VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

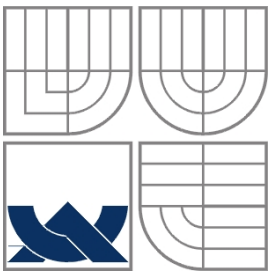
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

HARDWAROVÁ AKCELERÁCIA ÚTOKU NA ŠIFROVANIE

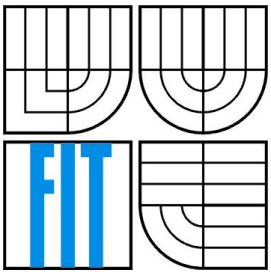
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE ADAM OKULIAR
AUTHOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

HARDWAROVÁ AKCELERÁCIA ÚTOKU NA ŠIFROVANIE

HARDWARE ACCELERATION OF CIPHER ATTACK

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ADAM OKULIAR

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ZDENĚK VAŠÍČEK

BRNO 2009

Abstrakt

Hardwarová akcelerácia výpočtu býva často vhodným nástrojom ako docieliť výrazne lepšieho výkonu pri spracovávaní veľkého množstva dát alebo pri realizácii algoritmu ktorý je možné dobre paralelizovať. Cieľom práce je demonštrovať výsledky použitia FPGA obvodov na implementáciu algoritmu s exponenciálnou zložitosťou. Zvoleným algoritmom je útok hrubou silou na šifrovací algoritmus WEP so 40 bitovým klúčom. Účelom práce je porovnať vlastnosti a výkon softwarovej a hardwarovej implementácie algoritmu.

Abstract

Hardware acceleration is often good tool to achieve significantly better performance of processing great ammount of data or of realization of parallel algoritms. Aim of this work is to demonstrate resoluts of using FPGA circuits for implementation exponentially complex algorithm. As example has chosen brute-force attack on WEP cryptographic algorithm with 40-bit long key. Goal of this work is to compare properties and performance of software and hardware implementation of choosen algorithm.

Klíčová slova

kryptografia, hardware, hradlové polia, FPGA, akcelerácia, implementácia, šifra, RC4, útok hrubou silou, hashovanie, sieťový rámec, algoritmus, konečný automat, optimalizácia, priepustnosť, analýza, výkon

Keywords

cryptography, hardware, gate arrays, FPGA, acceleration, implementation, cipher, RC4, brute-force attack, hashing, network frame, algorithm, finite-state machine, optimalization, throughput, analysis, performance

Citace

Adam Okuliar: Hardwarová akcelerácia útoku na šifrovanie, bakalářská práce, Brno, FIT VUT v Brně, 2009

Hardwarová akcelerácia útoku na šifrovanie

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne, pod vedením Ing. Zdeňka Vašíčka.
Uviedol som všetky literárne pramene a publikácie z ktorých som čerpal.

.....
Adam OKULIAR
18.5.2009

Poďakovanie

Týmto by som chcel poďakovať pánovi Ing. Zdeňkovi Vašíčkovi, vedúcemu mojej práce za jeho cenné rady, ochotu pri riešení zdanlivo neriešiteľného a nemalé množstvo času ktoré mi venoval počas tvorby tejto práce.

Obsah

1 Úvod

Ľudia sa od nepamäti snažili ochrániť citlivé informácie a tajomstvá pred zneužitím. Jednou z najúčinnejších metód ochrany citlivej správu alebo dokumentu je jeho zašifrovanie. Pod pojmom šifrovanie rozumieme prevod správy do formy ktorá je čitateľná len adresátovi, ktorý má určitú špeciálnu znalosť. Špeciálnou znalosťou môže byť napríklad znalosť transpozičnej schémy, prípadne substitučného pravidla. Táto špeciálna znalosť sa veľmi často nazýva dešifrovacím kľúčom.

S rastúcou potrebou civilizácie utajovať citlivé informácie rastie i snaha získať čitateľný obsah depeše bez vlastnosti dešifrovacieho kľúča, teda rozlúštiť šifru. Tieto snahy umožnili vzniknúť novému vednému odboru zvanému *kryptografia.* Kryptografia v sebe zahŕňa dve základné časti: *Kryptológiu*, vedu o tvorbe a vlastnosti šifier a *kryptoanalýzu*, vedu o lúštení zašifrovaných správ. Veľmi často sa všetky časti vedy ktoré priamo súvisia so šifrovaním nazývajú súhrnne *kryptológia*.

Vznik a použitie číslicových počítačov dal do rúk kryptológii veľmi efektívny nástroj. Umožnila úspešne analyzovať mnohé šifry ktoré boli dlho nerozluštené, ale aj vytvoriť nové efektívnejšie šifrovacie metódy. Prvý krát bol použitý výpočetný prostriedok (ešte nie celkom počítač) v roku 1942 na prelomenie ponorkového kódu používaného wehrmachtom. Použité zriadenie sa volalo The Bombe a simulovalo niekoľko desiatok šifrovacích zariadení testujúcich rôzne kľúče. Z pohľadu modernej kryptografie, sa tento typ útoku nazýva útokom hrubou silou, kedy sa postupne skúšajú všetky možné kľúče.

Súčasná kryptológia pozná niekoľko metód jako získať obsah zašifrovanej správy. Do prvej skupiny metód patria metódy matematické. Tieto sa snažia pomocou vhodného vzorku zašifrovaných a rozšifrovaných dát určiť vlastnosti a slabé miesta šifry a prípadne zistiť šifrovací kľúč

Medzi matematické metódy patrí lineárna a diferenciálna kryptoanalýza a mnohé ďalšie Tieto metódy sú extrémne náročné na znalosti z oblasti diskrétnej matematiky, a vyžadujú rozsiahly výskum. Nie vždy vedú k úspechu. Ak sa však podarí nájsť slabé miesto v šifrovacom algoritme dokáže táto metóda odhaliť šifrovací kľúč s relatívne malým množstvom výpočetných prostriedkov.

Druhou skupinou krytoanalytických útokov sú útoky hrubou silou. Tieto si zvyčajne nevyžadujú veľmi hlboké matematické znalosti, avšak vyžadujú obrovské množstvo výpočetného výkonu ktorý využívajú po dlhú dobu. Princíp týchto útokov je v spustení dešifrovacieho algoritmu pre všetky možné kľúče. Útočník v tomto prípade nerobí nič iné než právoplatný adresát správy, takže pravdepodobnosť nájdenia správneho kľúča sa blíži istote. V tomto prípade je však čas kľúčovým faktorom. Veľkosť stavového priestoru rastie exponenciálne s dĺžkou kľúča, a čas potrebný pre nájdenie kľúča môže dosiahnuť desiatky až stovky rokov.

Tento text ďalej pojednáva o princípoch a metódach aplikovateľných pri hardwarovej a softwarovej implementácii kryptografických algoritmov a snahách o čo najvyšší stupeň optimalizácie. Ako ukážkový algoritmus v mojej práci poslúži šifra RC4 používaná na zabezpečenie prístupu do bezdrôtovej siete. Nekladieme si za cieľ vytvoriť komplexný nástroj na prelomenie tejto šifry, skôr porovnať vlastnosti a výkon softwarovej a hardwarovej implementácie. Touto prácou chcem v prvom rade demonštrovať možnosti moderného programovateľného hardwaru v náročných výpočtových úlohách. V texte práce diskutujem vlastnosti zvolenej šifry, jej použitie. Ďalej budú popísané vlastnosti a výkon referenčnej softwarovej implementácie a implementácie v obvode FPGA. Výsledkom práce by malo byť zhodnotenie výkonu a celkových nákladov jednotlivých implementácii. Historické fakty v tomto texte boli čerpané z [1] a [2].

2 Rozbor problému

2.1 Vlastnosti a klasifikácia moderných šifier

Z kryptografického hľadiska možno za šifru považovať usporiadanú množinu krokov, vedúcu k zašifrovaniu a rozšifrovaniu pôvodnej správy V ďalšom texte sa budeme teda na šifru pozerať ako na algoritmus, ktorý má určité vstupy a výstupy a vyznačuje sa časovou a pamäťovou zložitosťou.

Vstupom šifrovacieho algoritmu je nešifrovaný, čistý text a šifrovací kľúč. Výstupom tohto algoritmu je šifrovaná správa. V optimálnom prípade je možné previesť šifrovanú správu naspäť na čistý text pomocou dešifrovacieho algoritmu len za znalosti dešifrovacieho kľúča

Moderné šifry je možné deliť na základe viacerých kritérií, avšak najčastejšie delenie býva:

Delenie na základe symetrickosti:

- Symetrické šifry, kde na šifrovanie a zašifrovanie používa ten istý kľúč Šifrovací kľúč je zhodný s dešifrovacím kľúčom Z uvedeného vyplýva, že pred zahájením komunikácie si musia obe strany vymeniť bezpečným spôsobom dohodnutý kľúč
- Asymetrické šifry, kde sa na zašifrovanie používa iný kľúč ako na rozšifrovanie. Kľúč potrebný na zašifrovanie má prívlastok verejný, dešifrovácí kľúč sa nazýva privátnym.

Hlavnou výhodou asymetrickej kryptografie je skutočnosť že komunikujúce strany si nemusia vymeniť tajný kľúč Asymetrické kryptografické algoritmy sú okrem toho použiteľné aj na elektronicky podpis. Hlavnou nevýhodou asymetrickej kryptografie sú jej veľké nároky na výpočetné prostriedky.Vzhľadom na to, že zvolená šifra *WEP* je šifrou symetrickou, budeme sa v ďalšom texte venovať prevažne symetrickým šifrám.

Delenie na základe typu vstupných dát

- Blokové šifry, ktoré vyžadujú na svojim vstupe blok dát určitej minimálnej veľkosti. V prípade že je potrebné šifrovať menšie množstvo dát, je nevyhnutné ho doplniť do minimálnej veľkosti. Vzniká tak zarovnanie. Bloková šifra transformuje blok vstupných dát na iný blok výstupných dát. Na blokovú šifru sa teda môžeme pozerať ako na akúsi prekladovú tabuľku. V praxi je preklad implementovaný pomocou vhodného algoritmu. Typickým zástupcom tohto typu sú algoritmy Lucifer, DES, 3DES, AES
- Prúdové šifry, ktoré dokážu šifrovať kontinuálny tok dát. Princíp týchto šifier je v generovaní pseudonáhodného prúdu dát ktorý je vhodným spôsobom kombinovaný s prúdom šifrovaných dát. Generovaný prúd dát sa v angličtine nazýva keystream, vzhľadom na neexistujúci slovenský preklad budem používať pôvodný termín. Keystream sa s prúdom šifrovaných dát kombinuje pomocou operácie *XOR*. Týmto šifry sa často označujú aj za stavové, pretože musia uchovávať stav generátoru keystreamu. Typickými zástupcami sú A5, Helix, a RC4

Prúdové šifry sa v praxi ukazujú ako implementačne jednoduchšie a výkonnejšie ako blokové šifry. Prúdové šifry však trpia závažnými bezpečnostnými problémami ak niesu použité korektne [3] Základným pravidlom je, že šifra by nemala byť nikdy použitá viackrát s rovnakým štartovacím stavom. Toto sa snažia rôzne algoritmy dosiahnuť napríklad použitím inicializačných vektorov, ktoré sa použijú ako časť kľúča

Je dôležité poznamenať že z hľadiska počítačovej bezpečnosti dokážu šifrovacie funkcie zabezpečiť dôvernosť správy, ale nemajú prostriedky na zabezpečenie integrity. Inými slovami šifrovanie dokáže zaručiť, že šifrované dáta budú čitateľné len subjektom vlastniacim dešifrovací kľúč ale nedokážu overiť, či šifrovaná správa bola alebo nebola v priebehu prenosu modifikovaná.

2.2 Vlastnosti a činnosť hashovacích funkcii

Hashovacia funkcia je funkcia, ktorá prevádza zadanú postupnosť premennej dĺžky na jej signatúru alebo odtlačok (anglicky hash) pevnej dĺžky. Signatúra tak vytvára jednoznačnú charakteristiku vstupných dát, nakoľko je veľmi obtiažne nájsť inú postupnosť vstupných dát pre ktorú by hashovacia funkcia vrátila rovnaký hash. V kryptografii sa hashovacie funkcie využívajú hlavne na implementáciu digitálneho podpisu a autentizáciu správ.

Hlavné požiadavky na vhodnú hashovaciu funkciu sú:

- Akékoľvek množstvo vstupných dát poskytuje rovnako veľký hash.
- Malou zmenou vstupných dát dostaneme výraznú zmenu dát výstupných
- Vysoká pravdepodobnosť, že dve správy z rovnakým hashom sú rovnaké
- Vysoká obtiažnosť nájdenia takého vstupu ktorý zodpovedá požadovanému hashu

Na základe vysokej obtiažnosti nájdenia vhodného vstupu funkcie na základe známeho výstupu, hovoríme, že funkcia je jednosmerná. Ak hashovacia funkcia vygeneruje dva rovnaké hashe pre rôzne vstupy hovoríme o kolízii v hashovaní. Dôležitým parametrom hashovacých funkcii je pravdepodobnosť vzniku kolízie.

Existuje veľké množstvo hashovacích funkcii, avšak nie všetky majú z kryptografického hľadiska vhodné vlastnosti. Pre kryptografické účely sa dajú použiť funkcie *MD5*, *SHA1* a *SHA2*.

Kryptografické hashovacie funkcie sú často využívané pre autentizáciu správy posielanej nezabezpečeným kanálom. Princíp autentizácie spočíva vo spočítaní hashu s pôvodnej správy a vhodného kľúča Následne sa spočítaný hash sa pripojí k správe a pošle nezabezpečeným kanálom. Na prijímacej strane sa z prijatej správy a zdieľaného tajného kľúča opäť spočíta hash a porovná sa s tým ktorý bol pripojený v správe. Oba hashe súhlasia iba za predpokladu že oba konce kanálu zdieľajú rovnaký kľúč a správa nebola cestu modifikovaná. V prípade že spočítaný hash súhlasí s hashom prijatým možno správu považovať za dôveryhodnú. Uvedený mechanizmus sa používa pomerne často a v rôznych obmenách. Tento spôsob zabezpečenia sa nazýva *Hash message authentication code (HMAC).*

Opäť je nevyhnutné poznamenať, že autentizácia správy dokáže zaručiť integritu správy, ale nezaručuje jej dôvernosť. Inými slovami, je možné zaručiť, že správa pochádza od dôveryhodného odosielateľa a že nebola v priebehu prenosu modifikovaná. Avšak správa zabezpečená pomocou HMAC je stále posielaná ako čistý text a je čitateľná pre kohokoľvek kto má prístup k prenosovému kanálu.

2.3 Zabezpečenie bezdrôtových sietí, algoritmus WEP

Bezdrôtové siete majú, narozdiel od sietí vybudovaných nad metalickým alebo optickým vedením oveľa väčšiu flexibilitu a zvyčajne i nižšie náklady na inštaláciu. Tieto nesporne dobré vlastnosti sú však vyvážené mnohými negatívami ktoré sú prekážkou pri ich širšom nasadzovaní do komerčného sektoru. Často diskutovaným problémom bývajú otázky bezpečnosti, v zmysle zachovania dôvernosti a integrity prenášaných dát. V optických alebo metalických sieťach je tento problém riešený prirodzene, pomocou ochrany fyzického prístupu k prenosovému médiu siete Prípadný útočník by musel najprv získať fyzický prístup k sietí, čo je zvyčajne ľahko zistiteľné. Prístup k bezdrôtovému prenosovému médiu je pomerne ľahké získať, vzhľadom na fyzikálnu podstatu šírenia elektromagnetického žiarenia priestorom. Útočníkovi tak stačí pre prístup k sieti vhodná smerová anténa s dostatočným ziskom. Naviac pasívny prístup k prenosovému médiu je veľmi ťažko zistiteľný.

Snahy o vytvorenie dostatočnej úrovne zabezpečenia priamo na linkovej vrstve bezdrôtovej siete viedli k vývoju algoritmu WEP. Samotný názov WEP je skratkou z Wired Equivalent Protection. Algoritmus bol predstavený už v roku 1997 spolu s prvými bezdrôtovými sieťami 802.11, ktoré dosahovali rýchlosť 1 mbit/s. Za bezpečný bol považovaný až do roku 2001, kedy bolo pomocou krytoanalýzy objavených niekoľko závažných bezpečnostných slabín. V dnešnej dobe je tento algoritmus považovaný za zastaralý a nemal by sa používať.

Nástupcami algoritmu WEP sú algoritmy WPA a WPA2, ktoré riešia bezpečnostné problémy WEPu a dodnes sú považované za bezpečné. Tieto algoritmy však vyžadujú vyšší výpočetný výkon na strane prístupového bodu i klienta a podporu v hardware bezdrôtového adaptéru, takže pre prechod na zabezpečenie WEP je nevyhnutný hardwardový upgrade. Okrem toho algoritmy WPA vyžadujú centrálny bod pre správu dočasných kľúčov (Temporal Key Integrity Protection) a ich implementácia je veľmi náročná v decentralizovaných Ad-Hoc sieťach.

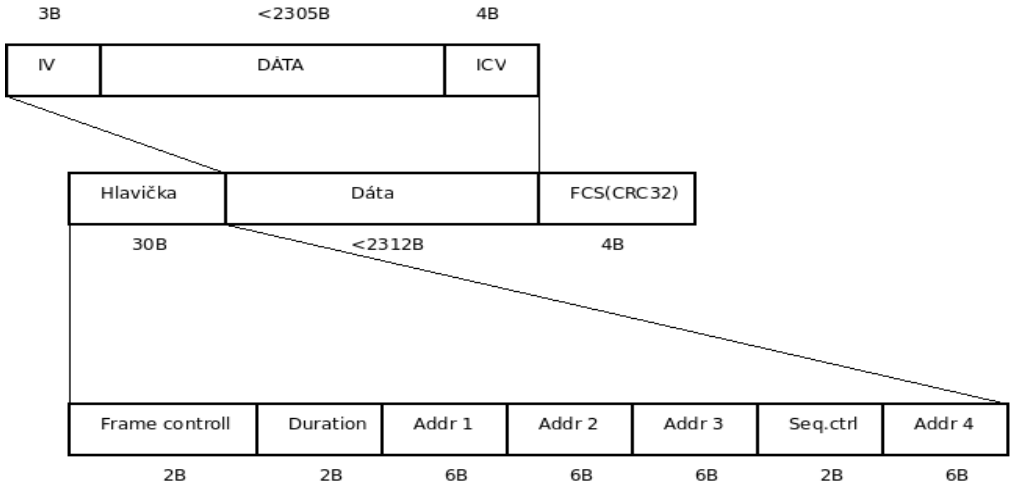
2.3.1 Implementácia algoritmu WEP

Na zabezpečenie dôvernosti je použitá prúdová šifra RC4. Táto používa 64 bitový kľúč, ktorý bol neskôr nahradený kľúčom 128 bitovým. Prvé tri bajty, teda 24bitov kľúča tvorí inicializačný vektor(ďalej len IV), ktorý sa náhodne generuje pre každý sieťový rámec, a je posielaný spolu v dátovej časti rámca v nešifrovanej podobe. Útočník tak môže zistiť prvé tri bajty kľúča len prostým odchytením dát zo siete. Tajnou časťou kľúča zostáva nasledujúcich 40 resp 104 bitov kľúča Tieto sa zadávajú na koncových sieťových zriadeniach vo forme ASCII znakov, alebo na niektorých zariadeniach ako skupina hexadecimálnych číslic.

Na zabezpečenie integrity správy používa algoritmus hodnotu ICV (Integrity Check Value) Táto hodnota sa počíta s prenášaných dát ako CRC32 kontrolný súčet, ktorý je následne zašifrovaný rovnakou šifrou ako dáta. Príjemca potom prijaté dáta rozšifruje za pomoci tajného kľúča a spočíta z nich CRC32. Odtlačok vrátený CRC funkciou porovná z dešifrovanou hodnotou ICV. Ak sa obe hodnoty zhodujú, je možné správu považovať za dôveryhodnú. Vzhľadom na skutočnosť že funkcia CRC32 spĺňa požiadavky na hashovaciu funkciu možno tento spôsob overovania pokladať za overovanie typu HMAC (popísane vyššie)

2.3.2 Formát datagramu zabezpečeného algoritmom WEP

Na obrázku 2.1 je znázornená štruktúra sieťového rámca linkovej vrstvy bezdrôtovej siete rodiny 802.11.



Obrázok 2.1: Štruktúra zašifrovaného sieťového rámca

V nasledujúcom texte stručne popíšeme význam jednotlivých položiek:

- Polia Frame controll a Duration obsahujú servisné informácie ktoré majú význam pre prístupové body a sieťové adaptéry pre správu vysielacieho výkonu, a identifikáciu typu rámca.
- Adresy Addr1 až Addr4 obsahujú MAC adresy zdrojovej a cieľovej stanice, adresu pristupového bodu, prípadne adresu WDS zariadenia.
- Položka Seq.ctrl sa používa v prípade že je povolená L2 fragmentácia. Fragmentácia datagramu na L2 vrstve rozdeľuje veľké datagramy na niekoľko menších, ktoré sú vysielané nárazovo. Takýto prístup zvyšuje réžiu, avšak v zarušených oblastiach prináša dobré výsledky, nakoľko prípadná strata datagramu spôsobí stratu menšieho množstva dát.
- Telo datagramu tvorí dátová časť Táto môže mať veľkosť až 2312 Bajtov, avšak v prípade použitia zabezpečenia WEP sú prvé tri bajty dátovej časti vyhradené pre inicializačný vektor a posledné štyri bajty pre autentizačnú ICV hodnotu.
- Celý rámec včetne hlavičky je chránený CRC32 kontrolným súčtom. Tento sa nachádza za dátovou časťou v poli FCS (Frame check sequence).Za zmienku stojí skutočnosť, že režijné informácie pre algoritmus WEP sú zahrnuté v dátovej časti rámca, takže zariadenia, ktoré nemajú implementovaný WEP protokol dokážu i naďalej pracovať s rámcom, nedokážu však interpretovať jeho obsah.

Dovolím si ešte poznamenať že účelom *ICV* hodnoty je autentizácia dátovej časti správy, na rozdiel od *FCS* hodnoty, ktorá slúži na detekciu chyby pri prenose. Tento kontrolný súčet chráni celý datagram, včetne hlavičky.

2.3.3 Činnosť šifry RC4

Šifrovací algoritmus pozostáva z inicializačnej časti a vlastného generovania keystreamu. Cieľom inicializačnej časti je na základe tajného kľúča inicializovať 256 bajtové stavové pole, ktoré určuje stav algoritmu. Po inicializačnej časti nasleduje vlastný beh algoritmu, kedy je v jednej iterácii vždy vygenerovaný jeden bajt keystreamu a vhodným spôsobom modifikované stavové pole. Práve neustále sa meniaci obsah stavového pola je kľúčovým prvkom bezpečnosti šifry, nakoľko potenciálny útočník by musel na úspešne rozlúštenie prenášaných dát replikovať celý obsah stavového poľa.[4]

Na nasledujúcom pseudokóde, ktorý bol prebraný z [4], bude ozrejmená funkcia inicializačnej fázy šifry:

```
1     for i from 0 to 255
2           S[i] := i
3     endfor
4     j := 0
5     for i from 0 to 255
6           j := (j + S[i] + key[i mod keylength]) mod 256
7           swap(&S[i],&S[j])
8     endfor
```

Algoritmus 2.1: Inicializácia stavového poľa šifry

Stavové pole je najprv lineárne inicializované od hodnotami od 0 do 255. Následne je pole postupne prechádzané od najmenšieho indexu po najvačší a hodnota na aktuálnom indexe je vymenená s hodnotou na indexe určenom hodnotou premennej `j`. Táto je tvorená ako súčet jej predchádzajúcej hodnoty s aktuálnou hodnotou v stavovom poli `S` na indexe `i` a aktuálnou hodnotou znaku tajného kľúča `key[i]`. Indexy polí sú ošetrené pomocou operácie modulo, tak aby indexovali cieľové pole kruhovým spôsobom. Z uvedených skutočností vyplýva že výsledná podoba stavového poľa závisí len na hodnote tajného kľúča `key`.

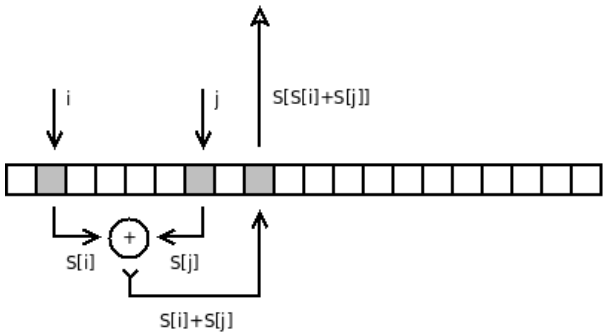
Po počiatočnej inicializácii sú oba indexy `i` a `j` vynulované, a nasleduje fáza generovania keystreamu. Tento proces beží iteratívne, kde počet iterácii je daný počtom bajtov správy ktorú treba zašifrovať. Princíp činnosti bude ozrejmený na pseudokóde, ktorý bol rovnako ako predchádzajúci prebraný z [4].

```
1     i := 0
2     j := 0
3     while GeneratingOutput:
4           i := (i + 1) mod 256
5           j := (j + S[i]) mod 256
6           swap(&S[i],&S[j])
7           output S[(S[i] + S[j]) mod 256]
8     endwhile
```

Algoritmus 2.2: Generovanie keystreamu

V priebehu generujúceho cyklu je stavové pole znova lineárne prechádzané (index `i`). Aktuálna hodnota je opäť vymenená s hodnotou na indexe `j`. Hodnota indexu `j` je ale teraz tvorená len ako súčet jej predchádzajúcej hodnoty s aktuálnou hodnotou v stavovom poli S na indexe i. Hodnota znakov kľúča už nieje braná v úvahu. Hodnota kľúča sa premietla počas inicializácie do počiatočného

stavu algoritmu, na ktorý táto fáza naväzuje. Po výmene hodnôt v stavovom poli bude z tohoto poľa vybraná výstupná hodnota, ktorá sa použije na zašifrovanie dát. Princíp výberu tejto hodnoty je znázornený na obrázku 2.2



Obrázok 2.2: Výber aktuálneho výstupu šifry

Po výmene nasleduje sa hodnotami i a j naindexuje stavové pole. Hodnoty `S[i]` a `S[j]` sa sčítajú a výsledná hodnota sa opäť použije jako index do stavového poľa. Výstupný bajt keystreamu sa teda prečíta zo stavového poľa z indexu `S[i]+S[j]`.

Výsledný bajt keystreamu sa kombinuje so šifrovanými dátami pomocou operácie *XOR*. Vzhľadom na skutočnosť, že operácia *XOR* je symetrickou operáciou a generovanie keystreamu prebieha vždy rovnako, dešifrovanie prebieha rovnako ako šifrovanie.

2.4 Útoky na WEP

2.4.1 Kryptoanalitický útok

V súčasnej dobe existuje niekoľko voľne dostupných nástrojov, ktoré sú za určitých podmienok schopné rozlúštiť WEP kľúč. Tieto nástroje využívajú bezpečnostných problémov, ktoré vnáša do algoritmu použitie inicializačných vektorov a autentizácia paketu pomocou ICV. Samotná šifra RC4 doposiaľ nebola prekonaná a je považovaná za bezpečnú.

Základným požiadavkom na bezpečné použitie prúdových šifier, je jednorázovosť kľúča Teda tajný kľúč by nemal byť nikdy použitý na zašifrovanie dvoch rôznych správ. Ak toto nieje dodržané, prípadný útočník získa dve, pripadne viac, rôznych správ zašifrovaných rovnakým keystreamom. Dostatočný počet rôznych správ zašifrovaných rovnakým rovnakým keystreamom dáva prípadnému útočníkovi možnosť rekonštruovať použitý keystream a následne aj tajný kľúč Algoritmus WEP sa snaží tento problém riešiť pomocou inicializačných vektorov, ktoré sú generované pre každú správu správu pseudonáhodne. Tento mechanizmus sa však časom ukázal nevyhovujúci, nakoľko:

- Inicializačný vektor je posielaný v nešifrovanej podobe a je útočníkovi známy
- IV má dĺžku 24 bitov, priemerne zaťažený AP vyčerpá všetky IV za približne 12 hod
- Ak sú IV generované náhodne, s pravdepodobnosťou 0.5 nastane kolízia už po 5000 paketoch (birthday paradox [5])
- Situácia je ešte horšia pri bezdrôtových kartách, ktoré IV generujú sekvenčne. Tieto karty typicky začínajú od IV z hodnotou 0 po každom reboote.
- WEP štandard [6] odporúča, avšak nepožaduje meniť hodnotu IV pre každý paket.

Z uvedeného je zrejmé, že principiálne je možné získať dostatočný počet paketov s rovnakým IV, avšak získanie vhodného materiálu môže byť veľmi náročné na čas a množstvo odchytených dát.

Ďalším slabým miestom zabezpečenia WEP je autentizácia správy pomocou kryptosúčtu ICV. Tento je počítaný ako CRC32 z prenášanej správy, ktorý je následne zašifrovaný rovnakou šifrou ako dáta. Je nevyhnutné poznamenať, že CRC niekde kryptografická hashovacia funkcia a jej primárnym cieľom je poskytnúť mechanizmus na detekciu vzniku chyby pri prenose. Skutočnosť, že je známy algoritmus na nájdenie kolíznych dát pre CRC, umožnila vzniknúť paket replay útoku.

Podstata paket replay spočíva v odchytení vhodného paketu, jeho vhodnej modifikácii a opätovného vyslania do siete. Účelom paket replay, je vhodne stimulovať prístupový bod, alebo klientské adaptéry tak, aby generovali veľké množstvo paketov. Tento prístup rapídne skracuje čas nevyhnutný pre odchytenie vhodného materiálu zo siete.

Z voľne dostupných nástrojov uvediem len *airodump-ng* ktorý umožňuje ukladať dáta zachytené na L2 v bezdrôtovej sieti,ďalej *aireplay-ng* umožňujúci stimulovať cieľovú sieť pomocou falošných arp dotazov a nakoniec *aircrack-ng* ktorý sa pokúša prelomiť heslo na základe nahromadených dát. Uvedené nástroje sú súčasťou aircrack baliku, ktorý je zdokumentovaný v [11] Za predpokladu že sa útočníkov podarí zachytiť dostatočné množstvo dát, je vlastné prelomenie hesla pomerne nenáročné na výpočetné zdroje. Prelomenie 104 bitového WEP si typicky vyžiada asi 3sekundy procesorového času, na systéme s procesorom Pentium M 1,7GHz a asi 3MB operačnej pamäte.

Kľúčovýmým problémom pri kryptografickom útoku je získať dostatočné množstvo vhodného materiálu. Toto môže byť dosiahnuté systematickým odpočúvaním siete, čo však vyžaduje mnoho času, alebo paket replay útokom, pri ktorom útočník riskuje odhalenie.

2.4.2 Útok hrubou silou

Útok hrubou silou sa snaží získať tajný kľúč odlišnou metódou ako rodina kryptoanalytických útokov. Princíp spočíva v slepom prehľadávaní stavového priestoru možných kľúčov Väčšina kryptografických principiálne algoritmov umožňuje úspešné útoky hrubou silou. Bezpečnosť týchto algoritmov je dosiahnutá vhodne zvolenou dĺžkou kľúča, vzhľadom na skutočnosť že mohutnosť množiny možných kľúčov rastie vzhľadom na dĺžku kľúča exponenciálne.

Náchylnosť na prelomenie šifry hrubou silou rastie spoločne s rastúcim výkonom výpočetných prostriedkov. Ako príklad uvediem prelomenie symetrickej šifry DES, ktorá používa 56bitov dlhý kľúč a v minulosti bola používaná dokonca ako bezpečnostný štandard Národným Bezpečnostným Úradom spojených štátov. Prvý úspešný útok hrubou silou na túto šifru bol realizovaný v roku 1998, za pomoci zariadenia zvaného Deepcrack. [7]. Toto zariadenie pozostávalo z 1200 aplikačne špecifických integrovaných obvodov, ktoré systematicky prehľadávali celý stavový priestor možných kľúčov Celé zriadenie malo dokázalo otestovať 90 miliard možných kľúčov za sekundu, a prehľadanie celého stavového priestoru trvalo približne 9 dní.

Na útok hrubou silou je náchylný predovšetkým WEP s dĺžkou kľúča 40bitov. Pri použití 40 bitového kľúča má množina všetkých možných kľúčov K mohutnosť vyjadrenú vzťahom 2.1.

$$|K| = 2^{40} \quad (2.1)$$

Mohutnosť tohto stavového priestoru môže byť ďalej redukovaná prehľadávaním len jeho časti. Typicky je vhodné prehľadať najviac pravdepodobnú časť možných kľúčov Tým útok hrubou silou stráca na úplnosti, avšak môže bežať niekoľkonásobne rýchlejšie, stále s dobrou pravdepodobnosťou na úspech. Typickým prístupom bývajú takzvané slovníkové útoky, ktoré stavový priestor redukujú na množinu slov (prirodzeného) jazyka ktorý potenciálny užívateľ používa.
Iný prístup redukuje množinu kľúčov na také, ktoré sa dajú zadať na klávesnici, teda na množinu ASCII znakov.

Základným predpokladom pre úspešnosť útoku hrubou silou je schopnosť rozpoznať ktorý z testovaných kľúčov je správny, teda ktorého použitie obnoví nešifrovanú správu. Toto je testovať buď na základe porovnania časti rozširovanej správy so známou časťou správy pôvodnej (npr. hlavička paketu) alebo pomocou porovnania kryptosúčtu dešifrovanej správy. Najspoľahlivejšou metódou pri útokoch na WEP sa ukazuje byť porovnávanie na zhodu ICV kryptosúčtu.

Z dostupných nástrojov ktoré sa snažia prelomiť algoritmus WEP hrubou silou uvediem napríklad GNU nástroj *wepcrack* ktorý dokáže realizovať systematický i slovníkový útok. Úspešnosť slovníkového útoku je merateľná len ťažko, a len na veľkej štatistickej vzorke prípadov. Úspech alebo neúspech v tomto prípade určuje vhodne zvolený slovník a jeho rozsah. V prípade systematického útoku je schopný tento nástroj odhaliť heslo na relatívne malej (niekoľko paketov) vzorke dát za niekoľko hodín, v závislosti na výpočetnom výkone stroja na ktorom je útok realizovaný.

2.4.3 Obrana

Vzhľadom na nedostatočné zabezpečenie ktoré poskytuje algoritmus WEP je vhodné ho nahradiť iným, bezpečnejším mechanizmom. Rôzny výrobcovia aktívnych prvkov poskytujú pre svoje produkty vylepšené verzie algoritmu WEP. Príkladom môže byť WEP+ alebo Dynamic WEP. Najčistejším riešením býva zvyčajne prechod na niektorý z mechanizmov IEEE802.11i teda *WPA* alebo *WPA2*, ktoré sú považované za bezpečné.

Ak prechod na WPA šifrovanie neje možný z titulu nevyhnutného hardwarového upgradu, je nevyhnutné doplniť tento mechanizmus ďalšími bezpečnostnými mechanizmami. V niektorýh prípadoch sa WEP zabezpečenie dopĺňa použitím MAC filtra na prístupovom bode. Tento postup nieje možné považovať za bezpečný, nakoľko mnohé sieťové karty umožňujú nastaviť ľubovolnú MAC adresu a tým obísť filtrovanie na základe L2 adries. Filtrovanie na základe MAC okrem toho neposkytuje žiadne prostriedky na zabezpečenie dôvernosti prenášaných dát.

Dostatočnú ochranu pred zneužitím zdrojov, i úniku informácii môžu zaistiť protokoly ktoré chránia prenášané dáta na sieťovej úrovni. Typicky sem patrí *PPTP*, *OpenVPN* a *IPsec(AH+ESP)*. Hlavnou nevýhodou šifrovania na L3, prípadne vyšších vrstvách OSI modelu, je pozorovateľný nárast záťaže klientských počítačov i prístupového servra. V prípade použitia IPsec býva častým problémom prechod šifrovaného spojenia cez preklad adries NAT.

Ak požadujeme zabezpečiť len časť sieťovej komunikácie, je vhodné zvážiť zabezpečenie na transportnej vrstve, pomocou protokolov *TLS/SSL*. Tieto však zvyčajne musí podporovať aplikácia, na oboch stranách, alebo je nevyhnutné použite vhodného SSL wrapperu, napríklad GNU aplikácie s*tunnel*

Uvedené je možno zhrnúť do záveru,že ak už je nevyhnutné WEP použiť mal by byť doplnený ďalšími bezpečnostnými mechanizmami, ktoré vhodne zabezpečia citlivé informácie pred zneužitím. Pri nákupe akéhokoľvek bezdrôtového zariadenia je vhodné sa presvedčiť či podporuje zabezpečenie WPA alebo WPA2.

3 Implementácia útoku

Súčasťou tejto práce je ukážková softwraová i hardwarová implementácia útoku hrubou silou na WEP so 40bitovým kľúčom Cieľom tejto implementácie nieje vytvoriť ďalší sofistikovaný nástroj na bezpečnostnú analýzu, ale demonštrovať vlastnosti a princípy tohto typu útoku a možnosti jeho akcelerácie v programovateľnom hardware. V priebehu implementácie som riešil množstvo problémov, ako napríklad odchytávanie vhodných dát zo siete, ich spracovanie a implementáciu kryptografických algoritmov. Klúčové poznatky, na ktoré som prišiel budú diskutované v nasledujúcom texte.

3.1 Voľba implementačného jazyka a hardwarovej platformy

Dôležitým rozhodnutím pri návrhu akéhokoľvek projektu je výber implementačného jazyka. Toto rozhodnutie musí byť dôkladne zvážené s prihliadnutím na požiadavky kladené na navrhovaný systém. V prípade hardwarovo akcelerovaného výpočtu je nevyhnutné okrem voľby implementačného jazyka vhodne zvoliť hardwarovú platformu pre obvod, v ktorom pobeží hardwarová časť výpočtu. Vzhľadom na komplexnosť analýzy požiadavkov sme ich rozdelili do dvoch skupín.

Kľúčové požiadavky na softwarovú časť projektu

- Funkčnosť a spoľahlivosť
- Prehľadnosť a udržovateľnosť
- Prenositeľnosť medzi rôznymi hardwarovými platformami a operačnými systémami
- Čo najvyšší výkon, na akejkoľvek hardwarovej platforme

Kľúčové požiadavky na hardwarovú časť projektu

- Funkčnosť a spoľahlivosť
- Prehľadnosť a udržovateľnosť
- Kompatibilita so vstupne/výstupnými prostriedkami súčasných osobných počítačov
- Dostatočná podpora výrobcu alebo komunity
- Dostupné API pre ovládanie hardwaru a prenos dát
- Vysoký výkon
- Nízka spotreba

Po dôkladnom posúdení všetkých uvedených požiadavkov sme zvolili implementačný jazyk C++. Tento je použiteľný na širokom spektre hardwarových platforiem i operačných systémov, je dostatočne univerzálny, a poskytuje bohatú sadu prostriedkov pre implementáciu navrhovaného systému

Po zvážení požiadavkou na hardwarovú časť projektu sme zvolili platformu poskytovanú spoločnosťou *PicoComputing*, konkrétne FPGA kartu *E14* [12]. Jedná sa o štandardnú PCMCIA Type-II kartu, obsahujúcu programovateľné hradlové pole *Virtex 4VFX60*, ďalej univerzálny procesor

Power-PC 405, 256MB operačnej pamäte typu DRAM, a 64 MB Flash pamäte pre uloženie designov pre FPGA a programov pre procesor. Kartu je možné programovať pomocou dodaného JTAG kábla, alebo priamo v PCMICIA slote pomocou dodanej aplikácie. Výrobca podporuje operačné systémy Windows i Linux a dodáva API pre jazyk C++. Spotreba celej karty je menšia ako 3.5W. Za implementačný jazyk hardwarovej časti bol zvolený VHDL, ktorý je v tejto oblasti priemyslovým štandardom.

3.2 Odchytenie vzorku dát zo siete

Základom úspešnej implementácie útoku je odchytenie vhodného kryptografického materiálu zo siete. Z pohľadu tohto projektu nevyhnutné odchytávať inicializačné vektory, dátové časti L2 rámcov a FCS kontrolné kryptosúčty. Tento materiál sa nachádza len v unicastových dátových rámcoch siete Je nevyhnutné odfiltrovať multicastový a broadcastový traffic. Ďalej je nevyhnutné odfiltrovať rámce pre riadenie siete ako beacon a probe request/response.

Z uvedeného vyplýva že pre účely získania vhodného materiálu a odfiltrovanie materiálu nevhodného je nutné pracovať s rámcami na linkovej vrstve OSI modelu. Tieto dáta sú zvyčajne spracovávané ovládačom sieťovej karty a jadrom operačného systému a užívateľskému programátorovi ostávajú skryté. Určitú možnosť poskytuje odchytávanie dát pomocou RAW socketov, ktoré umožňujú prístup k dátam na L2. Toto riešenie je však implementačne závislé na cieľovom operačnom systéme a prináša so sebou veľké množstvo komplikácii.

Na základe uvedených skutočností sme sa rozhodli na spracovanie linkových rámcov použiť knižnicu *libpcap* [14] ktorá rieši vačšinu spomenutých problémov. Je uvoľnená pod licenciou GNU a v súčastnosti je dostupná pre systémy Linux, Windows i rodinu systémov BSD. Zapuzdruje implementačne závislé konštrukcie a vytvára štandardné programátorské rozhranie a formát zachytených dát. Vďaka štandardnému formátu dát je možné použiť okrem priameho odchytávania dát zo siete aj off-line načítanie dát zo súboru vo formáte .pcap Tento formát súboru podporuje napríklad sieťový analyzátor *Wireshark* [13], alebo utilita tcpdump [14]. Odchytávanie dát je vďaka tomu možné vykonať na inom stroji ako vlastný útok.

Sada funkcii knižnice pcap pristupuje k súboru alebo sieťovému hardware pomocou takzvaného handle. Tento je možné vytvoriť volaním funkcii

```
pcap_t* handle = pcap_open_live(const char *device, int snaplen,
                                int promisc, int to_ms, char *errbuf);
```

Prvým argumentom je cesta k charakterovému súboru zariadenia, druhým maximálna dĺžka dát odchytených pred spracovaním. Tretím argumentom je príznak odchytávania dát v promiskuitnom režime. Pre potreby tohto projektu je promiskuitné odchytávanie nevyhnutné. Predposledným parametrom je počet milisekúnd, pred spracovaním každého bloku odchytených dát. Vzhľadom na skutočnosť, že dynamika a interaktivita navrhovaného systému niesu kritickými požiadavkami,bola táto hodnota nastavená na 1000ms Posledným argumentom je pole znakov, do ktorého bude uložené prípadné chybové hlásenie.

Alternatívou k priamemu odchytávaniu zo siete je načítanie dát zo súboru. To je možné vykonať pomocou

```
pcap_t* handle = pcap_open_offline(const char *fname, char *errbuf);
```

Prvým argumentom je meno súboru.Druhým pole znakov, pre prípad vzniku chyby. Obe spomínané funkcie vracajú NULL v prípade neúspechu.

Odchytávanie dát začína inicializáciou cyklu pomocou

```
int pcap_loop(pcap_t *p, int cnt,
              pcap_handler callback, u_char *user)
```

Prvým argumentom je handle,druhým počet odchytených paketov, alebo -1 pre stále odchytávanie. Parameter callback je ukazateľom na užívateľskú funkciu ktorej budú predané spracované dáta.

Táto funkcia má prototyp

```
void got_packet(u_char *user, const struct pcap_pkthdr *h,
                                    const u_char *packet);
```

Parameter `pcap_pkthdr` je štruktúra obsahujúca informácie o odchytenom pakete, ako jeho dĺžku a typ. Parameter `packet` je ukazateľom do pola obsahujúceho odchytené dáta, uložené po bajtoch vo formáte unsigned char. Keďže prototyp tejto funkcie je pevne daný, je nevyhnutné do nej vnášať ďalšie parametre a vynášať výsledky pomocou globálnych premenných.

Táto funkcia je vyvolaná pre každý zachytený rámec, takže je v nej nevyhnutné vykonať filtrovanie nepoužiteľných rámcov, ale aj výber najvhodnejšieho kandidáta pre útok. Výber vhodného kandidátneho rámca ma zásadný vplyv na produkované výsledky. Podstatným parametrom rámca je jeho dĺžka. S rastúcou dĺžkou rámca stúpajú nároky na výpočetný výkon a klesá celkový výkon útoku. S klesajúcou dĺžkou rámca možno očakávať rastúci počet kolízii pri výpočte ICV kryptosúčtu, teda viacero potenciálnych správnych kľúčov. Vzhľadom na stanovené ciele sme pri tomto rozhodovaní dali prednosť výkonu. Prax neskôr ukázala, že pri použití najmenšieho možného rámca, ktorý má dĺžku cca 60 bajtov, možno očakávať jednu až dve kolízie, v priebehu celého útoku. V prípade potreby možno negatívny vplyv kolízií eliminovať testovaním potenciálnych kľúčov na ďalších rámcoch.

V tejto funkcii je okrem filtrovania implementované aj spracovanie rámca a výber najvhodnejšieho kandidáta pre útok. Keďže presné špecifikácie rámca typu IEEE 802.11 nie sú bezplatne dostupné, vychádzali sme pri implementácii s verejne dostupných materiálov o tejto technológii. Výsledkom viacnásobného volania tejto funkcie je nasledovná dátová štruktúra:

```
struct frame{
      u_char srcmac[6];
      u_char dstmac[6];
      u_char ibss_mac[6];
      u_char iv[3];
      u_char icv[4];
      int datalen;
      bool encrypted;
      u_char data[2312];
};
```

Štruktúra obsahuje položky `srcmac` , `dstmac` , `ibssmac` pre uloženie zdrojovej a cieľovej MAC adresy a MAC adresy prístupového bodu, ktorý tento rámec odvysielal. Ďalej obsahuje položky pre uloženie inicializačného vektoru `IV` a kontrolného kryptosúčtu ICV. Položka `datalen` obsahuje dĺžku zachytených dát, ktoré sú uložené v poli data. Príznak `encrypted` je nastavený na true, jedna sa o zašifrovaný rámec. Premenná tohto typu je výstupom fázy zachytávania, alebo načítania dát a vstupom pre vlastný útok.

3.3 Spôsob prechádzania stavovéh priestoru a generovanie kľúčov

Princíp útoku hrubou silou spočíva v systematickom skúšaní všetkých možných, alebo niektorých najpravdepodobnejších kľúčov Na tento prístup je možne sa pozerať ako na slepé prehľadávanie celého stavového priestoru, alebo jeho určitej časti. Spôsob prehľadávania stavového priestoru môže v niektorých prípadoch urýchliť nájdenie kľúča, avšak na dostatočne veľkej štatistickej vzorke sa jeho význam stráca. Na nájdenie správneho kľúča je zo štatistického pohľadu prehľadať polovicu stavového priestoru. V tomto prípade naviac treba zohľadniť kolíznosť kryptosúčtu ICV.

Dramatický vplyv na rýchlosť a pravdepodobnosť úspechu má zvolená časť prehľadávaného stavového priestoru. Ak požadujeme absolútnu úspešnosť útoku, je nevyhnutné extenzívne prehľadať celý stavový priestor. Ak máme možnosť sa domnievať,že určitá podmnožina stavového priestoru je menej pravdepodobná, je vhodné ju prehľadávať až na koniec, prípadne ju neprehľadávať vôbec. V našej implementácii som sa rozhodol prehľadávať množinu kľúčov zložených s malých a veľkých písmen anglickej abecedy. Množina znakov z ktorých je tvorený kľúč je určená nesladovým poľom:

```
const char ascii[] =
{
      'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j',
      'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't',
      'u', 'v', 'w', 'x', 'y', 'z',
      'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',
      'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T',
      'U', 'V', 'W', 'X', 'Y', 'Z',
      '0', '1', '2', '3', '4', '5', '6', '7', '8', '9',
      ' ','!' };
```

Pomocou zmeny obsahu a poradia znakov v poli možno jednoduchým spôsobom modifikovať časť prehľadávaného stavového priestoru i poradie v akom sú jednotlivé potenciálne kľúče tvorené.

Dôležitou časťou vlastnej implementácie je spôsob tvorby kľúčov zo zadaných znakov. Nevyhnutnou požiadavkou na tvorbu kľúčov je úplnosť – algoritmus musí vygenerovať všetky možné kľúče, ktoré sa dajú z daných znakov utvoriť. Ďalším požiadavkom je rýchlosť. Algoritmus tvorby kľúčov by sa mal na celkovom procesorovom čase spotrebovanom na útok podieľať podla možnosti minimálne.

V našej implementácii sa kľúče zo znakov tvoria pomocou prevodu do číselnej sústavy, so základom, ktorý zodpovedá počtu znakov z ktorých sa kľúče generujú. Každý kľuč má priradené unikátne id, z ktorého možno pomocou prevodu do vhodnej číselnej sústavy vygenerovať jeho reprezentáciu. Ako príklad uvediem kľúč s id=0 má po po prevode hodnotu "`aaaaa`", kľúč s id=1 má hodnotu "`aaaab`" a posledný kľúč s id= 64^5 má po prevode hodnotu "`!!!!!`". Takýmto spôsobom je zabezpečené mapovanie lineárneho stavového priestoru, ktorý obsahuje hodnoty od 0 do 64^5 na nelineárny stavový priestor, obsahujúci ASCII hodnoty znakov.

3.4 Softwarový útok

Vlastný softwarový útok spočíva v opakovanom generovaní testovaného kľúča,dešifrovaní odchytenej dátovej časti paketu a výpočtu hodnoty ICV. Tieto činnosti sa dejú v cykle s veľmi veľkým počtom iterácii, a je vhodné aby boli implementované čo najefektívnejšie

Problematickým sa môže zdať zvolená metóda generovania kľúčov Pri prevode číselných sústav sa typicky používajú multiplikatívne operácie delenia a zvyšku po delení. Tieto operácie zvyčajne bývajú na všetkých architektúrach pomalé a náročné. Tu sa naskytá možnosť nahradiť delenie číslom 64, teda základom sústavy operáciou bitového posuvu o 6 bitov vľavo. Táto operácia má rovnakú sémantiku, a zvyčajne jej vykonanie býva zvyčajne rýchlejšie než vykonanie operácie delenia. Neskoršie testy a výkonnostný profiling však ukazujú, že generovanie kľúčov ma na celkový výkon len zanedbateľný vplyv. V záujme lepšej čitateľnosti kódu som použil klasický algoritmus na prevod do číselnej sústavy Generovanie kľúčov je implementované funkciou:

```
void make_pwd5(unsigned long dec,unsigned char pwd[5]);
```

Prvým parametrom funkcie je id kľúča, druhým je 5 bajtové pole znakov, ktoré bude naplnené kľúčom vygenerovaným na základe zadaného id.

Kritickým problémom je čo najefektívnejšie implementovať šifrovací algoritmus RC4.Tento pozostáva z inicializačnej časti, v ktorej vytvára stavové pole a vlastného generovania keystreamu. Implementáciou oboch častí algoritmu som prevzal z [4]. Túto implementáciu považujem za dostatočne odladenú a ďalej som ju nemodifikoval. Jedinou úpravou bolo zabalenie stavu algoritmu RC4 do nasledovnej štruktúry:

```
struct wep_state{
      unsigned char rc4_state[256];
      unsigned int rc4_i;
      unsigned int rc4_j;
};
```

Cieľom tohto opatrenia je sprehľadnenie kódu, ako aj redukcia počtu parametrov, ktoré sa predávajú do funkcii. V implementácii sa neskôr predávajú do funkcii len referencie na premennú tohto typu čo zamedzí kopírovaniu dátových štruktúr na zásobník funkcii. Súčasťou implementácie sú ďalej funkcie:

```
void rc4_init(unsigned const char *key,wep_state& w);
unsigned char rc4_output(wep_state& w);
```

Prvá z uvedených funkcii implementuje inicializáciu stavu šifrovacieho algoritmu, a je potrebné ju zavolať vždy pred zmenou používaného dešifrovaného kľúča(parameter `const char *key`). Druhá uvedená funkcia vracia jeden bajt keystreanu, ktorým budú dešifrované dáta. Z uvedeného vyplýva,že túto funkciu treba volať pre každý bajt správy, ktorá má byť dešifrovaná. Je dôležité poznamenať,že každé volanie tejto funkcie modifikuje vnútorný stav šifrovacieho algoritmu (parameter `wep_state& w`). Preto je nutné túto funkciu používať opatrne, vždy len na dešifrovanie jedného bajtu zašifrovanej správy. Tiež je nevyhnutné dodržiavať správne poradie volania tejto funkcie, a nepoužívať jej volanie v zložitých aritmetických výrazoch, nakoľko poradie vyhodnocovania výrazov nieje v jazyku C++ definované.

Ďalším výkonnostne kritickým krokom pri útoku hrubou silou je výpočet kontrolného súčtu CRC32, pre účely overenia kryptosúčtu ICV. Vypočítané CRC sa porovná s dešifrovanou hodnotou ICV a ak sa obe hodnoty zhodujú, kľúč s ktorým bola správa dešifrovaná je s veľkou pravdepodobnosťou kľúčom správnym. Pre dosiahnutie vysokého výkonu softwarového útoku je nevyhnutné implementovať tento algoritmus čo najefektívnejšie Z tohto dôvodu bola použitá tabuľková implementácia CRC algoritmu. Pri použití takejto implementácie sa nepočíta výsledné CRC pomocou tradičného delenia zvoleným polynómom bit po bite, ale používa sa predpočítaná 256 Bajtová tabuľka v ktorej sa nachádzajú hodnoty CRC pre všetky hodnoty, ktoré môže jeden bajt správy nadobúdať. Jedná sa vlastne o istý druh výmeny pamäťovej náročnosti za časovú. Celý algoritmus je implementovaný v nasledujúcich dvoch funkciách:

```
void crc_table();
unsigned int crc32(unsigned char *buffer, int BufferLength);
```

Funkcia crc_table inicializuje globálne pole `CRC32Table` predpočítanými hodnotami. Funkcia `crc32` vracia 32-bitovú hodnotu vypočítaného CRC. Algoritmus na výpočet sa zhoduje s oficiálnym algoritmom IEEE. Použitá hodnota polynómu je definovaná vo funkcii `crc_table` ako:

```
unsigned int ulPolynomial = 0x04c11db7;
```

Tento polynóm bol stanovený organizáciou IEEE za štandardný pre technológiu 802.3 ethernet,a používa sa aj pri tvorbe kontrolného ICV súčtu. Výsledná hodnota vypočítaného CRC sa zhoduje s výsledkami, ktoré produkuje POSIX utilita cksum.

Kompletný proces inicializácie stavu šifry,dešifrovania rámca a overenia jeho kryptosúčtu je implementovaný funkciou

```
bool wep_decrypt(unsigned char key[],frame & input,frame& output);
```

Táto očakáva vstupné parametre `key[]` a **`frame`** `& input,` teda dešifrovací kľúč a vstupný zašifrovaný rámec. Výstupom je rámec dešifrovaný zadaným klúčom. Funkcia vracia true, ak sa úspešne podarilo overiť kryptosúčet a dešifrovací kľúč je možno považovať za správny.

Vzhľadom, na spôsob overovania správnosti kľúča založeného na algoritme CRC je nevyhnutné počítať so vznikom občasných kolízii a teda aj nájdením viacerých potenciálne správnych klúčov. Pravdepodobnosť vzniku kolízie pri použití 32-bitového polynómu je $p = 2^{-32}$ .Vzhľadom na veľkosť stavového priestoru je možné očakávaný počet kolízii vyjadriť vzťahom 3.1

$$Ex = 64^5 \cdot 2^{-32} \quad (3.1)$$
$$Ex = 0.25$$

V priemere je teda možné očakávať kolíziu s pravdepodobnosťou 0.25.S prihliadnutím na túto skutočnosť v našej implementácii prehľadávame vždy celý stavový priestor kľúčov, a potenciálne kľúče ukladáme do poľa. Veľkosť poľa na potenciálne kľúče bola stanovená empiricky na 128 potenciálnych klúčov. Pravdepodobnosť 128 kolízii v priebehu útoku je natoľko nízka že je možné ju zanedbať.

Účelom implementácie softwarovej implementácie útoku bolo objasniť použité metódy , navrhnúť riešenie potenciálnych problémov a poskytnúť referenčný model pre výkonnostnú analýzu. Výstupom tejto časti implementácie, je program v jazyku C++, schopný odchytiť dáta zo siete alebo načítať ich zo súboru a pomocou výpočtu na štandardnom aritmetickom procesore osobného počítača odhaliť množinu potenciálnych šifrovacích kľúčov

3.5 Hardwarový útok

Princíp hardwarového útoku spočíva v nahraní odchyteného materiálu do zvolenej hardwarovej platformy, v ktorej je implementovaný algoritmus generovania kľúčov, vlastný dešifrovací algoritmus RC4 a algoritmus na výpočet kryptosúčtu CRC. Cieľom pri implementácii hardwarového útoku, je dosiahnuť vysoký stupeň paralelizmu a súčasne dosiahnuť vysokej hodinovej frekvencie, s ktorou bude výsledný obvod pracovať. Tomuto cieľu bude podriadený celý návrh obvodu i jeho implementácia.

3.5.1 Programovateľný hardware a platforma Virtex 4

Možnosti a vlastnosti cieľovej architektúry pri vývoji hardwarového obvodu je potrebné zohľadniť rovnakou mierov ako možnosti programovacieho jazyka a možnosti operačného systému pri vývoji softwaru. V tomto prípade je cieľovou architektúrou programovateľné hradlové pole typu Virtex 4. Toto ponúka vývojárovi v závislosti od verzie niekoľko desiatok tisíc konfigurovateľných logických blokov nazývaných slice. Každý slice obsahuje dve look-up tabuľky (LUT) pre realizáciu kombinačnej logiky a dva flip-flop (FF) klopné obvody pre realizáciu sekvenčných členov. Technológia ďalej obsahuje niekoľko desiatok až stoviek blokových RAM pamätí (BRAM), vhodných pre implementáciu rozsiahlych dátových štruktúr.

Look-up tabuľky sú typicky 4 vstupové kombinačné členy, ktoré mapujú kombináciu 4 logických vstupov na jeden výstup. Pomocou týchto prvkov sa dajú realizovať kombinačné logické funkcie, ako AND, OR, XOR a tiež napríklad sčítačky.

Pomocou Flip-Flop klopných obvodov sa dajú realizovať registre a iné sekvenčné prvky. Pomocou týchto prvkov je možné realizovať i menšie pamäte typu RAM, ktoré si neprajeme implementovať pomocou pamätí Block-RAM.

Nepostrádateľnou časťou programovateľného hradlového poľa je prepojovací systém, ktorý umožňuje vhodným spôsobom prepojiť jednotlivé logické bloky. Súčasťou tohto systému sú klasické dátové cesty, ako aj špeciálne cesty s nízkym oneskorením pre rozvod hodinového signálu.

Hradlové pole začína plniť implementovanú funkcionalitu až po nahraní takzvaného konfiguračného súboru. Tento je výsledkom procesu syntezy zdrojového kódu, a obsahuje konfiguráciu logických členov a informácie o ich vzájomnom prepojení. Vlastný konfiguračný súbor má veľkosť niekoľko MB a jeho nahratie do hradlového pola trvá rádove niekoľko sekúnd. Problematické je však vytvorenie, tzv syntéza, konfiguračného súboru so zdrojového kódu. Táto môže trvať rádove aj niekoľko hodín. Hradlové polia nedisponujú energeticky nezávislou pamäťou, takže konfiguračný súbor musí byť nahraný po každom pripojení na napájacie napätie.

Výhodou použitia programovateľného hradlového poľa je možnosť využiť prostriedky ktoré poskytuje architektúra na realizáciu práve takých aritmetických a logických funkcii, ktoré sú potrebné pre implementáciu zvolenej úlohy. Ďalšou výhodou ktorú poskytuje hardwarová realizácia je možnosť vykonávať niekoľko operácii paralelne, v nezávislých jednotkách.

Práve vysoká miera paralelizmu umožňuje často i niekoľkonásobne rýchlejší beh hardwarovo implementovaných algoritmov. Veľké množstvo zdrojov dostupných v moderných hradlových poliach často dovoľuje do jednej fyzickej súčiastky implementovať viacero paralelných jednotiek, implementujúcich rovnaký algoritmus. Hardwarová akcelerácia je teda obzvlášť výhodná pre problémy, ktoré je možné ľahko rozdeliť na podproblémy a vykonávať paralelne. Do tejto kategórie problémov typicky patrí spracovanie zvuku a obrazu, ale aj nami požadované prehľadávanie stavového priestoru. Pri tomto type úlohy je možné použiť stratégiu rozdeľuj a panuj – rozdeliť stavový priestor na menšie podpriestory, pričom každý podpriestor je prehľadávaný paralelne, nezávislou jednotkou.

Z uvedených skutočností plynú požiadavky na optimálnu implementáciu, ktorými sú hlavne: vysoký stupeň paralelizmu, vysoká synchronizačná frekvencia a rovnomerné využitie zdrojov hriadeľového poľa. Tieto požiadavky, i keď sa to tak na prvý pohľad nejaví, sú do značnej miery protichodné. Snahy o maximálnu paralelízáciu výpočtu zvyšujú oneskorenia vznikajúce v logike i v prepojovacom systéme. Dôsledkom toho veľká miera paralelizmu znemožňuje použitie vyšších synchronizačných frekvencii. Snaha o minimalizáciu oneskorení a zvyšovanie pracovnej frekvencie vedie k nerovnomernému využitiu technologických zdrojov. V našej implementácii sme preto zvolili čo najoptimálnejší kompromis, medzi všetkými uvedenými prístupmi.

3.5.2 Karta PicoComputing E14

Samotné hradlové pole je len veľmi ťažko použiteľné na akceleráciu výpočtu prebiehajúceho v osobnom počítači. Problémom je realizácia elektrického prepojenia medzi FPGA a niektorou s univerzálnych zberníc v PC. Ďalšie problémy vznikajú pri začleňovaní obvodu implementovaného hradlovým polom do adresného priestoru osobného počítača. Ďalším faktorom, ktorý komplikuje implementáciu je nevyhnutnosť implementácie ovládača pre nami implementovaný hardware.

Túto situáciu využívajú mnohé komerčné subjekty a prichádzajú s kitmi ktoré tieto problémy riešia. Nami zvolená platforma Pico E14 existuje vo forme štandardnej PCMCIA karty ku ktorej je dodávaný ovládač a ovládacia utilita, ktorá umožňuje vykonávať servisné úkony s

kartou,predovšetkým nahranie a mazanie konfigurácii pre FPGA a komunikáciu s kartou. Zapisovať a čítať dáta s Pico karty je možné aj pomocou dodaného ovládača prostredníctvom štandardných volaní operačného systému. Vývoj vlastného hardware na tejto platforme je podporený výrobcom dodanou implementáciou komunikácie po PCMCIA zbernici. Táto je dodávaná vo forme niekoľkých implementačných súborov v jazyku verilog. Po pridaní týchto súborov do projektu má užívateľský programátor k dispozícii zbernicu PicoBus s nasledovnými signálmi

```
PicoRst:        in  std_logic;
PicoClk:        in  std_logic;
PicoAddr:       in  std_logic_vector(31 downto 0);
PicoDataIn:     in  std_logic_vector(31 downto 0);
PicoRd:         in  std_logic;
PicoWr:         in  std_logic;
PicoDataOut:    out std_logic_vector(31 downto 0);
```

Signály `PicoDataIn` a `PicoDataOut` tvoria obojsmernú 32-bitovú dátovú zbernicu. Signál PicoAddr slúži pre zasielanie adresy. Riadenie zbernice zabezpečujú signály `PicoRst`,`PicoRd` a `PicoWr`. Jediným dostupným synchronizačným signálom je signál PicoClk, ktorý má frekvenciu 33MHz. Celková prenosová kapacita zbernice je teda 32bit*33Mhz = 133MB/s v režime full duplex.

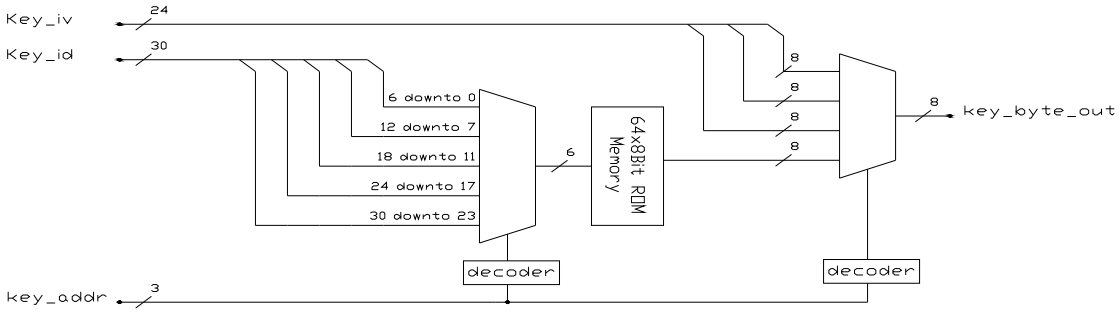
Nepríjemnou skutočnosťou je prítomnosť jediného hodinového signálu. Táto skutočnosť je prekážkou pri jemnom ladení výkonu celého zariadenia, dá sa však kompenzovať použitím komponentov na modifikáciu hodinového signálu, takzvanými DCM modulmi.

3.5.3 Kryptografická jednotka

Jadrom celého obvodu je jednotka, ktorá implementuje šifrovací algoritmus RC4 a overovanie kontrolného súčtu ICV. Štruktúra jednotky musí byť navrhnutá a implementovaná tak, aby bolo možné tieto jednotky prevádzkovať paralelne a nezávisle na sebe. Ďalej by kryptografická jednotka mala pozostávať s minimálneho množstva komponentov, tak aby bola čo najkompaktnejšia a spotrebovala čo najmenšie množstvo zdrojov v cieľovej technológii. Z uvedeného vyplýva, že jednotku je vhodné navrhnúť tak aby mohla byť riadená centrálne, spolu s ostatnými paralelne pracujúcimi jednotkami pomocou spoločného hardwarového automatu. Pri implementácii je tiež nevyhnutné myslieť na to, že pri overovaní kryptosúčtu môže dochádza ku kolíziám vo viacerých jednotkách súčasne,preto musí každá z nich obsahovať prostriedky na uchovanie potenciálnych kľúčov

Keďže si kladieme za cieľ paralelné testovanie viacerých kľúčov, musí byť súčasťou kryptografickej jednotky je kombinačná logika na prevod reprezentácie kľúča pomocou jeho id kľúča na jeho skutočnú podobu. Táto môže byť implementovaná pomocou niekoľkých ROM pamätí, ktoré bude vykonávať mapovanie id na skutočnú podobu. Avšak s prihliadnutím na skutočnosť, že šifra v jednom okamihu vyžaduje vždy len jeden bajt kľúča je možné implementovať prevod id pomocou jedinej ROM pamäte obsahujúcej ASCII hodnoty testovaných znakov. RTL schému tohto jednoduchého kombinačného obvodu uvádzame na obrázku 3.1. Vstupom do kombinačnej logiky je poradové ID kľúča key_id a signál key_addr ktorý určuje ktorý znak kľúča sa má objaviť na výstupe. Jedná sa o troj bajtový signál, ktorý môže nadobúdať hodnôt od "`000`" do "`111`" a ovláda oba

multiplexory v obvode. Pri hodnotách `key_addr` od `0` do `3` prepína výstupný multiplexor na výstup postupne hodnoty inicializačného vektoru `Key_iv`. Toto súhlasí s princípom algoritmu WEP, prvé tri bajty kľúča sú IV.



Obrázok 3.1: Kombinačná logika generátora klúčov

Pre hodnoty `key_addr` od `4` do `8` je výstupný multiplexor prepnutý tak, že na výstupe sa objaví výstup pamäte PROM. Na jej adresný vstup postupne privádza vstupný multiplexor 6 bitové časti signálu Key_id. Každá 6 bitová skupina zodpovedá jednému z 2^6 znakov v PROM pamäti. Signál key_byte_out je vstupom pre vlastný algoritmus inicializácie stavového poľa.

Vzhľadom na to,že RC4 je prúdová šifra, centrálnym bodom celej hardwarovej implementácie bude komponenta na uchovávanie aktuálneho stavu algoritmu. Stavové pole tohto šifrovacieho algoritmu má veľkosť 256 Bajtov, a algoritmus s ním v každom kroku intenzívne pracuje. Pamäťové operácie sa pomerne ťažko paralelizujú a prístup k pamäti sa vo výsledku môže stať úzkym hrdlom celej implementácie. Preto sme v našej implementácii zvolili implementáciu stavového poľa pomocou dvojportovej Block-RAM pamäte Táto umožňuje v priebehu jedného hodinového taktu vykonať ľubovolné dve vstupne-výstupné operácie. Cieľom implementácie bude čo najefektívnejšie využiť možnosť paralelnej realizácie pamäťových operácii a tak minimalizovať negatívny dopad závislosti na práci s pamäťou.

Implementáciu algoritmu možno rozdeliť do dvoch častí, ktorými sú inicializácia stavového poľa a vlastné dešifrovanie. S prihliadnutím na poznatky získané pri softwarovej implementácii je možné očakávať, že inicializačná časť algoritmu je výkonnostne náročnejšia ako vlastné dešifrovanie malého množstva testovacích dát. Preto budeme pri návrhu a implementácii venovať tejto časti zvýšenú pozornosť. Inicializačnú časť algoritmu je možné dekomponovať na nasledovnú postupnosť krokov:

- Pre všetky hodnoty `i` od `0` do `255`
  - Lineárna inicializácia stavového poľa S, tak že `S[i]=i`
- Pre všetky hodnoty `i` od `0` do `255`
  - Načítanie hodnoty `S[i]` a výpočet novej hodnoty `j`
  - Načítanie hodnoty `S[j]`
  - Vzájomná výmena hodnôt `S[i]` a `S[j]`

V prípade naivnej implementácie takéhoto riešenia by sme potrebovali 255 hodinových taktov na lineárnu inicializáciu a 3x255 taktov na vytvorenie požadovaného obsahu stavového poľa. Avšak v tomto prípade je potenciál dvojportovej pamäte nevyužitý,nakoľko okrem výmeny hodnôt `S[i]` a `S[j]` je vždy vykonávaná len jedna vstupne výstupná operácia.

Vzhľadom na skutočnosť, že sa jedná o útok hrubou silou, je možné očakávať, že inicializácie stavových polí budú vykonávané mnohokrát tesne po sebe. Tejto skutočnosti možno využiť a riešiť inicializáciou stavových polí metódou *double buffe*r. Princíp spočíva v použití dvoch stavových polí miesto jedného. Počas vytvárania obsahu jedného poľa je druhé lineárne inicializované a naopak. Za predpokladu použitia *double buffer* princípu je možné algoritmus inicializácie dekomponovať na nasledovnú postupnosť krokov

- Pre všetky hodnoty `i` od `0` do `255`
  - Načítanie hodnoty S[i] a výpočet novej hodnoty j a inicializácia hodnoty v druhom stavovom poli, na adrese S[i+256]
  - Načítanie hodnoty `S[j]`
  - Vzájomná výmena hodnôt `S[i]` a `S[j]`

Takáto implementácia vyžaduje 3x256 hodinových cyklov a lepšie využíva potenciálu dvojportovej Block-RAM pamäte Časová úspora ktorú použitie takéhoto prístupu prináša je 256/1024, teda 25%. Daňou za toto urýchlenie je nevyhnutnosť zdvojnásobiť množstvo pamäte vyhradenej na implementáciu stavového poľa. Toto však s prihliadnutím na cieľovú architektúru nieje nevýhodou, pretože platforma *Virtex 4VFX60* obsahuje 252 blokových RAM pamätí s konštantnou veľkosťou až 18kb. Bloková RAM pamäť bude obsadená rovnako, bez ohľadu na to, či z nej využijeme 256 alebo 512B. Tento princíp vyžaduje aby sme pred vlastným spustením útoku prvý krát inicializovali jeden z dvoch bufferov. Tento krok je potrebné vykonať len raz, a je vhodného ho vykonať hneď po resete spoločne s nulovaním registrov rámci prípravy obvodu na činnosť.

Celková implementácia algoritmu je znázornená na obrazovej prílohe 1 v hornej časti. Inicializácia začína tak, že multiplexor `Mux A` pripojí na dátový vstup pamäte Block-RAM `mem_b_in` signál `addr_cnt_in`. Jedná sa o výstup čítača ktorý je riadiacou logikov inkrementovaný v rozmedzí hodnôt od 0 do 255. Tento signál súčasne pripojí multiplexor `Mux B` na adresný port pamäte (signál `mem_b_adr`) takže aktiváciou zápisového signálu `mem_wen_a` sa lineárne inicializuje jedno pamäťové miesto.

Dôležitú úlohu majú signály `addr_a shift` a `addr_b_shift`, ktoré ovládajú najvýznamnejší bit adresy. Pomocou tohto bitu sa ovláda prepnutie na druhý buffer. V tomto okamihu musí byť dodržaná zásada, že ak jej jeden zo signálov `addr_a shift` a `addr_b_shift` na hodnote log.1 ten druhý musí byť na hodnote Log.0. To preto, že z jedného buffru sa číta hodnota stavového poľa a súčasne sa inicializuje hodnota v druhom buffri pre budúce použitie. V tom istom okamihu je však signál `addr_cnt_in` pripojený i na druhý adresový port (signál `mem_b_adr`).

Čítanie aktuálnej hodnoty , ktorá bude v ďalšom texte označovaná ako `S[i]` prebieha jednoducho tak, že signál `Addr_cnt_in` je priamo pripojený na adresový vstup `mem_a_addr`. S nábežnou hranou ďalšieho hodinového impulzu sa objaví požadovaná hodnota `S[i]` na výstupe `mem_a_out` odkiaľ pokračuje do sčítačky kde sa k nej pričíta aktuálna hodnota v registry `J-Reg`. Súčet pokračuje do ďalšej sčítačky v ktorej sa k tejto hodnote pripočíta hodnota výstupu bloku

Keygen. Táto reprezentuje aktuálny bajt dešifrovacieho klúča. Týmto spôsobom je hardwarovo implementovaný výraz v algoritme 2.1 na riadku č.6.

Nová hodnota j sa privedie na vstup registra J-Reg a súčasne cez multiplexor Mux B na adresový vstup mem_b_adr. V nasledujúcom hodinovom takte sa nová hodnota j uloží do J-Reg a na výstupe mem_b_out sa objaví hodnota S[j]. Táto hodnota je priamo privedená na vstup mem_a_in. Hodnota S[i] ktorá sa stále nachádza na porte mem_a_out sa cez Mux A privedie na vstup mem_b_in. Riadiaca logika súčasne aktivuje zápisové signály mem_wen_a a mem_wen_b a s nábežnou hranou ďalšieho hodinového impulzu sa hodnoty S[i] a S[j]. Takto je implementovaný krok algoritmu 2.1 na riadku 7.Všetky uvedené kroky sa opakujú pre každé i od 0 do 255. Takýmto spôsobom je po 256-tich iteráciách jedno stavové pole naplnené požadovanými hodnotami a pripravené na tvorbu keystreamu a druhé stavové pole lineárne inicializované a pripravené pre budúce použitie.

Ďalším krokom je implementácia vlastnej šifrovacej časti RC4 algoritmu, pri ktorom je generovaný kesytream. Prirodzeným požiadavkom je implementovať túto časť pokiaľ možno s využitím hardwarových komponent, ktoré sú využité v procese inicializácie stavového poľa. Vzhľadom na to, že oba procesy sú si veľmi podobné, nieje veľkým problémom tento požiadavok naplniť.

V procese generovania keystreamu sa znova vyčísľuje nová hodnota obsahu J-Registra. Do tejto hodnoty sa však už nepremieta hodnota dešifrovacieho kľúča Nová hodnota j sa tvorí podľa vzťahu definovaného v algoritme 2.2 na riadku 5.
Vplyv hodnoty aktuálneho bajtu keystreamu je možno zariadiť prepnutím multiplexora Mux C Tak aby na jeho výstupe bol potenciál log.0. Týmto sa na vstup príslušnej sčítačky dostane nulová hodnota, ktorá nijako neovplyvní vstup, a tento sa bude v nezmenenej forme propagovať na výstup.

Proces ďalej pokračuje rovnako ako v predchádzajúcom prípade výmenou hodnôt S[i] a S[j] po tomto kroku prichádza na rad vlastné generovanie bajty keystreamu. Toto sa deje podľa vzťahu v algoritme 2.2 na riadku 7. V hodinovom takte , v ktorom prebieha výmena hodnôt S[i] a S[j] sú obe tieto hodnoty na výstupoch pamäťe. Stačí ich sčítať v ďalšej sčítačke a priviesť cez multiplexor Mux B na adresný vstup pamäte mem_b_addr. Na dátovom výstupe sa v nasledujúcom takte objaví platná hodnota keystreamu, ktorú je možno použiť na dešifrovanie dát. Keďže generovanie keystreamu vyžaduje ďalší prístup do pamäte, jedna iterácia tohto algoritmu trvá 4 hodinové takty. To je o jeden takt viac ako v prípade inicializácie,v tomto prípade to však nieje na závadu, nakoľko budeme dešifrovať krátke vzorky dát.

Vygenerovaný keystream sa kombinuje so zašifrovanými dátami pomocou logického obvodu xor a vytvára tak pôvodnú nešifrovanú správu. Logika pre riadenie vlastného dešifrovania dát ja znázornená na spodnej časti obrázovej prílohy 1. Najprv sa cez multiplexor Mux D privádzajú na vstup logického člena xor odchytené dáta. Tieto postupne privádza na vstup frame_data_in ovládacia logika.

Po skombinovaní dát s keystreamom pokračujú dešifrované dáta do bloku CRC32. Tento je vlastne kombinačnou sieťou zloženou s xor členov implementujúcich crc polynóm. Blok v sebe obsahuje register na uchovanie výslednej hodnoty CRC. Táto sa postupne aktualizuje, v okamihoch kedy ovládacia logika nastaví signál data_rdy na log.1. V tomto okamihu sú navzorkované vstupné dáta premietnuté do výsledku a uložené do výstupného registra tohto bloku.

Po dešifrovaní všetkých testovacích dát a vyrátaní CRC32 je potrebné overiť kryptosúčet ICV. Tento treba najprv tiež dešifrovať a získať hodnotu CRC32, ktorú nastavil odosielateľ rámca. Keďže hodnota ICV je štvorbajtová potrebné ju dešifrovať postupne a jednotlivé dešifrované bajty uschovať v registroch ako medzivýsledky. Privádzanie správnych častí ICV na vstup xor členu zabezpečuje multiplexor `Mux E`. Na výstupe multiplexoru zachytáva dešifrované časti ICV jeden zo štvorice registrov. Povolovacie vstupy registrov riadi dekodér, ktorý dekóduje adresu vstupného multiplexoru do kódu 1 z N, čím aktivuje vždy práve jeden register. Po dešifrovaní ICV sa v každom registry nachádza jedna štvrtina pôvodného CRC. Výstupy oboch registrov sú konkatenované do jedného 32-Bitového signálu a privedené na vstup komparátora. Tu sa dešifrovaná hodnota CRC porovná s hodnotou vypočítanou v CRC bloku z dešifrovaných dát.

Ak obe hodnoty súhlasia, je možné, že bol nájdený správny dešifrovací kľúč, alebo došlo ku kolízii. V tomto okamihu nastaví ovládacia logika signál `crc_compare` na hodnoty log.1 a výstup komparátora sa propaguje na ovládací vstup multiplexora `Mux F`. Tento pripojí na adresový vstup pamäte `Key-RAM` výstup dvojbitového čítača, ktorý ukazuje na najnižšiu volnú adresu. Tento signál súčasne aktivuje zápis do pamäte, takže na najnižšiu volnú adresu sa zapíše hodnota aktuálneho `keyid`. Signálom zhody crc je tiež privedený na povoľovací vstup 2-bitového čítača, takže s nasledujúcou nábežnou hranou taktom bude inkrementovaný a opäť bude ukazovať na najnižšiu voľnú adresu.

Pre zabezpečenie efektívnej činnosti viacerých paralelných jednotiek je potrebné zabezpečiť aby všetky prehľadávali rovnako veľké a navzájom disjunktné časti stavového priestoru. Toto je zabezpečené privedením signálu `unit_id` do jednotky, ktoré určuje jej poradové číslo V jednotke sa tento signál použije na mieste najmenej významných bitov signálu `keyid` a celková šírka signálu `keyid` sa skráti tak aby v konkatenácii so signálom unit_id mal šírku 30 bitov. Tento systém je veľmi jednoduchý, ma ale zásadnú nevýhodu. Tou je skutočnosť že počet kryptografických jednotiek v systéme musí byť vždy mocninou dvojky. Táto skutočnosť´síce obmedzuje škálovateľnosť celého systému, ale pri implementácii na cieľovej architektúre sa neukázala byť výraznejšou nevýhodou.

Po dokončení výpočtu je potrebné zo všetkých kryptografických jednotiek prečítať výsledky. To znamená počet nájdených potenciálnych kľúčov, reprezentovaný dvojbitovým čítačom a obsah pamäte v ktorej sú uložené potenciálne kľúče Po skončení výpočtu je `Mux F` prepnutý tak, že na adresový vstup pamäti Key RAM sú pripojené dva najnižšie bity centrálnej adresovej zbernice `PicoAddr`. Pomocou multiplexora `Mux G` je možné pripojiť na výstup dátový výstup pamäte alebo stav čítača. Kvoli úspore prepojovacích prostriedkov technológie sú výstupy jednotky spojené internou trojstavovou zbernicou. Budič v jednotke pripája výstup na zbernicu, ak rozpozná na centrálnej adresovej zbernici na bitoch 16 až 22 adresu zhodnú s hodnotou `unit_id`. Týmto spôsobom je zabezpečené že v jednom okamihu je k internej zbernici pripojená najviac jedna jednotka.

3.5.4 Riadiaca logika

Samotná kryptografická jednotka obsahuje len nevyhnutné množstvo logiky ktoré je využité pri výpočte a nemôže byť spoločné pre všetky jednotky. Kryptografická jednotka teda obsahuje len komponenty ktorých činnosť závisí na aktuálnej hodnote testovaného kľúča Týmito komponentami sú hlavne pamäť stavového poľa, logika generovania kľúča, sčítačky, a obvod na overenie ICV. Jednotka z dôvodu úspory prostriedkov architektúry neobsahuje riadiaci automat a niektoré sekvenčné prvky, ktoré sa síce priamo podieľajú na výpočte, ale je možné aby boli spoločné pre všetky jednotky. Tieto sú hlavne pamäť RAM na uloženie testovacích dát a čítač, ktorý ukazuje do tejto pamäte na aktuálnu hodnotu dešifrovaného bajtu a register obsahujúci množstvo platných testovacích dát v pamäti.. Ďalej sem patrí čítač poradového čísla kľúča a čitáč ktorý ukazuje pri inicializácii stavového poľa na aktuálne spracovávanú hodnotu. Riadiaca logika je doplnená registrami na uloženie IV a ICV hodnoty testovacích dát. Zjednodušená schéma modulu ovládacej logiky je v prílohe 2. Sekvenčné prvky v tomto len dopĺňajú prostriedky pre výpočet v kryptografických jednotkách, skutočnosť, že sú súčasťou implementácie ovládacej logiky je len snaha vyjmuť maximálne množstvo prvkov "pred zátvorku" a minimalizovať spotrebu technologických zdrojov.

Jadrom ovládacej logiky je konečný automat ktorý ovláda všetky riadiace signály v kryptografických jednotkách. Správne riadenie multiplexorov a ovládacích vstupov, ktoré zabezpečuje tento hardwarový automat, robí činnosť kryptografických jednotiek zmysluplnou. Hardwarový automat je v RTL schéme na obrázku znázornený jako blok FSM. Jeho fyzická realizácia nieje s návrhovej stránky veľmi zaujímavá. Konečné automaty sa v hardwarová realizujú pomocou registra, ktorý obsahuje aktuálny stav automatu, kombinačnej logiky vyhodnocujúcej nasledujúci stav, a ďalšieho kombinačného obvodu, ktorý na základe vstupov a aktuálneho stavu tvorí hodnotu výstupov. Nakoľko výstupy automatu závisia od aktuálneho stavu, ale aj od stavu vstupov, je možné tento automat klasifikovať ako Mealyho. Diagram prechodov konečného automatu je zobrazený v obrazovej prílohe 3.

Stav *INIT* je počiatočným stavom, do ktorého sa uvedie automat po aktivácii globálneho Reset signálu. V tomto stave sa lineárne inicializuje vo všetkých jednotkách jedno stavové pole. Túto inicializáciu je potrebné vykonať vždy po resete. Ďalšie inicializácie budú vykonané už v priebehu výpočtu, v zmysle princípu double buffer, ktorý bol popísaný vyššie. V tomto stave zotrváva automat, dokedy nenadobudne čítač sbox_addr_cnt hodnotu 0xFF, teda dokedy sa nezinicializuje všetkých 255 bajtov pamäte Následne konečný automat prechádza do stavu *READY*. V tomto sú všetky vnútorné dátové štruktúry inicializované, a čaká sa na odštartovanie výpočtu. Automat zotrváva v stave, do okamihu, kedy sa zapíše do riadiaceho registra control_register hodnota ktorá má na najmenej významnom bite hodnotu log.1. Táto operácia odštartuje výpočet. Automat ďalej pokračuje do stavu *INIT_1a*. Stavy *INIT_1a*, *INIT_2a*, *INIT_3a* tvoria cyklus, v ktorom je implementovaný algoritmus inicializácie stavového poľa RC4. Každý z týchto stavov trvá 1 hodinový cyklus a implementuje jeden krok algoritmu, tak ako bol popísaný vyššie. Automat prestane cykliť ak čítač `sbox_addr_cnt` nadobudne hodnotu 0xFF, teda cyklí zakiaľ neinicializuje celé 255 Bajtové stavové pole. Automat potom prejde do stavu *KS_PREPa*. Tento implementuje prípravnú fázu, pred dešifrovaním testovacích dát. V priebehu tejto sa len nuluje obsah J-registra a inkrementuje hodnota čitača sbox_addr_cnt. S nasledujúcim hodinovým impulzom sa automat dostáva do cyklu, tvoreného

stavmi *KS_1a, KS_2a, KS_3a* a *KS_OUT_DATAa*. V priebehu jedného prechodu cyklom je vždy dešifrovaný jeden bajt testovacích dát. Tento je v stave *KS_OUT_DATAa* vzorkovaný blokom `CRC32`. Takýmto spôsobom sa priebežne ráta i CRC s dešifrovaných dát. V cykle s týchto štyroch stavov automat zotrváva dovtedy, dokedy má signál more_data hodnotu log.0. Tento signál je výstupom z komparátora porovnávacieho hodnotu registra data_size_register s hodnotou čítača, ktorý ukazuje na aktuálne dešifrovaný bajt v pamäť testovacích dát. Po dešifrovanú všetkých testovacích dát automat prechádza do ďalšej smyčky tvorenej stavmi *KS_ICV_1a*, *KS_ICV_2a*, *KS_ICV_3a* a *KS_OUT_ICVa*. Tento cyklus je veľmi podobný predchádzajúcemu, avšak na vstup dešifrujúceho xor člena je po bajtoch privádzaný kryptosúčet ICV. Dešifrované bajty kryptosúčtu sú na výstupe xor člena ukladané do registrov. V každej iterácii je dešifrovaný práve jeden bajt ICV, a automat sa dostane do nasledovného stavu po vykonaní štyroch iterácii Po vykonaní tohto cyklu sa automat dostane do stavu *CHECK_A*. V tomto stave je porovnaná hodnot CRC vyrátaná s dešifrovanej správy, s hodnotou CRC získanou dešifrovaním ICV. Ak obe hodnoty súhlasia s nasledujúcim hodinovým taktom bude aktuálna hodnota ICV zapísaná do Key Memorry.

Uvedený postup reprezentuje proces testu jedného kľúča Po vykonaní každého testu je v stave *CHECK_a* alebo *CHECK_b* testovaný stav čítača `keyid`. Ak stav zodpovedá najvyššej hodnote, znamená to, že bol už preskúmaný celý stavový priestor a automat prechádza do koncového stavu *FIN*, v ktorom ostane až do ďalšieho resetu celého obvodu. Ak nebol preskúmaný celý stavový priestor, inkrementuje sa hodnota keyid a automat vykonáva znova všetky uvedené kroky, avšak nad druhým bufferom. Tento bol predinicializovaný v priebehu prvého výpočtu.

V grafe reprezentujúcom automat sú obdĺžnikom vyznačené dve symetrické časti. Každá z nich vykonáva rovnaký výpočet avšak nad iným bufferom, teda inou časťou stavového poľa. Prechod oboma časťami sa cyklicky strieda až do preskúmania celého stavového priestoru. Vzhľadom na skutočnosť,že princíp práce druhej polovice automatu je symetrický k prvej, nemá zmysel ho znova detailne popisovať.

Dôležitou skutočnosťou, ktorá vyplýva z celkovej schémy uvedenej v obrazovej prílohe 2 je, že konečný automat označený ako FSM môže riadiť neobmedzené množstvo paralelne pracujúcich kryptografických jednotiek. Tieto totiž zdieľajú všetky riadiace ako aj vstupné dátové signály. Jediným signálom, ktorý nadobúda unikátnej hodnoty pre každú jednotku je unit_id.

3.5.5 Optimalizácia a škálovanie výkonu

Ďalšou fázou vývoja po navrhnutí a implementovaní riadiacej logiky a kryptografickej jednotky je správne stanoviť ich počet a pracovné podmienky, s prihliadnutím na maximálny výkon obvodu a fyzické obmedzenia ktoré na nás kladie zvolená cieľová technológia. Tento proces pozostáva z časti, v ktorej stanovíme maximálnu pracovnú frekvenciu celého obvodu a z časti v ktorej budeme určovať maximálny realizovateľný počet paralelných jednotiek. Vlastnosti výsledného obvodu by mali prijať pozitíva z oboch prístupov, teda mal by pracovať na maximálnej frekvencii a súčasne s vysokým stupňom paralelizmu.

Pri stanovení maximálnej hodinovej frekvencie je výrazným pomocníkom vývojové prostredie ISE. Toto dokáže v priebehu syntézy vyhodnotiť oneskorenia na všetkých kombinačných cestách a na základe toho odhadnúť maximálnu hodinovú frekvenciu. Hodnoty, ktoré poskytuje vývojové prostredie sú však len pomerne hrubým odhadom. Dôvodom nepresnosti je skutočnosť, že oneskorenia v technológii vznikajú aj na iných miestach, napríklad na vstupne/výstupných blokoch

(IOB) a v prepojovacom systéme. Vývojové prostredie sa snaží tieto faktory zohľadniť, ale vzhľadom na nedostatok informácii s ktorými pracuje v dobe syntézy, bývajú výsledkom často nereálne vysoké alebo nízke hodnoty. O skutočných oneskoreniach a schopnosti alebo neschopnosti navrhnutého obvodu pracovať na určenej frekvencii sa rozhodne až v procese place & route, kedy sú známe všetky technologické detaily.

Obvodu obsahujúcemu ovládaciu logiku a kryptografickú jednotku vychádza medzná frekvencia na 122MHz (Syntetizované pomocou ISE 10.1, optimization effort High). Tento kmitočet je však len ťažko použiteľný, pretože užívateľský programátor má k dispozícii len jeden zdroj synchronizácia a tým je signál PicoClk. Tento je zviazaný s frekvenciou PCMCIA zbernice a má frekvenciu 33MHz. Vyššie alebo nižšie synchronizačné kmitočty musia byť odvodené z tohto signálu pomocou DCGM modulu. DCGM je číslicový blok, ktorý dokáže na základe vonkajšej synchronizácie generovať hodinový signál ktorý je celočíselným násobkom alebo podielom vstupného kmitočtu. V architektúre Virtex 4 sú k dispozícii 4 takéto jednotky. Pre potreby našej aplikácie sme nastavili DCGM na 3x. Obvod s pracovnou frekvenciou 100Mhz sme úspešne vysyntetizovali a prakticky odskúšali

Pre stanovenie maximálneho počtu paralelných jednotiek poskytne základnú informáciu opäť prostredie ISE. Toto dokáže prostredníctvom *Module level utilization* zobraziť množstvo technologických zdrojov použitých jednotlivými blokmi. Bohužiaľ prostredie opäť nemá prostriedky pre monitorovanie využitia prepojovacieho systému. V prípade, že implementovaný design je náročný na prepojovacie dráhy, môže syntéza zlyhať, navzdory skutočnosti, že v technológii je ešte dostatok kombinačnej i sekvenčnej logiky. Vlastnosti a požiadavky výsledného obvodu sú známe až po ukončení place & route procesu. Na základe nám známych skutočností sme stanovili počet jednotiek na 128. Obvod so 128 paralelnými sme za použitia základnej frekvencie 33MHz vysyntetizovali a odskúšali. Takýto design spotreboval približne 73% logických blokov (slices) a 56% Block-RAM pamätí. Vzhľadom na skutočnosť, že počet jednotiek musí byť v navrhovanom designe mocninou dvojky, nieje ďalsie zvyšovanie výkonu možné.

Poslednou fázou stanovenia pracovných pomerov, je spojenie oboch prístupov. Výsledkom by mal byť obvod, ktorý obsahuje 128 jednotiek a pracuje na frekvencii 100MHz. Syntéza takéhoto obvodu však zlyhala, z dôvodu príliš vysokého oneskorenia v prepojovacom systéme. Pomocou nástroja TimingAn sme overili, že oneskorenia skutočne vznikajú vplyvom rozsiahleho prepojovacieho systému. Na základe získaných informácii sme stanovili novú pracovnú frekvenciu 66MHz. Na tejto pracovnej frekvencii sa obvod podarilo vysyntetizovať, i prakticky odskúšať. Výsledné riešenie je tak kompromisom medzi vysokou frekvenciou a stupňom paralelizmu.

3.5.6 Rozhranie obvodu

V tabuľke 3.1 sú prehľadne zhrnuté všetky vstupne výstupné adresy prostredníctvom ktorých je možné s obvodom komunikovať. Cieľom tejto state je poskytnúť detailnú vstupne/výstupnú špecifikáciu pre písanie ovládačov alebo ďalšieho aplikačného software.

Tabuľka 3.1: Rozhranie implementovaného obvodu

Adresa	Účel	Prístup	Poznámka
0x84000000	Riadiaci register	čítanie/ zápis	V súčastnosti implementované len rozpoznanie najnižšieho bitu, ktorý štartuje výpočet
0x83000000	ICV Register	čítanie/ zápis	32 Bitový register na uloženie kryptosúčtu.
0x82000000	IV Register	čítanie/ zápis	24 bitový register na uloženie inicializačného vektoru
0x81000000	Data size register	čítanie/ zápis	Množstvo platných dát v pamäti testovacích dát
0x80000000 až 0x800007FF	Pamäť testovacích dát	zápis	Pamäť na uloženie dátovej časti rámca odchyteného zo siete. Keďže PCMCIA zbernica neumožňuje nezarovnaný prístup, ukladá sa jeden bajt na každú štvrtú adresu. Celková kapacita je teda 2048B.
0xC3[ui]0000	Počet kľúčov v kryptografickej jednotke	čítanie	Hodnota [ui] symbolizuje číslo kryptografickej jednotky. Platné hodnoty na tomto mieste sú od 0x00 po 0x79. Teda od 0 do 127 dekadicky.
0xC2[ui]0000 až 0xC3[ui]000F	Pamäť potenciálnych kľúčov v jednotke	čítane	Hodnota [ui] symbolizuje číslo kryptografickej jednotky. Vzhľadom na nezarovnaný prístup k dátam sa číta z každej štvrtej adresy.

4 Analýza výkonu

Cieľom tejto kapitoly je odvodiť vzťah pre výkon implementovaného obvodu, vykonať teoretický výpočet výkonu pre konkrétnu sadu testovacích dát. Ďalej si kladieme za cieľ podložiť výpočty reálnymi meraniami a porovnať výkon obvodu s výkonom softwarovej implementácie na rôznych architektúrach. Ako jednotku pre meranie výkonu zavádzame počet kľúčov za sekundu. Táto jednotka je vhodnejšia ako priepustnosť systému v jednotkách MB/s, nakoľko neočakávame kontinuálne šifrovanie veľkého množstva dát. Jednotka počet kľúčov za sekundu (ďalej len Keys/s) poskytne reálnejší pohľad na priebeh útoku, a je možné z nej odvodiť predpokladanú dobu trvania výpočtu.
Vstupom pre odvodenie vzťahu pre výkon hardwarového obvodu je konečný automat, ktorý obvod riadi (obrazová príloha 3). Testovanie jedného kľúča pozostáva z inicializačnej fázy, šifrovania vlastných dát, dešifrovania ICV a kontroly ICV. Z konštrukcie automatu je zrejmé, že inicializácia vyžaduje tri hodinové takty pre každý bajt, dešifrovanie dát i ICV spotrebuje štyri takty na bajt. Overenie ICV je záležitosťou jedného hodinového taktu. Z uvedeného je možné odvodiť vzťah 4.1 pre výpočet počtu hodinových cyklov N potrebných pre overenie jedného kľúča

$$N = 3 \cdot 256 + 4 \cdot C + 4 \cdot 3 + 1 \; = \; 781 + 4 \cdot C \qquad (4.1)$$

Kde C je počet bajtov dátovej časti odchyteného rámca. Ak máme konštantný počet taktov na vyskúšanie jedného kľúča výkon obvodu ovplyvňuje len taktovacia frekvencia a počet paralelne pracujúcich jednotiek. Celkový výkon obvodu v počte kľúčov P za sekundu. možno vyjadriť vzťahom 4.2

$$P = \frac{J}{N \cdot (\frac{1}{f})} \; = \; \frac{J \cdot f}{N} \qquad (4.2)$$

Kde J je počet paralelných jednotiek, f je pracovná frekvencia obvodu a N je počet taktov na testovanie jedného kľúča (vzťah 4.2) Nami implementovaný obvod obsahujúci 128 jednotiek pracujúcich na frekvencii 66MHz sme testovali na dátovom rámci s dĺžkou 60B. Teoretický výkon obvodu pre zvolený rámec je 8 274 420 kľúčov za sekundu. Praktickým meraním sme získali hodnotu 8314996 kľúčov za sekundu. Odchýlka od teoretickej hodnoty je zrejme spôsobená len nepresným meraním časového úseku v ktorom obslužný program získava stav obvodu.

Pre porovnanie uvádzame v tabuľke 4.1 výkon počítačových zostáv a vybraných školských serverov pri softwarovom útoku. Pri každom testovanom systéme je uvedený typ procesoru, dĺžka trvania útoku a relatívny údaj v percentách dosiahnutého výkonu v porovnaní s implementovaným obvodom. Cieľom tejto tabuľky je poskytnúť prehľad o miere akcelerácie, ktorú môže vhodne navrhnutý obvod priniesť. Bohužiaľ, vzhľadom na rôznorodosť operačných systémov, nebolo možné zaručiť rovnaké testovacie podmienky pri teste na každom stroji. Skúsenosť však hovorí, že na rýchlosť výpočtu nemá použitý operačný systém výraznejší vplyv. Každý test bol realizovaný s rovnakým vzorkom vstupných dát a pred spustením testu bol projekt preložený nainštalovanou verziou GCC so zapnutou vysokou úrovňou optimalizácie -O3. Je dôležité podotknúť, že implementovaný softwarový útok je napísaný v jednom vlákne a neumožňuje efektívne využívať výpočetný výkon viacerých procesorov. Týmto sa síce do značnej miery vylúči vplyv plánovača operačného systému, výsledky testu však nehovoria o výkone systému ako celku,ale len o výkone jedného výpočetného jadra.

Z uvedeného je možné usúdiť, že jadro najvýkonnejšieho na fakulte dostupného procesoru dosahuje rádovo jednotky percent výkonu navrhnutého hardwarového obvodu.

Za porovnanie rozhodne stojí i parameter hovoriaci o spotrebe elektrickej energie a množstve odpadového tepla. Hodnoty tepelného výkonu procesoru Core2 Quad Q9550 je podľa katalógových listov spoločnosti Intel 95W [10]. Spotreba hradlového poľa je veľmi závislá na implementovanom designe, avšak v tomto prípade je možné hornú hranicu príkonu určiť s limitov, ktoré kladie špecifikácia zbernice PCMCIA. Tá stanovuje maximálny prúdový odber 1000mA pri napájacom napätí 3.3V [8]. Celkový maximálny príkon musí byť bezpodmienečne nižší ako 3.3W.

Tabuľka 4.1: Výsledky výkonnostnej analýzy

Architektúra	Pracovná frekvencia [MHz]	Operačný systém	Použitý prekladač	Absolútny výkon [kľúče/s]	Trvanie útoku [hh:mm:ss]	Relatívny výkon [%]	Relatívne zrýchlenie [-]
AMD Opteron 2216 na merlin.fit.vutbr.cz	2400	CentOS Linux 5.3	gcc 4.2.1	72932	04:05:22	0.88%	113.4x
Intel Pentium D930	3000	Windows XP, SP3	MinGW gcc 4.4.0	146666	02:02:01	1,77%	56,43x
Intel Xeon 5160 na eva.fit.vutbr.cz	3000	FreeBSD 7.1	gcc 4.2.1	180094	01:39:22	2,18%	45,94x
Intel Core2 Quad Q9550 na krok.fit.vutbr.cz	2830	Open Solaris 2008.11	gcc 3.4.3	298318	00:59:59	3,61%	27,73x
Virtex 4VFX60	66	-	ISE 10.1	8274420	00:02:09	100,00%	1x

Z vykonaných meraní výkonu je možné konštatovať, že navrhnutý obvod, implementovaný v FPGA Virtex 4VFX60 dosahuje 27 násobok výkonu najnovších procesorových jadier pri 28 násobne menšej spotrebe energie Prekážkou pri širšom nasadzovaní programovateľného hardware do bežnej praxe je jeho vysoká cena a relatívne vysoká obtiažnosť vývoja. Pre porovnanie uvádzam ceny z prvého kvartálu roka 2009 kedy bola cena procesora Core2 Quad Q9550 stanovená spoločnosťou Intel na 299USD, nákupná cena použitého kitu bola blízka 1000USD. Z tohto dôvodu sa komplexné FPGA prvky využívajú v súčastnosti hlavne ako testovacia platforma pre vývoj nového hardwaru a ASIC obvodov.

5 Záver

Na základe vypracovanej softwarovej a hardwarovej implementácie zvoleného šifrovacieho algoritmu a vykonaných výkonnostných porovnaní je možné konštatovať, že použitie aplikačne špecifického alebo programovateľného hardwaru môže priniesť výrazné urýchlenie priebehu výpočtu. V aplikáciách ktoré umožňujú nasadenie vysokej úrovne paralelizmu vynikajú možnosti hardwarovej implementácie výpočtu pred softwarovou obzvlášť výrazne. Nasadenie aplikačne špecifického hardwaru môže v mnohých prípadoch priniesť výraznú úsporu nákladov, rýchlejší beh aplikácie a nižšiu spotrebu elektrickej energie.

Z dlhodobého hľadiska možno povedať, že aplikačne špecifický a dokonca i programovateľný hardware si postupne hľadá cestu do bežných osobných počítačov, herných konzol a v neposlednom rade vysokorýchlostných sieťových aktívnych prvkov. V typickom osobnom počítači sa dnes už nachádza niekoľko desiatok obvodov, ktoré hardwarovo implementujú rôzne pomerne náročné úlohy a odľahčujú tak činnosť hlavnému procesoru CPU. Medzi tieto partia v prvom rade grafické akcelerátory, akcelerátory prehrávania videa vo vysokom rozlíšení, ale aj moderné sieťové chipsety. Medzi významné úspechy akcelerácie výpočtu pomocou aplikačne špecifického hardware možno považovať napríklad architektúru hernej konzoly PlayStation 3, ktorá kombinuje použitie viacúčelového procesory architektúry PowerPC s niekoľkými hardwarovými akcelerátormi nazývanými Synergistic Processing Unit. Tieto poskytujú v niektorých aplikáciach výrazne vyšší výkon ako architektúry založené na viacúčelových procesoroch, a súčasne sú zárukou vysokej škálovateľnosti a dobrého výkonnostného potenciálu platformy.

Rýchle prelomenie zvolenej šifry v tomto prípade nieje priamym dôkazom jej nesprávneho návrhu. Útok hrubou silou nevyužíva žiadne nedostatky ktoré boli v šifrovacom algoritme zistené. Úspešnosť útoku v tomto prípade demonštruje pokrok na poli výroby polovodičových prvkov a možnosť akcelerovať náročné algoritmy paralelnou implementáciou vo vhodnej hardwarovej platforme. V roku 1991 kedy bol šifrovací algoritmus WEP navrhovaný bol považovaný za bezpečný, a vytvorenie akéhokoľvek výpočetného prostriedku schopného úspešného útoku hrubou silou v rozumnom čase by bolo takmer nemysliteľné, alebo neúmerne drahé. Vzhľadom na rýchly vývoj v hardwarovej oblasti a pokroky v modernej kryptoanalýze je možné považovať každú šifru, ktorej nerozlušteľnosť nebola matematicky dokázaná považovať za zraniteľnú.

Cieľom tejto práce nebolo vyvolať diskusiu o zraniteľnosť súčasných šifrovacích algoritmov, ale uviesť do pozornosti možnosť s výhodou využiť hardwarovej akcelerácie pri realizovaní náročných algoritmov nad veľkým množstvom dát. Súčasný vývoj ukazuje, že v budúcnosti je vysoko pravdepodobné širšie nasadenie aplikačne špecifických hardwarových obvodov a do určitej miery i použitie programovateľného hardware. Hardwarová akcelerácia výpočtov by mohla priniesť významné zvýšenie priepustnosti budúcich počítačových systémov. Cieľom tejto práce bolo rozvinúť myšlienku hardwarovej realizácie algoritmov v bežnej praxi na príklade komplikovaného kryptografického výpočtu. Uverejnené riešenie problému by malo priniesť širšie pochopenie princípov použitých pri hardwarovej implementácii komplexných algoritmov ako aj výhod ktoré z neho plynú.

Literatúra

[1] Wikipedia:Cryptography [online] [cit. 1.3.2009]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Cryptography>

[2] Wikipedia:Cipher [online] [cit. 1.4.2009]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Cipher>

[3] Wikipedia:Stream Cipher [online] [cit. 1.4.2009]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Stream_cipher>

[4] Wikipedia:RC4 [online] [cit. 1.4.2009]. Dostupný z WWW: <http://en.wikipedia.org/wiki/RC4>

[5] Wikipedia:Birthday problem [online] [cit. 1.4.2009]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Birthday_paradox>

[6] Wikipedia:WEP [online] [cit. 1.4.2009]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Wired_Equivalent_Privacy>

[7] Wikipedia:Brute force attack [online] [cit. 20.4.2009]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Brute_force_attack>

[8] PCMCIA Assotiation: FAQ [online] [cit. 5.5.2009]. Dostupný z WWW: <http://www.pcmcia.org/faq.htm>

[9] Wikipedia:Field-programmable gate array [online] [cit. 8.5.2009]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Fpga>

[10] Intel: Core™2 Quad Processor Q9550 [online] [cit. 8.5.2009]. Dostupný z WWW: <http://ark.intel.com/cpu.aspx?groupId=33924>

[11] Aircrack-ng [online] [cit. 10.5.2009]. Dostupný z WWW: <http://www.aircrack-ng.org/doku.php?DokuWiki=1982537746e05483b14962b92182d1f3>

[12] PicoComputing: E14 Hardware manual [online] [cit. 10.5.2009]. Dostupný z WWW: <http://www.picocomputing.com/support/E-14.php>

[13] Wireshark: Documentation [online] [cit. 10.5.2009]. Dostupný z WWW: <http://www.wireshark.org/docs/>

[14] TCPdump: Documentation [online] [cit. 12.5.2009]. Dostupný z WWW: <http://www.tcpdump.org/>

[15] Michalis Galanis, Paris Kitsos, Giorgos Kostopoulos, Nicolas Sklavos, Costas Goutis:Comparison of the Hardware Implementation of Stream Ciphers [online] [cit. 14.5.2009]. Dostupný z WWW: <http://www.vlsi.ee.upatras.gr/~mgalanis/pubs/iajit.pdf>

Zoznam príloh