet. Miešania s touto vlastnosťou pomenoval Naor v práci [30] ako nepamätlivé miešania (v orig. oblivious shuffles).

Nepamätlivé miešania kariet sa dajú použiť v kryptografii pre konštrukciu pseudonáhodných permutácií a FPE schém.

2.1 Thorpovo miešanie a hra Faro

Dokonalé miešanie kariet je také, ktorého výsledkom je potenciálne každá permutácia kariet s rovnakou pravdepodobnosťou. Bežné ľudské miešanie však je zaťažené určitou nedokonalosťou[1] a nie je úplne náhodné. Väčšinou sa pri hraní hazardných kartových hier tento fakt prehliada a predpokladá použitie práve dokonalého miešania, ako napr. v jednej z prvých analýz kartovej hry Faro od Leonarda Eulera [16]. O vyše 200 rokov neskôr sa vplyvom miešania kariet na pravdepodobnosť úspechu hráčov pri hraní kartových hier zaoberal Thorp pri analýze hier Faro a Blackjack [47].

Kartová hra Faro. Hra sa hrá so štandardným balíčkom 52 kariet. Bankár najskôr balíček zamieša a prvá karta sa zahodí. Následne sa 25-krát opakuje nasledovná procedúra:

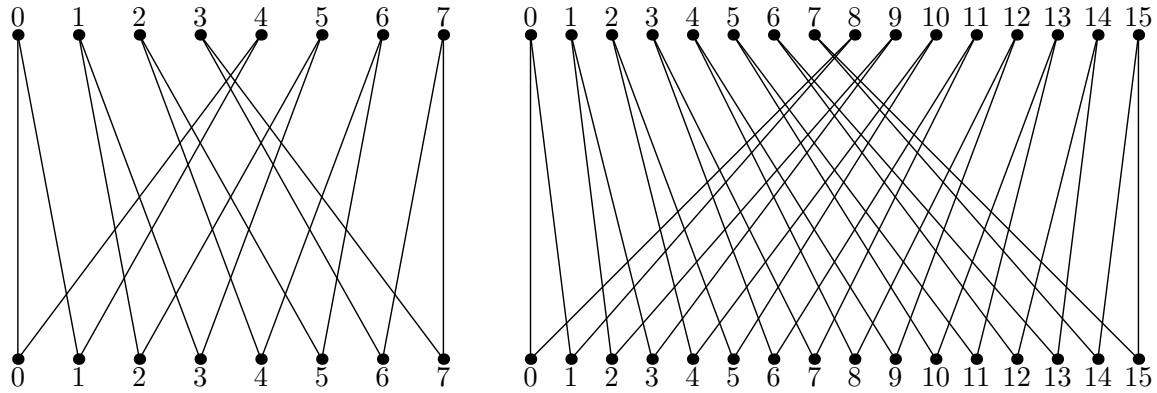
1. Hráči uzatvoria stávky na hodnotu kariet (nie na farbu).
2. Z balíčka sa vyložia dve karty, pričom prvá sa nazýva prehrávajúca a druhá vyhrávajúca.
3. Hráči, ktorí stavili na rovnakú hodnotu, ako má prehrávajúca karta, prehrali. Hráči, ktorí stavili na rovnakú hodnotu, ako má vyhrávajúca karta, vyhrali. Ak majú prehrávajúca aj vyhrávajúca karta rovnakú hodnotu, hráči, ktorí na ne stavili, prehrajú o polovicu stávky. Hráči, ktorí stavili na inú hodnotu, ako majú prehrávajúca a vyhrávajúca karta, nevyhrali ani neprehrali.

Thorpovo miešanie. Thorpovo miešanie použil Thorp vo svojej práci [47] ako model pre nedokonalé ľudské miešanie. Toto miešanie vychádza z jedného z najznámejších spôsobov miešania kariet, tzv. zipsového miešania. Matematicky ho môžeme popísať nasledovne:

Majme balíček kariet obsahujúci $2S$ kariet očíslovaných $0, 1, \ldots, 2S-1$. Kartu na pozícii i budeme označovať ako c_i pre $i \in \{0, \ldots, 2S-1\}$. Tento balíček rozdelíme na dve polovice obsahujúce karty $0, \ldots, S-1$ a $S, \ldots, 2S-1$. Potom karty zmiešame dokopy tak, že karty c_i a c_{S+i} uložíme na pozície $2i$ a $2i+1$ pre $i \in \{0, \ldots S-1\}$, pričom predpokladáme, že je rovnako pravdepodobné, že karty c_i a c_{S+i} skončia v tomto poradí

[1] napr. „zlepia“ sa dve susedné karty; pravdepodobnosť, že niektoré dve konkrétne karty skončia na susedných pozíciách nie je uniformná, atď.

na pozíciách $2i, 2i+1$ alebo $2i+1, 2i$. Aplikovanie jedného kola Thorpovho miešania vedie k jednej z 2^S permutácií balíčku kariet[2], pričom každá z týchto permutácií je rovnako pravdepodobná. Na obrázku 2.1 sú znázornené možné trajektórie kariet v Thorpovom miešaní pre prípady $S = 4$ a $S = 8$.



(a) $S = 4$ (b) $S = 8$

Obr. 2.1: Thorpovo miešanie

2.1.1 Matica pravdepodobností

Thorp v práci [47] používal pri analýze Thorpovho miešania maticu pravdepodobností, ktorej definíciu uvádzame nižšie spolu s niektorými vlastnosťami pri aplikácii na Thorpovo miešanie.

Definícia 2.1.1 (Matica pravdepodobností)**.** *Matica pravdepodobností miešania kariet P je štvorcová matica rozmeru $2S$, ktorej prvky $P_{i,j}$ predstavujú pravdepodobnosť, s akou karta c_i v jednom kole miešania s skončí na pozícii j.*

Matica pravdepodobností pre Thorpovo miešanie teda má $P_{i,2i} = P_{i,2i+1} = \frac{1}{2}$ pre $i \in \{0, \ldots 2S-1\}$, pričom všetky dolné indexy sú počítané modulo $2S$, a všetky ostatné prvky $P_{i,j}$ sú nulové. Matica pravdepodobnosti pre Thorpovo miešanie z obrázku 2.1(a)

[2]pre každú z S dvojíc kariet c_i, c_{S+1} sú prípustné dve možnosti usporiadania dvojice po zamiešaní

vyzerá nasledovne:

$$P = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

Vlastnosti matice pravdepodobností.

1. Súčet pravdepodobností v každom riadku aj v každom stĺpci je 1.
2. P^n je matica pravdepodobností po aplikovaní n kôl miešania.
3. Nech P je matica pravdepodobností, potom i-ty riadok matice P^{n+1} je priemer riadkov $2^n i, 2^n i + 1, 2^n i + 2, \ldots, 2^n(i+1) - 1$ matice P, kde $n \geq 0$.
4. Nech $n \geq 0$. Označme $P^{n+1} = (q_{i,s})$. Potom $q_{r,s} = \left(\sum_{i=2^n r}^{2^n(r+1)-1} P_{i,s}\right) / 2^n$.
5. Matica P^n obsahuje 0 práve vtedy, keď $1 \leq n \leq \lfloor \log_2 S - 1 \rfloor$.
6. Nech $P^n = (q_{i,j})$ a $n \geq 1$. Potom maximum z prvkov $q_{i,j}$ nadobúdajú prvky $q_{0,0}$, $q_{0,1}$, $q_{2S-1,2S-2}$, $q_{2S-1,2S-1}$ a minimum nadobúdajú prvky $q_{0,2S-2}$, $q_{0,2S-1}$, $q_{S,2S-2}$, $q_{S,2S-1}$.
7. Pre $2S = 2^n$ je pravdepodobnostná matica P^n uniformná, teda $P^n = (q_{i,j}) = (\frac{1}{2^n})$.

Vlastnosti 3 až 7 sú uvedené v práci [47], dôkazy tvrdení 1 a 2 sú intuitívne a v tejto práci ich neuvádzame.

2.1.2 Thorpova analýza

Thorp vo svojej práci [47] využíval maticu pravdepodobností a jej vlastnosti, na základe ktorých navrhol postup na zvýšenie šancí na výhru v hrách Faro a Blackjack pri použití Thorpovho miešania. Keďže analýza kartových hier je mimo náplň tejto práce, tu uvedieme iba výsledky o Thorpovom miešaní balíčka kariet zaujímavé z kryptografického hľadiska.

Pri miešaní balíčka kariet ($2S = 52$) aj po piatich kolách miešania obsahuje matica P^5 veľký počet núl (vyše 38%), a nenulové prvky majú hodnotu $\frac{1}{32}$. Po šiestich kolách

miešania už P^6 neobsahuje nuly, ale maximálny prvok je $\frac{3}{128}$ (rozdiel oproti „ideálnej“ hodnote $\frac{1}{52}$ je vyše 0,4%), no stále existujú permutácie, ktoré nedosiahneme ani po 6 kolách miešania. Jednoduchý dolný odhad na potrebu 9 kôl, aby sme boli schopní dosiahnuť každú permutáciu, získame odhadom najväčšieho počtu permutácií, ktoré môže vyprodukovať n kôl miešaní.

Z kryptografického hľadiska môžeme hru Faro prirovnať k PRP hre: ak hráč dokáže rozlíšiť použité miešanie, získava tým výhodu (čiastočnej) predikovateľnosti pozícií niektorých kariet. Útočníkova výhoda v PRP hre sa teda prejaví ako zvýšenie šance hazardného hráča na výhru. Na základe vyššie uvedených poznatkov o Thorpovom miešaní môžeme usúdiť, že pre malé počty kôl miešaní je výhoda hráča nezanedbateľná.

Thorpova analýza je zaujímavá pre porovnanie s kryptografickou analýzou Thorpovho miešania. Myšlienke použiť maticu pravdepodobností na analýzu bezpečnosti permutácii sa budeme bližšie venovať v časti 2.3.

2.1.3 Kryptografická analýza

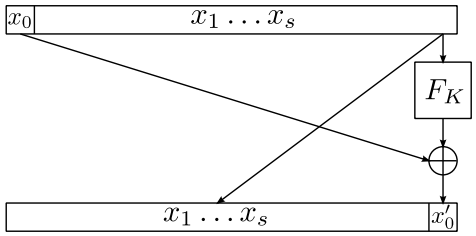
V úvode kapitoly sme zaviedli pojem nepamätlivého miešania. Thorpovo miešanie spĺňa vlastnosti nepamätlivého miešania v zmysle, že pre určenie pozície karty c_i po jednom kole miešania nám stačí jeden dotaz na generátor pseudonáhodných čísel (bitov), keďže karta c_i skončí na pozícii $2i + b \mod 2S$, kde b je náhodný bit. Pre určenie trajektórie karty c_i po n kolách miešania nám stačí poznať iba n náhodných bitov, nepotrebujeme vedieť, na ktorých pozíciách skončili ostatné karty.

V nasledujúcom texte uvažujme Thorpovo miešanie pre S kariet, kde $S = 2^s$. V tomto prípade môžeme jednotlivé karty interpretovať ako s-bitové binárne reťazce a Thorpovo miešanie ako maximálne nevyváženú Feistelovskú sieť. Nech $X = x_0 \dots x_{s-1}$ je karta. Potom pre transformáciu v jednom kole Feistelovskej siete platí:

$$\begin{aligned} L &= x_0 \\ R &= x_1 \dots x_{s-1} \\ L' &= R = x_1 \dots x_{s-1} \\ R' &= L \oplus F_i(R, K_i) = x_0 \oplus F_i(x_1 \dots x_{s-1}, K_i) \end{aligned}$$

Takto definovaná Feistelovská sieť teda realizuje jedno kolo šifrovania s predpisom $Y = 2X + b \mod 2S$, kde b predstavuje výstup pseudonáhodnej funkcie F, ktorá tu vystupuje v úlohe generátora pseudonáhodných bitov, čo zodpovedá Thorpovmu miešaniu. Jedno kolo Feistelovskej siete pre Thorpovo miešanie je znázornené na obrázku 2.2.

Výhody interpretácie Thorpovho miešania ako nevyváženej Feistelovskej siete sú zrejmé: môžeme pre jeho kryptografickú analýzu použiť osvedčené techniky a výsledky



Obr. 2.2: Thorpovo miešanie ako Feistelovská sieť

platné pre Feistelovské siete. Morris, Rogaway a Stegers v práci [29] ukázali, že z kryptografického hľadiska (s adekvátnym počtom kôl) predstavuje dobrú pseudonáhodnú permutáciu. Pre úplnosť uvedieme niektoré výsledky ich analýzy Thorpovho miešania.

Veta 2.1.1 (nPRP bezpečnosť [29])**.** *Nech* $\mathcal{E} = (E, D, \{s\}, \{T\}, \mathcal{K}, \{0,1\}^s, \perp)$ *je FPE schéma, kde E predstavuje Thorpovo miešanie s počtom kôl* $r(2s-1)$*,* $S = |\mathcal{X}|$ *a* $\mathcal{A}$ *je ľubovoľný neadaptívny PRP útočník, ktorý vykoná najviac q dopytov, kde* $q \leq S$*. Potom platí:*

$$\mathbf{Adv}_{\mathcal{E}}^{nPRP}(\mathcal{A}, q) \leq \frac{q}{r+1}\left(\frac{4sq}{S}\right)^r.$$

Veta 2.1.2 (dPRP bezpečnosť [29])**.** *Nech* $\mathcal{E} = (E, D, \{s\}, \{T\}, \mathcal{K}, \{0,1\}^s, \perp)$ *je FPE schéma, kde E predstavuje Thorpovo miešanie s počtom kôl* $r(4s-2)$*,* $S = |\mathcal{X}|$ *a* $\mathcal{A}$ *je ľubovoľný dPRP útočník, ktorý vykoná najviac q dopytov, kde* $q \leq S$*. Potom platí:*

$$\mathbf{Adv}_{\mathcal{E}}^{dPRP}(\mathcal{A}, q) \leq \frac{2q}{r+1}\left(\frac{4sq}{S}\right)^r.$$

Dôkazy uvedených viet sú uvedené v práci [29]. Autori tiež uvádzajú dôsledky týchto viet, podľa ktorých je $2rs$-kolové (resp. $4rs$-kolové) Thorpovo miešanie nPRP-bezpečné (resp. dPRP-bezpečné) s použitím najviac $S^{1-1/r}$ dopytov pre dostatočne veľké s.

2.2 FPE schémy založené na miešaniach kariet

V tejto časti práce uvedieme niektoré najznámejšie FPE schémy založené na miešaniach kariet: Recursive-merge, Swap-or-not, Mix-and-cut a Sometimes-recurse.

2.2.1 Recursive-merge

Granboulan a Pornin v práci [18] predstavili návrh algoritmu pre uniformný výber permutácie S-prvkovej množiny, s pamäťovou zložitosťou $O(\log S)$ a za použitia $O((\log S)^3)$ dopytov na generátor pseudonáhodných čísel. Tento algoritmus, ktorý Rogaway v práci [43] nazýva Recursive-merge, z teoretického hľadiska predstavuje ideálne miešanie vďaka dokázanej PRP-bezpečnosti, avšak z praktického hľadiska je jeho

použitie obmedzené. Algoritmus vyžaduje generovanie pseudonáhodných čísel s hypergeometrickou pravdepodobnostnou distribúciou. Generovanie takýto čísel je časovo neefektívne a algoritmus Recursive-merge je z tohto dôvodu nepraktický a pomalý pri použití ako Small-space FPE, čo priznávajú aj samotní autori[3].

Algoritmus využíva pseudonáhodný generátor čísel (PRNG) generujúci prvky množiny $\{0,1\}$, ktorý je možné inicializovať (seed). Pomocou tohto PRNG je skonštruovaná pseudonáhodná funkcia $repartitor(S,p,K)$ vracajúca náhodné číslo z množiny $\{0,\ldots,p\}, p \leq S$, pričom kľúč K je použitý ako seed PRNG. Funkcia *repartitor* má hypergeometrické pravdepodobnostné rozdelenie a využíva $O(\log S)$ volaní PRNG. Implementácia tejto funkcie so skutočným hypergeometrickým rozdelením navrhnutá autormi využíva mnoho operácií s desatinnými číslami a je najpomalšou časťou celého miešania[4].

Funkciu $repartitor(S,p,K)$ využíva funkcia $splitter(S,p,X,K), 0 \leq p, X < S$; X je číslo karty, ktorá uniformne zvolí jeden z $\binom{S}{p}$ výberov p prvkov z S-prvkovej množiny, preindexuje vybraté prvky pomocou čísel $0,\ldots,p-1$ a nevybraté prvky pomocou čísel $p,\ldots,S-1$ so zachovaním ich pôvodného poradia. Nakoniec vráti nový index prvku X. Pri implementácii využíva intervalový strom pre množinu $\{0,\ldots,S-1\}$, v ktorom udržiava informáciu o počte vybratých prvkov daného intervalu a pri traverzovaní týmto stromom vykoná $O(\log S)$ volaní funkcie $repartitor(S,p,K)$, teda dohromady $O((\log S)^2)$ volaní PRNG.

Posledná funkcia, $permutator(S,X,K)$, zvolí na základe parametra K jednu permutáciu ϕ množiny $[S]$ a vráti hodnotu $\phi(X)$. Idea výpočtu tejto hodnoty spočíva v rekurzívnom delení množiny $[S]$ na dve (takmer) rovnako veľké časti a využitia funkcie $splitter(S,\lfloor S/2 \rfloor, X, K)$ na určenie časti, do ktorej prvok X zobrazí permutácia ϕ. Takýchto rekurzívnych delení (a teda aj volaní funkcie *splitter*)je potrebných $O(\log S)$, čo dohromady dáva $O((\log S)^3)$ volaní PRNG.

Autori v práci [18] tiež uvádzajú algoritmy pre počítanie inverznej permutácie a dôkazy, že uvedené miešanie je PRP-bezpečné a $dPRP$-bezpečné, ak je PRNG skutočne náhodný generátor.

2.2.2 Swap-or-not

Hoang, Morris a Rogaway v práci [20] predstavili návrh efektívneho FPE šifrovania pre bitové reťazce ($S = 2^s$) a pre celočíselné domény $[S]$ založeného na jednoduchom miešaní kariet, avšak s relatívne dobrými bezpečnostnými vlastnosťami: šifrovanie

[3] šifrovanie jednej hodnoty pre $S = 10^9$ na 2GHz procesore v roku 2007 trvalo približne 0,48 s

[4] autori uvádzajú, že 98% času výpočtov tvoria operácie s desatinnými číslami

Swap-or-not s adekvátnym počtom kôl je PRP-bezpečné s použitím najviac $(1-\varepsilon)S$ dopytov pre ľubovoľné $\varepsilon > 0$, ak je v ňom použitá dobrá pseudonáhodná funkcia. Zároveň r-kolové miešanie Swap-or-not využíva $O(r)$ volaní použitej pseudonáhodnej funkcie.

Myšlienka miešania spočíva v spárovaní kariet c_x a $c_{x\oplus k}$, kde k je kľúč pre dané kolo miešania, a náhodnom rozhodnutí, či si karty v tomto páre svoje pozície vymenia alebo nie. Algoritmus pre miešanie je uvedený vo výpise 2.1, algoritmus pre šifrovanie je uvedený vo výpise 2.2, pričom K_i sú s-bitové kľúče a $F_i : \{0,1\}^s \to \{0,1\}$ sú pseudonáhodné funkcie použité v i-tom kole šifrovania. Dešifrovanie vyzerá rovnako, akurát kľúče a pseudonáhodné funkcie pre jednotlivé kolá použijeme v obrátenom poradí. Miešanie aj šifrovanie Swap-or-not je možné ľahko rozšíriť pre celočíselné domény $[S]$, ak použijeme grupu $G = ([S], +)$, kde operácia $+$ predstavuje sčítanie modulo S a jednotlivé páry budú tvoriť prvky $x, k - x$.

```
def SN_shuffle(c): # c je zoznam kariet v balicku
  k = randint(0,S-1)
  pairs = set([sorted([x, x⊕k]) for x in range(S)])
  for (A,B) in pairs:
    b = randint(0,1)
    if b == 1:
      c[A], c[B] = c[B], c[A]
```

Výpis 2.1: Miešanie Swap-or-not [20]

```
def E_K(X): # X je karta, K je kluc
  for i in range(r): # r je pocet kol sifrovania
    K_i = roundkey(K, i) # ziskame podkluc pre i-te kolo
    F_i = roundfunc(K, i) # ziskame transformacnu funkciu pre i-te kolo
    X' = X ⊕ K_i
    X̂ = max(X, X')
    if F_i(X̂) == 1:
      X = X'
  return X
```

Výpis 2.2: Šifrovanie Swap-or-not [20]

Bezpečnosť. V práci [20] je dokázaná PRP bezpečnosť s použitím nanajvýš $(1-\varepsilon)S$ dopytov a tiež odvodená výhoda CCA útočníka v dPRP hre. Hlavné výsledky uvádzame nižšie.

Veta 2.2.1 (nPRP bezpečnosť [20])**.** *Nech $\mathcal{E} = (E, D, \{s\}, \{T\}, \mathcal{K}, \{0,1\}^s, \perp)$ je FPE schéma, kde E predstavuje Swap-or-not miešanie s počtom kôl r, $S = |\mathcal{X}|$ a $\mathcal{A}$ je*

ľubovoľný neadaptívny PRP útočník, ktorý vykoná najviac q dopytov, kde $q \leq S$. Potom platí:

$$\mathbf{Adv}_{\mathcal{E}}^{nPRP}(\mathcal{A}, q) \leq \frac{2S^{3/2}}{r+2}\left(\frac{q+S}{2S}\right)^{r/2+1}.$$

Veta 2.2.2 (dPRP bezpečnosť [20])**.** *Nech $\mathcal{E} = (E, D, \{s\}, \{T\}, \mathcal{K}, \{0,1\}^s, \perp)$ je FPE schéma, kde E predstavuje Swap-or-not miešanie s počtom kôl $2r$, $S = |\mathcal{X}|$ a $\mathcal{A}$ je ľubovoľný dPRP útočník, ktorý vykoná najviac q dopytov, kde $q \leq S$. Potom platí:*

$$\mathbf{Adv}_{\mathcal{E}}^{dPRP}(\mathcal{A}, q) \leq \frac{4S^{3/2}}{r+2}\left(\frac{q+S}{2S}\right)^{r/2+1}.$$

2.2.3 Mix-and-cut

V tejto časti budeme používať pojem PRS bezpečnosti (pseudorandom separator), ako ho definovali Ristenpart a Yilek v práci [41]. Pseudonáhodný separátor Γ-PRS z pohľadu miešania kariet je pseudonáhodné rozdelenie balíčka $S = 2^s$ kariet na $\Gamma = 2^\gamma \leq S$ kôpok pre $\gamma \geq 1$. Môžeme sa naň pozerať ako na permutáciu množiny S, ktorá pre každú kartu vráti číslo kôpky, do ktorej karta patrí. Γ-PRS teda môžeme interpretovať ako funkciu $\{0,1\}^s \to \{0,1\}^\gamma$. Z kryptografického hľadiska je permutácia ϕ množiny $\{0,1\}^s$ (resp. zobrazenie $\{0,1\}^s \to \{0,1\}^s$) Γ-PRS-bezpečná práve vtedy, keď žiadny útočník nerozlíši prvých γ bitov výstupu ϕ od výstupu náhodnej permutácie π. Poznamenajme, že pre $\gamma = s$ je Γ-PRS bezpečnosť ekvivalentná PRP bezpečnosti, pre $\gamma < s$ je Γ-PRS bezpečnosť slabšou verziou PRP bezpečnosti.

Ristenpart a Yilek v práci [41] uviedli návrh konštrukcie PRP-bezpečného miešania $S = 2^s$ kariet pomocou iného, 2-PRS-bezpečného miešania kariet. Ako 2-PRS bezpečné miešanie kariet využili miešanie Swap-or-not popísané v časti 2.2.2 a dosiahli PRP bezpečnosť použitím $\Theta((\log(S))^2)$ volaní pseudonáhodnej funkcie [41, 28].

```
def MC_shuffle(c): # c je zoznam kariet v balicku
  S = len(c)
  if S > 1:
    SN_shuffle(c)
    MC_shuffle(c[0:S/2-1])
    MC_shuffle(c[S/2:S-1])
```

Výpis 2.3: Miešanie Mix-and-cut [41]

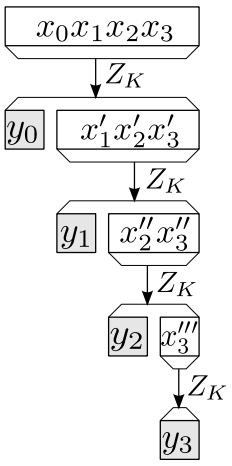
Algoritmus. Autorov článku inšpirovalo miešanie Recursive-merge opísané v časti 2.2.1, konkrétne myšlienka delenia balíčka kariet na menšie, avšak bez potreby hypergeometrického rozdelenia. Namiesto toho využili miešanie Swap-or-not na zamiešanie

balíčka 2^s kariet a následné rozdelenie na dve polovice. Keďže Swap-or-not je 2-PRS-bezpečné[5], tak rozdelenie balíčka kariet na dve polovice je z kryptografického (pseudo)náhodné. Pre dosiahnutie úplnej PRP bezpečnosti (a nie iba 2-PRS bezpečnosti) tento postup rekurzívne opakovali na oboch poloviciach. Algoritmus pre miešanie kariet je uvedený vo výpise 2.3.

```
def E_K(X): # X je karta, K je kluc
  r = s / γ
  Y = []
  for i in range(r):
    X = Z_K(X) # Z_K je Γ-PRS-bezpecne sifrovanie
    Y += X[0:γ-1]
    X = X[γ:]
  return Y
```

Výpis 2.4: Šifrovanie Mix-and-cut [41]

Pre vytvorenie Mix-and-cut šifrovania využili autori konštrukciu, ktorú nazvali Cencúľ (v orig. Icicle). Konštrukcia Cencúľ je znázornená na obrázku 2.3, algoritmus šifrovania je vo výpise 2.4. Myšlienka spočíva v postupnom počítaní prvých γ bitov výstupu pomocou Γ-PRS-bezpečného šifrovania Z, pričom musí byť splnená podmienka $\gamma|s$.



Obr. 2.3: Cencúľ, konštrukcia Mix-and-cut šifrovania [41]

2.2.4 Sometimes-recurse

Morris a Rogaway v práci [28] navrhli vylepšené miešanie Mix-and-cut (časť 2.2.3) s použitým Swap-or-not (časť 2.2.2) ako vnútorným miešaním. Pomocou malej zmeny

[5] v práci [41] je dokázané, že 2-PRS bezpečnosť Swap-or-not vyplýva z PRP bezpečnosti pre $q = S/2$

v algoritme Mix-and-cut dosiahli výrazné zrýchlenie – miešanie Sometimes-recurse potrebuje $\Theta(\log S)$ volaní pseudonáhodnej funkcie, pri zachovaní PRP-bezpečnosti.

Autori namiesto Γ-PRS-bezpečného miešania ako vnútorného miešania použili bezpečnejšieho miešanie, ktoré spĺňa PRP bezpečnosť s použitím najviac q dopytov. Ak je vnútorné miešanie dostatočne dobré, aby pseudonáhodne zamiešalo $S/2$ kariet[6], nemusíme túto polovicu ďalej miešať a v rekurzívnom volaní Mix-and-cut opäť zamiešame iba druhú polovicu kariet. Týmto spôsobom môžeme PRP-bezpečné miešanie s použitím najviac q_0 dopytov pre nejaké $q_0 \geq S/2$ vylepšiť na PRP-bezpečné miešanie. Formálne dôkazy bezpečnosti prístupu Sometimes-recurse je možné nájsť v práci [28].

Algoritmus miešania je mierne upravený algoritmus Mix-and-cut:

1. Zamiešame balíček S kariet pomocou vnútorného miešania.
2. Rozdelíme balíček na dve časti a rekurzívne zamiešame prvú časť.

Podrobnejší popis šifrovania Sometimes-recurse pre celočíselné domény $[S]$ s použitím zovšeobecneného šifrovania Swap-or-not ako vnútorného miešania uvádzame vo výpise 2.5, ktorom r_S predstavuje počet kôl šifrovania Swap-or-not, aby toto šifrovanie spĺňalo bezpečnostné predpoklady.

```
def E_K^S(X): # X je karta, S je velkost balicka kariet, K je kluc
  if S == 1:
    return X

  for i in range(r_S):
    K_i = roundkey(K, i) # ziskame podkluc pre i-te kolo
    F_i = roundfunc(K, i) # ziskame transformacnu funkciu pre i-te kolo
    X' = K_i - X (mod S)
    X̂ = max(X, X')
    if F_i(X̂) == 1:
      X = X'

  if X < ⌊S/2⌋:
    return E_K^⌊S/2⌋(X)
  else:
    return X
```

Výpis 2.5: Šifrovanie Sometimes-recurse [28]

[6] teda počet povolených dopytov q je aspoň $S/2$

2.2.5 Zhrnutie

V častiach 2.1 a 2.2 sme uviedli pravdepodobne najznámejšie miešania kariet, pomocou ktorých je možné konštruovať FPE schémy a ktoré predstavujú v súčasnosti používané princípy pri tvorbe nepamätlivých miešaní s kryptografickým využitím. Na záver týchto dvoch častí uvádzame v tabuľke 2.1 stručné zhrnutie jednotlivých miešaní, ich bezpečnosť a efektívnosť.

Miešanie	**Domény**[7]	**PRP pre** $q \ll S$	**PRP pre** $q = S$	**Volania PRNG**
Thorpovo miešanie [29]	$[2S]$	áno	nie	$O(\log S)$
Recursive-merge [18]	$[S]$	áno	áno	$O((\log S)^3)$
Swap-or-not [20]	$[S]$	áno	nie	$O(\log S)$
Mix-and-cut [41]	$[S]$	áno	áno	$O((\log S)^2)$
Sometimes-recurse [28]	$[S]$	áno	áno	$O(\log S)$

Tabuľka 2.1: Prehľad FPE schém založených na miešaniach kariet

2.3 Aplikácia Thorpovej metódy na FPE

Bezpečnosť dnešných kryptografických konštrukcií pre FPE schémy sa posudzuje na základe výhody útočníka v modeloch, ako sú PRP, SP, MP, MR. Ešte pred publikovaním prvých pokusov o FPE [8, 32] však Thorp v práci [47] opísal analýzu štatistických vlastností Thorpovho miešania (bližšie popísané v časti 2.1), pre ktoré takmer o 40 rokov neskôr vypracovali kryptografickú analýzu Morris, Rogaway a Stegers [29], ich výsledky sú uvedené v časti 2.1.3. V tejto časti sa venujeme, či by bolo možné použiť Thorpovu metódu s maticou pravdepodobností aj pre moderné FPE schémy založené na miešaniach kariet a ak áno, aké výsledky by sme dostali a ako by ich bolo možné interpretovať z pohľadu bezpečnosti. V literatúre sme nenašli zmienku o podobnej aplikácii Thorpovej analýzy miešania kariet na FPE schémy založené na miešaniach kariet.

2.3.1 Vplyv na bezpečnosť

Predstavme si, že máme maticu pravdepodobností P pre dané miešanie. Aké vlastnosti musí spĺňať P, resp. P^r pre nejaké $r \geq 1$, aby sme mohli miešanie prehlásiť za dobré, v istom zmysle bezpečné miešanie? Ak pri skúmaní bezpečnosti miešania vychádzame iba zo samotnej matice pravdepodobností P^r, bezpečnostné modely predstavené v časti

[7] pomocou cyklickej prechádzky je možné rozšíriť priestor správ na $[S]$, tu však uvádzame priestor správ, pre ktorý sú dané miešania priamo definované

1.2.2 nám veľmi nepomôžu. V nich sú totiž povolené viaceré dopyty na funkciu *Enc*, a z pohľadu útočníka môže byť pravdepodobnosť výsledku každého takéhoto dopytu podmienená výsledkami dopytov predchádzajúcich[8]. Takéto podmienené pravdepodobnosti nevieme popísať pomocou matice pravdepodobnosti bez toho, aby sme ad hoc aktualizovali maticu pravdepodobností P po každom dopyte na *Enc*.

Najprv zavedieme nový model bezpečnosti, ktorý zodpovedá jednoduchej hre, kartovému ekvivalentu rulety, v ktorej hráč môže staviť na jednu kartu vybratú zo zamiešaného balíka. Formálne hra predstavuje útočníkovu schopnosť rozlíšiť dané miešanie od uniformne náhodného miešania, pričom v danom nastavení môže použiť iba jeden dopyt na orákulum *Enc* a po tomto dopyte sa nastavenie miešania môže zmeniť.

Kartová ruleta. Nech bijektívne zobrazenie $\mathcal{H}_K : [S] \to [S]$ je nenáhodné (pseudonáhodné) miešanie kariet (prvky množiny $[S]$) určené parametrom $K \in \mathcal{K}$. V $\mathrm{RUL}_{\mathcal{H}}$ hre ide o útočníkovu schopnosť rozlíšiť miešanie kariet $\mathcal{H}$ od náhodného miešania kariet, resp. náhodnej permutácie kariet v balíčku. Hra prebieha nasledovne:

1. **Inicializácia.** Zvolíme náhodne jeden bit b z množiny $\{0, 1\}$.

2. **Hra.** Útočník môže volať ľubovoľne veľa krát s ľubovoľným korektným vstupom metódu $Enc(X)$ definovanú nasledovne, pričom $Perm([S])$ predstavuje množinu všetkých permutácií množiny $[S]$:

$$Enc(X) = \begin{cases} \mathcal{H}_K(X), & K \xleftarrow{\$} \mathcal{K}, & \text{ak } b = 1, \\ \pi(X), & \pi \xleftarrow{\$} Perm([S]), & \text{ak } b = 0. \end{cases}$$

3. **Finalizácia.** Po poslednom volaní $Enc(X)$ sa útočník pokúsi uhádnuť hodnotu bitu b a hádanú hodnotu vypíše na výstup ako b'. Finalizácia vráti na výstupe booleovskú hodnotu $(b = b')$.

Pre posúdenie útočníkovej úspešnosti v RUL_H použijeme pojem výhody definovaný pre $\mathrm{RUL}_{\mathcal{H}}$ hru nasledovne:

$$\mathbf{Adv}^{\mathrm{RUL}}_{\mathcal{H}}(\mathcal{A}) = 2 \cdot \Pr\left[\mathrm{RUL}^{\mathcal{A}}_{\mathcal{H}} \Rightarrow \mathrm{True}\right] - 1.$$

Definícia 2.3.1 (RUL bezpečnosť.)**.** *Nech $\mathcal{H}$ je nenáhodné miešanie kariet a $\mathcal{A}$ je ľubovoľný RUL útočník. Hovoríme, že miešanie $\mathcal{H}$ je bezpečné vzhľadom na hru $RUL_{\mathcal{H}}$ (resp. miešanie $\mathcal{H}$ je RUL-bezpečné) práve vtedy, keď platí:*

$$\mathbf{Adv}^{RUL}_{\mathcal{H}}(\mathcal{A}) = \mathrm{negl}(\log |\mathcal{K}|).$$

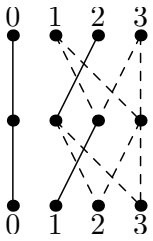
[8]napr. pri Thorpovom miešaní vieme, že na pozíciách $2i, 2i+1$ môže skončia karty c_i, c_{S+i}. Ak na pozícii $2i$ skončí karta c_i, potom nutne na pozícii $2i+1$ musí skončiť karta c_{S+i}

Uveďme vzťah tohto modelu k modelom definovaným v časti 1.2.2.

Veta 2.3.1 (RUL $\not\Rightarrow$ MR)**.** *Existuje také nenáhodné miešanie kariet $\mathcal{H} : \mathcal{X} \to \mathcal{X}$, ktoré je RUL-bezpečné a zároveň z neho odvodená FPE schéma*[9] $\mathcal{E} = (\mathcal{H}, \mathcal{H}^{-1}, N_0, T_0, \mathcal{X})$ *nie je MR-bezpečná.*

Dôkaz. Nech $\mathcal{H}$ predstavuje dvojkolové Thorpovo miešanie množiny $\mathcal{X} = [4]$. Potom matica pravdepodobnosti (pre jedno kolo) miešania $\mathcal{H}$ je totožná matici pravdepodobnosti $P^2 = (q_{i,j})$, kde P je matica pravdepodobnosti Thorpovho miešania. Podľa časti 2.1.1 je $q_{i,j} = \frac{1}{2^2}$ pre $\forall i, j \in [2^s]$, teda je uniformná a rovnaká, ako matica pravdepodobnosti náhodnej permutácie π. To znamená, že pri volaní funkcie Enc v hre $\mathrm{RUL}_{\mathcal{H}}$ je každý výsledok rovnako pravdepodobný bez ohľadu na hodnotu b. RUL útočník si preto môže hodnotu b iba tipnúť s pravdepodobnosťou úspechu $\frac{1}{2}$ a jeho výhoda je nulová. Miešanie $\mathcal{H}$ je preto RUL-bezpečné.

Ukážeme, že schéma $\mathcal{E}$ nie je MR-bezpečná, teda že existuje taký MR útočník $\mathcal{A}$, ktorého výhoda je väčšia ako akákoľvek zanedbateľná funkcia. Nech $\mathcal{A}(dist)$ je uniformná distribúcia. Na začiatku hry vykoná $\mathcal{A}$ jeden dopyt na funkciu Enc a následne sa pokúsi uhádnuť hodnotu X_0. Využije pri tom fakt, že výsledok tohto dopytu nerovnomerne ovplyvní pravdepodobnosti trajektórií zvyšných troch kariet. Predpokladajme, že útočník $\mathcal{A}$ zistí $Enc(N_0, T_0, 0) = 0$. Trajektórie kariet sú znázornené na obr. 2.4, pričom čiarkovane sú znázornené presuny kariet s pravdepodobnosťou $\frac{1}{2}$ a plnou čiarou presuny s pravdepodobnosťou 1 vzhľadom na podmienku $Enc(N_0, T_0, 0) = 0$.



Obr. 2.4: Thorpovo miešanie pri podmienke $Enc(N_0, T_0, 0) = 0$

Matica pravdepodobnosti $P_{\mathcal{H}}$ pre miešanie $\mathcal{H}$ a matica pravdepodobnosti P_π pre náhodné miešanie π za podmienky $Enc(N_0, T_0, 0) = 0$ vyzerajú nasledovne:

$$P_{\mathcal{H}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{4} & \frac{1}{4} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & \frac{1}{2} & \frac{1}{4} & \frac{1}{4} \end{pmatrix}, \qquad P_\pi = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{pmatrix}$$

Podobne vyzerajú dané matice aj za podmienky $Enc(N_0, T_0, 0) = y_0$ pre ostatné $y_0 \in [4]$, iba sú jednotlivé stĺpce inak usporiadané. Po zistení hodnôt $y_0 = Enc(N_0, T_0, 0)$

[9] pre korektnosť položme $E_K^{N,T} = \mathcal{H}_K$ pre $\forall N \in \mathcal{N}, T \in \mathcal{T}$

a $Y_0 = Test()$ útočníkom $\mathcal{A}$ môžu nastať dve možnosti: ak $Y_0 = y_0$, útočník vypíše 0 ako X' a uspeje s pravdepodobnosťou 1; inak útočník vypíše X' tak, aby platilo $q_{X',Y_0} = \frac{1}{2}$ a uspeje s pravdepodobnosťou $\frac{1}{2}$. Potom

$$\Pr\left[\mathrm{MR}_{\mathcal{E}}^{\mathcal{A}} \Rightarrow \mathrm{True}\right] = \frac{1}{4} \cdot 1 + \frac{3}{4} \cdot \frac{1}{2} = \frac{5}{8}.$$

Najlepší simulátor $\mathcal{S}$ pre tohto útočníka však môže iba raz zavolať $Eq(X)$ a potom musí hádať. Pretože $\mathcal{S}(dist)$ je uniformná, platí:

$$\Pr\left[\mathrm{MR}_{\mathcal{E}}^{\mathcal{S}} \Rightarrow \mathrm{True}\right] = \frac{1}{4} + \frac{3}{4} \cdot \frac{1}{3} = \frac{1}{2}.$$

Výhoda útočníka $\mathcal{A}$ v hre $\mathrm{MR}_{\mathcal{E}}$ preto je

$$\mathbf{Adv}_{\mathcal{E}}^{\mathrm{MR}}(\mathcal{A}) = \frac{5}{8} - \frac{1}{2} = \frac{1}{8},$$

čo znamená, že schéma $\mathcal{E}$ odvodená z RUL-bezpečného miešania nie je MR-bezpečná. □

V ďalšom texte budeme uvažovať PPT útočníka, ktorý okrem vstupu dostane ešte aj parameter τ, pričom pravdepodobnostný polynomiálny čas je myslený vzhľadom na daný parameter τ, nie na veľkosť vstupu. Útočník teda nemusí prečítať celý svoj vstupe, ak bude tento vstup príliš veľký, napr. ľubovoľne dlhý rozvoj desatinných čísel. V tomto prípade útočníkov vstup interpretujeme tak, že útočník načíta takýto vstup s obmedzenou presnosťou, teda z reprezentácie každého prvku na vstupe je schopný prečítať iba prvých niekoľko symbolov.

Lema 2.3.2. *Nech $\mathcal{H}_K : [S] \to [S]$ je miešanie, P je jeho matica pravdepodobností a $U = (1/S)$ je matica pravdepodobností uniformne náhodného miešania $[S] \to [S]$. Potom PPT útočník s parametrom*[10] $\log|\mathcal{K}|$ *dokáže rozoznať P od U práve vtedy, ak dokáže rozoznať aj jej maximum (minimum) od hodnoty $1/S$.*

Dôkaz.

$\Leftarrow$ Ak útočník dokáže rozlíšiť jeden prvok matice P od prvku $1/S$ matice U, potom vie rozlíšiť aj obe matice.

$\Rightarrow$ Dokážeme obmenenú implikáciu: ak útočník nedokáže rozoznať maximálny (minimálny) prvok P od $1/S$, potom nedokáže rozoznať ani P od U. PPT útočník s parametrom $\log|\mathcal{K}|$ môže pracovať iba s hodnotami s obmedzenou presnosťou ε, ktorá je daná jeho výpočtovým časom – dve hodnoty, ktorých reprezentácia sa líši až na mieste, ktoré útočník nie je schopný spracovať, vyzerajú z pohľadu útočníka rovnako.

[10] útočník teda načíta každý prvok matice s obmedzenou presnosťou

Ak by útočník nedokázal rozlíšiť maximálny (minimálny) prvok P od $1/S$, nedokázal by rozlíšiť žiadny prvok P od $1/S$. Potom by z pohľadu útočníka matica P aj U pri načítaní a spracovaní vyzerali rovnako. Nech by použil akékoľvek operácie na rozlíšenie matíc P a U (počítanie hodnosti, determinantov, atď.), v oboch prípadoch by reprezentácie hodnôt, s ktorými útočník pracuje, vyzerali rovnako. Z pohľadu útočníka by teda boli matice P a U nerozlíšiteľné. □

Lema 2.3.3. *Miešanie $\mathcal{H}$ je RUL-bezpečné práve vtedy, keď jeho matica pravdepodobností $P = (q_{i,j})$ nie je rozoznateľná od matice pravdepodobností U uniformne náhodného miešania (náhodnej permutácie) v pravdepodobnostnom polynomiálnom čase.*

Dôkaz. Útočník $\mathcal{A}$ je schopný v hre $\mathrm{RUL}_{\mathcal{H}}$ vypočítať maticu pravdepodobností skúmaného miešania s takmer ľubovoľnou presnosťou[11] pomocou metódy Monte Carlo (opakované dopyty na jednotlivé karty a zaznamenávanie relatívnych početností výsledných pozícií týchto kariet $q'_{i,j}$). Pri použití metódy Monte Carlo je relatívna chyba[12] úmerná faktoru $1/\sqrt{q}$, kde q je počet dopytov útočníka $\mathcal{A}$. Útočník tak môže dosiahnuť ľubovoľne malú chybu pri výpočte matice pravdepodobností, limitovaný je iba výpočtovým časom.

$\Leftarrow$ V tomto prípade sa výsledky dopytov na *Enc* správajú rovnako bez ohľadu na to, či *Enc* predstavuje $\mathcal{H}_K$ s náhodne voleným K alebo náhodne volenú permutáciu π, keďže každý výsledok je rovnako pravdepodobný (s ohľadom na čas behu útočníka). Z pohľadu útočníka vyzerá RUL hra rovnako bez ohľadu na hodnotu b a miešanie $\mathcal{H}$ je RUL-bezpečné.

$\Rightarrow$ Sporom. Predpokladajme, že $\mathcal{H}$ je RUL-bezpečné a P je rozoznateľná od U v PPT. Útočník $\mathcal{A}$ vypočíta metódou Monte Carlo maticu P' pomocou dopytov na *Enc*, ktorá bude ľubovoľne presnou aproximáciou[13] matice pravdepodobností miešania $\mathcal{H}$, ak $b = 1$, alebo matice pravdepodobností náhodných permutácií π, ak $b = 0$.

V prípade $b = 1$ bude matica P' rozoznateľná od U, pretože aj P je rozoznateľná od U. V prípade $b = 0$ nebude P' rozoznateľná od U. Útočník tak bude vedieť vždy uhádnuť hodnotu b, čo je spor s RUL-bezpečnosťou miešania $\mathcal{H}$. □

Veta 2.3.4 (MR $\Rightarrow$ RUL)**.** *Nech $\mathcal{H} : \mathcal{X} \to \mathcal{X}$ je miešanie kariet a $\mathcal{E} = (\mathcal{H}, \mathcal{H}^{-1}, n, t, \mathcal{X})$ je k nemu príslušná FPE schéma[14]. Ak je schéma $\mathcal{E}$ MR-bezpečná, potom je miešanie $\mathcal{H}$ RUL-bezpečné.*

[11] presnosť je limitovaná výpočtovým časom útočníka

[12] relatívna chyba je podiel $\frac{q'_{i,j} - q_{i,h}}{q_{i,j}}$, kde $q'_{i,j}$ sú vypočítané hodnoty metódou Monte Carlo a $q_{i,j}$ sú skutočné hodnoty

[13] s ohľadom na PPT

[14] pre korektnosť položme $E_K^{N,T} = \mathcal{H}_K$ pre $\forall N \in \mathcal{N}, T \in \mathcal{T}$

Dôkaz. Dôkaz obmenenou implikáciou. Nech miešanie $\mathcal{H}$ nie je RUL-bezpečné, potom ani schéma $\mathcal{E}$ nie je MR-bezpečná.

Ak miešanie $\mathcal{H}$ nie je RUL-bezpečné, potom podľa lemy 2.3.3 je matica pravdepodobností P miešania $\mathcal{H}$ rozoznateľná od matice pravdepodobností U náhodného miešania (permutácie) v PPT. Uvedieme spôsob, akým môže MR útočník $\mathcal{A}$ využiť rozlíšiteľnosť matíc P a U na úspešný útok v hre $\mathrm{MR}_{\mathcal{E}}$.

Základná myšlienka dôkazu spočíva v snahe nájsť takú správu X_l, ktorej šifrový text môže prezrádzať niečo o pôvodnej správe. Predpokladajme, že takáto správa $\mathcal{X}_l$ existuje. Potom by aspoň v niektorých prípadoch bol útočník $\mathcal{A}$ schopný pomocou $Test$ rozhodnúť, či $X_l = X_0$, čím získa výhodu v porovnaní so simulátorom, ktorý funkciu $Test$ použiť nemôže.

Označme ľubovoľné dve rôzne karty (otvorené texty) X_a, X_b. Potom a-ty a b-ty riadok matice P predstavujú pravdepodobnostné distribúcie šifrových textov prislúchajúcich k otvoreným textom X_a, X_b. Nech útočník $\mathcal{A}$ poskytne MR hre pravdepodobnostnú distribúciu $\mathcal{A}(dist)$, v ktorej bude pravdepodobnosť textov X_a aj X_b rovná $1/2$ počas hry ani raz nezavolá Enc. Útočník $\mathcal{A}$ sa pokúsi hodnotu X_0 uhádnuť iba z hodnoty $Test() = Y_0$ a matice pravdepodobností P, pričom pre danú hodnotu Y_0 vždy vypíše ako X_0' ten otvorený text z X_a, X_b, ktorý má podľa P väčšiu pravdepodobnosť zobrazenia na Y_0. Určíme pravdepodobnosť, že útočník $\mathcal{A}$ uspeje v hre $\mathrm{MR}_{\mathcal{E}}$. Použitím Iversonovej notácie dostávame:

$$\begin{aligned}\Pr\left[\mathrm{MR}_{\mathcal{E}}^{\mathcal{A}} \Rightarrow \mathrm{True}\right] &= \Pr\left[\mathcal{X}_0' = X_a | X_0 = X_a\right] + \Pr\left[\mathcal{X}_0' = X_b | X_0 = X_b\right] = \\ &= \sum_i \frac{q_{a,i}}{2}\left(q_{a,i} \geq q_{b,i}\right) + \frac{q_{b,i}}{2}\left(q_{a,i} < q_{b,i}\right) = \frac{1}{2}\left(p_a + p_b\right) \geq \frac{1}{2},\end{aligned}$$

kde p_a predstavuje súčet tých prvkov z riadku a matice P, ktoré sú väčšie-rovné ako prvky riadku b v tom istom stĺpci (teda $\mathcal{A}$ zvolí $X_0' = X_a$) a p_b predstavuje súčet tých prvkov z riadku b matice P, ktoré sú väčšie ako prvky riadku a v tom istom stĺpci (teda $\mathcal{A}$ zvolí $X_0' = X_b$). Potom zjavne platí $p_a + p_b \geq 1$.

Pravdepodobnosť úspechu simulátora $\mathcal{S}$ pre takéhoto útočníka je $1/2$, keďže nemôže použiť žiadnu z funkcií $Test$, Enc ani Eq a musí hneď vypísať X_0. Ešte však potrebujeme ukázať, že existujú také dva prvky X_a a X_b, pre ktoré bude $\Pr\left[\mathrm{MR}_{\mathcal{E}}^{\mathcal{A}} \Rightarrow \mathrm{True}\right] > 1/2 + \mathrm{negl}(\log|\mathcal{K}|)$.

Využijeme lemu 2.3.2. Keďže P je rozoznateľná od U, tak jej maximálny (minimálny) prvok m_1 sa líši od $1/S$ aspoň o hodnotu $\delta > \mathrm{negl}(\log|K|)$. Okrem prvku m_1 sa v tom istom stĺpci c matice P musí nachádzať aj nejaký prvok m_2, ktorý sa od maxima (minima) líši aspoň o δ, inak by súčet prvkov v tomto stĺpci nebol 1. Zvoľme za otvorené texty X_a a X_b texty zodpovedajúce riadkom, v ktorých sa vyskytuje m_1 a

m_2. BUNV nech $m_1 > m_2$. Potom platí:

$$p_a + p_b \geq \sum_{i \neq c} q_{b,i} + m_2 > \sum_{i \neq c} q_{b,i} + m_1 + \text{negl}(\log |K|) = 1 + \text{negl}(\log |K|).$$

Výhoda útočníka $\mathcal{A}$ v hre $\text{MR}_{\mathcal{E}}$ je

$$\mathbf{Adv}^{\text{MR}}_{\mathcal{H}}(\mathcal{A}) = \Pr\left[\text{MR}^{\mathcal{A}}_{\mathcal{E}} \Rightarrow \text{True}\right] - \Pr\left[\text{MR}^{\mathcal{S}}_{\mathcal{E}} \Rightarrow \text{True}\right] = \frac{1}{2}(p_a + p_b) - \frac{1}{2} > \text{negl}(\log |K|),$$

teda schéma $\mathcal{E}$ nie je MR-bezpečná, ak miešanie $\mathcal{H}$ nie je RUL-bezpečné. □

Dôsledok 2.3.5. *RUL je slabší bezpečnostný model ako MR.*

Dôkaz. Vyplýva z viet 2.3.1 a 2.3.4. □

Zaviedli sme nový bezpečnostný model, ktorý priamo súvisí s maticou pravdepodobností (lema 2.3.3) a ukázali sme, že je slabší ako model MR (dôsledok 2.3.5). Tento výsledok je očakávateľný, keďže v modeli RUL nevieme cielene vykonávať dopyty na miešanie s rovnakými parametrami a využívať podmienenú pravdepodobnosť na základe výsledkov predchádzajúcich dopytov, ako sme uviedli v úvode tejto časti.

Pri skúmaní RUL bezpečnosti miešania $\mathcal{H}$ budeme využívať jeho maticu pravdepodobností P a overovať, nakoľko je blízka matici pravdepodobnosti U uniformne náhodného miešania, keďže to spolu úzko súvisí (lema 2.3.3). Zamerajme sa na vlastnosti P, ktoré uvádzal Thorp vo svojej analýze a tiež na niekoľko potenciálnych vlastných návrhov.

Počet nulových prvkov. Nulový prvok $q_{i,j}$ v matici $P^r = (q_{i,j})$ znamená, že karta x_i sa po r kolách nemôže dostať na pozíciu j. Thorp zisťoval minimálny počet kôl potrebných na to, aby matica P^r neobsahovala žiadne nulové prvky. Poznamenajme, že tento minimálny počet kôl je dobre definovaný, to znamená, že ak P^{r_0} neobsahuje nulové prvky pre nejaké r_0, tak matica P^r neobsahuje nulové prvky ani pre $\forall r \geq r_0$.

Na základe nulových prvkov môže útočník $\mathcal{A}$ ľahko uspieť v hre $\text{RUL}_{\mathcal{H}^r}$, kde $\mathcal{H}^r$ je miešanie kariet pozostávajúce z r kôl. Stačí mu zvoliť riadok P^r obsahujúci najviac nulových prvkov a pomocou *Enc* zisťovať, kam sa pri miešaní dostala karta zodpovedajúca tomuto riadku. Ak *Enc* aspoň raz vráti pozíciu s nulovou pravdepodobnosťou, tak určite $b = 0$, inak je $b = 1$ s pravdepodobnosťou $1 - \left(1 - \frac{z}{|S|}\right)^q$, kde z je maximálny počet nulových prvkov v jednom riadku a q je počet dopytov útočníka na funkciu *Enc*. Prakticky to znamená, že útočník môže uspieť s pravdepodobnosťou ľubovoľne blízkou 1. Nevýhoda tejto vlastnosti matice pravdepodobností je, že ani matica P^r bez nulových prvkov nezaručí, že útočník nebude môcť v hre $\text{RUL}_{\mathcal{H}^r}$ uspieť s vysokou pravdepodobnosťou. Stačí napríklad, aby v niektorom riadku boli všetky prvky veľmi blízke nule, a jeden veľmi blízky 1.

Rozdiel maximálneho a minimálneho prvku. Vlastnosť, ktorú využíval Thorp pri analýze svojho miešania. Zmenšením tohto rozdielu môžeme zmenšiť výhodu MR útočníka $\mathcal{A}$ v útoku opísanom v dôkaze Vety 2.3.1. Taktiež minimalizáciou tohto rozdielu môžeme zabezpečiť RUL bezpečnosť miešania: ak bude tento rozdiel menší-rovný $\mathrm{negl}(\log|K|)$, tak bude aj rozdiel každého prvku matice pravdepodobností a hodnoty $1/S$ menší-rovný $\mathrm{negl}(\log|K|)$ a podľa lemy 2.3.3 je takéto miešanie RUL-bezpečné.

Na druhej strane, požiadavka, aby rozdiel maximálneho a minimálneho prvku bol nanajvýš zanedbateľný, je pomerne silná a okrem toho, samotný rozdiel nemusí byť príliš dobrou charakteristikou náhodného miešania. Ako protipríklad uveďme maticu, v ktorej bude S^2-1 prvkov rovnakých a jeden sa od nich bude líšiť. Za predpoklady, že je tento jeden prvok a jeho odchýlka od ostatných rozoznateľná, poskytuje potenciálnu možnosť útoku na RUL-bezpečné miešanie.

Medián, modus. Dve vlastnosti, ktoré sa zvyknú používať pri charakteristike štatistického súboru prvkov. Pre naše potreby sú však nevhodné, pretože ľahko vieme skonštruovať maticu pravdepodobností takú, aby medián aj modus jej prvkov bol $1/S$, avšak môže obsahovať nuly aj jednotky a teda zjavne nebude RUL-bezpečná.

Rozptyl. Vlastnosť využívaná v štatistike na vyjadrenie variability rozdelenia hodnôt štatistického súbora. Čím menšia variabilita, tým menšia variabilita hodnôt, naopak, čím väčšia variabilita, tým sú odchýlky hodnôt od priemeru (očakávanej hodnoty) väčšie. Inak povedané, v našom prípade predstavuje rozptyl prvkov matice pravdepodobností mieru, nakoľko je táto matica blízka matici pravdepodobností U náhodného miešania. Túto mieru chceme skúmať pri jednotlivých miešaniach a ich RUL bezpečnosti.

2.3.2 Ideálne miešanie

V práci [47] a v časti 2.1.1 boli prezentované niektoré vlastnosti Thorpovho miešania a jeho matice pravdepodobnosti P. Napríklad r-krát opakované miešanie 2^r kariet má maticu pravdepodobnosti P^r zhodnú s uniformnou maticou U. Takéto miešanie je podľa lemy 2.3.3 RUL-bezpečné. V tejto časti sa pokúsime aplikovať koncept matice pravdepodobnosti a RUL-bezpečnosti aj na niektoré známe miešania predstavené v časti 2.2. Taktiež sa pokúsime nájsť miešania vhodné pre konštrukciu FPE, resp. matice pravdepodobností týchto miešaní, ktoré istým spôsobom prekonávajú kvalitu Thorpovho miešania. Vhodným kritériom môže byť počet kôl potrebných pre dosiahnutie RUL bezpečnosti.

Swap-or-not. Miešanie Swap-or-not predstavené v časti 2.2.2 je zdanlivo podobné Thorpovmu miešaniu v zmysle, že každú kartu x môže zobraziť na dve možné pozície $x, x \oplus k$. Problém však nastáva, akonáhle tento fakt chceme reprezentovať v matici pravdepodobnosti. Riadok x by mal obsahovať iba nulu a dve hodnoty $1/2$ v stĺpcoch x a $x \oplus k$. Hodnotu $x \oplus k$ však nepoznáme bez znalosti kľúča k pre dané kolo miešania.

Pokiaľ by sme chceli skúmať maticu pravdepodobností pre Swap-or-not šifrovanie (kľúče k pre jednotlivé kolá generujeme z hlavného kľúča K), táto matica pravdepodobností P by závisela od hodnoty K a teda by nebola jednoznačne definovaná. Taktiež matica pravdepodobností po r kolách miešania by sa nemusela dať vyjadriť ako P^r, pretože v každom kole môže byť hodnota k iná a tak by matica pravdepodobností po r kolách miešania vyzerala ako súčin r matíc pravdepodobností pre jednotlivé kolá. Druhá možnosť je skúmať iba (nereverzibilné) miešanie Swap-or-not, teda voliť v každom kole kľúče k náhodne. V tom prípade je pravdepodobnosť každého k rovnaká, preto pravdepodobnosť, že sa x zobrazí na x' je $1/(2S)$ ak $x' \neq x$ a $1/2 + 1/(2S)$ ak $x' = x$. Príklad matice pravdepodobnosti pre Swap-or-not miešanie 4 kariet:

$$P = \begin{pmatrix} \frac{5}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} \\ \frac{1}{8} & \frac{5}{8} & \frac{1}{8} & \frac{1}{8} \\ \frac{1}{8} & \frac{1}{8} & \frac{5}{8} & \frac{1}{8} \\ \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{5}{8} \end{pmatrix}.$$

Týmto spôsobom konštruované matice pravdepodobností však nie sú pre naše potreby vhodné, takéto (nereverzibilné) miešania s náhodným parametrom nie je možné priamo použiť na vytvorenie FPE schém, keďže k šifrovaniu nebudeme vedieť definovať efektívne dešifrovanie. Na rovnaký problém narazíme aj pri konštrukcií matíc pravdepodobností pre ostatné miešania predstavené v časti 2.2. Podobne ako pre Swap-or-not, ani pre zvyšné miešania (z nich odvodené FPE šifrovania) nevieme skonštruovať jednoznačné matice pravdepodobností nezávislé na kľúči.

Na základe vyššie uvedených dôvodov sa v nasledujúcom texte budeme venovať hľadaniu jednoduchších miešaní podobných Thorpovmu, v ktorých je dopredu pre každý prvok jednoznačne určiteľné, na ktoré prvky sa môže zobraziť. Ideálne, nech sa každý prvok môže zobraziť potenciálne práve na dva rôzne prvky, pričom pri konkrétnej realizácii miešania (šifrovania) sa medzi týmito prvkami zvolí pomocou jedného (pseudo)náhodného bitu.

Z vlastností uvedených v závere časti 2.3.1 sme zvolili rozptyl ako kritérium, na základe ktorého prehlásime maticu pravdepodobností za dobrú. Presnejšie povedané, budeme hľadať také matice pravdepodobností P, že pre pevne zvolený parameter označený ako ε a počet kôl r bude rozptyl prvkov matice P^r menší ako ε.

Začneme malými maticami s rozmermi $S \times S$ a pomocou počítačového programu budeme hľadať vhodné matice pre rôzne ε a r. Postupne generujeme matice, ktoré obsahujú v každom riadku a každom stĺpci práve dve hodnoty 1/2, teda matice pravdepodobností pre jednoduché miešania podobné Thorpovmu miešaniu, ako sme uviedli vyššie.

Matíc, ktoré obsahujú v každom riadku aj stĺpci práve dve hodnoty 1/2 je veľké množstvo, ako ukazuje jednoduchý dolný odhad počtu matíc P, ktorých dolná a horná polovica je rovnaká (ako v prípade Thorpovho miešania). V prvom riadku potrebujeme zvoliť dva nenulové prvky z S, v druhom dva prvky z $S-2$, atď. Takýchto možností je

$$\prod_{i=0}^{S/2} \binom{S-2i}{2} = \frac{S!}{2^{S/2}} \approx \sqrt{2\pi S}\left(\frac{S}{e\sqrt{2}}\right)^S.$$

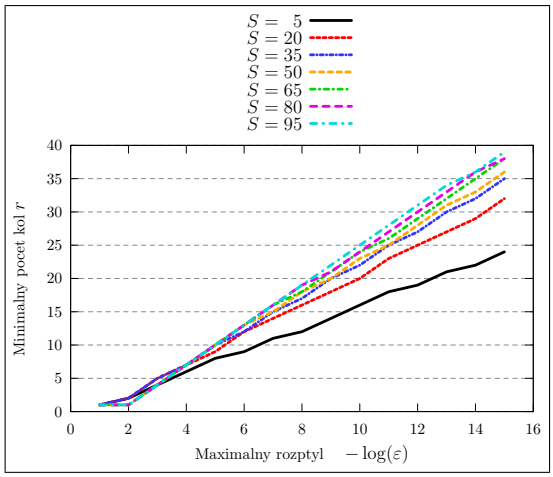
Posledný odhad vyplýva zo Stirlingovej aproximácie faktoriálu. Z tohto jednoduchého dolného odhadu vidíme, že už pre malé hodnoty S je hľadaných matíc neúnosne veľa, pre $S = 16$ je ich počet väčší ako 2^{36} a pre $S = 20$ väčší ako 2^{51}.

Z tohto dôvodu sme zvolili generovanie náhodných matíc spĺňajúcich danú vlastnosť namiesto úplného pre preberania celého priestoru matíc s dvoma prvkami 1/2 v každom riadku aj stĺpci. Pre všetky malé S sme takto vygenerovali 10000 takýchto matíc P rozmerov $S \times S$ a pre jednotlivé ε sme hľadali minimálne r také, aby rozptyl prvkov P^r bol menší ako ε.

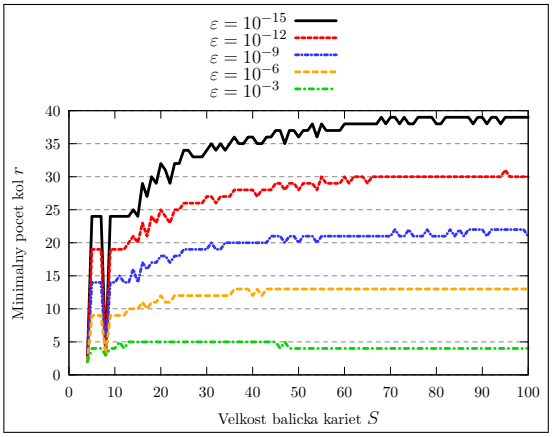
Program sme implementovali v jazyku Python s použitím knižnice Numpy na maticové výpočty a počítanie rozptylu prvkov. Grafické znázornenie závislosti minimálneho počtu kôl r od rozmeru matice S a požadovaného rozptylu ε je na obrázkoch 2.5. Pre porovnanie znázorňujeme aj Thorpovo miešanie. Časť získaných výsledkov vrátane počtu nájdených matíc dosahujúcich tieto minimálne hodnoty r uvádzame v prílohe A.1.

Experimentálne výsledky ukazujú, že žiadna z náhodne vygenerovaných matíc pravdepodobnosti nepredstavovala lepšie miešanie ako Thorpovo miešanie. Zároveň sa však niektoré miešania kvalitou (v zmysle minimálneho počtu kôl r) približujú k Thorpovmu miešaniu. Taktiež stoja za pozornosť lokálne minimá pre rozmery matíc, ktoré sú mocninami dvojky. Týmito pozorovaniami sa podrobnejšie zaoberáme v nasledujúcom texte.

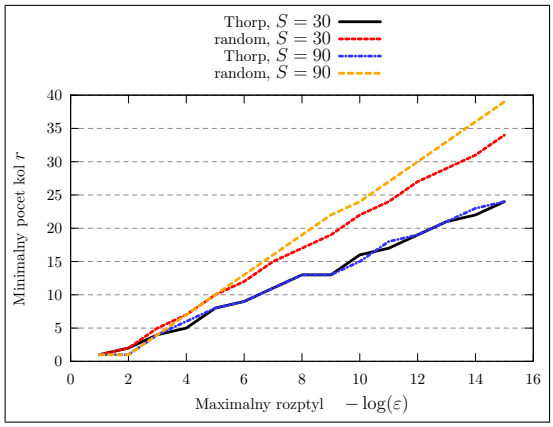
Lokálne extrémy. Pri umocňovaní matice pravdepodobnosti sú prvky matice P^r vždy súčtom súčinov niekoľkých prvkov 1/2 matice pravdepodobností P. Z tohto dôvodu sú nenulové prvky matíc P^r racionálne čísla s menovateľmi, ktoré sú mocniny dvojky. Ak by sme chceli dosiahnuť, aby $P^r = U$, tak nutná podmienka je, aby rozmery S matice P boli mocninou dvojky, keďže matica U obsahuje prvky $1/S$, ktorých menovateľ musí byť mocninou dvojky.



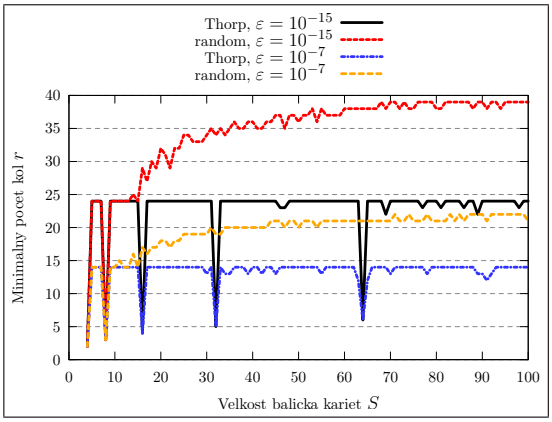
(a) Závislosť minimálneho r od ε



(b) Závislosť minimálneho r od S



(c) Závislosť minimálneho r od ε, porovnanie Thorpovho miešania s náhodnými miešaniami



(d) Závislosť minimálneho r od S, porovnanie Thorpovho miešania s náhodnými miešaniami

Obr. 2.5: Experimentálne zistené závislosti minimálneho počtu kôl náhodných miešaní

V práci [47] je uvedené, že nejaká mocnina matice pravdepodobnosti Thorpovho miešania je rovná uniformnej matici práve vtedy, keď je jej rozmer mocninou dvojky. Lokálne minimá pre matice s rozmermi tvaru 2^s sú teda očakávané.

Ideálne miešanie. Výsledky pokusu indikujú, že lepšie miešanie ako Thorpovo bude ťažké nájsť, ak vôbec existuje. Obzvlášť viditeľné je to najmä v oblasti lokálnych extrémov, kde Thorpovo miešanie jednoznačne dominuje. Toto pozorovanie formálne vyslovíme v nasledujúcej vete.

Veta 2.3.6 (Ideálne miešanie)**.** *Nech P je matica pravdepodobností rozmerov $2^s \times 2^s, s \geq 1$ obsahujúca iba prvky $1/2$ a 0, pričom v každom riadku aj každom stĺpci obsahuje práve dva prvky $1/2$. Označme r minimálny počet kôl potrebných na to, aby $P^r = U$, kde U je uniformná matica pravdepodobností obsahujúca iba prvky $1/2^s$. Potom platí $r \geq s$, pričom existuje miešanie, pre ktoré sa nadobúda rovnosť.*

Dôkaz. Na základe vlastností matice pravdepodobností Thorpovho miešania uvedených v časti 2.1.1 rovnosť $r = s$ platí pre Thorpovo miešanie. Ešte ukážeme, že pre všetky ostatné matice pravdepodobností P platí $r \geq s$. Využijeme pri tom pomocnú lemu.

Lema 2.3.7. *Nech P je matica pravdepodobností rozmerov $S \times S, S = 2^s, s \geq 1$ obsahujúca iba prvky $1/2$ a 0, pričom v každom riadku aj každom stĺpci obsahuje práve dva prvky $1/2$. Potom počet nenulových prvkov v každom riadku matice P^r pre $r \geq 1$ je nanajvýš 2^r.*

Dôkaz. Všetky prvky matice P^r sú nezáporné, keďže ich vieme napísať ako súčet súčinov prvkov matice P. Tvrdenie lemy dokážeme matematickou indukciou vzhľadom na parameter r.

1. Pre $r = 1$ lema triviálne platí, keďže podľa predpokladov obsahuje matica P v každom riadku 2^1 nenulových prvkov.

2. Predpokladáme, že matica P^r obsahuje najviac 2^r nenulových prvkov, Potom matica P^{r+1} obsahuje najviac 2^{r+1} nenulových prvkov.
Na základe definície násobenia matíc vyjadríme hodnoty jednotlivých prvkov matice $P^{r+1} = PP^r$. Označenie $A[i, j]$ použijeme pre prvok v i-tom riadku a j-tom stĺpci matice A.

$$P^{r+1}[i, j] = \sum_{k=1}^{S} P[i, k]P^r[k, j]. \tag{2.1}$$

Pretože v i-tom riadku matice P sú práve dva prvky nenulové (označme ich $P[i, k_{i_0}]$ a $P[i, k_{i_1}]$), môžeme rovnicu 2.1 upraviť na tvar:

$$P^{r+1}[i, j] = P[i, k_{i_0}]P^r[k_{i_0}, j] + P[i, k_{i_1}]P^r[k_{i_1}, j] = \frac{1}{2}\left(P^r[k_{i_0}, j] + P^r[k_{i_1}, j]\right). \tag{2.2}$$

Z indukčného predpokladu vieme, že riadky k_{i_0} aj k_{i_1} matice P^r obsahujú najviac 2^r nenulových (a teda kladných) prvkov. To znamená, že súčet $P^r[k_{i_0}, j] + P^r[k_{i_1}, j]$ v rovnici 2.2 môže nadobúdať kladnú hodnotu pre najviac $2 \cdot 2^r$ hodnôt $j \in 1, \ldots S$, a teda počet nenulových prvkov i-teho riadka matice P^{r+1} je nanajvýš 2^{r+1}.

Na základe oboch častí matematickej indukcie je lema dokázaná. □

Matica U obsahuje v každom riadku 2^s nenulových prvkov. Ak požadujeme, aby $P^r = U$, potom aj P^r musí obsahovať v každom riadku 2^s nenulových prvkov a podľa lemy 2.3.7 je preto $r \geq s$. □

Na tomto mieste ešte zdôrazníme, že z vety 2.3.6 vyplýva, že na dosiahnutie RUL bezpečnosti potrebujeme aspoň $\log_2 S$ kôl miešania. Zároveň z nášho experimentu sa zdá (obrázok 2.5(a)), že pre najlepšie (aj náhodné) miešania stačí $O(\log S)$ kôl miešania

na dosiahnutie RUL bezpečnosti. To aj skutočne platí, pretože Thorpovmu miešaniu naozaj stačí $O(\log S)$ kôl nielen na dosiahnutie RUL bezpečnosti, ale aj PRP bezpečnosti s použitím takmer S dopytov, ako sme uviedli v časti 2.1.3.

Náš experiment taktiež napovedá existenciu malého počtu miešaní rovnako dobrých ako Thorpovo (v zmysle minimálneho počtu kôl r), aspoň pre veľmi malé hodnoty S. Na overenie tejto hypotézy sme pre malé rozmery matíc $S \leq 8$ skúsili úplným preberaním nájsť všetky matice, ktoré na dosiahnutie požadovaného rozptylu prvkov potrebujú rovnaký počet kôl ako Thorpovo miešanie. Počty nájdených matíc uvádzame v tabuľke 2.2.

Optimálne matice rozmerov 4×4 je možné nájsť v prílohe A.2. Pri bližšom pohľade všetky tieto matice pripomínajú mierne modifikované Thorpovo miešanie. Pokiaľ ich interpretujeme ako miešania kariet, predstavujú rôzne doplnkové operácie k Thorpovmu miešaniu, napr. presun prvej karty na posledné miesto pred alebo po aplikovaní Thorpovho miešania, reverz jednej polovice kariet pred miešaním, reverz celého balíčka kariet.

S	**Počet optimálnych**[15]**matíc** P	**Počet všetkých matíc** P
4	12	90
5	240	2040
6	12240	67950
7	216720	3110940

Tabuľka 2.2: Optimálne matice pravdepodobností veľmi malých rozmerov

Z nájdených miešaní sa dá asi najzaujímavejšie interpretovať miešanie, ktorého maticu pre $S = 6$ vyzerá nasledovne:

$$P = \begin{pmatrix} \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & 0 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} \end{pmatrix}.$$

Z matematického hľadiska ide iba o transponovanú maticu Thorpovho miešania, avšak ako miešanie je jedna z možných interpretácii takáto: balíček kariet miešame rozdelením

[15]Počet matíc, ktoré dosiahli rozptyl menší ako 10^{-10} po rovnakom počte opakovaní, ako Thorpovo miešanie

na dve kôpky, pričom z balíčka vždy vezmeme vrchné dve karty a náhodne sa rozhodneme, ktorú kartu dáme na akú kôpku (do každej kôpky dáme práve jednu kartu). Nakoniec tieto dve kôpky spojíme do nového (zamiešaného) balíčka. Poznamenajme, že pri vhodnej voľbe kôpok pre jednotlivé karty je toto miešanie inverzným miešaním k Thorpovmu miešaniu.

Niektoré z nájdených optimálnych matíc pre $S = 5, 6, 7, 8$ sa nám však nepodarilo interpretovať pomocou jednoducho modifikovaného Thorpovho miešania, zároveň však rozmiestnenie jednotiek v uvedených maticiach nevykazovalo pravidelnosť, resp. vzorku, ktorú by sme mohli zovšeobecniť a na jej základe skonštruovať ideálne miešanie.

Kapitola 3

Súčasné trendy vo vývoji FPE

V tejto kapitole sa budeme zaoberať súčasným vývojom v oblasti šifier zachovávajúcich formát. Hoci sa v poslednom desaťročí problematike FPE venuje značná pozornosť, väčšina návrhov predstavuje skôr teoretické koncepty ako reálne použiteľné a preverené spôsoby šifrovania. Zmene tohto stavu by mala pomôcť štandardizácia schémy pre FPE, ktorá by bola preverená z bezpečnostného hľadiska a zároveň by bola aj dostatočne efektívna pre použitie v praxi. Aktuálne je v procese štandardizácie NISTom niekoľko návrhov FPE schém, ktoré bližšie popíšeme v časti 3.1.

Druhý aktuálny trend spočíva v optimalizácii existujúcich schém, prípadne v navrhovaní ich modifikácií, ktoré majú lepšie bezpečnostné parametre (napr. sú PRP-bezpečné pre väčší počet dopytov), prípadne efektívnejšie či už v zmysle počtu volaní PRNG (obyčajne využívajúcom AES), alebo v zmysle rýchlosti a času potrebného pre jedno šifrovanie. V tomto prípade zohráva svoju úlohu aj moderný hardvér, väčšina procesorov má zabudovanú podporu špeciálnych kryptografických inštrukcií. Optimalizáciám sa podrobnejšie venujeme v časti 3.2.

3.1 Štandardizácia schém

Už niekoľko desaťročí sa za akúsi autoritu v oblasti štandardizácie považuje americký úrad NIST, ktorý predstavuje úlohu Národného metrologického inštitútu patriaceho pod Ministerstvo obchodu USA. Oficiálnym cieľom NISTu je podpora inovácií Spojených štátov a priemyselnej konkurencieschopnosti vylepšovaním vedeckých meraní, štandardov a technológií s dôrazom na zvýšenie ekonomickej bezpečnosti a kvality života [36]. Mohutnému rozšíreniu algoritmov AES a predtým DES v značnej miere pomohlo práve štandardizovanie NISTom, resp. jeho predchodcom NBS [31, 37].

V roku 2013 vydal NIST návrh odporúčaní pre FPE mód blokových šifier pozostávajúci z troch metód označených ako FF1, FF2 a FF3 [38]. V súčasnosti je návrh

NISTu vo fáze verejného pripomienkovania a recenzovania.

Všetky tri metódy predstavujú konštrukciu celočíselných FPE schém pomocou Feistelovských sietí využívajúcich blokovú šifru AES na konštrukciu transformačných funkcií F_i. V tejto časti predstavíme jednotlivé navrhované metódy a ich stručnú históriu.

Parameter	**Vysvetlenie**
Kľúče $\mathcal{K}$	Neprázdna množina binárnych reťazcov
radix	Základ číselnej sústavy, v ktorej sú reprezentované jednotlivé správy. Správy pozostávajú zo znakov abecedy $\Sigma = \{0, \ldots, \text{radix} - 1\}$.
Dĺžky $\mathcal{N}$	Množina prípustných dĺžok správ, tvorí priestor formátov pre danú hodnotu radix.
Vylepšenia $\mathcal{T}$	Neprázdny priestor reťazcov. Bližší popis vylepšení bol uvedený v časti 1.2.1.
Sčítanie $\boxplus$	Spôsob realizácie operátora $\boxplus$ určuje, či čísla sčítavame po čísliciach (modulo radix) – hodnota parametra 0 – alebo po blokoch ako klasické čísla dĺžky N (modulo radix^N) – hodnota parametra 1.
Typ siete	Určuje typ použitej Feistelovskej siete: nevyváženú – hodnota parametra 1 – alebo alternujúcu sieť – hodnota parametra 2.
split(N)	Funkcia, ktorá určuje mieru nevyváženosti Feistelovskej siete, resp veľkosť ľavého bloku. Musí platiť: $N \in \mathcal{N}, 1 \leq \text{split}(N) \leq N/2$.
rnds(N)	Funkcia, ktorá určuje počet kôl vo Feistelovskej sieti. Argument $N \in \mathcal{N}$, a funkcia vracia párne kladné čísla.
F_i	Transformačné funkcie pre jednotlivé kolá i. Funkcie majú vstupné parametre $K \in \mathcal{K}$, $N \in \mathcal{N}, T \in \mathcal{T}$ a $B \in \Sigma^*$ a výstup je z množiny Σ^*, pričom ak i je párne alebo typ je nevyvážená sieť, tak $\lvert B\rvert = N - \text{split}N$ a $\lvert F_i(K, N, T, B)\rvert = \text{split}(N)$. V opačnom prípade je $\lvert B\rvert = \text{split}(N)$ a $\lvert F_i(K, N, T, B)\rvert = N - \text{split}(N)$.

Tabuľka 3.1: Parametre metódy FFX [3]

3.1.1 FF1: FFX[Radix]

Návrh metódy FF1 predstavili v roku 2010 Bellare, Rogaway a Spies pod názvom FFX [3] a jeho konkrétne dve inštancie pre šifrovanie binárnych a decimálnych reťazcov pod názvom FFX-A2 a FFX-A10. Na základe tohto návrhu o pár mesiacov neskôr predstavili novú inštanciu[1] FFX pod názvom FFX[Radix] [2]. FFX[Radix] predstavuje

[1] súbor parametrov FFX

vylepšenú verziu pôvodných súborov parametrov FFX-A2 a FFX-A10, podporuje väčšiu množinu základov číselnej sústavy (radix), väčšiu škálu dĺžok šifrovaných reťazcov a požaduje menší počet kôl šifrovania a dešifrovania.

Konštrukcia konkrétnej FPE schémy typu FFX je značne prispôsobiteľná širokou škálou parametrov: vylepšeniami, použitým typom Feistelovskej siete (nevyvážená, obr. 1.1(b), alebo alternujúca, obr. 1.1(c)), mierou nevyváženia Feistelovskej siete, počtom kôl, transformačnými funkciami F_i a „sčítaním“ vo Feistelovských sieťach, základmi číselnej sústavy a dĺžkami reťazcov. Prehľad jednotlivých parametrov FFX je uvedený v tabuľke 3.1. V tabuľke 3.2 uvádzame súbory parametrov pre inštancie FFX-A2, FFX-A10 a FFX[Radix] v štandardizačnom procese.

Parameter	**FFX-A2**	**FFX-A10**	**FFX[Radix]**
Kľúče $\mathcal{K}$	$\{0,1\}^{128}$	$\{0,1\}^{128}$	$\{0,1\}^{128}$
radix	2	10	radix $\in \{2,\ldots,2^{16}\}$
Dĺžky $\mathcal{N}$	$\{8,\ldots,128\}$	$\{4,\ldots,36\}$	$\{2,\ldots,2^{32}-1\}$ ak radix ≥ 10, $\{8,\ldots,2^{32}-1\}$ inak
Vylepšenia $\mathcal{T}$	$\{0,\ldots,255\}^{2^{64}-1}$	$\{0,\ldots,255\}^{2^{64}-1}$	$\{0,\ldots,255\}^{2^{32}-1}$
Sčítanie $\boxplus$	0 (po znakoch)	1 (blokové)	1 (blokové)
Typ siete	2 (alternujúca)	2 (alternujúca)	2 (alternujúca)
split(N)	$\lfloor N/2 \rfloor$	$\lfloor N/2 \rfloor$	$\lfloor N/2 \rfloor$
rnds(N)	12, ak $32 \geq N \geq 128$ 18, ak $20 \geq N \geq 31$ 24, ak $14 \geq N \geq 19$ 30, ak $10 \geq N \geq 13$ 36, ak $8 \geq N \geq 9$	12, ak $10 \geq N \geq 36$ 18, ak $6 \geq N \geq 9$ 24, ak $4 \geq N \geq 5$	10
F_i	výpis 3.1	výpis 3.2	výpis 3.3

Tabuľka 3.2: Súbory parametrov inštancií FFX [2, 3, 38]

```
def F_i(K, N, T, B):
  vers = 1; t = |T|_8
  P = [vers]^2 || [typ]^1 || [sčítanie]^1 || [radix]^1 || [N]^1 || [split(N)]^1 || [rnds(N)]^1
      || [t]^8
  Q = T || [0]^(-t-9 mod 16) || [i]^1 || 0^(64-|B|) || B
  Y = CBC-MAC_K(P || Q)
  if i mod 2 == 0:
    m = split(N)
  else:
    m = N - split(N)
  return Y[128-m:128]
```

Výpis 3.1: Transformačné funkcie F_i v FFX-A2 [3]

```
def F_i(K, N, T, B):
  vers = 1; t = |T|_8
  P = [vers]^2 || [typ]^1 || [sčítanie]^1 || [radix]^1 || [N]^1 || [split(N)]^1 || [rnds(N)]^1
      || [t]^8
  Q = T || [0]^(-t-9 mod 16) || [i]^1 || num_10(B)
  Y = CBC-MAC_K(P || Q)
  y' = num_2(Y[0:64]); y'' = num_2(Y[64:128])
  if i mod 2 == 0:
    m = split(N)
  else:
    m = N - split(N)
  if m ≥ 9:
    z = y'' mod 10^m
  else:
    z = (y' mod 10^(m-9)) · 10^9 + (y'' mod 10^9)
  return str^m_10(z)
```

Výpis 3.2: Transformačné funkcie F_i v FFX-A10 [3]

```
def F_i(K, N, T, B):
  vers = 1; t = |T|_8; β = ⌈N/2⌉; b = ⌈⌈β log_2(radix)⌉/8⌉; d = 4⌈b/4⌉
  P = [vers]^1 || [typ]^1 || [sčítanie]^1 || [radix]^3 || [rnds(N)]^1 || [split(N)]^1 || [N]^4
      || [t]^4
  Q = T || [0]^(-t-b-1 mod 16) || [i]^1 || [num_radix(B)]^b
  Y = CBC-MAC_K(P || Q)
  Y = (Y || AES_K(Y ⊕ [1]^16) || AES_K(Y ⊕ [2]^16) || AES_K(Y ⊕ [3]^16) ||...)[0:8(d+4)]
  y = num_2(Y)
  if i mod 2 == 0:
    m = ⌊N/2⌋
  else:
    m = ⌈N/2⌉
  z = y mod radix^m
  return str^m_radix(z)
```

Výpis 3.3: Transformačné funkcie F_i v FFX-Radix [2]

Vo výpisoch 3.1, 3.2 a 3.3 sú uvedené transformačné funkcie F_i použité v jednotlivých súboroch parametrov FFX-A2, FFX-A10 a FFX-Radix. Zápis $[X]^l$ predstavuje l-bajtovú reprezentáciu hodnoty X, zápis $|X|$ predstavuje dĺžku bitovej reprezentácie X a zápis $|X|_8 = \lceil |X|/8 \rceil$ predstavuje dĺžky bajtovej reprezentácie hodnoty X. Zápis $X[a:b]$ označuje číslo vytvorené z X vybratím číslic na pozíciách $a, \ldots, b-1$. Funkcia $\text{CBC-MAC}_K(X)$ predstavuje posledný blok blokovej šifry AES použitej v CBC móde s kľúčom K. Funkcia $\text{num}_{\text{radix}}(X)$ predstavuje číselnú hodnotu reťazca X zapísaného v sústave so základom radix. Naopak, funkcia $\text{str}^m_{\text{radix}}(X)$ vráti m-znakovú reprezentáciu čísla $X < \text{radix}^m$ v sústave so základom radix, teda napr. $\text{num}_2(1100) = 12$

a $\mathrm{str}_2^5(12) = 01100$.

Bezpečnosť. V práci [1] je ohraničená výhoda neadaptívneho SPI útočníka $\mathcal{A}$ na schémy typu FFX pomocou výhody PRF útočníka na transformačné funkcie F_i. Táto redukcia však nie je tesná. V práci [3] sú však uvedené aj výsledky viac vypovedajúce o bezpečnosti FFX-A2 a FFX-A10 pri praktickom použití. Označme $q_{\mathrm{radix}}(N)$ počet dopytov dPRP útočníka, pre ktoré je útočníkova výhoda nanajvýš 1/2. Autori odvodili nasledovné dolné ohraničenia pre niektoré hodnoty q_{radix} za použitia toho istého vylepšenia pre všetky dopyty: $q_2(32) > 580$, $q_2(50) > 37000$, $q_{10}(6) \geq 62$ a $q_{10}(16) > 77000$. Uvedené hodnoty predstavujú zatiaľ dolné odhady počtu povolených útočníkových dopytov, pre ktoré je dPRP bezpečnosť dokázaná. Častá zmena vylepšenia však pomôže tieto dolné odhady dramaticky zvýšiť.

Z dôvodu zabráneniu ľahkému útoku všeobecného typu Meet-in-the-middle na Feistelovské siete (opísaného napr. v [3]) požadujeme určitú minimálnu veľkosť priestoru správ, a teda takú minimálnu dĺžku textov min_l, aby $\mathrm{radix}^{\mathrm{min}_l} \geq 100$.

3.1.2 FF2: VAES3

Podrobná špecifikácia je uvedená v práci [48]. Názov VAES znamená Variabilný AES. Ako už bolo povedané v úvode tejto časti, aj metóda FF2 (s pôvodným názvom VAES3) je založená na Feistelovských sieťach a šifre AES ako transformačnej funkcii. VAES3 konkrétne predstavuje ďalší súbor parametrov FFX a prináša určité vylepšenia v porovnaní s FFX[Radix] najmä čo sa týka počtu volaní AES, bezpečnosti a životnosti kľúča. VAES3 totiž nepoužíva kľúč K priamo pri volaniach AES so vstupom odvodeným z parametrov F_i, ale najprv pomocou AES s kľúčom K a parametrami nezávislými od konkrétneho otvoreného textu odvodí podkľúč J, ktorý používa pri volaniach závislých od parametrov F_i.

Súbor parametrov VAES3 je uvedený v tabuľke 3.3 a transformačné funkcie F_i vo výpise 3.4. Funkcia len(X) vracia počet znakov reťazca X, teda ak je $X =$ „CAFEBABE“ v šestnástkovej sústave, tak $\mathrm{len}(X) = 8$. Inkrementovanie počítadla i na začiatku výpočtu F_i je z dôvodu kompatibility s FFX, kde sa používa počítadlo začínajúce od 0, pričom v pôvodnom návrhu VAES3 sa používalo počítadlo začínajúce od 1.

Na bezpečnosť metódy VAES sa vzťahujú výsledky uvedené pri FFX, keďže VAES predstavuje iba konkrétnu sadu parametrov schémy FFX.

Parameter	**VAES3**
Kľúče $\mathcal{K}$	$\{0,1\}^{128}$
radix	$\{2,\ldots,2^8\}$
Dĺžky $\mathcal{N}$	$\{2,3,\ldots,\max_N\}$, kde $\max_N = 2\lfloor 120/\log_2(\text{radix})\rfloor$
Vylepšenia $\mathcal{T}$	$\{0,\ldots,\text{radix}-1\}^{\max_T}$, kde $\max_T = \lfloor 104/\log_2(\text{radix})\rfloor$
Sčítanie $\boxplus$	1 (blokové)
Typ siete	2 (alternujúca)
split(N)	$\lfloor N/2\rfloor$
rnds(N)	10
F_i	výpis 3.4

Tabuľka 3.3: Súbory parametrov VAES3 [48]

```
def F_i(K, N, T, B):
  t = len(T); i += 1
  P = [radix]^1 || [t]^1 || [N]^1 || [num_radix(T)]^13
  J = AES_K(P)
  Q = [i]^1 || [num_radix(B)]^15
  Y = AES_J(Q)
  y = num_2(Y)
  if i mod 2 == 0:
    m = ⌊N/2⌋
  else:
    m = ⌈N/2⌉
  z = y mod radix^m
  return str^m_radix(z)
```

Výpis 3.4: Transformačné funkcie F_i vo VAES3 [48]

3.1.3 FF3: BPS

Metóda FF3 bola odoslaná NISTu v roku 2010 pod pôvodným názvom BPS [7]. Autori Brier, Peyrin a Stern v nej predstavili celočíselnú FPE schému BC podporujúcu rôzne základy číselných sústav a dĺžky do $\max_b = 2\lfloor \log_{\text{radix}}(2^{96})\rfloor$. Taktiež predstavili FPE schému BPS, ktorá predstavuje jemne modifikovaný[2] CBC mód blokovej šifry BC. BPS vďaka takejto konštrukcii podporuje vstupy do maximálnej dĺžky $2^{16}\max_b$. Faktor 2^{16} je určený na základe veľkosti použitého počítadla.

[2]namiesto operácie $\oplus$ sa používa sčítanie $\boxplus$ po znakoch modulo radix; ak nie je dĺžka vstupu násobkom $\max_b$, doplní sa potrebný počet znakov z posledného zašifrovaného bloku a po aplikovaní BC sa výsledok použije ako posledných $\max_b$ znakov výstupu, zároveň sa vylepšenia v BC modifikuje v každom kole pomocou 16-bitového počítadla.

V návrhu NISTu je z pôvodného návrhu BPS prevzatá schéma BC pod názvom FF3 [38]. BC predstavuje opäť alternujúcu Feistelovskú sieť. Na rozdiel od FFX však používa little-endian reprezentáciu čísel a pri konštrukcii transformačných funkcií F_i je možné použiť akúkoľvek štandardizovanú 128-bitovú blokovú šifru, nie iba AES. Použitú šifru budeme označovať CIPH a funkcia $\text{rev}(X)$ označuje znakový reverz čísla X, teda jeho big-endian reprezentáciu. V tabuľke 3.4 sú uvedené parametre FF3, resp. BC, vo výpise 3.5 transformačné funkcie F_i.

Parameter	**FF3**
Kľúče $\mathcal{K}$	kľúče pre zvolenú šifru CIPH
radix	$\{2, \ldots, 2^{16}\}$
Dĺžky $\mathcal{N}$	$\{2, \ldots, \max_b\}$, kde $\max_b = 2\lfloor \log_{\text{radix}}(2^{96}) \rfloor$
Vylepšenia $\mathcal{T}$	$\{0,1\}^{64}$
Sčítanie $\boxplus$	1 (blokové)
Typ siete	2 (alternujúca)
split(N)	$\lceil N/2 \rceil$
rnds(N)	8
F_i	výpis 3.5

Tabuľka 3.4: Parametre FF3 [7, 38]

```
def F_i(K, N, T, B):
  if i mod 2 == 0:
    m = ⌈N/2⌉
    W = T[32:64]
  else:
    m = ⌊N/2⌋
    W = T[0:32]
  P = rev([num_radix(rev(B))]^12) || W ⊕ rev([i]^4)
  Y = CIPH_K(P)
  y = num_2(rev(Y))
  z = y mod radix^m
  return rev(str^m_radix(z))
```

Výpis 3.5: Transformačné funkcie F_i v FF3 [7, 38]

Poznamenajme, že v pôvodnom návrhu [7] je šifrovanie BC definované ako funkcia s parametrami $F, \text{radix}, b, w, X, K, T$, kde F predstavuje transformačné funkcie F_i, b dĺžku bloku (teda $b \in \{2, \ldots, \max_b\}$), w počet kôl. Pri CBC konštrukcii BPS je však na predposlednom riadku v Algoritme 3 použité volanie BC s parametrom len miesto $\max_b$, kde len predstavuje dĺžku celého vstupu BPS a teda nemusí platiť $len \leq \max_b$ a šifrovanie $\text{BC}_{F,\text{radix},len,w}$ nemusí byť definované. Pravdepodobne ide o preklep a namiesto

len má byť použité $\max_b$. Taktiež sa tento preklep vyskytuje aj v Algoritme 4 pre dešifrovanie BPS.

Výhodou FF3 v porovnaní s FF1 je menší počet volaní AES. Značnú nevýhodu pri implementácii však predstavujú testovacie vektory, ktoré na rozdiel od FF1 a FF2 schém nie sú dostupné online na webe NISTu [35].

3.2 Optimalizácia schém

V priebehu posledných desiatich rokov bolo predstavených viacero FPE schém, reprezentatívnu časť z nich sme uviedli aj v tejto práci. Aj v súčasnosti sa objavujú práce prinášajúce nové schémy, avšak aktuálne návrhy vychádzajú vždy z rovnakých princípov (Feistelovské siete, miešania kariet) a častokrát predstavujú iba vylepšenie už existujúcich schém. Ako príklad môžeme uviesť vylepšenie návrhu Mix-and-cut (časť 2.2.3) návrhom Sometimes-recurse (časť 2.2.4), pričom oba návrhy sú založené na dobrých vlastnostiach miešania Swap-or-not (časť 2.2.2).

Súčasné trendy v optimalizácii existujúcich schém môžeme rozdeliť do dvoch hlavných skupín: zlepšovanie bezpečnosti a rýchlosti šifrovaní. Jednotlivým skupinám sa venujeme nižšie.

3.2.1 Bezpečnosť

Pre zvýšenie bezpečnosti FPE schém sa používa viacero techník, s niektorými sme sa už stretli pri miešaniach a šifrovaniach predstavených v tejto práci, aj keď sme to nie vždy explicitne zdôraznili. V tejto časti uvedieme jednotlivé techniky aj príklady ich použitia.

Zvýšenie počtu kôl. Asi najjednoduchšou metódou zvýšenia bezpečnosti je zvýšenie počtu kôl šifrovania (miešania). Pri popise bezpečnosti Swap-or-not šifrovania sme uvádzali výsledky z práce [20], v ktorej autori odvodili výhody nPRP a dPRP útočníkov s použitím najviac q dopytov. V oboch odhadoch vystupoval počet kôl šifrovania, pričom so vzrastajúcim počtom kôl klesala výhoda útočníkov.

Pre malé domény je taktiež vhodné primerane zvýšiť počet kôl ako prevenciu pred Meet-in-the-middle útokom[3] [3]. Autori návrhu FFX s týmto počítali a pri parametroch FFX-A2 a FFX-A10 bol pre krátke texty zvolený väčší počet kôl šifrovania (tabuľka 3.2).

[3]resp. zvýšeniu jeho časovej náročnosti tak, aby bolo pre útočníka jednoduchšie vyskúšať všetky kľúče úplným preberaním

Viacnásobné šifrovanie. V časti 2.2.3 využívalo šifrovanie Mix-and-cut princíp opakovaného šifrovania niektorých častí vstupného textu. Tento spôsob konštrukcie miešania, resp. šifrovania nazvali autori Cencúľ a je graficky znázornený na obr. 2.3. Pomocou cencúľovej konštrukcie dosiahli autori v práci [41] zvýšenie bezpečnosti, keď ako základ zvolili 2-PRS-bezpečné miešanie Swap-or-not a vytvorili PRP-bezpečné miešanie Mix-and-cut.

Použitie rôznych vylepšení. V závere časti 3.1.1 sme zdôvodnili potrebu častého používania rôznych vylepšení pri praktických aplikáciách FFX módu, kedy prezradenie už aj relatívne malého počtu dvojíc otvorených a šifrových textov s tým istým vylepšením môže viesť ku kompromitácii PRP bezpečnosti. Aj z tohto dôvodu zahrnuli autori v práci [7] použitie počítadla v schéme BPS, ktorá predstavuje CBC mód šifrovania BC. Pomocou počítadla modifikovali vylepšenie, ktoré vstupovalo do transformačných funkcií F_i v šifrovaní BC, čím spolu s meniacim sa indexom i v závislosti od konkrétneho kola Feistelovskej siete zabezpečili, že žiadna transformačná funkcia nebola použitá viackrát s rovnakými parametrami.

Premenovanie prvkov. Naor a Reingold v práce [30] ako prví navrhli aplikovanie párových nezávislých permutácií PwIP (Pair-wise Independent Permutation) na vstupný text do Feistelovskej siete a na jej výstup. Použitie permutácii predstavuje premenovanie prvkov vstupujúcich do siete a taktiež premenovanie prvkov na jej výstupe. Tieto permutácie sú ďalším parametrom FPE konštrukcií a prinášajú ďalšiu náhodnosť a zvyšujú bezpečnosť šifrovania. Vďaka pseudonáhodnému premenovaniu prvkov znižuje pravdepodobnosť úspešného použitia základnej diferenčnej a lineárnej kryptoanalýzy [12].

V pôvodnej práce Naora a Reingolda [30] boli PwIP skonštruované pomocou poľa $GF(2^n)$. Toto riešenie má jeden nedostatok, a to absenciu podpory ľubovoľných dĺžok, resp. celočíselných domén. Riešenie predstavili Dara a Fluhrer v práci [12]. Nimi navrhnutá schéma FNR (Flexible Naor and Reingold) používa na konštrukciu PwIP regulárne matice $N \times N$ nad poľom $GF(2)$.

Generátor náhodných čísel. S technologickým pokrokom v oblasti procesorov sa objavuje zaujímavá možnosť využitia hardvérového generátora náhodných čísel Intel DRNG (digital random number generator), ktorý je súčasťou procesorov Intel Ivy Bridge a novších. DRNG obsahuje zdroj entropie, ktorého výstup je následne spracovaný pomocou deterministického pseudonáhodného generátora. Prístup k tomuto hardvérovému DRNG je sprostredkovaný pomocou (assemblerovej) inštrukcie RDRAND,

ktorá naplní daný register 16-, 32- alebo 64-bitovou hodnotou a nastaví Carry Flag, ak v čase vykonania inštrukcie bola náhodná hodnota dostupná [10].

Použitie DRNG ako ďalšieho zdroja náhodnosti by mohlo priniesť zlepšenie bezpečnosti a rýchlosti, nakoľko prístup k tomuto generátoru je jednoduchý. Avšak je dobré nespoliehať sa iba na tento jeden zdroj náhodnosti a kombinovať jeho výstup s inými zdrojmi [19].

3.2.2 Efektívnosť, rýchlosť

Druhá kategória optimalizácií spočíva vo vylepšovaní efektívnosti FPE schém v zmysle počtu volaní PRNG a rýchlosti šifrovania.

Čiastočné šifrovanie. V časti 2.2.4 o Sometimes-recurse predstavovalo toto miešanie (šifrovanie) vylepšenú verziu Mix-and-cut. Autori Morris a Rogaway využili fakt, že ak nejaké miešanie je dostatočne dobré na to, aby pseudonáhodne zamiešalo ľubovoľných q kariet, tak prvých q kariet z balíčka už môžeme v Mix-and-Cut považovať za zamiešané a opakovane miešať iba zvyšné karty. Týmto ušetríme miešanie časti kariet a tiež volania PRNG.

Pri FPE šifrovaní sa tento fakt prejaví tak, že akonáhle sa karta presunie na niektorú z prvých q pozícií aktuálne miešaného balíčka(hodnota šifrového textu bude menšia ako q), už na ňu ďalej neaplikujeme šifrovanie. Morris a Rogaway vďaka tomuto vylepšeniu zmenšili počet volaní PRNG pri šifrovaní Sometimes-recurse na odmocninu z počtu volaní pri šifrovaní Mix-and-cut.

Hardvérová podpora šifrovania. Moderné procesory od Intelu a AMD podporujú hardvérové šifrovanie prostredníctvom inštrukcií AES, resp AES-NI (AES New Instructions). Použitím hardvérového šifrovania je možné značne urýchliť FPE šifrovanie pri praktickom nasadení. Využili to aj Data a Fluhrer v práci [12] pri implementácii FNR, ktorej sa budeme venovať v časti 4.1.3.

Ďalším príkladom použitia AES inštrukcií je práca Stefanova a Shi [46], v ktorej predstavili FPE schému asymptoticky horšiu ako Recursive-merge (časť 2.2.1), avšak vďaka menším nárokom[4] na PRNG a použitiu hardvérového šifrovania AES dosiahli 1000-8000 zrýchlenie v porovnaní s Recursive-merge pre domény s veľkosťou do 2^{31}.

[4]bez potreby hypergeometrického rozdelenia oproti Recursive-merge

Kapitola 4

Použitie FPE v praxi

Doteraz sme sa venovali najmä teoretickým aspektom šifrovaní zachovávajúcich formát. V kapitole 3 sme uviedli návrhy FPE schém, ktoré sú v súčasnosti v procese štandardizácie NISTom. V tejto kapitole sa venujeme aktuálnym možnostiam použitia FPE v praxi. V časti 4.1 predstavujeme v súčasnosti dostupné implementácie aj s ich stručným popisom. V časti 4.2 sa venujeme vlastnej implementácii FPE v jazyku Python s podporou rôznych formátov.

4.1 Dostupné implementácie

Súčasné implementácie interne využívajú jednu z troch spomínaných FPE schém[1]: FFX (časť 3.1.1), BPS (časť 3.1.3) a FNR (spomínaná v časti 3.2.1). Dostupné implementácie pochádzajú od komerčných firiem, komunity open-source vývojárov aj od jednotlivcov. V tejto časti prinášame ich zoznam s krátkym popisom a zhodnotením ich výhod a nevýhod.

4.1.1 FFX

FFX je konštrukcia predstavená v práci [3]. Niekoľko súborov parametrov pre túto konštrukciu je v súčasnosti v procese štandardizácie NISTom, ako sme uviedli v časti 3.1. Nasledujúce implementácie vychádzajú buď priamo z FFX konštrukcie, alebo z konštrukcií veľmi podobných FFX, prípadne jej predchodcov FE1 a FE2 predstavených v práci [1]. FE1 predstavuje FFX schému s nevyváženou Feistelovskou sieťou a FE2 s alternujúcou Feistelovskou sieťou.

[1] prípadne veľmi podobnú schému

HP FPE

Proprietárna implementácia firmy Hewlett-Packard [21], využíva schému podobnú FFX[2] v kombinácii s AES-256, ktorá je odvodená od návrhu Spiesa z roku 2006 chráneného US patentom 7,864,952 [22, 39].

Botan

Botan [5] je kryptografická knižnica pre C++ distribuovaná pod BSD-2 licenciou. Autori zvolili ako základ pre implementáciu FPE schémy schému FE1.

Knižnica poskytuje dve funkcie s podporou FPE, `fe1_encrypt(N, X, K, T)` a `fe1_decrypt(N, X, K, T)`, kde formát N určuje diel domény $\mathcal{X}_N = [N]$. Implementácia umožňuje použiť $N \leq 2^{128}$.

Podľa oficiálnej dokumentácie [6] knižnica Botan vo vývojovej vetve od verzie 1.11.14, ktorá vyšla vo februári 2015, podporuje aj Python. Zatiaľ však pre Python nie je dostupná kompletná funkčnosť knižnice a rozhranie FPE funkcií v čase písania práce ešte nie je implementované.

DotFPE

DotFPE [11] je knižnica poskytujúca FPE pre platformu .NET, využíva portovaný kód pre FPE z knižnice Botan a je distribuovaná pod BSD licenciou. Knižnica teda poskytuje rovnaké funkcie s podporou FPE ako knižnica Botan. Podľa domovskej stránky je DotFPE projektom jednotlivca, o ktorom sa nám nepodarilo zistiť viac, a má ambíciu byť prvou rozšírenou implementáciou FPE na platforme .NET. Avšak projektu chýba akákoľvek dokumentácia a príklady použitia a podľa aktivity v diskusii sa už autor tomuto projektu nevenuje.

JavaFPE

Knižnica JavaFPE [42] vznikla portovaním knižnice DotFPE do jazyka Java, je projektom jednotlivca rovnako ako DotFPE a rovnako je distribuovaná pod BSD licenciou. Je to mladý projekt, na GitHube sa objavil v decembri 2014. Stránka projektu však na rozdiel od DotFPE obsahuje príklad použitia aj testy. O autorovi sa nám viac zistiť nepodarilo.

libFFX

Knižnica libFFX [15] je implementácia FFX módu v jazyku Python distribuovaná pod licenciou GPLv3. Pochádza z prvej polovice roku 2014, jej autorom je Kevin P.

[2] alternujúca Feistelovská sieť s voliteľnými parametrami

Dyer, Ph.D. študent Portlandskej štátnej univerzity a softvérový inžinier spolupracujúci napr. na projektoch Tor, uProxy, fteproxy [14]. Stránka projektu obsahuje príklady použitia[3], unit testy aj výkonové testy. Zároveň knižnica bola otestovaná pomocou oficiálnych testovacích vektorov pre mód FFX.

Knižnica poskytuje triedu `FFXEncrypter(K, radix)` s dvoma metódami pre FPE, `encrypt(T, X)` a `decrypt(T, X)`. Implementácia umožňuje použiť základ radix z množiny $\{2, \ldots, 62\}$ a správy s dĺžkami z množiny $\{2, \ldots, 128\}$.

4.1.2 BPS

Miracl

Miracl (Multiprecision Integer and Rational Arithmetic C Library) Crypto SDK [9] je kryptografická knižnica pre jazyk C, často používaná v embedded zariadeniach, mobilných aplikáciách a SCADA systémoch. Knižnica Miracl je vyvíjaná spoločnosťou Certivox a je dostupná pod komerčnou licenciou aj pod licenciou AGPLv3.

Knižnica obsahuje aj podporu pre FPE, implementovaný je pôvodný návrh BPS [7], ktorý je základom pre súčasný návrh FF3 v procese štandardizácie. Knižnica poskytuje dve funkcie pre FPE, `FPE_encrypt(radix, K, TL, TR, X, N)` a `FPE_decrypt(radix, K, TL, TR, X, N)`, kde TL a TR sú ľavá a pravá časť vylepšenia a N je dĺžka správy X.

4.1.3 FNR

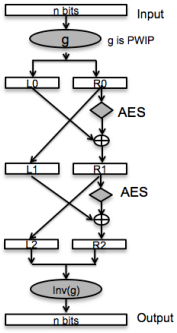
FNR je FPE schéma od firmy Cisco z roku 2014 [12]. Schéma FNR je založená na konštrukcii NR, ktorú použili Naor a Reingold v práci [30]. Vstup a výstup z Feistelovskej siete sa transformujú pomocou PwIP. Autori FNR vylepšili konštrukciu PwIP pomocou regulárnych matíc nad poľom $GF(2)$, čím dosiahli podporu pre ľubovoľné dĺžky bitových reťazcov. Princíp FNR je demonštrovaný na obrázku 4.1.

V práci [12] autori vytvorili transformačné funkcie $F_i(K, T, B)$ pomocou blokovej šifry AES, v ktorej kľúčom K zašifrovali text $i \,||\, T \,||\, B$, ako sčítanie $\boxplus$ použili klasický xor po bitoch a nevyváženú Feistelovskú sieť so 7 kolami a parametrom split $= \lfloor n/2 \rfloor$, kde n predstavuje dĺžku binárneho bloku. Schéma FNR podporuje formáty (veľkosti dielov domén) $N = 2^n$ pre $n \in \{2, \ldots, 128\}$.

libFNR

Knižnica libFNR je implementácia schémy FNR od firmy Cisco distribuovaná pod licenciou GPLv2.1. Vzhľadom na to, že návrh FNR je dosť mladý a nie je v štandar-

[3] príklady v example.py ani v README.md však nezodpovedajú aktuálnemu rozhraniu knižnice libFFX



Obr. 4.1: Dvojkolová konštrukcia NR [30, 12]

Zdroj obrázka: [12]

dizačnom procese, firma Cisco v súčasnosti označuje túto implementáciu ako experimentálnu [24]. Vývoj stále prebieha, posledný commit obsahujúci nové testy a opravy drobných preklepov v komentároch je z marca 2015. Na GitHube sú dostupné testy, ukážky kódu aj dokumentácia.

Veľkosť bloku (formát) sa nastavuje parametrom pri odvádzaní kľúča pomocou `FNR_expand_key(K,k,n)`, kde n je veľkosť bloku a k je veľkosť AES kľúča, ktorý sa vygeneruje z K. Knižnica poskytuje funkcie pre FPE `FNR_encrypt(K',T,P,C)` a `FNR_decrypt(K',T,C,P)`, kde K' je odvodený kľúč, P je otvorený text a C je šifrový text.

Za zmienku stojí implementácia Feistelovskej siete. Na rozdiel od tradičných implementácií, v ktorých sa blok rozdeľuje na dve časti a tie sa po transformácii medzi sebou vymieňajú, implementácia libFNR využíva na rozlíšenie blokov bity na párnych a nepárnych pozíciách a pomocou bitovej masky určuje, ktorá časť je v i-tom kole vstupom do F_i a ktorá časť sa xoruje s výstupom F_i.

jFNR

Knižnica jFNR je rozšírenie libFNR pre jazyk Java [23]. Využíva Java Native Access pre prístup k natívnej knižnici libFNR. Kľúč, vylepšenie a veľkosť bloku sa nastavujú pri vytváraní objektu `FNR`, ktorý poskytuje dve metódy pre FPE, `encrypt(P)` a `decrypt(C)`, kde P a C sú polia bytov.

Knižnica navyše obsahuje triedu `FNRUtils`, ktorá poskytuje statické metódy pre podporu IPv4 formátu[4] (prístup Ohodnoť a šifruj) a generovanie kryptograficky bezpečných pseudonáhodných bytov (použitých ako soľ) a AES kľúčov.

[4]formát FPE schémy, ktorému zodpovedajúci diel domény tvoria IPv4 adresy

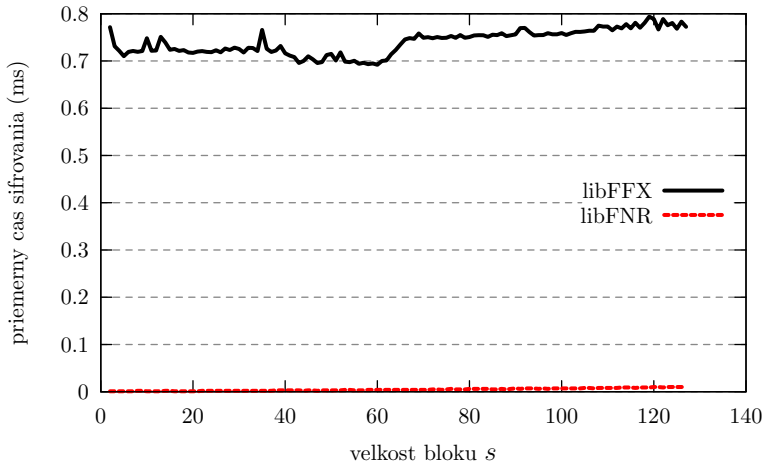
Zhrnutie. Prehľadný sumár podpory formátov pre jednotlivé implementácie uvádzame v tabuľke 4.2, do ktorej je zahrnutá aj vlastná implementácia pyFNR.

4.2 Vlastná implementácia

V časti 4.1 sme predstavili v súčasnosti dostupné implementácie FPE. Pre jazyk Python je dostupná iba jedna implementácia, libFFX. LibFFX neposkytuje žiadnu podporu bežných formátov. Taktiež je samotná Feistelovská sieť implementovaná v čistom Pythone, čo má negatívny vplyv na rýchlosť v porovnaní napr. s knižnicami implementovanými v C/C++. Porovnanie rýchlosti s knižnicou libFNR je na obr. 4.2. Ďalšia nevýhoda libFFX je absencia podpory pre Python3.

Rozhodli sme sa implementovať vlastnú knižnicu pre FPE v jazyku Python[5], ktorá by odstránila oba vyššie spomínané nedostatky libFFX. Naša knižnica by mala obsahovať podporu bežných formátov, mala by poskytovať možnosť ľahkého doplnenia vlastných formátov, a jej základná funkcionalita by mala využívať natívny kód z dôvodu zvýšenia rýchlosti.

Inšpirovali sme sa najmä knižnicou libFNR a jej rozšírením jFNR. Naša implementácia predstavuje rozšírenie libFNR pre jazyk Python, pridáva podporu formátov pre celočíselné domény $[N]$, kde $N \in \{2, \ldots, 2^{128} - 1\}$, ďalej podporu niektorých bežných formátov (IPv4, CCN, rodné čísla, evidenčné čísla vozidiel) a tiež možnosť podpory formátu pre ľubovoľný regulárny jazyk opísaný pomocou deterministického konečného automatu.



Obr. 4.2: Porovnanie výkonu knižníc libFFX a libFNR

[5]s podporou pre Python vo verziách 2.6 až 3.4

4.2.1 pyFNR

Implementácia

Naša knižnica pyFNR[6] je implementovaná v jazyku Python. Obsahuje dve triedy, FNR, FNR2 a modul Util pre podporu bežných formátov (rank, unrank funkcie).

Trieda FNR poskytuje metódy pre šifrovanie/dešifrovanie čísel (typ `int`), reťazcov (typ `str`), bajtových polí (typ `bytearray`) a znakových polí z jazyka C (typ `c_char_Array`). Táto trieda interne využíva funkcie z libFNR a libssl (OpenSSL), ku ktorým pristupuje pomocou knižnice ctypes jazyka Python. Nižšie uvádzame základný popis implementácie metód triedy FNR, podrobnejší popis je možné nájsť v dokumentácii pomocou funkcie `help(pyFNR.FNR)`.

- `FNR(key,tweak,block_size,salt)` – konštruktor triedy FNR vygeneruje kľúč pre FNR z parametrov `key` a `salt` pomocou funkcie `PKCS5_PBKDF2_HMAC_SHA1()` z OpenSSL a následne inicializuje knižnicu libFNR pre použitie s formátom zodpovedajúcim priestoru textov $[2^{\texttt{block\_size}}]$.
- `encrypt_raw(plaintext, ciphertext), decrypt_raw(ciphertext, plaintext)` – metódy pre šifrovanie/dešifrovanie znakových polí jazyka C volajú priamo príslušné funkcie `FNR_encrypt(K',T,P,C)` a `FNR_decrypt(K',T,C,P)` z libFNR.
- `encrypt_bytes(plaintext), decrypt_bytes(ciphertext)` – metódy určené pre bajtové polia konvertujú vstup na znakové pole jazyka C a výstup príslušnej funkcie z libFNR konvertujú naspäť na bajtové pole.
- `encrypt_int(plaintext), decrypt_int(ciphertext), encrypt_str(plaintext), decrypt_str(ciphertext)` – metódy určené pre čísla a reťazce konvertujú vstup na bajtové pole, zavolajú príslušnú funkciu `*_bytes` a výstup konvertujú na číslo, resp. reťazec.
- `close()` – uvoľní zdroje využívané knižnicou libFNR

Celá implementácia triedy FNR je pokrytá testami využívajúcimi knižnicu unittest jazyka Python. Testy konverzí medzi číselnými a reťazcovými typmi a bajtovými poliami sú implementované v triede TestConversions, testy šifrovania a dešifrovania sú implementované v triede TestFNRCrypt. Spustenie konkrétnych testov je možné pomocou príkazu `python tests.py [NazovTriedy]`, resp. `python3 tests.py [NazovTriedy]`. Ukážka použitia triedy FNR je vo výpise B.1

[6] Zdrojové kódy sú dostupné na `https://github.com/laciKE/pyFNR`

Optimalizácia

Využitím natívnej knižnice libFNR sme dosiahli značne rýchle šifrovanie/dešifrovanie bajtových a znakových polí, avšak pomerne veľa času pri šifrovaní/dešifrovaní celých čísel venujeme konverzii čísel na ich bajtovú reprezentáciu vo formáte little-endian, ako ukazuje výstup z profilera pre pokusný test šifrovania a dešifrovania 100 000 náhodných 128-bitových čísel:

```
$ python2 -m cProfile benchmark.py | grep "[12]00000.*__init__\.py\|ncalls"
   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
   100000    2.290    0.000    3.851    0.000 __init__.py:125(encrypt_bytes)
   100000    2.273    0.000    3.841    0.000 __init__.py:142(decrypt_bytes)
   100000    0.904    0.000    8.519    0.000 __init__.py:199(encrypt_int)
   100000    0.893    0.000    8.494    0.000 __init__.py:215(decrypt_int)
   200000    3.650    0.000    4.056    0.000 __init__.py:250(_int_to_bytes)
   200000    3.011    0.000    3.468    0.000 __init__.py:266(_bytes_to_int)
```

Z výstupu profilera vidíme, že v Pythone2 aj v Pythone3 trvá konverzia medzi číslami a bajtovými poliami[7] takmer rovnako dlho, ako samotné šifrovanie a dešifrovanie bajtových polí.

Pôvodná konverzia čísel na bajtové polia bola implementovaná jednoduchým algoritmom pomocou binárnych posunov a súčinov s hodnotami `0xff`. Táto metóda sa ukázala ako neefektívna. Problém je, že samotný jazyk Python do verzie 3.2 neobsahuje funkcie na konverziu medzi číslami a ich bajtovými reprezentáciami pre viac ako 64-bitové čísla[8]. Druhý problém (ne)kompatibilita niektorých Pythonových funkcií a typov naprieč verziami, napr. `binascii.hexlify` vyžaduje argument typu `str` v Python2 a argument typu `bytes` v Python3. Takéto nekompatibility vyžadujú dodatočné konverzie medzi jednotlivými typmi, čo taktiež negatívne vplýva na rýchlosť konverzie.

Po preskúmaní a otestovaní viacerých rôznych riešení či už vlastných, alebo verejne dostupných sme zvolili pre konverziu bajtových polí na čísla a naopak riešenia na výpisoch 4.1 a 4.2, pričom sme sa snažili zohľadniť odlišnosti jednotlivých verzií Pythonu.

[7] `_bytes_to_int` a `_int_to_bytes`

[8] pre čísla typu `unsigned long long` existuje konverzia na ich reťazcovú reprezentáciu pomocou `struct.pack('<Q',intval)`, ale následne by sme potrebovali konvertovať reťazce na bajtové polia

```
def _int_to_bytes(self, intval):
    if sys.hexversion >= 0x03020000:
        return bytearray(int(intval).to_bytes(self.
            _block_size_bytes,
                                                byteorder='little
                                                   '))
    hexval = self._hex_format_string.format(intval) # "{0:0Nx}"
    if sys.hexversion >= 0x02070000:
        bytesval = bytearray.fromhex(hexval)
    else:
        # workaround for Python 2.6 unicode requirement
        bytesval = bytearray.fromhex(unicode(hexval))
    bytesval.reverse() # little endian
    return bytesval
```

Výpis 4.1: Konverzia celých čísel na bajtové polia

```
def _bytes_to_int(self, bytesval):
    if sys.hexversion >= 0x03020000:
        return int.from_bytes(bytesval, byteorder='little')
    bytesval_copy = bytearray(bytesval)
    bytesval_copy.reverse() # little endian
    if sys.hexversion >= 0x03000000:
        strval = binascii.hexlify(bytesval_copy)
    else:
        # workaround for python 2.x string/read-only buffer
            argument
        strval = binascii.hexlify(bytes(bytesval_copy))
    return int(strval,16)
```

Výpis 4.2: Konverzia bajtových polí na celé čísla

Táto optimalizácia sa ukázala ako úspešná, urýchlila konverzie približne trojnásobne. V tabuľke 4.1 sú uvedené priemerné časy 100 000 volaní jednotlivých metód[9].

Z našich meraní vyplynulo, že použitie natívnej metódy `int.to_bytes` v Pythone 3.2 a novšom je mierne pomalšie, ako nami zvolená konverzia pomocou hexadecimálnych reprezentácií. Z tohto dôvodu vo výslednej knižnici nepoužívame pre Python 3.2 a novší natívnu metódu `int.to_bytes`, čím sme čas behu konverzie skrátili na úroveň dosahovanú pri Pythone 3.1. Použitie natívnej metódy `int.from_bytes` bolo podľa očakávaní najrýchlejšie.

[9]metódy s príponou „2“ sú pôvodné implementácie, metódy bez číselnej prípony aktuálne implementácie

Metóda	Python 2.6	Python 2.7	Python 3.1	Python 3.4
`_int_to_bytes`	0,00365 ms	0,00367 ms	0,00321 ms	0,00432 ms
`_int_to_bytes2`	0,01579 ms	0,01858 ms	0,01340 ms	0,01356 ms
`_bytes_to_int`	0,00391 ms	0,00589 ms	0,00406 ms	0,00259 ms
`_bytes_to_int2`	0,01143 ms	0,01212 ms	0,01039 ms	0,01065 ms
`encrypt_bytes`	0,02041 ms	0,02081 ms	0,02072 ms	0,02398 ms
`decrypt_bytes`	0,02047 ms	0,02098 ms	0,02079 ms	0,02381 ms

Tabuľka 4.1: Optimalizácia konverzií: priemerné časy behu metód

Rozšírenie na ľubovoľné číselné domény

Knižnica libFNR a aj trieda FNR v knižnici pyFNR podporujú iba diely domény tvaru $[2^s - 1]$, kde $s \in \{1, \ldots, 128\}$ je veľkosť bloku. Tento nedostatok sme odstránili v triede FNR2, ktorá podporuje ľubovoľné celočíselné diely domény $[S]$, kde $S \in \{2, \ldots, 2^{128} - 1\}$. Nižšie uvádzame základný popis implementácie metód triedy FNR2, podrobnejší popis je možné nájsť v dokumentácii pomocou funkcie `help(pyFNR.FNR2)`.

- `FNR2(key,tweak,domain,salt)` – konštruktor triedy FNR2, z parametra `domain` = S vypočíta s a vytvorí inštanciu triedy FNR, ktorú interne využíva.
- `encrypt(plaintext)`, `decrypt(ciphertext)` – metódy pre šifrovanie/dešifrovanie čísel z množiny [textttdomain], sú implementované pomocou cyklickej prechádzky.
- `close()` – uvoľní zdroje využívané triedou FNR.

Trieda FNR2 interne využíva objekt triedy FNR pre šifrovanie a dešifrovanie celých čísel z domény $[2^s - 1]$, kde $s = \lceil \log_2 S + 1 \rceil$ pomocou metódy Cyklickej prechádzky predstavenej v časti 1.3.1. Ukážka tejto metódy je vo výpise 4.3. Kvôli použitému while cyklu by sa mohlo zdať, že šifrovanie bude trvať príliš dlho, avšak ako sme zdôvodnili v časti 1.3.1, táto metóda je korektná, skončí v konečnom počte krokov a navyše očakávaný počet volaní `_fnr.encrypt_int` je v našom prípade $(S+1)/2^s \leq 2$.

```
def encrypt(self, plaintext):
    ciphertext = self._fnr.encrypt_int(plaintext)
    while (ciphertext > self.domain):
        ciphertext = self._fnr.encrypt_int(ciphertext)
    return ciphertext
```

Výpis 4.3: Šifrovanie pomocou metódy cyklickej prechádzky

Podpora bežných formátov

Modul Util obsahuje triedy pre podporu bežných formátov[10] (regulárnych jazykov skonštruovaných pomocou DKA) a tiež triedy pre ľahké rozšírenie o ďalšie formáty pomocou DKA.

Podpora formátov je založená na prístupe Ohodnoť a šifruj predstaveného v časti 1.3.2 pomocou DKA pre regulárne jazyky. Modul Util obsahuje triedu DFA, ktorá z parametrov `Q`, `Sigma`, `delta`, `q0` a `F` vytvorí DKA zo stavmi reprezentovanými celými číslami a ktorý je vhodný pre použitie v algoritmoch uvedených vo výpise 1.2. Poznamenajme, že trieda DFA umožňuje ako parameter `delta` použiť buď slovník, v ktorom sú kľúče dvojice (stav, symbol) a hodnoty sú stavy, alebo funkciu, ktorá má argumenty stav, symbol a vracia stav. Modul obsahuje ukážku použitia oboch možností.

Trieda FPE_Format obsahuje implementáciu algoritmu z výpisu 1.2 a slúži ako základ pre všetky ďalšie triedy implementujúce podporu formátov. Základný popis triedy uvádzame nižšie, viac informácii je v používateľskej dokumentácii pomocou funkcie `help(pyFNR.Util.FPE_Format)`.

- `FPE_Format(DFA, N)` – konštruktor triedy, pomocou algoritmu `buildTable(N)` z výpisu 1.2 vypočíta pre každý stav a dĺžku neprečítaného slova počet suffixov, ktoré DFA akceptuje. Inštancia tejto triedy zodpovedá jazyku $L_{DFA} \cap \Sigma^N$, teda slovám dĺžky N, ktoré akceptuje daný DFA.
- `get_words_count()` – vráti počet slov daného formátu, teda $S = |L_{DFA} \cap \Sigma^N|$.
- `rank(X)`, `unrank(c)` – funkcie pre ohodnotenie slov daného formátu, bijektívne zobrazenie jazyka na množinu $[S - 1]$.

V súčasnosti modul Util obsahuje podporu pre formáty IPv4, IPv6 (ohodnotenie je implementované pomocou funkcií z knižnice socket), ECV a LuhnR_N. Formáty IPv4 a IPv6 sú dostatočne známe, podrobnejšie vysvetlíme formáty ECV a LuhnR. Dokumentácia k jednotlivým formátom je dostupná prostredníctvom funkcie `help(pyFNR.Util.`*`format`*`)`.

ECV. Formát pre evidenčné čísla vozidiel so štandardnou logistikou na Slovensku, predstavuje reťazce dĺžky zložené z číslic a veľkých písmen abecedy, pričom prvé dva znaky sú identifikátor okresu, nasledujú 3 číslice a 2 písmená, napr. BA000AA. Konštruktor triedy ECV nemá žiadne parametre. V triede je zadefinovaný DKA pre formát ECV, ktorého prechodová funkcia `delta` je reprezentovaná slovníkom. Tento formát obsahuje triviálnu časť (trojčíslie a dve písmená) a netriviálnu časť (identifikátor okresu).

[10] ohodnotenie slov daného jazyka je realizované prostredníctvom funkcií `rank` a `unrank`

Myšlienka konštrukcie DKA pre netriviálnu časť spočíva v pamätaní si už prečítanej časti identifikátora v stave a použití pravidiel, ktoré dovoľujú prečítať iba znaky, ktoré spolu s už prečítanou časťou tvoria prefix platných identifikátorov. Myšlienka konštrukcie DKA pre triviálnu časť spočíva v počítaní dĺžky už prečítaného reťazca pomocou stavu, pričom na základe tejto dĺžky sa určuje, či sa na vstupe DKA očakáva číslica alebo písmeno.

LuhnR. Formát pre jazyk $Luhn_M^R$, do ktorého patria také slová, ktorých reverz má hodnotu Luhnovho kontrolného súčtu [25] rovnú M. Konštruktor triedy LuhnR má dva parametre, hodnotu kontrolného súčtu $M \in \{0, \ldots, 9\}$ a dĺžku slov N.

Tento formát je možné použiť pri konštrukcii FPE pre čísla kreditných kariet (CCN), v ktorom budú šifrované celé čísla kariet (použijeme `LuhnR(0, 16)`), prípadne pre šifrovanie časti čísel kreditných kariet (napr. necháme nezašifrované prvé 6-číslie použité ako identifikátor vydavateľa karty a posledné štvorčíslie častokrát používané na identifikáciu konkrétnej karty) – vtedy použijeme M rovné pôvodnej hodnote Luhnovej kontrolnej sumy pre šifrovanú časť karty a N rovné počtu číslic šifrovanej časti.

V trieda LuhnR je zadefinovaný DKA pre jazyk $Luhn_M^R$, ktorého prechodová funkcia je reprezentovaná funkciou jazyka Python. Myšlienka konštrukcie DKA spočíva v počítaní hodnoty Luhnovej kontrolnej sumy pomocou stavu. V stave si taktiež udržiavame informáciu o parite počtu prečítaných znakov. Množina stavov je teda $Q = [9] \times [1]$, pričom prechodová funkcia vyzerá nasledovne:

$$\delta((a,b),c) = \left(\left(a + c + (1-b)\left(c + \left\lfloor \frac{c}{5} \right\rfloor\right)\right) \mod 10, 1-b\right),$$

kde a predstavuje hodnotu Luhnovej sumy pre doteraz prečítanú časť vstupu, b udržiava informáciu o parite počtu doteraz prečítaných znakov a c je aktuálny znak na vstupe.

Ukážky použitia knižnice pyFNR na šifrovanie textov v rôznych formátoch uvádzame v prílohe B.

4.3 Porovnanie implementácii

V tejto časti prinášame porovnanie nami vytvorenej implementácie pyFNR z časti 4.2 a ostatných dostupných implementácií predstavených v časti 4.1.

4.3.1 Podpora formátov

Prevažná väčšina súčasných knižníc pre FPE podporuje iba formáty pre celočíselné domény, avšak žiadne špecifickejšie formáty navyše. Pre prehľadnosť sme podporu for-

mátov súčasných implementácii zhrnuli do tabuľky 4.2. Hodnota N/A značí verejne neprístupný údaj, resp. že sa nám údaj nepodarilo nájsť.

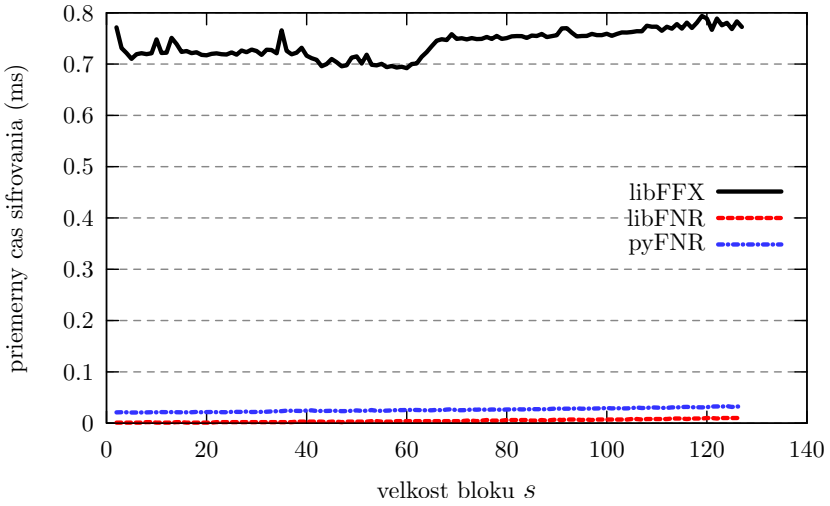
Knižnica	$[b^s - 1]$	$[S - 1]$	IPv4/IPv6	CCN	ostatné
HP FPE	áno, N/A	áno, N/A	N/A	áno	N/A
Botan	áno, $b = 2, s \leq 128$	áno, $S \leq 2^{128}$	nie	nie	nie
DotFPE	áno, $b = 2, s \leq 128$	áno, $S \leq 2^{128}$	nie	nie	nie
JavaFPE	áno, $b = 2, s \leq 128$	áno, $S \leq 2^{128}$	nie	nie	nie
libFFX	áno, $b \leq 62, s \leq 128$	nie	nie	nie	nie
Miracl	áno, $b \leq 2^{16}, s \leq 2\lfloor \log_b 2^{96} \rfloor$	nie	nie	nie	nie
libFNR	áno, $2 = 2, s \leq 128$	nie	nie	nie	nie
jFNR	áno, $2 = 2, s \leq 128$	nie	iba IPv4	nie	nie
pyFNR	áno, $2 = 2, s \leq 128$	áno	áno	áno	reg. jazyky

Tabuľka 4.2: Podpora formátov v súčasných implementáciách FPE

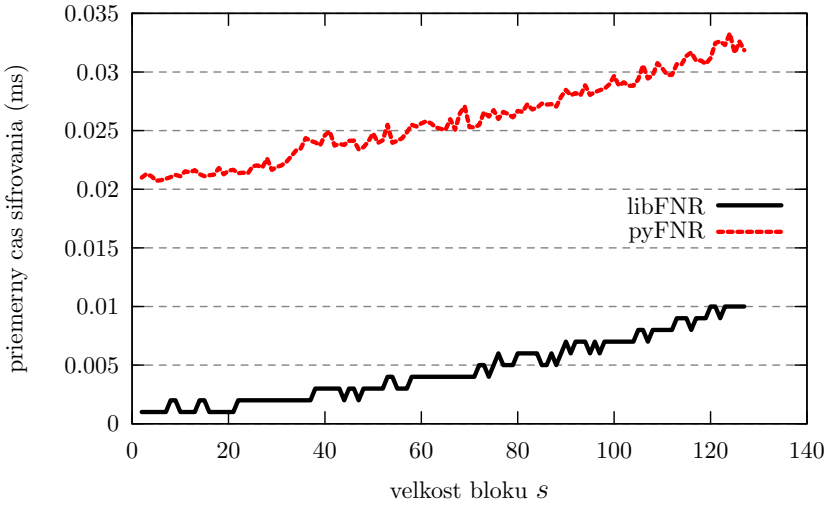
4.3.2 Výkon

Väčšina dostupných knižníc pre FPE je implementovaná v jazyku C, resp. C++ a využíva natívny strojový kód, čo im dáva určitú rýchlostnú výhodu v porovnaní s implementáciami v interpretovanom jazyku Python. Graf 4.3 porovnáva rýchlosti niekoľkých vybraných implementácií pri šifrovaní 1 000 náhodných celých čísel z množiny $[2^s - 1]$ pre rôzne s. Počas testovania sme zaznamenali nestabilitu knižnice libFFX (chyby, ktoré sa nám podarilo zreprodukovať, sme nahlásili autorovi).

Na základe našich meraní je možné usúdiť, že rýchlosť šifrovania našej knižnice pyFNR je v porovnaní s libFNR približne troj- až desaťnásobne pomalšia (pre väčšie bloky je rozdiel menší), avšak v porovnaní s knižnicou libFFX pre Python je pyFNR približne 30 až 40-krát rýchlejšia ako knižnica libFFX. Navyše v porovnaní s oboma uvedenými knižnicami naša knižnica pyFNR priamo podporuje šifrovanie celočíselných domén aj textových reťazcov.



(a) libFFX, libFNR a pyFNR



(b) libFNR a pyFNR

Obr. 4.3: Porovnanie rýchlosti vybraných knižníc

Záver

Šifrovania zachovávajúce formát sú pomerne mladou a rýchlo sa rozvíjajúcou oblasťou kryptológie.

V práci sme skúmali súvis medzi šifrovaniami zachovávajúcimi formát a miešaniami kariet, predstavili sme najznámejšie FPE schémy a konštrukcie založené na miešaniach kariet. Taktiež sme aplikovali metódu použitú Thorpom pri analýze kartových hier [47] na analýzu FPE schém, podľa nám dostupných informácií sme sa týmito súvislosťami zaoberali ako prví. Definovali sme nový model bezpečnosti zodpovedajúci Thorpovej analýze a v súlade so zadefinovanými kritériami sme sa našli ideálne miešanie. Dokázali sme, že v rámci našich kritérií lepšie miešanie ako Thorpovo neexistuje, a existujú miešania rovnako dobré ako Thorpovo.

V druhej časti práce sme sa zaoberali aktuálnym stavom štandardizácie a implementácií FPE. Analyzovali sme návrhy FPE schémy, ktoré sú v súčasnosti v procese štandardizácie NISTom. Predstavili sme v súčasnosti dostupné implementácie. Následne sme vytvorili vlastnú knižnicu pyFNR pre celočíselnú FPE schému v jazyku Python. Naša knižnica taktiež obsahuje najširšiu podporu pre rôzne formáty zodpovedajúce regulárnym jazykom, ktoré je možné definovať pomocou deterministických konečných automatov.

Z hľadiska implementácie FPE by v budúcnosti bolo zaujímavé zamerať sa na integráciu FPE šifrovania do existujúcich databázových systémov. Po teoretickej stránke by bolo možné pokračovať v skúmaní súvislostí medzi FPE a kartovými hrami, konkrétne súvislosťou útokov na FPE schémy a víťazných stratégií v hrách. Podobne zaujímavá je aplikácia kryptoanalytických modelov z oblasti FPE v teórii hier, napr. určenie kartových hier, ktoré zodpovedajú jednotlivým modelom, analýza vzťahu výhody útočníka a pravdepodobnosti na výhru a vplyv použitého miešania kariet na víťazné stratégie v hrách.

Literatúra

[1] Mihir Bellare, Thomas Ristenpart, Phillip Rogaway, and Till Stegers. Format-Preserving Encryption. In *Selected Areas in Cryptography*, pages 295–312. Springer, 2009.

[2] Mihir Bellare, Phillip Rogaway, and Terence Spies. Addendum to „The FFX Mode of Operation for Format-Preserving Encryption“ Draft 1.0. [online, navštívené 11.5.2015], 2010. `http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/ffx/ffx-spec2.pdf`.

[3] Mihir Bellare, Phillip Rogaway, and Terence Spies. The FFX Mode of Operation for Format-Preserving Encryption. Draft 1.1. [online, navštívené 11.5.2015], 2010. `http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/ffx/ffx-spec.pdf`.

[4] John Black and Phillip Rogaway. Ciphers with Arbitrary Finite Domains. In *CT-RSA*, pages 114–130, 2002.

[5] Botan. Botan: Crypto and TLS for C++11. [online, navštívené 28.4.2015]. `http://botan.randombit.net/index.html`.

[6] Botan. Python Binding. [online, navštívené 28.4.2015]. `http://botan.randombit.net/manual/python.html`.

[7] Eric Brier, Thomas Peyrin, and Jacques Stern. BPS: a format-preserving encryption proposal. [online, navštívené 11.5.2015], 2010. `http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/bps/bps-spec.pdf`.

[8] Michael Brightwell and Harry Smith. Using datatype-preserving encryption to enhance data warehouse security. In *20th National Information Systems Security Conference Proceedings (NISSC)*, pages 141–149, 1997.

[9] Certivox: Distributed Cryptography Company. Miracl. [online, navštívené 28.4.2015].
`https://www.certivox.com/miracl`.

[10] Intel Corporation. Intel Digital Random Number Generator (DRNG): Software Implementation Guide, Revision 1.1, 2012.
`https://software.intel.com/sites/default/files/m/d/4/1/d/8/441_Intel_R__DRNG_Software_Implementation_Guide_final_Aug7.pdf`.

[11] dakill (pseudonym). DotFPE - A free format-preserving encryption implementation for .net. [online, navštívené 28.4.2015].
`http://dotfpe.codeplex.com`.

[12] Sashank Dara and Scott Fluhrer. FNR: Arbitrary length small domain block cipher proposal. *IACR Cryptology ePrint Archive*, 2014:421, 2014.

[13] Richard Durstenfeld. Algorithm 235: Random Permutation. *Communications of the ACM*, 7(7):420, 1964.

[14] Kevin P. Dyer. About me. [online, navštívené 28.4.2015].
`https://kpdyer.com`.

[15] Kevin P. Dyer. FFX. [online, navštívené 28.4.2015].
`https://github.com/kpdyer/libffx`.

[16] Leonard Euler. Sur l'avantage du banquier au jeu de Pharaon. *Mémoires de l'académie de sciences de Berlin*, 20:144–164, 1764.
Anglický preklad dostupný online, navštívený 11.5.2015.
`http://cerebro.xu.edu/math/Sources/Euler/E313.pdf`.

[17] Horst Feistel. Block cipher cryptographic system. US Patent 3,798,359, 1974.

[18] Louis Granboulan and Thomas Pornin. Perfect block ciphers with small blocks. In *Fast Software Encryption*, pages 452–465. Springer, 2007.

[19] Mike Hamburg, Paul Kocher, and Mark E. Marson. Analysis of Intel's Ivy Bridge digital random number generator. [online, navštívené 11.5.2015], 2012.
`https://www.cryptography.com/public/pdf/Intel_TRNG_Report_20120312.pdf`.

[20] Viet Tung Hoang, Ben Morris, and Phillip Rogaway. An Enciphering Scheme Based on a Card Shuffle. In *Advances in Cryptology–CRYPTO 2012*, pages 1–13. Springer, 2012.

[21] HP Security Voltage. Format-Preserving Encryption (FPE), Data Masking, Datatype Agnostic, Referential Integrity. [online, navštívené 28.4.2015]. `https://www.voltage.com/technology/data-encryption/hp-format-preserving-encryption/`.

[22] HP Security Voltage. Streamlining Information Protection through a Data-centric Security Approach. [online, navštívené 28.4.2015]. `https://www.voltage.com/resource/streamlining-information-protection-through-a-data-centric-security-approach-2/`.

[23] Cisco Systems Inc. jFNR. [online, navštívené 28.4.2015]. `https://github.com/cisco/jfnr`.

[24] Cisco Systems Inc. LibFNR. [online, navštívené 28.4.2015]. `http://cisco.github.io/libfnr/`.

[25] ISO/IEC 7812-1:2006. Identification cards – Identification of issuers – Part 1: Numbering system, 2006.

[26] Donald Ervin Knuth. *The Art of Computer Programming. Volume 2., (2nd Ed.)*, pages 139–140. Addison-Wesley series in computer science and information processing. Addison-Wesley, Reading (Mass.), Menlo Park (Calif.), London, 1981.

[27] Michael Luby and Charles Rackoff. How to Construct Pseudorandom Permutations from Pseudorandom Functions. *SIAM Journal on Computing*, 17(2):373–386, 1988.

[28] Ben Morris and Phillip Rogaway. Sometimes-Recurse Shuffle: Almost-Random Permutations in Logarithmic Expected Time. *IACR Cryptology ePrint Archive*, 2013:560, 2013.

[29] Ben Morris, Phillip Rogaway, and Till Stegers. How to Encipher Messages on a Small Domain. In *Advances in Cryptology-CRYPTO 2009*, pages 286–302. Springer, 2009.

[30] Moni Naor and Omer Reingold. On the Construction of Pseudo-Random Permutations: Luby-Rackoff Revisited. *Journal of Cryptology*, 12(1):29–66, 1997.

[31] National Bureau of Standards. *FIPS PUB 46, Data Encryption Standard*. U.S.Department of Commerce, 1977.

[32] National Bureau of Standards. *FIPS PUB 74, Guidelines for Implementing and Using the NBS Data Encryption Standard*. U.S.Department of Commerce, 1981.

[33] Národná rada Slovenskej republiky. Zákon o rodnom čísle, December 1995. 301/1995 Z.z.

[34] Národná rada Slovenskej republiky. Zákon o ochrane utajovaných skutočností, March 2004. 215/2004 Z.z.

[35] National Institute of Standards and Technology. Computer Security Division - Computer Security Resource Center. [online, navštívené 27.4.2015]. `http://csrc.nist.gov/groups/ST/toolkit/BCM/modes_development.html#03`.

[36] National Institute of Standards and Technology. NIST General Information. [online, navštívené 27.4.2015]. `http://www.nist.gov/public_affairs/general_information.cfm`.

[37] National Institute of Standards and Technology. *FIPS PUB 197, Advanced Encryption Standard*. U.S.Department of Commerce, 2001.

[38] National Institute of Standards and Technology. Recommendation for Block Cipher Modes of Operation: Methods for Format-Preserving Encryption. Special Publication 800-38G Draft, [online, navštívené 11.5.2015], 2013. `http://csrc.nist.gov/publications/drafts/800-38g/sp800_38g_draft.pdf`.

[39] Matthew. J. Pauker, Terence Spies, and Luther. W. Martin. Data processing systems with format-preserving encryption and decryption engines. US Patent 7,864,952, 2011.

[40] Sean Poulter. Credit card alert as hackers target 77 million playstation users. [DailyMail online; navštívené 1.5.2015], April 2011. `http://www.dailymail.co.uk/sciencetech/article-1381000/Playstation-Network-hacked-Sony-admits-hackers-stolen-77m%5dusers-credit-card-details.html`.

[41] Thomas Ristenpart and Scott Yilek. The Mix-and-Cut Shuffle: Small-Domain Encryption Secure against N Queries. In *Advances in Cryptology–CRYPTO 2013*, pages 392–409. Springer, 2013.

[42] robshep (pseudonym). FPE for Java. [online, navštívené 28.4.2015]. `https://github.com/robshep/JavaFPE`.

[43] Phillip Rogaway. A Synopsis of Format-Preserving Encryption. [online, navštívené 10.5.2015], 2010.
`http://web.cs.ucdavis.edu/~rogaway/papers/synopsis.pdf`.

[44] Terence Spies. Feistel finite set encryption mode. [online, navštívené 11.5.2015], 2008.
`http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/ffsem/ffsem-spec.pdf`.

[45] Terence Spies. Format preserving encryption. [online, navštívené 11.5.2015], 2008.
`http://csrc.nist.gov/groups/ST/PEC2011/presentations2011/spies.pdf`.

[46] Emil Stefanov and Elaine Shi. FastPRP: Fast Pseudo-Random Permutations for Small Domains. *IACR Cryptology ePrint Archive*, 2012:254, 2012.

[47] Edward Thorp. Nonrandom Shuffling with Applications to the Game of Faro. *Journal of the American Statistical Association*, 68(344):842–847, 1973.

[48] Joachim Vance. VAES3 scheme for FFX. An addendum to „The FFX Mode of Operation for Format-Preserving Encryption“ Draft 1.0. [online, navštívené 11.5.2015], 2011.
`http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/ffx/ffx-ad-VAES3.pdf`.

Dodatok A

Optimálne miešania kariet

A.1 Náhodné miešania

V tabuľke A.1 uvádzame časť získaných výsledkov. Počas experimentu sme určovali minimálny počet kôl r potrebných na to, aby rozptyl prvkov matice pravdepodobností r-kolového miešania bol menší ako ε. Hodnotu r sme určovali náhodným výberom 10000 matíc pravdepodobností (ktoré obsahovali v každom riadku aj v každom stĺpci práve dve hodnoty $\frac{1}{2}$) pre veľkosti matíc $S \in \{4, 5, \ldots, 100\}$ a $\varepsilon \in \{10^{-1}, 10^{-2}, \ldots, 10^{-15}$. V nasledujúcej tabuľke uvádzame hodnotu r náhodných matíc v závislosti od S pre $\varepsilon = 10^{-10}$, ako aj hodnotu r_T matice pravdepodobností Thorpovho miešania pre dané S a ε. V stĺpci $\#_f$ je uvedený počet nájdených rôznych matíc, ktorých prvky mali rozptyl menší ako ε po r kolách miešania. V stĺpci $\#_b$ je uvedený počet náhodne vygenerovaných matíc (nie nutne rôznych), ktorých prvky mali rozptyl väčší ako ε aj po 100 kolách miešania.

S	r	r_T	$\#_f$	$\#_b$
4	2	2	12	633
5	16	16	230	286
6	16	16	1538	353
7	16	16	667	242
8	3	3	4	182
9	16	15	106	146
10	16	16	32	122
11	16	15	5	136
12	16	16	7	75
13	16	16	1	88
14	17	16	1	50
15	17	16	2	56
16	19	4	3	46
17	18	16	1	50
18	20	16	10	24
19	19	16	1	28
20	20	16	1	31
21	20	16	2	26
22	19	16	1	25
23	20	16	1	29
24	21	16	6	19
25	22	16	29	19
26	22	16	11	22
27	21	16	1	13
28	22	16	11	18
29	21	15	1	12
30	22	16	2	10
31	22	15	5	19
32	22	5	3	17
33	22	15	3	7
34	22	16	3	12
35	22	16	1	14
36	23	16	12	13
37	23	16	10	8
38	23	16	7	6
39	23	16	4	14
40	23	16	6	6
41	23	16	8	12
42	23	16	9	7
43	23	14	1	11
44	23	16	4	6
45	23	16	1	4
46	23	16	1	11
47	23	16	2	8
48	23	16	1	10
49	24	16	31	6
50	23	16	1	5
51	24	16	29	4
52	23	16	1	7
53	24	16	24	4
54	23	16	2	8
55	24	16	14	6
56	23	16	1	6
57	23	16	1	5
58	24	15	12	7
59	24	16	10	6
60	24	16	7	3
61	24	16	11	2
62	24	16	8	6
63	24	16	8	5
64	24	6	6	7
65	24	16	12	6
66	24	16	10	1
67	24	15	9	9
68	24	15	3	5
69	24	15	3	4
70	24	15	1	3
71	24	16	2	6
72	24	15	6	2
73	24	16	3	3
74	24	16	2	3
75	24	15	3	2
76	24	16	3	3
77	24	16	2	5
78	24	15	1	3
79	24	16	1	5
80	24	16	1	2
81	24	16	1	0
82	24	16	1	3
83	25	16	75	1
84	24	15	1	1
85	24	16	1	2
86	24	15	4	0
87	25	16	89	3
88	24	16	1	0
89	24	16	1	1
90	24	15	1	5
91	24	15	3	2
92	24	16	1	3
93	25	16	70	1
94	24	16	2	1
95	25	15	56	5
96	24	16	1	1
97	25	16	58	2
98	24	16	1	2
99	25	15	59	2
100	24	16	1	2

Tabuľka A.1: Minimálny počet kôl miešaní pre náhodné miešania a $\varepsilon = 10^{-10}$

A.2 Matice optimálnych miešaní veľmi malých rozmerov

Optimálne matice rozmerov 4×4. Tieto matice pripomínajú mierne modifikované Thorpovo miešanie. Pokiaľ ich interpretujeme ako miešania kariet, predstavujú rôzne doplnkové operácie k Thorpovmu miešaniu, napr. presun prvej karty na posledné miesto pred alebo po aplikovaní Thorpovho miešania, reverz jednej polovice kariet pred miešaním, reverz celého balíčka kariet, prípadne inverzné miešanie k Thorpovmu miešaniu (rozdeľovanie dvojíc kariet na dve kôpky, ako bolo uvedené v časti 2.3.2).

$$\begin{pmatrix} 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \end{pmatrix} \quad \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \end{pmatrix}$$

$$\begin{pmatrix} \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \end{pmatrix} \quad \begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \end{pmatrix}$$

$$\begin{pmatrix} \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 \end{pmatrix} \quad \begin{pmatrix} \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \end{pmatrix}$$

$$\begin{pmatrix} \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix} \quad \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

Dodatok B

Príklady použitia pyFNR

V nasledujúcich výpisoch uvádzame príklady použitia našej knižnice pyFNR spolu s ukážkami ich výstupov. Tieto programy je možné spustiť pomocou Pythonu vo verzii 2.6 až 3.4. Zdrojové kódy sú taktiež dostupné online na Githube[1].

```
import pyFNR
salt = pyFNR.generate_salt()
fnr16 = pyFNR.FNR(key='password', tweak='tweak-is-string',
   block_size=16, salt=salt)
fnr64 = pyFNR.FNR(key='password', tweak='tweak-is-string',
   block_size=64, salt=salt)
plain_int = 47
cipher_int = fnr16.encrypt_int(plain_int)
plain2_int = fnr16.decrypt_int(cipher_int)
print(str(plain_int) + '␣->␣' + str(cipher_int) + '␣->␣' + str(
   plain2_int))
cipher_int = fnr64.encrypt_int(plain_int)
plain2_int = fnr64.decrypt_int(cipher_int)
print(str(plain_int) + '␣->␣' + str(cipher_int) + '␣->␣' + str(
   plain2_int))
plain_str = "Hello"
cipher_str = fnr64.encrypt_str(plain_str)
plain2_str = fnr64.decrypt_str(cipher_str)
print(repr(plain_str) + '␣->␣' + repr(cipher_str) + '␣->␣' + repr(
   plain2_str))
fnr16.close()
fnr64.close()
```

Výpis B.1: Šifrovanie celých čísel a reťazcov: `fpe.py`

[1] `https://github.com/laciKE/pyFNR/tree/master/examples`

```
import pyFNR
import pyFNR.Util
ipv4 = pyFNR.Util.IPv4()
fnr = pyFNR.FNR2(key='password', tweak='tweak-is-string', domain=
   ipv4.get_words_count()-1)
plain = '10.0.0.42'
cipher = ipv4.unrank(fnr.encrypt(ipv4.rank(plain)))
plain2 = ipv4.unrank(fnr.decrypt(ipv4.rank(cipher)))
print(plain + '␣->␣' + cipher + '␣->␣' +plain2)
fnr.close()
```

Výpis B.2: Šifrovanie IPv4 adries: `ipv4.py`

```
import pyFNR
import pyFNR.Util
ipv6 = pyFNR.Util.IPv6()
fnr = pyFNR.FNR2(key='password', tweak='tweak-is-string', domain=
   ipv6.get_words_count()-1)
plain = '2001:DB8:2de::e13'
cipher = ipv6.unrank(fnr.encrypt(ipv6.rank(plain)))
plain2 = ipv6.unrank(fnr.decrypt(ipv6.rank(cipher)))
print(plain + '␣->␣' + cipher + '␣->␣' +plain2)
fnr.close()
```

Výpis B.3: Šifrovanie IPv6 adries: `ipv6.py`

```
import pyFNR
import pyFNR.Util
ccn = pyFNR.Util.LuhnR(0, 16)
fnr = pyFNR.FNR2(key='password', tweak='tweak-is-string', domain=
   ccn.get_words_count()-1)
plains = ['4024007162012628', '5260106710301747', '
   6011001620745085']
ciphers = [ccn.unrank(fnr.encrypt(ccn.rank(plain))) for plain in
   plains]
plains2 = [ccn.unrank(fnr.decrypt(ccn.rank(cipher))) for cipher in
    ciphers]
for i in range(len(plains)):
  print(plains[i] + '␣->␣' + ciphers[i] + '␣->␣' + plains2[i])
fnr.close()
```

Výpis B.4: Šifrovanie čísel kreditných kariet: `ccn.py`

```
import pyFNR
import pyFNR.Util
ecv = pyFNR.Util.ECV()
fnr = pyFNR.FNR2(key='password', tweak='tweak-is-string', domain=
    ecv.get_words_count()-1)
plain = 'KE007JB'
cipher = ecv.unrank(fnr.encrypt(ecv.rank(plain)))
plain2 = ecv.unrank(fnr.decrypt(ecv.rank(cipher)))
print(plain + '␣->␣' + cipher + '␣->␣' +plain2)
fnr.close()
```

Výpis B.5: Šifrovanie evidenčných čisiel vozidiel: `ecv.py`

```
import pyFNR
import pyFNR.Util
"""
Example of custom format: regular language with symbols 'a', 'b'
    such that number of 'a' is divisible by 3 and number of 'b' is
    divisible by 2.
"""
# properties of custom DFA
Q = [(count_a, count_b) for count_a in range(3) for count_b in
    range(2)]
Sigma = ['a', 'b']
def delta(q, c):
  count_a, count_b = q[0], q[1]
  count_a = (count_a + 1) % 3 if c == 'a' else count_a
  count_b = (count_b + 1) % 2 if c == 'b' else count_b
  return (count_a, count_b)
q_0 = (0, 0)
F = [(0, 0)]
dfa = pyFNR.Util.DFA(Q, Sigma, delta, q_0, F)
myFormat = pyFNR.Util.FPE_Format(dfa, 12) # words with length 12
fnr = pyFNR.FNR2(key='password', tweak='tweak-is-string', domain=
    myFormat.get_words_count()-1)
plains = ['bbaaabbaaabb', 'aaaaaaaaaaaa', 'aaaaaabbbbbb', '
    bbbbbbbbbbbb']
for plain in plains:
  cipher = myFormat.unrank(fnr.encrypt(myFormat.rank(plain)))
  plain2 = myFormat.unrank(fnr.decrypt(myFormat.rank(cipher)))
  print(plain + '␣->␣' + cipher + '␣->␣' +plain2)
fnr.close()
```

Výpis B.6: Šifrovanie slov regulárneho jazyka: `dfa.py`

Ukážka výstupov programov z výpisov B.1, B.2, B.3, B.4, B.5 a B.6.

```
$ python examples/fpe.py
47 -> 27144 -> 47
47 -> 14176901634242626597 -> 47
'Hello' -> '\x7f\x0bt\x1c\xec\xf7\xed\\' -> 'Hello'

$ python examples/ipv4.py
10.0.0.42 -> 213.239.53.204 -> 10.0.0.42

$ python examples/ipv6.py
2001:DB8:2de::e13 -> 3e5e:34d9:653f:4d08:a9a2:963b:28:7c88 -> 2001:db8:2de::e13

$ python examples/ccn.py
4024007162012628 -> 6123608756271168 -> 4024007162012628
5260106710301747 -> 8758292354597791 -> 5260106710301747
6011001620745085 -> 0797941620321432 -> 6011001620745085

$ python examples/ecv.py
KE007JB -> LE816WE -> KE007JB

$ python examples/dfa.py
bbaaabbaaabb -> ababaaaabbbb -> bbaaabbaaabb
aaaaaaaaaaaa -> abaabbaababb -> aaaaaaaaaaaa
aaaaaabbbbbb -> babbababbaaa -> aaaaaabbbbbb
bbbbbbbbbbbb -> abaaabbabbab -> bbbbbbbbbbbb
```