VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ODOLNOST AES PROTI ČASOVACÍ ANALÝZE

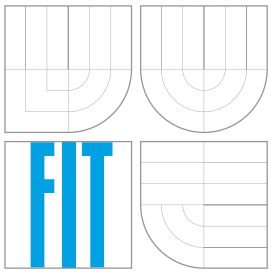
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

JURAJ ONDRUŠ

BRNO 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ODOLNOST AES PROTI ČASOVACÍ ANALÝZE

AES TIMING ANALYSIS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — JURAJ ONDRUŠ
AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. DANIEL CVRČEK, Ph.D.
SUPERVISOR

BRNO 2007

Abstrakt

Táto práca sa zaoberá rozborom možností použitia útoku časovou analýzou proti šifrovaciemu algoritmu AES. V práci je uvedená nutná teória implementácie algoritmu *Rijndael*, ktorý bol zvolený ako AES. Pre tento typ útoku je nutné poznať taktiež princíp vyrovnávacej pamäte cache v procesore a jej architektúru. V práci sú diskutované najzávažnejšie nedostatky AES a útoky, ktoré tieto nedostatky využívajú. Je tu uvedených niekoľko typov časových útokov. Podľa experimentov by tieto útoky mali byť úspešné na súčasne, v praxi bežne používané implementácie AES a architektúry procesorov. Ďalej sú analyzované možnosti použitia útoku na niekoľko typov implementácii AES. V závere je zhrnutý výsledok tejto práce a možné bezpečnostné opatrenia pri návrhu algoritmu proti tomuto typu útoku a taktiež návrhy na ďalší možný výskum.

Klíčová slova

AES, kryptoanalýza, šifra, Rijndael, analýza, runda, čas, cache, postranný kanál, časový útok.

Abstract

This thesis deals with timing analysis of the AES (Advanced Encryption Standard). The design of *Rijndael*, which is the AES algorithm, is described here. For the side channel attacks is necessary to know the principles of the cache memory in CPU and its architecture. In this thesis are involved major security problems of AES which can be used for successful attacks. Several different implementations of AES are discussed too. Several types of timing attaks are also described. According to the experimentations these attacks should be efficient to the most presently used AES implementations. Finally are described the results of this work, possible countermeasures against this attack and motions for the next research.

Keywords

AES, cryptoanalysis, cipher, timing attack, Rijndael, analysis, round, side-channel attack, cache.

Citace

Juraj Ondruš: Odolnost AES proti časovací analýze, diplomová práce, Brno, FIT VUT v Brně, 2007

Odolnost AES proti časovací analýze

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. D. Cvrčka, Ph.D.. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Juraj Ondruš
17. května 2007

Zadání diplomové práce/5777/2006/xondru06

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2006/2007

Zadání diplomové práce

Řešitel: **Ondruš Juraj**
Obor: Výpočetní technika a informatika
Téma: **Odolnost AES proti časovací analýze**
Kategorie: Bezpečnost

Pokyny:

1. Prostudujte pozadí architektury algoritmu AES (Advanced Encryption Standard) s důrazem na možnosti implementace.
2. Najděte několik existujících implementací algoritmu a vyzkoušejte možnosti časové analýzy, popište výsledky, identifikujte a analyzujte příp. zdroje informací použitelných pro časovací analýzu.
3. Vytvořte vlastní implementaci a pokuste se zabránit použití časovací analýzy.
4. Pokuste se o implementaci algoritmu pro čipovou kartu a porovnejte výsledky časovací analýzy z čipové karty s výsledky na počítači.
5. Diskutujte možnosti časovací analýzy pro algoritmus AES.

Literatura:

- D. J. Bernstein: Cache-timing attacks on AES. http://cr.yp.to/antiforgery/cachetiming-20050414.pdf

Při obhajobě semestrální části diplomového projektu je požadováno:

- Bez požadavků.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním paměťovém médiu (disketa, CD-ROM), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Cvrček Daniel, doc. Ing., Ph.D.**, UITS FIT VUT
Datum zadání: 1. listopadu 2006
Datum odevzdání: 24. ledna 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav inteligentních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. **Pan**

Jméno a příjmení: **Juraj Ondruš**
Id studenta: 22724
Bytem: Pelhřimovská 1193/3, 026 01 Dolný Kubín
Narozen: 12. 10. 1982, Dolný Kubín
(dále jen "autor")

a

2. **Vysoké učení technické v Brně**

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
diplomová práce

Název VŠKP: Odolnost AES proti časovací analýze
Vedoucí/školitel VŠKP: Cvrček Daniel, doc. Ing., Ph.D.
Ústav: Ústav inteligentních systémů
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:
tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☒ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

....

Nabyvatel Autor

Obsah

Kapitola 1

Úvod

Komunikácia medzi ľudmi prebieha rôznymi spôsobmi, či už osobne alebo pomocou správ. Dnes sa vo veľkej miere využívajú, okrem iných, počítačové technológie (počítač, internet, pamäťové médiá, e-mail, atd.), ktoré uľahčujú a hlavne urýchľujú prenos informácií. S rastúcim objemom takto prenášaných dát vznikajú aj problémy s ich utajením, resp. zašifrovaním tak, aby nemohlo dôjsť k ich zneužitiu.

K vyriešeniu tohto problému slúži veda nazývaná *kryptografia*. Jej úlohou je zabezpečit informácie proti zneužitiu ale zároveň musia byť zašifrované informácie spätne dešifrovateľé. Prvé známe algoritmické šifrovanie používal už rímsky cisár Caesar. Táto šifra je pre nás v dnešnej dobe síce jednoduchá, ale v tých časoch bola dostatočne bezpečná. V súčasnosti je však potreba oveľa zložitejších algoritmov pre šifrovanie, pretože útočníci majú k dispozícii veľa nástrojov a spôsobov ako šifru prelomiť. Postupom času a zdokonaľovaním šifrovania sa vyvinulo mnoho druhov šifier, ktoré sa líšia spôsobom priebehu šifrovania a tiež používaním kľúčov. Niektoré šifry dokázali prelomeniu odolávať i stovky rokov (napr. Vigenerova šifra), iné sa ukázali ako veľmi slabé a nepoužiteľné a bolo ich treba nahradiť inými, odolnejšími. Veľmi dôležitým míľnikom v dejinách kryptológie je určite obdobie prvej a druhej svetovej vojny. V tomto čase vzniklo niekoľko veľmi zaujímavých a prepracovaných šifier (napr. nemecký šifrovací stroj Enigma) ale hlavne v 70. rokoch 20. storočia vznikla potreba vytvárať kvalitné šifrovacie algoritmy.

V dnešnej dobe, spolu s vývojom technológií sa s kryptografiou alebo aj s kryptoanalýzou, mnoho z nás stretáva takmer na každom kroku a málokto si to uvedomuje. Potreba vyvíjať a analyzovať nové algoritmy je veľmi dôležitá, pretože tieto algoritmy často chránia naše súkromie a majetok. Preto je veľmi dôležité vytvoriť také šifry, ktoré budú dostatočne odolné voči útokom (kryptoanalýze) rôznych typov. Je potrebné dôkladne analyzovať každý takýto algoritmus, ale hlavne tie, ktoré sú určené pre široké použitie alebo dokonca ktoré sú vyhlásené ako medzinárodný štandard. V tejto práci boli zhodnotené reálne riziká útoku na algoritmus AES (Advanced Encryption Standard), možnosti ochrany pred útokom a možnosti využitia tohoto algoritmu, pretože pri výberovom konaní na tento algoritmus sa nevzali do úvahy všetky možné riziká a už krátko po uvedení tohoto štandardu sa objavili prvé úspešné útoky.

Je to veľmi ťažká a zodpovedná úloha, pretože existuje mnoho útokov (útok silou, útok otvorený text - šifrovaný text, časová analýza, frekvenčná analýza a iné). V súčasnosti zrejme neexistuje šifra používaná v bežnom živote, ktorá by nebola prelomiteľná niektorým typom kryptoanalýzy a rozvojom technológií sa riziko útoku zvyšuje.

Táto práca nadväzuje na predchádzajúci semestrálny projekt [5], v rámci tohoto projektu bolo riešené pozadie šifrovacieho algoritmu, ktorého základ tvorí algoritmus Rijndael.

1.1 Cieľ práce

Cieľom tejto práce je rozobrať pozadie algoritmu AES, ktorý bol v roku 2000 prijatý NIST (National Institute of Standards and Technology) za štandard na najbližších minimálne 25 rokov ako náhrada za terajší algoritmus DES.

Ďalším cieľom je porovnať a zhodnotiť rôzne už existujúce implementácie tohto algoritmu a určiť ich prínos z hľadiska bezpečnosti a odolnosti proti časovej analýze. Práca by mala ukázať štatistické vlastnosti jednotlivých implementácií, poprípade aj ich spoločné črty a taktiež možnosti ich ďalšieho využitia v budúcnosti.

Ďalej by táto práca mala predostrieť úvahu o možných útokoch na túto šifru v blízkej i ďalekej budúcnosti, rozobrať riziká, na ktoré treba pamätať pri návrhu nových implementácii a tiež možnosti ochrany pred útokmi časovou analýzou.

1.2 Rozdelenie práce

V druhej kapitole je uvedený stručný popis fungovania algoritmu Rijndael so zameraním na hlavné funkčné prvky.

V tretej kapitole je priblížená teória útoku časovou analýzou. Sú tu uvedené princípy takéhoto útoku a taktiež nutné znalosti, ktoré útočník musí poznať, aby jeho útok bol úspešný.

V štvrtej kapitole sú detailne popísané analyzované implementácie Rijndeal, ktorá implementácia je na čo zameraná, či využíva nejaké optimalizácie. Ďalej, v tejto kapitole sú pri každej implementácii vyhodnotené výsledky meraní a pozorovaní.

V piatej kapitole sú uvedené niektoré možnosti ochrany pred útokmi časovej analýzy a ich možné použitie na reálne implementácie, prípadne určité vylepšenia teórie tak, aby daná ochrana bola použiteľná v praxi.

V záverečnej kapitole sú zhrnuté získané výsledky a znalosti. Taktiež načrtnutie možného ďalšieho vývoja šifry a útokov do budúcnosti.

Kapitola 2

Návrh algoritmu AES

Táto kapitola pojednáva o návrhu a implementácii algoritmu Rijndael, ktorý bol vybraný ako AES. Tu sú detailnejšie popísané hlavné stavebné bloky algoritmu.

2.1 Špecifikácia algoritmu Rijndael

NIST v roku 1997 vyhlásil súťaž o nový algoritmus, ktorý mal nahradiť už bezpečnostne nedostačujúce algoritmy DES a 3DES. Prihlásilo sa 15 algoritmov. Po troch rokoch výberu uspel algoritmus Rijndael (ďalej v texte už len AES) od tvorcov Vincenta Rijmena a Joana Daemena. Podrobný popis implementácie je uvedený v ich prácach [1] a [2] . Tento algoritmus bol zvolený aj preto, lebo používa základnú dĺžku kľúča 128 bitov (maximálna dĺžka je 256 bitov), ktorá je dostatočne odolná proti útoku hrubou silou (tento útok je efektívne použiteľný do 80 bitového kľúča). Implementácia pre AES má malú odlišnosť oproti originálnemu algoritmu Rijndael a to tú, že pracuje s pevnou veľkosťou vstupného bloku, 128 bitov. V súčasnosti existuje niekoľko desiatok implementácii v najrôznejších programovacích jazykoch. V následujúcich riadkoch bude vysvetlený základný princíp a stavebné bloky, ktoré sú potrebné pre ktorúkoľvek implementáciu AES.

AES pozostáva zo štyroch hlavných transformácií otvoreného textu na šifru a naopak:

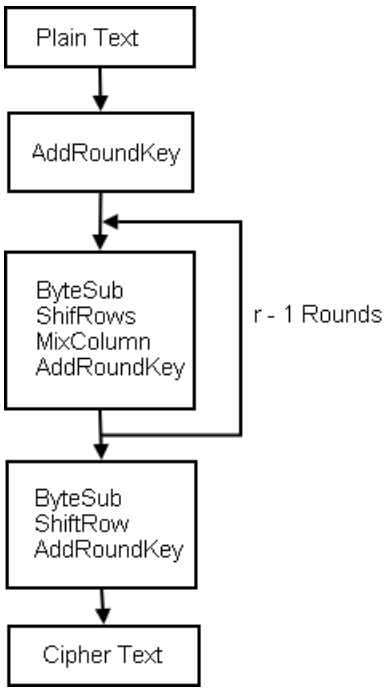
- **Round Key Addition**
- **Row Shifting**
- **Byte Substitution**
- **Column Mixing**

K správnemu a rýchlemu fungovaniu sú potrebné dve tabuľky (lookup tables), takzvané S-Boxy, navrhnuté NIST-om. Ich návrh je prísne utajovaný, pretože sú zásadné pre bezpečnosť. Niektoré implementácie, najmä z hľadiska optimalizácie rýchlosti výpočtu, rozdeľujú tieto tabuľky na štyri tabuľky T_0, T_1, T_2, T_3, ktoré sú už predom pevne dané a nedochádza tým k zdržaniu počítaním hodnôt potrebných pre (de)šifrovanie. Tieto tabuľky, či už sú implementované len dve alebo štyri, sú vo väčšine prípadov uložené do jednorozmerného poľa s počtom prvkov 256. AES je iteračná bloková šifra. Iteračná znamená, že vyššie uvedené transformácie sa opakujú (obrázok 2.1) v tzv. rundách. Počet rúnd N_r je funkciou závislou na veľkosti vstupných dát (ďalej len text) a kľúča, ako ukazuje tabuľka 2.1. Bloková šifra pracuje s určitou, pevnou veľkosťou bloku vstupných dát N_b. Ak veľkosť

N_k/N_b	**4**	**5**	**6**	**7**	**8**
4	10	11	12	13	14
5	11	11	12	13	14
6	12	12	12	13	14
7	13	13	13	13	14
8	14	14	14	14	14

Tabulka 2.1: Počet rúnd N_r ako funkcia N_b = dĺžka bloku/32 a N_k = dĺžka kľúča/32.

dát nie je násobkom čísla 16, tak posledný blok je doplnený dátami do potrebnej veľkosti. Po dešifrovaní musia byť tieto dáta odstránené. V našom prípade budeme medzivýsledok šifrovania bloku označovať ako *stav* X^i. Tento stav je rozdelený na šesťnásť jednotlivých bajtov x_0,...,x_{15} ako jednotlivé prvky poľa. Môže byť interpretovaný poľom so štyrmi riadkami a počet stĺpcov bude určený N_b (v prípade AES sa jedná o pevnú veľkosť $N_b = 4$). Kľúč je interpretovaný obdobne a jeho veľkosť bude označovaná N_k. Jednotlivé bajty textu i kľúča sú do matice mapované "po stĺpcoch", nie "po riadkoch".



Obrázek 2.1: Priebeh šifrovania.

2.1.1 Round Key Addition

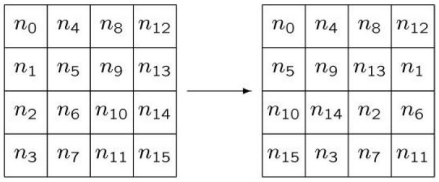
Pri tejto transformácii je každý bajt stavu X^i XORovaný príslušným bajtom expadnovaného kľúča. Pri každom ďalšom vykonávaní sa použije ďalších 16 bajtov expandovaného kľúča, čo znamená, že sa nikdy opakovane nepoužije tá istá časť kľúča. Výstupom je stav X^{i+1}. Pri dešifrovaní sa použije rovnaký postup, ale miesto vstupného textu sa použije šifrovaný text.

2.1.2 Byte Substitution

Táto transformácia je jediná nelineárna. Pracuje s každým bajtom stavu osobitne a nahrádza ho na základe indexu hodnotou z tabuľky S-Box (pri dešifrovaní sa použije inverzná S-Box tabuľka). Dešifrovanie prebieha obdobne.

2.1.3 Row Shifting

Pri tejto transformácii sa cyklicky posúvajú prvky riadku matice stavu. Prvý riadok (s indexom 0) zostáva bez zmeny. Druhý (index 1) sa pre veľkosť textu 128 bitov posunie o jeden bajt doľava. Tretí riadok o dva a štvrtý riadok o tri bajty ako ukazuje obrázok 4.11. Inverzná operácia posunu riadkov sa vykoná rovnako ako pri šifrovaní.



Obrázek 2.2: Posun riadkov.

2.1.4 Column Mixing

Pri tejto transformácii dochádza k násobeniu jednotlivých stĺpcov stavu s riadkami násobiacej matice (obrázok 2.3). Výsledky násobenia sú vzájomne XORované a výstupom sú nové bajty stĺpca stavu. Toto sa opakuje pre všetky stĺpce matice stavu. Následujúce rovnice názorne ukazujú výpočet prvého stĺpca novej matice.

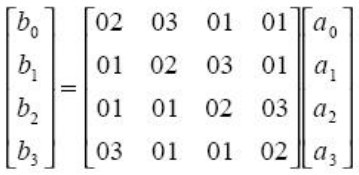
$$b_0 = (b_0 * 2) \oplus (b_1 * 3) \oplus (b_2 * 1) \oplus (b_3 * 1) \tag{2.1}$$
$$b_1 = (b_0 * 1) \oplus (b_1 * 2) \oplus (b_2 * 3) \oplus (b_3 * 1) \tag{2.2}$$
$$b_2 = (b_0 * 1) \oplus (b_1 * 1) \oplus (b_2 * 2) \oplus (b_3 * 3) \tag{2.3}$$
$$b_3 = (b_0 * 3) \oplus (b_1 * 1) \oplus (b_2 * 1) \oplus (b_3 * 2) \tag{2.4}$$

Matematickým základom tejto operácie je násobenie *Galois poľa* [3] [4] . V mnohých implementáciach je toto pole reprezetované dvoma tabuľkami (jedna pre šifrovanie a druhá, inverzná, pre dešifrovanie).

Pri dešifrovaní sa postup opakuje, ale násobenie prebieha s inou násobiacou maticou, viď tabuľku 2.2 .

0E	0B	0D	09
09	0E	0B	0D
0D	09	0E	0B
0B	0D	09	0E

Tabulka 2.2: Násobiaca matica pre dešifrovanie.



Obrázek 2.3: Násobenie stavu maticou (šifrovanie).

2.1.5 Expanzia kľúča

K šifrovaniu i dešifrovaniu je potrebný takzvaný *expandovaný kľúč* (round key), ktorý sa používa v každej runde v transformácii *Round Key Addition*. Aby bolo zaručené, že v každej runde sa použije odlišná časť tohoto kľúča, jeho veľkosť musí byť rovná N_b * (N_r + 1). Takže napríklad, pre veľkosť bloku 16 bajtov a kľúča 24 bajtov bude expandovaný kľúč 26 bajtov dlhý (208 bitov).

Prvá časť expandovaného kľúča W_0 je vždy zhodná s originálnym kľúčom (pôvodnou maticou). Výpočet kľúča W_i prebieha tak, že sa vezmú posledné 4 bajty kľúča W_{i-1} (posledný stĺpec matice). V ďalšom kroku sa tieto bajty posunú o jeden nahor (v literatúre označované ako *RotWord*). To znamená ze vrchný bajt bude na konci, druhý bude navrchu atd.

Následne sa vykoná substitúcia jednotlivých bajtov podľa S-Box tabuľky. V ďalšom kroku je potrebná, predom daná, tabuľka označovaná ako *Rcon*. Pri tejto operácii sa vybraný pozmenený stĺpec matice W_{i-1} XORuje s prvým stĺpcom tej istej matice a tento výsledok je následne XORovaný prvým stĺpcom tabuľky *Rcon*. Výsledok je prvým stĺpcom matice W_i, ktorý sa použije pri ďalšom *Round Key Addition*.

Druhý stĺpec získame operáciou XOR medzi už získaným prvým stĺpcom matice W_i a druhým stĺpcom z W_{i-1}. Obdobne získame tretí a štvrtý stĺpec kľúča W_i. Tento proces sa opakuje, pokiaľ nebude dĺžka expandovaného kľúča dostatočná.

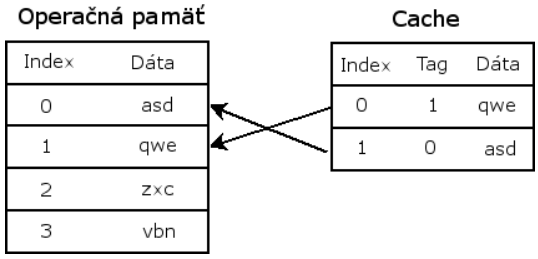
Kapitola 3

Útoky časovou analýzou

V tejto kapitole si priblížime podstatu *side channel* útokov, medzi ktoré patrí aj časová analýza (timing attack), ktorá bude rozobratá podrobnejšie. Priblížime si princíp fungovania pamäti chace, ktorú využívajú procesory (CPU) a ktorá je hlavným prvkom, potrebným k uskutočneniu útokov tohto typu. Ďalej tu bude uvedená teória niektorých úspešných útokov na AES.

3.1 Pamäť Cache

Pamäť cache [9] je malá (rádovo stovky kilobajtov), ale za to veľmi rýchla vyrovnávacia pamäť. Jej úlohou je dočasne uchovávať duplicitné informácie, ktoré sa tiež nachádzajú v hlavnej operačnej pamäti, ale kedže sú často využívané, sú v nej uložené kvôli rychlejšiemu prístupu procesoru k týmto dátam (obrázok 3.1). Týmto sa výrazne skracuje doba výpočtu. Samozejme, dáta ktoré sa prestanú používať sú zapísané do hlavnej pamäti, alebo sú "zahodené" rôznymi algoritmami na uvolňovanie cache pre iné potrebné dáta. Táto



Obrázek 3.1: Zjednodušený princíp pamäti cache.

rýchla pamäť sa dnes využíva takmer vo všetkých jednotkách CPU a to či už sa jedná o viacúčelové alebo jednoúčelové procesory.

Štruktúra pamäti cache môže byť rôzna, môže byť ich použitých aj viac (L1 a L2 cache). V najrozšírenejších 32-bitových architektúrach CPU (Intel, AMD) sa často využíva riadková štruktúra pamäti. Riadok ma určitú dĺžku a môžu byť v ňom uložené aj rôzne dáta.

Pre ešte väčšie urýchlenie doby prístupu CPU k potrebným dátam, sa začali využívať pamäti s dvoma (troma) úrovňami. Jedna je menšia (L1) a druhá je väčšia (L2). Keď CPU nenájde žiadanú informáciu v L1, tak sa odkáže na L2. Presné postupy sa líšia od jednotlivých typov a výrobcov CPU, každý používa svoj algoritmus.

Keď sa žiadané dáta nachádzajú v niektorej cache pamäti, hovoríme o *zásahu* (cache hit, collision). Ak tieto dáta nie sú prístupné v cache, hovoríme o *výpadku* (cache miss). Podrobný popis spôsobu ošetrenia zásahu a výpadku je uvedený v [10] .

Práve to, či sa jedná o zásah alebo nie, spôsobuje rôzne dlhé doby výpočtu a toto je miesto pre časovú analýzu, kde útočník môže podľa časov zistiť o aké operácie sa jedná a ktoré bajty sú práve spracovávané a tak prelomiť šifru.

3.2 Side Channel útoky

Side-channel útoky [7] [8] v poslednej dobe zažívajú veľký rozmach a vývoj smerom dopredu. Tento typ útokov sa nezameriava na šifru ako takú, nevyužíva slabých teoreticko-matematických miest v algoritme, ale útočí na konkrétnu implementáciu používanú na systémoch, ktoré poskytujú dodatočné a vedľajšie informácie o priebehu výpočtu [7]. V čase výberového konania algoritmu pre AES boli tieto útoky považované ako použiteľné len na určité aritmeticko-logické operácie, ktorým sa však mnoho algoritmov vyhýbalo. Rozvojom hardwarového vybavenia počítačov, zvyšujúcim sa výkonom a ľahkou dostupnosťou takýchto zariadení sa tieto útoky stali skutočnosťou pre široký okruh šifrovacích algoritmov. Vybavenie potrebné k útoku už nepresahuje náklady rádovo v tisícoch dolárov a je dostatočne výpočetne výkonné.

V minulosti bol šifrovací proces vnímaný tak, že do zariadenia vstupuje text, ktorý sa v zariadení zašifruje a výstupom je šifrovaný text a naopak [7]. K tomuto boli využívané útoky, ktoré používali silné matematické techniky a vedomosti, napríklad diferenciálna alebo lineárna kryptoanalýza.

Najnovšie algoritmy sú už však voči takýmto útokom vo veľkej miere odolné. Preto bolo treba nájsť iné cesty na prelomenie šifier, vrátane AES. Informácie potrebné k zisteniu kľúča však môžme hľadať aj inde, ako len v teoreticko-matematických nedostatkoch algoritmu. A to v jeho implementácii a vo vlastnostiach systému, na ktorom sa daná implementácia algoritmu vykonáva. Najviac informácií je možné získať sledovaním jednotky CPU, ktorá vykonáva jednotlivé výpočty algoritmu. Takto môžeme získať informácie napríklad o dobe trvania výpočtu, o energetickej náročnosti alebo o elektromagnetickom či tepelnom žiarení [7]. V ďalšej časti tejto práce sa detailnejšie zameriame práve na hodnotu a použiteľnosť získaných informácií o dobe trvania výpočtu.

K vykonaniu útoku takéhoto typu sú však vo väčšine prípadov potrebné výborné znalosti z oblasti matematiky, štatistiky, hardware a systému, na ktorý sa má útočiť. Celkové zostrojenie útoku je zložitá úloha.

3.3 Časová analýza

Princíp tohto útoku spočíva v tom, že útočník sa snaží získať čo najviac informácii popisujúcich dĺžku spracovania jednotlivých častí algoritmu [6] [7] na základe použitého CPU a cache. Ako je známe, rôzne operácie môžu mať odlišnú zložitosť a preto ich CPU spracováva rôzne dlhú dobu, resp. je potrebný rozdielny počet cyklov (napríklad násobenie a delenie). Taktiež doba spracovávania záleží na tom, či sú potrebné dáta uložené v pamäti cache alebo sa musia načítať z hlavnej operačnej pamäti.

Tieto informácie môžu viesť až k odhaleniu celého tajného kľúča. Je však potrebný určitý počet vzoriek meraní, ktorý je uvedený v tabuľke 3.1. Ich počet závisý od typu stratégie útoku [11] [12] [13] [14] a taktiež od architektúry cache.

Útok	Počet vzoriek	Typ vzorky	Úspešnosť
Bernstein [11]	$2^{27.5}$	Plaintext	celý kľúč
Tsunoo	2^{26}	Plaintext	celý kľúč
First round attack [13]	$2^{14.58}$	Plaintext	60 bitov kľúča
Final round attack [13]	2^{15}	Ciphertext	celý kľúč
Expanded final round attack [13]	2^{13}	Ciphertext	celý kľúč

Tabulka 3.1: Počet vzoriek potrebných k uspešnosti útoku.

K úspešnemu útoku pomocou časovej analýzy je potrebná podrobná znalosť systému, ktorý sa má stať obeťou a to nie len z hardwarovej stránky, ale taktiež je potrebné vedieť aké iné aplikácie v dobe šifrovania využívajú cache a CPU. Je to dôležité z toho hľadiska, že tieto aplikácie môžu vyžadovať prednostné prerušenia CPU a taktiež aj načítanie nových dát do cache, čo má potom vplyv na časový priebeh výpočtu šifry.

V dnešnej dobe sú však už pamäte cache dostatočne veľké a dokážu uchovávať množstvo dát z rôznych aplikácii a v bežnej praxi sa nepoužívajú aplikácie tak náročné na výpočet, aby zabrali celú cache, čím by dochádzalo k odtraňovaniu dát iných aplikácii, v tomto prípade šifrovacieho algoritmu. Treba mať taktiež na zreteli to, že z času na čas sa v pamäti cache vykonávajú tzv. čistiace algoritmy (napríklad LRU - Least Recently Used alebo Garbage Collector), ktorých účelom je uvolniť pamäť zapísaním málo používaných dát do operačnej pamäte.

3.3.1 Útoky s využitím informácii z cache

Princíp útoku na AES pomocou informácií získaných z cache a CPU je popísaný vyššie. Avšak aj tieto útoky sa dajú rozdeliť do rôznych kategórii podľa použitej taktiky [12] [13] [11] a úspešnosti. Následujúce riadky popisujú postup útoku na typ implementácie AES, ktorý využíva pri výpočte štyri tabuľky. Tento princíp útokov je však podľa všetkého použiteľný aj na iné typy implementácii, avšak návrh a vytvorenie algoritmu sa môže v praxi veľmi líšiť a môže byť rôzne náročný, v závislosti na type AES.

Všetky útoky využívajú to, že v dnešnej dobe sa veľké množstvo dát ukladá do cache, ktorá je dostatočne veľká a neprebiehajú tým pádom dodatočné migrácie dát, ktoré by mohli vnášať do merania chyby. Komplikácie pri meraniach môže spôsobiť to, že cache neuchováva jednotlivé bajty ale skupiny bajtov - skupiny bajtov s rovnakou adresou. Toto je však zavislé na konkrétnej architektúre použitého procesoru.

Typická dĺžka riadku pre dnešné najpoužívanejšie procesory je 32 až 128 bajtov pre Intel Pentium III alebo AMD Athlon. Takže ak má riadok veľkosť 32 bajtov, veľkosť lookup tabuľky AES sú 4 bajty, to znamená, že sa na jeden riadok zmestí 8 tabuliek [13]. Z toho vyplýva, že osem načítaní bude smerovaných do jedného riadku a tým budú trvať rovnakú dobu. Z toho je možné predpokladať to, že doba načítania dvoch rozdielnych hodnôt z jedného riadku bude kratšia ako doba načítania z rôznych riadkov, pri dostatočne veľkom množstve náhodných šifrovaní s rovnakým kľúčom.

Najdôležitejšia informácia pre uskutočnenie útoku je tá, ktorá ukazuje na tzv. kolíziu v cache (cache collision) alebo tiež označované ako zásah. Táto kolízia nastane v situácii keď pre dva bajty stavu x_i, x_j platí $\langle x_i \rangle = \langle x_j \rangle$. Z toho plynie, že pre dostatočne veľké množstvo vzoriek bude doba šifrovania kratšia pre $\langle x_i \rangle = \langle x_j \rangle$ ako pre $\langle x_i \rangle \neq \langle x_j \rangle$ [13].

3.3.2 First round útoky

Tento typ útoku časovou analýzou je založený na štatistických výsledkoch a pozorovaniach cieľového systému útočníkom, pri ktorom bola úspešne zistená veľká časť kľúča [13] alebo aj celý kľúč [11]. Prvá runda AES je zraniteľná preto, lebo bajt stavu je možné vyjadriť jednoduchou rovnicou $x_i^0 = p_i \oplus k_i$. V príprave k útoku bolo nazbierané množstvo časových

$$x_0^0 = p_0 \oplus k_0, x_1^0 = p_1 \oplus k_1, \ldots, x_{15}^0 = p_{15} \oplus k_{15}$$

Obrázek 3.2: Závislosť jednotlivých bajtov vstupu a bajtoch kľúča.

vzoriek x_i, pre určenie priemerného času spracovania. V ďalšom kroku bolo zhromaždené množstvo časových údajov z cieľového počítača o bajte textu p_i. Rozdiel medzi týmito priemernými časmi, by mala byť práve indícia k získaniu kľúča.

Pred útokom sa predpokladalo, že načítaním premennej poľa s indexom $T_0[p_0 \oplus k_0]$ na začiatku počítania AES je možné predpokladať, že doba načítania tejto hodnoty je závislá na jej indexe v poli. Z tohto prepokladu je možné určiť hodnotu $\langle x_i^0 \rangle = \langle k_0 \rangle \oplus \langle p_0 \rangle$. Tento predpoklad sa ukázal ako správny a vedúci k želanému výsledku. Na základe toho, že S-Box tabuľky sa rozdeľujú na ďalšie štyri, vznikajú štyri skupiny bajtov, ktoré hodnoty sú indexami v jednej tabuľke (obrázok 3.3). Ako bolo spomínané skôr, ak platí $\langle x_i^0 \rangle = \langle x_j^0 \rangle$,

$$\begin{aligned} X^{i+1} = \{&T_0[x_0^i] \oplus T_1[x_5^i] \oplus T_2[x_{10}^i] \oplus T_3[x_{15}^i] \oplus \{k_0^i, k_1^i, k_2^i, k_3^i\}, \\ &T_0[x_4^i] \oplus T_1[x_9^i] \oplus T_2[x_{14}^i] \oplus T_3[x_3^i] \oplus \{k_4^i, k_5^i, k_6^i, k_7^i\}, \\ &T_0[x_8^i] \oplus T_1[x_{13}^i] \oplus T_2[x_2^i] \oplus T_3[x_7^i] \oplus \{k_8^i, k_9^i, k_{10}^i, k_{11}^i\}, \\ &T_0[x_{12}^i] \oplus T_1[x_1^i] \oplus T_2[x_6^i] \oplus T_3[x_{11}^i] \oplus \{k_{12}^i, k_{13}^i, k_{14}^i, k_{15}^i\}\}. \end{aligned}$$

Obrázek 3.3: Jedna runda šifrovania, kde X^{i+1} je výstupný stav a x_i^0 sú bajty stavu.

tak $\langle k_i \rangle \oplus \langle p_i \rangle = \langle k_j \rangle \oplus \langle p_j \rangle$ a po úprave $\langle p_i \rangle \oplus \langle p_j \rangle = \langle k_i \rangle \oplus \langle k_j \rangle$. Na základe tohoto je možné zostaviť tabuľku s nameranými priemernými časmi a z nej pomocou štatistických metód vybrať vhodné hodnoty a zostaviť sústavy rovníc (obrázok 3.4), v ktorých budú vystupovať premenné z rovnakej skupiny bajtov. Je tu však jeden problém a ten je v pamäti cache. Keďže dáta v tejto pamäti sú ukladané po riadkoch, nemôžeme o rovnosti $x_i^0 = x_j^0$ prehlásiť, že tieto dva stavy bajtu sú identické, ale iba to, že sú na jednom riadku cache a prístupová doba k nim je rovnaká. Toto budeme označovať práve $\langle\ x_i^0\ \rangle = \langle\ x_j^0\ \rangle$

Kôli tomu sa útokom uvedeným v [13] nedá zistiť celý kľúč, v priemere pre 128 bitový kľúč sa podarí odhaliť 60 bitov. Čo je však stále málo pre praktické použitie. Vylepšenie

$$\langle k_0 \rangle \oplus \langle k_4 \rangle = \Delta_1, \langle k_0 \rangle \oplus \langle k_8 \rangle = \Delta_2, \langle k_0 \rangle \oplus \langle k_{12} \rangle = \Delta_3,$$

$$\langle k_4 \rangle \oplus \langle k_8 \rangle = \Delta_4, \langle k_4 \rangle \oplus \langle k_8 \rangle = \Delta_5, \langle k_8 \rangle \oplus \langle k_{12} \rangle = \Delta_6$$

Obrázek 3.4: Sústava získaných rovníc, kde $\Delta = p_i \oplus p_j$.

alebo rozšírenie tohoto útoku o analýzu dvoch rúnd (two round attack) [12] je už viac

úspešné a je možné doplniť chýbajúce bity kľúča (oproti first round útoku [13]). Útok sa však stáva zložitejším a náročnejším, ale na druhú stranu sa zníži počet potrebných vzoriek z $2^{27.5}$ na približne 2^{15} vzoriek, čo je výrazný posun a taktiež nie je potreba mať referenčný počítač.

V útoku [11] boli využité vedomosti o tom, že v prvej runde šifrovania sú prvky v tabuľke zviazané práve s jedným bajtom textu p_i a práve s jedným bajtom kľúča k_i, ktoré potom tvoria jeden bajt stavu x_i^0 (obrázok 3.2).

K vykonaniu útoku bola potrebná detailná znalosť systému, na ktorý sa útočilo a relatívne veľké množstvo vzoriek. Bol potrebný identický referenčný počítač. Útok prebehol tak, že bola na sieti sledovaná šifrovaná komunikácia medzi vzdialeným serverom a cieľovým počítačom za použitia *OpenSSL* v implementácii AES. Na systémoch bol použitý procesor Pentium III, ktorý bol v tej dobe najrozšírenejší. Dôvodom úspešnosti nie sú nedostatky v *OpenSSL* komunikácii, ale v samotnom algoritme AES. Je veľmi ťažké navrhnúť časovo konštantnú implementáciu pre bežne používané viacúčelové procesory.

Počas útoku bolo potrebných $2^{27.5}$ vzoriek, aby bolo možné odhaliť celý kľúč. Zozbieranie a roztriedenie takého množstva dát môže trvať rádovo desiatky až stovky minút, v závislosti na použitom systéme. Pri útoku boli z útočníkovho počítača zasielané pakety rôznej dĺžky na server, ktorý bol obeťou. Postupným znižovaním dĺžky paketu a miernym nárastom počtu vzoriek, bolo odhaľovaných čoraz viac bajtov kľúča a pre zvyšné bajty sa znižoval počet možných hodnôt. Nakoniec bolo nazbierané dostatočné množstvo údajov k dopočítaniu chýbajúcich bajtov kľúča. Bol to jeden z prvých útokov na šifru AES, ktorý bol úspešný vďaka použitiu vedľajších kanálov v cache. Poukázal na vážne nedostatky tejto šifry a bol odrazovým mostíkom pre ďalšie útoky podobného typu s rôznymi vylepšeniami.

3.3.3 Final Round Attack

K útoku je potrebný známy zašifrovaný text k zisteniu kľúča. Tento útok je postavený na výnimočnosti poslednej rundy algoritmu Rijndael. Posledná runda je výnimočná v tom, že vynecháva operáciu *Column Mixing* a použije sa jediná S-Box tabuľka. Priebeh tejto rundy je vyjadrený rovnicou na obrázku 3.5 [13]. Znova môžeme predpokladať kolíziu dvoch bajtov

$$\begin{aligned} C = \{&T_4[x_0^{10}] \oplus k_0^{10}, T_4[x_5^{10}] \oplus k_1^{10}, T_4[x_{10}^{10}] \oplus k_2^{10}, T_4[x_{15}^{10}] \oplus k_3^{10}, \\ &T_4[x_4^{10}] \oplus k_4^{10}, T_4[x_9^{10}] \oplus k_5^{10}, T_4[x_{14}^{10}] \oplus k_6^{10}, T_4[x_3^{10}] \oplus k_7^{10}, \\ &T_4[x_8^{10}] \oplus k_8^{10}, T_4[x_{13}^{10}] \oplus k_9^{10}, T_4[x_2^{10}] \oplus k_{10}^{10}, T_4[x_7^{10}] \oplus k_{11}^{10}, \\ &T_4[x_{12}^{10}] \oplus k_{12}^{10}, T_4[x_1^{10}] \oplus k_{13}^{10}, T_4[x_6^{10}] \oplus k_{14}^{10}, T_4[x_{11}^{10}] \oplus k_{15}^{10}\}. \end{aligned}$$

Obrázek 3.5: Posledná runda bez *Column Mixing* kde C je výstupných 16 bajtov šifry.

$x_u^{10} = x_w^{10}$ v cache, z ktorej odvodíme následujúce rovnice:

$$T_4[x_u^{10}] = T_4[x_w^{10}] = \alpha \tag{3.1}$$

$$c_i = k_i^{10} \oplus T_4[x_u^{10}] \tag{3.2}$$

$$c_i = k_i^{10} \oplus \alpha \tag{3.3}$$

$$c_j = k_j^{10} \oplus T_4[x_w^{10}] \tag{3.4}$$

$$c_j = k_j^{10} \oplus \alpha \tag{3.5}$$

$$c_i \oplus c_j = k_i^{10} \oplus k_j^{10} \tag{3.6}$$

Keď nedôjde ku kolízii, čiže $x_u^{10} \neq x_w^{10}$, tak je potrebné uvažovať dve hodnoty α a β ako výsledok načítania z tabuľky. Môžeme teda napísať, že

$$\alpha \oplus \beta = \gamma = c_i \oplus c_j \oplus k_i^{10} \oplus k_j^{10} \tag{3.7}$$

Tento vzťah je možné napísať na základe nelinearity S-Box tabuľky, čo ma byť jej hlavná výhoda, ale tu sa ukazuje, že práve nelinearita dáva priestor na útok [13]. K úspešnému útoku sú však ešte potrebné ďalšie kroky. Keďže α a β sú výsledkom vráteným z S-Box tabuľky, tak hodnota γ nám nezaručuje presnú hodnotu indexov použitých na výber hodnoty z S-Box.

Cieľom útoku je zaznamenávať časové údaje o dobe trvania výpočtu pre každú hodnotu náhodného textu $\Delta = c_i \oplus c_j$. Pre každú pozorovanú dvojicu šifra - čas sa hodnota doby trvania výpočtu uloží do tabuľky $t[i,j,\Delta]$. Hľadá sa jedna hodnota $\Delta'_{i,j}$ taká, že pre každé i, j bude $t[i,j,\Delta'_{i,j}] < \bar{t}$, kde $\bar{t}$ je priemerný čas šifrovania všetkých dvojíc. Tieto hodnoty $\Delta'_{i,j}$ môžu byť práve tie hodnoty na určenie $\Delta i, j = k_i \oplus k_j$, ktoré spôsobujú kratšiu dobu šifrovania. Týmto odpadá problém dát na jednom riadku v cache.

Tieto hodnoty môžu umožniť útočníkovi odhadnúť posledných 16 bajtov expandovaného kľúča. Keďže proces expandovania kľúča je plne reverzibilný ak je známych 16 po sebe idúcich bajtov expandovaného kľúča, je možné sa takto dopracovať až k originálnemu kľúču. Pre každý takýto odhad program útočníka vykoná reverzný expand kľúča a porovná výsledok s dvojicou text - šifra. Tabuľka 3.2 ukazuje priemerný počet potrebných dvojíc šifra - čas, *(C,t)*, k odhaleniu celého 128 bitového kľúča [13].

CPU	L1 cache	L2 cache
Pentium III 1.0 GHz	2^{16}	2^{15}
Pentium IV Xeon 3.2 GHz	$2^{19.9}$	2^{16}
UltraSparc III+ 0.9 GHz	$2^{18.7}$	2^{15}

Tabulka 3.2: Počet vzoriek potrebných k uspešnosti útoku.

Nevýhodnou útoku je to, že sa do úvahy berú iba kolízie v cache zapríčinené načítaním hodnoty z tabuľky s rovnakým indexom. Ale ak je v riadku cache viac položiek tabuľky, tak sa nebude jednať o kolíziu z dôvodu rovnakého indexu, ale len o načítanie hodnôt rôznych indexov z jedného riadku cache. Útok je veľmi rýchly, odhalenie kľúča trvá rádovo v sekundách, avšak nie je stopercentne účinný.

Vylepšenie útoku prináša využitie algoritmov umelej inteligencie (*belief propagation a local optimalization search, simulované žíhanie*). Ich použitím síce narastá náročnosť útoku, jeho vykonanie trvá rádovo desiatky minút, ale za to je veľmi úspešný, nie je potrebný referenčný počítač a útočník ho môže vykonávať takpovediac "offline" po tom, ako nazbiera dostatočné množstvo vzoriek.

Oba druhy útokov je možné použiť aj v "dešifrovacom móde", to znamená že je známy text a neznáma šifra a kľúč. Útok je nezávislý na platforme, na ktorej prebieha šifrovanie a v súčasnosti už takmer aj na použitom CPU, keďže architektúra cache je u najpoužívanejších CPU veľmi podobná. Predpokladá sa, že pred začatím útoku je cache prázdna a CPU

vykonával inú výpočtovú činnosť. Problémy môžu nastať pri zväčšení veľkosti cache. Taktiež nepriaznivý vplyv na výkon útoku môžu mať najnovšie, viac jadrové, architektúry CPU, ktoré spolupracujú s cache inak. Môže byť na nich ťažšie rozlíšiteľné či sa jedná o zásah alebo nie.

Praktické využitie takéhoto napadnutia je ale veľmi otázne, i keď je odolnejší voči šumom oproti iným útokom. Preto je dobré pri návrhu myslieť na to, že i keď len experimentálne, ale predsa je reálne vykonateľný.

3.3.4 Two Round Attack

Nevýhodou útoku na prvú rundu je to, že tento útok iba odhaľuje vzťahy medzi jednotlivými bajtmi kľúča, čo je zapríčinené architektúrou cache [13]. Výber hodnôt z tabuliek v druhej runde je závislý najmenej na štyroch bajtoch kľúča. Táto vlastnosť výrazne posiluje možnosti útoku na prvú rundu a vedie k odhaleniu celého kľúča.

V prvej runde je možné zistiť vzájomné závislosti bajtov, ktoré sú indexmi v jednej tabuľke. V druhej runde je možné zistiť závislosť aspoň jedného bajtu zo všetkých štyroch skupín tabuliek. Uvažujme preto napríklad bajt x_0^1, ktorý je použitý ako index v tabuľke T_0:

$$x_0^1 = 2 \bullet S(p_0 \oplus k_0) \oplus 3 \bullet S(p_5 \oplus k_5) \oplus S(p_{10} \oplus k_{10}) \oplus S(p_{15} \oplus k_{15}) \oplus S(k_{13} \oplus 0x01) \oplus k_0 \quad (3.8)$$

Tento bajt je najľahšie napadnuteľný, pretože práve tento bajt je závislý len od piatich bajtov kľúča. Okrem toho, $k_5 \oplus k_{13}$ je vzťah, ktorý je známy z prvej rundy. Aplikovaním princípu z prvej rundy a odhadom hodnoty bajtu kľúča k_{13} je potom celkom jednoduché zistiť bajt x_0^1 pri známom otvorenom texte. Z prvej rundy sú taktiež známe vzťahy pre bajty $\langle x_0^0 \rangle$, $\langle x_4^0 \rangle$, $\langle x_8^0 \rangle$ a $\langle x_{12}^0 \rangle$ a všetky tieto bajty patria k indexom tabuľky T_0. Ak sa $\langle x_0^1 \rangle$ rovná niektorému z týchto vzťahov, tak počas šifrovania muselo dôjsť k zásahu. Táto udalosť umožní lepší odhad hodnoty daného bajtu kľúča, na základe sledovania časových hodnôt pre jednotlivé dvojice text - čas, ktoré boli namerané počas pozorovania niekoľkých šifrovaní [13]. Aj napriek miernemu vylepšeniu je stále potrebné prácne dopočítať aspoň 33 bitov kľúča. Priemerne je potrebných 2^{16} vzoriek dvojíc pre správny odhad kľúča. Časová zložitosť tohto útoku je však ešte o niečo vyššia, čo robí tento útok v niektorých situáciach nepoužiteľným.

Kapitola 4

Implementácie AES

V tejto kapitole sa zoznámime s niekoľkými implementáciami algoritmu AES. Rozoberieme si ich podrobnejšie, či využívajú nejaké optimalizácie, výhody a nevýhody. Takisto si porovnáme implemetáciu AES v jazyku C a v jazyku Java. Popisované implementácie sú k dispozícii na priloženom kompaktnom disku.

4.1 Meranie času výpočtu

K presnému meraniu času bol použitý hlavičkový súbor *pctimer.h*, ktorý je taktiež priložený na CD. Tento súbor bol použitý pri meraniach časov implementácií v jazyku C/C++. Časy boli merané s presnosťou na nanosekundy. Väčšia presnosť nie je nutná, kedže procesor pracuje s frekvenciou 1.8GHz, čo znamená, že jeden takt trvá približne 5^{-10}s. V celkovom meraní teda dochádzalo k veľmi malému zaokrúhľovaniu, ktoré nemohlo nejak výrazne ovplyvniť výsledky merania.

Pre implementáciu v jazyku Java bola využitá metóda merania v balíčku *java.lang.System* a to *System.nanoTime()*.

Merania časov prebehli na systéme Windows XP SP2, AMD Sempron 1,8 GHz, 448 MB RAM. Ďalšie merania prebehli na servery Merlin, patriaci FIT VUT.

4.2 Rijndael

Táto implemetácia bola zostavená v prostredi *MinGW 2.05* s prekladačom gcc 3.3.1 pod systémom Windows XP SP2 a taktiež v prostredí *NetBeans 5.5* s JDK 1.6.0 a JRE v6. Táto implementácia sa dá považovať za referenčnú implementáciu algoritmu Rijndael.

Je zostavená podľa teoretického popisu [1] a tak, ako ju pôvodne zostrojili jej tvorcovia. Nie sú v nej použité žiadne sofistikované optimalizácie, či už využitia pamäte alebo spotreby času. Je napísaná intuitívne, jednoducho, ako názorná ukážka. Keďže sa jedná o implementáciu Rijndael, pracuje s dĺžkou bloku dát od 128 bitov až do 256 bitov. Jednoduchou úpravou je však možné stanoviť pevnú dĺžku bloku na 128 bitov. Stav bloku je navrhnutý ako dvojrozmerné pole so štyrmi riadkami a počtom stĺpcov odpovedajúcim dĺžke bloku (pre 128 bitový blok sú to štyri stĺpce).

V implementácii sú definované dva dátové typy, *word8* ako unsigned char a *word32* ako unsigned long int.

Počty rúnd pre jednotlivé dĺžky bloku a kľúča sú uložené v poli *numrounds*, takisto v poli je aj uložený počet posunov riadkov *stavu*.

Využívajú sa dve tabuľky S-Box, jedna pre šifrovanie S a druhá pre dešifrovanie Si, vytvorené ako jednorozmerné pole s 256 prvkami. Podobne sú navrhnuté tabuľky *Logtable, Alogtable* používané vo funkcii *MixColumns* pre násobenie Galois poľom.

Výstup nameraných časov je presmerovaný do súborov pomocou funkcie (v jazyku Java je to metóda) *outprint()*. Jednotlivé súbory sú pomenované podľa jednotlivých operácií, z ktorých časy sú v ňom uložené. Formát výstupného súboru je XML. Nie je to však úplne korektný XML súbor. Sú v ňom podľa značiek iba rozdelené časy podľa dĺžky bloku a kľúča a treba ručne doplniť hlavičku definovanú pre XML súbory (<?xml version="1.0" encoding="ISO-8859-1"?>) a začiatočnú a koncovú značku (v tomto prípade je to <times> a </times>). Pre transformáciu z XML formátu bol vytvorený jednoduchý formátovací súbor *tab.xsl*, ktorý je priložený na CD. Pri meraniach iba jednej dĺžky vstupu a kľúča, je výstup do súboru bez XML značiek, keďže nie je potreba rozlišovať dĺžky a aj kvôli zmenšniu veľkosti súboru.

Funkcie vykonávajúce jednotlivé operácie algoritmu, ktoré sú popísané v kapitole druhej, sú pomenované tak, aby ich názvy vystihovali danú operáciu:

- ***AddRoundKey(word8 a[4][MAXBC],word8 rk[4][MAXBC], int x)*** - parametrami tejto funkcie sú premenné reprezentujúce stav, príslušnú časť expandovaného kľúča a pomocná premenná určujúca smer (šifrovanie, dešifrovanie)
- ***Substitution(word8 a[4][MAXBC], word8 box[256], int x)*** - parametre: stav, S-Box, pomocná premenná
- ***ShiftRows(word8 a[4][MAXBC], word8 d)*** - parametre: stav, pomocná premenná
- ***MixColumns(word8 a[4][MAXBC])*** - parametre: stav
- ***InvMixColumns(word8 a[4][MAXBC])*** - parametre: stav
- ***rijndaelKeySched(word8 k[4][MAXKC],word8 W[MAXRD+1][4][MAXBC])*** - parametre: šifrovací kľúč, kľúč rundy

Tieto funkcie sú potom volané pri šifrovaní a dešifrovaní s príslušnými hodnotami parametrov. Po spustení programu sa vygeneruje vstupný text pre šifrovanie, ktorý sa v ďalšom šifrovaní mení iba v jednom náhodnom bajte. Kľúč sa generuje pseudonáhodne. Ich dĺžka zaleží od premenných *BC a KC*, ktoré sa menia v cykloch od hodnoty 4 pre 128 bitov, po hodnotu 8 pre 256 bitov dĺžky (v tomto prípade je hodnota stanovená na 4 u oboch premenných).

4.2.1 Vyhodnotenie meraní a pozorovaní

Merania boli zamerané hlavne na dĺžku vstupu 16 bajtov. V priebehu na seba nadväzujúcich šifrovaní sa menil len jeden bajt vstupu. Základné merania boli zamerané hlavne na dĺžku kľúča 128 bitov. Niektoré merania však prebehli pre všetky možné dĺžky kľúča, aby sa overilo chovanie algoritmu s meniacou sa veľkosťou kľúča.

Meraním a pozorovaním nameraných hodnôt bolo zistené, že vyššie spomínané informácie je naozaj možné získať z pamäti cache. Útočník pri dostatočne dlhom pozorovaní môže nazbierať potrebné množstvo časových údajov o priebehu (de)šifrovania, nutných k zostaveniu úspešného útoku, ktorý odhalí tajný šifrovací kľúč.

Merané boli jednotlivé dielčie časti výpočtu šifrovania, to znamená napríklad, že bola meraná každá jedna operácia $\oplus$ vo funkcii *AddRoundKey*. Bližším skúmaním nameraných hodnôt, bolo zistené, že mnohé hodnoty sa v rámci jedného procesu šifrovania (dešifrovania) často opakujú. Čo dokazuje tvrdenie, že doba výpočtu je kratšia keď $\langle x_i \rangle = \langle x_j \rangle$ ako pre $\langle x_i \rangle \neq \langle x_j \rangle$. Taktiež to zmanemá, že hodnoty, i keď s rôznymi indexami v poli, ležia na jednom riadku v cache. Boli pozorované aj doby potrebné na výpočet prvej, resp. poslednej rundy, na ktoré sa útoky najviac zameriavajú.

Napríklad pre dĺžku bloku i kľúča 128 bitov vo funkcii *AddRoundKey*, pri jej vykonávaní v prvej runde boli zistené *zásahy* v cache. Nasledujúce rovnice ukazujú niekoľko zásahov

$$\langle x_1^0 \rangle = \langle x_2^0 \rangle = \langle x_4^0 \rangle = \langle x_6^0 \rangle = \langle x_7^0 \rangle \tag{4.1}$$

$$\langle x_3^0 \rangle = \langle x_{15}^0 \rangle \tag{4.2}$$

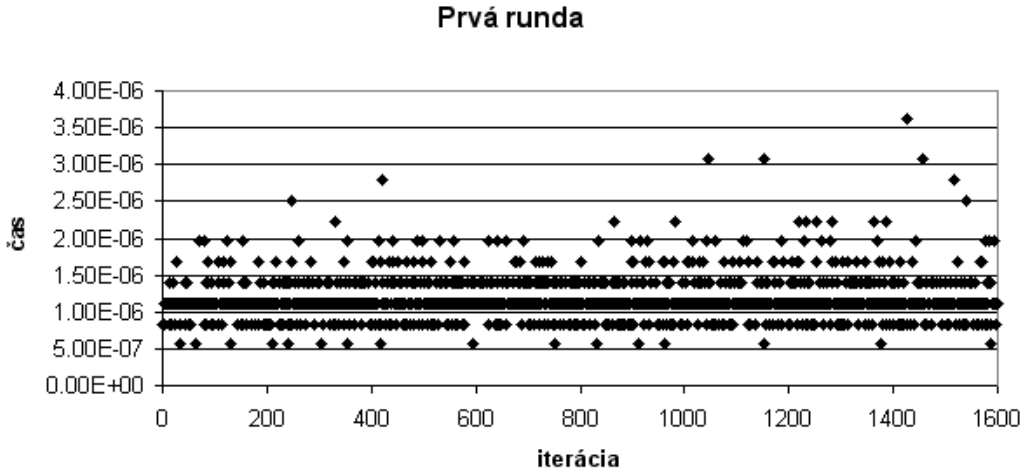
Taktiež došlo k zásahom aj v priebehu poslednej rundy šifrovania, niektoré z nich vyjadrujú nasledujúce rovnice

$$\langle x_0^{10} \rangle = \langle x_2^{10} \rangle = \langle x_3^{10} \rangle = \langle x_4^{10} \rangle = \langle x_6^{10} \rangle \tag{4.3}$$

$$\langle x_5^{10} \rangle = \langle x_8^{10} \rangle \tag{4.4}$$

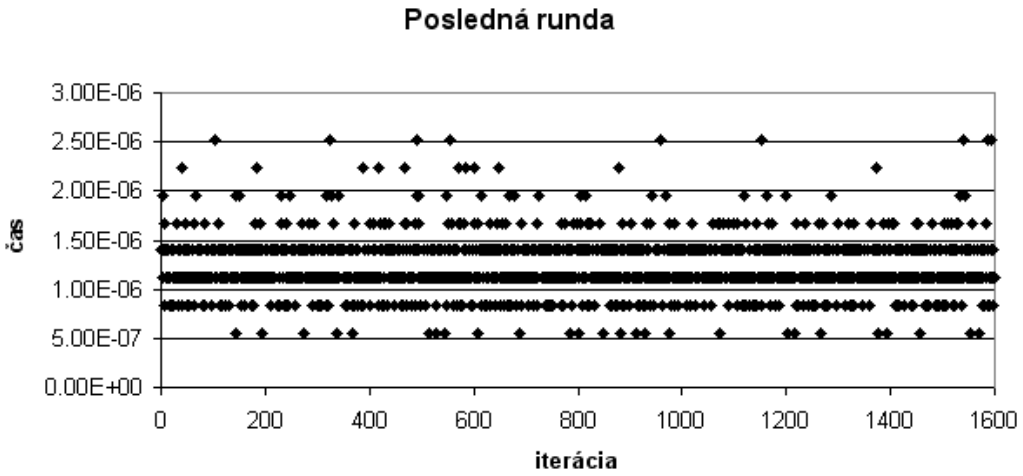
Z týchto rovníc vyplýva aj to, že v cache je uložené dostatočné množstvo dát, aby dochádzalo k častým zásahom, ktoré sú využiteľné útočníkom.

Po spustení šifrovania stokrát po sebe objavilo množstvo *zásahov*, ktoré spájali bajty stavu s jedným riadkom v cache. Takéto množstvo zásahov je spôsobené tým, že šifrovania na seba kontinuálne nadväzovali a tým padom v cache ostávali načítané tabuľky. Pre útok je teda výhodnejšie, keď je pred šifrovaním cache prázdna, aby sa do nej mohlo načítať čo najviac dát, ktoré sa aktuálne používajú a aby tam nezostávali dáta z predchádzajúceho šifrovania. Časový priebeh spracovávania bajtov stavu v prvej, resp. poslednej runde v operácii *AddRoundKey* ukazujú obrázky 4.1, resp. 4.2. Údaje z prvej rundy šifrovania môžu byť pre



Obrázek 4.1: Časový priebeh *AddRoundKey* v prvej runde pri sto šifrovaniach so 128 bitovým kľúčom.

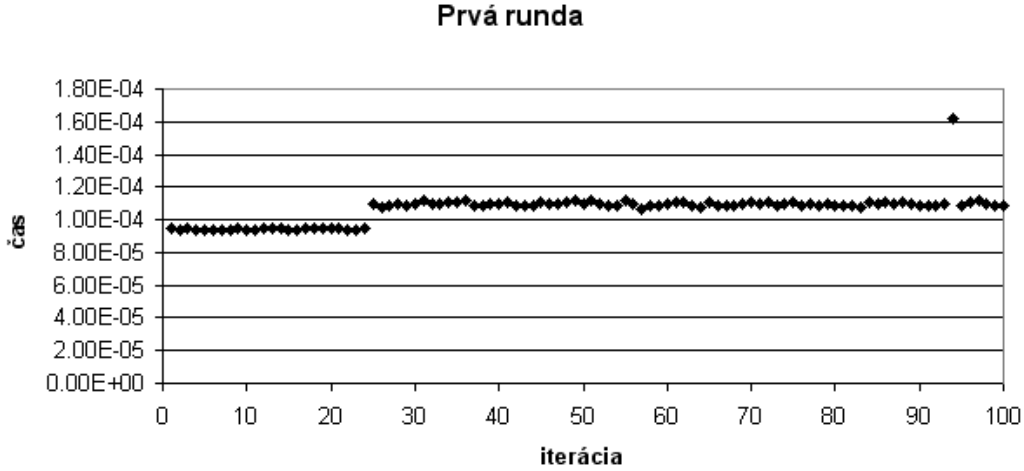
útočníka veľmi užitočné a podľa uvedených rovností je možné odvodiť rovnice popísané v princípe *first round* útoku vyššie. Pri pokusných meraniach sa počet zásahov v prvej runde



Obrázek 4.2: Časový priebeh *AddRoundKey* v poslednej runde pri sto šifrovaniach so 128 bitovým kľúčom.

pohyboval medzi 2 až 4. Útočník v prvej runde potrebuje získať $2^{14.58}$ vzoriek, tak pri priemernom počte zásahov 3 v prvej runde, potrebuje sledovať len približne 2^{13} šifrovaní, čo je pri dnešnej výkonnosti počítačov otázka niekoľkých minút až desiatok minút.

Treba však počítať s tým, že tieto rovnice nevyjadrujú identickosť bajtov ale to, že tieto hodnoty ležia na spoločnom riadku v cache. Počet zásahov počas celého šifrovania so 128 bitovým je veľký. K zásahom dochádza takmer pravidelne. Priebeh šifrovaní pre ostatné dĺžky kľúča vykazuje rovnaké správanie. Jednotlivé časy sú pre dlhšie kľúče o trochu väčšie, ale k zásahom dochádza v približne rovnakom množstve. K takýmto *zásahom* do cache dochádza aj pri všetkých ostatných operáciach algoritmu a tiež aj pri dešifrovaní.

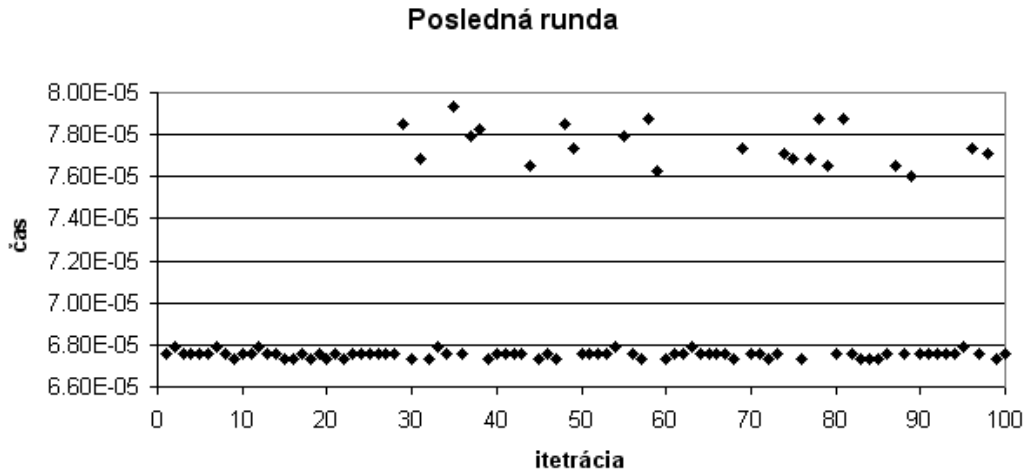


Obrázek 4.3: Časový priebeh prvej rundy pri sto šifrovaniach so 128 bitovým kľúčom.

Takisto sledovaním časov, ktoré boli potrebné na výpočet stavu po prvej, resp. poslednej runde ukazujú, že napriek tomu, že v jednej runde sa používa pomerne veľké množstvo

rôznych dát a operácii, mnoho z nich sa nachádza v cache na jednom riadku a tým pádom sú pozorované rovnaké doby trvania, ako ukazujú obrázky 4.3 a 4.4. Aj toto dokazuje to, že dnešné pamäte cache sú dostatočne veľké na uchovávanie potrebných dát k výpočtu. Pre útok je to vhodné, pretože nedochádza k migrácii dát medzi cache a operačnou pamäťou, prípadne v opačnom smere. Na základe toho je potom chybných meraní minimum a útočník môže zostaviť potrebné množstvo relevantných vzoriek.

Dôležité je tiež to, že v rámci jednej dĺžky kľúča je priebeh takmer konštatný, dochádza len k malým odchýlkam, prípadne aj k väčším, ktoré sú však spôsobené prerušeniami CPU. Tieto prerušenia sa po viacnásobnom opakovaní menili v rôznych iteráciach, to útočníkovi trochu komplikuje úlohu, pretože s tým musí počítať, že nedokáže predpovedať, kedy presne nastane nejaké prioritné prerušenie spracovávania a tým pádom aj k dlhšej dobe výpočtu.



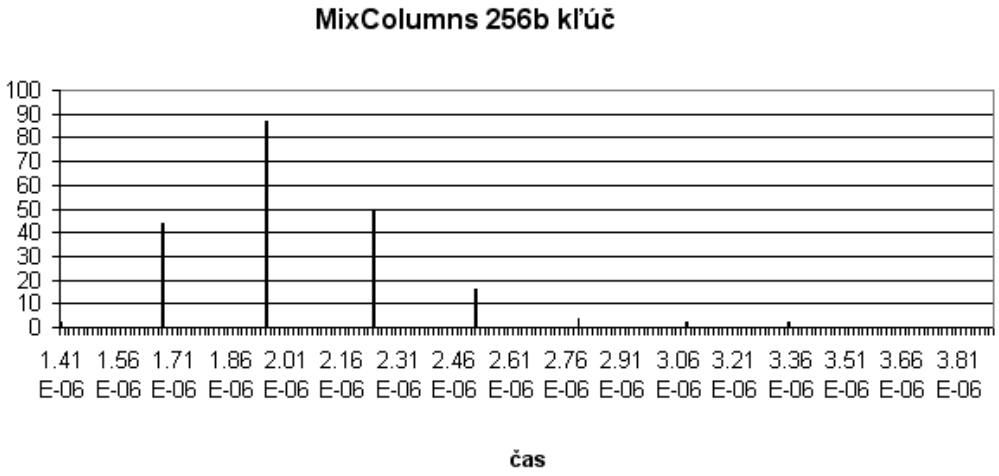
Obrázek 4.4: Časový priebeh poslednej rundy pri sto šifrovaniach so 128 bitovým kľúčom.

Obrázok 4.5 názorne ukazuje počet výskytov (četnosť) rovnakého času v operácii *MixColumns* pre dĺžku kľúča 256 bitov. Táto operácia bola zvolená na ukážku, že aj v ďalších častiach algoritmu dochádza k podobnému javu.

Z týchto meraní je vidieť, že k zásahom dochádza aj v tejto implementácii algoritmu AES. Vhodným naštudovaním detailného priebehu šifrovania, je možné použiť teoretické znalosti o útoku v prvej, resp. poslednej runde a vytvoriť tak útok na túto implementáciu. Problémom môže byť rozdelenie jednotlivých bajtov stavu do skupín tak, ako je to možné pri implementácii so štyrmi tabuľkami. V tomto prípade, keď sa používa jedna tabuľka pre šifrovanie a jedna tabuľka pre dešifrovanie, je veľká pravdepodobnosť že hľadané dáta budú na jednom riadku v cache a dôjde k zásahu aj bez toho, aby boli tieto bajty v nejakom vzájomnom vzťahu. Je síce možné k útoku princípy popísané vyššie, avšak samotný algoritmus útoku musí byť upravený presne ma mieru tohto typu AES, aby dokázal rozoznávať bajty, ktoré majú a ktoré nemajú medzi sebou nejaký vzťah.

4.3 Rijndael Java

Implementácia v jazyku Java sa významovo a obsahovo nelíši od predchádzajúcej implementácie. Líši sa samozrejme len syntaxou príkazov. Cieľom bolo ukázať, že odolnosť voči



Obrázek 4.5: Výskyt rovnakých časov v operácii *MixColumns* pri šifrovaní s 256 bitovým kľúčom.

útoku nezáleží na programovacom jazyku, prípadne prekladači daného jazyka. Je známe, ze prekladač jazyka Java je v určitom zmysle pomalší a využíva rôzne optimalizácie, ale hlavne funguje na inom princípe ako prekladač jazyka C.

4.3.1 Vyhodnotenie merania a pozorovania

Meranie časov bolo vykonávané pomocou už spomínanej metódy *System.nanoTime()*. Táto metóda meria čas procesoru s presnosťou na nanosekundy.

Napriek tomu, že táto implementácia je sémanticky zhodná s predchádzajúcou implementáciou v jazyku C, pozorovanie jej chovania vykazuje značné odlišnosti. Tieto odlišnosti sú spôsobené hlavne prekladačmi daných jazykov. Každý z prekladačov pristupuje k prekladu programu odlišne a tým pádom je odlišné aj následné uloženie dát v cache ako napríklad vyhradenie miesta pre jednotlivé premenné daných dátových typov.

Pri pozorovaní časov v prvej runde algoritmu je síce mnoho časov rovnakých, nedochádza však k tak častým zásahom ako v implementácii v jazyku C (obrázok 4.6). Následujúce rovnice vyjadrujú niekoľko cache zásahov v operácii *AddRoundKey*

$$\langle x_1^0 \rangle = \langle x_{15}^0 \rangle \tag{4.5}$$

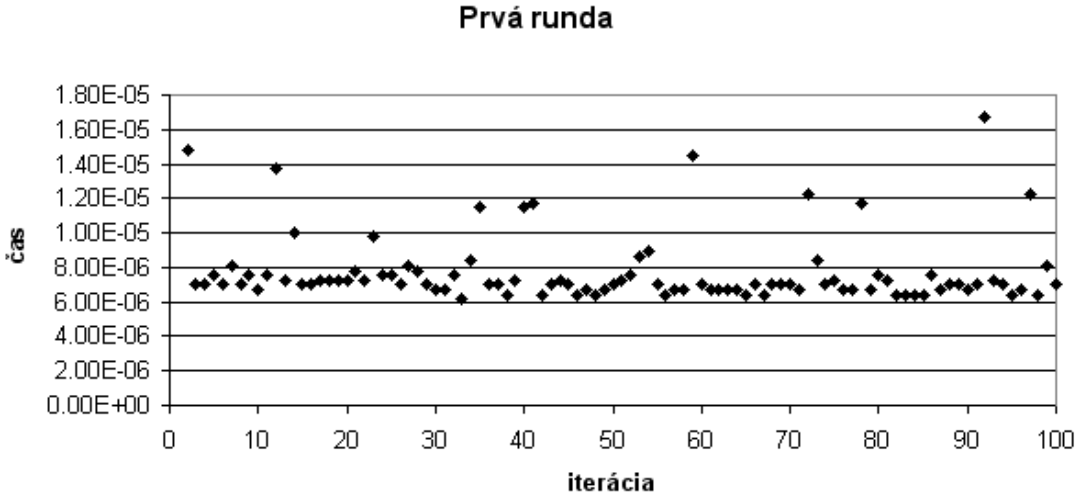
$$\langle x_2^0 \rangle = \langle x_7^0 \rangle = \langle x_{14}^0 \rangle \tag{4.6}$$

$$\langle x_3^0 \rangle = \langle x_4^0 \rangle \tag{4.7}$$

$$\langle x_8^0 \rangle = \langle x_9^0 \rangle \tag{4.8}$$

V poslednej runde bol pozorovaný podobný jav, nedochádzalo k častým zásahom, či už pred alebo aj po vyčistení cache ako je vidieť na obrázku 4.7.

Obrázok 4.8 vyjadruje četnosť výskytov rovnakých časových intervalov v operácii *MixColumns*. Pre ostatné operácie algoritmu má graf četnosti výskytov veľmi podobný priebeh, mierne sa líši len počet rovnakých časov.



Obrázek 4.6: Časový priebeh prvej rundy v 100 šifrovaniach pre 128 bitový kľúč.

Pri pozorovaní priebehu *AddRoundKey* v prvej a poslednej runde bolo opäť pozorované to, že pri stonásobnom opakovaní šifrovania so 16 bajtovým vstupom, ktorý sa menil len v jednom bajte a pri nemeniacej sa dĺžke kľúča, je časový priebeh veľmi vyrovnaný a líši sa len málo (obrázok 4.9). Iba prvá iterácia prvej, resp. poslednej rundy sa veľmi výrazne líši od celkového priebehu, čo je zrejme spôsobené tým, že sa práve v tomto okamihu mapujú do cache potrebné dáta. Merania prebehli s prázdnou cache pred šifrovaním, to znamená, že v nej predtým neboli žiadne dáta súvisiace so šifovaním. Toto je pre útočníka dôležitá informácia, pretože ak je cache zaplnená dátami z iných aplikácii, tak dochádza k neželanej migrácii dát.

Z týchto pozorovaní vyplýva, že implementácia v jazyku Java môže byť odolnejšia proti časovému útoku. Nie však vďaka tomu žeby nedochádzalo k potrebným časovým zásahom v cache, ale kvôli tomu, že týchto zásahov je podstatne menej ako v implementácii jazyka C, čo môže mať za následok väčšie množstvo nameraných vzoriek potrebných k odhaleniu kľúča, čiže k pozorovaniu väčšieho množstva šifrovaní. Čo však pri dnešnej výkonnosti počítačov nemusí byť neprekonateľný problém.

4.4 Optimalizovaná implementácia AES

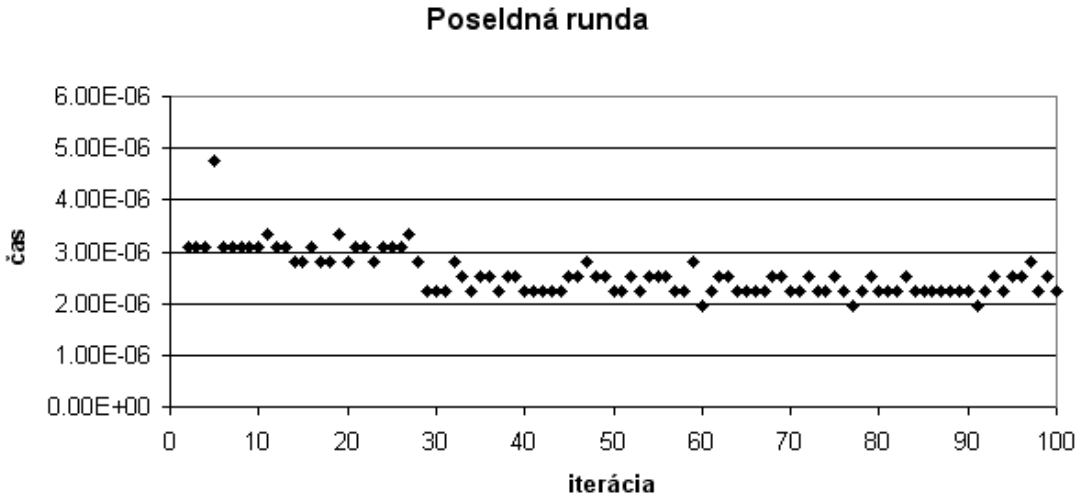
Hlavným cieľom tejto implementácie [15] je optimalizovať využitie pamäťových prostriedkov. V druhom rade sa dbalo na časovú optimalizáciu.

Pôvodne je táto implementácia navrhnutá pre 8-bitový procesor čipovej karty. V tomto prípade bola navrhnutá pre klasické 32-bitové procesory v jazyku C.

Stav má veľkost 16 bajtov a obsahuje práve (de)šifrované dáta. Oproti iným implementáciam nie je stav uchovávaný v matici ale v jednorozmernom poli, stĺpec za stĺpcom (obrázok 4.10).

V programe je definovaný dátový typ *word8* ako *unsigned char* a dátový typ *word32* ako *unsigned long int*.

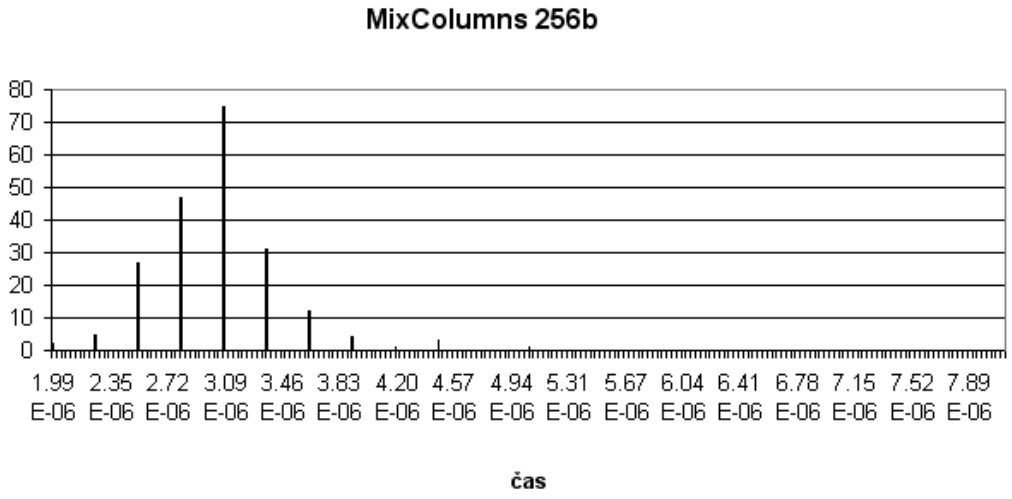
Priebeh algoritmu je implementovaný do následujúcich funkcií



Obrázek 4.7: Časový priebeh poslednej rundy v 100 šifrovaniach pre 128 bitový kľúč.

- ***encryptData(word8 *data, word8 *key, word32 data_length)***, kde premenná data je ukazateľ na text ktorý ma byť šifrovaný, premenná key je ukazateľ na kľúč a premenná data_length definuje dĺžku bloku
- ***decryptData(word8 *data, word8 *key, word32 data_length)***, kde premenné majú rovnaký význam ako pri encryptData s rozdielom, že data ukazujú na šifru určenú k dešifrovaniu
- ***addRoundKeyAndSubstituteBytes(word8 *state, word8 *round_key, int x)***, kde state je ukazateľ na stav, ktorý má byť XORovaný s ukazateľom na round_key, premenná x určuje to, či sa jedná o šifrovanie alebo naopak, dešifrovanie
- ***shiftRows(word8 *state, int x)***, kde state je ukazateľ na aktuálny stav a x určuje smer priebehu algoritmu
- ***mixColumns(word8 *state)***, kde state je ukazateľ na stav, ktorý má byť upravený
- ***inverseMixColumns(word8 *state)***, obdobne ako mixColumns
- ***keyExpansion(word8 *round_key, word8 *round_constant)***, kde round_key je ukazateľ na kľúč danej rundy a round_constant je ukazateľ na konštantu kľúča potrebnú k výpočtu expandovaného kľúča
- ***inverseKeyExpansion(word8 *round_key, word8 *round_constant)***, parametre sú rovnakého významu ako v keyExpansion

Ako je možné vidieť s predchádzajúceho popisu, v implementácii chýba samostatná operácia *Byte Substitution*. Táto funkcia je spojená s funkciou *Round Key Addition*. Tieto dve operácie je možé spojiť vďaka tomu, že obe pracujú s jednotlivými bajtmi stavu, tak je možné ich vykonať spoločne v dvoch krokoch. V prvom kroku sa aktuálny stav XORuje s odpovedajúcou hodnotou expandovaného kľúča a následne v druhom kroku je výsledok tejto operácie použitý ako index pre vyhľadanie hodnoty v tabuľke S-Box.



Obrázek 4.8: Počet rovnakých časových hodnôt v priebehu celého šifrovania 256 bitovým kľúčom v *MixColumns*.

Operácia *ShiftRows* je implementovaná trochu zložitejšie. Keďže dáta nie sú uložené po riadkoch, je potrebné aby táto operácia prechádzala aktuálny stav z ľava doprava a posúvala bajty stavu, ktoré sú na nesprávnom mieste, na to miesto kam patria (obrázok 4.11). K tomu slúži pomocná funkcia *swap*, ktorá pracuje s ukazateľmi na jednotlivé bajty stavu.

Na obrázku 4.12 je znázornený princíp operácie *iverseShiftRows*, ktorá sa len málo líši od predchádzajúcej.

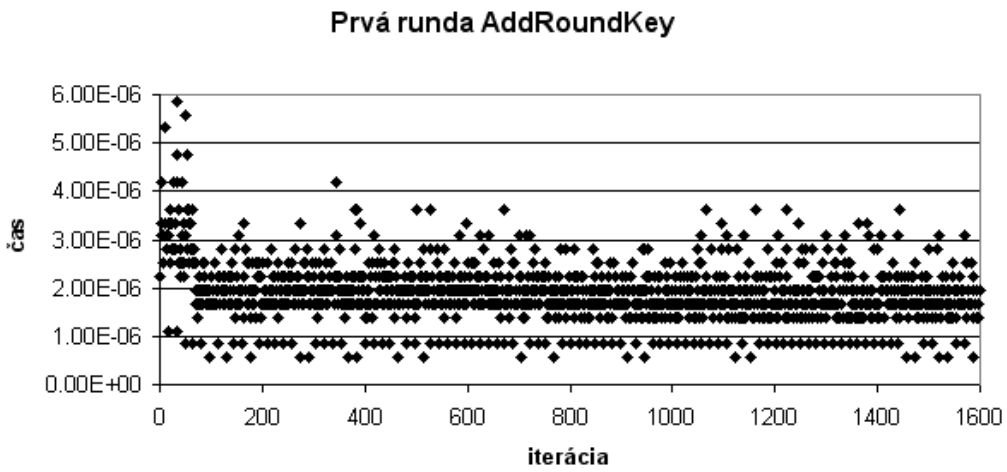
Funkcie *mixColumns* a *inverseMixColumns* sú implementované ako je uvedené v popise AES, ale na rozdiel od predošlej implementácie nie je táto operácia zapísaná do jedného príkazu, ale je rozpísaná podľa jednotlivých riadkov a bajtov stavu, ktoré sú medzi sebou XORované.

Expandovaný kľúč je počítaný v každej runde, v ktorej je potrebný. Takto je možné v pamäti uchovávať iba tú časť kľúča, ktorá je práve používaná. Týmto sa v pamäti zaberie menej miesta oproti ukladaniu celehého kľúča, ktorý je aspoň desaťkrát dlhší. Samotná implenetácia vyzerá tak, že na začiatku sa inicializuje konštanta rundy round_constant, v ďalšom kroku sa vypočítajú dočasné štyri bajty slova, ktoré sú následne substituované hodnotami s S-Box. Tento výsledok je XORonovaný s inicializovanou konštantou. Tieto štyri bajty sú XORované s štyrmi bajtmi predchádzajúceho expandovaného kľúča. Ďalší blok nového expandovaného kľúča sa získa XORovaním predošlého bloku nového kľúča s blokom predchádzajúceho kľúča.

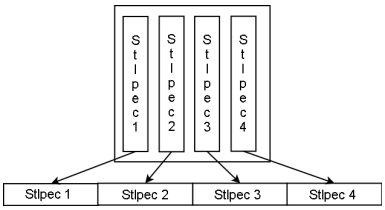
4.4.1 Vyhodnotenie meraní a pozorovaní

Merania a pozorovania prebiehali rovnako ako v predchádzajúcich prípadoch. K meraniu času spracovania bola použitý ten istý hlavičkový súbor *pctimer.h*, popísaný vyššie.

I keď sa jedná o značne odlišnú a optimalizovanú implementáciu AES od prvej uvedenej implementácie v jazyku C, pozorovaním a porovnávaním časov bolo zistené v podstate to isté. Aj v tomto prípade je možné získať informácie, ktoré poskytuje cache.



Obrázek 4.9: Časový priebeh *AddRoundKey* v prvej runde pri sto šifrovaniach so 128 bitovým kľúčom.



Obrázek 4.10: Štruktúra stavu.

V prvej runde šifrovania so 128 bitovým kľúčom v operácii *AddRoundKeySubstitution* boli pozorované napríklad tieto zásahy:

$$\langle x_2^0 \rangle = \langle x_3^0 \rangle = \langle x_7^0 \rangle = \langle x_{13}^0 \rangle \tag{4.9}$$

$$\langle x_1^0 \rangle = \langle x_8^0 \rangle \tag{4.10}$$

$$\langle x_4^0 \rangle = \langle x_5^0 \rangle = \langle x_6^0 \rangle = \langle x_{11}^0 \rangle \tag{4.11}$$

V poslednej runde toho istého šifrovania došlo k týmto zásahom:

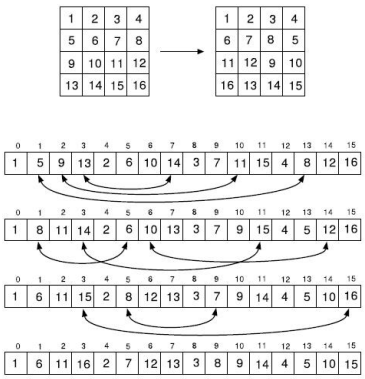
$$\langle x_1^{10} \rangle = \langle x_8^{10} \rangle = \langle x_9^{10} \rangle = \langle x_{15}^{10} \rangle \tag{4.12}$$

$$\langle x_2^{10} \rangle = \langle x_6^{10} \rangle \tag{4.13}$$

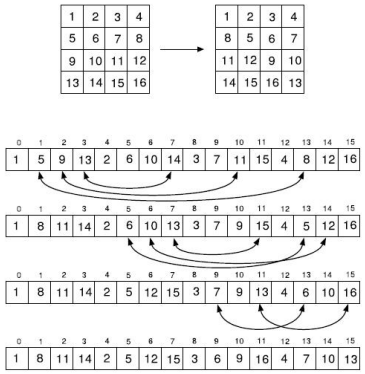
$$\langle x_3^{10} \rangle = \langle x_4^{10} \rangle = \langle x_5^{10} \rangle = \langle x_{10}^{10} \rangle = \langle x_{11}^{10} \rangle = \langle x_{13}^{10} \rangle \tag{4.14}$$

$$= \langle x_8^{10} \rangle = \langle x_9^{10} \rangle = \langle x_{14}^{10} \rangle \tag{4.15}$$

Následujúci obrázok 4.13 znázorňuje časový priebeh šifrovania v prvej runde pre operáciu *addRoundKeyAndSubstituteBytes*. Ako je vidieť, aj v tomto prípade je priebeh vo veľkej miere vyrovnaný. Tak isto aj pri šifrovaní s prázdnou cache boli pozorované výsledky takmer identické.



Obrázek 4.11: Implementácia *ShiftRows* .

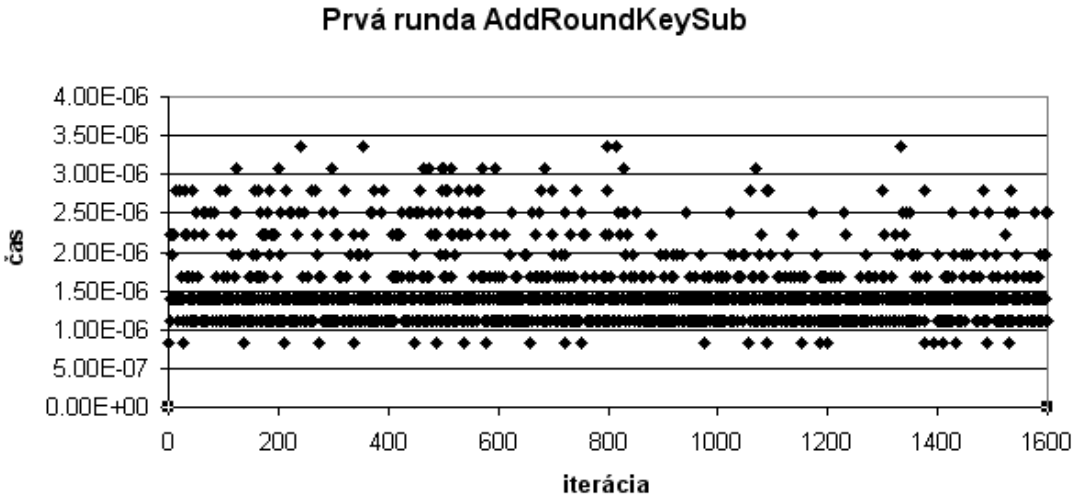


Obrázek 4.12: Implementácia *InverseShiftRows* .

Na ďalšom obrázku 4.14 je znázornený graf pre funkciu *addRoundKeyAndSubstitute-Bytes* v poslednej runde šifrovania. Merania boli opakované aj po vyprázdnení cache, výsledky boli len málo odlišné. Z toho vyplýva, že i pre dve nezávislé šifrovania dochádza v určitých častiach algoritmu k zásahom, čo znamená že sa vykonávajú operácie s dátami ktoré sú vždy uložené v cache, a to aj vtedy, keď cache pred šifrovaním neobsahovala žiadne dáta spojené s algoritmom.

Ďalší obrázok 4.15 ukazuje celkový priebeh prvej a poslednej rundy šifrovania. Táto implementácia je značne rýchlejšia ako prvá spomínaná v jazyku C. Avšak i napriek tomu že táto optimalizovaná implementácia je rýchlejšia v šifrovaní, nebráni vzniku zásahom. Vzhľadom na to, že pri optimalizácii sa zameriavalo hlavne na efektívnejšie využitie pamäte, je priebeh odlišný od priebehu napríklad prvej rundy pre prvú spomínanú implementáciu. Táto vlastnosť môže viesť k skomplikovaniu útoku, k väčšiemu počtu potrebných vzoriek.

Ani optimalizácia využitia pamäte nezabráni tomu, aby sa dáta ukladali do cache a vznikali tak zásahy, vďaka ktorým je tento algoritmus tak zraniteľný. Na obrázku 4.16 je uvedená četnosť výskytov rovnakých časových hodnôt pri sto šifrovaniach. Počet je celkom veľký, treba však počítať aj s tým, že nie každá takáto zhoda je použiteľná pre útok. Na čas má určite vplyv aj to, že táto implementácia bola pôvodne určená pre architektúru procesoru, ktorý súčasne spracováva len veľmi málo ďalších operácii. Merania však prebehli na bežnom počítači, na ktorom v pozadí pracuje viac aplikácii a rôznych služieb, ktoré mohli



Obrázek 4.13: Časový priebeh v prvej runde pre sto šifrovaní.

prerušovať prácu CPU a mohlo tak dôjsť k skresleniu výsledného času.

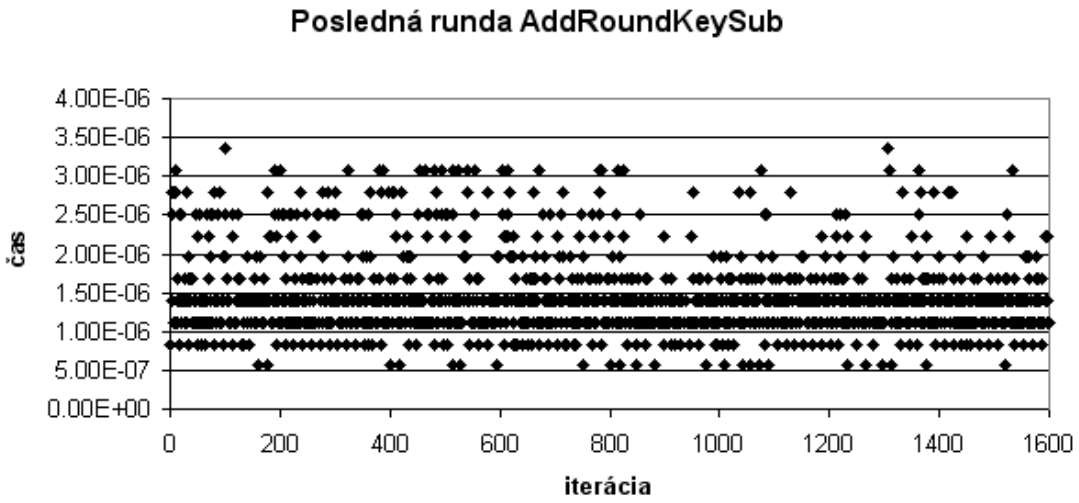
Aj napriek pokusu o optimalizáciu využitia pamäti a časovej náročnosti, stále dochádza k zásahom v cache. Je veľmi ťažké navrhnuť implementáciu AES takú, aby prístupové časy do cache pre všetky operácie konštanté. Je to ťažké aj z toho dôvodu, že tento typ útokokov *Side channel* neútočí na šifru ako takú, ale využíva vlastnosti hardwaru v systéme. Veľkou výhodou tejto implementácie je to, že expadnovaný kľúč sa počíta v každej runde. To zabraňuje tomu, aby si útočník pred každým šifrovaním vyčistil úplne cache tak, aby v nej nezostávali tabuľky, ktoré nie sú pre útok potrebné a mohli by vznikať nepotrebné vzorky dát. Takto sa do cache pri každom šifrovaní a počítaní expandovaného kľúča pristupuje viackrát, čo môže viesť k zvýšeniu potrebného počtu vzoriek, z ktorých bude veľký počet nepoužiteľných k odvodeniu bajtov kľúča a tým aj k väčšej odolnosti algoritmu.

4.5 Optimalizovaný ANSI C kód AES

Táto implementácia AES je značne optimalizovaná so zameraním sa hlavne na časovú výkonnosť. Nie sú v nej žiadne nadbytočné operácie alebo funkcie. To má ale za dôsledok to, že kód je ťažšie čitateľný a menej prehľadný. Pre výpočet sú definované tri dátové typy

- **unsigned char u8**
- **unsigned short u16**
- **unsigned int u32**

Tabuľky S-Box sú implementované ako jednorozmerné pole s 256 prvkami. Na rozdiel od predchádzajúcich typov, sú tieto tabuľky rozdelené celkom do desiatich tabuliek, z ktorých každá obsahuje už predpočítané hexadecimálne hodnoty. Tabuľky *Te0[]* až *Te4[]* sú určené pre šifrovanie, z toho *Te4[]* výhradne pre poslednú rundu šifrovania. Tabuľky *Td0[]* až *Td4[]* sú tabuľkami pre dešifrovanie, kde *Td4[]* je taktiež určená len pre poslednú rundu.



Obrázek 4.14: Časový priebeh v poslednej runde pre sto šifrovaní.

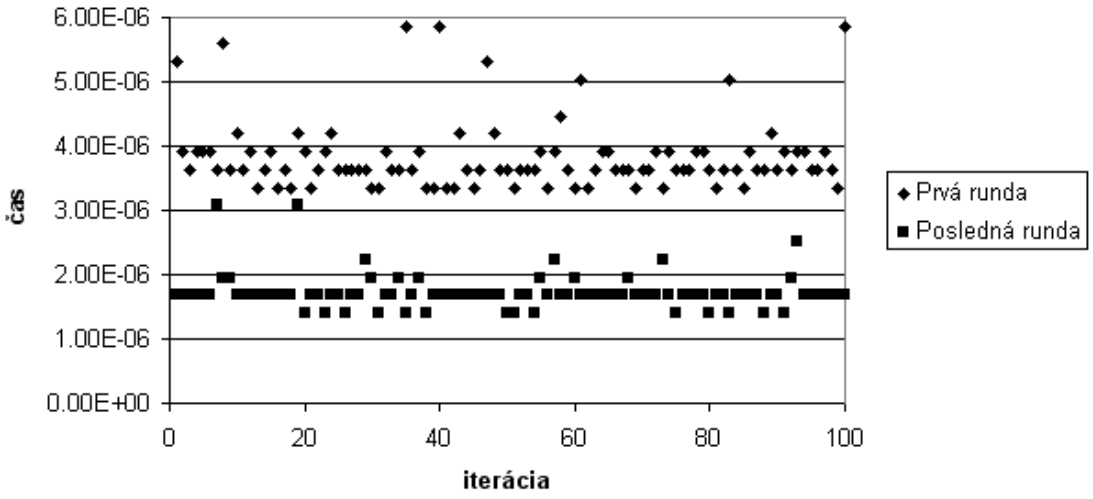
Môže sa zdať, že toľko veľa tabuliek zaberá moc miesta či už v zdrojovom kóde alebo aj po preklade programu v pamäti, čo je pravda, ale na druhú stranu zas prinášajú pozorovateľné zrýchlenie výpočtu, ktoré je v dnešnej dobe veľmi žiadané.

V tejto implementácii nie sú definované základné funkcie pomenované podľa jednotlivých operácii algoritmu tak ako v predchádzajúcich popisovaných typoch. Kód je rozdelený len do štyroch nutných funkcií, dve pre šifrovanie a dve pre dešifrovanie. Tieto funkcie a funkcia pre expanziu kľúča sú deklarované ako:

- ***rijndaelKeySetupEnc(u32 rk[], const u8 cipherKey[], int keyBits)*** je funkcia na expanziu kľúča cipherKey do premennej rk, v závislosti na dĺžke kľúča keyBits pri šifrovaní
- ***rijndaelKeySetupDec(u32 rk[], const u8 cipherKey[], int keyBits)***, inverzná k predchádzajúcej funkcii
- ***rijndaelEncrypt(const u32 rk[], int Nr, const u8 pt[16], u8 ct[16])*** funkcia pre hlavné šifrovanie vstupu pt expandovaným kľúčom rk, Nr určuje počet rúnd a ct je blok dát
- ***rijndaelDecrypt(const u32 rk[], int Nr, const u8 ct[16], u8 pt[16])***, funkcia pre dešifrovanie

Táto implementácia je súčasťou programu, ktorý využíva AES na šifrovanie a vlastne ho testuje. Tento program je vytvorený tak, aby bolo možné otvorený text šifrovať pomocou AES v dvoch základných šifrovacích režimoch, a to *Cipher Block Chainig* (CBC) a *Electronic Code Book* (ECB). V týchto dvoch režimoch sú simulované rôzne možnosti, ktoré môžu pri šifrovaní nastať. Sú to napríklad šifrovanie so stálym vstupným textom a meniacim sa kľúčom, alebo naopak, meniaci sa text a pevný kľúč a test typu Monte Carlo.

Optimalizácia tejto implementácie spočíva v tom, že jednotlivé operácie substitúcie bajtov, posúvanie riadkov stavu, XORovanie bajtov s expandovaným kľúčom a násobenie



Obrázek 4.15: Časový priebeh prvej a poslednej rundy.

v Column Mixing sú zhrnuté do jedného príkazu tak, aby sa všetky tieto operácie aplikovali v jednom kroku na celý jeden riadok stavu. Z toho potom vyplýva, že jedna runda obsahuje len štyri príkazy priradenia hodnoty z niektorej z tabuliek na základe indexu, ktorý je výsledkom posunu riadku a násobenia. Takto získané hodnoty zo štyroch tabuliek sú XORované s príslušnou časťou expandovaného kľúča. Takáto úprava algoritmu má za následok značné zrýchlenie výpočtu.

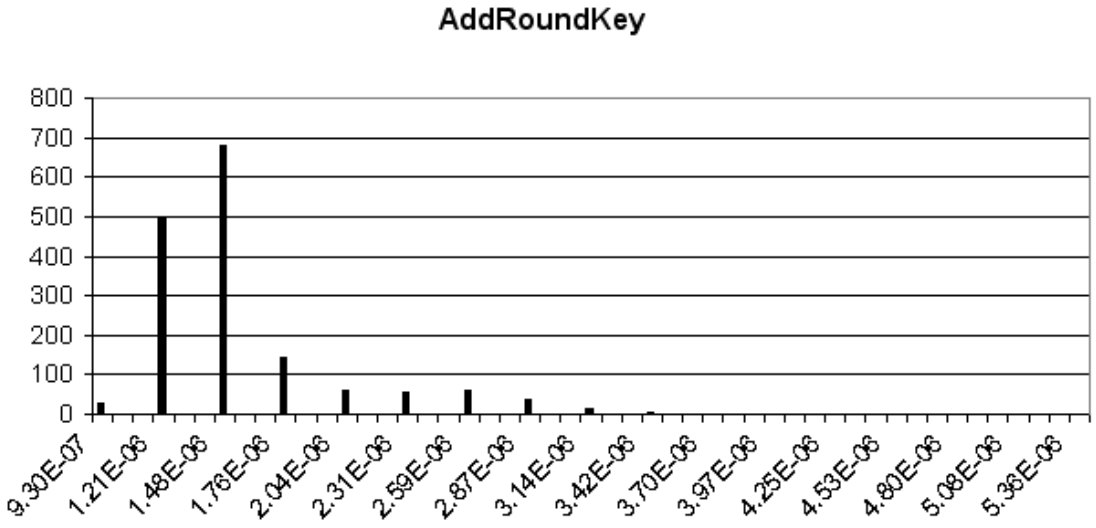
4.5.1 Vyhodnotenie meraní a pozorovaní

Ako je popísané v predchádzajúcom odstavci, táto implementácia nemá implementované jednotlivé operácie oddelene, ale sú vykonávané v jendom príkaze. Preto nebolo možné efektívne zmerať čas potrebný k výpočtu hodnoty pre konkrétnu operáciu, ale len čas, ktorý trvalo spracovanie prvej, resp. poslednej rundy, prípadne čas spracovania výpočtu jedného bajtu vstupného stavu.

Obrázok 4.17 znázorňuje priebeh približne 6 tisíc iterácii v prvej runde režimu ECB. Z priebehu je vidno, že časy sú veľmi malé ale hlavne, odlišujú sa len minimálne a je tam veľké množstvo zásahov v cache. Merania tejto implementácie prebehli na školskom servery Merlin.

Na ďalšom obrázku 4.18 sú časové hodnoty pre približne 8 tisíc vzoriek skupiny bajtov stavu, ktoré sú indexami pre tabuľku Te0, taktiež v ECB režime. Ako je vidieť, počas celého šifrovania s 128 bitovým kľúčom dochádza k častým zásahom v cache. Tieto zásahy síce nevyjadrujú zhodné hodnoty bajtov, ale vyjadrujú to, že tieto bajty ležia v jednom riadku cache a sú indexami pre vyhľadanie hodnôt pre danú tabuľku. To značne zmenšuje počet možných hodnôt pre konkrétny bajt kľúča a následný odhad tejto hodnoty vychádza z neporovnateľne menšej množiny hodnôt.

Pri analýze šifrovania v režime CBC bol pozorovaý rovnaký jav. Počas šifrovania sú aj v tomto režime uložené jednotlivé tabuľky v cache a v nich sa vyhľadávajú hodnoty na základe indexu, ktorým je konkrétny bajt stavu. Na obrázku 4.19 sú zobrazené časy trvania



Obrázek 4.16: Četnosť (osa y) výskytu rovnakých časových itervalov (osa x).

spracovania bajtov v tabuľke Te0 pre 1800 opakovaní.

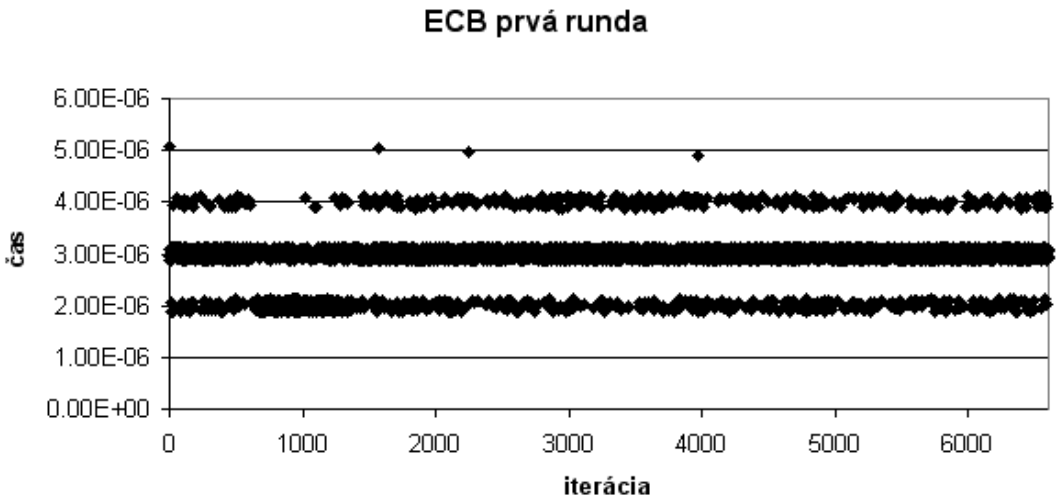
Napriek tomu, že sa jedná o optimalizovaný kód, ktorého hlavnou úlohou bolo zrýchlenie šifrovania alebo dešifrovania, nemá to žiadny pozitívny vplyv na odolnosť proti útokom časovej analýzy. Skôr naopak, práve to, že S-Box tabuľky sú rozdelené, čo rozdeľuje jednotlivé bajty stavu do určitých skupín a značne uľahčuje útok. Hodnota z tabuľky je ľahšie identifikovateľná pre konkrétny bajt stavu, kým v klasickej implementácii je to o niečo ťažšie, keďže sa pre každý bajt stavu používa jedna tabuľka. Táto vlastnosť je použitá v útoku, ktorý je popísaný v následujúcom odstavci.

4.5.2 Realizácia útoku

V tomto útoku [13] sa predpokladá, že platia vzťahy ktoré sú uvedené v kapitole 3. Útok je napísaný v jazyku C a je spustitelný v príkazovom riadku operačného systému UNIX/LINUX. Program je možné upraviť, prekompilovať a použiť niektorý z vyššie uvedených útokov. Útok využíva hlavne útok na prvú rundu s rozšírením aj pre rundu druhú (two-round attack).

Po spustení programu sa ako prvé generuje veľké množstvo časových vzoriek spúštaním šifrovacieho algoritmu s náhodným otvoreným textom a s využitím *OpenSSL* v AES. Tieto časové vzorky sú previazané s odpovedajúcimi časťami výstupného zašifrovaného textu. Každá dvojica čas - šifra je priebežne ukladaná do tabuľky. Medzi jednotlivými šifrovaniami sa musí vyprázdniť cache, aby v nej neostávali dáta s predchádzajúcich šifrovaní. Po každom šifrovaní sú jednotlivé bajty šifry otestované či šifrovanie naozaj skončilo a nečaká sa na dokončenie nejakej inštrukcie pri výpadku cache. Toto opatrenie je nutné hlavne pri použití takých architektúr procesorov, ktoré podporujú vykonávanie inštrukcií mimo poradia (out of order).

Po získaní dostatočného množstva vzoriek sa spustí samotný algoritmus útoku. Programu, ktorý útok vykonáva, sú dodávané dáta namerané v predchádzajúcom kroku pokým nie je odhalený tajný kľúč. K útoku nie sú použité vzorky časov, ktorých hodnoty sú aspoň



Obrázek 4.17: Časový priebeh prvej rundy v ECB režime s dĺžkou kľúča i vstupu 128 bitov.

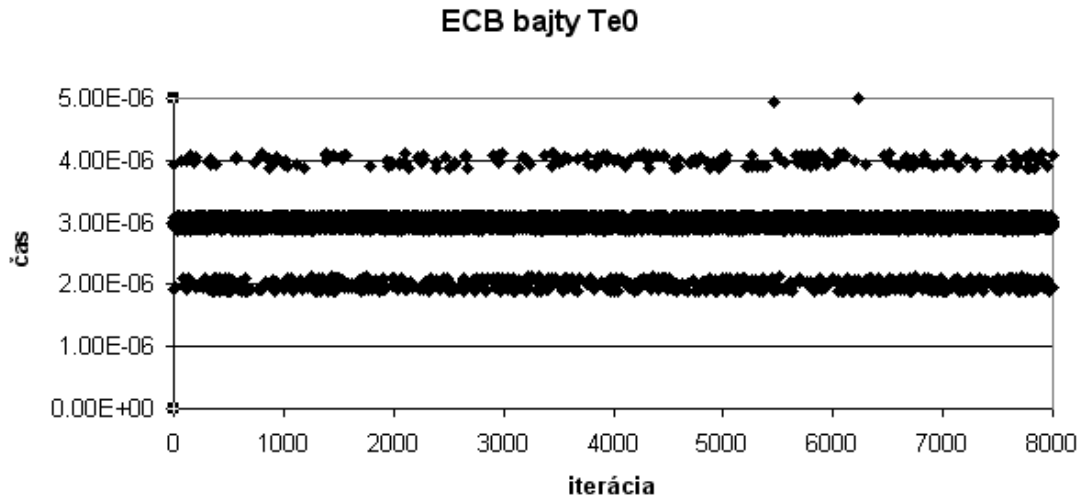
dvojnásobne vyššie ako je priemerná hodnota. Týmto sa čiastočne odstránia nepoužiteľné informácie a iné šumy, spôsobené napríklad výpadkami alebo prerušeniami inými aplikáciami.

Vyprázdnenie cache pred každým šifrovanie je potrebné aby sa zabránilo neželanému ovplyvneniu merania časových údajov, pretože pri priebehu tesne po sebe nasledujúcich šifrovaní by v cache mohlo zostávať veľa nepotrebných dát, ktoré by mohli do merania vnášať chyby. To znamená, že pri útoku v reálnom svete musí útočník nejakým spôsobom zabezpečiť pravidelné vymazanie cache, hlavne vymazanie vyhľadávacích S-Box tabuliek, aby údaje ktoré sa snaží zmerať, mali význam. Jednoduchým riešením je počkať, kým sa cache vyprázdni sama alebo uloženie tabuliek prepíšu svojimi dátami iné aplikácie bežiace na systéme. Pri útoku je tiež nutné, aby bol expandovaný kľúč spočítaný iba raz a jeho hodnota uložená po celú dobu šifrovania. Ak sa expandovaný kľúč počíta opakovane pred každým šifrovaním, môže to vniesť vedľajší efekt, načítanie tabuliek späť do cache.

Pri odhaľovaní kľúča v tomto útoku sú taktiež použité algoritmy z kategórie umelej inteligencie. Využívajú sa najmä pri útoku na poslednú rundu. Napomáhajú k správnemu odhadu konkrétnej hodnoty bajtu kľúča, keďže z meraní je možné zistiť iba určité vzťahy medzi týmito bajtmi, ale tieto vzťahy presne vymedzujú množinu hodnôt.

Pred samotným spustením útoku je potrebné si pomocou priloženého generátoru *genKey* vygenerovať náhodný kľúč, ktorý bude použitý pri šifrovaní a na ktorý bude vykonaný útok. Je možné zapísať vlastný kľúč do súboru, ktorý sa použije ako vstupný pri útoku. Z tohto súboru sa použije prvých 16 bajtov pre hodnoty 128 bitového kľúča. K zostaveniu programu je priložený potrebný *makefile* súbor. Po kompilácii je vytvorený spustiteľný súbor *aes_attack*, ktorého vstupnými parametrami môže byť súbor, v ktorom je uložený kľúč následovaný číslom, ktoré určuje aký počet vzoriek sa má skúmať. Je to vlastne exponent čísla 2. Výsledky sú zobrazované na štandardný výstup. Pri vhodnom zvolení počtu pozorovaných vzoriek je výpočet veľmi rýchly a odhalenie kľúča trvá rádovo v jednotkách sekúnd. Pri testovaní tohto útoku boli najlepšie výsledky dosahované pri zvolenom počte vzoriek 12 až 16. čiže bolo skúmaných 2^{12} až 2^{16} vzoriek.

K útoku je možné použiť po rekompilácii i mierne upravené implementácie AES, napríklad



Obrázek 4.18: Časový priebeh spracovania bajtov v skupine tabuľky Te0 v režime ECB.

bez použitia tabuľky T_4 pre poslednú rundu šifrovania. Takisto je možné zmeniť typ útoku.

4.6 GNT Token

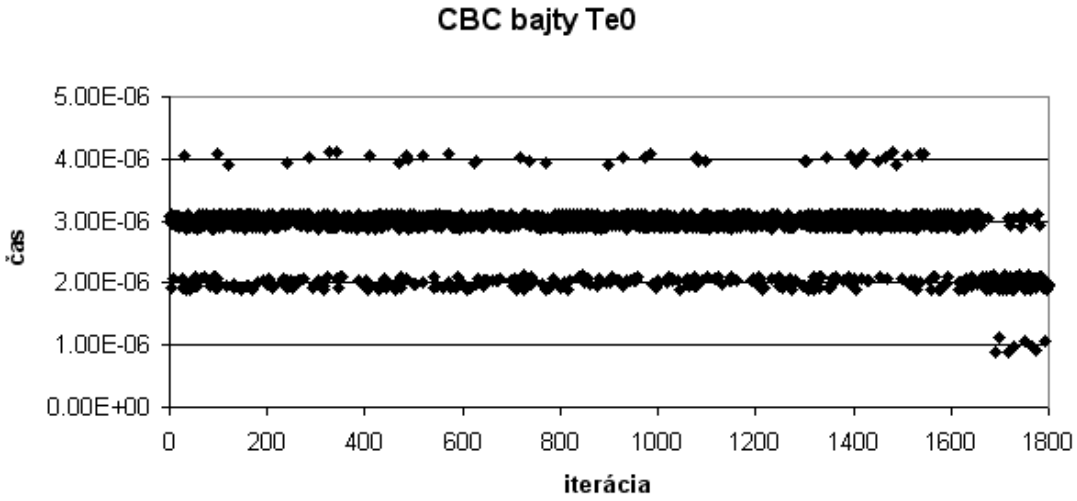
GNT USB Token (ďalej len Token) je kryptografické zariadenie, ktoré vykonáva kryptografické operácie so zameraním na šifrovanie, autentifikáciu a kontrolu itegrity dát [17]. Toto zariadenie podporuje ako symetrické tak aj asymetrické šifrovanie a taktiež počítanie hašovacích funkcií. Tieto vlastnosti môžu byť použité pri (de)šifrovaí dokumentov, elektronickom podpise apod. Všetky potrebné algoritmy sú uložené v pamäti Tokenu.

Token obsahuje nevolatilnú pamäť EEPROM, ktorá má veľkosť 64 kilobajtov. Táto pamäť je rozdelená na dve časti, na užívateľskú a systémovú. Užívateľská časť má veľkosť 63 kilobajtov a systémová časť má 1 kilobajt. Komunikácia s počítačom prebieha pomocou rozhrania USB 1.1.

K potrebám tejto práce bol použitý algoritmus AES, ktorý je uložený na Tokene. Tento modul pracuje s implementáciou AES podľa popisu FIPS 197 (Federal Information Processing Standards) [16]. Táto implementácia je veľmi podobná implementácii uvedenej v odstavci 4.2. Jedná sa vlastne o referenčný popis algoritmu AES. Presnejšie informácie o implementácii na Tokene sa nepodarilo zistiť, ale dá sa predpokladať, že sa využíva veľmi podobná implementácia, to znamená s definovanými jednotlivými operáciami podľa popisu a používajúc dve S-Box tabuľky, tak ako je to uvedené v popise FIPS. Token k šifrovaniu používa 128 bitový kľúč.

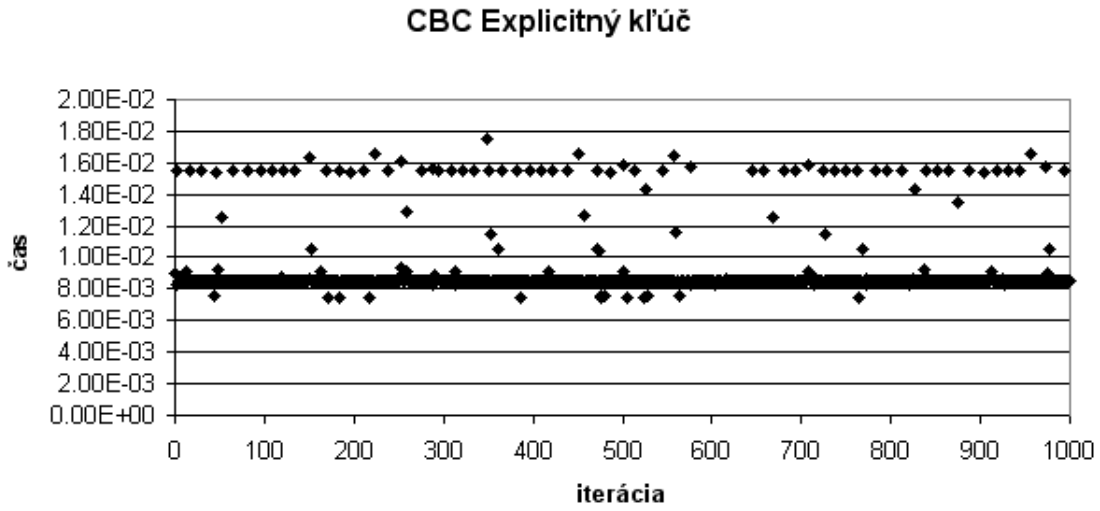
4.6.1 Popis obslužného programu

Tento program je implementovaný v jazyku C++ v prostredí Visual Studio 2003. Na meranie času potrebného na šifrovanie bol použitý hlavičkový súbor ako v predchádzajúcich implementáciach v jazyku C, *pctimer.h*. Program ďalej využíva súbor *gntds.h* a knižnicu *gntds.lib*, ktoré sú dodávané spoločne s Tokenom. V týchto súboroch sú definované a



Obrázek 4.19: Časový priebeh spracovania bajtov v skupine tabuľky Te0 v režime CBC.

deklarované premenné, štruktúry a funkcie potrebné k práci s Tokenom. Šifrovanie bolo testované v režimoch CBC a ECB a za použitia explicitne generovaného kľúča pre každé šifrovanie alebo kľúča uloženého v pamäti Tokenu. Zariadenie je odolné proti útokom *Simple Power Analysis* a *Differential Power Analysis* [17]. Takisto autentifikačný algoritmus Tokenu je odolný proti jednoduchým útokom časovou analýzou.



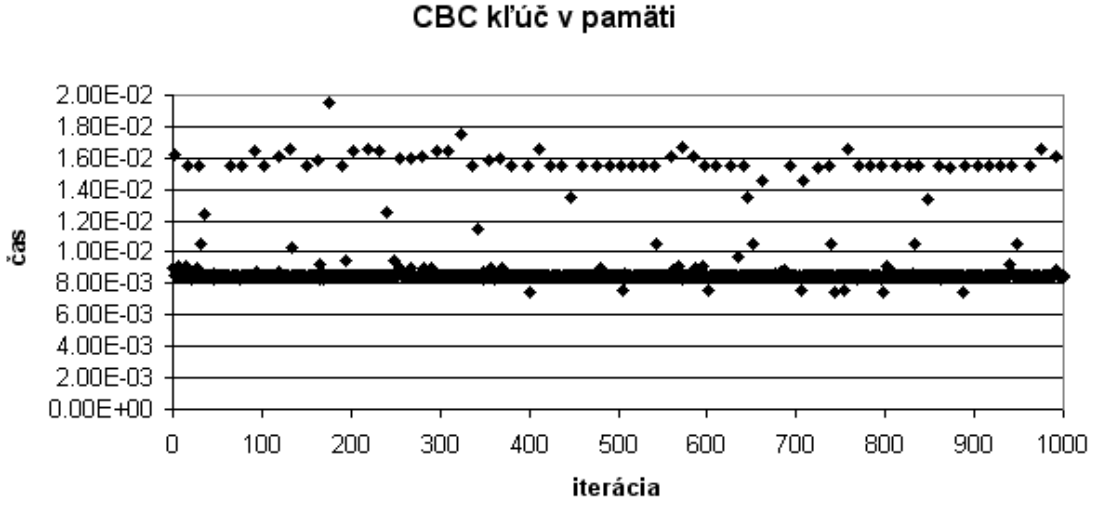
Obrázek 4.20: Časový priebeh šifrovania v režime CBC s explicitne zadaným kľúčom.

Token umožňuje rôzne spôsoby práce a uchovávanie kľúča. Je možné generovať kľúč náhodne pomocou generátoru implementovaného na Tokene alebo je možné hodnotu kľúča zadať z prípadného dialógového okna pre prácu s Tokenom, prípadne zo súboru. Token umožňuje tiež uložiť kľúč na určité miesto v pamäti bez potreby exportovať kľúč mimo

Token. V tejto práci boli využité možnosti explicitne zadávaného pseudonáhodného kľúča a taktiež možnosť uloženia kľúča v pamäti Tokenu.

K samotnej komunikácii s Tokenom bolo potrebné v programe definovať užívateľský PIN kód a následné aktivovať Token príkazom *GNT_Issue()*. Táto funkcia má jeden z parametrov ukazateľ na štruktúru dát, v ktorej sú definované hodnoty aké sa majú na Tokene nastaviť a ako bude možné s nimi narábať. Po úspešnom vykonaní aktivácie Tokenu je nutné prihlásenie sa užívateľa k práci s Tokenom na základe definovaného PIN kódu, ktorý je jedným z parametrov funkcie *GNT_UserLogin()*, druhým parametrom je jednoznačný identifikátor Tokenu. Po úspešnej autorizácii je možné s Tokenom pracovať.

V prvom prípade bol použitý režim šifrovania CBC s kľúčom uloženým v pamäti Tokenu. Po alokovaní pamäte potrebnej k uloženiu kľúča bol vygenerovaný náhodný kľúč. Ďalej v programe sa pracovalo s ukazateľom na toto miesto v pamäti. Pred volaním šifrovacej funkcie *GNT_Encrypt()* bolo potrebné nastaviť parameter *pCrOpt*, ktorý je ukazateľ na štruktúru *PGNT_CRYPT_OPTIONS*, ktorá obsahuje informácie o algoritme aký sa má použiť (v našom prípade to je AES), aký režim sa má použiť (CBC) a v tomto prípade aj adresu pamäte kde je uložený kľúč. Šifrovacej funkcii sa ďalej ako parametre predávajú ukazateľe na dáta *pbInData*, ktoré sa majú šifrovať, dĺžka týchto dát, ukazateľ na miesto v pamäti kde bude uložená šifra *pbOutData* a dĺžka výstupných dát *&dwOutDataLen*. Po overení úspešného šifrovania bolo pre kontrolu následne vykonané dešifrovanie pomocou funkcie *GNT_Decrypt()*, ktorá má parametre obdobného významu ako funkcia na šifrovanie. Pri použití režimu CBC s explicitne definovaným kľúčom bol postup nastavenia takmer rov-



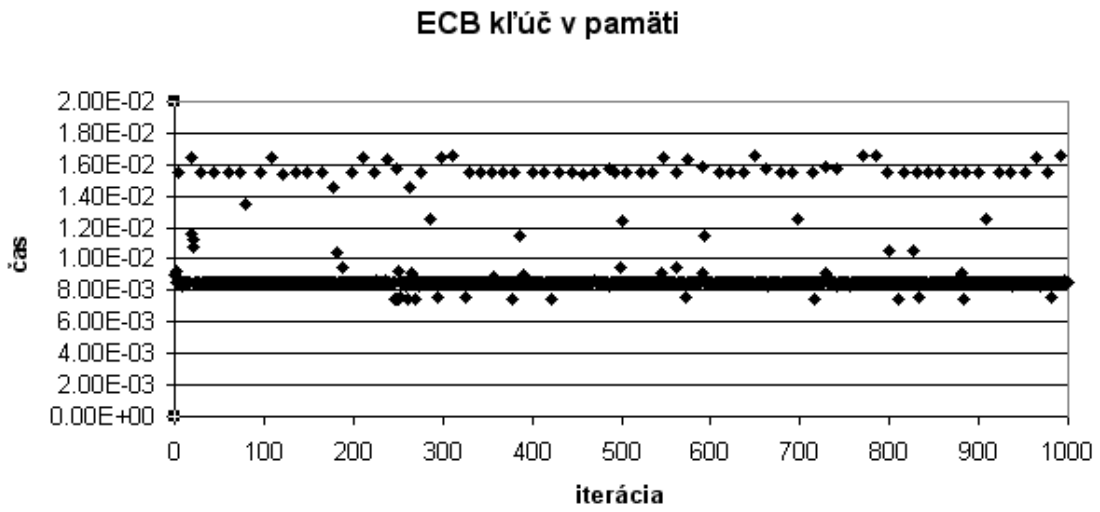
Obrázek 4.21: Časový priebeh šifrovania v režime CBC s kľúčom uloženým v pamäti.

naký ako v predchádzajúcom prípade len s rozdielom, že miesto definovania adresy pamäte s kľúčom bolo nastavené pseudonáhodné generovanie kľúča. Ten istý postup bol aplikovaný aj na prípady použitia režimu ECB s kľúčom v pamäti alebo explicitne definovaným.

4.6.2 Vyhodnotenie meraní a pozorovaní

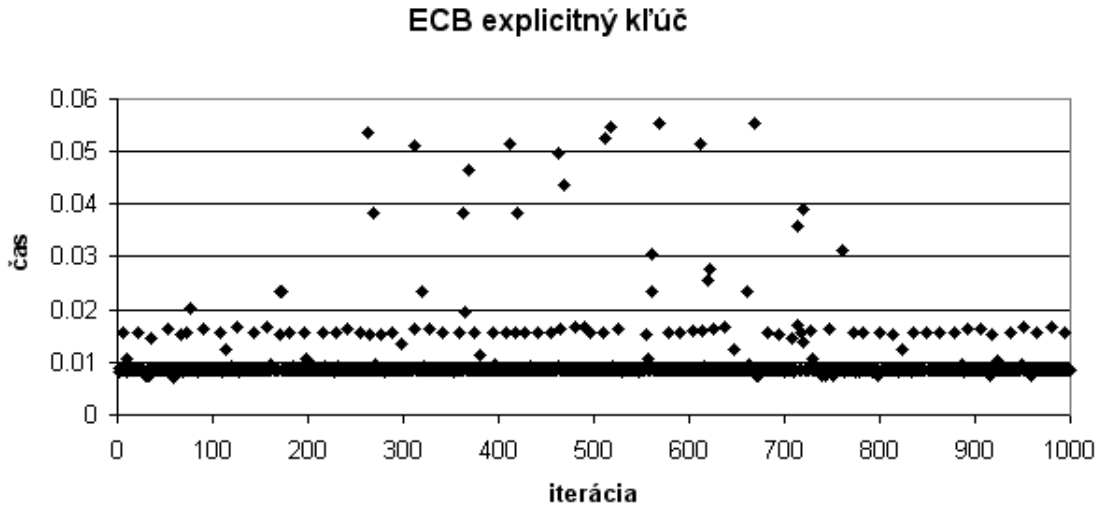
Keďže nebolo možné priamo pristúpiť k implementácii algoritmu na Tokene a to ani s právami administrátora a tým pádom zmerať časy pre jednotlivé operácie a rundy, merané

boli len časy tisíc kontinuálnych šifrovaní. To, že sa nepodarilo pristúpiť detailnejšie do pamäti, je veľká výhoda proti *side channel* útokom, keďže tieto útoky vyžadujú prístup ku cache pamäti a v prípade časového útoku je potrebný prístup aspoň k jednotlivým rundám algoritmu.



Obrázek 4.22: Časový priebeh šifrovania v režime ECB s kľúčom uloženým v pamäti.

Zamedzenie prístupu je jednou z dobrých možností ochrany. V tomto prípade je to aj efektívne, pretože sa jedná o zariadenie určené na presne vymedzený malý počet operácii, ktoré je schopné vykonávať. Využiť ale tento princíp nie je dosť dobre možné na platformách, ktoré sú určené pre širšie a všeobecnejšie použitie.



Obrázek 4.23: Časový priebeh šifrovania v režime ECB s explicitne zadaným kľúčom.

Z pozorovaných časov, uvedených na obrázkoch 4.20 4.21 4.22 a 4.23 vyplýva, že aj v

tomto prípade dochádza k zásahom v cache. Priebehy šifrovania spusteného tisíckrát po sebe pre jednotlivé režimy a možnosti kľúča sú si veľmi podobné.

Je tu tiež pozorovateľné to, že spracovanie šifrovania textu o veľkosti 16 bajtov trvá o niečo dlhšie ako na bežnom počítači, čo je celkom samozrejmý fakt vzhľadom na technické parametre Tokenu, ktoré nie sú moc vzdialené parametrom čipových kariet.

Pri implementácii AES pre čipovú kartu bude teda jedna z najvhodnejších možností ochrany pred časovým utokom podobná ochrana akú má Token. Zamedzením prístupu do EEPROM pamäte karty, v ktorej sú uložené dáta pri šifovaní sa znemožní útokčníkovi nazbierať potrebné množstvo časových vzoriek potrebných k útoku. Naskytuje sa však otázka, či útočník bude schopný nejakým iným spôsobom získať kontrolu nad cache, prípadne si môže zaobstarať vlastnú čipovú kartu bez takejto ochrany, ktorá bude identická s kartou, na ktorú chce zaútočiť. Na tejto karte si môže vyskúšať správanie sa daného algoritmu, chovanie sa pamäte na karte a zistiť tak, ako sa dá získať prístup do EEPROM poprípade zostrojiť modifikovaný časový útok.

4.7 Zhrnutie

Analýzou a pozorovaním jednotlivých implemetácii algoritmu AES bolo ukázané, že *Side channel* útoky, v tomto prípade útok *časovou analýzou*, naozaj nezávisí na samotnom šifrovacom algoritme a typoch jeho implementácie, ale je závislý na systéme, na ktorom sa šifruje ale hlavne na architektúre pamäte cache. Pomocou vlastností tejto pamäte je možné sledovať potrebné časti výkonu šifrovania (dešifrovania) a získavať tak informácie, ktoré síce nesúvisia priamo s algoritmom, ale pomocou nich je možné značne množinu hľadaných hodnôt zmenšiť a tým aj uľahčiť vykonanie útoku. Tento typ útoku je s najväčšou pravdepodobnosťou použiteľný aj na implementácie pre čipové karty. I keď architektúra karty je odlišná od bežného počítača, pri šifrovaní musí byť využitá EEPROM pamäť a práve na ňu je možné sa zamerať a aplikovať princípy útokov z bežných počítačov.

Kapitola 5

Možnosti ochrany proti útokom

V tejto kapitole si priblížime niektoré potenciálne možnosti, ktoré môžu prispieť k väčšej odolnosti AES proti útokom časovej analýzy. Navrhnúť takto odolnú implementáciu je úloha veľmi ťažká, pretože nevychádza priamo z vlastností algoritmu, ale z vlastností systému, na ktorom je používaný. Taktiež je veľmi obtiažne navrhnúť program, hlavne šifrovací, ktorý by zakaždým trval konštantnú dobu. To je síce možné, ale v tom prípade by jednotlivé operácie museli byť tak pomalé, ako tá najpomalšia z nich.

5.1 Zamedzenie prístupov do cache

Časová analýza je útok založený na prístupe do cache a získavania časových údajov o spracovávaní jej obsahu. Jednou možnosťou ako tomuto útoku zabrániť, je znemožniť útočníkovi získavať tieto informácie. Možnosťou ako zabrániť útočnikovi sledovať tieto hodnoty je nahradenie vyhľadávacích S-Box tabuliek odpovedajúcou postupnosťou logických operácii, najlepšie takou, v ktorej by vykonanie každej operácie trvalo rovnako dlhý časový usek [12]. V tomto prípade je však cenou za takéto opatrenie zníženie celkového výkonu implementácie, hlavne časová náročnosť značne stúpne, čo otvára možnosti využitia iných typov útokov.

Ďalšou možnosťou, ktorá môže značne skomplikovať útok je použitie implementácie, ktorá bude schopná súbežne spracovávať niekoľko šifrovaní. Veľmí však záleží na návrhu takéhoto programu, ako budú jednotlivé vlákna využívať zdielanú pamäť a cache a hodnoty v nich tak, aby útočník nemohol len tak ľahko zistiť ktoré dáta patria ku ktorému šifrovaniu. Takýto návrh je však veľmi viazaný na konkrétne architektúry procesorov. V tomto môžu byť veľmi nápomocné nové typy procesorov, ktoré dokážu spracovávať viac vlakien súbežne alebo aj najnovšie viacjadrové CPU, kde princíp práce s cache môže byť už odlišný, čo bude vyžadovať zmenu taktiky útokov.

Uloženie vyhľadávacích tabuliek do registrov a nie do cache môže byť tiež vhodným riešením problému útoku časovou alanýzou. Niektoré dnešné procesory poskytujú dostatočne veľký priestor v registroch na uloženie 256 bajtových tabuliek. Tu sa však vynára otázka výkonnosti takéhoto návrhu, ktorý bude taktiež silne závislý na danom procesore. Taktiež sa tu otvárajú dvere pre použitie iných typov útokov [12].

Na zmätenie útočníka je tiež možné použiť aj to, že pri akomkoľvek prístupe do cache za účelom výberu hodnoty z tabuľky, sa prístup uskutoční aj na všetky ostatné riadky cache, i keď dáta z nich nie sú v danej chvíli potrebné. Toto je však veľmi neefektívny prístup. Je možné ho však vylepšiť. Jednou možnosťou je uložiť jednu tabuľku na presne určený

počet riadkov cache a pri hľadaní hodnoty v nej bude program pristupovať k všetkým týmto riadkom. Použiť tento princíp pre celý priebeh šifrovania je nevhodné. Najviac zraniteľné sú prvá a posledná runda. Takže ak sa tento princíp použije len na tieto dve rundy a pre ostatné rundy sa použije klasický prístup, zníženie výkonnosti nemusí byť až tak závažné pre bežné použitie [12]. Treba však myslieť tiež na to, že dnešné procesory sú vo veľkej miere schopné analyzovať inštrukcie a následne preskladať ich poradie. Je preto potrebné zabezpečiť aby tieto inštrukcie neboli závislé na dátach a brať ohľad na časovanie jednotlivých inštrukcií.

5.2 Použitie alternatívnych vyhľadávacích tabuliek

Táto možnosť je použiteľná hlavne pri druhoch implementácii, ktoré využívajú štyri tabuľky tak, ako je popísaný predposledný typ v predchádzajúcej kapitole.

Jednou možnosťou je rozdeliť tieto tabuľky následovne: jedna 256 bajtová tabuľka pre S-Box, dve 256 bajtové tabuľky, jedna 1024 bajtová tabuľka T_0 a jedna 2048 bajtová tabuľka, ktorá bude obsahovať hodnoty tabuliek T_0 až T_3. Obdobne sa postupuje aj pre tabuľky používané v poslednej runde. Toto rozdelenie spôsobí to, že jednotlivé bajty v prvej, prípadne poslednej runde nebudú patriť do skupín, podľa ktorých sú hlavne orientované útoky [12].

5.3 Zatienenie prístupu do pamäti

Možnosťou ako znížiť riziko útoku je pseudonáhodné obmienanie jednotlivých dát v cache medzi sebou. Treba však zaručiť to, aby v čase obmeny nebola cache prístupná pre iné aplikácie. Toto si vyžaduje značnú podporu v hardwarovom vybavení, najmä v procesore a správe pamäti [12].

Jednoduché pridanie šumu pri prístupe do cache, napríklad zo súbežného falošného šifrovania zvýši počet potrebných vzoriek, ktoré sú potrebné k útoku ale samotný útok nijak neeliminuje.

5.4 Úprava hardware

Možnosťou ako zabrániť útokom časovej analýzy je aj úprava hardwaru. A to tak, aby pamäť cache neposkytovala potrebné časové infomrácie o implementácii. Na takto upravenom hardware však zrejme nebude možné použiť ľubovoľnú implementáciu AES a ak áno, tak nie každá môže byť odolná proti útoku. Táto možnosť má nevýhody v bežnom použití a je ťažko prenostiteľná, avšak pri použití špecializovaného hardwaru a špecialného návrhu AES pre presne definované účely, sa javí táto možnosť ako reálna.

5.5 Obmedzenie zdieľania cache

Úlohou tejto varianty ochrany je zamedziť v prístupe do pamäti využivanej šifrovacím algoritmom iným aplikáciam, ktoré by mohli sledovať jednotlivé časy [12]. Táto varianta je síce dosť účinná, ale jej realizovanie je značne náročné a taktiež nesie so sebou niekoľko dosť zložitých problémov, ktoré treba vyriešiť aby bola ochrana účinná. Problémom pri návrhu je hlavne samotný procesor, niektoré typy pri svojich výpočtoch používajú jedno vlákno, iné počítajú súbežne viac vlákien. To znamená, že cache pri použití viac vlákien bude rozdelená na viac logických častí, čo niektoré moderné procesory nemusia podporovať.

5.6 Zakázanie pamäte cache

Táto možnosť je síce účinná na sto percent, avšak cena za ňu je veľmi veľká [12]. Výpočet sa veľmi výrazne spomalí, preto je potreba túto možnosť upraviť tak, aby bola reálne použiteľná. Úprava spočíva v tom, že pri výpadku v pamäti sa zabráni opätovnému načítaniu dát do cache a vykoná sa ich priame použitie z operačnej pamäte. Postup takéhoto výpočtu je, že sa najskôr načítajú vyhľadávacie tabuľky do cache, aktivuje sa režim zabraňujúci výpadkom, spustí sa šifrovanie alebo dešifrovanie a po skončení sa zruší tento režim. Týmto prístupom však súbežiace aplikácie stratia na výkone, pretože nebudú môcť dostatočne využívať cache. Takisto je nutné ošetriť spoluprácu s operačným systémom, pretože sa jedná o zásah do správy pamäti a napríklad v linuxe si to vyžaduje aby algoritmus mal určité privilégia. Ďalej je tiež nutné aby danú operáciu podporovala architektúra procesoru, ktorý je v systéme.

5.7 Zamedzenie merania časov

Toto je asi jedna z najprirodzenejších metód ako zabrániť časovému útoku, znie jednoducho, ale realizácia je značne náročná. Hlavnou úlohou je to, aby časy, ktoré útočník získa z cache nedávali možnosť nejakej súvislosti s konkrétnymi bajtmi či už stavu alebo kľúča, pridaním nepodstatných informácií, šumu. Jednou možnosťou je pridanie náhodného oneskorenia k operáciám, ktoré sú predmetom záujmu [12]. Toto podstatne zvýši potrebný počet vzoriek.

Ďalšou možnosťou je návrh takej implemetácie AES, v ktorej by všetky operácie trvali rovnaký časový úsek. Čo však vedie k tomu, že aj tie najrýchlejšie operácie budú musieť byť tak pomalé, ako najpomalšia operácia.

5.8 Dynamické ukladanie vyhľadávacích tabuliek

Pretože princípom časovej analýzy je sledovanie kde a ako sú uložené tabuľky v pamäti cache v priebehu výpočtu, je možné zabrániť útoku tým, že sa odstránia vzťahy medzi jednotlivými tabuľkami a ich uložením. Jednou variantou je použitie určitého počtu kópií jednej tabuľky a pri šifrovaní náhodne vyberať nejakú kópiu, z ktorej sa bude vyberať hľadaná hodnota. Tu ale hrozí nebezpečenstvo zvýšenia počtu výpadkov [12] a predĺženiu doby šifrovania.

Iná možnosť je počas výpočtu preskupovať jednotlivé časti tabuľky. Treba však brať ohľad na efektivitu tohoto preskupovania, aby sa dáta zbytočne dlho nehľadali po taktomo preskupení.

Výkonnosť a spoľahlivosť takéhoto opatrenia je však silne závisla od kofigurácie systému a architektúry procesoru a jeho pracovnej frekvencie, ako rýchlo bude schopný reagovať na takéto zmeny v cache [12].

5.9 Ochrana prvej a poslednej rundy

Niektoré z vyššie uvedených možností ochrany sú neefektívne keď sa použijú na celý algoritmus. Z popisu útokov vyplýva, že najzraniteľnejšími časťami AES sú prvá, resp. druhá runda a posledná runda. Vhodným aplikovaním niektorej z uvedených možností práve a iba na tieto rundy nemusí viesť k zhoršeniu výkonnosti a môže do značnej miery skomplikovať aplikáciu niektorého z útokov.

5.10 Zhrnutie

Možností ako zabrániť alebo aspoň obmedziť použitiu útoku časovou analýzou je mnoho. Ale každá možnosť nesie so sebou mnoho často veľmi komplikovaných problémov, ktoré treba prekonať aby bola ochrana účinná. Efektívnosť jednotlivých opatrení sa môže líšiť od typu použitého operačného systému, implementácie AES, od architektúry procesoru a vyrovnávacej pamäte cache. Preto je vhodné a veľmi dôležité, aby sa pred samotným zostrojením nejakej implementácie AES zhodnotili všetky možnosti a riziká. Kde a ako bude program používaný, na akom systéme, procesore, aká je reálna dostupnosť útočníka k systému, na ktorom sa implementácia bude využívať, či bude potrebné aby návrh bol schopný šifrovať v krátkom čase s väčším využitím cache, alebo naopak, menšie využitie cache na úkor času a podľa toho vhodne zvoliť jednu alebo viac možností ochrany.

Kapitola 6

Záver

Táto diplomová práca sa zaoberala vplyvom útokov side channel na šifrovací algoritmus AES. Konkrétne typu útoku časovou analýzou.

V práci bol vysvetlený teoretický princíp algoritmu AES, princípy jednotlivých útokov a boli analyzované niektoré možné implementácie tohto algoritmu a možný dosah útoku na ne. Práca bola rozdelená do jednotlivých tématických celkov, v ktorých boli tieto témy detailne popísané.

V prvej časti práce bola vysvetlená funkcia jednotlivých častí AES, z ktorých pozostáva a ako sa dá dopracovať od otvoreného textu k šifre a naopak. Ďalej tu bola vysvetlená podstata side channel útokov, ako k nim môže môjsť a čo je potrebné k takémuto útoku. Keďže táto práca je zameraná na časovú analýzu AES, boli detailne popísané hlavné a úspešné útoky, ktoré ohrozujú bezpečnosť tejto šifry.

V práci bolo diskutovaných päť implementácii AES, ktoré boli navrhnuté v dvoch rôznych programovacích jazykoch, s rôznymi optimalizáciami, ale i bez optimalizácií. Jedna implementácia bola určená pre operačné systémy založené na UNIX/LINUX, ostatné pre systém Windows z toho jedna z nich pre špeciálny šifrovací USB modul. Sú tu uvedené výsledky jednotlivých meraní a pozorovaní a zhrnuté možnosti útoku na tieto implementácie.

V záverečnej časti boli navrhnuté a diskutované možnosti ochrany pred týmto typom útoku. Niektoré z nich v takom znení v akom sú uvedené v práci nie sú použiteľné v bežnom živote, avšak vhodnou úpravou ich princípu môžu byť veľmi účinné. Je veľmi zložité, možno až nemožné, navrhnuť algoritmus, ktorý by bol odolný proti všetkým typom útokov. V dnešnej dobe je ich veľmi veľa a to aj vďaka tomu, že technické prostriedky sú oveľa ľahšie dostupné. K vykonaniu útoku však nestačia len technické prostriedky, ale hlavne veľmi dobré znalosti z oblastí matematiky, softwarového a hardwarového inžinierstva, pretože útoky sú veľmi náročné a komplikované na zostrojenie.

6.1 Zhodnotenie práce

V práci boli zhrnuté vedomosti o útokoch, ktoré vo svojej podstate nevyužívajú slabín v algoritme, ale k odhaleniu kľúča využívajú vlastnosti technického vybavenia počítača. Bolo ukázané, v ktorých častiach priebehu algoritmu je možné, aby útočník získaval informácie vedúce k prelomeniu šifry.

Táto práca poukazuje na chyby, ktoré sa stali pri výberovom konaní na algoritmus AES a sú zhrnuté známe úspešné útoky. Túto prácu je možné chápať ako súhrn najdôležitejších informácií nielen o útokoch ale aj o ochrane proti nim. V práci sú diskutované rôzne otázky,

ktoré by si mal položiť každý návrhár algoritmu AES a zvážiť čo najviac možných dôsledkov, ktoré budú vyplývať s používanie jeho implementácie.

Prácu nemožno chápať ako konečné a jediné riešenie v prípade ochrany pred útokmi. Dá sa povedať, že toto je beh na veľmi dlhú trať a je potrebných ešte ďaleko viac informácii, skúseností a pozorovaní, aby bolo možné vytvoriť čo najodolnejšiu implementáciu AES. Pretože tá, čo je v súčasnosti používaná ako štandard, je veľmi zraniteľná, čo môže mať nepriaznivé následky v budúcnosti.

6.2 Smery ďalšieho výskumu

K tomu, aby sa AES stal odolnejším proti útokom časovej analýzy a zvýšila sa tým pádom aj jeho kvalita, sú navrhnuté tieto smery ďalšieho výskumu:

- Možnosti uplatnenia popisovaných opatrení nie len na bežných počítačoch, ale aj pre 8 bitové architektúry procesorov na čipových kartách.
- Úprava algoritmu Rijndael tak, aby nebolo možné z cache získavať časové informácie o šifrovaní.
- Rozšírenie analýzy o ostatné typy side channel útokov.
- Vytvoriť súhrn informácii pre jednotlivé typy CPU, na ktorých je AES viac alebo menej zraniteľný.
- Možnosti uplatnenia upraveného algoritmu AES v bežnom užití.

Literatura

[1] Daemen, J., Rijmen, V..: *The Design of Rijndael.* Springer, Berlin, 2002

[2] Daemen, J., Rijmen, V..: *AES Proposal: Rijndael.* Dokument dostupný na URL http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael.pdf (apríl 2007)

[3] *Wikipedia* http://en.wikipedia.org/wiki/Galois_field (apríl 2007)

[4] Berent, A.: *Advanced Encryption Standard by Example.* Dokument dostupný na URL http://www.infosecwriters.com/text_resources/pdf/AESbyExample.pdf (apríl 2007)

[5] Ondruš, J.: *Odolnost AES proti časovací analýze.* [Semestrální práce], Brno 2006

[6] *Wikipedia* http://en.wikipedia.org/wiki/Timing_attack (apríl 2007)

[7] Bar-El, H.: *Introduction To Side Channel Attacks.* Dokument dostupný na URL http://www.hbarel.com/publications/Introduction_To_Side_Channel_Attacks.pdf (apríl 2007)

[8] *Wikipedia* http://en.wikipedia.org/wiki/Side-channel_attack (apríl 2007)

[9] *Wikipedia* http://en.wikipedia.org/wiki/L1_cache (apríl 2007)

[10] Percival, C.: *Cache missing for fun and profit.* Dokument dostupný na URL http://www.daemonology.net/papers/htt.pdf (apríl 2007)

[11] Bernstein, D.J.: *Cache-timing attacks on AES.* Dokument dostupný na URL http://cr.yp.to/antiforgery/cachetiming-20050414.pdf (apríl 2007)

[12] Osvik, D.A., Shamir, A., Tromer, E.: *Cache Attacks and Countermeasures: the Case of AES.* Dokument dostupný na URL http://www.osvik.no/pub/cache.pdf (apríl 2007)

[13] Bonneau, J., Mironov, I.: *Cache-Collision Timing Attacks Against AES.* Dokument dostupný na URL http://www.stanford.edu/~jbonneau/AES_timing_full.pdf (apríl 2007)

[14] Koeune, F., Quisquater, J..J.: *A timing attack against Rijndael.* Dokument dostupný na URL http://web.engr.oregonstate.edu/~aciicmez/osutass/data/Koeune99.pdf (apríl 2007)

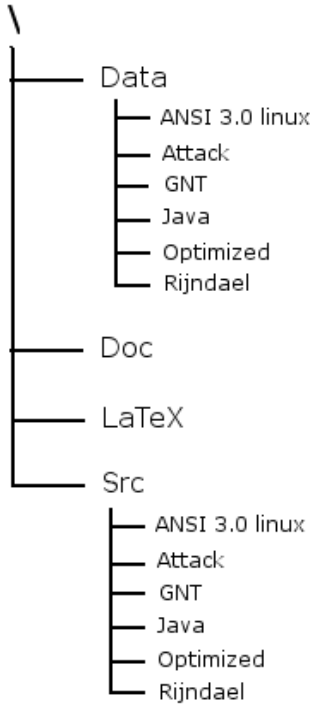
[15] Lechner, J., Tatzgern, M.: *Efficient implementation of the AES encryption algorithm for Smart-Cards.* Dokument dostupný na URL http://www.iaik.tugraz.at/teaching/10_seminare-projekte/ (apríl 2007)

[16] Lechner, J., Tatzgern, M.: *Announcing the Advanced Encryption Standard (AES).* Dokument dostupný na URL http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf (máj 2007)

[17] SoftIdea, s.r.o.: *GNT USB Token.* 2004

Dodatek A

Obsah priloženého CD



Obrázek A.1: Stromová štruktúra CD.

- **\Data** - Tento adresár obsahuje podzložky pomenované podľa implementácií a tie obsahujú .xml a .txt súbory s dátami vygenerovanými pri jednotlivý meraniach.
- **\Doc** - Tento adresár obsahuje súbory MS Excel s nameranými hodnotami v tabuľkách a s grafmi použitými v práci.
- **\LaTex** - V tejto zložke je umiestnený zdrojový kód LaTex tejto diplomovej práce vrátane výsledného pdf súboru.
- **\Src** - V tejto zložke a podzložkách sú umiestnené zdrojové kódy spolu so spustiteľnými súbormi.