MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

Generovanie náhodných a pseudonáhodných dát v mobilných zariadeniach s iOS

BAKALÁRSKA PRÁCA

Ján Kováč

Brno, 2013

Prehlásenie

Prehlasujem, že táto bakalárska práca je mojim pôvodným autorským dielom, ktoré som vypracoval samostatne. Všetky zdroje, pramene a literatúru, ktoré som pri vypracovaní používal alebo z nich čerpal, v práci riadne citujem s uvedením úplného odkazu na príslušný zdroj.

.....

Podpis

Vedúci práce: Mgr. et Mgr. Jan Krhovják, Ph.D.

Poďakovanie

Chcel by som poďakovať vedúcemu svojej bakalárskej práce pánovi Mgr. et Mgr. Janu Krhovjákovi, Ph.D. za cenné rady pri jej tvorbe a za všetok čas, ktorý mi týmto venoval.

Zhrnutie

Cieľom tejto bakalárskej práce je preštudovanie možností generovania náhodných a pseudonáhodných dát v mobilných zariadeniach s operačným systémom iOS.

Teoretická časť sa venuje základnej problematike generovania a bližšie opisuje operačný systém iOS a jeho metódy programovania.

Praktickú časť tvoria dva naimplementované generátory pseudonáhodných čísel ANSI X9.31 a Fortuna.

Kľúčové slová

ANSI X9.31, Fortuna, generátor, kryptografia, iOS, pseudonáhodné čísla, bezpečnosť, šifrovanie

Obsah

Úvod

Mobilné zariadenia už nie sú len jednoúčelové prístroje slúžiace na komunikáciu s inými osobami, ale dajú sa považovať za prenosné osobné počítače, ktoré zohrávajú v živote jednotlivca čím ďalej tým väčšiu úlohu. Slúžia napríklad ako hudobný prehrávač, fotoaparát, herná konzola, ale aj ako úložisko citlivých informácii a hesiel, čo môže byť veľkým lákadlom pre potenciálneho útočníka. Preto je nevyhnutné klásť dôraz na dostatočné zabezpečenie ukladaných dát. Jednou z možností ako to docieliť je použitie kryptografických prostriedkov.

Jedným zo základných stavebných prvkov, ktoré súčasné kryptografické metódy využívajú, je generátor náhodných či pseudonáhodných čísel. Avšak generovanie skutočne náhodných dát je veľmi zložité na získanie dostatočného množstva, pri ktorom by sa zachovala kvalita a nepredvídateľnosť, preto sa v praxi používajú deterministické algoritmy, ktoré generujú pseudonáhodné sekvencie s cieľom dosiahnuť nerozlíšiteľnosť od skutočne náhodných dát. Počiatočné hodnoty(semienka) sa získavajú z rôznych zdrojov náhodnosti, ako napríklad údaje z kamery alebo vstavaného mikrofónu a tie potom majú s jednoznačnou určitosťou vplyv na priebeh následného chodu generátora.

Principiálne nie je možné dokázať, že sú dáta skutočne náhodné, ale existujú určité sady testov, ktoré do istej miery dokážu detekovať nenáhodné správanie. Napríklad sady testov NIST[12] obsahujú 15 testov, kde každý z nich testuje rozdielne typy nenáhodnosti, ktoré sa môžu vyskytovať vo výstupnej sekvencii generátora. Zatiaľ neexistuje žiadny priemyselný štandard, ktorý by bol efektívny a zároveň aj kvalitný, avšak americká organizácia NIST (National Institute of Standards and Technology) v roku 2007 vydala upravenú publikáciu SP800-90A [13] (Recommendation for Random Number Generation Using Deterministic Random Bit Generators), ktorá sa týka rôznych doporučení pre generovanie pseudonáhodných čísel. Tento dokument mimo iné navrhuje aj nové spôsoby generovania založené na hašovacích a šifrovacích algoritmoch.

Tie sú súčasťou generátorov ANSI X9.31 a Fortuna, ktoré budeme implementovať v praktickej časti tejto bakalárskej práce. V teoretickej časti sa zameriame na dve konkrétne veci. Poprvé na generovanie pseudonáhodných sekvencií s dôrazom kladeným na bezpečnosť a podruhé popíšeme základné prístupy a metódy programovania na iOS.

1 Generovanie (pseudo)náhodných sekvencií

V tejto kapitole sa budeme zaoberať teoretickou časťou generovania náhodných a pseudonáhodných sekvencií, zadefinujeme si základné pojmy týkajúce sa danej problematiky, zameriame sa na možné útoky a následné zabezpečenie pri vytváraní pseudonáhodných dát.

1.1 Základná teória

Náhodné sekvencie čísiel majú široké spektrum použitia v rôznych oblastiach, kde je potrebné zaistiť premenlivosť systému. Jednou z takýchto oblastí je aj simulovanie prirodzených javov, kedy vytvárame podmienky čo najviac nerozlíšiteľné od reálnych, lebo získanie skutočných údajov by bolo nesmierne náročné a finančne nákladné. Takéto simulácie sa využívajú prakticky všade, či už ide o architektúru, šport alebo počasie.

Teraz si môžeme konkrétnejšie uviesť, aké využitie majú náhodné čísla v oblasti informatiky. V prvom rade zohrávajú dôležitú úlohu v náhodnostných algoritmoch, čo sú vlastne špeciálne nedeterministické algoritmy, ktoré sa na určitých častiach svojej implementácie rozvetvujú, čo spôsobí zakaždým iné chovanie výpočtu. Rozhodovanie do ktorej vetvy vstúpiť je potom otázkou nejakého náhodného čísla, čo spôsobí, že každá vetva by mala byť dosiahnuteľná s rovnakou pravdepodobnosťou ako všetky ostatné. Takéto algoritmy majú veľmi bohaté praktické využitie, preto je výskumu tohto odvetvia venovaná značná pozornosť. Za zmienku stojí ich využitie napríklad v počítačových hrách, kedy má užívateľ možnosť zmeniť dejovú líniu na základe jeho momentálneho rozhodnutia, čo ponúka nespočetné množstvo spôsobov, ako hru ukončiť a tým pádom väčší zážitok zákazníka.

Náhodné čísla ale zohrávajú kľúčovú úlohu v oblasti kryptografie, kde sa v skoro každom algoritme alebo protokole využíva náhodnosť, ktorá je základným predpokladom pre dostatočne bezpečné a kvalitné fungovanie ľubovoľného systému. Práve v tejto oblasti vyžadujú generátory pseudonáhodných sekvencií prísnejšie podmienky ako v iných oblastiach, lebo nedostatočne kvalitný generátor môže znehodnotiť teoreticky silnú šifru, ak by útočník dokázal predvídať hodnoty vytvárané generátorom, čo by malo fatálne následky na bezpečnosť celého systému.

Zatiaľ sme si nedefinovali pojem náhodná sekvencia, aj keď väčšina z nás má aspoň intuitívnu predstavu o tom čo to je. Napríklad keď porovnáme nasledujúce

sekvencie bitov, isto budeme vedieť rozlíšiť, ktorá postupnosť je náhodná a ktorá vytvorená pomocou nejakého algoritmu:

10010010010010010010010 01
00101011101000111010011 10

Prvá postupnosť dvadsiatich piatich bitov je zrejme vytvorená pomocou jednoduchého pravidla, kedy po číslici 1 nasledujú dve číslice 0 a je veľmi ľahké odhadnúť, ako bude vyzerať ďalšia časť takto vygenerovanej sekvencie. Avšak druhá postupnosť sa nám javí ako náhodné usporiadanie jednotiek a núl a preto nie sme schopný zistiť následnú postupnosť daných bitov. Táto sekvencia vznikla klasickým spôsobom generovania náhodných čísel a to takým, že postupne pridávame jednotku alebo nulu podľa nejakého náhodného mechanizmu, kde vybratie každého čísla má rovnakú pravdepodobnosť a zároveň je nezávislé na svojom predchodcovi. Náhodný mechanizmus bude predstavovať v našom prípade hod mincou. Jedna strana mince reprezentuje vybratie číslice 1 a druhá strana číslicu 0. Celkovo máme teda 2^{25} rôznych možností, ako bude vyzerať vygenerovaná sekvencia, všetky s rovnakou pravdepodobnosťou hodu. Tým pádom sa ale môže stať, že takto náhodne vytvoríme postupnosť, v ktorej budeme môcť navonok spozorovať určitý vzor. Napríklad jedna z týchto 2^{25} možností bude identická s prvou postupnosťou bitov z nášho príkladu spomenutého vyššie, preto by sme mali považovať obe sekvencie za náhodné, čo však odporuje intuitívnemu konceptu náhodnej postupnosti. Kvôli tomu je vhodné si zadefinovať presnejší termín náhodnosti s rozložením čísel, ktoré neobsahujú viditeľný vzor. [9] Takýto mechanizmus hodu mince je ale najlepší generátor nezávislej sekvencie bitov, ktorú je potrebné generovať automatizovane a výkon vytvárania takéhoto prúdu dát musí byť pre naše účely dostatočne rýchly. Len ako docieliť, aby sme automatizovane vytvárali náhodné údaje?

Musíme k tomu použiť nejaké zdroje náhodnosti, ktoré získame tak, že budeme sledovať určitý prirodzene náhodný dej, pre ktorý musí platiť to, že vytvára skutočne náhodné nezávislé dáta a nesmie byt ovplyvniteľný potenciálnym útočníkom. Takéto zdroje môžeme teoreticky rozdeliť na hardwarové a softwarové podľa toho, akým spôsobom ich získavame.

V prvom prípade ide o dáta z fotoaparátu, zvukové údaje z mikrofónu alebo o dáta z akcelerometra a gyroskopu, ktoré určujú momentálne zrýchlenie a orientáciu v priestore. Môže ísť ale aj o meranie hodnôt nejakého fyzikálneho javu ako napríklad frekvenčná nestabilita oscilátorov, rozpad rádioaktívnych prvkov, vzduchová turbulencia vo vnútri pevného disku a podobne. Niekedy môžu byť ale takto získané údaje z časti nepresné a vtedy dochádza ku skresleniu, čo sa prejavuje tak, že na výstupe sa nachádza jedna hodnota oveľa častejšie ako druhá, čiže pravdepodobnosť získania bitov nula a jedna nemá rovnaký pomer alebo ku korelácii,

čo znamená, že údaje získané zo zdroja náhodnosti sú závislé na svojom predchodcovi.

Druhý prípad sa viaže na možnosti operačného systému interagujúceho s mnoho užívateľmi a jeho rôzne štatistiky, ktoré sa používajú veľmi často z dôvodu nenáročnej implementácie. Konkrétnejšie to väčšinou býva systémový čas, sieťové štatistiky, identifikačné čísla procesov, momentálna záťaž systému a v neposlednom rade aj užívateľský vstup, ktorý má viacero variant zdrojov. Môžeme si uviesť jednu takúto variantu, ktorá sa týka mobilných zariadení s dotykovou obrazovkou a to súradnice miesta dotyku užívateľa s displayom. Tak ako aj hardwarový zdroj, aj softwarový trpí určitým nedostatkom, tentoraz týkajúci sa bezpečnosti. Potenciálny útočník je schopný v niektorých prípadoch odhadnúť buď konkrétne údaje alebo aspoň množinu hodnôt, ktoré sa potenciálne vyskytnú v danom výstupe, ba ich dokonca ovplyvniť.

V obidvoch prípadoch je ale generovanie skutočne náhodných dát vo veľkých objemoch pomalé a nesmierne náročné, preto sa v praxi využíva pseudonáhodné vytváranie sekvencií a skutočne náhodné dáta slúžia len ako semienka, teda počiatočné údaje vnášajúce do procesu tvorby náhodnosť. Tie určujú, ako bude vyzerať výsledný bitový prúd, ktorý je už vygenerovaný deterministickým algoritmom a ktorý by mal byť oveľa dlhší ako vstupné údaje, aby prípadný útočník nebol schopný určiť, či sa jedná len o pseudonáhodné dáta alebo skutočne náhodnú sekvenciu bitov.

1.2 Definície

V tejto časti si bližšie zadefinujeme pojmy, s ktorými sa budeme ďalej v texte stretávať.

- **Entropia**
  Nech X je náhodná premenná, ktorá nadobúda hodnoty na konečnej množine $x_1, x_2, \ldots, x_n$, s pravdepodobnosťou $P(X = x_i) = p_i$, kde $0 \leq p_i \leq 1$ pre každé i, $1 \leq i \leq n$, a kde $\sum_{i=1}^{n} p_i = 1$. Entropia X je matematická miera množstva informácií nesených premennou X. Ekvivalentne, je to neistota ohľadom výsledku pred pozorovaním X. Entropia je tiež užitočná pre aproximáciu priemerného počtu bitov potrebných k zakódovaniu prvkov X. [10, str. 56]

- **RBG** *(random bit generator)*
  Generátor náhodných bitov je zariadenie alebo algoritmus, ktorého výstupom sú nezávislé a neovplyvnené bitové čísla. [10, str. 170, def. 5.1]

- **PRBG** *(pseudorandom bit generator)*
  Generátor pseudonáhodných bitov je deterministický[1] algoritmus, ktorého vstupom sú skutočne náhodné bitové sekvencie dĺžky k a výstupom sú bitové sekvencie dĺžky $l \gg k$, ktoré sa javia ako náhodné. [10, str. 170, def. 5.3]

- **RNG** *(random number generator)*
  Generátor náhodných čísiel je algoritmus, ktorý generuje náhodné čísla z intervalu <0,n> tak, že každé jedno číslo typu integer sa vytvára pomocou bitovej sekvencie dĺžky $\log_2(n) + 1$ generovanej pomocou RBG. V prípade, že takto vytvorené číslo prekročí hodnotu n bude odignorované a je potrebné vygenerovať rovnakým spôsobom nové číslo.

- **PRNG** *(pseudorandom number generator)*
  Generátor pseudonáhodných čísiel je založený na rovnakom princípe ako RNG a generované sekvencie bitov sa vytvárajú pomocou PRBG.

Posledné štyri definície sa zaoberali rôznymi typmi generátorov náhodných dát a väčšina z nich vykazuje vysokú rýchlosť generovania na úkor bezpečnosti. Avšak tieto typy nie sú príliš vhodné pre kryptografické účely. Na tie nám slúži kryptografický generátor pseudonáhodných bitov, ktorého presnému zadefinovaniu predchádzajú dve definície týkajúce sa testov daného generátora.

- **Definícia 1.1:** Nech PRBG prejde všetkými štatistickými testami s polynomiálnou časovou zložitosťou, ak neexistuje polynomiálny algoritmus, ktorý korektne rozlíši výstupnú sekvenciu generovanú PRBG od skutočne náhodnej sekvencie s rovnakou dĺžkou s pravdepodobnosťou výrazne väčšou ako 1/2 [10, str. 171, def. 5.5].

- **Definícia 1.2:** Nech PRBG prejde testom nasledujúceho bitu, ak neexistuje polynomiálny algoritmus, ktorý na vstupe prvých l bitov výstupnej sekvencie s dokáže predpovedať, akú hodnotu bude mať $l + 1$ bit sekvencie s s pravdepodobnosťou výrazne väčšou ako 1/2 [10, str. 171, def. 5.6].

Tieto dve definície sú ekvivalentné, aj keď sa nám môže na prvý pohľad javiť, že jedna definícia v sebe zahrňuje aj definíciu druhú. Samozrejme túto ekvivalentnosť je možné dokázať [11]. Teraz nám už nič nebráni uviesť definíciu kryptografického generátora pseudonáhodných bitov.

[1] *Deterministický* v tomto prípade znamená, že s rovnakými vstupnými údajmi

- **CSPRBG** *(cryptographically secure pseudorandom bit generator)*
  Kryptograficky bezpečný generátor pseudonáhodných bitov je generátor pseudonáhodných bitov, ktorý prejde testom nasledujúceho bitu [10, str. 171, def. 5.8].

V tejto definícii sa predpokladá, že vnútorný stav generátora zostáva nekompromitovaný, ale reálne môže nastať situácia, kedy útočník získa buď čiastočné alebo dokonca aj úplné informácie o vnútornom stave generátora, preto sa požadujú ešte tieto dve ďalšie vlastnosti, ktoré značne zvyšujú bezpečnosť celého generátora.

- **Dopredná bezpečnosť**
  Útočník, ktorý skompromitoval zabezpečenie generátora nedokáže spätne zrekonštruovať sekvenciu, ktorá bola týmto generátorom vytvorená.

- **Spätná bezpečnosť**
  Útočník, ktorý skompromitoval zabezpečenie generátora nedokáže odhadnúť nasledujúcu sekvenciu alebo jej časť, ktorá bude týmto generátorom vytvorená.

O generátoroch CSPRBG môžeme tvrdiť, že majú vysoký stupeň bezpečnosti dosahujúceho využívaním ťažkých problémov teórie čísel alebo zložitosti vo svojich algoritmoch ako napríklad RSA problém, ktorý využívajú najmä generátory RSA a Micali-Schnorr. Ďalej sa využíva tiež problém faktorizácie celých čísel hlavne v generátore Blum-Blum-Shub [2]. Samozrejme existujú aj mnohé generátory využívajúce iné ťažké problémy z oblasti teórie čísel, avšak všetky aj vyššie spomenuté, trpia jedným nedostatkom a tým je rýchlosť generovania, ktorá je často oveľa pomalšia, ako klasické PRNG založené na kryptografických funkciách, ktorými sa budeme venovať v praktickej časti tejto bakalárskej práce.

1.3 Útoky na generátory pseudonáhodných sekvencií

Keďže výstupné sekvencie generované PRNG sú zastúpené v podstate v každej kryptografickej situácii ako napríklad pri tvorbe symetrických kľúčov, inicializačných vektorov alebo sa vyskytujú ako náhodné parametre v DSA podpise, hrozí veľké riziko, že sa potenciálny útočník pokúsi skompromitovať funkčnosť takéhoto generátora. Teoreticky sa za útok považuje akýkoľvek prípad, kedy dokážeme rozlíšiť výstup z PRNG od náhodných údajov. Na niektoré tieto útoky,

[2] niekedy sa označuje aj ako $x^2 \bmod n$ generátor

ktoré môžu v určitých situáciách vznikať aj v praxi, sa pozrieme v nasledujúcej časti textu, kde ich rozdelíme do troch kategórií podľa spôsobu, akým sa snažia narušiť korektné fungovanie generátora.[8]

Priame kryptoanalytické útoky

Do tejto kategórie patria útoky, ktoré sú založené na princípe bezprostredného zisťovania, či sa výstup generátora líši od náhodných dát. Útočník sa teda nezaujíma priamo o PRNG, ale iba o výstupy ním vytvorené. Tento druh útoku je aplikovateľný na väčšinu generátorov, ale existujú aj niektoré, ktoré sú voči nemu odolné, lebo výstupné sekvencie sú pre útočníka skryté a nemá možnosť sa k ním bližšie dopátrať. K takýmto patria napríklad generátory slúžiace na tvorbu 3DES kľúčov, alebo generátory nachádzajúce sa ako súčasť nejakého väčšieho procesu.

Útoky na vstup generátora

Tieto útoky sú založené rovnako na princípe rozlíšenia výstupu od skutočne náhodných dát, avšak proces akým to zisťujú sa od prvej kategórie značne líši, lebo útočník využíva znalosť vstupnej sekvencie generátora, alebo v niektorých prípadoch je dokonca schopný zameniť údaje posielané na vstup. Útoky na vstup môžeme ďalej rozdeliť do podkategórií podľa prístupu k vstupným dátam a to na útoky zvoleného vstupu, prehraného vstupu a na útoky známeho vstupu.

Pri útokoch zvoleného vstupu má útočník možnosť si vybrať vzorku entropie, ktorú pošle ako vstupné údaje generátora. Samozrejme takáto možnosť napadnutia sa týka iba generátorov, ktorých inicializačné semienka bývajú vo forme ľahko ovplyvniteľných údajov ako napríklad prichádzajúce správy, poprípade heslá užívateľov.

Útoky prehraného vstupu sa príliš nelíšia od útokov zvoleného vstupu, avšak tentokrát nedokáže útočník poslať na vstup vlastné dáta, ale iba údaje, ktoré už generátor niekedy spracovával. Používajú sa na napadnutie rovnakých generátorov ako sme sa zmienili v prvom prípade.

Poslednú podkategóriu tvoria útoky známeho vstupu, kedy útočník nijakým spôsobom neovplyvňuje údaje posielané generátorom a zároveň nijak nenarúša plynulý beh procesu. Avšak má možnosť pozorovať údaje, ktoré idú na vstup generátora. Príkladom môže byť nejaká aplikácia, ktorá využíva ako inicializačné dáta PRNG latenciu harddisku nachádzajúceho sa na sieti spolu s útočníkom, ktorý tieto údaje rovno pozoruje.

Útoky založené na kompromitácii vnútorného stavu generátora

Základným predpokladom tohto útoku je fakt, že útočník nejakým spôsobom narušil bezpečnosť systému a odhalil v určitom čase vnútorný stav generátora. Touto informáciou sa snaží obnoviť neznáme údaje z výstupu pred jeho kompromitáciou alebo aj po tom, ako zistil vnútorný stav generátora. V praxi sa stáva, že sa útoky založené na kompromitácii vnútorného stavu generátora vyskytujú väčšinou pri inicializácii vytvárania náhodných sekvencií v dôsledku nedostatočne veľkého vzorku entropie, čím môže byť vnútorný stav uhádnuteľný. Tak ako aj útoky na vstup generátora, aj tieto sa delia na podkategórie. Pri implementácii jednotlivých druhov generátorov sa snažíme docieliť to, aby boli schopné čo najviac odolať všetkým typom ďalej spomenutých útokov.

Prvú podkategóriu tvoria spätné útoky. Tie sa snažia pomocou známeho vnútorného stavu zistiť, aké hodnoty boli na výstupe pred jeho kompromitáciou.

Trvalo kompromitujúce útoky môžeme považovať za najviac škody páchajúce, lebo útočník pomocou znalosti vnútorného stavu je schopný obnoviť všetky vstupy, ktoré vznikli pred kompromitáciou, ale zároveň dokáže predvídať aj sekvencie, ktoré vznikli až následne, čiže útočník úplne prelomil zabezpečenie systému.

Ďalším typom útokov sú útoky opakovaným hádaním, kedy útočník využíva informáciu o vnútornom stave generátora v určitom čase t a jeho plynúcich výstupov na zistenie stavu v čase $t + \varepsilon$ dovtedy, dokým dokáže uhádnuť sekvenciu vstupných hodnôt v tomto časovom rozpätí.

Posledný typ útokov založených na kompromitácii vnútorného stavu generátora predstavujú útoky stretnutím uprostred, kedy útočník využíva informáciu o vnútornom stave generátora v určitých časoch t a $t + 2\varepsilon$ na zistenie stavu generátora v čase $t + \varepsilon$. Ide v podstate o kombináciu útokov opakovaným hádaním a spätných útokov.

1.4 Štatistické testy

Keďže nemálo kryptografických útokov využíva rôzne závislosti, ktoré sa vyskytujú vo výstupných sekvenciách generátorov a matematicky sa nedá dokázať, či výstupy sú skutočne náhodné dáta, je potrebné, aby boli tieto generátory podrobené štatistickými testami zisťujúcimi mieru náhodnosti daných výstupov. Základom každého štatistického testu je teda test nulovej hypotézy H_0 ktorá predpokladá, že vygenerovaná sekvencia je náhodná. Vzniklo už nespočetne veľa štatistických testov, ktoré zisťujú prítomnosť určitého vzoru vo výstupných údajoch, avšak ani pri jednom teste nemôžeme povedať, že sa jedná o kompletné otestovanie toho, či binárne sekvencie sú alebo nie sú skutočne náhodne vygenerované dáta. Naviac výsledok nulovej hypotézy nie je absolútny, skôr sa jedná o pravdepodobnostný

dôkaz, či vygenerovaná sekvencia uspela, preto treba výsledky testov porovnať s takzvanou kritickou hodnotou, ktorá rozhoduje, či bude generátor zamietnutý, alebo testom prejde. Ide väčšinou o percentuálne ohodnotenie výsledkov testov s hodnotou zvyčajne okolo 95 percent, lebo ak by bola kritická hodnota nastavená príliš malá, mohlo by sa stať, že by testom prešiel aj generátor, ktorého určitá špecifická vlastnosť mohla byť potencionálnym útočníkom rozpoznateľná. Ak by ale naopak bola kritická hodnota nastavená až moc veľká, mohlo by sa zase stať, že test zamietne generátor, ktorý generuje skutočne náhodné dáta.

Keďže otestovanie všetkými druhmi testov, ktoré existujú by bolo veľmi časovo náročné, vznikli sady (batérie) testov snažiace sa o čo najspoľahlivejšie a najefektívnejšie rozhodnutie, či generátor vytvára náhodné dáta alebo nie. Aj keď každé batérie testov majú svoje špecifické vlastnosti, stratégia štatistickej analýzy bitových sekvencií ostáva viacmenej rovnaká. Táto problematika je detailne rozpracovaná v publikácii *NIST Special publication 800-22* [12], kde sa proces štatistickej analýzy delí na 5 častí.

1. **Výber generátora**
   Generátor by mal vytvárať ako výstupné údaje bitové sekvencie. Ak nevytvára, hodnoty ním generované sa prevádzajú na binárny prúd.

GENERÁTORY
G G **G** G G G G G G G G G G

2. **Generovanie binárnej sekvencie**
   Do nejakého súboru ukladáme *m* sekvencií, každá s dĺžkou *n* vytvorených pomocou vybraného generátora.

BINÁRNE SEKVENCIE

$S_1 = \{0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, ... 1\}$
$S_2 = \{1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, ... 0\}$
$S_3 = \{1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, ... 1\}$
.
.
.
$S_m = \{0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, ... 1\}$

3. **Testovanie binárnych sekvencií**
   Každá z *m* sekvencií bude otestovaná všetkými testami z danej batérie testov.

BATÉRIE ŠTATISTICKÝCH TESTOV

	$Test_1$	$Test_2$	...	$Test_n$
$S_1 = \{$	$P_{1,1}$	$P_{1,2}$	...	$P_{1,n}\}$
$S_2 = \{$	$P_{2,1}$	$P_{2,2}$	...	$P_{2,n}\}$
$S_3 = \{$	$P_{3,1}$	$P_{3,2}$	...	$P_{3,n}\}$
		.		
		.		
		.		
$S_m = \{$	$P_{m,1}$	$P_{m,2}$	...	$P_{m,n}\}$

4. Porovnanie výsledkov testov

Výsledky testov (P-hodnoty) sú pravdepodobnostné hodnoty nachádzajúce sa v intervale $< 0 , 1 >$. Tie sa porovnajú s kritickými hodnotami.

P-HODNOTY

$P_{1,1} = 0{,}983$	$P_{1,2} = 0{,}976$	...	$P_{1,n} = 0{,}993$
$P_{2,1} = 0{,}942$	$P_{2,2} = 0{,}989$	...	$P_{2,n} = 0{,}998$
$P_{3,1} = 0{,}980$	$P_{3,2} = 0{,}991$	...	$P_{3,n} = 0{,}962$
	.		
	.		
	.		
$P_{m,1} = 0{,}973$	$P_{m,2} = 0{,}966$	...	$P_{m,n} = 0{,}997$

5. Rozhodnutie o splnení nulovej hypotézy

Pokiaľ všetky sekvencie prejdú úspešne všetkými testami, generátor je možné považovať za náhodný.

$$PASS = \{P_{1,1}\ P_{1,2}\ P_{1,3}\ P_{2,2\ ...}\}$$
$$FAIL = \{P_{2,1}\ P_{4,1}\ P_{8,3}\}$$

2 Operačný systém iOS

V tejto časti práce sa bližšie pozrieme na históriu, štruktúru a funkcionalitu operačného systému iOS, ktorý využívajú všetky mobilné zariadenia spoločnosti Apple Inc. Návrh tohto systému bol od začiatku spojený s tým, že ho budú využívať prenosné zariadenia s obmedzeným výkonom procesora, veľkosťou pamäte a neposlednom rade s obmedzeným zdrojom energie, preto je kladený veľmi veľký dôraz na prácu s týmito prostriedkami s úlohou čo najviac zefektívniť výkon pri čo najmenšej spotrebe energie.

V pamäti zariadenia zaberá systém a jeho vstavané aplikácie menej ako 500MB. Pri dnešných kapacitách pamätí zariadení napríklad najnovší iPhone 5 disponuje kapacitou až 64 GB, čo je priam zanedbateľná veľkosť. Tým, že sa vyvíjajú zariadenia s väčšou kapacitou pamäte, lepším výkonom a dlhšou výdržou batérie, vyvíja sa aj samostatný systém, ktorý dokáže s týmito prostriedkami efektívnejšie pracovať.

Jednou s najdôležitejších vlastností operačného systému iOS je uzavretosť. Užívateľ nemá možnosť zasahovať do systému rôznymi formami nadstavieb, poprípade možnosťou vlastnej individualizácie prostredia alebo funkcionality určitých prvkov systému. Aplikácie, ktoré sú vytvárané tretími stranami sa po dôkladnom otestovaní spoločnosťou Apple zobrazia na AppStore, čo je vlastne vstavaná aplikácia slúžiaca ako obchod so všetkými otestovanými aplikáciami kompatibilnými s iOS. Tým sa značne znižuje riziko narušenia bezpečnosti, vďaka čomu je operačný systém iOS jedným s najbezpečnejších systémov na trhu. Avšak existuje spôsob ako obísť uzavretosť systému a tým je Jailbreak, o ktorom budeme písať neskôr v podkapitole zameranej na bezpečnosť a možné riziká.

2.1 História

Vznik operačného systému iOS bol podmienený vznikom prvého mobilného telefónu značky Apple, ktorý mal v sebe zabudovaný tento systém. Keď 9. januára 2007 Steve Jobs predstavil na konferencii Macworld Conference & Expo v San Franciscu nový iPhone, zmienil len, že zariadenie "beží" na OS X a do 8. marca nemal tento operačný systém žiadny oficiálny názov.

Teraz sa pozrieme na jednotlivé verzie systému od jeho vzniku až po súčasnosť a zhrnieme všetky najdôležitejšie inovácie v rámci jednotlivých verzií.

iPhone OS 1

Jedná sa o úplne prvú verziu operačného systému, ktorá vyšla zároveň s predstavením prvého telefónu iPhone 29. júna 2007. Bol to vtedy revolučný systém, lebo ako prvý prišiel so softwarovou klávesnicou reagujúcou na dotyk prstu. Takýto typ klávesnice je dnes najrozšírenejším a najviac používaným prvkom všetkých značiek mobilných telefónov. Taktiež prevratné bolo využívanie viacdotykových gest, ako napríklad ťahanie dvoma prstami k sebe alebo od seba na zmenšenie, poprípade zväčšenie obsahu obrazovky. Užívateľské prostredie bolo veľmi intuitívne a jednoduché, systém reagoval bez nejakého výraznejšieho spomalenia a celková stabilita bola takmer bezchybová, čo sa zachovalo do dnešnej verzie. V tomto systéme ale nemal užívateľ možnosť zakúpenia aplikácií a bol nútený používať iba prednainštalované aplikácie. Príkladom takýchto aplikácii bol webový prehliadač Safari, ktorý umožňoval prehliadať klasické webové stránky, hudobný prehliadač, kalendár, budík, písanie poznámok a iné základné aplikácie. Avšak chýbali tu niektoré iné aplikácie pre niektorých používateľov dôležité ako napríklad diktafón, vďaka čomu vznikla snaha o prelomenie uzavretosti systému, čo sa hackerom aj zakrátko podarilo. Jednalo sa teda o prvý jailbreak mobilného telefónu iPhone.

iPhone OS 2

Druhá verzia operačného systému bola vydaná 11. júla 2008 a obsahovala niekoľko významných vylepšení. V prvom rade sa jednalo o zvýšenie celkovej bezpečnosti systému, takže jailbreak, ktorý bol použitý v prvej verzii systému sa stal nepoužiteľný. To samozrejme neodradilo hackerov od prelomenia bezpečnosti a tak za krátky čas vznikol ďalší jailbreak, tento raz pre druhú verziu systému. Prakticky pri každom update verzie až do najnovšej iOS 6 sa Apple snaží odstrániť určitú bezpečnostnú medzeru, vďaka ktorej sú hackeri schopní prelomiť uzavretosť systému. Najvýznamnejšou novinkou sa stala prítomnosť aplikácie Appstore, kde si užívatelia môžu kupovať aplikácie tretích strán. Týmto krokom zaistil Apple že aplikácie, ktoré chýbali v telefóne, teraz mohli byť dostupné a tým sa stal iPhone ešte populárnejším zariadením. Na začiatku roka 2013 sa z Appstoru stiahlo vyše 40 miliárd aplikácii, z toho takmer 20 miliárd stiahnutí bolo v priebehu roku 2012.

iPhone OS 3

S treťou verziou operačného systému prišla integrácia českého a slovenského jazyka. Už pred touto verziou bol ale neoficiálne dostupný pre užívateľov s jailbreakom. Napriek tomu, že tento update systému nepriniesol žiadne ohromujúce funkcie, aktualizoval systém veľkým množstvom pomerne malých vylepšení. Medzi niektoré vylepšenia patrí zlepšenie presnosti GPS prijímača zabudovaného v zariadení a pridanie magnetického kompasu. V rámci aplikácie Kamera pribudla možnosť natáčania videa a telefón už bol schopný okrem podpory SMS správ odosielať a prijímať správy MMS. Za zmienku stojí služba Find my iPhone, ktorú priniesla táto

verzia systému. Ide o službu, kedy pri strate telefónu má možnosť majiteľ zistiť približnú polohu zariadenia, poprípade na diaľku uzamknúť telefón, alebo dokonca aj vymazať všetky uložené dáta. Nachádza sa vo forme aplikácie zadarmo v AppStore.

iOS 4

17. júna 2010 Apple vydal štvrtú väčšiu aktualizáciu operačného systému. Zmenil sa aj názov a z iPhone OS vznikol už len iOS kvôli intregrácii systému na iPad. Tento update so sebou priniesol množstvo noviniek ako napríklad funkciu FaceTime, ktorá umožňuje videohovor medzi používateľmi, ďalej vytváranie priečinkov na pracovnej ploche telefónu. Pribudla taktiež možnosť zmeny tapety v pozadí, čo staršie verzie systému neumožňovali. Medzi aplikáciami znamenal ďalší míľnik predstavenie iBooks, slúžiaci ako obchod a čítačka elektronických publikácií. Najvýznamnejšiu funkciu ale predstavuje podpora multitaskingu, ktorý však funguje inak ako klasický multitasking na iných zariadeniach, pri ktorých sú všetky aplikácie stále aktívne. Na iOS to funguje tak, že väčšina aplikácií beží na pozadí iba krátku dobu, potom ich systém automaticky uspí a uloží si stav, v akom sa momentálne nachádzali. Tým pádom nedochádza k spotrebe takmer žiadnych systémových prostriedkov. Samozrejme niektoré aplikácie, ktoré musia bežať na pozadí, túto možnosť majú, ale musia mať explicitne pri vývoji zadané, že aplikácia bude podporovať background mód. Jedná sa o aplikácie na prehrávanie hudby, ďalej aplikácie, ktoré pracujú s GPS signálom, alebo aplikácie poskytujúce Voice-over-IP služby.

iOS 5

Táto verzia priniesla viac než 200 noviniek, čím sa jedná o zatiaľ najväčšiu aktualizáciu od vydania iPhone OS 1. Jednou z najväčších zmien sa dočkalo posielanie SMS a MMS správ. Pribudla funkcia iMessage, ktorá je integrovaná priamo do aplikácie Správy, čiže užívateľ posiela správy tak ako pred tým a nemusí otvárať samostatnú aplikáciu. Jedná sa vlastne o službu bezplatného posielania správ pomocou internetového pripojenia. Pri zapnutí služby sa automaticky zaregistruje telefónne číslo a Apple ID, ktoré sa ukladá na Apple server a keď užívateľ začne posielať niekomu správu, táto služba zistí, či dotyčná osoba má aktívnu funkciu iMessage, čo sa prejaví zmenou farby tlačidla na odoslanie správy, takže užívateľ je schopný rozlíšiť medzi klasickou a iMessage správou. Okrem toho pribudli ďalšie novinky ako napríklad centrum notifikácii, integrácia so sieťou Twitter, Wi-Fi synchronizácia zariadenia, inteligentná hlasová asistentka Siri a mnohé iné.

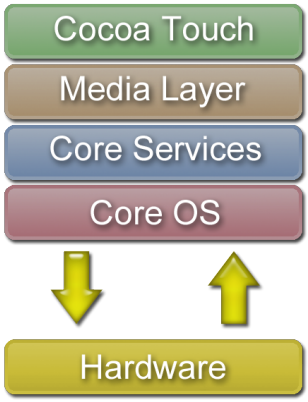
iOS 6

Jedná sa zatiaľ o poslednú veľkú aktualizáciu operačného systému. Pribudla integrácia Facebooku, fotoaparát najnovšie dokáže vytvárať panoramatické zábery, zlepšilo sa taktiež súkromie pre užívateľov a okrem iného, niektoré predinštalované

aplikácie sa dočkali zmeny grafického prostredia. Pribudli ale aj negatíva týkajúce sa zmeny aplikácie mapy, ktorú predtým spravovala firma Google. Navigácia v tejto aplikácii je často chybová a nespoľahlivá. No aj tak sa jedná o najspoľahlivejší a jeden z najrýchlejších operačných systémov pre prenosné zariadenia na trhu.

2.2 Štruktúra systému

Operačný systém iOS je prakticky derivát Mac OS X určeného pre osobné počítače, s ktorým zdieľa ako základ opensource systém Darwin, čiže patrí do kategórie systémov podobným Unixu a taktiež základ systému tvorí unixové jadro Mach, ktoré je zodpovedné za každý aspekt operačného systému, čiže spravuje virtuálnu pamäť, súborový systém, vlákna a mnohé iné. Jedná sa teda o odľahčenú verziu Mac OS X s jednoduchším grafickým rozhraním, ale s podporou dotykového ovládania. Systém je rozdelený na štyri základné abstraktné vrstvy, ktoré pre rozdielne fungujúci hardware vytvárajú spoločné rozhranie. Najvyššia vrstva sa nazýva Cocoa Touch a smerom k najnižšej vrstve sa postupne striedajú Media Layer, Core Services a Core OS.



Obrázok 1.0: Architekúra operačného systému iOS

Cocoa Touch
Táto vrstva obsahuje kľúčové frameworky slúžiace ako podpora pri tvorbe aplikácií. Ak aplikácia potrebuje poskytnúť určitú funkcionalitu, je vhodné preskúmať technológie dostupné v tejto vrstve, či sa tam takáto služba nachádza a ak nie, je nutné použiť prostriedky nižších vrstiev. Pri začiatku vývoja aplikácie sa teda začína práve s touto softwarovou vrstvou. Zároveň Cocoa Touch poskytuje služby na interakciu s užívateľom ako napríklad dotykové ovládanie a tvorí základ pri implementácii grafického rozhrania. Medzi vysokoúrovňové systémové služby tejto vrstvy patria aj funkcie ako multitasking, ochrana dát, kedy keď určitá aplikácia označí súbor ako chránený, automaticky sa ukladá do pamäte zariadenia v zašifrovanej podobe, ďalej

push notifikácie, čo je vlastne funkcia slúžiaca na upozornenie užívateľa o nových informáciách nejakej aplikácie, ktorá momentálne nie je spustená. Tie fungujú na princípe komunikácie servera výrobcu aplikácie, ktorej sa má poslať notifikácia a ktorý ju musí inicializovať so serverom Apple snažiacim sa túto notifikáciu odoslať zariadeniam. Okrem toho sú tu funkcie ako zdieľanie súborov a tiež peer-to-peer komunikácia medzi viacerými zariadeniami cez bezdrátovú technológiu Bluetooth. Táto vrstva má dokopy 8 frameworkov. Sú to Address Book UI Framework (poskytuje prednastavené rozhranie pre zobrazenie a správu kontaktov v telefóne), Event Kit UI Framework (poskytuje prednastavené rozhranie pre zobrazenie a správu udalostí v kalendári), Game Kit Framework (pridáva možnosť peer-to-peer sieťovej komunikácie), iAd Framework (vytvára bannerové reklamy zobrazujúce sa v aplikácii), Map Kit Framework (poskytuje mapové rozhranie, ktoré je možné integrovať do aplikácie), Message UI Framework (poskytuje prednastavené rozhranie na vytváranie a odosielanie SMS správ alebo e-mailov), Social Framework (pridáva integráciu sociálnych sietí) a UIKit Framework, ktorý je najdôležitejší framework Cocoa Touch vrstvy a tvorí kľúčovú úlohu pri implementácii grafického prostredia.

Media Layer

Táto vrstva obsahuje pokročilejšie grafické, zvukové a animačné technológie vytvorené pre čo najlepší užívateľský zážitok z používania aplikácie. Rozdeľujeme ju na tri kategórie podľa toho, akú multimediálnu technológiu využívajú. Prvá kategória sa zaoberá grafickými technológiami, ktoré poskytujú rôzne možnosti vytvárania grafiky a animácii v aplikácii, kedy už nestačí používať štandardné prerenderované obrázky a komponenty. V takýchto situáciách je možné využiť následné technológie a frameworky, Core Graphics[3] (používa sa na vytváranie 2D vektorovej grafiky a renderovanie obrázkov), Core Animation (vytvára pokročilejšie animácie), OpenGL ES (stará sa o hardwarovú akceleráciu 2D a 3D objektov), Core Text (ide o engine na vykresľovanie textu), Image I/O (podpora väčšiny grafických formátov), The Assets Library framework (umožňuje prístup k multimediálnej zložke užívateľa). Ďalšiu kategóriu tvoria technológie pre zvuk schopné prehrávať alebo aj nahrávať vysoko kvalitné zvukové záznamy a umožňujú využívať vibrácie, ak to hardware zariadenia podporuje. Systém poskytuje viacero možností pracovania so zvukom, lebo napríklad vysokoúrovňové frameworky (The Media Player framework a AV Foundation) nedovoľujú upravovať zvukové záznamy, prehrávať viackanálový alebo streamovaný zvuk a neponúkajú až takú flexibilitu ako nízkoúrovňové frameworky (OpenAL, Core Audio Framework).

[3] známy tiež ako Quartz

Core Services

Poskytuje vysokoúrovňové služby, ktoré používajú všetky aplikácie aj keď ich nemusia využívať priamo, mnoho častí systému je založených práve na týchto službách. Medzi kľúčové technológie dostupné v tejto vrstve patria SQLite (zjednodušená verzia SQL databázy, ktorá umožňuje ukladať rôzne užívateľské dáta), podpora XML (umožňuje spracovávať XML súbory prostredníctvom triedy NSXMLParser, dodatočnú podporu pre prácu s XML súbormi poskytuje libXML2 knižnica), iCloud úložisko (umožňuje ukladanie rôznych dát na Apple server a poskytuje synchronizáciu týchto súborov na všetky užívateľské zariadenia), objekty typu Block (jedná sa o jazykový konštrukt reprezentujúci anonymnú funkciu, často sa využívajú aj ako callback), Grand Central Dispatch (umožňuje správu úloh v aplikácii a podporu asynchrónneho programovania), Automatic Reference Counting (funkcia na kompilačnej úrovni starajúca sa o životnosť objektov), In App Purchase (stará sa o platby vo vnútri aplikácie) a lokalizačné služby, umožňujúce sledovanie aktuálnej polohy zariadenia prostredníctvom GPS signálu. Obsahuje taktiež aj množstvo frameworkov nevyhnutných pre pokročilejšie aplikácie ako napríklad CFNetwork framework, umožňujúci komunikáciu pomocou sieťového rozhrania a System Configuration Framework slúžiaci na zisťovanie dostupnosti pripojenia k mobilnej sieti alebo k internetu a jeho nastaveniam.

Core OS

Táto vrstva obsahuje nízkoúrovňové funkcie, na ktorých sú postavené všetky ostatné technológie nachádzajúce sa vo vyšších vrstvách systému. Dokonca aj keď sa tieto technológie nevyužívajú priamo, tak s najväčšou pravdepodobnosťou sú v aplikáciách použité ako súčasť iných frameworkov. V situáciách kedy sa treba explicitne zaoberať bezpečnosťou alebo komunikáciou s externým hardwarovým príslušenstvom je nutné použiť frameworky z tejto vrstvy. Medzi takéto nízkoúrovňové frameworky patrí Accelerate Framework (obsahuje rozhranie pre vykonávanie matematických výpočtov), External Accessory Framework (umožňuje komunikáciu s externými zariadeniami pomocou BlueTooth alebo cez kábel a získava informácie o pripojených zariadeniach) a Security Framework, ktorý sa stará o bezpečnosť ukladaných citlivých dát. Jedná sa o doplnok vstavaných bezpečnostných vlastností operačného systému.

2.3 Vývoj aplikácií na iOS

Vyvíjanie aplikácii na operačný systém iOS je pomerne špecifické zameranie, lebo je potrebné splniť niekoľko začiatočných podmienok. V prvom rade je nutné vlastniť počítač značky Apple minimálne s verziou operačného systému Mac OS X 10.5 Leopard. Avšak existuje spôsob ako nainštalovať Mac OS X na počítače iných

výrobcov, no táto metóda nie je spoľahlivá a niektoré komponenty systému nemusia pracovať korektne, preto sa to všeobecne neodporúča.

Ďalšou podmienkou, ktorú je potrebné splniť je registrácia u spoločnosti Apple ako vývojár. Aby sme mohli komerčne predávať aplikácie, musíme sa zaregistrovať do plateného programu, ktorý sa delí na dve kategórie: pre jednotlivcov, kde je ročný poplatok v súčasnosti $99 a pre spoločnosti za $299. Tieto dva programy sa líšia v počte developerských zariadení využívajúcich sa na vývoj softwaru. Pre spoločnosti je to neobmedzené množstvo, zatiaľ čo jednotlivec má k dispozícii len niekoľko zariadení. Apple ponúka aj možnosť registrácie zadarmo v rámci iOS Developer University programu určeného pre študentov a zamestnancov univerzít, ale vytvorené aplikácie sa nesmú poskytovať komerčne a môžu sa rozširovať iba medzi členmi vývojového tímu.

Po registrácii získame prístup k iPhone SDK, vývojovému prostrediu XCode, beta verziám operačného systému a okrem iného aj k dokumentáciam, ukážkovým zdrojovým kódom a množstvom tipov a návodov, ako postupovať pri tvorbe aplikácii. Všetky potrebné informácie sa nachádzajú na stránkach Apple Developer[5]. Taktiež získame certifikát, ktorým budeme podpisovať aplikáciu, aby sme ju boli schopní otestovať na jednotlivých zariadeniach, nie len na simulátoroch, ktoré sú súčasťou iPhone SDK a následne predávať na AppStore. Simulátory neobsahujú niektoré hardwarové prvky ako napríklad gyroskop alebo akcelerometer, preto ak aplikácia využíva tieto technológie, je nutné ju otestovať priamo na zariadení.

Súčasťou vývojového prostredia XCode je aj Interface Builder slúžiaci na vytváranie grafického rozhrania aplikácie, ktorý je veľmi prepracovaný a pomerne jednoduchý na používanie. Základ tvoria takzvané storyboardy, kde má vývojár možnosť vytvárať a následne spravovať všetky okná, ktoré zobrazuje aplikácia a taktiež mu poskytuje prvky na modelovanie interakcií medzi jednotlivými oknami.

Okrem toho je v rámci iPhone SDK dostupný nástroj Instruments slúžiaci na analýzu výkonu a správania sa aplikácie v reálnom čase, kedy má vývojár možnosť pozorovať okrem iného aj prípadné pamäťové úniky. Týmto spôsobom dokážeme jednoducho opraviť a optimalizovať aplikáciu.

2.3.1 Programovací jazyk

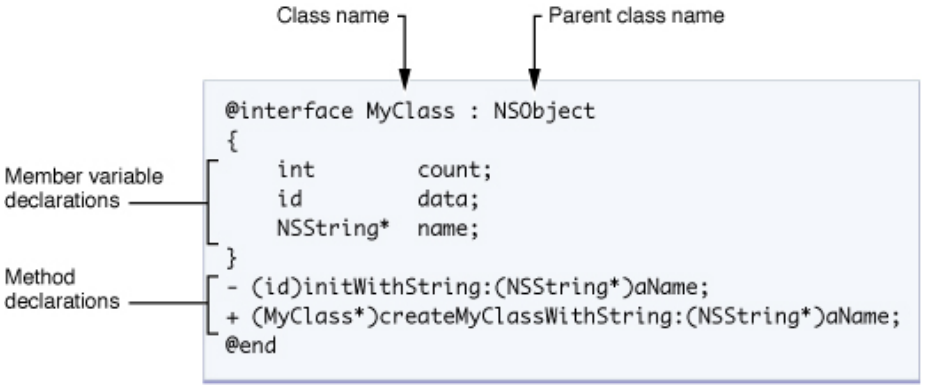
Na tvorbu aplikácií sa využíva programovací jazyk Objective-C. Jedná sa o objektovo orientovaný jazyk implementovaný ako rozšírenie jazyka C so systémom posielania správ prevzatý z jazyka Smalltalk a je tu prítomných mnoho tradičných objektovo orientovaných konceptov ako napríklad zapuzdrenie, dedičnosť alebo polymorfizmus. Kód, ktorý je napísaný iba v jazyku C môžeme kompilátorom Objective-C bez problémov preložiť a bude isto stopercentne funkčný. Tento fakt sa

využíva v aplikáciách relatívne často a dokonca niektoré knižnice sú napísane výlučne jazykom C.

Objective-C sa používa hlavne firmou Apple na svojich systémoch iOS a Mac OS X a zároveň ide o prakticky jediný oficiálny jazyk pre vytváranie aplikácií na operačný systém iOS, lebo základná knižnica Cocoa Touch je celá napísaná v tomto jazyku. Na iných platformách sa jazyk Objective-C viac menej vôbec nevyužíva.

Rovnako ako vo väčšine ostatných objektovo orientovaných jazykov, triedy v Objective-C poskytujú základnú konštrukciu pre zapuzdrenie niektorých dát s akciami, ktoré pôsobia na týchto dátach. Objekt je inštancia triedy a obsahuje v pamäti vlastnú kópiu ukazovateľov na metódy a premenné deklarované v tejto triede. Špecifikácie triedy v Objective-C vyžadujú dve samostatné veci, rozhranie a implementáciu.

Rozhranie časť obsahuje deklaráciu triedy a definuje premenné inštancie a metódy spojené s triedou. Rozhranie je zvyčajne v súbore s príponou *.h*. Implementácia obsahuje konkrétny kód metód triedy a nachádza sa zvyčajne v súbore s príponou *.m*.

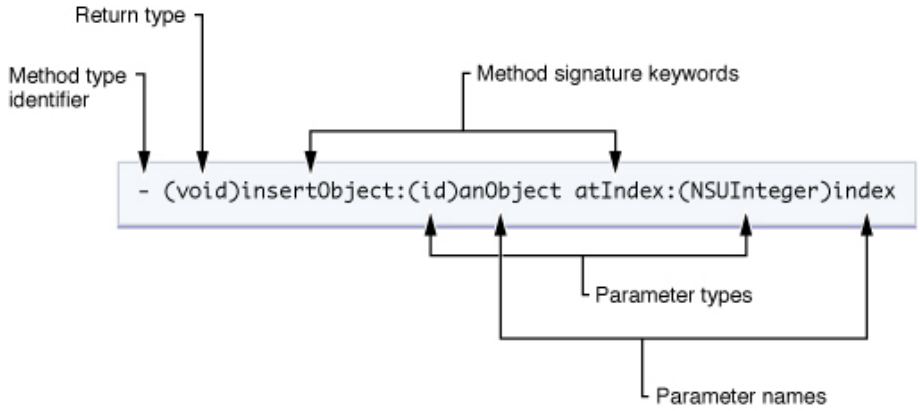


Obrázok 1.1: Rozhranie triedy

Obrázok 1.1 ukazuje syntax pre deklarovanie triedy s názvom MyClass, ktorá dedí zo základnej triedy NSObject. Deklarácia triedy začína smernicou *@interface* a končí smernicou *@end*. Za názvom triedy oddeleného dvojbodkou je názov rodičovskej triedy. Premenné sú deklarované v bloku kódu, ktorý je oddelený hranatými zátvorkami. Ďalej nasleduje zoznam metód deklarovaných triedou, kde bodkočiarka označuje koniec jednotlivých deklarácii.

Pri definovaní premenných z obrázku môžeme vidieť, že jazyk Objective-C podporuje silné ale aj slabé typovanie. Silné typovanie obsahuje v deklarácii názov triedy značiaci typ premennej, v tomto prípade sa jedná o riadok *NSString* name;* a slabo typované jazyky používajú objekt typu *id.* Výhoda slabého typovania spočíva v tom, že niekedy nevieme určiť aký konkrétny typ sa bude do premennej ukladať.

Mohlo by sa zdať, že takéto typovanie bude spôsobovať veľa problémov, no naopak, ponúka to oveľa väčšiu flexibilitu pri písaní programov. V triede Objective-C môžeme deklarovať dva typy metód, metódy inštancie, ktorej realizácia je vymedzená na konkrétny objekt a metódy triedy nevyžadujúce vytvorenie inštancie. Deklarácia metódy pozostáva z identifikátora typu metódy, návratového typu, jedného alebo viacerých kľúčových slov, kedy za dvojbodkou nasleduje premenná, ktorá sa posiela príjemcovi správy. Obrázok 1.2 ukazuje deklaráciu insertObject: atIndex: metódy inštancie.



Obrázok 1.2: Deklarácia metódy

Znamienko mínus na začiatku definície označuje, že sa jedná o metódu inštancie, zatiaľ čo znamienko plus značí metódu triedy. Nasleduje v zátvorke návratový typ. Za ním sú kľúčové slová *insertObject* a *atIndex* s dvoma parametrami. Ak metóda neobsahuje žiadne parametre, tak má len jedno kľúčové slovo s vynechanou dvojbodkou na jeho konci.

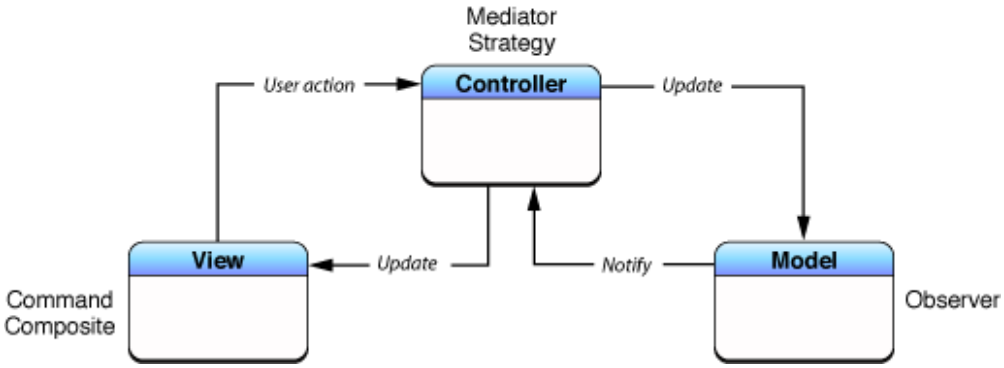
Volanie metódy sa uskutočňuje formou posielania správ objektu uzavretých do hranatých zátvoriek, kde sa na ľavej strane nachádza objekt, ktorý obdrží danú správu a na pravej konkrétna správa so všetkými požadovanými parametrami. Ak chceme napríklad poslať správu insertObject: atIndex: objektu uloženého do premennej myArray, mali by sme použiť nasledujúcu syntax:

[myArray insertObject:anObject atIndex:0];

Takýto spôsob posielania umožňuje vytváranie zložitejších vnorených správ, kde návratová hodnota jednej správy môže slúžiť ako príjemca inej správy alebo parameter posielaný nejakému objektu. Vyhneme sa tak ukladaniu dočasných výsledkov do lokálnych premenných.

2.3.2 Návrhový vzor Model-View-Controller

Model-View-Controller (MVC) je pomerne starý návrhový vzor, ktorého variácie vznikli pri vytvorení jazyka Smalltalk. Jedná sa o vysokoúrovňový vzor v tom, že pôsobí ako globálna architektúra aplikácie, triedi predmety podľa všeobecných úloh, ktoré hrajú v aplikácii a zároveň definuje spôsob komunikácie medzi objektmi. Programy využívajúce tento vzor sa dokážu celkovo viac prispôsobiť meniacim sa požiadavkám a sú ľahšie rozšíriteľné než programy, ktoré nie sú založené na MVC.



Obrázok 1.3: Schéma návrhového vzoru MVC

Ako už podľa obrázku 1.3 vidno, tento vzor sa delí na tri kategórie, model, view a controller. Okrem toho schéma znázorňuje komunikáciu medzi jednotlivými objektmi. Ak nejaká užívateľská akcia spôsobí zmenu v aplikácii, cez objekt typu controller sa aktualizuje alebo vytvorí nový model objekt. Rovnaký princíp platí aj opačne. Teraz si rozoberieme podrobnejšie jednotlivé objekty MVC:

Model

Ide o objekt zapuzdrujúci aplikačné dáta, ktorý zároveň definuje výpočty a operácie manipulujúce s týmito údajmi. Dobre navrhnutá MVC aplikácia má všetky dôležité údaje zapuzdrené v modelových objektoch. V ideálnom prípade model objekt by nemal mať explicitné pripojenie k užívateľskému rozhraniu slúžiacemu na prezentáciu alebo úpravu jeho dát. V praxi toto rozdelenie nemusí byť vždy to najlepšie. Ako príklad môžeme uviesť aplikáciu na kreslenie rôznych geometrických objektov, kde každý objekt predstavuje objekt typu model, v ktorom sú uložené dáta o tvare, veľkosti a všetkého potrebného na vykreslenie v aplikácii. V tomto prípade by sa teda objekty nemali spoliehať na view objekt, ktorý ho vykreslí, ale mali by už v sebe obsahovať metódu na vykreslenie a view objekt by len zavolal túto metódu vždy, keď chce zobraziť daný geometrický tvar.

View
Tento objekt vie, ako sa zobraziť v grafickom rozhraní aplikácie, je zodpovedný za interakciu medzi užívateľom a aplikáciou, taktiež prezentuje informácie z model objektu pre užívateľa, ktorý ich môže upravovať, ak to aplikácia dovolí. View objekt by nemal byť zodpovedný za ukladanie dát, ktoré zobrazuje. To však neznamená, že nikdy neukladá tieto dáta. Niekedy je dokonca vhodné uložiť dočasne zobrazené údaje kvôli výkonnostným dôvodom. View objekty majú tendenciu byť používané opakovane, preto je ich veľké množstvo vopred zadefinované v knižnici nástroja Interface Builder. Napríklad objekty typu *NSButton* predstavujú klasické tlačidlá, ktoré využívajú všetky aplikácie na iOS, čím sa zaručuje vysoká úroveň konzistencie vzhľadu a správania v rôznych aplikáciách.

Controller
Pôsobí ako prostredník medzi view a model objektom a uisťuje sa, že view objekt má k dispozícii dáta na zobrazenie z model objektu. Dokáže spravovať viacero objektov, nemusí sa vždy jednať o spojenie jeden model objekt a jeden view objekt. Taktiež vykonáva nastavenia a koordináciu úloh v aplikácii a správu životných cyklov iných objektov. V typickom návrhu MVC, keď užívateľ zadá alebo zmení hodnotu zobrazenú na displayi, posiela sa táto zmena do controller objektu, ktorý následne spracováva prijaté hodnoty. Ako príklad môžeme uviesť tlačidlo, ktoré sa pri stlačení nakrátko zablokuje. Keď ho užívateľ stlačí, informácia že bola vykonaná akcia sa predá controller objektu, ktorý pošle spätne príkaz na zablokovanie tohto tlačidla. Trieda *UIViewController* tvorí základ každej iOS aplikácie, lebo spravuje jednotlivé okná. Tá sa ďalej rozširuje na rôzne špecializované controllery, ako je napríklad *UITableViewController*, ktorý pracuje s tabuľkami.

2.4 Bezpečnosť v iOS aplikáciach

Apple navrhol platformu iOS tak, aby bola zaistená bezpečnosť už v samotnom jadre systému[6]. Udržanie informácií na mobilných zariadeniach v bezpečí je jedna z najdôležitejších vecí pre užívateľov, aby nedošlo k neoprávnenému prístupu k rôznym citlivým firemným, zákazníckym alebo bankovým údajom uložených v pamäti. iOS zariadenia poskytujú prísne bezpečnostné technológie a funkcie zaručujúce vysokú úroveň ochrany systému bez toho, aby bol narušený užívateľský dojem a ľahkosť ovládania. Mnoho bezpečnostných funkcií je v predvolenom nastavení povolených, takže odpadáva nutnosť vykonávania rozsiahlej konfigurácie a dokonca niektoré kľúčové prvky, ako je šifrovanie zariadenia, nie sú konfigurovateľné, takže sa nemôže stať, že by ich používatelia omylom zakázali. Nízkoúrovňové hardwarové a softwarové funkcie ochraňujú systém proti malware a vírusom, zatiaľ čo vysokoúrovňové funkcie umožňujú chránený prístup k dátam.

Veľa systémov je založených na priemyselnom štandarde bezpečnostných návrhov, avšak Apple vylepšil tieto návrhy a zaistil tak, že systém ako celok je viacej chránený bez narušenia jeho funkčnosti. Architektúra bezpečnosti sa rozdeľuje do štyroch základných pilierov zaisťujúcich zabezpečenie rôznych častí systému. Sú to ochrana zariadenia, ochrana dát, ochrana siete a ochrana aplikácií.

Ochrana zariadenia

Základ ochrany zariadenia tvorí možnosť nastaviť prístupový kód, ktorým sa prístroj odomyká. iOS podporuje dva druhy prístupových kódov. Poprvé štvormiestny digitálny kód a druhú, zložitejšiu alternatívu tvorí alfanumerický kód neobmedzenej dĺžky. Taktiež prístupový kód poskytuje entropiu pre šifrovacie kľúče, ktoré nie sú uložené v zariadení. To znamená, že útočník vlastniaci zariadenie nedokáže získať prístup k dátam bez zadania hesla. Naviac prístupový kód je chránený aj pred útokmi brutálnou silou na zistenie hesla, keďže po každej chybnej iterácii sa predlžuje doba čakania na jeho znovu zadanie. Pri šesťmiestnom alfanumerickom kóde vyskúšanie všetkých kombinácii bude trvať päť a pol roka nepretržitého útoku alebo dva a pol roka pri deväťmiestnom číselnom kóde, čiže takýto útok je pre zariadenie nevyhovujúci. Naviac systém umožňuje nastaviť, aby sa po zadaní desať krát nesprávneho hesla vymazali všetky údaje uložené v prístroji. Okrem toho ponúka systém aj mnoho iných nastavení pre prístupové heslá ako napríklad čas, kedy vyprší jeho platnosť pri nečinnosti alebo čas, po ktorom ho treba znova zadať. Ďalšia vlastnosť ochrany zariadenia je schopnosť blokovania iTunes, zálohovania do iCloud a podobne.

Ochrana dát

Táto ochrana patrí medzi najdôležitejšie v systéme, aby nedošlo ku kompromitácii citlivých informácii pri možných útokoch. Využíva pri tom rôzne kryptografické funkcie. Vo všeobecnosti sú ale kryptografické operácie veľmi zložité a môžu spomaliť výkon zariadenia a zvýšiť spotrebu energie, ak nie sú navrhnuté dostatočne korektne. Avšak každé zariadenie s operačným systémom iOS má vlastný šifrovací AES 256 motor zabudovaný do DMA cesty medzi flash pamäťou na ukladanie dát a systémovou pamäťou, takže prípadné šifrovanie prebieha počas ukladania, čím sa stáva vysoko účinné. Jedinečný identifikátor zariadenia (UID) spolu s identifikátorom skupiny zariadení (GID) tvoria 256 bitové AES kľúče, ktoré sa ukladajú do procesora pri výrobe prístroja a sú prístupné iba z AES motora.[6] Žiadny firmware ani software nie je schopný tieto identifikátory prečítať priamo. Môžu vidieť iba výsledky šifrovania, takže neexistuje spôsob, ako ich zistiť. UID kľúč je unikátny pre každé zariadenie a nie je zaznamenaný spoločnosťou Apple, ani žiadnym z jej dodávateľov. Týmto sa zaručuje najvyššia možná ochrana dát. Pre vývojárov je štandardne k dispozícii kryptografická knižnica CommonCrypto obsahujúca širokú škálu kryptografických algoritmov.

Ochrana siete

Okrem opatrení Apple prijatých na ochranu dát uložených na iOS zariadeniach existuje veľa opatrení pre zabezpečenie siete, kedy sa citlivé informácie prenášajú z alebo do prístroja pomocou rôznych technológií ako napríklad cez Wi-Fi, Bluetooth alebo mobilnú dátovú sieť. Keďže prístroje so systémom iOS sú prenosné, je veľká pravdepodobnosť, že sa takýto prenos informácií bude konať cez nezabezpečenú sieť kdekoľvek na svete, preto je nevyhnutné overiť, či sú autorizované a chránené počas prenosu. iOS používa a poskytuje prístup k štandardným sieťovým protokolom pre autentizáciu, autorizáciu a šifrovanú komunikáciu. Zároveň poskytuje tiež osvedčené technológie a najnovšie štandardy na dosiahnutie týchto bezpečnostných cieľov ako pre Wi-Fi tak aj mobilnú dátovú sieť. Na iných platformách je potrebný k ochrane otvorených komunikačných portov firewall, no na operačných systémoch iOS ho netreba, lebo vždy dochádza k obmedzeniu počtu načúvajúcich portov a zároveň k odstráneniu zbytočných sieťových nástrojov. Naviac komunikácia pomocou technológií vytvorených spoločnosťou Apple ako je napríklad FaceTime alebo iMessage je plne šifrovaná a overená.

Ochrana aplikácií

Okrem toho, že každá aplikácia, ktorá je dostupná na AppStore prešla dôkladnou kontrolou, či neobsahuje škodlivý software alebo inak nenarúša bezpečnosť zariadenia, systém ponúka niekoľko mechanizmov na ochranu. Hlavný prvok zabezpečenia je izolovanie aplikácií medzi sebou pomocou sandboxu čo znamená, že každá aplikácia má definované miesto v pamäti a nemôže pristupovať k údajom mimo prideleného priestoru. Nemôže sa teda stať, že by aplikácia pristupovala k inej aplikácii priamo, no iOS ponúka možnosť interakcie cez rozhranie API, ktoré poskytuje. Čo však systém nezaručuje je kompromitácia samostatnej aplikácie, kde o zabezpečenie sa stará priamo vývojár. Avšak aj keď útočník prelomí bezpečnosť, nemá prístup k celému zariadeniu ako tomu je v mnoho iných operačných systémoch. No môže sa dostať k niektorým citlivým údajom systému pokiaľ ich daná aplikácia využíva. Každá aplikácia sa podpisuje certifikátom vývojára, ktorý slúži k uchovaniu celistvosti binárneho kódu a je vždy testovaný pri jej spustení. Certifikát sa vždy viaže na konkrétne ID vývojára a získava sa po prihlásení na webových stránkach vývojového programu Apple. Ak sa zistí, že nejaká aplikácia je napadnutá útočníkom alebo sa správa škodlivo, pomocou tohto certifikátu môžeme autora aplikácie explicitne určiť.

Aj keď z vyššie opísanej architektúry bezpečnosti vyzerá , že tento systém nemá žiadnu chybu a je ideálny, hackerom sa podarilo zistiť niektoré bezpečnostné chyby [7].

Za zmienku stojí hlavne jailbreak. Jedná sa o proces odstránenia obmedzenia zariadení Apple s operačným systémom iOS pomocou hardwarovej a softwarovej

zraniteľnosti, teda systém sa stáva otvoreným. Jailbreakom sa odstráni okrem iného aj proces overovania certifikátov aplikácií, takže je možné inštalovať do zariadenia neautorizované aplikácie. Väčšina výskumu bezpečnosti systému je vykonávaná na zariadeniach s jailbreakom. Je to preto, lebo otvorený operačný systém iOS umožňuje koreňový prístup a tým má užívateľ úplnú kontrolu nad zariadením.

Najväčšie riziko, ktoré predstavuje jailbreak je značne zvýšená možnosť napadnutia prístroja potenciálnym útočníkom. Dokonca bola objavená chyba, ktorá pri určitom procese jailbreaku umožňovala hackerom vzdialene nasadiť malware do zariadenia. Taktiež prvý červ v iOS systéme sa objavil práve na takto otvorených prístrojoch. Aj napriek výhodám, ktoré ponúka otvorený systém sa vo všeobecnosti neodporúča jailbreak, hlavne kvôli schopnosti narušiť bezpečnosť a stabilitu celého zariadenia.

3 Generátor pseudonáhodných čísel ANSI X9.31

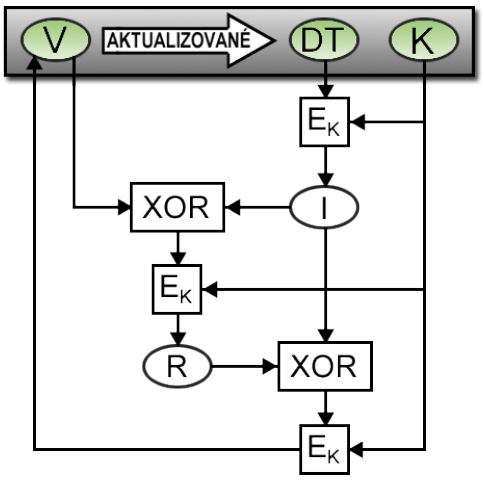
K jedným z najznámejších generátorov pseudonáhodných čísel v oblasti kryptografie patrí ANSI X9.31, ktorý sa často používa pre rôznorodé účely, typicky v sektore bankovníctva. Nemá zadefinované presné meno, no označuje sa podľa dokumentu *NIST-Recommended Random Number Generator Based on ANSI X9.31 Appendix A.2.4 Using the 3-Key Triple DES and AES Algorithms*[4], v ktorom nájdeme jeho špecifikáciu, ako *ANSIX9.31 PRNG*.

Vyvinuli ho pôvodne na generovanie pseudonáhodných kľúčov slúžiacich algoritmu DES alebo inicializačným vektorom. Bežne sa v mechanizme využívajú blokové šifry AES alebo 3DES, ktoré sú uvedené aj v štandarde [3] rozlišujúce sa hlavne rôznou dĺžkou šifrovaných blokov, kde AES používa na šifrovanie 128bitové bloky a 3DES 64bitové bloky. Existujú už aj iné varianty, ktoré sú založené na hašovacích funkciách, ale v našej implementácii budeme používať blokovú šifru AES, preto si ďalej rozoberieme práve túto verziu.

3.1 ANSI X9.31 PRNG so šifrou AES

Ako prvý krok si zadefinujeme jednotlivé symboly, s ktorými sa budeme stretávať v tejto časti práce.

- K – 128bitový AES kľúč
- $E_K(X)$ – šifrovacia funkcia AES s kľúčom K aplikovaná na 128bitový blok X
- V – 128bitové semienko
- DT – 128bitová časová známka
- I – medzivýsledok
- XOR – exkluzívna disjunkcia
- R – pseudonáhodné číslo



Obrázok 3.1: Schéma generátora ANSI X9.31

Z obrázku 3.1 môžeme vidieť princíp fungovania generátora. Pseudonáhodné číslo sa vygeneruje nasledovne $R = E_K(I \; XOR \; V)$, kde $I = E_K(DT)$ a aktualizácia semienka prebieha ako $V = E_K(R \; XOR \; I)$. Vstupné hodnoty algoritmu sú teda *V*, *DT* a *K*. Výstupom generátora je 128 bitov pseudonáhodných dát, ktoré priamo ovplyvňujú aktualizáciu hodnoty semienka.

3.2 Implementácia generátora v iOS

Aplikácia využívajúca generátor ANSI X9.31 bola naimplementovaná v rámci praktickej ukážky jeho funkčnosti, preto sa nebudeme zameriavať na jej štruktúru alebo grafické rozhranie. Naprogramovaný algoritmus nám poslúži ako model, kde si rozoberieme bezpečnosť tohto generátora a jeho možné riziká útokov.

Implementácia sa delí na dve triedy obsahujúce všetky potrebné metódy a objekty, ktoré využíva generátor. Prvá trieda má názov *X9_31_AES*. V nej sa nachádza samotný algoritmus generátora a všetky jeho premenné vrátane zásobníka entropie. Druhá trieda s názvom *NSData+Encryption* je vlastne kategória, ktorá slúži na rozšírenie štandardnej triedy *NSData* o dve metódy potrebné pre funkčnosť generátora a to *AES128EncryptWithKey:* a *XOR:*.

V implementácii aplikácie sa využívajú kryptografické funkcie AES, ktorú používa samotný algoritmus a SHA1 na hašovanie zásobníka entropie. Zásobník entropie o veľkosti 320 bitov sa napĺňa vždy pri prvej požiadavke na vygenerovanie pseudonáhodného čísla, teda pri spustení generátora. Obsahuje náhodné hodnoty z audio streamu, z akcelerometra a systémové údaje týkajúce sa správy pamäti v zariadení. Zo zásobníka sa po naplnení inicializujú premenné tak, že prvých 128 bitov sa uloží do *K*, ktorá slúži ako AES kľúč a musí mať preto vždy fixný počet bitov, v tomto prípade 128 bitov. Ďalších 128 bitov sa použije len raz pri prvom

spustení generátora na naplnenie semienka *V*, ktoré sa pri nasledujúcich iteráciách algoritmu aktualizuje pomocou pseudonáhodného čísla a medzivýsledku predchádzajúcej iterácie. Posledných 64 bitov uložených v zásobníku použijeme na doplnenie časovej známky *DT* z dôvodu, že operačný systém iOS generuje časovú známku maximálne o veľkosti 64 bitov. Naprogramovaná kostra algoritmu vyzerá nasledovne:

I = [DT AES128EncryptWithKey:K];

R = [[I XOR: V] AES128EncryptWithKey:K];

V = [[R XOR: I] AES128EncryptWithKey:K];

3.3 Bezpečnosť generátora

Naimplmentovaný generátor nemôžeme považovať za kryptograficky bezpečný, lebo ak dôjde ku kompromitácii jeho vnútorného stavu, nie je zaistená ani spätná ani dopredná bezpečnosť.

Pokiaľ ale nedôjde ku kompromitácii vnútorného stavu, útočník musí použiť buď priamy kryptoanalytický útok, kedy sa snaží o kryptoanalýzu šifrovacieho algoritmu AES, čo sa zatiaľ nikomu nepodarilo, alebo môže útočiť na vstup pomocou prehrania vstupných údajov tak, že po určitom množstve výstupných dát bude útočník schopný rozlíšiť medzi pseudonáhodnou sekvenciou generovanou algoritmom a skutočne náhodnými dátami. Táto hrozba je ale reálna iba v prípade, že útočník pozná kľúč *K* a dokáže upraviť generovanie časovej známky *DT* tak, aby táto hodnota bola stále konštantná.

Avšak pokiaľ útočník kompromituje vnútorný stav generátora, dokáže určiť aké boli predchádzajúce výstupné údaje vďaka tomu, že AES kľúč *K* sa v priebehu chodu aplikácie vôbec nemení. K čím vzdialenejším hodnotám sa chceme dostať, tým to bude ťažšie, lebo časová známka *DT* sa pravidelne mení, čo značí, že spätná bezpečnosť je priamo závislá na bezpečnosti práve tejto hodnoty.

Keďže časová známka sa skladá z dvoch blokov o veľkostiach 64 bitov, z toho jedna polovica je fixná, lebo operačný systém je pri generovaní obmedzený maximálnou veľkosťou časovej známky, útočník potrebuje odhadnúť iba túto 64bitovú polovicu. Fixnú časť pozná z dôvodu kompromitácie vnútorného stavu generátora.

Jediné, o čo sa útočník zaujíma je čo najpresnejší odhad časovej požiadavky na generovanie, kedy sa časová známka naplní náhodnými údajmi a tým je schopný odhadnúť, ako budú vyzerať budúce výstupy generátora. Ak by uhádol 128bitovú časovú známku *DT* s presnosťou na sekundu, musel by vyskúšať teoreticky 8000

možností, čo sa rovná približne 13 bitom entropie. Avšak ak sa odhad pohybuje v rozmedzí niekoľkých minút, počet teoretických možností sa zvýši na rádovo niekoľko sto tisíc.

Najdôležitejšou vlastnosťou generátora u ktorého poznáme jeho vnútorný stav je to, že sa už nikdy nezotaví z takto kompromitovaného stavu. Je to spôsobené tým, že jeho vnútorný stav pravidelne neaktualizujeme. Napríklad občasné vygenerovanie nového AES kľúča *K* by podstatne prispelo k zlepšeniu jeho bezpečnosti.

3.4 Zhrnutie

Generátor ANSI X9.31, ktorý sme implementovali v aplikácii z vyššie uvedených dôvodov, nemôžeme považovať za kryptograficky bezpečný, preto ho nedoporučujeme používať v praxi na tvorbu pseudonáhodných údajov slúžiacich v rámci systémov alebo aplikácií, ktoré potrebujú vyššiu mieru bezpečnosti.

Avšak existuje mnoho spôsobov ako nami implementovaný generátor dodatočne vylepšiť, aby poskytoval väčšiu ochranu proti útokom, no to sme nemali ani v úmysle. Hlavným prvkom vytvorenia aplikácie bolo osvojenie si základov programovania na operačný systém iOS a zistenie, aké zdroje náhodnosti ponúkajú zariadenia, na ktorých bude aplikácia spustiteľná. Zdrojový kód našej aplikácie sa nachádza na priloženom CD.

Niektoré časti aplikácie ako napríklad obsluha a získavanie náhodných údajov z akcelerometra, audio streamu a z dotykovej obrazovky sú použité v implementácii ďalšieho pokročilejšieho a kryptograficky oveľa bezpečnejšieho generátora, ktorému sa budeme venovať v nasledujúcej kapitole tejto práce.

4 Generátor pseudonáhodných čísel Fortuna

V tejto kapitole sa bližšie zoznámime s generátorom pseudonáhodných čísel Fortuna, ktorý ponúka oveľa vyššiu mieru bezpečnosti ako generátor ANSI X9.31, popísaný v predchádzajúcej kapitole. Generátor Fortuna vznikol modifikáciou staršieho generátora Yarrow[2], ktorý vytvorili známi kryptológovia Neils Ferguson a Bruce Schneier. Oni stáli taktiež aj za vznikom jeho nástupcu. Hlavným dôvodom modifikácie generátora Yarrow bolo to, že manipulácia so zdrojmi entropie nebola príliš efektívna. Bolo to spôsobené tým, že algoritmus musel odhadovať, aké veľké množstvo entropie sa dodáva od jednotlivých zdrojov náhodnosti. Najlepšie bolo, ak sa takýto odhad robil za behu algoritmu, čo však je principiálne veľmi náročné a znamenalo to, že výsledná implementácia mala problémy s analýzou bezpečnosti.

Avšak generátor Fortuna sa dočkal zlepšenia tohto konceptu práce so zdrojmi entropie a už nie je závislý na presnosti odhadov množstva náhodných dát. Rovnako sa zmenil systém ukladania entropie do zásobníkov. Autori architektúru generátora a jeho doporučenie k implementácii zosumarizovali v knihe *Practical Cryptography* [1], kde však chýba dôkaz o kryptografickej bezpečnosti generátora a taktiež ho neuvádza ani žiadny iný zdroj. No aj cez tento chýbajúci dôkaz sa Fortuna PRNG považuje za kryptograficky bezpečný generátor pseudonáhodných čísel.

4.1 Architektúra PRNG Fortuna

Návrh generátora Fortuna je rozdelený na tri komponenty, ktoré sú rovnocenné pri generovaní pseudonáhodných dát so zameraním na bezpečnosť.

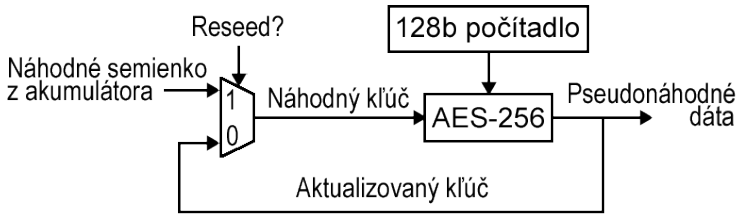
Prvá komponenta tvorí samotný generátor, ktorý obsahuje jednoduchý algoritmus pre vytváranie ľubovoľne dlhých pseudonáhodných sekvencií zo semienka fixnej dĺžky. Takýto generátor môžeme použiť aj samostatne bez ostatných komponent pokiaľ nehľadíme na to, aby boli vygenerované dáta bezpečné, čo využívajú napríklad rôzne formy simulácii.

Druhá komponenta sa nazýva akumulátor, ktorého funkcia je spravovať zdroje náhodnosti a následné ukladanie ich generovaných dát do zásobníkov entropie. Mimo tieto vlastnosti tiež pravidelne reinicializuje generátor.

Poslednú komponentu tvorí inicializačný súbor, ktorý má za úlohu nahradiť akumulátor, ak ešte nevygeneroval dostatočne veľké množstvo entropie pre korektné fungovanie algoritmu. Obsahuje náhodné dáta, ktoré vznikli pri predchádzajúcich chodoch aplikácie.

V nasledujúcej časti práce sa do detailov zameriame na jednotlivé komponenty.

4.1.1 Generátor



Obrázok 4.1: Schéma generátora Fortuna PRNG

Ako vidno z obrázku 4.1 generátor tvorí jednoduchý algoritmus, konkrétne sa jedná o blokovú šifru, ktorá sa nachádza v counter móde. V tomto prípade sa ako bloková šifra používa AES s 256bitovým kľúčom a 128bitovým blokom, čo je vlastne šifra doporučená priamo autormi generátora Fortuna. Jeho tajný vnútorný stav sa skladá z dvoch hodnôt a to zo 128bitového počítadla a 256bitového kľúča.

Dopredná bezpečnosť je zaručená tým, že sa zahadzuje kľúč, ktorý slúžil blokovej šifre AES na generovanie pseudonáhodných dát. Takýmto spôsobom aj keby útočník kompromituje vnútorný stav generátora, nemôže sa dostať k predchádzajúcim vygenerovaným dátam. Nový kľúč sa vytvára pomocou dvoch blokov generovaných vždy po tom, ako algoritmus dostane požiadavku na generovanie pseudonáhodných dát. Akumulátor má na starosti zase spätnú bezpečnosť.

Štatistické vlastnosti vygenerovaných blokov majú ale jeden problém. Tým problémom je jedinečnosť výstupu šifrovacej funkcie v counter móde. Ak máme konštantný kľúč a počítadlo, ktoré rastie stále rovnako, neexistuje možnosť, že by sa dva rozdielne výstupné bloky zhodovali, čo ale v rámci náhodného správania nie je dobré, lebo štatisticky táto situácia môže nastať. Táto chyba sa ale až tak príliš neprejaví, ak obmedzíme množstvo dát vygenerovaných bez zmeny kľúča. Autori generátora doporučujú obmedziť toto množstvo na 1 MB, čo je kompromisné riešenie, aby sa defekt neprejavil až v takej miere a zároveň, aby veľkosť vygenerovaných dát bola dostačujúca.

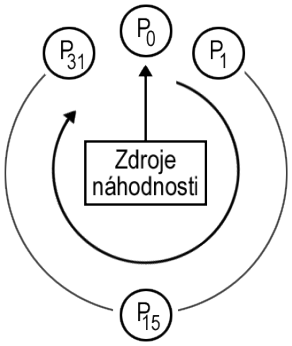
Autori ďalej nedoporučujú resetovať zakaždým spolu s kľúčom aj počítadlo, lebo ak by sa tak stalo, mohli by sa pri náhodnom opakovaní kľúča vytvoriť krátke cykly. Keďže má počítadlo veľkosť 128 bitov čo znamená, že na pretečenie by muselo byť vygenerovaných 2^{128} blokov pseudonáhodných údajov, tak v praxi nehrozí, že by táto situácia nastala z dôvodu výpočtových možností a životnosti zariadení.

Aplikačné rozhranie Fortuna PRNG sa skladá zo štyroch funkcií, čiže sa jedná o veľmi jednoduchú schému. Podrobnejšie sú tieto funkcie rozobraté v knihe [1, str. 164–167], ale pre pochopenie si ich stručne popíšeme.

Ako prvú funkciu generátora voláme *InitializeGenerator* slúžiacu na vynulovanie počítadla a kľúča, čím sa dostane do stavu, kedy ešte nie je schopný generovať náhodné dáta a čaká na zavolanie funkcie *Reseed*, ktorá má za úlohu inkrementovať počítadlo a nastaviť kľúč.

Po zavolaní týchto dvoch funkcií sa generátor nachádza v stave, kedy je už schopný vytvárať pseudonáhodné bloky volaním funkcie *PseudoRandomData*, ktorá sa stará mimo generovania blokov aj o aktualizáciu kľúča. *PseudoRandomData* volá funkciu *GenerateBlocks* zaisťujúcu potrebné množstvo dát, ktoré vznikajú šifrovaním počítadla.

4.1.2 Akumulátor



Obrázok 4.2: Schéma akumulátora Fortuna PRNG

Druhá komponenta sa nazýva akumulátor, ktorý má za úlohu zbierať údaje do zásobníkov entropie z rôznych zdrojov náhodnosti. Taktiež sa stará o pravidelnú reinicializáciu generátora.

Zdrojov náhodností má byť podľa autorov čo najviac, lebo sa môže stať, že nejaká časť zdrojov bude dodávať nenáhodné údaje, napríklad kvôli tomu, že ich potenciálny útočník bude vedieť predvídať. Avšak generátor funguje správne, ak existuje aspoň jeden zdroj náhodností, ku ktorému nemá útočník prístup, čiže nemôže nejakým spôsobom ovplyvniť alebo predvídať generované dáta. Takéto korektné fungovanie zaisťuje 32 zásobníkov entropie a reinicializačný mechanizmus.

Do zásobníkov sa ukladajú rovnomerne náhodné dáta, ktoré generujú rôzne zdroje náhodností. Podľa obrázku 4.2 vidno, že napĺňanie jednotlivých zásobníkov prebieha cyklicky, takže ak útočník nedokáže predvídať alebo iným spôsobom znehodnotiť výstupné dáta aspoň jedného zdroja, ten zaplní všetky zásobníky, ktoré obsahujú útočníkom známe údaje a tým pádom ostanú bezpečné pre použitie.

Realizačný mechanizmus funguje na princípe toho, že sa reinicializácie postupne očíslujú a keď 2^i delí toto počítadlo, tak sa použije na získanie dát zásobník P_i, čiže v praxi to vyzerá tak, že sa zásobník P_0 použije pri každej reinicializácii, zásobník P_1 zase pri každej druhej a takto to exponenciálne rastie. Tým pádom sa stane, že počas jednej reinicializácie sa použije viacero zásobníkov. Tie zlúčením dát, ktoré obsahujú, vytvoria výslednú sekvenciu.

Vďaka realizačnému mechanizmu nie je na zásobníkoch rovnaké množstvo dát, lebo napríklad zásobník P_1 sa používa dva krát častejšie ako P_2, z čoho vyplýva, že má zhruba dvakrát menej entropie. Táto vlastnosť opäť zvyšuje bezpečnosť celého generátora, lebo ak by náhodou útočník odhadol alebo iným spôsobom zistil obsah zásobníka P_0, mohol by predvídať výstup generátora len do ďalšej reinicializácie, keďže sa zreťazí obsah P_0 s ďalším zásobníkom, ktorý určite obsahuje viac entropie a pre útočníka je oveľa ťažšie odhadnúť výstup. Ak by sa mu to aj podarilo, každou novou reinicializáciou sa stáva výstup menej známym až do bodu, kedy máme istotu, že ho nedokáže narušiť. Takéto zotavenie závisí od množstva entropie generujúceho zdrojmi, ktoré sú neovplyvnené útočníkom.

Odhad množstva dát v najhoršom prípade, ktoré potrebuje generátor na to, aby sa zotavil z kompromitácie, je podľa autorov 2^{13} bitov [1, str. 170]. Avšak ak by neovplyvnené zdroje náhodností po zavolaní 2^{32} krát reinicializácie nedodali do zásobníkov dostatok dát, potom by ani posledný zásobník P_{31} neobsahoval toľko náhodných vzoriek, aby zabránil útočníkovi predvídať budúci výstup. To by sa ale mohlo stať iba v prípade, že útočník vloží do zásobníkov veľmi veľké množstvo vlastných dát. Autori ale brali toto na vedomie a ako prvok ochrany proti takýmto typom útokov doporučujú zredukovať počet reinicializácii na maximálne 10 za sekundu, čo by znamenalo, že zásobník P_{31} by sa použil po prvý krát po trinástich rokoch.

Akumulátor sa skladá zase z troch funkcií. Prvá funkcia sa nazýva *InitializePRNG*, ktorá slúži na vynulovanie počítadla reinicializácií a zavolania funkcie *InitializeGenerator*.

Druhá funkcia je *RandomData* obsluhujúca proces vytvárania pseudonáhodných dát, čiže sa jedná o funkciu, ktorá je pre používateľa najdôležitejšia. Tá na začiatku zisťuje, či je možné reinicializovať generátor tak, že overí kedy naposledy došlo k reinicializácii, pričom minimálny čas je 100 ms a či obsahuje prvý zásobník P_0 potrebných 256 bitov a viac. Je teda potrebné na začiatku odhadovať obsah zásobníka, ale nemá to nejaký zásadný vplyv na bezpečnosť návrhu.

Tretia funkcia *AddRandomEvent* slúži na pridávanie entropie do zásobníkov zdrojmi náhodnosti obsluhovanými cyklicky pomocou unikátneho identifikačného čísla, ktorým sú tieto zdroje označené.

4.1.3 Inicializačný súbor

Inicializačný súbor tvorí poslednú komponentu slúžiacu pre správny počiatočný chod generátora, kedy akumulátor ešte nedodal dostatočné množstvo entropie. Obsahuje permanentné dáta, ktoré vznikli pri predchádzajúcich behoch aplikácie, čiže sa jedná o trvalý zásobník entropie. Tento koncept je teoreticky veľmi jednoduchý, ale v praxi závisí hlavne od operačného systému, kde jeho implementácia nemusí byť vôbec triviálna.

Aby bola bezpečnosť súboru čo najväčšia, je potrebné splniť niektoré požiadavky, bez ktorých by mal útočník pomerne dobrú šancu zistiť aspoň časť dát uložených v súbore. Napríklad jednou z takýchto požiadaviek je prístup k inicializačnému súboru. Ten by mala mať iba aplikácia implementujúca generátor, lebo inak by útočník mohol bez väčších problémov odhadnúť prvé výstupné pseudonáhodné dáta, keďže by vedel počiatočný stav generátora a ak by mimo to mal pod kontrolou aj niektoré zdroje náhodností, odhad výstupných údajov by bol po určitú dobu útočníkovi známy. Preto je veľmi dôležité, aby nebol schopný tento súbor prečítať alebo dokonca pozmeniť.

Ďalšou veľmi podstatnou požiadavkou je potreba aktualizovať súbor hneď po jeho použití a to ešte pred tým, ako je schopný generovať pseudonáhodné dáta pre používateľa, ktorý o to žiada. Je to z toho dôvodu, že pokiaľ by sa súbor neaktualizoval a na zásobníkoch by ešte nebolo dostatok entropie, vnútorný stav generátora by ostal stále rovnaký, čo je nevhodné pre bezpečnosť. Problémom aktualizácie býva bufferovanie operácii v rámci nízkoúrovňového zápisu na disk, lebo nie sme si nikdy istý, či už skutočne došlo k zmene dát. Taktiež by sa nemal inicializačný súbor zálohovať, aby sa nestalo, že generátor bude mať dvakrát rovnaký vnútorný stav.

Najväčší problém nastáva, ak sa generátor používa po prvý krát, lebo inicializačný súbor neobsahuje žiadne dáta, na čo neexistuje nejaké bezpečné štandardné riešenie. Naviac hneď po spustení generátora je vysoká pravdepodobnosť, že vygenerovaná pseudonáhodná sekvencia poslúži na rôzne veľmi citlivé prvky ako napríklad kryptografické kľúče a podobne. Kvôli tomu je jasné, že implementácia inicializačného súboru tak, aby bola bezpečná, je obzvlášť náročnou úlohou.

4.2 Implementácia generátora v iOS

Generátor pseudonáhodných čísel Fortuna sme implementovali na zariadení iPhone 5 s najnovším operačným systémom iOS 6 a jeho funkcionalitu sme odskúšali aj na starších zariadeniach, kde pracoval tiež bez najmenších problémov. To je samozrejme spôsobené konzistencionalitou operačného systému, čiže aplikácia naimplementovaná v iOS 6 bude fungovať korektne na všetkých zariadeniach s týmto

operačným systémom, takže nemusíme nejako špeciálne upravovať kód, aby bol kompatibilný s čo najviac zariadeniami. Toto je veľká výhoda oproti iným operačným systémom ako napríklad Android alebo Symbian.

Návrh generátora Fortuna sa zakladá na tom, že jeho chod je kontinuálny, čiže beží v pozadí celého systému aj keď momentálne nie je aplikácia aktívna a pomedzi to zbiera údaje zo zdrojov entropie a zároveň obsluhuje požiadavky na generovanie dát pre aplikácie tretích strán. To ale v operačnom systéme iOS vôbec nie je jednoduchá úloha, keďže prísne pravidlá, ktorými podliehajú všetky aplikácie určené na distribúciu, nedovoľujú spolupracovať aplikáciám súbežne, takže je nemožné, aby nejaká aplikácia dala požiadavku generátoru a on by v pozadí vytvoril pseudonáhodné údaje, ktoré by jej predal. Taktiež nastáva problém s nepretržitým behom generátora v pozadí, lebo systém je spravený tak, že po desiatich minútach činnosti v pozadí aplikáciu vynútene ukončí.

Avšak existuje možnosť ako tieto dva závažné problémy vyriešiť, síce iba čiastočne, ale na správny chod aplikácie je to dostačujúce.

Na prvý problém týkajúci sa komunikácie medzi dvoma aplikáciami a ich aktívnym behom súčasne existuje čiastočné riešenie pomocou implementácie vlastných URL schém, čo je vlastne jediný spôsob ako môžu aplikácie tretích strán medzi sebou vzájomne komunikovať. Ak nejaká aplikácia odošle systému URL schému s identifikátorom inej aplikácie, v tomto prípade generátoru Fortuna, systém prepne generátor do aktívneho stavu, čiže otvorí aplikáciu, kde užívateľ vidí jej grafické rozhranie a zároveň pošle do pozadia aplikáciu, ktorá túto schému odosielala.

URL schéma odosielaná generátoru Fortuna je tvorená nasledujúcim textovým reťazcom:

@"fortuna://urlSenderApp#numberOfBlocks"

Začiatok tohto reťazca, konkrétne *fortuna://* je identifikátor slúžiaci systému, aby vedel ktorú aplikáciu má otvoriť. Ďalšou časťou je *urlSenderApp*, čo vlastne označuje URL schému aplikácie, ktorá zaslala požiadavku na generovanie dát generátoru. Bez tejto komponenty by generovanie dát nemalo zmysel, lebo aplikácia by nevedela, komu má poslať naspäť vytvorenú sekvenciu údajov, keďže URL schéma normálne neobsahuje informáciu o odosielateľovi. Posledná časť označuje množstvo blokov, ktoré sa majú vygenerovať.

Spracovanie URL schém prebieha v delegátovi aplikácie, ktorý má na starosti prvotné spustenie a mnoho iných prvkov týkajúcich sa jej chodu. Aby mohla aplikácia narábať s touto schémou, je potrebné v delegátovi naimplementovať metódu *application:handleOpenURL:*.

Tento problém má čiastočné riešenie práve preto, že neexistuje spôsob ako predať požiadavku inej aplikácii bez toho, aby neprešla do aktívneho stavu, čo ale

nijako nenarušuje funkčnosť generátora, no prepínanie medzi aplikáciami môže pôsobiť pre užívateľa rušivo.

Druhý a oveľa dôležitejší problém nastáva, ak aplikácia beží v pozadí viac ako desať minút. Vtedy, ako sme už spomínali, systém násilne ukončí jej chod. Toto sa dá obísť tým, že nastavíme niektorý z background módov, čím explicitne prikážeme systému, aby ju neukončoval. Aplikácie podporujú nasledujúce background módy: location (slúži na prijímanie GPS signálu), audio (prehráva audio stream), VoIP (umožňuje komunikáciu po internete) a Bluetooth mód, slúžiaci na komunikáciu medzi zariadeniami.

Aplikácia, ktorá má nastavený niektorý z vyššie spomenutých background módov a zároveň je určená k distribúcii, musí aj tieto služby efektívne využívať. Tým pádom sa nemôže stať, že by sme povolili backround mód VoIP v generátore, keďže jeho primárny cieľ je iný. Takáto aplikácia by neprešla schvaľovacím procesom spoločnosti Apple.

Audio mód nemôžeme použiť z dôvodu, že slúži iba na prehrávanie hudby, takže nahrávanie streamu z mikrofónu, čo by sa hodilo na získavanie vzoriek entropie, by nebolo možné. Taktiež nemôžeme použiť ani VoIP a Bluetooth, lebo pre účel aplikácie sú tieto služby nepodstatné.

Posledným a zároveň jediným vhodným módom pre generátor Fortuna je mód location. Ten budeme v aplikácii využívať ako ďalší zdroj na získavanie náhodných dát. Prijímanie GPS signálu je ale energeticky veľmi náročné, no je to jediný spôsob, ako dosiahnuť nepretržitý beh aplikácie v pozadí. Avšak pri implementácii je možné znížiť spotrebu energie na čo najmenšiu hodnotu nastavením rôznych príznakov, o ktorých budeme písať neskôr.

Okrem samostatnej aplikácie generátora Fortuna sme pre účely vývoja a testovania vytvorili ďalšiu pomocnú klientskú aplikáciu, ktorá odosiela požiadavky na generovanie zadaného množstva pseudonáhodných dát a následne ich spracováva. Tú si ale nebudeme bližšie popisovať, lebo pre jednoduchosť implementácie je pochopenie jej funkcionality možné z priloženého zdrojového kódu.

Hlavná aplikácia, ktorá obsahuje všetky potrebné prvky pre generovanie dát, by sa mala spúšťať automaticky hneď pri štarte mobilného zariadenia a pracovať v pozadí po celý čas chodu prístroja z dôvodu zaistenia vyššej bezpečnosti, čo je však v prostredí iOS nemožné, lebo aplikácie nemajú právo nastavenia príznaku, aby ich systém spustil hneď po zapnutí. Preto pre korektné fungovanie je nútený užívateľ prvý krát po štarte zariadenia zapnúť aplikáciu manuálne, no potom je už nezávislá a pracuje až do vypnutia prístroja, alebo ak ju užívateľ explicitne neukončí.

Teraz si bližšie rozoberieme jednotlivé triedy naimplementovaného generátora Fortuna.

Trieda *AppDelegate* má na starosti všetky najdôležitejšie časti aplikácie ako napríklad staranie sa o zbieranie dát zo zdrojov entropie do zásobníkov, ale aj spracovávanie klientskych požiadaviek na generovanie pseudonáhodných dát.

Pri štarte aplikácie sa v delegátovi zavolá metóda *application:didFinishLaunching WithOptions*, kde sa inicializuje trieda *FortunaPRNG* reprezentujúca samotný generátor, jednotlivé zdroje entropie a objekt *locationManager*, ktorý mimo to, že obsluhuje zdroj náhodnosti prijímajúci GPS signál, zároveň umožňuje pracovanie aplikácie v pozadí. Popri tom pravidelne každých 5 minút aktualizuje inicializačný súbor tak, že sa jeho obsah obnoví štyrmi blokmi vygenerovaných pseudonáhodných dát.

Ďalšia trieda má názov *FortunaPRNG* a ako už bolo spomenuté vyššie, predstavuje samotný Fortuna PRNG implementujúci všetky tri komponenty jeho architektúry a to generátor vyskytujúci sa v triede *FGenerator*, akumulátor skladajúci sa zo zásobníkov triedy *FPool* a inicializačný súbor, ktorý je reprezentovaný pomocou triedy *NSFileHandle*. Tá ma na starosti celkovú manipuláciu, čiže čítanie a zapisovanie do neho.

Implementácia triedy *FGenerator* je spravená presne podľa návrhu rozobratého v kapitole 4.1.1. V nej sa nachádza 128bitové počítadlo, ktoré slúži na generovanie dát a 256bitový AES kľúč.

Funkcia *InitializeGenerator* popísaná v návrhu nie je ako jediná naimplementovaná v rámci samostatnej metódy, ale inicializácia prebieha priamo pri samotnom vytváraní objektu.

Ďalšia funkcia *Reseed:* aktualizuje AES kľúč pomocou hašovacej funkcie SHA256, kde sa hašuje reťazec vytvorený spojením starého kľúča s argumentom metódy. Následne sa inkrementuje počítadlo, čo je nevyhnutné urobiť pri prvom volaní tejto metódy a pri nasledujúcich volaniach to už nie je potrebné, no z dôvodu zjednodušenia implementácie inkrementácia počítadla prebieha stále.

Tretia metóda sa nazýva *PseudoRandomData:*, ktorá má na starosti generovanie *n* bajtov pseudonáhodných dát. Tá na začiatku skontroluje, či klientom požadovaná veľkosť dát nie je väčšia ako 1MB. Ak áno, po vygenerovaní tieto dáta oreže. Následne sa aktualizuje AES kľúč prostredníctvom ďalších dvoch blokov pseudonáhodných sekvencií vytvorených navyše. Dáta sa nevytvárajú priamo v tele metódy, ale pomocou privátnej metódy *GenerateBlocks* generujúcej *n*/16 blokov dát, kde každý blok má pevnú veľkosť 128 bitov a vzniká zašifrovaním počítadla, ktoré sa hneď po zašifrovaní inkrementuje. Ešte na začiatku je potrebné skontrolovať, či sa počítadlo nenachádza v počiatočnom (nenulovom) stave. Slúži na to pomocná metóda *IsInitialized*, keďže sa nejedná o číselný typ premennej, ale má tvar reprezentujúci 128bitovú binárnu sekvenciu. Taktiež bola implementovaná ďalšia pomocná metóda na inkrementovanie počítadla *Inc*.

Akumulátor sa nachádza v triede *FortunaPRNG* ako pole objektov triedy *FPool*. Celkový počet objektov v tomto poli je presne 32, kde každý z nich predstavuje zásobník entropie, do ktorého ukladajú dáta jednotlivé zdroje náhodnosti. Návrh autorov ráta s neobmedzenou kapacitou zásobníkov, no v našej implementácii sú vytvorené prostredníctvom hašovacích kontextov z praktických dôvodov. Tým

pádom aj proces odhadu entropie musí byť iný, keďže veľkosť zásobníkov je fixná na 256 bitov a pôvodný návrh sa opieral o odhad množstva entropie uloženej v zásobníku. V triede *FPool* sa preto nachádza atribút číselného typu *iEntropy*, ktorý reprezentuje odhad entropie tak, že ukladá jej teoretickú veľkosť prichádzajúcu zo zdrojov. Keďže Fortuna PRNG nepotrebuje pre korektné fungovanie až tak presne odhadovať veľkosť entropie, za odhad je v našom prípade považovaná dĺžka prichádzajúcich dát zo zdrojov náhodnosti.

Verejná metóda *addRandomEvent:entropy:* triedy *FPool* slúži na pridávanie dát do zásobníkov. Tie nevolajú zdroje náhodnosti priamo, ale prostredníctvom metódy s rovnakým názvom v triede *FortunaPRNG*, ktorá sa stará o správnu distribúciu jednotlivým zdrojom. Je to naimplementované z bezpečnostných dôvodov, aby zdroje náhodnosti neboli schopné ovplyvňovať ukladanie dát do zásobníkov.

Na získanie dát zo zásobníka sa používa metóda *data*, ktorá po tom ako poskytne svoj obsah naplnený vzorkami entropie, ho vyprázdni.

Inicializačný súbor *seedFile.rnd* je uložený v privátnej zložke aplikácie, takže ju dokáže čítať iba samotná aplikácia a okrem toho, pri jej prvotnom spustení, kedy ešte súbor neexistuje a je vytváraný, nastaví sa príznak *NSFileProtectionComplete UnlessOpen*, čím sa zabezpečí najvyššia možná ochrana. Súbor je v pamäti zašifrovaný a pokiaľ je prístroj uzamknutý, nemá povolenie na zápis ani na čítanie z neho. Samozrejme okrem prípadu, že s ním momentálne pracuje aplikácia. Vtedy zostáva pre ňu prístupný.

Generátor je inicializovaný iba vtedy, keď súbor obsahuje minimálne 64 bajtov. Ak je súbor dostatočne dlhý, aktualizuje sa pomocou nanovo vygenerovaných pseudonáhodných dát o veľkosti presne 64 bajtov. Pokiaľ ale neobsahuje dostatočné množstvo dát, tie sa z bezpečnostného hľadiska nepoužijú, tým pádom sa ani generátor môže neinicializovať. To isté platí aj keď súbor ešte neexistuje. Po tom ako sa vytvorí, zostáva nezaplnený a čaká na doplnenie dát zo zásobníkov entropie, pričom ignoruje všetky požiadavky klientov. Inicializačný súbor sa pravidelne aktualizuje každých 5 minút pomocou objektu typu *NSTimer* nachádzajúceho sa v triede *AppDelegate*.

Pravidelná reinicializácia generátora prispieva k zaisteniu bezpečnosti a tým pádom vytvára kryptograficky bezpečný generátor. Táto reinicializácia sa vykonáva pred začiatkom tvorby pseudonáhodných dát, ak je v zásobníku P_0 minimálne 256 bitov entropie a ak čas od poslednej reinicializácie je väčší ako 100ms. Reinicializácia prebieha podľa toho, že ak 2^i delí počítadlo reinicializacií *ReseedCnt*, použije sa zásobník P_i, čo je presne rovnaký princíp, aký sme už spomínali skôr v práci.

Nami naprogramovaný generátor pseudonáhodných čísel Fortuna využíva štyri zdroje náhodnosti, z toho tri boli použité v implementácii predchádzajúcej aplikácie reprezentujúcej generátor ANSI X9.31. Dáta sú zo zdrojov náhodnosti do zásobníkov distribuované podľa jednoznačného identifikačného čísla každého zdroja.

Zdroj náhodnosti s najnižším identifikačným číslom zbiera údaje z akcelerometra, ktoré predstavujú konkrétny postoj zariadenia v čase získavania vzoriek. Postoj zariadenia je definovaný pomocou eulerových uhlov pitch, roll a yaw, ktoré určujú presnú orientáciu v priestore vzhľadom na súradnicový systém. Ich hodnoty sú v radiánoch a vďaka pomerne veľkej odchýlke pri meraní je tento zdroj entropie veľmi vhodný pre účely generovania. Nami naimplementovaný zdroj náhodnosti generuje 192 bitov na jeden zber hodnôt, čo predstavuje 9600 bitov dát za sekundu. Avšak toto číslo by sa dalo pomerne ľahko zväčšiť, keby popri eulerových uhloch zbierame aj iné hodnoty, ktoré obsahuje akcelerometer, ako napríklad momentálne zrýchlenie a podobne. Treba ale upozorniť, že získavanie dát z akcelerometra je pomerne náročné na spotrebu energie, preto sme aj vďaka tomu nastavili časový interval zberu na jednu minútu.

Druhý zdroj náhodnosti zbiera údaje z mikrofónu. Najprv sa musí objekt slúžiaci na prístup k mikrofónu inicializovať. V procese inicializácie sa nastavia potrebné parametre ako napríklad vzorkovacia frekvencia, ktorú sme nastavili na 44100 Hz, počet kanálov na jeden a kvalitu vzoriek na najnižšiu, aby veľkosť entropie bola čo najväčšia. Dáta z mikrofónu sa zbierajú po dobu jednej sekundy, no nenahráva sa priamo zvukový stream, ale kontroluje sa sila nahrávaného zvuku každých 5ms. Po porovnaní údajov z nášho naimplementovaného riešenia a klasického nahrávaného zvukového streamu nám vyšlo, že práve naše riešenie poskytuje oveľa kvalitnejšie vzorky entropie, ale za cenu menšieho množstva. Celkovo, za jednu sekundu zberu získame 1280 bitov dát, čo je dostatočné množstvo pre účely aplikácie, keďže zásobníky majú fixnú veľkosť.

Tretí zdroj náhodnosti môžeme považovať len za pomocný, lebo poskytuje najmenej entropie. Jedná sa o systémovú štatistiku pamäte RAM, ktorá poskytuje pre naše účely 64 bitov dát. Taktiež kvalita entropie nie je až taká veľká ako v predchádzajúcich dvoch zdrojov, pokiaľ sa zariadenie nepoužíva. Ak je zariadenie stále využívané v priebehu časového intervalu väčšieho ako jedna minúta, kvalita entropie sa značne zvýši.

Posledný zdroj využívaný nami naimplementovanou aplikáciou je pre potreby generátora najdôležitejší. Aby sme odstránili nedostatok týkajúci sa násilného ukončenia aplikácie systémom, bolo nevyhnutné tento zdroj naimplementovať. No je to zároveň zdroj s najväčšou spotrebou energie, preto sme museli znížiť záťaž nastavením distanceFilter na 100 metrov a distanceAccuracy na hodnotu kCLLocationAccuracyThreeKilometers, ktorá označuje presnosť prijímaného signálu na tri kilometre. Údaje z tohto zdroja prinášajú 96 bitov dát.

Tieto štyri zdroje náhodnosti sme vybrali z toho dôvodu, že prinášajú najviac entropie v porovnaní so spotrebou energie. Mimo nich existujú desiatky ďalších zdrojov, no pri bližšej analýze buď neposkytujú dostatočné množstvo dát, alebo sú pomerne náročné na výkon zariadenia.

4.3 Zhrnutie

V tejto časti práce sme sa zaoberali architektúrou a implementáciou generátora pseudonáhodných čísel Fortuna. Pri implementácii na operačný systém iOS sme postupovali podľa doporučení autorov Schneiera a Fergusona nachádzajúcich sa v publikácii Practical Cryptography. Hlavný faktor ovplyvňujúci nielen výkon, ale aj bezpečnosť nami vytvoreného generátora je použitie štyroch zdrojov náhodnosti akcelerometra, systémových štatistík, mikrofónu a modulu na prijímanie GPS signálu. Pri generovaní dát sa aplikácia kvôli systémovým obmedzeniam prepne do popredia, čo je v rozpore s návrhom autorov, no pre celkovú funkčnosť to nemá žiadny vplyv. Keďže primárne určenie mobilného zariadenia nie je oblasť kryptografie, bolo potrebné zaistiť, aby používanie generátora malo čo najmenší zásah na činnosť ostatných aplikácii.

Zdrojové kódy generátora Fortuna a jeho pomocnej aplikácie sú dostupné na priloženom CD.

Záver

Generovanie pseudonáhodných sekvencií na mobilné zariadenia s operačným systémom iOS je pomerne nová problematika, keďže pri skúmaní dostupných aplikácii zaoberajúcich sa podobným problémom sme nenašli žiadnu, ktorá by bola určená priamo pre komerčné využitie. No pri tom ako sa stávajú mobilné zariadenia viac osobnejšie a ukladajú citlivé údaje, je potrebné klásť čoraz väčší dôraz na bezpečnosť, preto sme sa zaoberali v tejto bakalárskej práci práve generátorom pseudonáhodných sekvencií, ktoré sú neodkladnou súčasťou kryptografických protokolov a rozobrali dopodrobna túto tematiku. Taktiež z dôvodu neustáleho zvyšovania výkonu mobilných zariadení máme väčšie možnosti na implementáciu takýchto generátorov, ktoré sme zužitkovali v praktickej časti tejto práce.

Začiatok práce sa venuje teoretickej časti problematiky týkajúcej sa generovania pseudonáhodných sekvencií. Aby čitatelia presnejšie pochopili ďalšie kapitoly práce, zaviedli sme všetky dôležité pojmy vyskytujúce sa neskôr v texte. Taktiež sme podrobne popísali operačný systém iOS, jeho základné princípy, históriu, architektúru a v neposlednom rade jeho bezpečnostné prvky, ktoré zohrávajú dôležitú úlohu pre pochopenie naimplementovaných aplikácií v praktickej časti tejto bakalárskej práce.

Práve praktická časť obsahujúca dve kapitoly, tvorí hlavný prínos celej práce. Každá z nich sa venuje rozdielnemu typu generátora pseudonáhodných čísel. Prvý typ predstavuje generátor ANSI X9.31, ktorý ale nie je kryptograficky bezpečný a slúži iba na bližšie sa zoznámenie s programovaním na operačný systém iOS a preskúmaním možností používania zdrojov náhodnosti v mobilných zariadeniach. Z dôvodu jednoduchosti architektúry generátora sa tieto zdroje využívajú jednorázovo iba pri prvom, počiatočnom generovaní pseudonáhodných sekvencií. Naimplemetnované zdroje náhodnosti sú v prvom rade údaje z mikrofónu prístroja, ďalej dáta z akcelerometra a ako posledné systémové údaje týkajúce sa správy pamäti v zariadení.

V druhej kapitole praktickej časti sa venujeme podrobnejšie kryptograficky bezpečnému generátoru pseudonáhodných čísel Fortuna vytvoreného známymi kryptológmi Schnneierom a Fergusonom. Všetky informácie sme využívali z ich publikácie Practical Cryptography, kde popisujú jeho architektúru a doporučenia na implementáciu. Práve pri tvorbe aplikácie z uvádzaných doporučení nastali dva vážne problémy, ktoré sa nám čiastočne podarilo vyriešiť, keďže sme boli limitovaný bezpečnostnými prvkami systému iOS. No na korektné fungovanie aplikácie takéto čiastočné riešenie nemalo negatívny vplyv, takže implementácia generátora bola celkovo úspešne zvládnutá.

Keďže mobilné zariadenia nemajú primárnu funkciu určenú pre kryptografické účely, ale jedná sa iba o doplnkovú službu, je potrebné, aby nijakým radikálnym spôsobom neobmedzovali ich normálny chod. Ako príklad môžeme uviesť jeden zdroj náhodnosti, ktorý sme ale v našej implementácii nepoužili a to údaje z fotoaparátu. Pri pravidelnom zbieraní vzoriek entropie by mohlo nastať to, že by užívateľ nemohol korektne používať fotoaparát, lebo prístup k nemu je exkluzívny. Z tohto dôvodu sme sa rozhodli ho v implemetnácii nepoužiť.

Pravidelný interval zbierania dát zo zdrojov náhodnosti sme nastavili na jednu minútu, čo sa javí ako dobrý kompromis medzi spotrebou energie a dodávaným množstvom entropie. Je to dôležité z toho hľadiska, že kapacita energie v mobilných zariadeniach je značne obmedzená a naviac, udržanie behu aplikácie v pozadí má na výdrž batérie veľký vplyv, lebo sa používa jediný dostupný spôsob, ktorý implicitne zakáže systému ju násilne ukončiť a to prijímanie GPS signálu.

Výsledkom práce sú dve naimplmentované aplikácie reprezentujúce generátory pseudonáhodných čísel ANSI X9.31 a Fortuna, ktoré úspešne dosahujú požadovanú funkcionalitu. Okrem toho sa na priloženom CD nachádza aj aplikácia slúžiaca ako príklad komunikácie s generátorom Fortuna. Dúfame, že táto bakalárska práca ponúkne dostatočné informácie o bezpečnosti v operačnom systéme iOS a prispeje k zvýšeniu ochrany aplikáciám tretích strán.

Literatúra

[1] Ferguson, Niels a Schneier, Bruce: *Practical Cryptography*, chapter 10.3 Fortuna. John Wiley & Sons, 2003. ISBN 978-0471223573.

[2] Kelsey, John a Schneier, Bruce a Ferguson, Niels: *Yarrow-160: Notes on the Design and Analysis of the Yarrow Cryptographic Pseudorandom Number Generator*. In Sixth Annual Workshop on Selected Areas in Cryptography. Springer Verlag, 1999. [online]. [cit. 20. 4. 2013]. Dostupné z: <http://www.schneier.com/paper-yarrow.pdf >.

[3] American National Standards Institute. *American National Standard X9.31-1992: Public Key Cryptography Using Reversible Algorithms for the Financial Services Industry*, 1993.

[4] Keller, Sharon S.: *NIST-Recommended Random Number Generator Based on ANSI X9.31 Appendix A.2.4 Using the 3-Key Triple DES and AES Algorithms*. National Institute of Standards and Technology, Information Technology Laboratory, Computer Security Division, 2005. [online]. [cit. 20. 4. 2013]. Dostupné z: <http://csrc.nist.gov/groups/STM/cavp/documents/rng/931rngext.pdf >

[5] *iOS Developer Library* [online]. [cit. 20. 4. 2013]. Dostupné z: <http://developer.apple.com/library/ios/navigation/index.html>

[6] *iOS Security* [online]. [cit. 20. 4. 2013]. Dostupné z: <http://www.apple.com/ipad/business/docs/iOS_Security_Oct12.pdf>

[7] Miller, Charlie a Blazakis, Dion a DaiZovi, Dino a Esser, Stefan a Iozzo, Vincenzo a Weinmann, Ralf-Philip: *iOS Hacker's Handbook,* John Wiley & Sons, 2012. ISBN 978-1118204122.

[8] Kelsey, John a Schneier, Bruce a Wagner, David a Hall, Chris: *Cryptanalytic Attacks on Pseudorandom Number Generators* [online]. [cit. 20. 4. 2013]. Dostupné z: <http://www.schneier.com/paper-prngs.pdf>

[9] Chaitin, Gregory J.: *Randomness and Mathematical Proof*. Scientific American, May 1975, vol. 232, no. 5, str. 47 – 52 [online]. [cit. 20. 4. 2013]. Dostupné z: <http://www.cs.auckland.ac.nz/~chaitin/sciamer.html>

[10] Menezes, Alfred J. a C. van Oorschot, Paul a Vanstone, Scott A.: *Handbook of Applied Cryptography*, CRC Press, 1996. ISBN 0-8493-8523-7.

[11] Yao, Andrew C.: *Theory and application of trapdoor functions*. Symposium on Foundations of Computer Science, 1982, str. 80–91 [online]. [cit. 20. 4. 2013]. Dostupné z: <http://www.di.ens.fr/users/phan/secuproofs/yao82.pdf>

[12] Federal Information Processing Standards Special Publication 800-22 Rev. 1a: *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Aplications*. [online]. [cit. 20. 4. 2013]. Dostupné z: <http://csrc.nist.gov/publications/nistpubs/800-22-rev1a/SP800-22rev1a.pdf>

[13] NIST Special Publication 800-90A: *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*. [online]. [cit. 20. 4. 2013]. Dostupné z: <http://csrc.nist.gov/publications/nistpubs/800-90A/SP800-90A.pdf>

Dodatok A

Obsah priloženého CD

Priložené CD obsahuje:

- Zdrojový kód aplikácie ANSI X9.31.
- Zdrojový kód aplikácie Fortuna.
- Zdrojový kód ukážkovej klientskej aplikácie.
- Bakalársku prácu.