PROPUNEREA UNOR SOLUŢII DE ACCELERARE A ALGORITMULUI AES CU AJUTORUL UNUI PROCESOR GRAFIC

Teză destinată obţinerii
titlului ştiinţific de doctor inginer
la
Universitatea „Politehnica” din Timişoara
în domeniul CALCULATOARE şi TEHNOLOGIA INFORMAŢIEI
de către

Ing. Tomoiagă Radu-Daniel

Conducător ştiinţific:	prof. univ. dr.ing. Stratulat Mircea
Referenţi ştiinţifici :	prof. univ. dr. ing. Victor-Valeriu Patriciu
	prof. univ. dr. ing. Daniela Elena Popescu
	prof. univ. dr. ing. Mircea Popa

Ziua susţinerii tezei: 11.03.2011

Seriile Teze de doctorat ale UPT sunt:

1. Automatică
2. Chimie
3. Energetică
4. Ingineria Chimică
5. Inginerie Civilă
6. Inginerie Electrică
7. Inginerie Electronică şi Telecomunicaţii
8. Inginerie Industrială
9. Inginerie Mecanică
10. Ştiinţa Calculatoarelor
11. Ştiinţa şi Ingineria Materialelor

Universitatea „Politehnica” din Timişoara a iniţiat seriile de mai sus în scopul diseminării expertizei, cunoştinţelor şi rezultatelor cercetărilor întreprinse în cadrul şcolii doctorale a universităţii. Seriile conţin, potrivit H.B.Ex.S Nr. 14 / 14.07.2006, tezele de doctorat susţinute în universitate începând cu 1 octombrie 2006.

România, 300159 Timişoara, Bd. Republicii 9,
tel. 0256 403823, fax. 0256 403221
e-mail: editura@edipol.upt.ro

Cuvânt înainte

Este indiscutabil faptul că domeniul tehnologiei informaţiei a devenit o componentă de bază a societăţii actuale şi că el face parte, într-o manieră din ce în ce mai evidentă, din realitatea pe care o trăim. Evoluţia permanentă a tehnologiei de înaltă performanţă a permis accesul unor categorii de utilizatori din ce mai largi şi în acelaşi timp din ce în ce mai diversificaţi. Pe de altă parte, acumulările teoretice şi experimentale au determinat mutaţii tehnologice semnificative în cele mai diverse domenii ale vieţii ştiinţifice, economice, sociale şi chiar culturale.

Actualitatea şi importanţa temei rezultă din modul cum a evoluat tehnologia modernă privind protecţia şi transmiterea informaţiilor pe diferite canale de comunicaţii publice. În acest context, este firească dorinţa fiecăruia dintre noi de a avea acces la servicii de comunicaţie cu grad ridicat de securitate. Securitatea informaţiei este un subiect mult discutat şi dezbătut în prezent. Există mai multe domenii ale securităţii informaţiei, dar indiferent de modelul de securitate ales, ca punct central, toate au ca ultimă măsură asigurarea confidenţialităţii datelor.

Lucrarea de faţă este rezultatul unei activităţi susţinute de studii şi experimente, implicând un efort considerabil, cu rezultate deosebite în domeniul protecţiei datelor. Contribuţia esenţială a tezei constă în validarea unor metode de accelerare a procesului de criptare/ decriptare a datelor.

Lucrarea se adresează tuturor celor interesaţi a cunoaşte aspectele principale legate de metode de accelerarea a primitivelor criptografice folosind procesoare grafice şi nu numai. Folosind procesoare neconvenţionale, alegând moduri de paralelizare a algoritmilor, folosind medii de dezvoltare corespunzătoare şi implementând soluţii de accelerare se pot obţine timpi de computaţie mai buni decât în cazul procesoarelor clasice. Importanţă majoră prezintă cunoaşterea arhitecturii procesorului folosit, a mediului de dezvoltare ales şi particularităţile primitivei criptografice.

Consider că lucrarea este un suport ştiinţific de luat în seamă pentru cercetările viitoare care vor avea ca subiect de pornire tematica accelerării algoritmilor criptografici folosind platforme neconvenţionale.

Timişoara, martie 2011 Tomoiagă Radu-Daniel

Mulţumiri

Alese mulţumiri şi profundă recunoştinţă se cuvin adresate conducătorului de doctorat, domnului prof. dr. ing. Stratulat Mircea, care m-a ajutat în stabilirea temei pentru această teză şi care a avut încredere în mine pentru finalizarea acestei sarcini. Totodată doresc să îi mulţumesc pentru ideile şi sfaturile oferite în perioada stagiului, pentru consilierea permanentă şi îndrumarea atentă pe tot parcursul realizării lucrării.

Adresez mulţumiri deosebite domului Prof. Dr. ing. Victor-Valeriu Patriciu, Secretar Ştiinţific al Senatului Academiei Tehnice Militare Române, doamnei Prof. Dr. ing. Daniela Elena Popescu, Şefă de Catedră la Facultatea de Inginerie Electrică şi Tehnologia Informaţiei din cadrul Universităţii din Oradea, domnului Prof. Dr. ing. Octavian Proştean, Decan al Facultăţii de Automatică şi Calculatoare din cadrul Universităţii Politehnica din Timişoara şi domnului Prof. Dr. ing. Mircea Popa, Prodecan al Facultăţii de Automatică şi Calculatoare din cadrul Universităţii Politehnica din Timişoara pentru asistenţa acordată în diferite etape ale cercetării ştiinţifice, pentru sprijinul oferit în analiza diferitelor articole, referate intermediare şi a tezei de doctorat, precum şi pentru sfaturile date de aceştia.

Doresc să mulţumesc şi domnului ş.d.l. dr. ing Groza Bogdan pentru interesul şi disponibilitatea oferită faţă de domeniul de cercetare ales de mine şi faţă de munca pe care am desfăşurat-o. De asemenea, doresc să îi mulţumesc domnului Conf. Dr. ing. Ion Tutănescu, din cadrul Universităţii Piteşti, domnului Prof. Dr. Ing. Dpl. Axel Sikora din cadrul Dualen Hochschule Baden-Württemberg Lörrach (Germania), domnului Prof. Dr. Petre Dini (IARIA Committees Board Chair) şi domnului Dr. rer. nat. Wolfgang Leister (Norvegia) pentru sprijinul oferit în analiza diferitelor articole şi a tezei de doctorat, precum şi pentru feedback-ul primit de la aceştia.

În final, aduc mulţumiri familiei mele şi tuturor celor care mi-au oferit o susţinere morală şi profesională.

Tomoiagă, Radu-Daniel

Propunerea unor soluţii de accelerare A AES prin intermediul unui GPU folosind CUDA

Teze de doctorat ale UPT, Seria 14, Nr. 1, Editura Politehnica, 2011, 145 pagini, 71 figuri, 65 tabele.

ISBN: 978-606-554-258-7
ISSN: 2069-8216
ISSN-L: 2069-8216
Cuvinte cheie: accelerare AES, paralelizare, benchmark, primitive criptografice, GPU, CUDA,
Rezumat,

Prin subiectul abordat, teza de doctorat răspunde unor probleme de maximă actualitate privind accelerarea algoritmilor criptografici folosind platforme neconvenţionale cu rolul de coprocesoare şi eliberând procesorul de sarcini. Contribuţia esenţială a tezei constă în validarea unor metode de accelerare a procesului de criptare/ decriptare a datelor. S-au utilizat facilităţile oferite de procesoarele grafice în vederea reducerii timpilor de execuţie obţinuţi în urma protejării datelor prin criptarea acestora. Pentru a putea valida soluţiile de accelerare s-a realizat un set de teste ale primitivelor criptografice pentru a analiza performanţa acestora şi s-a elaborat o sinteză despre stadiul actual al domeniului coprocesării primitivelor criptografice prin folosirea de plăci video, ASIC şi FPGA.

CUPRINS

1. Introducere

Securitatea informaţiei este un subiect mult discutat şi dezbătut. Există multe domenii ale securităţii informaţiei, mulţi experţi în aceste domenii şi multe studii, analize şi proiecte de cercetare aferente acestor domenii. Putem exemplifica diverse domenii de securitate: securitatea reţelelor cu subdomenii, securitatea calculatoarelor (de asemenea, cu subdomenii), securitatea documentelor (cu subdomenii), ramificări şi intercalări între acestea. Există, desigur, şi alte domenii, cum este, de exemplu, criptografia asupra căreia ne vom îndrepta atenţia în această lucrare. Indiferent de modelul de securitate ales, ca punct central, toate au ca ultimă măsură de asigurare a confidenţialităţii criptarea datelor. Plecând de la [SHO08], [SCH07], [BAM01], [BRE89], [FLE07], [MOU07] şi [ASH07] s-a realizat în [TOM09/1] o analiză a diferitelor modele existente şi s-a propus un model de securitate care are ca scop protejarea datelor clasificate ţinând cont şi de cadrul legislativ naţional. Acest model se bazează pe mai multe straturi de protecţie, iar punctul central este criptografia, care are rolul de asigurare a confidenţialităţii datelor [TOM09/2]. Subiectul pe care îl vom analiza este evaluarea performanţelor unui set de primitive criptografice din punctul de vedere al timpului computaţional necesar rulării acestora. Practic, se vor realiza benchmark-uri pentru a putea obţine rezultate, iar pe baza acestor rezultate, se vor genera grafice şi se vor trage concluzii. Pentru a putea înţelege mai bine temele discutate în această lucrare, se vor descrie algoritmii folosiţi, modul de implementare al acestora în diferitele medii de programare şi platformele care au stat la baza acestor teste.

Scopul acestei lucrări constă în propunerea a două soluţii de accelerare a algoritmului de criptare simetrică AES, prin adaptarea acestuia să ruleze pe un procesor grafic. Pentru a putea crea o imagine asupra comportamentului primitivelor criptografice pe procesoare clasice, în prima parte a lucrării se vor realiza teste cu rolul de a evalua performanţele diferitelor primitive criptografice. În urma unui număr mare de teste efectuate pe un număr mare de primitive criptografice se vor obţine rezultate care vor fi folosite pentru stabilirea unui clasament în rândul procesoarelor testate şi al mediilor de dezvoltare alese. Se va putea compara performanţa obţinută pentru platformele clasice cu cea obţinută în urma implementării algoritmului pe GPU. Evaluarea performanţelor se va face din punctul de vedere al timpului necesar rulării algoritmilor. Platformele alese şi sistemele de test sunt prezentate în subcapitolul 3.1.4.

Efectuarea acestor teste a fost realizată din necesitatea de a ne forma o imagine asupra comportamentului platformelor tradiţionale. Totodată, s-a încercat o diferenţiere între mediile de programare. În paralel cu testele acestea, s-a creat şi o comparare între sisteme de operare. S-a realizat astfel o implementare unică din punctul de vedere al volumului mare de teste, al mediilor de programare testate, al numărului de platforme şi al varietăţii componentelor hardware ale acestora. O altă trăsătură unică a testelor efectuate constă în diversitatea valorilor de intrare.

Scopul rulării acestor teste a fost acela de a observa care dintre platformele, sistemele de operare şi limbajele folosite oferă performanţe mai bune.

Necesitatea rulării testelor a constat în obţinerea unei imagini de ansamblu asupra primitivelor criptografice şi a posibilităţii obţinerii de performanţe sporite pe un anumit tip de platformă (procesor) sau mediu de dezvoltare. În majoritatea lucrărilor analizate ([SOL08], [KAH08], [KIP09], [LUK09], [MAN07], [NAO95], [NAO96], [YON08] , [ZHA04], [GRO09], [SCH96], [GUS95], [HAI03], [HIN06], [JAR10]) s-a constatat că testele au fost fie realizate pe un număr mic de date de

intrare, fie pe un număr mic de algoritmi, fie pe un număr mic de platforme. Niciunul din testele efectuate în materialele amintite nu a acoperit întreaga arie de teste, de algoritmi, de platforme care au fost realizate în lucrarea de faţă. În [BAR08] s-a propus un model de securitate folosind AES, dar nu s-a realizat un test pentru evaluarea performanţelor maşinilor pe care se instalează aplicaţia.

Lucrarea de faţă aduce ca noutate, în comparaţie cu alte lucrări similare, volumul mare de date de intrare, numărul mare de algoritmi testaţi, numărul mare de platforme pe care s-au executat testele şi varietatea mediilor de dezvoltare.

În urma testelor din prima parte se va analiza posibilitatea alegerii unei platforme pentru adaptarea algoritmului AES astfel încât, în urma rulării, să obţină timpi de computaţie mai mici. Dacă nicio platformă nu poate oferi performanţe mai bune pe întregul set de teste se va identifica o platformă neconvenţională care să fie capabilă să ofere timpi de computaţie mult mai buni. Algoritmul AES va fi adaptat să ruleze pe o astfel de platformă, obiectivul final fiind ca performanţa oferită să fie mult mai bună, ca platforma aleasă să degreveze procesorul de sarcini, rezultatele obţinute să fie comparabile cu soluţii similare, iar costurile unei astfel de implementări să fie mici în comparaţie cu soluţiile comerciale profesionale.

Această lucrare este structurată pe 6 capitole. În primul capitol se face o scurtă introducere, punându-se accentul pe aspecte generale privind benchmark-urile, metrici şi evaluarea performanţelor.

În capitolul al doilea se prezintă primitivele criptografice, clasele, funcţiile şi metodele utilizate în efectuarea testelor. S-a realizat o taxonomie a acestor primitive după cum urmează:

- HMAC sau algoritmi bazaţi pe autentificarea mesajelor
  - HMACSHA1
  - HMACSHA256
  - HMACSHA512
  - HMACMD5
- Funcţii hash
  - SHA1
  - SHA256
  - SHA512
  - MD5
- Algoritmi simetrici
  - DES
  - 3DES
  - AES

În acelaşi capitol se va detalia şi justifica alegerea acestor primitive criptografice.

Capitolul 3 conţine descrierea detaliată a aplicaţiei şi a mediilor de testare. Astfel, se descriu platformele de testare, mediile de dezvoltare utilizate şi staţiile pe care se realizează testele. Capitolul 3 este structurat pe 3 subcapitole în care se prezintă mediul de programare Visual C# şi Visual Basic.Net (Visual Studio 2008) şi Java de pe platforma Windows (Windows XP SP3 si Windows 2000 SP4, atât pe platforma Windows cât şi pe platforma Unix şi bibliotecile OpenSSL implementate pe platforma Unix.(Ubuntu) şi se trag concluzii.

Testele din acest capitol au fost efectuate pe 5 platforme distincte, sub 3 medii diferite (C#,VB,OpenSSL) şi sub 3 sisteme de operare diferite (Windows XP SP3, Windows 2000 SP4 si UNIX). Astfel de teste au mai fost făcute ([S0L08] şi alţii), dar majoritatea benchmark-urilor au fost efectuate pentru dimensiuni foarte mici de intrare, urmărindu-se măsurarea timpilor de computaţie pentru funcţii care preiau

date din RAM. De exemplu, [SOL08] a avut ca intrări 0...80 bytes. Testele din această lucrare au fost efectuate pe fişiere de mari dimensiuni, şi, separat, au fost realizate teste cu acelaşi set de funcţii, dar pentru date de valori mici care sunt preluate din RAM, programele fiind scrise în Visual Basic, C# şi JAVA.

Platformele alese au fost următoarele: un sistem portabil, un sistem foarte performant, un sistem cu performanţe mai slabe, un sistem performant, un sistem ultramobil (netbook) şi un sistem mediu ca performanţe, fiecare având caracteristicile descrise în subcapitolele 3.1.4 şi 3.2.4. Astfel, s-au ales procesoare de categorii, frecvenţe şi clase diferite (procesor Hyper Threading, Dual Core Mobile, Core 2 Duo, Atom şi un Quad CORE), memorii de latenţe şi capacităţi diferite, şi hard disc-uri de producători, respectiv capacităţi, diferite. Aceste elemente au un impact foarte mare asupra criptării.

Aplicaţiile sub Windows au fost dezvoltate folosind biblioteci standard puse la dispoziţie de limbajele de programare. A fost aleasă aceasta variantă, deoarece algoritmii din biblioteci sunt standardizaţi. Implementarea unor algoritmi proprii ar fi impus oricum şi testarea algoritmilor standard pentru a vedea care sunt mai performanţi, iar în lipsa unui spor de performanţă, nu s-ar fi justificat. Numărul algoritmilor fiind mare (11), dezvoltarea acestora pe 3 medii diferite (total 33) ar fi fost o muncă de foarte mare durată, iar scopul lucrării nu era dezvoltarea de aplicaţii, ci testarea algoritmilor pe fişiere de mari dimensiuni.

Spre deosebire de alte abordări, în testele efectuate s-a avut în atenţie şi cazul în care datele sunt de dimensiuni mari şi provin de pe diverse suporturi de stocare cu timpi de acces diferiţi. În cazurile de teste efectuate pe intrări de mici dimensiuni, performanţele algoritmilor ţin doar de viteza de procesare şi rapiditatea memoriilor. Acest lucru se schimbă în cazul volumelor mari, deoarece transferul de date şi procesarea lor este afectată şi de alte echipamente, astfel, rezultatele fiind „reale" pentru utilitatea practică în cazul criptării de volume mari. În cazul criptărilor din RAM, sau a datelor „on the fly" partea de „realitate" o dau aplicaţiile de criptare pentru Instant Message, Voice Over IP, etc.

În capitolul 4 se găsesc rezultatele testelor efectuate în urma implementării algoritmilor prezentaţi în capitolul 2, în mediile de dezvoltare şi pe platformele prezentate în capitolul 3. În finalul capitolului, se vor trage concluzii în urma analizării rezultatelor şi graficelor prezentate, concluzii legate de timpul de execuţie necesar algoritmilor criptografici. Concluziile privesc compararea performanţelor primitivelor criptografice testate pe diferitele medii de programare şi platforme. În urma acestor concluzii s-a constatat necesitatea identificării unei soluţii/ platforme care să ofere performanţe mai bune decât cele testate. S-a ales un singur algoritm AES, iar ca platformă s-a ales un procesor grafic. Algoritmul ales a fost AES, deoarece acesta reprezintă standardul actual pentru criptarea simetrică.

În capitolul 5 se prezintă soluţiile propuse de accelerare a algoritmului AES pe un GPU. Se realizează o prezentare a plăcii grafice folosite, a arhitecturii acesteia şi a mediului de dezvoltare CUDA. Apoi, se prezintă posibilităţile de implementare ale AES pe o astfel de platformă, se realizează analize pentru a găsi cea mai bună optimizare a lui AES pe procesorul grafic, folosind CUDA. Se prezintă avantajele şi dezavantajele celor două metode alese pentru implementare. În urma implementării s-au testat cele două metode. Rezultatele obţinute au fost împărţite pe două categorii: date din memorie şi date de mari dimensiuni. În urma acestor teste, s-a ales una din cele două metode pentru a se realiza o extensie a OpenSSL cu scopul de a beneficia de accelerarea AES pe CUDA.

Testele din capitolul 5 au fost efectuate pe platforma care avea o placă video capabilă să ruleze cod C pentru CUDA.

Rezultatele obţinute în capitolul 5 sunt analizate în capitolul 6 care conţine concluziile întregii lucrări şi se prezintă ceea ce s-a realizat până în acest moment şi ce urmează să se realizeze în cercetarea viitoare.

În finalul lucrării sunt prezentate contribuţiile personale şi posibile direcţii de cercetare. O mare parte din contribuţii au fost validate prin publicarea a 10 lucrări ştiinţifice, la care autorul este prim autor după cum urmează:

- o lucrare publicată în volumul unei conferinţe internaţionale indexate ISI Proceedings;
- 4 lucrări publicate în volumele unor conferinţe internaţionale indexate IEEExplore;
- 5 lucrări publicate în volumele unor conferinţe internaţionale neindexate.

1.1. Aspecte generale despre benchmark-uri

Putem defini noţiunea de benchmark (în domeniul IT) ca fiind rezultatul rulării unui program de calculator, sau a unui set de instrucţiuni/ algoritmi, având ca scop evidenţierea performanţelor unui obiect, prin efectuarea unor seturi de teste considerate standard asupra acestuia. Având în vedere dezvoltarea domeniului IT&C şi a varietăţii software-ului se poate afirma că este greu de realizat un benchmark specific pentru o anumită categorie. Astfel, realizarea unui benchmark pentru evaluarea performanţei rămâne un subiect discutabil.

S-a observat că este mai uşor să se dezvolte aplicaţii de benchmark orientate pe anumite nişe atingând puncte izolate. Un avantaj care ar putea veni în sprijinul benchmark-urilor este simplitatea rezultatelor obţinute şi prezentate în urma benchmark-ului, acestea fiind mai uşor de înţeles. Orice componentă, fie ea software sau hardware, este achiziţionată pe baza caracteristicilor pe care le prezintă, caracteristici obţinute şi prezentate în urma unor benchmark-uri (frecvenţă procesor, capacitate memorie, capacitate hard disc, număr de pagini pe minut, timp de răspuns, suprafaţă acoperită, lăţime de bandă, etc.).

Un ajutor în dezvoltarea benchmark-urilor este adus de consorţiul SPEC (Standard Performance Evaluation Cooperative) [SPE98] şi TPC (Transactions Processing Council) [TRA98], care s-a înfiinţat în anul 1998. Acesta pune la dispoziţie benchmark-uri şi ghiduri de realizare a testelor pentru a putea îmbunătăţi calitatea acestora. Acestea din urmă au apărut ca reacţie la faptul că multe programe de benchmark erau realizate incorect, sau erau folosite în mod eronat, ajungându-se în final la rezultate greşite.

În general, un benchmark execută un număr finit de instrucţiuni. Sistemul care finalizează acel set de instrucţiuni în timpul cel mai scurt este plasat în topul testului. În antiteză, putem prezenta modele de benchmark care nu necesită finalizarea testului într-un interval de timp, ci sunt axate pe calculul volumului de muncă. În [SOL08] se vorbeşte despre un benchmark denumit HINT [GUS95] [HIN06] care nu face parte din nicio categorie prezentată mai sus.

Pentru a ţine pasul cu dezvoltarea tehnologică, benchmark-urile trebuie actualizate periodic în scopul obţinerii unor rezultate corecte. Există tendinţa unor producători de a vinde produsele specificând rezultatele obţinute pe anumite seturi de teste care avantajează produsul respectiv. Mai mult decât atât, se proiectează produse pentru a rula mai bine pe anumite benchmark-uri cu scopul de a obţine rezultate mai bune decât alte produse concurente, sau chiar se dezvoltă benchmark-

uri care rulează în timp mai scurt, pe anumite produse, pentru a le avantaja pe piaţă.

Pentru a putea interpreta corect rezultatele unui benchmark este necesară înţelegerea algoritmilor folosiţi, a funcţiilor şi a metricilor de evaluare a performanţei.

O implementare interesantă este o extensie a OpenSSL care să ofere suport pentru puzzle-uri criptografice prezentată în [CRI09]. În această lucrare, autorul a dezvoltat o extensie a OpenSSL care oferă suport pentru puzzle-uri criptografice, a realizat o interfaţă grafică pentru testarea puzzle-urilor criptografice şi a prezentat statistici cu privire la timpul de rezolvare a acestor puzzle-uri.

1.2. Metrici şi evaluarea performanţei

„Performanţele se stabilesc prin luarea în considerare a duratei în care se obţine soluţia finală şi a consumului de resurse necesare pentru rezolvarea integrală a problemelor curente. Nivelul performanţei este influenţat de modul în care produsului program i s-au asigurat calitatea de concepţie-proiectare, calitatea de execuţie, calitatea de conformitate, capacitatea de utilizare curentă şi capacitatea de mentenanţă”. [ZUG04].

Indicii de calitate sunt organizaţi pe mai multe nivele: caracteristici, atribute, metrici şi elemente de evaluare [ZUG04].

Caracteristicile de calitate ale unui sistem sunt condiţionate de atributele de calitate, iar fiecare atribut se măsoară prin una sau mai multe metrici. Unei metrici îi corespund unul sau mai multe elemente de evaluare [ZUG04].

Costurile ridicate care sunt caracteristice procesului de dezvoltare a produselor software sunt corelate performanţelor acestora. Performanţele sunt stabilite prin luarea în considerare a timpului în care este obţinută soluţia finală şi rezultatele problemei propuse. Nivelul performanţei este direct influenţat de calitatea produsului software. Calitatea unui program de calculator poate fi împărţită în mai multe subcategorii după cum urmează: calitatea de proiectare, calitatea de execuţie, calitatea de conformitate, uşurinţa de utilizare şi posibilitatea de mentenanţă.

Evaluarea performanţei se face, în general, în funcţie de: caracteristici, atribute, metrici şi elemente de evaluare.

Calitatea este determinată de atributele de calitate. Fiecare atribut se măsoară prin una sau mai multe metrici, iar unei metrici îi corespund unul sau mai multe elemente de evaluare.

Conform [STE06], evaluarea performanţei se poate face prin măsurarea timpului. Când vorbim de timp, ne referim la timpul de răspuns sau la timpul de execuţie. Aşadar, evaluăm diferenţa dintre începutul şi sfârşitul unui eveniment. G. Ştefănescu defineşte performanţa în [STE06], ca fiind (Timp de execuţie)$^{-1}$. Pentru măsurarea timpului de execuţie se poate folosi ceasul calculatorului, mai exact perioada ceasului care se măsoară în nanosecunde sau frecvenţa ceasului măsurată în megahertzi. O evaluare a unui sistem informatic se poate face în funcţie de două noţiuni şi anume MIPS (Milions Intructions Per Second) şi FLOPS/ MFLOPS (Floating Point Operations Per Second). Ca măsură de bază în evaluarea performanţei rămâne timpul.

Sora, I., în [SOR08], discută despre algoritmi pentru calcul paralel tratând într-un capitol evaluarea performanţelor şi a metricilor în programele paralele. În această prezentare se iau în considerare ca metrici de performanţă timpul de

execuţie, costul, eficienţa, accelerarea, suprasarcina, urmărindu-se câştigul de performanţă în programarea paralelă faţă de cea secvenţială. Spre exemplu, se împarte timpul de execuţie în două componente: timpul de execuţie secvenţial (definit ca timpul de la începutul şi până la sfârşitul execuţiei programului pe un calculator secvenţial) şi timpul de execuţie paralel (adică timpul de la începutul execuţiei până la terminarea ultimului fir de execuţie paralel). O altă metrică de performanţă poate fi suprasarcina (în cazul programării paralele - Total Parallel Overhead) care se defineşte ca fiind diferenţa dintre timpul total de lucru însumat al tuturor procesoarelor şi timpul necesar celui mai rapid algoritm secvenţial. Timpul total de lucru însumat al tuturor procesoarelor se defineşte ca fiind timpul de calcul la care se adaugă timpul de comunicare şi timpul de inactivitate temporară cumulat pentru toate procesoarele [SOR08].

O metrică aparte o constituie accelerarea. Aceasta a apărut ca răspuns pentru întrebarea : „De câte ori se rezolvă mai repede un algoritm în paralel decât cel secvenţial?” Accelerarea este definită ca raportul dintre timpul necesar rulării algoritmului pe un procesor şi timpul necesar rezolvării algoritmului în paralel pe mai multe procesoare identice. Trebuie specificat faptul că accelerarea este o metrică ce evaluează performanţa algoritmului şi nu a sistemului de calcul în paralel [SOR08]. O altă metrică ce se referă tot la procesoare paralele este eficienţa şi se defineşte ca raportul dintre accelerare şi numărul de procesoare. Valoarea acesteia, în mod normal, este subunitară şi atinge valoarea 1 doar în cazul ideal când sistemul paralel nu prezintă Overhead. Mai poate fi introdusă o metrică, şi anume costul. Acesta este de fapt timpul total de lucru însumat al tuturor procesoarelor. Putem astfel reveni, pentru a exprima eficienţa ca fiind timpul computaţional în cazul secvenţial raportat la cost. În cazul în care costul rezolvării problemei cu ajutorul algoritmului paralel este egal cu timpul de execuţie al celui mai rapid algoritm secvenţial, se spune că algoritmul paralel este optimal în cost (sau costul este optimal) [SOR08].

Buligiu, I., în [BUL06], împarte metricile în mai multe categorii: metrici de evaluare privind viteza de procesare şi timpul de răspuns, metrici care studiază fluxurile de transfer din sistem şi între componentele sale, metrici referitoare la siguranţa sistemelor, metrici privind disponibilitatea sistemelor şi metrici de scalabilitate. În cazul vitezei de procesare, una din metricile folosite este timpul de răspuns al sistemului. În cazul unor teste/ benchmark-uri este indicat să se calculeze o valoare medie a timpului de răspuns, valoare care se obţine ca medie a timpilor măsuraţi pe un număr mare de teste efectuate. În funcţie de sistemele care necesită să fie evaluate, metrica este adaptată cerinţelor cazului respectiv. În cazul testelor algoritmilor criptografici pe fişiere de mari dimensiuni, timpul necesar finalizării unui proces este mare, iar în cazul particular al criptării simetrice unde se foloseşte un algoritm de tip bloc, fiecare bloc din fişier este citit criptat şi scris în fişierul de ieşire. Astfel, avem un număr mare de iteraţii cu intrări diferite care sunt procesate secvenţial. Numărul de iteraţii depinde de dimensiunea fişierului şi se obţine ca raport dintre dimensiunea fişierului şi dimensiunea blocului de intrare. În acest caz, obţinem un timp ce reflectă mai mult realitatea practică a unei aplicaţii de criptare a informaţiilor de dimensiuni mari ce sunt stocate pe un suport de memorie sau hard disc, iar acest timp este teoretic mai mare decât cel preconizat pentru rularea algoritmului doar pe valori imediate care sunt accesibile din memoria internă, deoarece la acesta se mai adaugă şi timpul de citire/ scriere de pe hard disc. În partea a doua a testelor, se obţin timpii de rulare pentru algoritmi atunci când datele de intrare sunt în memorie, nefiind aduse de pe un suport extern. În acest caz, o simplă parcurgere a algoritmului nu este suficientă. Algoritmii se repetă de un număr mai mare de ori : 100.000 în cazul [SOL08] şi 1.000.000 în cazul de

faţă. În urma rulării, se va calcula media şi astfel se obţine un rezultat mediu care reflectă timpul necesar rulării algoritmului. Testarea timpilor de calcul al funcţiilor criptografice s-a realizat în cazul [SOL08] pe dimensiuni de 0 bytes, 26 bytes, 62 bytes şi 80 bytes, iar în lucrarea de faţă pe dimensiunea de 16 bytes. Un alt benchmark este cel realizat de B. Groza în [GRO09]. În acest material se discută despre implementarea în Java a unui protocol de autentificare pentru aplicaţii pe telefoane mobile. Se testează atât algoritmii HMAC cât şi cei hash implementaţi în această lucrare. Platforma de test a fost un telefon Nokia 6288, sistemul de operare fiind multitasking, iar numărul de repetiţii a fost de 100 de ori. Pentru ca maşina virtuală să nu fie influenţată de alte task-uri, testele au fost efectuate în profilul „flight mode", deoarece toate funcţiile de comunicaţii sunt oprite.

Un alt tip de metrici pentru măsurarea vitezei sunt metricile de latenţă, care măsoară timpii de aşteptare sau de întârziere pentru un sistem sau o componentă [BUL06].

Pocatilu, P. prezintă în [POC00] mai multe modele. Primul dintre acestea este cel de complexitate al sistemului informatic McCabe, prezentat şi în [MCA96], acesta referindu-se la evaluarea complexităţii programelor. Un al doilea model ar fi acela de performanţă COCOMO (COnstructive COst MOdel). Acest model se referă la estimarea costurilor, a efortului şi a timpului necesar pentru dezvoltarea unui soft. Mai este amintit şi modelul Halstead care are ca ţintă evaluarea complexităţii calculelor din punctul de vedere al codului sursă.

Pentru a înţelege sistemul de proiectare şi pentru a ajunge la un design cât mai bun este necesară modelarea performanţei încă de la început [SOL08].

Kahate, A. în studiul efectuat referitor la impactul algoritmilor criptografici asupra performanţei aplicaţiilor [KAH08], concluzionează că, indiferent de algoritm, timpul necesar criptării sau decriptării este aproape la fel şi că dimensiunea folosită la intrare nu are un impact major în timpul necesar computaţiei. Algoritmii folosiţi au fost de tip message digest (MD5, SHA1, SHA 512), algoritmi simetrici (AES, 3DES, Blowfish) şi algoritmi asimetrici (RSA). Lungimea intrării a variat între 14 şi 203 caractere. În lucrarea sa nu există detalii legate de modul în care s-au realizat testele, dacă a fost folosită iterarea şi de câte ori. Cert este că lungimea de intrare este prea restrictivă şi prea mică pentru a putea crea o imagine clară asupra impactului acesteia în cazul aplicaţiilor criptografice.

Schneier, B., în [SCH96], stabileşte un etalon al performanţelor algoritmilor criptografici după cum urmează: pentru SHA1, 75000 bytes/secundă, pentru DES, 45000 blocuri de 64 de biţi pe secundă, iar pentru RSA, criptarea se realizează în 0.03 secunde, decriptarea în 0.16 secunde, semnătura digitală în 0.16 secunde şi verificarea în 0.02 secunde. Desigur, acest etalon nu mai reprezintă valori reale, fiind stabilit în 1996.

2. Primitive criptografice

Prin primitive criptografice se înţeleg algoritmi criptografici bine stabiliţi, prezentaţi în literatura de specialitate şi în standarde. Solga, M., în [SOL08], realizează o evaluare a performanţelor pentru funcţii criptografice folosind patru categorii de primitive criptografice: funcţii MAC, funcţii hash, algoritmi simetrici şi algoritmi asimetrici. În lucrarea de faţă au fost alese doar trei categorii şi anume: funcţii MAC, funcţii hash şi algoritmi simetrici. În capitolul acesta se prezintă primitivele criptografice alese. Alegerea acestora a fost făcută pe baza taxonomiei realizate de Schneier, B., în [SCH03] şi de Menezes, A., în [MEN96]. S-a plecat de la NSA Suite B Cryptography [NSA10] care recomandă în vederea criptării simetrice folosirea algoritmului AES cu cheie de 128 de biţi sau 256 de biţi, iar pentru algoritmii hash familia SHA (SHA 256 şi SHA 384). Acestea sunt folosite pentru protecţia informaţiilor clasificate până la nivelul SECRET. În [BLU10] se recomandă folosirea algoritmilor AES 128 biţi şi SHA 256 pentru protecţia informaţiilor clasificate până la nivelul SECRET şi AES 256 biţi şi SHA384 pentru protecţia informaţiilor clasificate până la nivelul TOP SECRET. Totodată, în raportul D.SPA.13, din cadrul ECRYPT II [ECR10] sunt recomandaţi algoritmi criptografici (cifru bloc, funcţii hash, scheme de semnare, etc.) şi dimensiunea cheilor corespunzătoare pentru a atinge obiectivele de securitate specificate. SHA384 a fost omis deoarece, conform [SCH03], este nefolositor din cauza faptului că, pentru obţinerea rezultatului se rulează algoritmul bazat pe SHA512, iar apoi se renunţă la un număr de biţi. [TOM10/6] a constituit punctul de plecare al acestui demers de a evalua performanţele primitivelor criptografice. În alegerea primitivelor criptografice s-a plecat de la analiza realizată de [SOL08]. S-a ţinut cont de recomandările [NSA10], [ECR10] şi [BLU10] pentru algoritmii simetrici şi funcţiile hash. În stabilirea finală a algoritmilor din teste s-au avut în vedere şi clasificările din [SCH03] şi [MEN96].

Algoritmii aleşi au fost:

- HMAC sau algoritmi bazaţi pe autentificarea mesajelor
  - HMACMD5. Acesta nu mai prezintă nivelul de securitate recomandat, dar a fost ales pentru teste deoarece a fost folosit intens în trecut şi mai este folosit şi acum.
  - HMACSHA1. Acesta nu mai prezintă nivelul de securitate recomandat, dar a fost ales pentru teste deoarece a fost folosit intens în trecut şi mai este folosit şi acum.
  - HMACSHA256. Acesta reprezintă nivelul de securitate recomandat şi de aceea a fot ales pentru testele din această lucrare.
  - HMACSHA512. Acesta depăşeşte nivelul de securitate recomandat şi de aceea a fost ales pentru testele din această lucrare.
- Funcţii hash criptografice
  - MD5. Acesta nu mai prezintă nivelul de securitate recomandat, dar a fost ales pentru teste deoarece a fost folosit intens în trecut şi mai este folosit şi acum.
  - SHA1. Acesta nu mai prezintă nivelul de securitate recomandat, dar a fost ales pentru teste deoarece a fost folosit intens în trecut şi mai este folosit şi acum.

- SHA256. Acesta reprezintă nivelul de securitate recomandat în [NSA10], [ECR10], [BLU10] şi de aceea a fot ales pentru testele din această lucrare.
- SHA512. Acesta depăşeşte nivelul de securitate recomandat în [NSA10], [ECR10], [BLU10]. SHA384 este cel recomandat, dar acesta a fost omis deoarece, conform [SCH03], este nefolositor din cauza faptului că, pentru obţinerea rezultatului se rulează algoritmul bazat pe SHA512.

- Algoritmi simetrici
  - DES. Acesta nu mai prezintă nivelul de securitate recomandat, dar a fost ales pentru teste deoarece a reprezentat standardul criptării simetrice în trecut şi mai este folosit şi astăzi.
  - 3DES. Acesta nu mai prezintă nivelul de securitate recomandat, dar a fost ales pentru teste deoarece a reprezentat standardul criptării simetrice în trecut şi mai este folosit şi astăzi.
  - AES. Acesta reprezintă standardul criptării simetrice la momentul actual şi nivelul de securitate recomandat în [NSA10], [ECR10], [BLU10]. În testele efectuate în prima parte a lucrării a fost ales AES cu cheie de 128 de biţi.

Conform [CISCO SEC], performanţele primitivelor alese sunt prezentate în tabelul 2.1.

Primitivă	**Observaţii**	**Viteză**
Algoritmi simetrici		
DES	Resurse medii	Viteză medie
3DES	Resurse medii	Viteză mică
AES	Resurse puţine	Viteza mare
HMAC		
HMACMD5	Resurse puţine	Viteza mare
HMACSHA1	Resurse medii	Viteză medie
HMACSHA256	Resurse medii	Viteză mică
HMACSHA512	Resurse medii	Viteză mică
hash		
MD5	Resurse puţine	Viteza mare
SHA1	Resurse medii	Viteză medie
SHA256	Resurse medii	Viteză mică
SHA512	Resurse medii	Viteză mică

Tabel 2.1 Primitivele criptografice-performanţe[CISCO SEC]

O categorie aparte o constituie algoritmii asimetrici. Aceştia au la bază noţiunea de Secret Sharing ([SZA79], [FRE00]). În această lucrare nu au fost testaţi algoritmi asimetrici, deoarece aceştia sunt concepuţi pentru a fi folosiţi în criptarea datelor de mici dimensiuni, iar testele propuse de noi în lucrarea de faţă evaluează, pe lângă date de mici dimensiuni, şi volume mari de date care ajung la 10 Gb. Se poate spune că algoritmii asimetrici sunt deseori folosiţi pentru criptarea cheilor algoritmilor simetrici ce criptează datele.

Conform [SIN00] viitorul criptografiei îl constituie criptografia cuantică.

2.1. Algoritmi HMAC (coduri de autentificare a mesajelor)

Algoritmii bazaţi pe coduri de autentificare a mesajelor (MAC- message authentification code) sunt algoritmi care primesc la intrare o cheie secretă şi un şir

de lungime variabilă care necesită autentificare şi care va genera la ieşire un şir de lungime fixă. Acest şir de lungime fixă este numit MAC. O categorie din această clasă sunt algoritmii HMAC. Un algoritm MAC construit folosind primitive criptografice de tipul funcţiilor hash este numit HMAC [SCH08]. Algoritmii implementaţi în partea practică din această categorie sunt :

- HMACSHA1
- HMACSHA256
- HMACSHA512
- HMACSHAMD5

De exemplu, pentru a obţine MAC-ul unui mesaj M, folosind o cheie k şi un algoritm hash se utilizează formula MAC=h(k‖M) [MAO03]. Această metodă este considerată banală, şi este necesar ca în proces să se mai calculeze funcţia hash aplicată asupra rezultatului formulei amintite [AND08]. HMAC au fost definite în FIPS 198, iar apoi au fost înlocuite de FIPS 198-1.

Având în vedere că un MAC (sau HMAC) este bazat pe o primitivă criptografică şi pe o cheie secretă, este uşor de înţeles că securitatea unui MAC rezidă în securitatea algoritmului criptografic şi a cheii [MEL00].

Într-un studiu de caz, [MEL00] prezintă un tabel în care ordonează, în funcţie de viteza de execuţie, mai mulţi algoritmi criptografici. Concluzia la care se ajunge este aceea că MAC se execută mai lent decât algoritmii fără cheie. Astfel, s-au propus mai multe soluţii pentru sporirea performanţei algoritmilor MAC. În acest sens, o lucrare publicată în 1996 de Mihir Bellare este punctul de plecare pentru deja standardizatele HMAC-uri. Un alt aspect care vine în sprijinul funcţiilor HMAC este acela că protocoale precum SSL şi IPsec folosesc HMAC-uri standardizate [MEN96].

OpenSSL are implementate ca funcţii standard ale aplicaţiei mai multe primitive criptografice. Printre aceste primitive sunt şi funcţiile HMAC care pot fi folosite, deşi acestea nu sunt documentate în manualele aplicaţiei sau manualele online [www8].

Scopul algoritmilor HMAC este acela de a autentifica atât sursa cât şi integritatea mesajului fără a mai fi nevoie de alte mecanisme [FIPS198-1].

2.2. Algoritmi simetrici

Criptografia datelor se bazează pe algoritmi criptografici. Aceşti algoritmi sunt de mai multe tipuri. În cazul în care pentru procesul de criptare şi pentru procesul de decriptare se foloseşte aceeaşi cheie atunci algoritmul se numeşte simetric (algoritm cu cheie simetrică). Dacă în procesul de criptare se foloseşte o cheie, iar în procesul de decriptare se foloseşte altă cheie, atunci vorbim despre criptare asimetrică. Având în vedere faptul că un algoritm asimetric este un mare consumator de putere de calcul, se obişnuieşte să se uziteze criptarea asimetrică pentru mesaje de dimensiuni reduse, sau este folosit pentru criptarea unei chei, iar această cheie este folosită pentru criptare simetrică.

Criptarea simetrică se bazează pe un cifru. Cifrul poate să fie de tip bloc sau flux (*block cipher,* respectiv *stream cipher*). Când vorbim despre criptare o putem descrie ca pe o funcţie care este cunoscută, funcţie ce aplică o transformare asupra mesajului de intrare folosind în proces şi cheia pentru a obţine ca rezultat criptotextul [GHE05]. Având în vedere acest criptosistem, securitatea acestuia rezidă în păstrarea secretă a cheii. Pentru a obţine din nou mesajul din criptotext se foloseşte transformarea inversă împreună cu aceeaşi cheie. Cheia nu trebuie să fie identică cu cea de la criptare, dar este necesar ca una să fie uşor de dedus din cealaltă [GRO08].

Pentru partea practică am ales următorii algoritmi simetrici:

- DES- Data Encryption Standard
- 3DES- Triple DES
- AES- Advanced Encryption Standard

Aceşti algoritmi au fost aleşi deoarece au reprezentat standardele în domeniu (DES, 3DES) şi reprezintă standarde actuale (AES).

DES foloseşte chei cu dimensiuni de 56 de biţi şi dimensiunea blocului de 64 de biţi. Date fiind caracteristicie algoritmului, putem spune că acesta nu mai îndeplineşte condiţiile de securitate ale tehnicii moderne [KES07]. Un sistem de securitate suficient pentru această perioadă este un sistem care asigură o securitate de minim 128 de biţi [SCH03]. Ca reacţie la acest neajuns al lui DES a fost dezvoltat 3DES care are un bloc de criptare alcătuit din trei blocuri DES secvenţiale. Dezavantajul teoretic al lui 3DES este că lucrează la o treime din viteza lui DES. 3DES foloseşte o etapă de criptare DES, apoi o a doua etapă de decriptare DES şi a treia etapă de criptare DES, în fiecare din etape fiind folosite chei diferite [SCH07]. Dacă pentru criptare 3DES se folosesc criptare-decriptare-criptare DES atunci pentru decriptare 3DES se folosesc decriptare-criptare-decriptare DES cu cheile folosite în faza de criptare [CISCO NS]. Algoritmul DES foloseşte 16 runde (etape) cu 16 chei pe etapă de 48 de biţi. Viteza de criptare/decriptare a lui DES este una medie, iar consumul de resurse este tot unul mediu, pe când viteza lui 3 DES este una mică, iar consumului de resurse unul mediu [CISCO SEC].

AES este noul standard recomandat de guvernul S.U.A. şi de Agenţia de Securitate Naţională a S.U.A. (National Security Agency, NSA). Acesta nu a fost dezvoltat în regie proprie, ci a fost lansat un concurs în cadrul comunităţii criptografice, în urma căruia a fost ales algoritmul Rijndael, dezvoltat şi propus de autorul cu acelaşi nume. AES foloseşte un bloc de 128 de biţi şi chei de 128 de biţi (10 runde), 192 de biţi (12 runde) sau 256 de biţi (14 runde). Conform [CISCO SEC], viteza de criptare/ decriptare este una mare, iar consumul de resurse este unul mic.

În procesul de proiectare al algoritmului AES s-a ţinut cont de următoarele criterii: algoritmul trebuia să fie rezistent la toate atacurile cunoscute, codul trebuia să fie compact, viteza să fie mare pe mai multe platforme şi proiectarea acestuia să fie simplă. [www11]

Vom discuta, în cele ce urmează, în amănunt, despre algoritmul de criptare simetrică AES. Astfel, mesajul (fie textul în clar, fie cel cifrat), un bloc de 128 de biţi este segmentat în 16 bytes. Blocul de intrare are 16 unităţi de 8 biţi şi poate fi reprezentat ca fiind $InBl=m_0,m_1,...,m_{15}$[MAO03].

Rijndael se bazează pe teoria câmpului Galois, în sensul că anumite operaţiuni sunt definite la nivel de octet, iar octeţii reprezintă elemente în câmpul finit $GF(2^8)$. Cum toate reprezentările câmpului finit $GF(2^8)$ sunt izomorfe, se poate alege reprezentarea clasică polinomială cu impact pozitiv asupra complexităţii implementării. [www11]

Toate valorile (în bytes) vor fi reprezentate ca fiind concatenarea celor 8 biţi (de valoare 0 sau 1), în ordinea $\{b_7,b_6,b_5,b_4,b_3,b_2,b_1,b_0\}$. Un byte poate fi scris şi în reprezentare polinomială :

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0x^0 = \sum_{i=0}^{7} b_i x^i$$ [FIPS197]

Structura internă a unui bloc de intrare o reprezintă o matrice 4x4:

$$InBl = \begin{pmatrix} m_0 m_4 m_8 m_{12} \\ m_1 m_5 m_9 m_{13} \\ m_2 m_6 m_{10} m_{14} \\ m_3 m_7 m_{11} m_{15} \end{pmatrix}$$ [MAO03]

Intern, operaţiile algoritmului AES se efectuează pe un vector bidimensional de bytes numit **State** (stare). Acesta este alcătuit din patru rânduri de bytes, fiecare conţinând patru valori de bytes (dimensiunea blocului împărţită la 32). Starea este asemănătoare unei matrice, şi fiecare element va avea doi indici astfel:

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

Elementele din blocul de intrare (m_0, m_1,...,m_{15}) sunt copiate în stare. Operaţiile de cifrare sau decifrare sunt aplicate asupra stării, iar după ultima etapă acestea sunt copiate într-o matrice de ieşire [FIPS197]:

m_0	m_4	m_8	m_{12}
m_1	m_5	m_9	m_{13}
m_2	m_6	m_{10}	m_{14}
m_3	m_7	m_{11}	m_{15}

→

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

→

out_0	out_4	out_8	out_{12}
out_1	out_5	out_9	out_{13}
out_2	out_6	out_{10}	out_{14}
out_3	out_7	out_{11}	out_{15}

Algoritmul AES se bazează pe un număr de iteraţii ce aplică diverse transformări numite **rounds** (runde). O rundă în cadrul algoritmului se prezintă după cum urmează:

Round (State, RoundKey) [MAO03],

unde RoundKey este cheia corespunzătoare rundei respective, obţinută din cheia furnizată la începutul algoritmului.

Starea în cazul primei runde va prelua ca intrare valorile din InBL (în cazul criptării – mesajul în clar, iar in cazul decriptării - mesajul criptat), iar în cazul rundei finale va oferi mesajul criptat (sau decriptat). Runda (oricare în afară de runda finală) se compune din patru transformări diferite :

```
Round (State, RoundKey)
{
SubBytes(State)
ShiftRows(State)
MixColumns(State)
AddRoundKey(State, RoundKey)
}[MAO03]
```

Runda finală diferă de o rundă normală prin faptul că îi lipseşte transformarea MixColumns. În cazul decriptării se folosesc funcţiile inverse : $Round^{-1}$(State, RoundKey).

Funcţiile interne ale algoritmului funcţionează într-un câmp finit. Aceasta se realizează prin polinoame modulo f(x), unde f(x) este un polinom ireductibil:

$$f(x)= x^8+x^4+x^3+x+1. \text{ [MAO03]}$$

şi orice polinom modulo f(x) va avea gradul mai mic decât 8 şi se poate reprezenta astfel:

$$b_7x^7 + b_6x^6 +b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0 \text{ [MAO03]},$$

unde $b_7b_6b_5b_4b_3b_2b_1b_0$ formează un byte sau reprezintă un număr întreg pe 8 biţi.

Dacă avem două numere în hexa „57" şi „83", atunci „57" $\oplus$ „83"= „D4", cu alte cuvinte 01010111 $\oplus$ 10000011 =11010100, iar sub formă polinomială acest lucru se scrie $(x^6+x^4+x^2+x+1) \oplus (x^7+x+1)= x^7+x^6+x^4+x^2$[FIPS197]. Prin acest exemplu s-a descris operaţia de adunare modulo 2 (sau XOR) unde $1 \oplus 1=0$, $1 \oplus 0=1$ şi $0 \oplus 0=0$.

În cazul înmulţirii modulo f(x) pentru valorile din exemplul anterior „57" şi „83" vom avea „57" • „83"= „C1" deoarece

$(x^6+x^4+x^2+x+1) \bullet (x^7+x+1)= x^{13}+x^{11}+x^9+x^8+x^7+ x^7 + x^5 + x^3 + x^2 + x + x^6 + x^4 + x^2 +x+ 1 = x^{13}+x^{11}+x^9+x^8 + x^6 + x^5 + x^4 + x^3 + 1$ modulo $f(x)= x^7 + x^6 + 1$ [FIPS197].

Spre deosebire de operaţia de adunare, în cazul înmulţirii nu mai există operaţii simple la nivel de byte. Operaţia de înmulţire definită mai sus este asociativă, iar dacă avem un polinom b(x) de grad mai mic decât 8 atunci $b^{-1}(x)$ este inversul lui b(x). Dacă b(x)a(x) + f(x)c(x)=1 şi a(x) • b(x) mod f(x) =1, atunci putem scrie că $b^{-1}(x)=$ a(x)mod f(x), şi astfel a(x) •(b(x)+c(x))=a(x) •b(x)+a(x) •c(x) [FIPS197].

Aceste tipuri de operaţii alături de operaţii pe bytes de tipul xtime() sunt folosite ca operaţii interne în cazul algoritmului AES. După cum se poate vedea, acestea nu sunt operaţii complexe şi consumatoare de putere de calcul la fel cum sunt cele din algoritmii asimetrici.

În cazul cifrării, ca transformări interne sunt folosite **SubBytes(),ShiftRows(), MixColumns()** şi **AddRound Key()**. Acestea se vor detalia în cele ce urmează.

Transformarea SubBytes() este o substituţie neliniară ce operează independent pe fiecare byte folosind un tabel de substituţie(S-box). Acest tabel (fig. 2.2.1) este construit folosind două transformări:

- ✓ Se foloseşte funcţia b(x)a(x) + f(x)c(x)=1
- ✓ Se aplică transformarea $b_i'= b_i \oplus b_{(i+5)mod8} \oplus b_{(i+6)mod8} \oplus b_{(i+7)mod8} \oplus c_i$ unde b_i şi c_i este bitul i din byte-ul b, respectiv c.

Sub formă matricială se poate transcrie astfel:

$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} 10001111 \\ 11000111 \\ 11100011 \\ 11110001 \\ 11111000 \\ 01111100 \\ 00111110 \\ 00011111 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$ [FIPS197]

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figura 2.2.1 S-box Valori de substituire în format Hexazecimal [FIPS197]

În transformarea ShiftRows(), un byte este shiftat ciclic în ultimele trei rânduri ale Stării (fig 2.2.2). Astfel, efectul este acela că un byte este mutat în poziţii "inferioare" ale rândului. De menţionat este faptul că primul rând nu se shiftează.

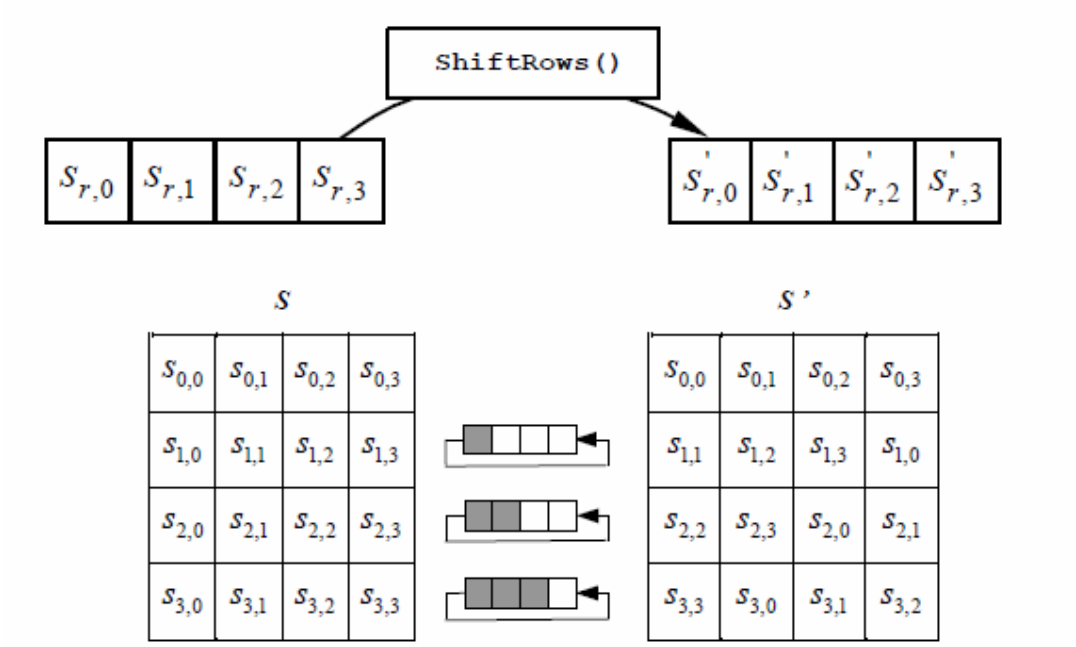

Figura 2.2.2 ShiftRows(). Deplasarea ciclică a ultimelor trei rânduri din Stare [FIPS197]

În transformarea MixColumns() se operează pe **Stare** coloană cu coloană. Fiecare coloană este privită ca un polinom cu patru termeni. Acesta este înmulţit modulo x^4+1 cu un polinom a(x), unde $a(x)=\{03\}x^3+\{01\}x^2+\{01\}x+\{02\}$ [FIPS197]. Acest lucru poate fi transcris astfel: $s'(x)=a(x)\otimes s(x)$ iar

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad \text{[FIPS197]}$$

În urma operaţiei cei patru bytes din coloană sunt înlocuiţi cu aceştia:

$$s'_{0,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$

$$s'_{1,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c}$$

$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c})$$

$$s'_{3,c} = (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c}).$$

[FIPS197]

În figura 2.2.3 se prezintă modul de operare al MixColumns.

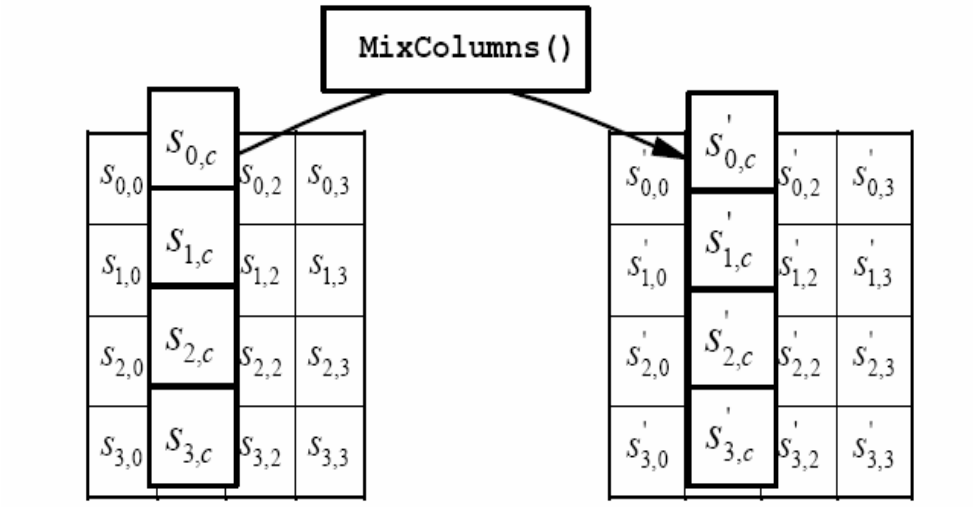

Figura 2.2.3 MixColumns().Operaţii pe Stare coloană cu coloană

[FIPS197]

În transformarea AddRoundKey() se adaugă o cheie de rundă (RoundKey) prin operaţie XOR() la Stare ca în figura 2.2.4.

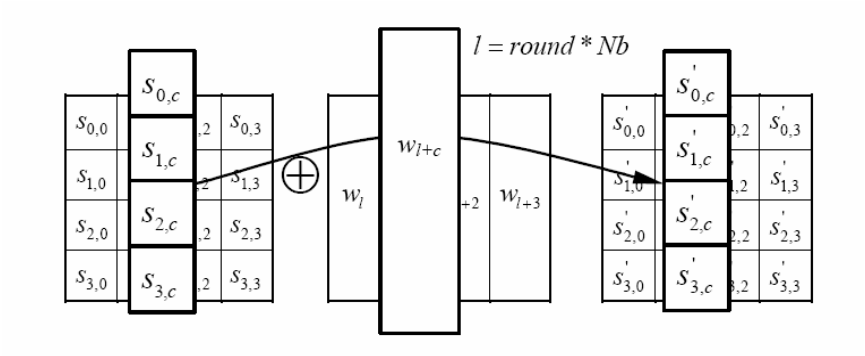

Figura 2.2.4 AddRoundKey().[FIPS197]

O analiză legată de complexitatea computaţională a algoritmului Rijndael şi a altor algoritmi o fac Fabrizio Graneli şi Giulia Boato în [GRA04]. În această lucrare se realizează o comparare a algoritmilor Rijndael, Camelia şi Shacal-2. În urma cercetării efectuate, se ajunge la concluzia că *"Rijndael este foarte bun şi astfel poate fi folosit ca referinţă pentru benchmark"* [GRA04]. În tabelul 2.2.1 sunt prezentate valorile corespunzătoare lui AES conform testelor efectuate de autori.

Denumire	Operaţii			
	ŞI	SAU	Shift(bytes)	Adunare 32 bit
AES General	5836	4254	1336	0
Key expansion	1536	1536	846	144
Criptare	4912	3624	1188	0
Decriptare	14896	11112	3654	0
Operaţie	Algoritm (dimensiune cheie)			
	128	192	256	
SI	7236	8784	10334	
SAU	5418	6536	7667	

Tabel 2.2.1. Complexitate computaţională AES. Operaţii [GRA04]

Simplitatea algoritmului AES este dată şi de faptul că algoritmul nu foloseşte operaţii aritmetice, ci doar operaţii la nivel de şiruri de biţi, nu foloseşte componente criptografice externe, cum ar fi cutii S, biţi aleatori sau şiruri de cifre obţinute din computaţii complexe. Proiectarea clară a AES nu permite ascunderea unui "trap door". [GRA04]

În [HAI02], autorul analizează complexitatea computaţională pentru algoritmi criptografici de tip bloc, stream şi hash. În urma unor teste, se ajunge la concluzia că algoritmii de tip bloc necesită în medie 84 de cicluri pentru procesarea unui byte, algoritmii de tip stream necesită 20, iar algoritmii de tip hash 15. În cadrul algoritmilor bloc au fost testaţi AES, Blowfish, 3DES, IDEA şi RC6. Dintre aceştia, 3DES a necesitat 187 de cicluri, deoarece aplică aceleaşi operaţii de trei ori, cu trei chei diferite triplându-se complexitatea computaţională. Potrivit lui [HAI02], AES are o complexitate computaţională mai mare decât a celorlalţi 3 algoritmi rămaşi. În cazul testării algoritmilor pe procesoare paralele, s-a ajuns la concluzia că numărul

de instrucţiuni ce pot fi executate într-un ciclu a crescut cu 64% în cazul algoritmilor de tip bloc, atunci când numărul de ALU creşte de la 1 la 2, cu 32% atunci când numărul de ALU creşte de la 2 la 4, şi o creştere mai mică de 1% atunci când se folosesc mai mult de 4 ALU [HAI02].

Teste efectuate pentru procesorul Intel IXP2800 (network processor) [ZHA04] au arătat că algoritmii hash pot atinge lăţimi de bandă de peste 5 Gbps, iar în cazul algoritmilor stream şi bloc se obţine în medie o lăţime de bandă de 1,5 Gbps (izolat, unii algoritmi au obţinut şi 2Gbps). În cazul algoritmilor AES şi DES se obţine o creştere a performanţelor „per-flow" şi a lăţimii de bandă în configuraţiile cu patru, respectiv opt fire de execuţie faţă de configuraţia „single thread" [ZHA04].

2.3. Funcţii hash

Ca funcţii hash sau funcţii uni-sens (one way hash functions [KAM04], compression function, message digest, fingerprint [MEN96], cryptographic checksum [SCH96], funcţii neglijabile [GRO07]) au fost alese următoarele :

- SHA1
- SHA256
- SHA512
- MD5

Aceste funcţii sunt considerate iterative. O funcţie hash iterativă presupune împărţirea intrării în blocuri de dimensiune fixă. Aceste blocuri sunt procesate în ordine folosind funcţii de compresie şi stări intermediare recursive. Rezultatul ultimei iteraţii este rezultatul funcţiei hash [SCH03].

MD5 este algoritmul dezvoltat de Ron Rivest [MD592]. Acest algoritm împarte intrarea în blocuri cu dimensiunea de 512 biţi. Starea internă a acestui algoritm este de 128 de biţi care este împărţită în patru stări de 32 de biţi. Funcţia de compresie are 4 etape (sau runde) şi în fiecare rundă, mesajul bloc şi starea sunt mixate. Această operaţie constă în folosirea de operatori XOR, ŞI, SAU şi rotaţia, aplicate stărilor de 32 de biţi [LIN04]. Literele din MD5 vin de la *message digest.* Anderson, R. afirmă că MD4, respectiv MD5 nu mai prezintă siguranţă, deoarece se găsesc cu uşurinţă coliziuni [AND08]. Prin urmare, a fost necesară o nouă familie de funcţii hash. Există site-uri specializate ([www2], [www3], [www4]) care oferă căutări rapide ale datelor de intrare dacă se furnizează valoarea hash generată de MD5. [www1] prezintă o metodă de a folosi procesorul video pentru a calcula şi găsi mai repede valoarea de intrare a unei valori MD5 prin metoda de brute force.

Funcţiile din familia SHA au fost dezvoltate de NSA [SCH03], iar apoi au fost standardizate de NIST [FIPS96]. Abrevierea SHA vine de la *Secure hash Algorithm*. Prima funcţie, care a fost numită şi SHA-0, conţinea o slăbiciune. SHA-1 este următoarea versiune şi este o funcţie pe 160 de biţi ce are la bază algoritmul MD4 folosind cinci stări de 32 de biţi. SHA 256 şi SHA 512 sunt algoritmi hash care au ca ieşiri valori de 256, respectiv 512 biţi. Aceştia din urmă sunt şi mai lenţi decât MD5 sau SHA1. În prezent, există 3 familii de algoritmi SHA şi anume SHA0, SHA1 şi SHA2. În familia SHA 2 intră SHA224, SHA256, SHA384 SHA512. [CISCO CCNA EXP]. Pentru o a treia familie de funcţii SHA, şi anume SHA3 se procedează ca şi în cazul lui AES [HSH08].

Proprietăţile pe care trebuie să le aibă o funcţie hash sunt [MAO03]:

- Transformare-mix. Pentru orice intrare x, rezultatul h(x) trebuie să fie de nedistins computaţional dintr-un şir de caractere din intervalul $[0,2^h)$.

- Rezistenţă la coliziuni. Este necesar ca să fie imposibil de găsit două intrări x, z cu x ≠ z astfel încât h(x)=h(z).
- Unisens. Având rezultatul algoritmului hash h ar trebui să fie imposibil de calculat intrarea x, astfel încât h=h(x).
- Eficienţă practică. Fiind dată intrarea x, calculul lui h(x) trebuie să se efectueze într-un timp suficient de scurt.
- Keyless. Algoritmul nu foloseşte nicio cheie în procesul de obţinere a rezultatului

De fapt, o funcţie hash este [SOL08], o funcţie ce primeşte ca intrare un şir de lungime variabilă şi după prelucrare oferă ca rezultat un şir de lungime fixă, având proprietatea că din rezultatul final este greu de inversat [GRO07]. Cu toate acestea în [CIU03] se prezintă o implementare de criptare simetrică ce are la bază funcţii hash folosite în Counter Mode. Plecând de la această propunere şi folosind codurile Huffman [JEB04] am realizat o adaptare a soluţiei propuse de [CIU03] prin arhivarea datelor [TOM08].

2.4. Concluzii

În cadrul acestui capitol s-a realizat o taxonomie şi o descriere teoretică a primitivelor criptografice alese pentru a fi testate în capitolele următoare.

S-a introdus o clasificare pe categorii a primitivelor alese după modul acestora de operare.

S-a conceput o justificare a alegerii acestor algoritmi care urmează să fie folosiţi în testele efectuate în capitolele următoare.

3. Aplicaţia de testare

În acest capitol se vor descrie aplicaţiile implementate în cele patru medii de dezvoltare. Pentru platforma Windows (XP SP3 şi 2000 SP4), au fost dezvoltate aplicaţii în Visual Studio 2008 (Licenţă academică [MSA06]), mai exact Visual C# şi Visual Basic .Net. Pentru a putea rula aplicaţiile în varianta împachetată stand alone (împachetat cu toate bibliotecile necesare pentru a pute fi rulat pe orice platformă) sistemul de operare trebuie să aibă instalate Dot. Net. Framework 3.5. SP1. [MSDN08]. În subcapitolele 3.1 şi 3.2 se vor descrie clasele folosite în dezvoltarea aplicaţiilor de test. Aceste clase aparţin de System.Security.Cryptography [MSDN].

3.1. Aplicaţiile de criptare pentru date stocate pe hard disc

3.1.1. Implementarea algoritmilor în mediul de dezvoltare Visual Basic

În Visual Basic, punctul de plecare îl constituie bibliotecile standard puse la dispoziţie de acest mediu de programare.

Astfel, pentru algoritmii HMAC folosim clasele din tabelul 3.1.1.1.

Nume clasă	Descriere
HMAC	Reprezintă clasa abstractă din care trebuie să deriveze toate implementările HMAC
HMACMD5	Calculează un HMAC folosind funcţia hash MD5.
HMACSHA1	Calculează un HMAC folosind funcţia hash SHA1.
HMACSHA256	Calculează un HMAC folosind funcţia hash SHA256.
HMACSHA512	Calculează un HMAC folosind funcţia hash SHA512.

Tabel 3.1.1.1. Clase HMAC folosite în aplicaţie[MSDN]

În tabelul 3.1.1.2 sunt prezentate metodele, constructorii, proprietăţile şi câmpurile clasei HMAC.

Clasa HMAC	Descriere	
Câmpuri	hashSize Value	Reprezintă dimensiunea în biţi a valorii hash calculate
	hashValue	Reprezintă valoarea hash calculată
	KeyValue	Cheia folosită în algoritmul HMAC.
	State	Reprezintă starea de calcul a funcţiei hash
Metode	Clear	Eliberează toate resursele folosite de clasa hashAlgorithm
	Computehash	Calculează valoarea hash pentru datele de intrare specificate
	Create	Creează o instanţă a implementării implicite a unui HMAC
	Equals	Determină dacă obiectul specificat este echivalent cu obiectul curent
	Finalize	Permite unui obiect să elibereze resurse şi să realizeze alte operaţii de curăţare înainte ca obiectul să fie colectat de garbage collection.
	Gethash Code	Serveşte ca funcţie pentru cazuri particulare
	hashCore	Când este suprascrisă într-o clasă derivată, rutează datele scrise în obiect în algoritmul HMAC default pentru a calcula valoarea hash.
	hashFinal	Când este suprascrisă într-o clasă derivată, finalizează calculul hash după ce ultima informaţie este procesată de către obiectul fluxului criptografic.
	Initialize	Iniţializează o clasă a implementării HMAC implicite.
Proprietăţi	BlockSize Value	Returnează sau setează dimensiunea bloc care se foloseşte în valoarea hash.
	hash	Valoarea hash calculată
	hashName	Returnează sau setează numele algoritmului hash care se foloseşte pentru calcul.
	hashSize	Dimensiunea, în biţi, a valorii hash calculate
	Key	Returnează sau setează cheia folosită în algoritmul hash
Constructori	HMAC	Iniţializează o nouă instanţă a clasei HMAC.

Tabel 3.1.1.2 Membrii clasei HMAC [MSDN]

În tabelul 3.1.1.3 sunt prezentate o parte din metodele, constructorii, proprietăţile şi câmpurile clasei HMACMD5.

Clasa HMACMD5	Descriere	
Câmpuri	hashSize Value	Reprezintă dimensiunea în biţi a valorii hash calculate
	hashValue	Reprezintă valoarea hash calculată
	KeyValue	Cheia folosită în algoritmul HMAC.
	State	Reprezintă starea de calcul a funcţiei hash
Metode	Clear	Eliberează toate resursele folosite de clasa hashAlgorithm
	Computehash	Calculează valoarea hash pentru datele de intrare specificate
	Create	Creează o instanţă a implementării implicite a unui HMAC
	Equals	Determină dacă obiectul specificat este echivalent cu obiectul curent
	Finalize	Permite unui obiect să elibereze resurse şi să realizeze alte operaţii de curăţare înainte ca obiectul să fie colectat de garbage collection.
	Gethash Code	Serveşte ca funcţie pentru cazuri particulare
	hashCore	Când este suprascrisă într-o clasă derivată, rutează datele scrise în obiect în algoritmul HMAC default pentru a calcula valoarea hash.
	hashFinal	Când este suprascrisă într-o clasă derivată, finalizează calculul hash după ce ultima informaţie este procesată de către obiectul fluxului criptografic.
	Initialize	Iniţializează o clasă a implementării HMAC implicite.
Proprietăţi	BlockSize Value	Returnează sau setează dimensiunea bloc care se foloseşte în valoarea hash.
	hash	Valoarea hash calculată
	hashName	Returnează sau setează numele algoritmului hash ce se foloseşte pentru calcul.
	hashSize	Dimensiunea, în biţi, a valorii hash calculate
	Key	Returnează sau setează cheia folosită în algoritmul hash
Constructori	HMACMD5	Iniţializează o nouă instanţă a clasei HMACMD5.

Tabel 3.1.1.3 Membrii clasei HMAC MD5 [MSDN]

Nu se vor mai descrie şi clasele HMACSHA1, HMACSHA256, HMACSHA512 deoarece acestea au metode, constructori, proprietăţi şi câmpuri similare cu cele prezentate. Trecând la algoritmii simetrici, vom prezenta clasele folosite de aplicaţii în testarea vitezei de criptare pentru algoritmii din tabelul 3.1.1.4

Nume clasă	Descriere
DES	Reprezintă clasa de bază pentru algoritmul DES din care trebuie să deriveze toate implementările DES
DESCryptoServiceProvider	Defineşte un obiect care accesează
TripleDes	Reprezintă clasa de bază pentru algoritmul 3DES din care trebuie să deriveze toate implementările 3DES.
AES	Reprezintă clasa de bază pentru algoritmul AES pe care o moştenesc toate implementările AES.
AESCryptoServiceProvider	Realizează criptări şi decriptări folosind implementarea Cryptographic Application Programming Interfaces a algoritmului AES
AesManaged Class	Asigură o implementare organizată a algoritmului AES

Tabel 3.1.1.4 Clase criptare simetrică folosite în aplicaţie[MSDN]

În tabelul 3.1.1.5 sunt prezentate o parte din metodele, constructorii, proprietăţile şi câmpurile clasei DES.

Clasa DES	Descriere	
Câmpuri	IVValue	Vectorul de iniţializare pentru algoritm
	KeySizeValue	Dimensiunea, în biţi, a cheii secrete
	KeyValue	Cheia secretă folosită în algoritm
Metode	Clear	Eliberează toate resursele folosite de clasa DES
	GenerateIV	Când este suprascrisă într-o clasă derivată generează un vector de iniţializare aleator (IV) pentru a fi folosit de algoritm
	Create	Creează o instanţă a implementării implicite a unui obiect DES
	Equals	Determină dacă obiectul specificat este echivalent cu obiectul curent
	Finalize	Permite unui obiect să elibereze resurse şi să realizeze alte operaţii de curăţare înainte ca obiectul să fie colectat de garbage collection.
	Gethash Code	Serveşte ca funcţie pentru cazuri particulare
Proprietăţi	BlockSize Value	Dimensiunea blocului
	IV	Vectorul de iniţializare pentru algoritm
	KeySize	Dimensiunea, în biţi, a cheii secrete
	Key	Cheia secretă folosită în algoritm.
Constructori	DES	Iniţializează o nouă instanţă a clasei DES.

Tabel 3.1.1.5. Membrii clasei DES[MSDN]

În tabelul 3.1.1.6 sunt prezentate o parte din metodele, constructorii, proprietăţile şi câmpurile clasei AES.

Clasa AES	Descriere	
Câmpuri	IVValue	Vectorul de iniţializare pentru algoritm
	KeySizeValue	Dimensiunea, în biţi, a cheii secrete
	KeyValue	Cheia secretă folosită în algoritm.
Metode	Clear	Eliberează toate resursele folosite de clasa AES
	GenerateIV	Când este suprascrisă într-o clasă derivată generează un vector de iniţializare aleator (IV) pentru a fi folosit de algoritm
	Create	Creează o instanţă a implementării implicite a unui obiect AES
	Equals	Determină dacă obiectul specificat este echivalent cu obiectul curent
	Finalize	Permite unui obiect să elibereze resurse şi să realizeze alte operaţii de curăţare înainte ca obiectul să fie colectat de garbage collection.
	GenerateKey	Când este suprascris într-o clasă derivată generează aleator o cheie pentru a fi folosită de algoritm
Proprietăţi	BlockSize	Returnează sau setează dimensiunea blocului în biţi
	IV	Vectorul de iniţializare pentru algoritm
	KeySize	Dimensiunea, în biţi, a cheii secrete
	Key	Cheia secretă folosită în algoritm
Constructori	Aes	Iniţializează o nouă instanţă a clasei AES

Tabel 3.1.1.6 Membrii clasei AES[MSDN]

După algoritmii HMAC şi cei simetrici prezentăm şi o parte din funcţiile MAC, mai exact membrii claselor acestora.

Clasa MD5	Descriere	
Câmpuri	hashSize Value	Reprezintă dimensiunea în biţi a valorii hash calculate
	hashValue	Reprezintă valoarea hash calculată
	State	Reprezintă starea de calcul a funcţiei hash
Metode	Clear	Eliberează toate resursele folosite de clasa hashAlgorithm
	Computehash	Calculează valoarea hash pentru datele de intrare specificate
	Create	Creează o instanţă a implementării implicite a unui HMAC
	Equals	Determină dacă obiectul specificat este echivalent cu obiectul curent
	Finalize	Permite unui obiect să elibereze resurse şi să realizeze alte operaţii de curăţare înainte ca obiectul să fie colectat de garbage collection.
	Gethash Code	Serveşte ca funcţie pentru cazuri particulare
	hashCore	Când este suprascrisă într-o clasă derivată, rutează datele scrise în obiect în algoritmul HMAC default pentru a calcula valoarea hash.
	hashFinal	Când este suprascrisă într-o clasă derivată, finalizează calculul hash după ce ultima informaţie este procesată de către obiectul fluxului criptografic.
	Initialize	Iniţializează o clasă a implementării HMAC implicite.
Proprietăţi	Input BlockSize	Returnează sau setează dimensiunea bloc ce se foloseşte în valoarea hash.
	hash	Valoarea hash calculată
	hashSize	Dimensiunea, în biţi, a valorii hash calculate
Constructori	MD5	Iniţializează o nouă instanţă a clasei CMD5.

Tabel 3.1.1.7 Membrii clasei MD5 [MSDN]

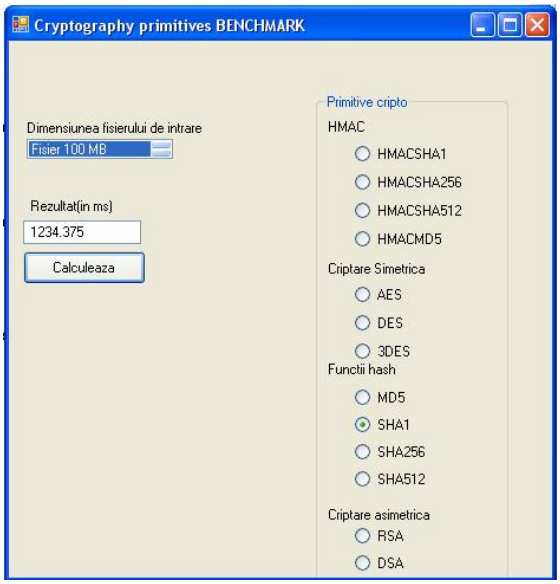

Figura 3.1.1.1. Aplicaţia în Visual Basic

În figura 3.1.1.1, este prezentată o captură a aplicaţiei în Visual Basic 2008. În ceea ce priveşte partea aplicaţiei, putem spune că a fost dezvoltată plecând de la clasele standard descrise anterior şi au fost apelate funcţiile pentru criptare corespunzătoare primitivelor criptografice.

Astfel, s-a implementat clasa HMACMD5 (conform [MSDN]):

```
Public Class HmacMD5ex
        Public Shared Sub EncodeFile(ByVal key() As Byte, ByVal sourceFile As
String, ByVal destFile As String)
      Dim myhmacMD5 As New HMACMD5(key)
      Dim inStream As New FileStream(sourceFile, FileMode.Open)
      Dim outStream As New FileStream(destFile, FileMode.Create)
      Dim hashValue As Byte() = myhmacMD5.Computehash(inStream)
      inStream.Position = 0
      outStream.Write(hashValue, 0, hashValue.Length)
      myhmacMD5.Clear()
      inStream.Close()
      outStream.Close()
      Return
   End Sub
End Class
```

Pentru ca în program să fie apelată astfel:

```
HmacMD5ex.EncodeFile(_key, "test1mb.txt", "test1mb.hsh")
```

Numele fişierelor variază în funcţie de ce fişier se doreşte a fi selectat pentru a rula algoritmul de test. Astfel test1mb.txt este fişierul de intrare, iar ca ieşire, în fişierul test1mb.hsh se va obţine codul HAMACMD5 (în cazul acesta), alte coduri HMAC, hash sau fişierul criptat cu algoritmul simetric ales.

Pentru chei au fost alese următoarele valori, prima fiind folosită pentru toţi algoritmii HMAC, următoarele două fiind folosite pentru criptarea simetrică ce

implementează algoritmul AES şi 3DES, iar ultimele două sunt folosite în cadrul algoritmului DES [VBE08]:

```
Private _key() As Byte = {132, 42, 53, 124, 75, 56, 87, 38, 9, 10, 161, 132, 183,
91, 105, 16, 117, 218, 149, 230, 221, 212, 235, 64}
Private _iv() As Byte = {83, 71, 26, 58, 54, 35, 22, 11, 83, 71, 26, 58, 54, 35, 22,
11}
Private Shared Des_key() As Byte = {1, 2, 3, 4, 5, 6, 7, 8}
Private Shared Des_iv() As Byte = {1, 2, 3, 4, 5, 6, 7, 8}
```

În cazul criptării simetrice (AES) a fost implementată [VBE08], [MSDN] procedura detaliată mai jos.

```
Private Shared Sub AESEncryptData(ByVal inName As String, ByVal outName As
String,  ByVal aesKey() As Byte, ByVal aesIV() As Byte)
      Dim aes As New AesCryptoServiceProvider
      aes.Key = aesKey
      aes.Mode = CipherMode.ECB
      aes.Padding = PaddingMode.ISO10126
      Dim encryptor = aes.CreateEncryptor()
      Dim fin As New FileStream(inName, FileMode.Open, FileAccess.Read)
      Dim fout As New FileStream(outName, FileMode.OpenOrCreate, _
        FileAccess.Write)
        fout.SetLength(0)
      Dim bin(100) As Byte
      Dim rdlen As Long = 0
      Dim totlen As Long = fin.Length
      Dim len As Integer
      Dim encStream As New CryptoStream(fout, _
  aes.CreateEncryptor(aes.Key, aesIV), CryptoStreamMode.Write)
      While rdlen < totlen
         len = fin.Read(bin, 0, 100)
         encStream.Write(bin, 0, len)
         rdlen = rdlen + len
      End While
      encStream.Close()
  End Sub
```

S-a observat că, dacă se foloseşte Dim bin(4096) As Byte respectiv len = fin.Read(bin, 0, 4096), viteza generală de criptare este mai mare. Acest lucru s-a observat mai clar la fişierele de dimensiuni mai mari (peste 1 GB), pe când în cazul fişierelor de mici, dimensiunile din teste (1 MB, respectiv 100 MB) nu se observă o diferenţă sesizabilă. În versiunea finală a testului s-a folosit ca buffer dimensiunea de 4096.

Apelarea se face astfel:

```
AESEncryptData("test1MB.txt", "test1MB.hsh", _key, _iv)
```

unde _key şi _iv sunt cele prezentate mai sus.

Timpul se calculează în modul următor:

```
Dim now As Date = Date.Now
sha512.Computehash(f)
Dim after As Date = Date.Now
Dim time As TimeSpan = after - now
Dim zile As Integer = time.Days
Dim ore As Integer = time.Hours
```

```
        Dim minute As Integer = time.Minutes
        Dim secunde As Integer = time.Seconds
        Dim millisecunde As Integer = time.Milliseconds
        Dim timp As Double
        timp = (millisecunde + secunde * 1000 + minute * 60000 + ore *
3600000 + zile * 86400000) / 1
```

sau

```
        TextBox1.Text = Str(time.TotalMilliseconds)
```

În versiunea finală a fost aleasă varianta a doua fiind mai comodă această abordare şi rezultatul mai exact (cu zecimale). Rezultatul este pus la dispoziţie într-un textbox pentru uşurinţa preluării acestuia într-un fişier Excel(xls). Din cauza volumului foarte mare de calcul şi a perioadelor foarte lungi de execuţie s-a optat pentru alegerea manuală a algoritmilor şi a fişierelor asupra cărora se aplică funcţiile.

În figura 3.1.1.1, este prezentată o captură a imaginii aplicaţiei în Visual Basic, astfel, partea vizuală a aplicaţiei este simplă: utilizatorul poate alege algoritmul de testat (din partea dreaptă) şi mai selectează din listă fişierul asupra căruia se vor face testele (partea stângă sus). După această etapă se lansează în execuţie prin butonul „Calculează”. La sfârşitul operaţiei se afişează timpul în milisecunde în textbox-ul ferestrei.

3.1.2. Implementarea algoritmilor în mediul de dezvoltare Visual C#

Dezvoltarea aplicaţiei în C# s-a gândit în mod similar celei în Visual Basic, folosind tot bibliotecile standard oferite de Visual Studio 2008 în System. Security. Cryptography. Clasele folosite sunt aceleaşi ca cele din capitolul 3.1.1 (vezi tabel 3.1.1.1, 3..11.2, 3.1.1.3, 3.1.1.4,3.1.1.5,3.1.1.6, 3.1.1.7). În figura 3.1.2.1, este prezentată o captură de imagine a aplicaţiei în C#. După cum se poate observa, atât aplicaţia în Visual Basic cât şi cea în C# au fost dezvoltate în mod similar, atât din punct de vedere vizual, funcţional, cât şi din punctul de vedere al programării. Partea vizuală a aplicaţiei este simplă: utilizatorul poate alege algoritmul de testat (din partea dreaptă) şi mai selectează din listă fişierul asupra căruia se vor face testele (partea stângă, sus). După această etapă, se lansează în execuţie prin butonul „Calculează”. La sfârşitul operaţiei se afişează timpul în milisecunde în textbox-ul ferestrei.

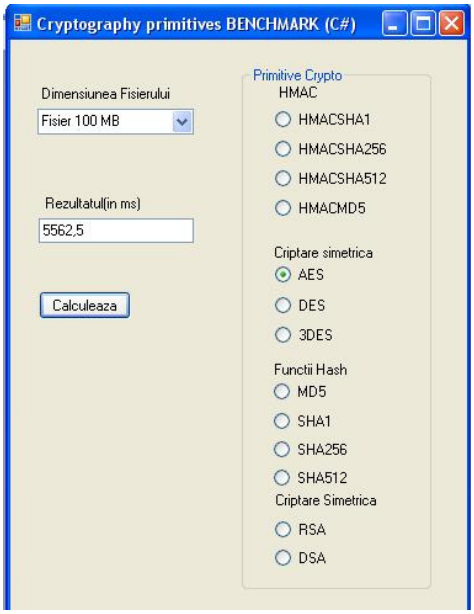

Figura 3.1.2.1 Aplicaţia în C#

Pentru evaluarea timpului de execuţie a unei instanţe s-a apelat la metoda următoare :

```
DateTime startTime = DateTime.Now;
HMACSHA1ex.EncodeFile(_key, "test1mb.txt", "test1mb.hsh");
DateTime stopTime = DateTime.Now;
TimeSpan duration = stopTime - startTime;
   double timp = duration.TotalMilliseconds;
```

Pentru implementarea funcţiilor s-au folosit clasele descrise anterior. Pentru exemplificare, se descrie HMACSHA1[MSDN].

```
public class HMACSHA1ex

{
   public static void EncodeFile(byte[] key, String sourceFile, String destFile)
   {
      HMACSHA1 myhmacsha1 = new HMACSHA1(key);
   FileStream inStream = new FileStream(sourceFile, FileMode.Open);
   FileStream outStream = new FileStream(destFile, FileMode.Create);
      byte[] hashValue = myhmacsha1.Computehash(inStream);
      inStream.Position = 0;
      outStream.Write(hashValue, 0, hashValue.Length);
      int bytesRead;
      byte[] buffer = new byte[1024];
      myhmacsha1.Clear();
      inStream.Close();
      outStream.Close();
      return;
```

```
        }
```

Apelarea funcţiei se face prin :
HMACSHA1ex.EncodeFile(_key, "test1mb.txt", "test1mb.hsh");
În cazul algoritmului 3DES se procedează astfel [RIJ07][MSDN]:

```
private static void TDESEncryptData(String inName, String outName, byte[]
tdesKey, byte[] tdesIV)
     {
 FileStream fin = new FileStream(inName, FileMode.Open, FileAccess.Read);
FileStream fout = new FileStream(outName, FileMode.OpenOrCreate,
FileAccess.Write);
        fout.SetLength(0);
        byte[] bin = new byte[4096];
        long rdlen = 0;
        long totlen = fin.Length;
        int len;
TripleDESCryptoServiceProvider tdes = new TripleDESCryptoServiceProvider();
CryptoStream encStream = new CryptoStream(fout, tdes.CreateEncryptor(tdesKey,
tdesIV), CryptoStreamMode.Write);
        while (rdlen < totlen)
        {  len = fin.Read(bin, 0, 4096);
           encStream.Write(bin, 0, len);
           rdlen = rdlen + len;
        }
        encStream.Close();
        fout.Close();
        fin.Close();
     }
```

Cheile secrete folosite sunt aceleaşi prezentate şi la capitolul 3.1. Diferenţele de performanţă dintre cele două limbaje de programare vizuale se vor discuta în capitolul 4.

3.1.3. Realizarea testelor folosind algoritmii OpenSSL UNIX

Pentru a testa algoritmii nominalizaţi în acest benchmark pe sistemul de operare UNIX a fost aleasă distribuţia Live CD Kubuntu 9.04 [KUB10]. Se puteau folosi pentru testele sub UNIX şi biblioteci GNU Crypto ([GNU02]), dar pentru această etapă s-a optat pentru folosirea OpenSSL fiind mai facilă. A fost aleasă soluţia de Live CD deoarece aceasta a fost mai comodă, a putut fi rulată pe toate platformele fără a fi necesară instalarea unui sistem de operare şi a risca să întâmpinăm probleme de instalare la una din platforme. Faptul că fişierele au fost citite şi scrise pe partiţie NTFS este un dezavantaj din cauza faptului că sistemele de operare rulează nativ pe EXT2/EXT3, dar în acelaşi timp este un avantaj, deoarece se poate compara mai uşor cu mediile vizuale din capitolele 3.1.1 şi 3.1.2 ţinând cont de faptul că acestea au fost rulate tot pe partiţii NTFS. În cadrul testelor efectuate sub UNIX s-a optat pentru biblioteci din OpenSSL [OSP99]. Pentru a lista comenzile disponibile se foloseşte comanda:
$ **openssl help**

Această comandă are ca efect[OSC04]:

Standard commands
asn1parse ca ciphers crl crl2pkcs7
dgst dh dhparam dsa dsaparam
ec ecparam enc engine errstr
gendh gendsa genrsa nseq ocsp
passwd pkcs12 pkcs7 pkcs8 prime
rand req rsa rsautl s_client
s_server s_time sess_id smime speed
spkac verify version x509
Message Digest commands
md2 md4 md5 rmd160 sha
sha1
Cipher commands
aes-128-cbc aes-128-ecb aes-192-cbc aes-192-ecb aes-256-cbc
aes-256-ecb base64 bf bf-cbc bf-cfb
bf-ecb bf-ofb cast cast-cbc cast5-cbc
cast5-cfb cast5-ecb cast5-ofb des des-cbc
des-cfb des-ecb des-ede des-ede-cbc des-ede-cfb
des-ede-ofb des-ede3 des-ede3-cbc des-ede3-cfb des-ede3-ofb
des-ofb des3 desx rc2 rc2-40-cbc
rc2-64-cbc rc2-cbc rc2-cfb rc2-ecb rc2-ofb
rc4 rc4-40

Prin intermediul comenzii „openssl speed” se poate realiza un benchmark. Această comandă testează mai degrabă ca rezultat numărul de operaţii ce pot fi efectuate într-o perioadă de timp decât perioada necesară rulării unui număr de operaţii. Se prezintă în continuare rezultatele obţinute pe un procesor 2.16GHz Intel Core 2[OSC04].

The 'numbers' are in 1000s of bytes per second processed.

type	*16 bytes*	*64 bytes*	*256 bytes*	*1024 bytes*	*8192 bytes*
md2	*1736.10k*	*3726.08k*	*5165.04k*	*5692.28k*	*5917.35k*
mdc2	*0.00*	*0.00*	*0.00*	*0.00*	*0.00*
md4	*18799.87k*	*65848.23k*	*187776.43k*	*352258.73k*	*474622.63k*
md5	*16807.01k*	*58256.45k*	*160439.13k*	*287183.53k*	*375220.91k*
hmac(md5)	*23601.24k*	*74405.08k*	*189993.05k*	*309777.75k*	*379431.59k*
sha1	*16774.59k*	*55500.39k*	*142628.69k*	*233247.74k*	*288382.98k*
rmd160	*13854.71k*	*40271.23k*	*87613.95k*	*124333.06k*	*141781.67k*
rc4	*227935.60k*	*253366.06k*	*261236.94k*	*259858.09k*	*194928.50k*
des cbc	*48478.10k*	*49616.16k*	*49765.21k*	*50106.71k*	*50034.01k*
des ede3	*18387.39k*	*18631.02k*	*18699.26k*	*18738.18k*	*18718.72k*
idea cbc	*0.00*	*0.00*	*0.00*	*0.00*	*0.00*
rc2 cbc	*19247.24k*	*19838.12k*	*19904.51k*	*19925.33k*	*19834.98k*
rc5-32/12 cbc	*0.00*	*0.00*	*0.00*	*0.00*	*0.00*
blowfish cbc	*79577.50k*	*83067.03k*	*84676.78k*	*84850.01k*	*85063.00k*
cast cbc	*45362.14k*	*48343.34k*	*49007.36k*	*49202.52k*	*49225.73k*
aes-128 cbc	*58751.94k*	*94443.86k*	*111424.09k*	*116704.26k*	*117997.57k*
aes-192 cbc	*53451.79k*	*82076.22k*	*94609.83k*	*98496.85k*	*99150.51k*
aes-256 cbc	*49225.21k*	*72779.84k*	*82266.88k*	*85054.81k*	*85762.05k*
sha256	*9359.24k*	*22510.83k*	*40963.75k*	*51710.29k*	*56014.17k*
sha512	*7026.78k*	*28121.32k*	*54330.79k*	*86190.76k*	*104270.51k*

Testele efectuate sub UNIX au fost cuantificate cu ajutorul comenzii time disponibilă în acest sistem de operare [www10].

Rezultatele acestor teste sunt prezentate şi discutate în capitolul 4.

Având în vedere faptul că rezultatul comenzii time este de forma „4m35.900s”, pentru a putea converti rezultatul în forma în care era prezentat şi pentru testele realizate sub Windows, am aplicat formula:

*LEFT(F84;FIND("m";F84)-1)*60000+MID(F84;FIND("m";F84)+1;FIND(".";F84)-FIND("m";F84)-1)*1000+MID(F84;FIND(".";F84)+1;FIND("s";F84)-FIND(".";F84)-1)*

Trebuie menţionat că pe platforma PC5 nu s-a putut rula LiveCD şi nici instala o distribuţie Linux, deoarece această platformă prezenta două hard disc-uri în RAID, iar procesul de instalare pentru distribuţia Kubuntu întâmpina o eroare şi nu putea continua.

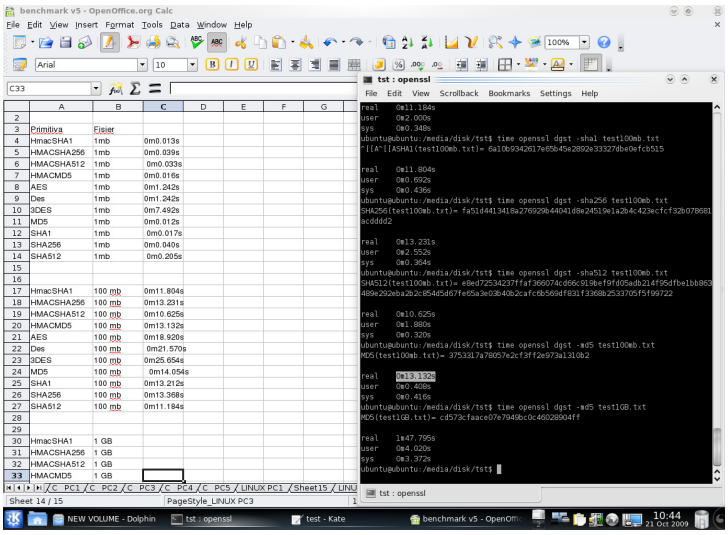

Figura 3.1.3.1 Benchmark in Kubuntu

3.1.4. Platformele de test.

Pentru rularea aplicaţiei au fost alese cinci platforme care constau în cinci sisteme informatice. Alegerea acestora a fost făcută astfel încât să poată fi testaţi algoritmii pe mai multe tipuri de procesoare şi sisteme. În continuare, se descriu cele 5 platforme alese pentru teste.

Prima platformă este un laptop Toshiba Satellite Pro A300 cu procesor Intel Core 2 Duo P8400 la 2.26GHz, memorie 3GB DDR2-800 (400 MHz) şi hard disc Western Digital 250 GBytes 5400 RPM. O descriere mai detaliată se găseşte în tabelul 3.1.4.1.

Denumire produs	Descriere
Notebook	TOSHIBA Satellite Pro A300
Procesor	Intel(R) Core(TM)2 Duo CPU P8400 @ 2.26GHz, CPU Code NameWolfdale, LGA775, L1 I-Cache32 KB L1 D-Cache32 KB L2 Cache3072 KB, One Physical Processor / 2 Cores / 2 Logical Processors / 64 bits
Memorie (3GB)	Slot 1 Manufacturer Samsung Capacity 2048 MBytes Memory Type DDR2 SDRAM SpeedDDR2-800 (400 MHz) Data Width 64 bits
	Slot 3 Manufacturer Samsung Capacity 1024 MBytes Memory TypeDDR2 SDRAM Speed DDR2-800 (400 MHz) Data Width 64 bits
Hard disc	Western Digital Size 250 GBytes Rotational Speed 5 400 RPM (Nominal) Current UDMA mode 6 (ATA-133)

Tabel 3.1.4.1 Descriere platformă 1 (informaţii obţinute cu SIW)

A doua platformă este un calculator PC cu procesor Intel Core 2 Duo E6750 la 2.66GHz memorie 4GB dual channel DDR2-666 (333 MHz) şi hard disc Seagate 500 GBytes 7200 RPM. O descriere mai detaliată se găseşte în tabelul 3.1.4.2.

Denumire produs	Descriere
PC	Personal Computer
Procesor	Intel (R) Core(TM)2 Duo CPU E6750 @ 2.66GHz , Code Name Conroe LGA775, L1 I-Cache32 KB L1 D-Cache32 KB L2 Cache4096 KB, L2 Cache Speed2666.69 MHz
Memorie (4GB)	Slot 1 Manufacturer Kingston Capacity 2048 MBytes DDR2 SDRAM SpeedDDR2-666 (333 MHz) Data Width64 bits
	Slot 3 Manufacturer Kingston Capacity 2048 MBytes DDR2 SDRAM SpeedDDR2-666 (333 MHz) Data Width64 bits
Hard disc	Vendor Seagate Model ST3500320AS SATA Size500 GBytes Current UDMA mode 5 (ATA-100) Rotational Speed 7200 RPM (Nominal)
Video Adapter	NVIDIA GeForce 8800 GT 512 Mbytes

Tabel 3.1.4.2 Descriere platformă 2 (informaţii obţinute cu SIW)

A treia platformă este un calculator PC HP Compaq dc7600 cu. Intel(R) Pentium(R) 4 CPU 3.00GHz memorie 512 MB dual channel DDR2 533 MHz şi hard disc Western Digital 80 GBytes. O descriere mai detaliată se găseşte în tabelul 3.1.4.3.

Denumire produs	Descriere
PC	HP Compaq dc7600
Procesor	Intel(R) Pentium(R) 4 CPU 3.00GHz CPU Code NamePrescott-2M LGA775 L1 T-Cache12 K L1 D-Cache16 KB L2 Cache2048 KB
Memorie	XMM1 ManufacturerJTAG Technologies Capacity256 MBytes DDR2 Speed533 MHz
	XMM3 ManufacturerJTAG Technologies Capacity256 MBytes DDR2 Speed533 MHz
Hard disc	Western Digital Model WDC WD800JD-60LSA5 Size 80 GB Current UDMA mode5 (ATA-100) Rotational Speed7200 RPM (Nominal)

Tabel 3.1.4.3 Descriere platformă 3 (informaţii obţinute cu SIW)

A patra platformă este un calculator PC cu procesor Intel Core 2 Duo E2140 la 1.60GHz memorie 1 GB dual channel DDR2-666 (333 MHz) şi hard disc Maxtor de 250 GBytes 7200 RPM. O descriere mai detaliată se găseşte în tabelul 3.1.4.4.

Denumire produs	Descriere
PC	Personal Computer
Procesor	Intel(R) Pentium(R) Dual CPU E2140 @ 1.60GHz CPU Code Name Conroe-1M LGA775MHz L1 I-Cache32 KB L1 D-Cache32 KB L2 Cache1024 KB
Memorie (3GB)	Slot 1 Manufacturer Nanya Technology Capacity512 MBytes DDR2 SDRAM DDR2-666 (333 MHz) Data Width64 bits
	Slot 2 Manufacturer Nanya Technology Capacity512 MBytes DDR2 SDRAM Speed DDR2-666 (333 MHz) Data Width64 bits
Hard disc	Maxtor MAXTOR STM3250820A Size250 GBytes Rotational Speed 7200 RPM (Nominal)

Tabel 3.1.4.4 Descriere platformă 4 (informaţii obţinute cu Sistem Info for Windows)

A cincea platformă este un calculator PC cu procesor Intel Core 2 Quad la 2.66GHz, memorie 4GB dual channel DDR2800 (400 MHz) şi hard disc Seagate 500 GBytes 7200 RPM. O descriere mai detaliată se găseşte în tabelul 3.1.4.5.

Denumire produs	Descriere
PC	Personal Computer
Procesor	Intel Core 2 Quad CPU Code NameYorkfield LGA775 L1 I-Cache32 KB L1 D-Cache32 KB L2 Cache3072 KB
Memorie (4GB)	Slot 1 Manufacturer Kingston Capacity2048 MBytes DDR2 SDRAM SpeedDDR2-800 (400 MHz) Data Width64 bits
	Slot 3 Manufacturer Kingston Capacity 2048 MBytes DDR2 SDRAM SpeedDDR2-800 (400 MHz) Data Width64 bits
Hard disc	2 HDD WD 1 TB RAID
Video Adapter	NVIDIA GeForce 9400 GT512 Mbytes

Tabel 3.1.4.5 Descriere platformă 5 (informaţii obţinute cu Sistem Info for Windows)

În continuare, ne vom referi la aceste platforme ca PC1, PC2, PC3, PC4, PC5. Cele cinci platforme au fost alese din următoarele motive:

- ✓ oferă procesoare din categorii diferite:
  - 1 procesor cu un singur nucleu (PC 3)
  - 2 procesoare cu două nuclee (PC2 şi PC4)
  - 1 procesor mobil cu două nuclee (Laptop PC1)
  - 1 procesor cu patru nuclee (PC5)
- ✓ oferă memorii cu caracteristici diferite şi de la producători diferiţi
  - Memorii dual channel (PC 2, PC4, PC5)
  - Memorii single chanel (PC1, PC3)
  - Memorie 512 MB (PC 3)
  - Memorie 3GB (PC 4)
  - Memorie 3 GB (PC1)
  - Memorie 4 GB (PC 2, PC5)
- ✓ oferă hard disc-uri de la producători diferiţi, cu capacităţi diferite şi viteze de rotaţie diferite

3.2. Aplicaţiile de criptare pentru date din memorie.

În subcapitolele 3.1.1,3.1.2,3.1.3, s-a discutat despre aplicaţii folosite pentru evaluarea performanţelor în funcţie de timp ale algoritmilor criptografici descrişi în tema acestei lucrări. Aceşti algoritmi sunt testaţi în condiţiile în care sunt folosiţi pentru criptarea datelor care se află stocate pe un hard disc. Dimensiunea fişierelor este foarte mare (ajung la dimensiuni de 10 Gb). În aceste teste la timpul de criptare se mai adaugă şi timpul de acces (citire/ scriere) la datele care se află pe

hard disc. După cum se poate vedea şi din rezultatele prezentate în capitolul 4, timpii variază în funcţie de viteza de citire/ scriere a hard disc-urilor din platformele de test. În această secţiune se reiau testele pentru algoritmii amintiţi fiind simulată criptarea datelor „on the fly”. Se rulează algoritmii „în gol” folosind recursivitatea astfel *buffer = algoritm_criptare(buffer)*. Numărul de iteraţii ales a fost 1.000.000. Algoritmii au fost implementaţi în Visual Basic, C#, Java şi rulaţi pe sistem de operare Windows XP SP3.

3.2.1. Aplicaţia în Visual Basic

În Visual Basic s-a plecat de la bibliotecile standard puse la dispoziţie de acest mediu de programare. Constructorii, funcţiile, clasele, câmpurile, metodele, proprietăţile, sunt descrise în tabelele 3.1.1.1-3.1.1.7.

În figura 3.2.1.1 este prezentată aplicaţia. La apăsarea butonului „generare”, se rulează fiecare algoritm de un număr de 1.000.000 de ori. Se măsoară timpul necesar de rulare, iar apoi se împarte la numărul de iteraţii, obţinându-se în acest mod, timpul mediu de rulare al algoritmului pentru o iteraţie.

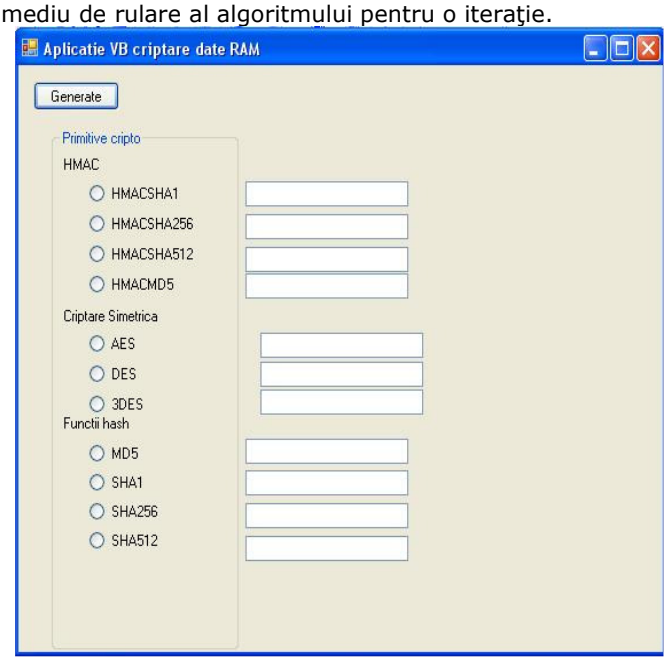

Figura 3.2.1.1 Aplicaţie VB. Criptare date din RAM

Pentru început, se generează o valoare aleatorie ce va fi folosită pentru criptare. Codul este prezentat în cele ce urmează:

```
Dim RandomNumber As Integer = Int((1000000 - 0 + 1) * Rnd())
Dim aux As String = getMd5hash(Str(RandomNumber))
```

Algoritmii sunt grupaţi în trei categorii: funcţii hash, funcţii HMAC şi algoritmi de criptare simetrici. Se va detalia câte un algoritm din fiecare categorie.

Algoritmul DES din categoria algoritmilor de criptare simetrici este implementat astfel[MSDN]:

```
Public Shared Function Encrypt(ByVal PlainText As String, ByVal key As
SymmetricAlgorithm) As Byte()
Dim ms As New MemoryStream()
Dim encStream As New CryptoStream(ms, key.CreateEncryptor(),
CryptoStreamMode.Write)
Dim sw As New StreamWriter(encStream)
sw.WriteLine(PlainText)
sw.Close()
encStream.Close()
Dim buffer As Byte() = ms.ToArray()
ms.Close()
Return buffer
End Function
```

Iar funcţia este apelată astfel:

```
        Dim Des As New DESCryptoServiceProvider()
    Dim now As Date = Date.Now
    For i = 0 To NrIteratii
       buffer = Encrypt(buffer, Des)
    Next
    Dim after As Date = Date.Now
    Dim time As TimeSpan = after - now
    Dim timp As Double = time.TotalMilliseconds / NrIteratii
```

Timpul de calcul este obţinut în mod asemănător cu timpii calculaţi pentru aplicaţiile de benchmark pentru fişiere de mari dimensiuni, singura deosebire fiind aceea că, în acel caz se obţinea timpul pentru o rulare, iar aici prin împărţirea timpului la numărul de iteraţii se obţine un timp mediu pentru rularea algoritmului.

Algoritmul HMAC SHA256 din categoria HMAC este implementat astfel [MSDN]:

```
Function getHMACsha256hash(ByVal input As String) As String
Dim HMACsha256hasher As New HMACSHA256(key:=_key)
Dim data As Byte() =
HMACsha256hasher.Computehash(Encoding.Default.GetBytes(input))
Dim sBuilder As New StringBuilder()
Dim i As Integer
  For i = 0 To data.Length - 1
       sBuilder.Append(data(i).ToString("x2"))
  Next i
Return sBuilder.ToString()
End Function
```

Iar funcţia este apelată astfel:

```
    now = Date.Now
    For i = 0 To NrIteratii
       aux = getHMACsha256hash(aux)
    Next
    after = Date.Now
    time = after - now
    timp = time.TotalMilliseconds / NrIteratii
```

Algoritmul MD5 din categoria funcţiilor hash este implementat astfel[MSDN]:

```
Function getMd5hash(ByVal input As String) As String
```

```
Dim md5hasher As MD5 = MD5.Create()
Dim data As Byte() = md5hasher.Computehash(Encoding.Default.GetBytes(input))
Dim sBuilder As New StringBuilder()
Dim i As Integer
  For i = 0 To data.Length - 1
          sBuilder.Append(data(i).ToString("x2"))
  Next i
Return sBuilder.ToString()
End Function
```

Iar funcţia este apelată astfel:

```
          now = Date.Now
       For i = 0 To NrIteratii
          aux = getMd5hash(aux)
       Next
       after = Date.Now
       time = after - now
       timp = time.TotalMilliseconds / NrIteratii
```

Rezultatele testelor sunt prezentate în capitolul 4.

3.2.2. Aplicaţia în C#

În dezvoltarea aplicaţiei în C# s-a gândit în mod similar celei în Visual Basic, folosind tot bibliotecile standard oferite de Visual Studio 2008 în System. Security. Cryptography. Clasele folosite sunt aceleaşi ca cele din capitolul 3.1 (vezi tabel 3.1.1.1, 3.1.1.2, 3.1.1.3, 3.1.1.4, 3.1.1.5, 3.1.1.6, 3.1.1.7). În figura 3.2.2.1, este prezentată o captură de imagine a aplicaţiei în C#. După cum se poate observa, atât aplicaţia în Visual Basic cât şi cea în C#, au fost dezvoltate în mod similar, atât din punct de vedere vizual, funcţional, cât şi din punctul de vedere al programării.

Pentru început, se generează o valoare aleatorie ce va fi folosită pentru criptare. Codul este similar celui descris în subcapitolul 3.2.1.

Algoritmii sunt grupaţi în trei categorii: funcţii hash, funcţii HMAC şi algoritmi de criptare simetrici. Se va detalia câte un algoritm din fiecare categorie.

Algoritmul AES din categoria algoritmilor de criptare simetrici este implementat astfel [MSDN]:

```
public static byte[] Encrypt(string PlainText, SymmetricAlgorithm key)
      {
         MemoryStream ms = new MemoryStream();
         CryptoStream encStream = new CryptoStream(ms, key.CreateEncryptor(),
CryptoStreamMode.Write);
         StreamWriter sw = new StreamWriter(encStream);
         sw.WriteLine(PlainText);
         sw.Close();
         encStream.Close();
         byte[] buffer = ms.ToArray();
         ms.Close();
         return buffer;
      }
```

Iar funcţia este apelată astfel:

```
                Aes aes = new AesCryptoServiceProvider();
```

```
DateTime startTime = DateTime.Now;
   for (int i = 0; i <=NrIteratii ; i++)
   {
                byte[] buffer = Encrypt(aux, aes);
   }
DateTime stopTime = DateTime.Now;
TimeSpan duration = stopTime - startTime;
double timp = duration.TotalMilliseconds/NrIteratii;
```

Algoritmul HMAC SHA512 din categoria hash MAC este implementat astfel [MSDN]:

```
public static string gethmacsha512hash(string input)
    {
       HMACSHA512 hmacSHA512hasher = new HMACSHA512();
       byte[] data =
hmacSHA512hasher.Computehash(Encoding.Default.GetBytes(input));
       return data.ToString();
    }
```

Iar funcţia este apelată astfel:

```
            startTime = DateTime.Now;
       for (int i = 0; i <= NrIteratii; i++)
       {
          aux = gethmacsha512hash(aux);
       }
       stopTime = DateTime.Now;
       duration = stopTime - startTime;
       timp = duration.TotalMilliseconds / NrIteratii;
       TextBox3.Text = timp.ToString();
```

Algoritmul MD5 din categoria funcţiilor hash este implementat astfel[MSDN]:

```
public static string getMd5hash(string input)
    {
       MD5 md5hasher = MD5.Create();
       byte[] data = md5hasher.Computehash(Encoding.Default.GetBytes(input));
       StringBuilder sBuilder = new StringBuilder();
       for (int i = 0; i < data.Length; i++)
       {
          sBuilder.Append(data[i].ToString("x2"));
       }
       return sBuilder.ToString();
    }
```

Iar funcţia este apelată astfel:

```
            startTime = DateTime.Now;
       for (int i = 0; i <= NrIteratii; i++)
       { aux = getMd5hash(aux); }
       stopTime = DateTime.Now;
       duration = stopTime - startTime;
       timp = duration.TotalMilliseconds / NrIteratii;
```

Rezultatele testelor sunt prezentate în capitolul 4.

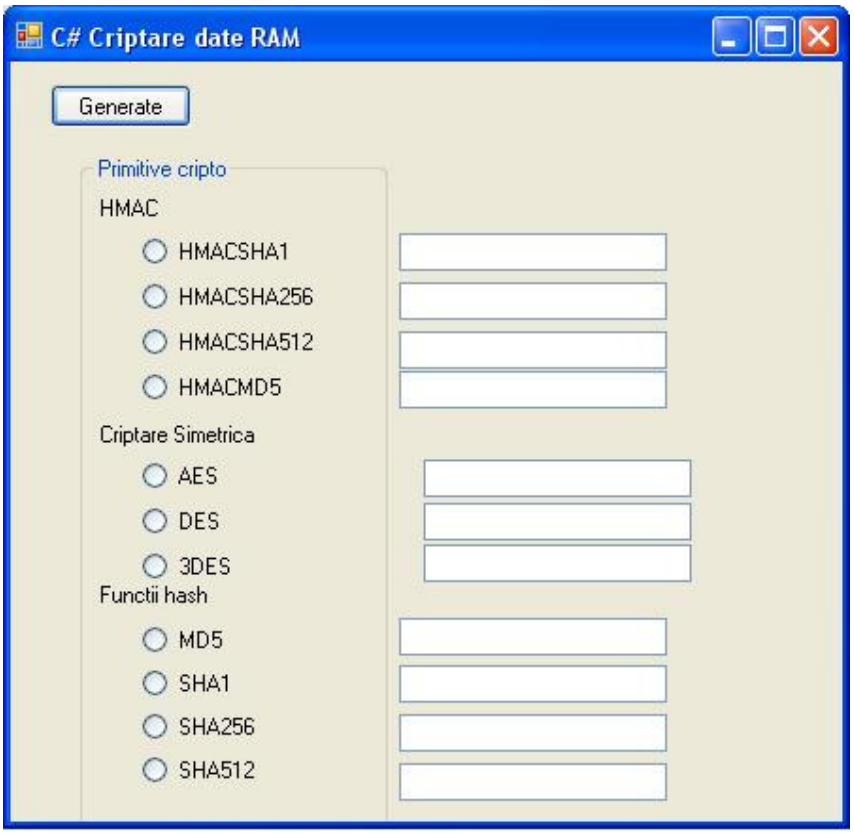

Figura 3.2.2.1 Aplicaţie C#. Criptare date din RAM

3.2.3. Aplicaţia în Java.

Uneltele de bază pentru dezvoltarea de aplicaţii Java criptografice sunt oferite de JCA (Java Cryptography Architecture) şi JCE (Java Cryptography Extension). Acestea oferă dezvoltatorului acces direct la algoritmi criptografici prin folosirea aşa-numitelor *„factory classes"*[HOO05]. Aspectul practic al celor enunţate mai devreme este acela că aplicaţia invocă unele clase interne, iar acestea vor asigura funcţionalitatea solicitată de aplicaţie.

Pentru mediul de dezvoltare Java sunt folosite în dezvoltarea aplicaţiei clasele folosite de [SOL08] : javax.crypto şi java.security. Pentru implementarea aplicaţiei a fost folosit Java JDK 1.6.0.17 alături de platforma Eclipse 3.1.2.

Folosirea claselor se face prin :

```
import javax.crypto.*;
import java.security.*;
```

În figura 3.2.3.1 este prezentat mediul Eclipse.

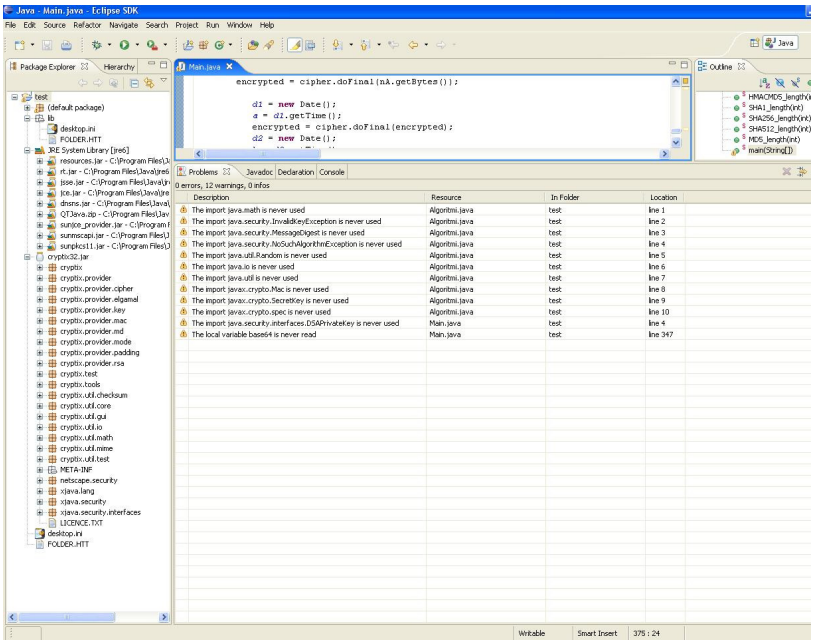

Figura 3.2.3.1 Eclipse 3.1.2

Cu ajutorul JCA şi JCE, nu mai este nevoie de mii de linii de cod pentru implementarea unui algoritm criptografic sau a unei funcţii hash sau MAC, ci este suficientă invocarea clasei, generarea unei chei (acolo unde este cazul) şi apelarea unei metode a clasei invocate. Astfel, programatorul solicită un algoritm unui *„cryptographic service provider”* (sau doar provider). Acesta returnează funcţionalitatea solicitată. O implementare simplă şi rapidă este următoarea [WEI08]:

MessageDigest bcMD5Implementation= MessageDigest .getInstance(“MD5”, “BC”);

Folosind clasa javax.crypto.Cipher putem cripta sau decripta date folosind un cifru din cele încorporate. **Cipher** fiind o clasă abstractă nu poate fi invocată direct. Folosirea acesteia se poate realiza în trei paşi [KNU04]:

- ✓ Se obţine un cifru folosind metoda *getInstance()*,
- ✓ Se iniţializează cifrul folosind *init(),* un modul şi o cheie,
- ✓ Se criptează/ decriptează datele folosind *update()* sau *doFinal().*

Un exemplu este [KNU04]:

```
Cipher cipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
cipher.init(Cipher.ENCRYPT_MODE, key);
byte[] raw = cipher.doFinal(stringBytes);
```

Similar lui [KNU04], [WEI08] şi [SOL08] codul pentru implementarea unui algoritm din cei descrişi şi la capitolele anterioare este următorul (pentru restul algoritmilor codul fiind similar):

```
mdsha1= null;
mdbis = null;
try {
    mdsha1= MessageDigest.getInstance("SHA-1");
```

```
            mdig.update(buffer);
            mdbis = mdig.digest();
            } catch (NoSuchAlgorithmException ex) {
            ex.printStackTrace();
        }
          mdig.update(mdbis);
          mdbis = mdig.digest(mdbis);
```

Pentru calculul unei funcţii hash folosind doar clasele javax.crypto şi java.security [KNU04] se poate folosi codul:

```
MessageDigest md = MessageDigest.getInstance("SHA");
md.update(inputData);
    byte[] digest = md.digest();
```

3.2.4. Platformele de test. Date din memorie

Pentru rularea aplicaţiilor descrise în capitolul 3.2 şi care se referă la testarea timpilor computaţionali ai primitivelor criptografice descrise au fost alese trei platforme care constau din trei sisteme informatice. Alegerea acestora a fost făcută astfel încât să poată fi testaţi algoritmii pe mai multe tipuri de procesoare din familii diferite. În continuare, se descriu cele 3 platforme alese pentru teste.

Prima platformă este un calculator PC cu processor Intel Core 2 Duo E6750 la 2.66GHz memorie 4GB dual channel DDR2-666 (333 MHz) şi hard disc Seagate 500 GBytes 7200 RPM. O descriere mai detaliată se găseşte în tabelul 3.2.4.1

Denumire produs	Descriere
PC	Personal Computer
Procesor	Intel (R) Core(TM)2 Duo CPU E6750 @ 2.66GHz , Code Name Conroe LGA775, L1 I-Cache32 KB L1 D-Cache32 KB L2 Cache4096 KB, L2 Cache Speed2666.69 MHz
Memorie (4GB)	Slot 1 Manufacturer Kingston Capacity 2048 MBytes DDR2 SDRAM SpeedDDR2-666 (333 MHz) Data Width64 bits
	Slot 3 Manufacturer Kingston Capacity 2048 MBytes DDR2 SDRAM SpeedDDR2-666 (333 MHz) Data Width64 bits

Tabel 3.2.4.1 Descriere platformă 1 (informaţii obţinute cu Sistem Info for Windows)

A doua platformă este un Netbook marca MSI cu processor Atom N270 la 1.60GHz memorie 1GB dual channel DDR2-533 (266 MHz). O descriere mai detaliată se găseşte în tabelul 3.2.4.2

Denumire produs	Descriere
Netbook	NetBook
Procesor	Intel(R) Atom(TM) CPU N270 @ 1.60GHz Dothan L2 Cache Speed 1600.02 MHz L1 I-Cache 32 KB L2 Cache 512 KB
Memorie (1GB)	Slot 2 Manufacturer Samsung 1024 MBytes DDR2 SDRAM DDR2-666 (333 MHz)

Tabel 3.2.4.2 Descriere platformă 2 (informaţii obţinute cu Sistem Info for Windows)

A treia platformă este un Laptop marca Dell Inspiron 9400 cu processor Intel Core Duo T2400 la 1.83GHz memorie 1GB dual channel DDR2-533 (266 MHz). O descriere mai detaliată se găseşte în tabelul 3.2.4.3

Denumire produs	Descriere
Laptop	Laptop
Procesor	Intel Core Duo Genuine Intel(R)CPUT2400 @ 1.83GHz Yonah DC L1 I-Cache 32 KB L1 D-Cache 32 KB L2 Cache 2048 KB L2 Cache Speed 1828.77 MHz
Memorie (1GB)	Slot 1 Manufacturer Samsung Capacity 512 MBytes Memory Type DDR2 SDRAM Speed DDR2-533 (266 MHz) Data Width 64 bits
	Slot 3 Manufacturer Samsung Capacity 512 MBytes Memory Type DDR2 SDRAM Speed DDR2-533 (266 MHz) Data Width 64 bits

Tabel 3.2.4.3 Descriere platformă 3 (informaţii obţinute cu Sistem Info for Windows)

3.3. Concluzii

În cadrul acestui capitol s-au prezentat mediile de dezvoltare folosite pentru implementarea algoritmilor criptografici care urmează să fie testaţi.

S-a realizat o justificare a alegerii acestor medii de dezvoltare punctând avantajele fiecărui mediu ales.

S-a descris în detaliu fiecare etapă de implementare a primitivelor criptografice şi au fost prezentate capturi de ecran cu aplicaţiile finale.

S-au prezentat clasele folosite de către limbajele de programare pentru a se putea justifica similaritatea funcţională dintre aplicaţiile vizuale dezvoltate. Totodată s-au descris şi tool-urile folosite în dezvoltarea aplicaţiilor.

S-au prezentat aplicaţiile dezvoltate pentru ambele tipuri de teste: date din memorie (de mici dimensiuni) şi date de pe hard disc (volume mari de date).

S-au prezentat exemple de cod sursă în diferite limbaje de programare pentru diferite primitive criptografice alese.

S-a realizat o descriere detaliată a tuturor platformelor alese pentru teste şi la final s-a justificat alegerea acestora.

4. Rezultate experimentale

În acest capitol sunt prezentate rezultatele obţinute în urma testelor de benchmarking descrise în capitolul anterior la punctele 3.1 şi 3.2 aplicate platformelor prezentate în 3.5. Testele au fost efectuate în două etape. Prima etapă a constat în evaluarea performanţei algoritmilor care au fost rulaţi pe fişiere de dimensiune de 1 MB şi 100Mb. A doua etapă a fost mai amplă şi s-a concretizat în rularea aplicaţiilor pe fişiere de 1 GB, 2GB, 3 GB, ...10 GB.

4.1. Rezultate Visual Basic

4.1.1. Fişiere de 1 MB şi 100 MB

În tabelul 4.1.1.1 sunt prezentate rezultatele testelor efectuate pe fişierul de 1 MB. Rezultatele sunt exprimate în milisecunde.

Algoritm	PC1	PC2	PC3	PC4	PC5
HmacSHA1	8,5070	31,0000	7,8100	7,8126	7,8127
HMACSHA256	40,8648	54,5000	23,4400	62,5004	15,6254
HMACSHA512	62,1795	164,0000	62,5000	148,4385	31,2508
HMACMD5	8,7328	7,5000	7,8100	15,6251	7,8127
AES	46,1711	187,5000	140,6300	156,2510	112,5457
Des	24,4031	46,5000	85,9400	46,8753	132,8159
3DES	62,3980	93,5000	117,1900	148,4385	179,6921
MD5	20,3000	7,5000	7,8100	7,8128	7,8127
SHA1	18,7000	7,5000	7,8100	7,8128	7,8127
SHA256	14,0000	15,5000	15,6300	23,4384	15,6254
SHA512	39,0000	39,0000	39,0600	70,3152	15,6254

Tabel 4.1.1.1 Rezultate timpi de computaţie algoritmi pe fişier 1 MB.

În tabelul 4.1.1.2 sunt prezentate rezultatele testelor efectuate pe fişierul de 100 MB. Rezultatele sunt exprimate în milisecunde.

Algoritm	PC1	PC2	PC3	PC4	PC5
HmacSHA1	126,4688	398,0000	617,1900	671,9008	593,7652
HMACSHA256	2424,2031	3460,5000	6000,0000	5773,6592	1898,4861
HMACSHA512	5917,3773	8796,5000	14062,5000	14594,3104	2976,6387
HMACMD5	265,2420	343,5000	507,8100	585,9600	343,7588
AES	9344,7651	4867,0000	16234,3800	9070,6608	8216,6951
Des	6726,8781	2664,0000	7414,0600	7007,9471	9226,7987
3DES	4721,5618	2617,0000	11203,1300	7343,8910	5742,3345
MD5	62,0000	171,5000	640,6300	289,0681	171,8794
SHA1	67,1000	195,0000	281,2500	328,1313	296,8826
SHA256	43,9000	1718,5000	2937,5000	2906,3058	937,5240
SHA512	1021,8000	4398,0000	6750,0000	7476,7061	1484,4130

Tabel 4.1.1.2 Rezultate timpi de computaţie algoritmi pe fişier 100 MB.

4.1.2. Fişiere de dimensiuni mari

În tabelul 4.1.2.1 sunt prezentate rezultatele testelor efectuate pe fişierul de 1 GB. Rezultatele sunt exprimate în secunde.

Algoritm	PC1	PC2	PC3	PC4	PC5
HmacSHA1	3,007578	9,414000	27,421880	38,859624	7,711135
HMACSHA256	24,995300	35,617000	61,921880	64,148848	19,352058
HMACSHA512	50,360156	90,101500	138,000000	154,336925	30,438279
HMACMD5	2,364533	1,765500	15,570310	19,476687	1,734419
AES	31,554768	50,148000	147,250000	110,735084	169,518245
Des	21,807813	27,039000	107,156250	99,699681	199,708237
3DES	19,492578	27,234000	166,351560	96,588125	195,036243
MD5	4,406000	1,765500	11,726560	23,125854	1,742232
SHA1	4,953000	1,984000	11,578130	19,653986	3,078204
SHA256	40,656000	17,671500	31,492190	31,722908	9,547119
SHA512	91,453000	44,835500	69,656250	76,329117	15,211327

Tabel 4.1.2.1 Rezultate timpi de computaţie algoritmi pe fişier 1 GB.

În tabelul 4.1.2.2 sunt prezentate rezultatele testelor efectuate pe fişierul de 2 GB. Rezultatele sunt exprimate în secunde.

Algoritm	PC1	PC2	PC3	PC4	PC5
HmacSHA1	7,471050	14,078000	47,851560	73,715447	17,500448
HMACSHA256	46,753100	71,263000	137,359380	130,437676	38,704116
HMACSHA512	111,091400	181,992000	588,695310	304,290796	61,032812
HMACMD5	5,326550	7,109000	51,414060	74,694834	7,023617
AES	112,351550	69,854000	441,210940	226,740736	368,231472
Des	84,803100	113,664000	1168,328130	190,944090	388,283377
3DES	85,294500	99,390500	0,000000	188,680976	401,658720
MD5	4,762500	6,890500	101,812500	36,869075	9,828377
SHA1	4,346800	4,015500	211,562500	42,055226	6,054843
SHA256	11,218700	35,367000	60,687500	61,758603	19,430185
SHA512	22,395300	89,843500	137,757810	153,860532	30,633597

Tabel 4.1.2.2 Rezultate timpi de computaţie algoritmi pe fişier 2GB.

În tabelul 4.1.2.3 sunt prezentate rezultatele testelor efectuate pe fişierul de 3 GB. Rezultatele sunt exprimate în secunde.

Algoritm	PC1	PC2	PC3	PC4	PC5
HmacSHA1	31,848400	24,406000	247,609380	107,148438	63,087553
HMACSHA256	68,896050	131,054500	181,710940	192,685023	62,251594
HMACSHA512	155,396850	287,195000	415,859380	459,648644	106,682419
HMACMD5	31,409350	35,570000	73,031250	112,562500	30,867978
AES	140,359350	217,921500	536,546880	307,815874	651,242184
Des	104,298400	246,953000	256,429690	252,446028	671,946889
3DES	99,543750	247,562500	285,054690	256,148438	661,055985
MD5	5,704600	17,632500	36,289060	52,890625	15,820718
SHA1	5,679600	18,000000	36,468750	67,503888	14,969133
SHA256	12,514000	53,234000	90,054690	91,644549	34,149312
SHA512	27,689000	171,742000	207,265630	239,697478	49,329388

Tabel 4.1.2.3 Rezultate timpi de computaţie algoritmi pe fişier 3GB.

În tabelul 4.1.2.4 sunt prezentate rezultatele testelor efectuate pe fişierul de 4 GB. Rezultatele sunt exprimate în secunde.

Algoritm	PC1	PC2	PC3	PC4	PC5
HmacSHA1	45,457800	108,664000	139,703130	152,797310	42,126078
HMACSHA256	95,607000	158,078000	250,289060	244,703539	133,065906
HMACSHA512	233,983550	398,226500	556,218750	606,413767	37,469709
HMACMD5	45,853900	67,710500	70,687500	144,835938	39,626067
AES	199,298400	263,921500	600,976560	503,500000	721,120518
Des	141,260900	331,031000	401,914060	355,703125	769,113439
3DES	141,058550	370,734000	360,507810	399,333586	1089,637269
MD5	49,962500	20,937500	35,781250	73,657470	21,953687
SHA1	29,662500	21,242000	84,062500	71,532034	19,234867
SHA256	17,082800	72,812500	119,328130	125,492964	41,329183
SHA512	37,331200	181,835500	274,593750	327,055129	66,298572

Tabel 4.1.2.4 Rezultate timpi de computaţie algoritmi pe fişier 4GB.

În tabelul 4.1.2.5 sunt prezentate rezultatele testelor efectuate pe fişierul de 5 GB. Rezultatele sunt exprimate în secunde.

Algoritm	PC1	PC2	PC3	PC4	PC5
HmacSHA1	55,900750	81,554500	138,796880	177,248183	83,814646
HMACSHA256	113,733550	190,703000	299,601560	299,640625	116,049846
HMACSHA512	251,964800	474,960500	818,765630	758,413370	201,825479
HMACMD5	56,000000	53,710500	141,703130	176,828420	74,251901
AES	263,456250	312,000000	925,289060	534,878423	997,138236
Des	182,730450	413,203000	579,125000	466,773871	1098,028109
3DES	181,389800	432,007500	575,367190	422,202545	1230,515875
MD5	10,146800	27,562500	69,375000	89,352134	53,079484
SHA1	10,175000	40,109000	70,359380	88,399003	79,736416
SHA256	20,654600	58,781000	220,843750	149,400616	236,162296
SHA512	45,832800	225,312500	1082,343750	376,362389	192,692433

Tabel 4.1.2.5 Rezultate timpi de computaţie algoritmi pe fişier 5GB.

În tabelul 4.1.2.6 sunt prezentate rezultatele testelor efectuate pe fişierul de 6GB. Rezultatele sunt exprimate în secunde.

Algoritm	PC1	PC2	PC3	PC4	PC5
HmacSHA1	134,757500	67,132500	257,046875	209,860718	37,547836
HMACSHA256	259,867000	214,500000	359,328125	373,564891	160,269728
HMACSHA512	558,257500	541,812500	846,125000	921,153009	182,598424
HMACMD5	138,335500	64,492000	114,765625	211,385973	23,953738
AES	612,773000	474,264000	1045,828125	798,426985	989,788533
Des	386,484000	525,695000	534,671875	527,410815	1074,089996
3DES	380,531000	545,492000	494,093750	508,737015	1134,638421
MD5	13,821800	34,546500	116,218750	220,985789	23,297471
SHA1	13,521800	32,625000	131,109375	209,203798	37,157201
SHA256	25,845300	106,351500	482,984375	361,645456	114,362303
SHA512	56,926500	269,703000	917,640625	903,892258	182,973434

Tabel 4.1.2.6 Rezultate timpi de computaţie algoritmi pe fişier 6GB.

În tabelul 4.1.2.7 sunt prezentate rezultatele testelor efectuate pe fişierul de 7GB. Rezultatele sunt exprimate în secunde.

Algoritm	PC1	PC2	PC3	PC4	PC5
HmacSHA1	316,726500	70,750000	1032,156250	249,687500	91,471092
HMACSHA256	436,453000	249,554500	425,203125	424,715297	145,566226
HMACSHA512	763,312500	631,343500	978,656250	1059,151482	217,943079
HMACMD5	304,468500	70,890500	153,406250	245,675020	64,517277
AES	779,906000	435,609000	1232,281250	705,884035	1324,678824
Des	593,273000	644,226500	578,234375	597,382646	1759,763799
3DES	587,929500	571,281000	760,437500	610,226561	1311,205441
MD5	30,012500	35,562500	145,171875	269,519075	62,954737
SHA1	31,187500	35,406000	147,625000	249,096938	97,283740
SHA256	43,595300	123,906000	432,937500	416,317829	141,128613
SHA512	77,232800	315,289000	1151,812500	1048,794674	218,849352

Tabel 4.1.2.7 Rezultate timpi de computaţie algoritmi pe fişier 7GB.

În tabelul 4.1.2.8 sunt prezentate rezultatele testelor efectuate pe fişierul de 8 GB. Rezultatele sunt exprimate în secunde.

Algoritm	PC1	PC2	PC3	PC4	PC5
HmacSHA1	180,648000	105,820000	1238,587500	285,2849	462,582154
HMACSHA256	348,765500	287,984000	510,243750	476,0060	384,009830
HMACSHA512	747,046500	726,445000	1174,387500	1263,7036	286,257328
HMACMD5	180,742000	100,632500	184,087500	282,3629	84,127154
AES	714,140500	578,796500	1478,737500	1101,7920	1759,483873
Des	528,296500	650,523000	693,881250	690,0340	2002,035626
3DES	501,093500	661,281000	912,525000	694,4770	1952,737489
MD5	18,062500	50,593500	174,206250	281,5160	81,080201
SHA1	18,040600	50,382500	177,150000	279,8136	107,440250
SHA256	34,701500	141,656000	519,525000	484,0312	324,523933
SHA512	74,340000	358,851500	1382,175000	1206,8792	274,975789

Tabel 4.1.2.8 Rezultate timpi de computaţie algoritmi pe fişier 8GB.

În tabelul 4.1.2.9 sunt prezentate rezultatele testelor efectuate pe fişierul de 9 GB. Rezultatele sunt exprimate în secunde.

Algoritm	PC1	PC2	PC3	PC4	PC5
HmacSHA1	281,359000	113,914000	1341,803125	357,8212	114,562500
HMACSHA256	466,226500	320,156000	552,764063	577,3158	407,375000
HMACSHA512	905,085500	817,328000	1272,253125	1368,7943	385,515625
HMACMD5	279,601500	111,453000	199,428125	341,3286	88,171875
AES	884,679500	606,439000	1601,965625	941,5367	1962,787724
Des	660,609000	729,912000	751,704688	781,3841	1992,609375
3DES	665,593500	758,125000	988,568750	788,2159	1951,406250
MD5	28,971800	55,882500	188,723438	315,7795	98,000000
SHA1	28,762000	55,804500	191,912500	321,0171	87,375000
SHA256	48,039000	161,992000	562,818750	583,3855	205,078125
SHA512	128,048400	404,140500	1497,356250	1384,6963	291,343750

Tabel 4.1.2.9 Rezultate timpi de computaţie algoritmi pe fişier 9GB.

În tabelul 4.1.2.10 sunt prezentate rezultatele testelor efectuate pe fişierul de 10 GB. Rezultatele sunt exprimate în secunde.

Algoritm	PC1	PC2	PC3	PC4	PC5
HmacSHA1	210,789000	198,453000	1455,340313	435,95591	320,930091
HMACSHA256	583,601500	355,890500	599,536406	669,78553	600,562249
HMACSHA512	958,414000	898,726500	1379,905313	1559,22254	373,915822
HMACMD5	208,546500	113,781000	216,302813	444,67472	224,224490
AES	973,304500	615,750000	1737,516563	1131,36661	2103,245324
Des	658,203000	875,914000	815,310469	953,05297	2436,859257
3DES	667,351500	900,835500	1072,216875	947,41939	2126,507562
MD5	20,487000	53,252500	204,692344	437,12316	127,175131
SHA1	20,390000	83,710500	208,151250	434,89340	165,129227
SHA256	43,125000	195,273000	610,441875	752,48919	200,645761
SHA512	95,210900	465,054500	1624,055625	1539,47231	361,946766

Tabel 4.1.2.10 Rezultate timpi de computaţie algoritmi pe fişier 10GB.

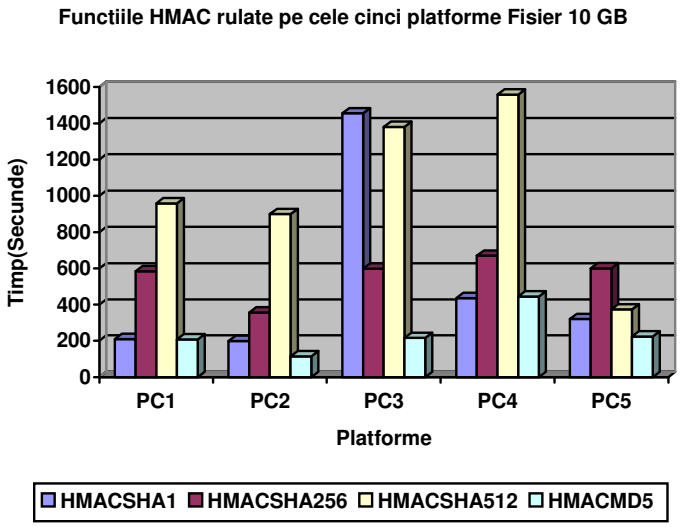

Figura 4.1.2.1 Grafic funcţii HMAC pe fişier 10GB

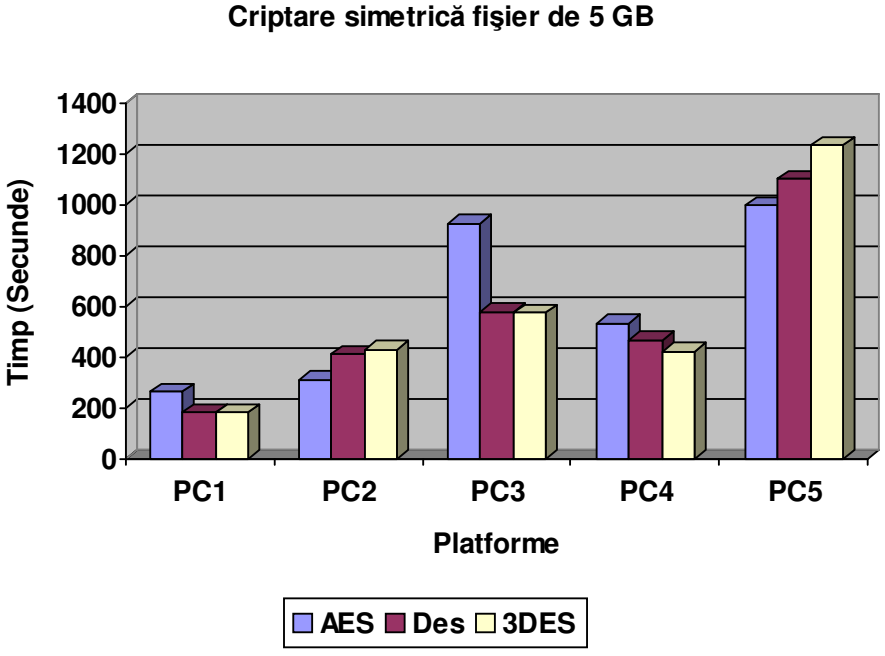

Figura 4.1.2.2 Grafic criptare simetrică pe fişier 5GB

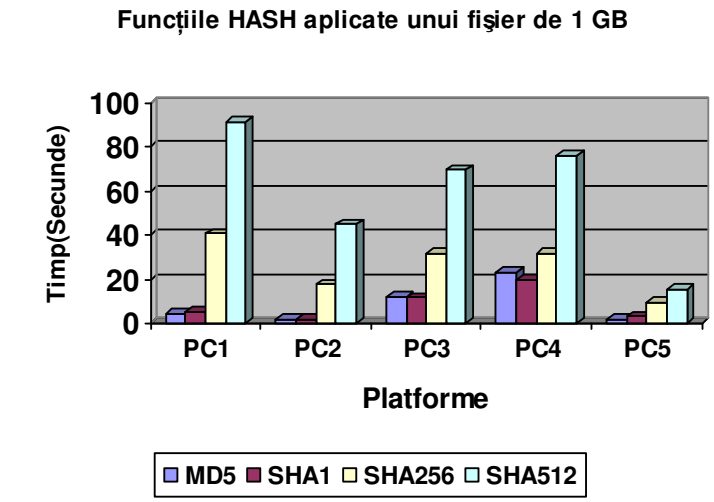

Figura 4.1.2.3 Funcţii hash aplicate pe fişier de 1 GB

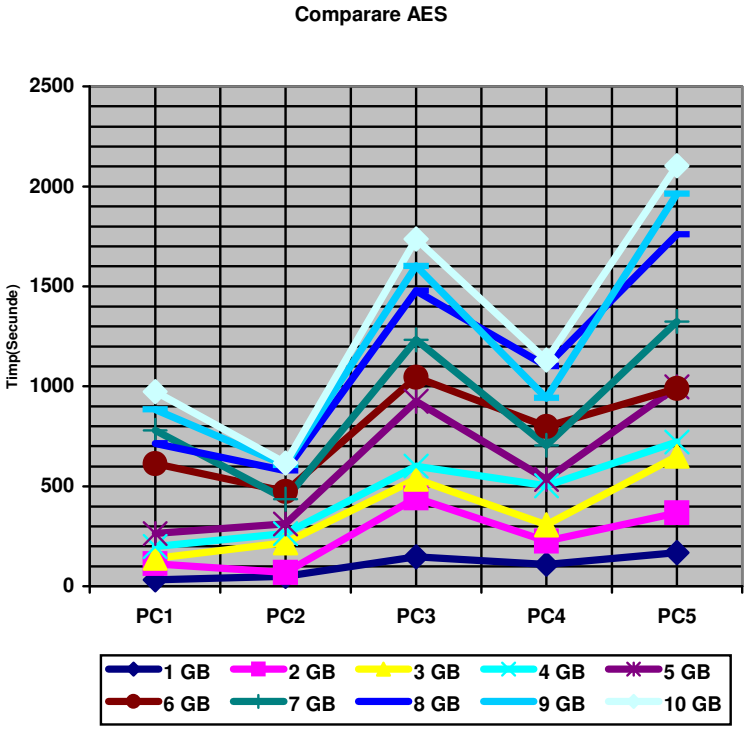

Figura 4.1.2.4 Algoritm AES

În figura 4.1.2.4, se poate observa că pentru algoritmul AES platforma PC2 are, per ansamblu, cele mai bune performanţe. Un aspect surprinzător este acela că platforma PC5 a avut cei mai mari timpi de computaţie pentru acest algoritm. La platforma PC3 era de aşteptat ca timpii necesari execuţiei algoritmilor să fie relativ mari în comparaţie cu platformele bazate pe procesoare cu mai multe nuclee.

4.2. Rezultate C#

4.2.1. Fişiere de 1 MB şi 100 MB

În tabelul 4.2.1.1 sunt prezentate rezultatele testelor efectuate pe fişierul de 1 MB. Rezultatele sunt exprimate în milisecunde.

Algoritm	PC1	PC2	PC3	PC4	PC5
HmacSHA1	15,625	15,6250	265,6250	265,6250	1546,9146
HMACSHA256	46,875	31,2500	78,1250	46,8750	46,8762
HMACSHA512	109,375	78,1250	359,3750	265,6250	109,3778
HMACMD5	15,625	15,6250	15,6250	31,2500	15,6254
AES	46,875	15,6250	406,2500	93,7500	112,5457
Des	31,250	46,8750	93,7500	46,8750	93,7524
3DES	78,125	62,5000	109,3750	125,0000	421,8858
MD5	15,625	15,6250	15,6250	15,6250	15,6254
SHA1	15,625	15,6250	15,6250	15,6250	15,6254
SHA256	31,250	31,2500	62,5000	46,8750	31,2508
SHA512	93,750	93,7500	171,8750	140,6250	31,2508

Tabel 4.2.1.1 Rezultate timpi de computaţie algoritmi pe fişier 1 MB.

În tabelul 4.2.1.2 sunt prezentate rezultatele testelor efectuate pe fişierul de 100 MB. Rezultatele sunt exprimate în milisecunde.

Algoritm	PC1	PC2	PC3	PC4	PC5
HmacSHA1	562,500	390,6250	1796,8750	3625,0000	2234,4322
HMACSHA256	4171,875	3453,1250	5890,6250	5734,3750	1968,8004
HMACSHA512	9312,500	8781,2500	13546,8750	14828,1250	393,8292
HMACMD5	484,375	343,7500	515,6250	593,7500	343,7588
AES	10140,625	2437,5000	9203,1250	6281,2500	8216,6951
Des	8843,750	3281,2500	5937,5000	4562,5000	4359,4866
3DES	8453,125	5921,8750	8046,8750	10531,2500	7922,0778
MD5	468,750	359,3750	515,6250	578,1250	343,7588
SHA1	1140,625	921,8750	1406,2500	1562,5000	937,5240
SHA256	4093,750	3453,1250	5875,0000	5734,3750	1984,4258
SHA512	9343,750	8796,8750	13453,1250	14671,8750	3031,3276

Tabel 4.2.1.2 Rezultate timpi de computaţie algoritmi pe fişier 100 MB.

4.2.2. Fişiere de dimensiuni mari

În tabelul 4.2.2.1, sunt prezentate rezultatele testelor efectuate pe fişierul de 1 GB. Rezultatele sunt exprimate în secunde.

Algoritm	PC1	PC2	PC3	PC4	PC5
HmacSHA1	5,484375	14,781250	23,109375	40,234375	15,953533
HMACSHA256	42,031250	35,281250	61,109375	65,031250	19,500499
HMACSHA512	94,156250	90,359375	137,859375	165,000000	31,844565
HMACMD5	4,828125	3,531250	22,921875	37,812500	3,578217
AES	47,812500	23,734375	132,171875	87,000000	162,348235
Des	50,140625	27,250000	117,015625	90,785450	211,786672
3DES	76,765625	61,578125	107,125000	118,838772	246,537561
MD5	4,812500	3,500000	22,859375	39,737935	3,562591
SHA1	11,828125	9,437500	22,937500	39,097253	9,218986
SHA256	41,968750	35,375000	60,500000	63,439609	19,875509
SHA512	94,328125	89,906250	137,671875	153,364283	31,172673

Tabel 4.2.2.1 Rezultate timpi de computaţie algoritmi pe fişier 1 GB.

În tabelul 4.2.2.2 sunt prezentate rezultatele testelor efectuate pe fişierul de 2 GB. Rezultatele sunt exprimate în secunde.

Algoritm	PC1	PC2	PC3	PC4	PC5
HmacSHA1	42,046875	20,968750	47,078125	75,173300	195,583132
HMACSHA256	82,218750	70,890625	120,031250	125,439106	50,813801
HMACSHA512	184,125000	179,750000	275,312500	313,438778	62,392222
HMACMD5	9,328125	7,062500	47,078125	76,187500	7,312687
AES	110,609375	72,687500	210,562500	190,859375	368,257212
Des	124,421875	139,593750	189,843750	185,203125	545,607717
3DES	209,687500	136,859375	195,109375	227,843750	432,464196
MD5	36,812500	13,375000	47,000000	74,671875	16,469172
SHA1	22,703125	18,890625	84,078125	73,548206	18,594226
SHA256	82,062500	70,921875	120,359375	131,462301	38,469735
SHA512	184,406250	180,109375	275,171875	310,252923	61,079689

Tabel 4.2.2.2 Rezultate timpi de computaţie algoritmi pe fişier 2GB.

În tabelul 4.2.2.3 sunt prezentate rezultatele testelor efectuate pe fişierul de 3 GB. Rezultatele sunt exprimate în secunde.

Algoritm	PC1	PC2	PC3	PC4	PC5
HmacSHA1	58,250000	36,781250	85,218750	106,656933	86,517840
HMACSHA256	125,375000	106,031250	194,140625	179,563649	271,491325
HMACSHA512	278,937500	270,578125	415,625000	455,265037	193,442452
HMACMD5	58,203125	37,375000	72,750000	114,438232	50,220036
AES	207,140625	155,421875	316,609375	258,379432	652,364233
Des	217,671875	247,125000	305,046875	265,102992	639,797628
3DES	297,843750	262,562500	333,046875	349,612811	647,672830
MD5	58,281250	36,968750	72,703125	105,125673	30,313276
SHA1	113,359375	107,062500	119,203125	108,281943	120,753091
SHA256	125,156250	107,296875	180,187500	181,969659	63,611003
SHA512	277,093750	270,984375	414,515625	452,343750	96,658724

Tabel 4.2.2.3 Rezultate timpi de computaţie algoritmi pe fişier 3GB.

În tabelul 4.2.2.4 sunt prezentate rezultatele testelor efectuate pe fişierul de 4 GB. Rezultatele sunt exprimate în secunde.

Algoritm	PC1	PC2	PC3	PC4	PC5
HmacSHA1	81,656250	41,796875	70,625000	142,281250	124,378184
HMACSHA256	167,218750	142,437500	244,421875	242,490371	107,565254
HMACSHA512	370,890625	361,906250	549,968750	599,453690	194,645608
HMACMD5	80,421875	41,421875	70,968750	141,391530	84,502163
AES	270,093750	196,703125	326,406250	354,064259	722,364235
Des	281,281250	334,171875	294,765625	350,906250	1271,673179
3DES	399,718750	295,359375	466,656250	579,828125	749,284806
MD5	81,546875	41,265625	70,609375	142,015625	37,860344
SHA1	130,359375	141,531250	177,593750	143,546875	40,110402
SHA256	166,656250	219,406250	241,171875	246,171875	79,783292
SHA512	375,828125	360,921875	572,750000	616,125000	125,925099

Tabel 4.2.2.4 Rezultate timpi de computaţie algoritmi pe fişier 4GB.

În tabelul 4.2.2.5 sunt prezentate rezultatele testelor efectuate pe fişierul de 5 GB. Rezultatele sunt exprimate în secunde.

Algoritm	PC1	PC2	PC3	PC4	PC5
HmacSHA1	108,843750	54,343750	138,859375	190,328125	46,860575
HMACSHA256	218,703125	178,765625	305,203125	398,500000	168,129304
HMACSHA512	468,453125	450,687500	783,406250	757,671875	176,895153
HMACMD5	108,187500	53,937500	247,687500	178,328125	81,548963
AES	327,468750	255,609375	550,156250	436,750000	992,347235
Des	344,812500	418,406250	555,937500	442,984375	948,289901
3DES	528,328125	373,218750	671,187500	596,640625	1145,998087
MD5	108,406250	53,625000	138,593750	200,156250	50,423166
SHA1	108,562500	174,703125	207,046875	184,765625	180,879630
SHA256	219,000000	177,859375	301,187500	305,062500	105,830834
SHA512	473,812500	492,906250	770,687500	753,921875	168,348060

Tabel 4.2.2.5 Rezultate timpi de computaţie algoritmi pe fişier 5GB.

În tabelul 4.2.2.6 sunt prezentate rezultatele testelor efectuate pe fişierul de 6GB. Rezultatele sunt exprimate în secunde.

Algoritm	PC1	PC2	PC3	PC4	PC5
HmacSHA1	779,500000	64,781250	241,787813	227,203125	40,547913
HMACSHA256	372,078125	215,984375	366,514688	363,125000	127,659518
HMACSHA512	629,328125	567,734375	863,271900	922,750000	186,567276
HMACMD5	192,750000	83,484375	117,060938	226,687500	22,953713
AES	435,625000	679,140625	1066,744688	577,578125	989,326324
Des	447,812500	602,109375	545,365313	523,450282	1563,368146
3DES	666,265625	439,968750	503,975625	687,685963	1127,810121
MD5	128,234375	72,437500	118,543125	215,859721	23,469351
SHA1	309,250000	219,328125	133,731563	216,125000	53,438868
SHA256	265,718750	331,312500	492,644063	367,366562	116,924868
SHA512	602,500000	809,843750	935,993438	913,350371	184,379720

Tabel 4.2.2.6 Rezultate timpi de computaţie algoritmi pe fişier 6GB.

În tabelul 4.2.2.7 sunt prezentate rezultatele testelor efectuate pe fişierul de 7GB. Rezultatele sunt exprimate în secunde.

Algoritm	PC1	PC2	PC3	PC4	PC5
HmacSHA1	179,671875	75,637500	1052,799375	248,984375	66,079817
HMACSHA256	329,812500	274,156250	433,707188	424,368153	137,222263
HMACSHA512	666,828125	723,687500	998,229375	1057,859375	196,128213
HMACMD5	177,156250	72,343500	156,474375	285,234375	75,486307
AES	527,890625	732,487500	1256,926875	761,883284	1333,455325
Des	458,484375	649,140675	589,799063	707,972194	1330,471559
3DES	737,046875	521,390625	775,646250	834,156250	2008,457665
MD5	181,515625	72,265625	148,075313	258,828125	63,126616
SHA1	177,296875	249,328125	150,577500	268,812500	239,084245
SHA256	323,250000	247,968750	441,596250	429,875000	137,144136
SHA512	663,937500	651,968750	1174,848750	1222,781250	216,739923

Tabel 4.2.2.7 Rezultate timpi de computaţie algoritmi pe fişier 7GB.

În tabelul 4.2.2.8 sunt prezentate rezultatele testelor efectuate pe fişierul de 8 GB. Rezultatele sunt exprimate în secunde.

Algoritm	PC1	PC2	PC3	PC4	PC5
HmacSHA1	166,875000	104,640625	1263,359250	348,765625	76,571826
HMACSHA256	333,312500	286,968750	520,448625	533,257198	157,222263
HMACSHA512	845,859375	723,031250	1197,875250	1347,572821	206,128213
HMACMD5	165,609375	106,281250	187,769250	316,728788	81,486307
AES	515,671875	589,921875	1508,312250	724,467125	1758,732235
Des	529,484375	686,593750	707,758875	743,338284	1530,471559
3DES	844,281250	634,484375	930,775500	970,125000	2178,457665
MD5	164,453125	105,625000	177,690375	289,444480	73,126616
SHA1	252,078125	299,312500	180,693000	298,470172	289,084245
SHA256	362,734375	290,437500	529,915500	483,265625	177,144136
SHA512	760,234375	727,765625	1409,818500	1239,562500	256,739923

Tabel 4.2.2.8 Rezultate timpi de computaţie algoritmi pe fişier 8GB.

În tabelul 4.2.2.9 sunt prezentate rezultatele testelor efectuate pe fişierul de 9 GB. Rezultatele sunt exprimate în secunde.

Algoritm	PC1	PC2	PC3	PC4	PC5
HmacSHA1	188,546875	273,750000	1368,639188	327,000000	81,571826
HMACSHA256	412,406250	399,437500	563,819344	553,046529	187,222263
HMACSHA512	859,734375	861,031250	1297,698188	1387,543387	226,128213
HMACMD5	187,140625	130,875000	203,416688	317,593750	85,486307
AES	598,125000	789,625000	1634,004938	784,375000	1965,263235
Des	610,390625	780,390625	766,738781	784,156250	1690,471559
3DES	968,468750	712,812500	1008,340125	1052,629155	2190,457665
MD5	184,312500	240,484375	192,497906	347,984375	93,126616
SHA1	585,296875	389,671875	195,750750	338,828125	359,084245
SHA256	427,625000	326,734375	574,075125	643,343750	267,144136
SHA512	937,000000	871,515625	1527,303375	1525,958911	305,727125

Tabel 4.2.2.9 Rezultate timpi de computaţie algoritmi pe fişier 9GB.

În tabelul 4.2.2.10 sunt prezentate rezultatele testelor efectuate pe fişierul de 10 GB. Rezultatele sunt exprimate în secunde.

Algoritm	PC1	PC2	PC3	PC4	PC5
HmacSHA1	261,937000	111,000000	1484,447119	484,187712	92,955505
HMACSHA256	919,015000	366,953125	611,527134	723,852847	229,474624
HMACSHA512	978,671875	922,359375	1407,503419	1540,592202	323,805164
HMACMD5	259,093750	112,546875	220,628869	446,561187	105,987088
AES	645,031250	812,812500	1772,266894	1007,636335	2102,356123
Des	662,203125	846,250000	831,616678	994,367242	1940,119978
3DES	1041,296875	766,281250	1093,661213	1511,687500	2259,698472
MD5	257,468750	112,843750	208,786191	440,974394	129,143931
SHA1	255,578125	375,390625	212,314275	439,519331	403,041568
SHA256	529,875000	404,984375	622,650713	740,587604	451,761565
SHA512	954,562500	958,000000	1656,536738	1579,214136	320,070694

Tabel 4.2.2.10 Rezultate timpi de computaţie algoritmi pe fişier 10GB.

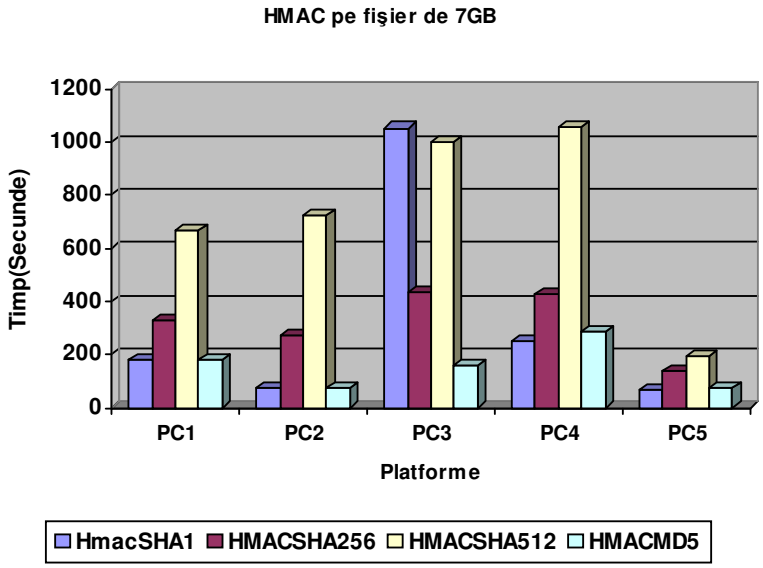

Figura 4.2.2.1 Algoritmi HMAC aplicaţi pe fişier de 7 GB

În figura 4.2.2.1, se poate observa că pentru algoritmii HMAC platforma PC5 are, per ansamblu, cele mai bune performanţe. În medie, HMACSHA512 are timpul de computaţie cel mai mare în comparaţie cu ceilalţi algoritmi.

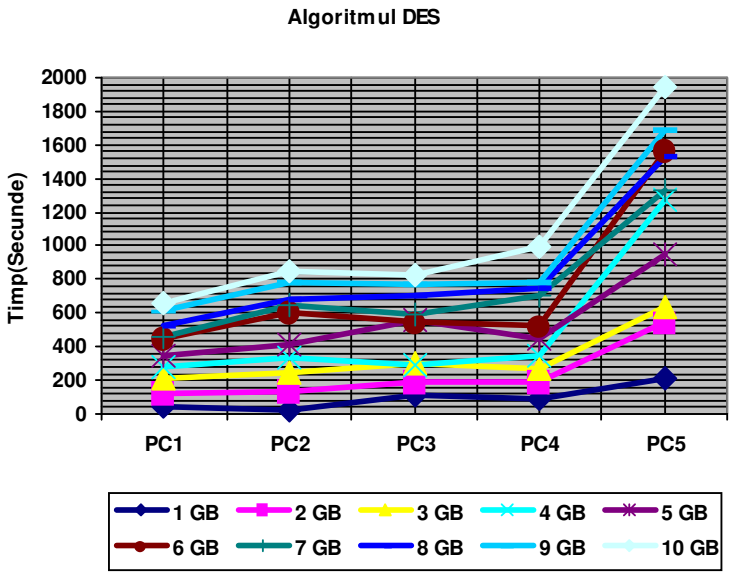

Figura 4.2.2.2 Algoritm DES

În figura 4.2.2.2, se poate observa că pentru algoritmul DES platforma PC1 are per ansamblu cele mai bune performanţe. Un aspect surprinzător este acela că platforma PC3, platforma PC4 şi platforma PC2 au rezultate destul de apropiate grafic. Ca şi în cazul figurii 4.1.2.4, platforma PC5 are cele mai slabe rezultate.

4.3. Rezultate OpenSSL LINUX

4.3.1. Fişiere de 1 MB şi 100 MB

În tabelul 4.3.1.1 sunt prezentate rezultatele testelor efectuate pe fişierul de 1 MB. Rezultatele sunt exprimate în milisecunde.

Algoritm	PC1	PC2	PC3	PC4	PC5
HmacSHA1	0.028	0.01	0.013	0.018	N/A
HMACSHA256	0.046	0.027	0.039	0.174	N/A
HMACSHA512	0.031	0.897	0.033	2.033	N/A
HMACMD5	0.008	0.009	0.016	0.014	N/A
AES	0.647	0.259	1.242	0.268	N/A
Des	1.199	0.087	1.242	0.317	N/A
3DES	0.458	0.171	7.492	2.351	N/A
MD5	0.713	0.009	0.012	0.929	N/A
SHA1	0.024	0.011	0.017	0.367	N/A
SHA256	0.064	0.026	0.04	0.126	N/A
SHA512	0.314	0.019	0.205	0.027	N/A

Tabel 4.3.1.1 Rezultate timpi de computaţie algoritmi pe fişier 1 MB.

În tabelul 4.3.1.2 sunt prezentate rezultatele testelor efectuate pe fişierul de 100 MB. Rezultatele sunt exprimate în secunde.

Algoritm	PC1	PC2	PC3	PC4	PC5
HmacSHA1	1.084	0.379	11.804	0.502	N/A
HMACSHA256	1.784	1.596	13.231	2.571	N/A
HMACSHA512	0.964	0.881	10.625	1.424	N/A
HMACMD5	0.425	1.151	13.132	3.861	N/A
AES	4.517	5.835	18.92	4.746	N/A
Des	13.5	7.478	21.57	19.037	N/A
3DES	15.751	10.113	25.654	24.815	N/A
MD5	0.396	0.307	14.054	0.502	N/A
SHA1	0.456	0.397	13.212	3.828	N/A
SHA256	1.864	1.535	13.368	2.62	N/A
SHA512	2.64	0.871	11.184	1.62	N/A

Tabel 4.31.2 Rezultate timpi de computaţie algoritmi pe fişier 100 MB.

4.3.2. Fişiere de dimensiuni mari

În tabelul 4.3.2.1 sunt prezentate rezultatele testelor efectuate pe fişierul de 1 GB. Rezultatele sunt exprimate în secunde.

Algoritm	PC1	PC2	PC3	PC4	PC5
HmacSHA1	3.767	3.436	94.154	42.706	N/A
HMACSHA256	17.483	17.777	84.009	47.522	N/A
HMACSHA512	10.056	14.240	93.238	41.962	N/A
HMACMD5	12.148	2.719	90.043	42.829	N/A
AES	36.015	64.344	189.608	137.433	N/A
Des	140.421	75.934	196.320	164.585	N/A
3DES	182.505	104.821	218.818	219.446	N/A
MD5	22.902	2.668	107.795	41.421	N/A
SHA1	22.759	3.452	149.282	43.179	N/A
SHA256	18.875	3.452	96.815	42.349	N/A
SHA512	9.523	8.559	81.911	44.545	N/A

Tabel 4.3.2.1 Rezultate timpi de computaţie algoritmi pe fişier 1 GB.

În tabelul 4.3.2.2 sunt prezentate rezultatele testelor efectuate pe fişierul de 2 GB. Rezultatele sunt exprimate în secunde.

Algoritm	PC1	PC2	PC3	PC4	PC5
HmacSHA1	52.991	6.735	169.826	79.213	N/A
HMACSHA256	60.215	31.089	174.916	78.984	N/A
HMACSHA512	56.869	17.076	169.456	79.672	N/A
HMACMD5	56.693	22.997	169.826	79.459	N/A
AES	265.514	128.653	404.436	275.900	N/A
Des	304.119	0.897	401.446	321.717	N/A
3DES	343.772	215.027	484.715	404.692	N/A
MD5	53.062	5.460	157.104	83.252	N/A
SHA1	53.418	6.679	169.695	79.144	N/A
SHA256	57.070	31.297	184.268	81.938	N/A
SHA512	56.794	17.414	166.065	78.657	N/A

Tabel 4.3.2.2 Rezultate timpi de computaţie algoritmi pe fişier 2GB.

În tabelul 4.3.2.3 sunt prezentate rezultatele testelor efectuate pe fişierul de 3 GB. Rezultatele sunt exprimate în secunde.

Algoritm	PC1	PC2	PC3	PC4	PC5
HmacSHA1	70.154	37.337	232.293	115.993	N/A
HMACSHA256	70.596	51.340	239.246	114.081	N/A
HMACSHA512	70.227	39.299	238.078	115.491	N/A
HMACMD5	70.131	36.106	285.566	115.735	N/A
AES	386.048	194.588	579.660	428.471	N/A
Des	470.842	231.953	579.520	496.619	N/A
3DES	527.083	331.593	687.445	612.853	N/A
MD5	66.440	35.295	240.253	151.512	N/A
SHA1	67.462	39.785	262.913	117.365	N/A
SHA256	70.868	49.364	254.845	124.155	N/A
SHA512	66.398	36.179	225.594	116.274	N/A

Tabel 4.3.2.3 Rezultate timpi de computaţie algoritmi pe fişier 3GB.

În tabelul 4.3.2.4 sunt prezentate rezultatele testelor efectuate pe fişierul de 4 GB. Rezultatele sunt exprimate în secunde.

Algoritm	PC1	PC2	PC3	PC4	PC5
HmacSHA1	90.0090	46.2280	299.9950	156.3010	N/A
HMACSHA256	90.1430	67.0530	335.7250	159.0140	N/A
HMACSHA512	89.8870	48.4140	304.9800	157.0730	N/A
HMACMD5	89.6990	47.1100	287.1030	158.0610	N/A
AES	594.0260	265.1460	691.3220	588.1540	N/A
Des	613.7560	330.2770	746.4800	706.9310	N/A
3DES	696.2980	430.2430	938.3530	833.9850	N/A
MD5	86.5790	46.2450	284.3160	156.5460	N/A
SHA1	90.2740	48.5340	291.6430	153.9580	N/A
SHA256	90.4670	71.6030	314.9900	163.9280	N/A
SHA512	90.2820	47.4200	284.6970	162.4810	N/A

Tabel 4.3.2.4 Rezultate timpi de computaţie algoritmi pe fişier 4GB.

În tabelul 4.3.2.5 sunt prezentate rezultatele testelor efectuate pe fişierul de 5 GB. Rezultatele sunt exprimate în secunde.

Algoritm	PC1	PC2	PC3	PC4	PC5
HmacSHA1	115.1170	57.4990	380.3470	193.8000	N/A
HMACSHA256	115.2960	94.9420	407.4430	188.4780	N/A
HMACSHA512	114.9000	59.6420	406.9160	195.4210	N/A
HMACMD5	115.0550	57.3120	383.8270	190.9800	N/A
AES	668.5820	326.5650	905.8590	747.2100	N/A
Des	762.1270	393.9320	924.2630	927.1940	N/A
3DES	874.6590	521.3430	1135.6100	1016.3970	N/A
MD5	115.6800	57.1110	482.9330	193.1480	N/A
SHA1	115.4770	57.0800	426.0270	192.3180	N/A
SHA256	115.3910	86.4450	413.9330	197.3270	N/A
SHA512	114.7810	65.4270	388.2110	192.8410	N/A

Tabel 4.3.2.5 Rezultate timpi de computaţie algoritmi pe fişier 5GB.

În tabelul 4.3.2.6 sunt prezentate rezultatele testelor efectuate pe fişierul de 6GB. Rezultatele sunt exprimate în secunde.

Algoritm	PC1	PC2	PC3	PC4	PC5
HmacSHA1	216.7100	68.7760	389.6690	231.0530	N/A
HMACSHA256	216.7220	105.3700	427.3060	228.2290	N/A
HMACSHA512	216.6480	71.7710	398.6970	232.5840	N/A
HMACMD5	213.5730	68.9520	395.6200	229.8710	N/A
AES	840.7550	408.6450	1168.7670	841.5570	N/A
Des	931.3810	477.5250	1246.1320	1023.1100	N/A
3DES	1107.7000	650.3560	1439.0650	1239.0250	N/A
MD5	216.7280	69.8540	398.8430	233.0190	N/A
SHA1	217.0160	69.9560	388.3910	230.3060	N/A
SHA256	217.4620	103.7800	441.3640	231.4100	N/A
SHA512	216.9850	74.5260	439.2530	230.1490	N/A

Tabel 4.3.2.6 Rezultate timpi de computaţie algoritmi pe fişier 6GB.

În tabelul 4.3.2.7 sunt prezentate rezultatele testelor efectuate pe fişierul de 7GB. Rezultatele sunt exprimate în secunde.

Algoritm	PC1	PC2	PC3	PC4	PC5
HmacSHA1	180.2260	79.5680	441.0390	267.7190	N/A
HMACSHA256	180.2900	115.8400	496.6880	268.2280	N/A
HMACSHA512	179.7660	82.9390	462.1810	266.1890	N/A
HMACMD5	179.9880	78.7250	448.6730	269.8380	N/A
AES	925.3660	507.8940	1215.6430	1004.7340	N/A
Des	1090.9140	557.4150	1476.7350	1235.9930	N/A
3DES	1212.4380	773.9300	1601.4910	1484.3210	N/A
MD5	180.0390	81.4110	487.2450	273.2340	N/A
SHA1	180.3990	81.6230	446.5340	270.5360	N/A
SHA256	180.2810	139.1620	501.5050	282.2290	N/A
SHA512	180.0980	83.3520	452.4630	268.6760	N/A

Tabel 4.3.2.7 Rezultate timpi de computaţie algoritmi pe fişier 7GB.

În tabelul 4.3.2.8 sunt prezentate rezultatele testelor efectuate pe fişierul de 8 GB. Rezultatele sunt exprimate în secunde.

Algoritm	PC1	PC2	PC3	PC4	PC5
HmacSHA1	209.554	107.226	503.375	308.144	N/A
HMACSHA256	209.327	139.653	576.961	310.482	N/A
HMACSHA512	209.380	108.472	537.697	308.897	N/A
HMACMD5	220.719	106.154	514.433	305.201	N/A
AES	1008.725	529.863	1352.276	1155.237	N/A
Des	1273.067	634.629	1705.935	1389.548	N/A
3DES	1354.206	883.986	1779.752	1658.521	N/A
MD5	209.692	108.219	561.367	307.040	N/A
SHA1	209.297	107.616	592.972	306.629	N/A
SHA256	210.083	134.784	567.237	318.916	N/A
SHA512	209.320	108.974	583.826	312.531	N/A

Tabel 4.3.2.8 Rezultate timpi de computaţie algoritmi pe fişier 8GB.

În tabelul 4.3.2.9 sunt prezentate rezultatele testelor efectuate pe fişierul de 9 GB. Rezultatele sunt exprimate în secunde.

Algoritm	PC1	PC2	PC3	PC4	PC5
HmacSHA1	282.386	118.696	716.723	346.727	N/A
HMACSHA256	284.775	165.299	766.326	357.782	N/A
HMACSHA512	282.915	123.229	659.278	344.511	N/A
HMACMD5	283.632	118.480	707.523	343.464	N/A
AES	1257.056	602.034	1774.236	1343.430	N/A
Des	1407.236	732.473	2271.327	1567.584	N/A
3DES	1624.185	1001.879	2519.233	1904.167	N/A
MD5	282.728	118.842	693.872	341.934	N/A
SHA1	280.137	118.903	742.536	342.977	N/A
SHA256	281.407	162.888	717.326	351.283	N/A
SHA512	283.142	122.347	776.245	346.044	N/A

Tabel 4.3.2.9 Rezultate timpi de computaţie algoritmi pe fişier 9GB.

În tabelul 4.3.2.10 sunt prezentate rezultatele testelor efectuate pe fişierul de 10 GB. Rezultatele sunt exprimate în secunde.

Algoritm	PC1	PC2	PC3	PC4	PC5
HmacSHA1	249.862	112.250	814.179	451.396	N/A
HMACSHA256	250.880	164.885	886.037	453.266	N/A
HMACSHA512	249.648	118.759	816.623	451.383	N/A
HMACMD5	245.768	113.033	850.962	449.754	N/A
AES	1316.646	656.470	2482.434	1636.518	N/A
Des	1632.087	806.494	3092.567	1793.093	N/A
3DES	1741.586	1068.296	3598.543	2211.078	N/A
MD5	249.243	113.779	779.679	448.974	N/A
SHA1	246.006	114.190	731.012	459.615	N/A
SHA256	251.897	187.683	836.033	446.992	N/A
SHA512	249.992	123.257	771.696	450.457	N/A

Tabel 4.3.2.10 Rezultate timpi de computaţie algoritmi pe fişier 10GB.

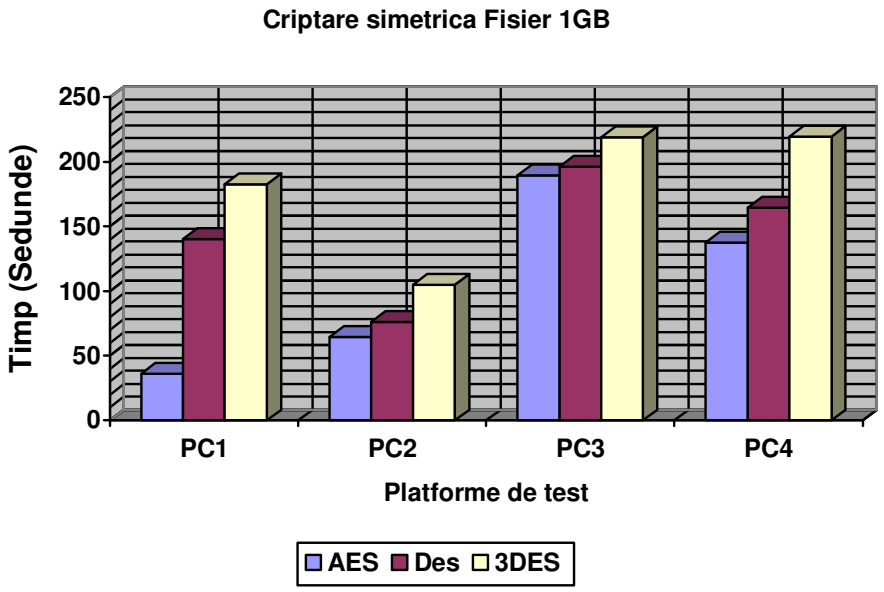

Figura 4.3.2.1 Criptare simetrica fişier de 1 GB

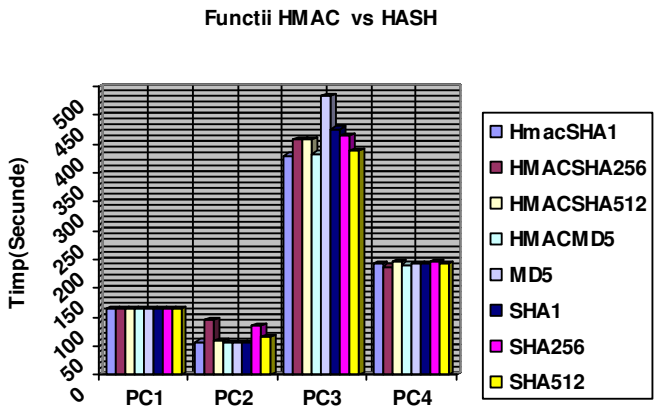

Figura 4.3.2.2 Funcţii HMAC vs. hash Fişier 5 GB

Din figurile 4.3.2.1 şi 4.3.2.2, se poate distinge foarte clar faptul că platforma PC3 este cea mai lentă atât la criptarea simetrică, cât şi în cazul funcţiilor HMAC şi hash. În figura 4.3.2.1, se poate observa că timpul necesar computaţiei funcţiilor criptografice simetrice creşte de la AES către 3DES şi acest aspect se repetă pe toate cele 4 platforme testate. Sintetizând cele prezentate în figura 4.3.2.2, se observă că procesorul Intel Pentium (PC3) are o viteză de procesare mult mai mică, valorile timpilor obţinuţi fiind mai mari decât în cazul procesoarelor cu două nuclee.

4.4. Comparaţie între rezultatele obţinute pe diferite platforme şi între limbajele de programare folosite

În subcapitolele anterioare s-au prezentat rezultatele obţinute pe fiecare limbaj de programare/ sistem de operare pentru platformele detaliate în capitolul 3.5. Vom începe prin a compara algoritmul **AES** AES.

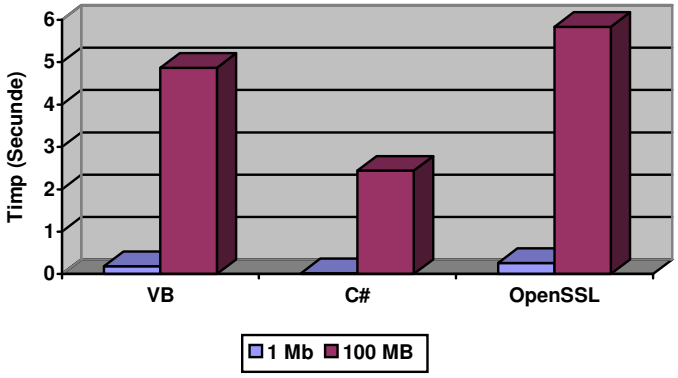

Figura 4.4.1 Platforma PC2 Fişiere 1 MB si 100 MB

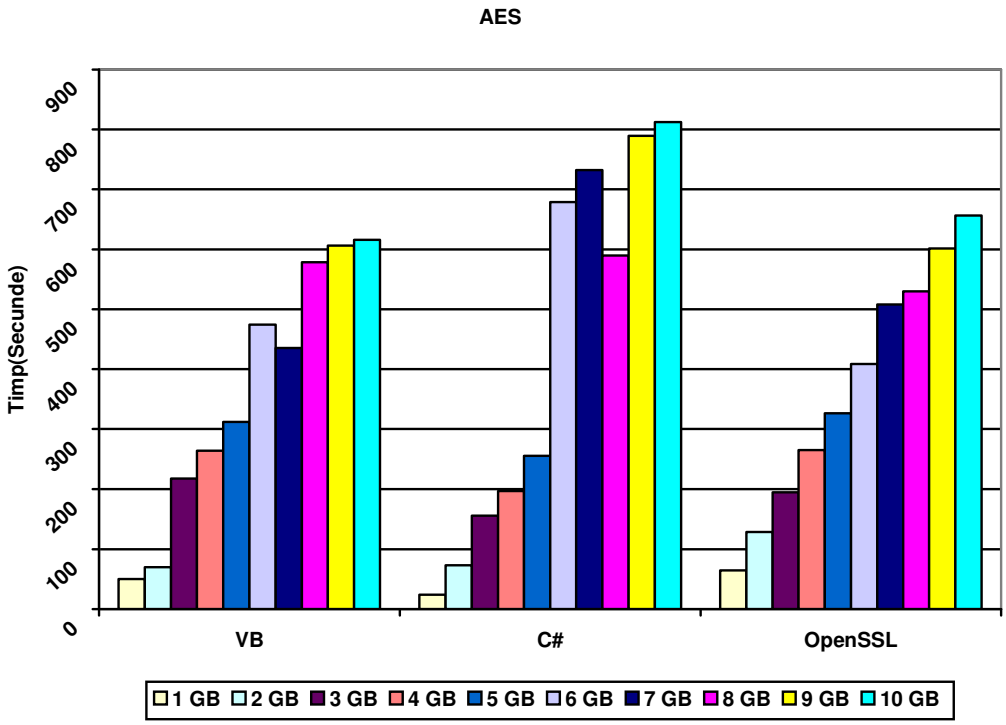

Figura 4.4.2 Platforma PC2. Fişiere de dimensiuni mari

După cum se poate observa la fişierele de mici dimensiuni, performanţă sporită au algoritmii în C#. La fişierele de dimensiuni mari până la dimensiunea de 5 GB C#, aceasta creşte în timp uniform şi are perioadă de computaţie mai mică decât VB sau OpenSSL. Începând de la fişierele de 6 GB, se observă o creştere mult mai mare decât la VB sau OpenSSL. Singura creştere uniformă pe toate cele zece fişiere de mari dimensiuni o are OpenSSL, dar timpii obţinuţi nu sunt mai buni decât cei scoşi de VB respectiv C#.

Din rezultatele obţinute se observă că platforma PC2 a obţinut cei mai mici timpi de computaţie, în medie, pe toţi algoritmii testaţi şi pentru toate fişierele de test de mari dimensiuni. În figura 4.4.3, este exemplificată perioada de computaţie pentru funcţia hash SHA512 aplicată asupra fişierului de test de 7 GB pe cele 4 platforme în OpenSSL.

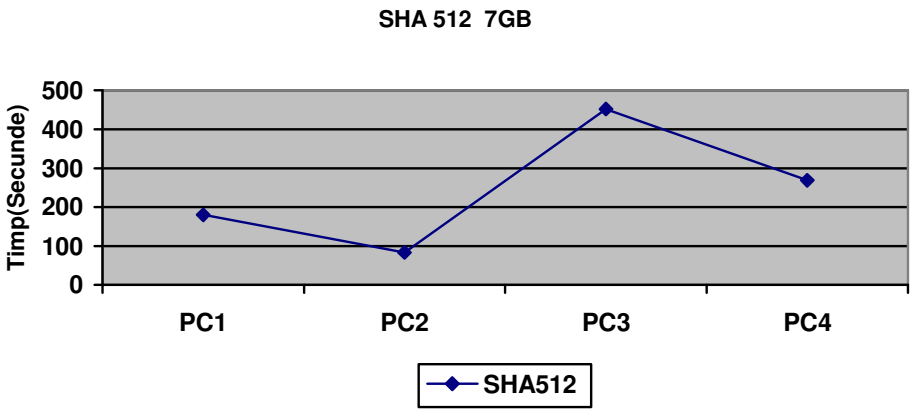

Figura 4.4.3 Funcţia hash SHA512 Fişier 7 GB

În figura 4.4.4 se prezintă timpii de computaţie pentru HMAC SHA256, când se aplică asupra unui fişier de 4 GB pe platforma PC1. Se poate observa că OpenSSL are timp apropiat de VB, pe când C# are nevoie de aproape dublul lui VB sau OpenSSL pentru a rula task-ul solicitat. În figura 4.4.5 sunt afişate grafic performanţele funcţiei HMAC MD5 când se testează pe un fişier de 5 GB pe platforma PC4. Se poate observa faptul că, în acest caz, timpii necesari au valori apropiate.

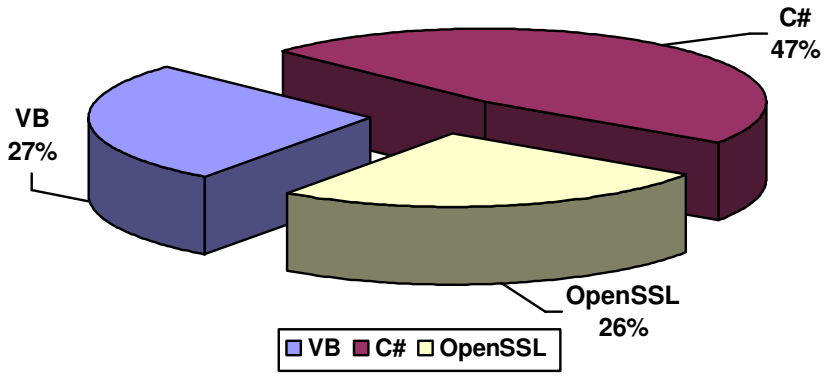

Figura 4.4.4 HMAC SHA256 Fişier 4GB

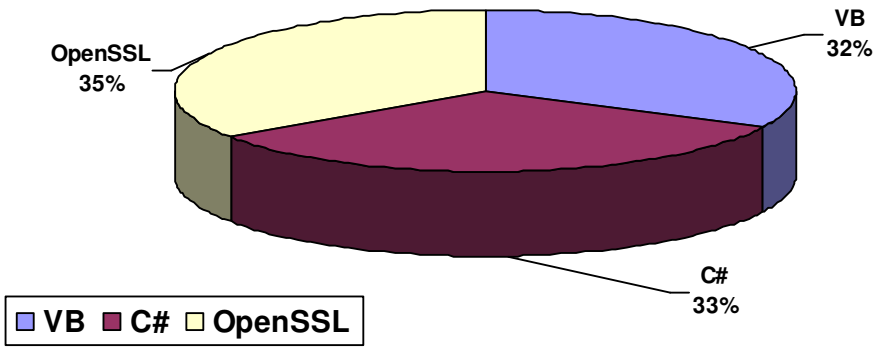

Figura 4.4.5 HMAC MD5 Fişier 5GB

În figura 4.4.6 se compară timpii de computaţie ai funcţiilor hash pe platforma PC1, sub mediul OpenSSL, pentru fişierul de intrare de 8 GB. Primitivele SHA1 şi SHA512 au timpii cei mai mici, pe când MD5 îl are mai mare, iar SHA256 durează cel mai mult. Timpii sunt apropiaţi ca valori, după cum se poate observa şi din figura 4.4.6. În figura 4.4.7, se face o comparaţie similară, pentru platforma PC4. Deşi procesoarele sunt cu două nuclee, ele fiind din clase diferite, se observă o creştere a timpului de computaţie în testele efectuate pe platforma PC4. Din cele două grafice se poate concluziona că funcţiile hash sunt influenţate de cantitatea de memorie, dat fiind faptul că PC4 are doar 1 GB RAM memorie, pe când PC1 are 3 GB RAM. Surprinzător este faptul că SHA256 are timp mai mare decât SHA512.

În figurile 4.4.9 şi 4.4.8 sunt prezentate graficele omoloage pentru limbajul C#. În această situaţie se observă că SHA512 este cel mai lent algoritm din cei patru testaţi, iar timpii de computaţie sunt mai mari în cazul platformei PC4 decât în cazul platformei PC1.

În figurile 4.4.10 şi 4.4.11 sunt prezentate graficele omoloage pentru limbajul Visual Basic. Timpii scoşi de platforma PC1 folosind bibliotecile Visual Basic sunt cei mai mici din acest segment de teste. Creşterea timpilor respectă tendinţa de creştere din cazul C# în care MD5 are timpul de computaţie cel mai mic (indiferent de platformă), iar SHA512 are timpul de computaţie cel mai mare.

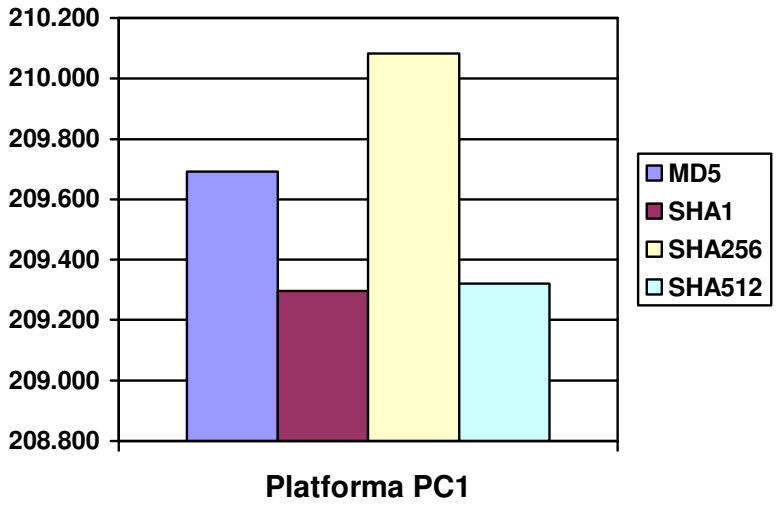

Figura 4.4.6 Comparare funcţii hash platformă PC1 fişier 8 GB, OpenSSL

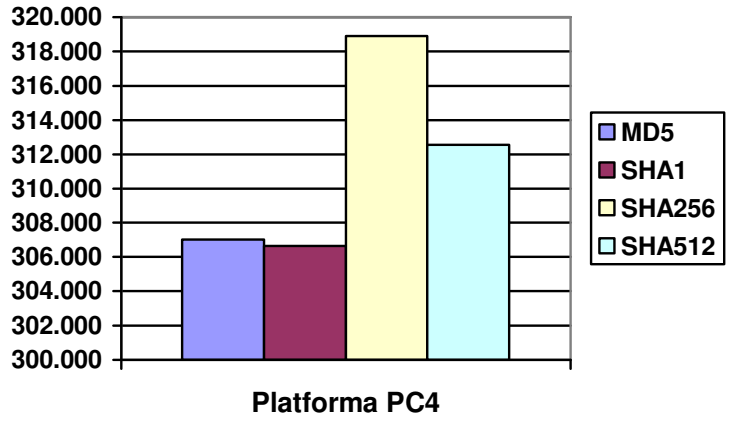

Figura 4.4.7 Comparare funcţii hash platformă PC4 fişier 8 GB, OpenSSL

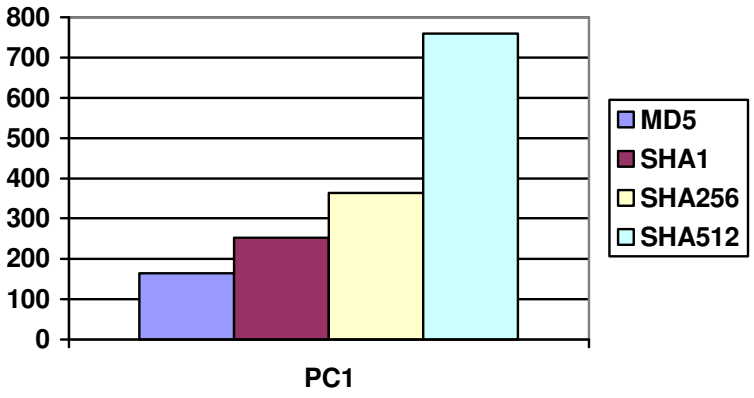

Figura 4.4.8 Comparare funcţii hash platformă PC1 fişier 8 GB, C#

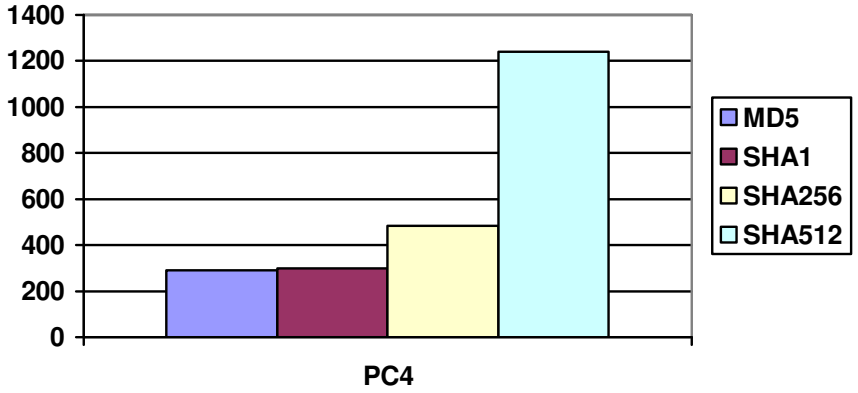

Figura 4.4.9 Comparare funcţii hash platformă PC4 fişier 8 GB, C#

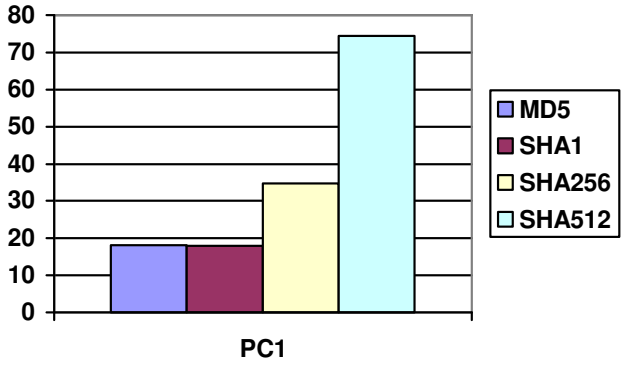

Figura 4.4.10 Comparare funcţii hash platformă PC1 fişier 8 GB, Visual Basic

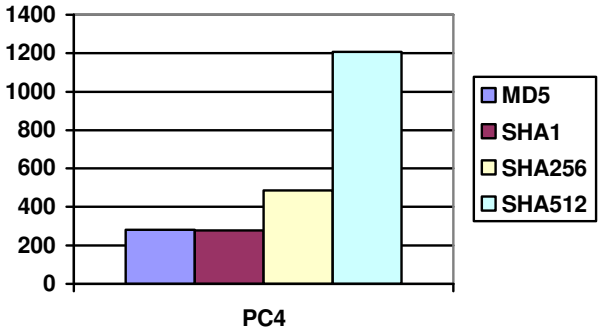

Figura 4.4.11 Comparare funcţii hash platformă PC4 fişier 8 GB, Visual Basic

În următoarea etapă se evaluează performanţa algoritmului DES pe cele trei medii. Fişierul ales a fost cel de 6 GB. Pe platforma PC2 se observă că sub Unix, DES obţine cel mai bun timp de computaţie. Visual Basic are o creştere de cca. 48 de secunde faţă de OpenSSL, iar C# are o creştere de cca. 125 de secunde faţă de OpenSSL şi de cca. 77 de secunde faţă de VB. În schimb, pe platforma PC3 sub Unix, DES obţine cel mai slab timp de computaţie. Cel mai bun timp este obţinut în Visual Basic urmând aproape şi C#.

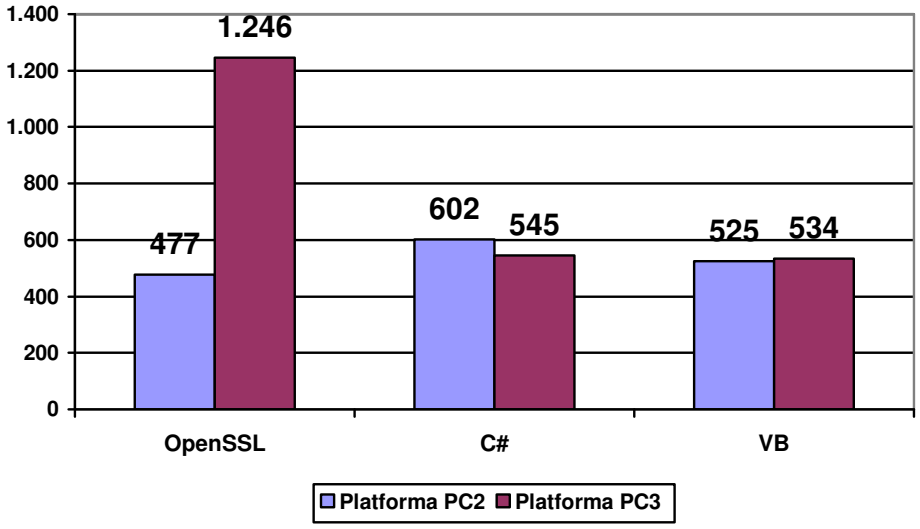

Figura 4.4.12 Comparare DES platformă PC2 fişier 6 GB

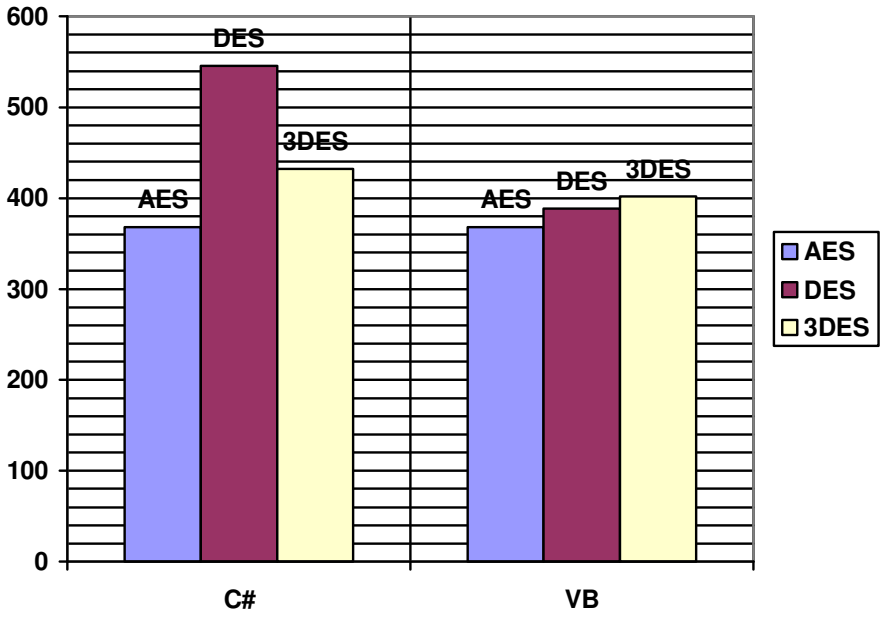

Figura 4.4.13 Comparare criptare simetrică, platformă PC5 fişier 2 GB

În figura 4.4.13 sunt comparaţi timpii de execuţie ai algoritmilor simetrici pe platforma PC5. Testele au fost efectuate în Visual Basic şi C#. Algoritmul AES necesită aproape aceiaşi timpi de computaţie. În cazul DES se observă o diferenţă de aproximativ 157 de secunde între C# şi Visual Basic. 3DES în C# are nevoie de aproximativ 31 de secunde, mai mult decât Visual Basic. În general, în cazul algoritmilor simetrici, pentru platforma PC5 timpii de computaţie sunt sensibil mai mari decât ai celorlalte platforme (figura 4.2.2.2) pe când, în cazul funcţiilor HMAC, timpii necesari rulării sunt cei mai mici comparativ cu celelalte platforme (figura 4.2.2.1). De aici putem deduce faptul că durata de citire/scriere a echipamentelor de stocare influenţează performanţa primitivelor criptografice.

În pasul următor se prezintă o comparaţie omoloagă, fişierul de test fiind cel de 10 GB (figura 4.4.14). Astfel, se observă că în cazul algoritmului AES, timpii de computaţie sunt aproximativ egali. O diferenţă apare în cazul DES, unde raportul s-a inversat faţă de situaţia precedentă şi acum C# necesită cu aproximativ 496 de secunde mai puţin decât Visual Basic. În cazul lui 3DES se respectă tendinţa din exemplul anterior şi C# necesită cu 133 mai multe secunde pentru criptarea fişierului decât Visual Basic.

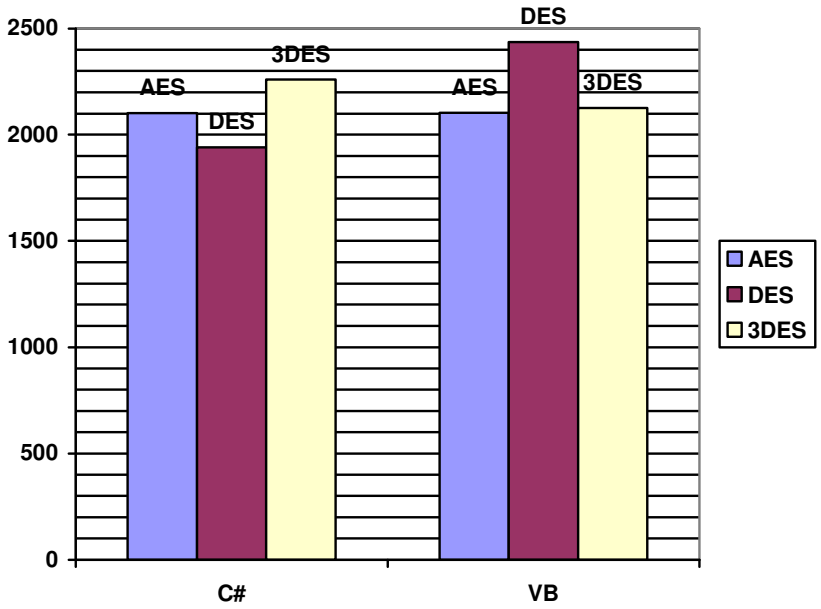

Figura 4.4.14 Comparare criptare simetrică, platformă PC5 fişier 10GB

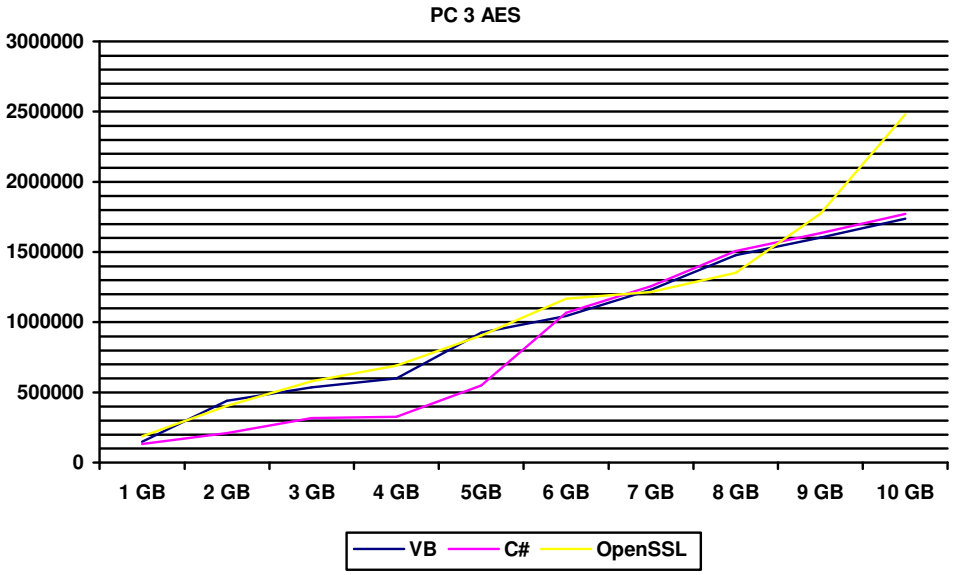

Figura 4.4.15 Platforma PC3. Creştere timpi funcţie volum de intrare. AES

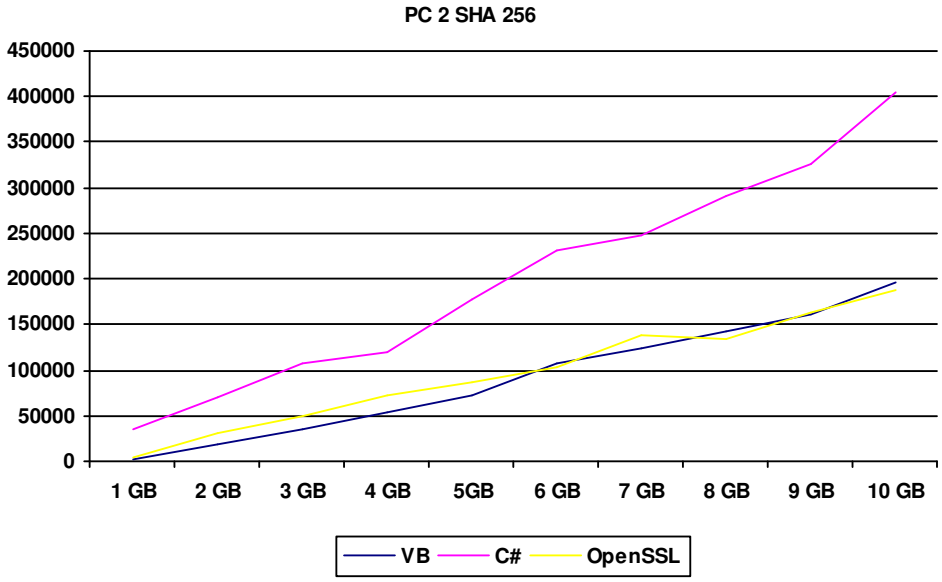

Figura 4.4.16 Platforma PC2. Creştere timpi funcţie volum de intrare. SHA256

4.5. Rezultate obţinute în urma testelor de criptare a datelor din memorie

În tabelul 4.5.1 sunt prezentate rezultatele testelor efectuate pe platforma 1. Rezultatele sunt exprimate în milisecunde.

Primitive	PC1		
	VB	C#	Java
HmacSHA1	0,0233125000	0,025453125000	0,0037500000
HMACSHA256	0,0192812500	0,017031250000	0,0055160000
HMACSHA512	0,0700625000	0,058281250000	0,0201720000
HMACMD5	0,0223437500	0,025468750000	0,0017500000
AES	0,1653437500	0,156812500000	0,1563240000
Des	0,2920000000	0,281796875000	0,2209423400
3DES	0,1688125000	0,156187500000	0,5678910000
MD5	0,0187656250	0,019562500000	0,0004680000
SHA1	0,0196718750	0,021093750000	0,0009370000
SHA256	0,0163593750	0,008250000000	0,0026560000
SHA512	0,0412187500	0,017937500000	0,0099070000

Tabel 4.5.1 Rezultate timpi de computaţie algoritmi Platforma 1.

În tabelul 4.5.2 sunt prezentate rezultatele testelor efectuate pe platforma 2. Rezultatele sunt exprimate în milisecunde.

Primitive	PC2		
	VB	C#	Java
HmacSHA1	0,2933360930	0,3601562500	0,0358440000
HMACSHA256	0,2163240760	0,2533437500	0,0543430000
HMACSHA512	0,5879347380	0,6456562500	0,2075780000
HMACMD5	0,4121734240	0,4249062500	0,0131250000
AES	0,2117968750	0,2117968750	0,4054388588
Des	1,2245685340	1,3301093750	0,5662048336
3DES	0,2284345270	0,2120781250	1,4693082017
MD5	0,1723813500	0,1663750000	0,0033130000
SHA1	0,1762031250	0,1762031250	0,0098440000
SHA256	0,2543234620	0,2879687500	0,0250160000
SHA512	1,1680093800	1,1402187500	0,0997810000

Tabel 4.5.2 Rezultate timpi de computaţie algoritmi Platforma 2.

În tabelul 4.5.3 sunt prezentate rezultatele testelor efectuate pe platforma 3. Rezultatele sunt exprimate în milisecunde.

Primitive	PC3		
	VB	C#	Java
HmacSHA1	0,0367187500	0,0394843750	0,0070000000
HMACSHA256	0,0235156250	0,0211875000	0,0105150000
HMACSHA512	0,0531093750	0,0456093750	0,0421570000
HMACMD5	0,0349218750	0,0400781250	0,0030470000
AES	0,0326562500	0,0318593750	0,2037381200
Des	0,1223750000	0,1216562500	0,2845250420
3DES	0,0324218750	0,0324218750	0,7383458300
MD5	0,0307500000	0,0299843750	0,0007970000
SHA1	0,0317968750	0,0314687500	0,0018590000
SHA256	0,0242812500	0,0115937500	0,0050940000
SHA512	0,0424531250	0,0163750000	0,0207500000

Tabel 4.5.3 Rezultate timpi de computaţie algoritmi Platforma 3.

În figura 4.5.1 sunt prezentaţi timpii de computaţie obţinuţi de algoritmul AES pe cele trei platforme sub Java. În figura 4.5.3 sunt prezentaţi timpii de computaţie obţinuţi de algoritmul AES pe cele trei platforme în Visual Basic. În figura 4.5.3 sunt prezentaţi timpii de computaţie obţinuţi de algoritmul AES pe cele trei platforme în C#.

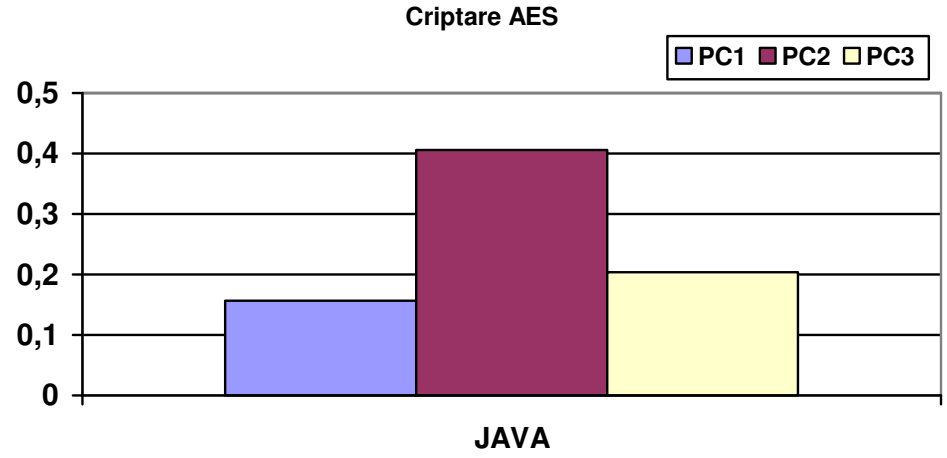

Figura 4.5.1 Comparare criptare simetrică AES. Java

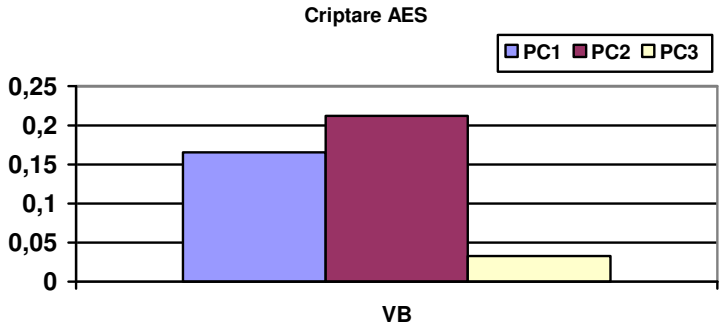

Figura 4.5.2 Comparare criptare simetrică AES.VB

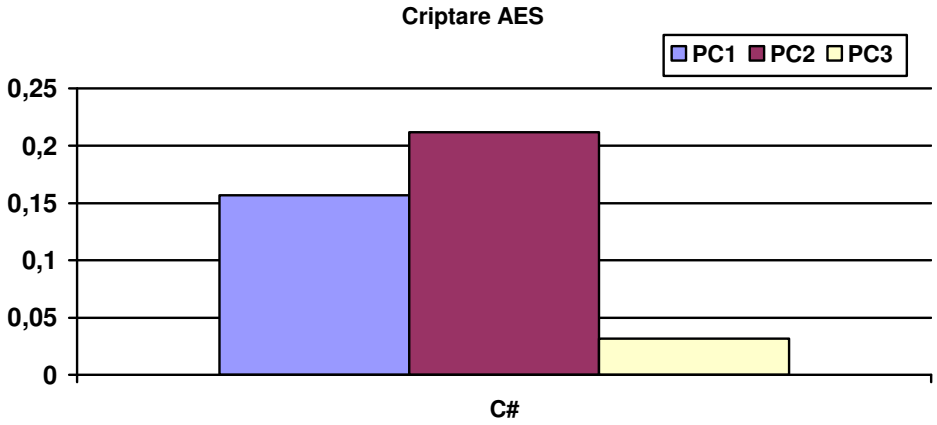

Figura 4.5.3 Comparare criptare simetrică AES.C#

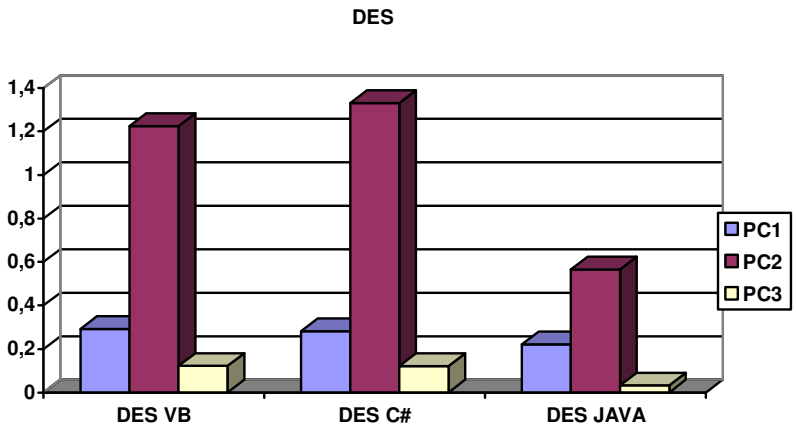

Figura 4.5.4 Comparare criptare simetrică DES

În [TOM10/4] s-a realizat o analiză a algoritmului AES atunci când primeşte la intrare şi volume mici de date, stocate în memorie. Comportamentul algoritmului AES din figurile 4.5.1, 4.5.2, 4.5.3 a fost discutat şi în această lucrare în care s-a concluzionat că Java necesită timpi de computaţie sensibili mai mari decât Visual Basic şi C#. Totodată pe toate cele trei medii de dezvoltare platforma PC2 obţine timpii cei mai slabi. În schimb, sub Java PC3 obţine performante mai slabe decât PC1, dar situaţia se inversează în cazul Visual Basic şi C# unde PC3 obţine performanţe mai bune decât PC1 [TOM10/4].

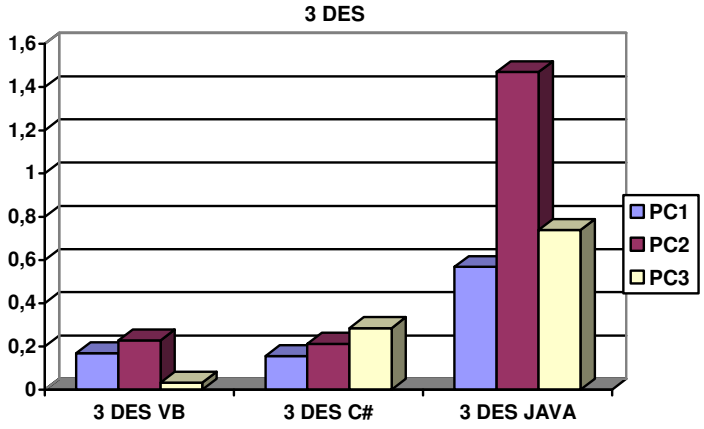

Figura 4.5.5 Comparare criptare simetrică 3DES

În figura 4.5.4, sunt prezentaţi timpii de computaţie obţinuţi de algoritmul DES pe cele trei platforme sub Visual Basic, C# şi Java, iar în figura 4.5.5 sunt prezentaţi timpii de computaţie obţinuţi de algoritmul 3DES pe cele trei platforme sub Visual Basic, C# şi Java.

Din toate figurile prezentate în această secţiune se observă că procesorul Atom al platformei 2 obţine rezultatele cele mai slabe (cu excepţia 3DES sub C#), făcându-l pe acesta o alegere proastă pentru folosirea intensă a aplicaţiilor criptografice în comparaţie cu celelalte platforme.

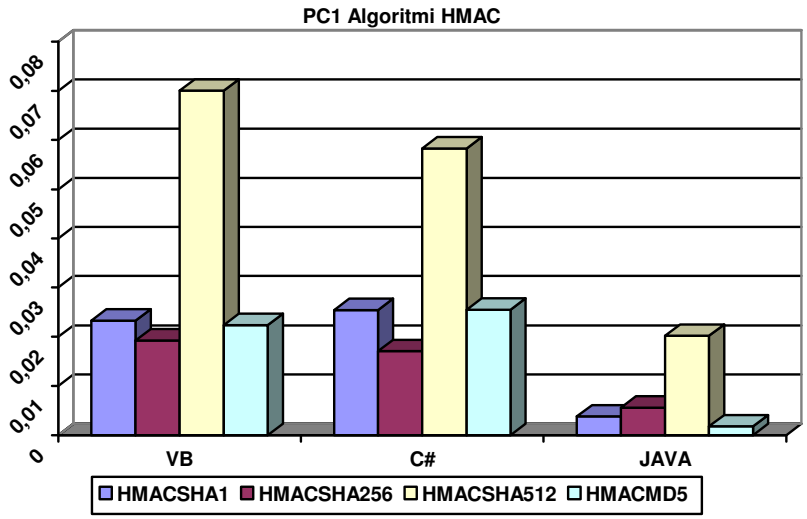

Figura 4.5.6 Algoritmi HMAC platforma PC1

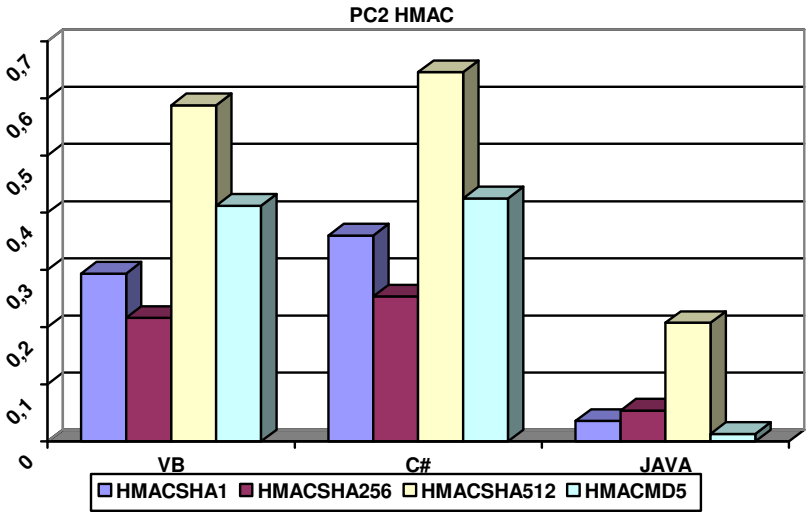

Figura 4.5.7 Algoritmi HMAC platforma PC2

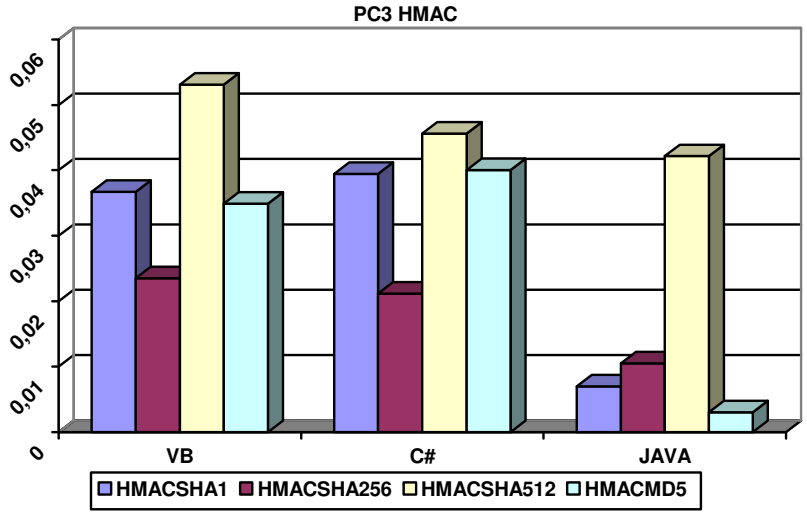

Figura 4.5.8 Algoritmi HMAC platforma PC3

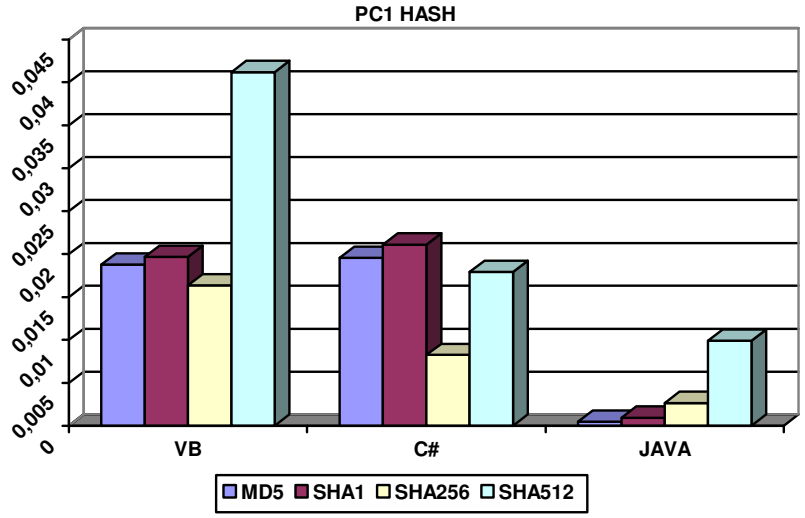

Figura 4.5.9 Algoritmi hash platforma PC1

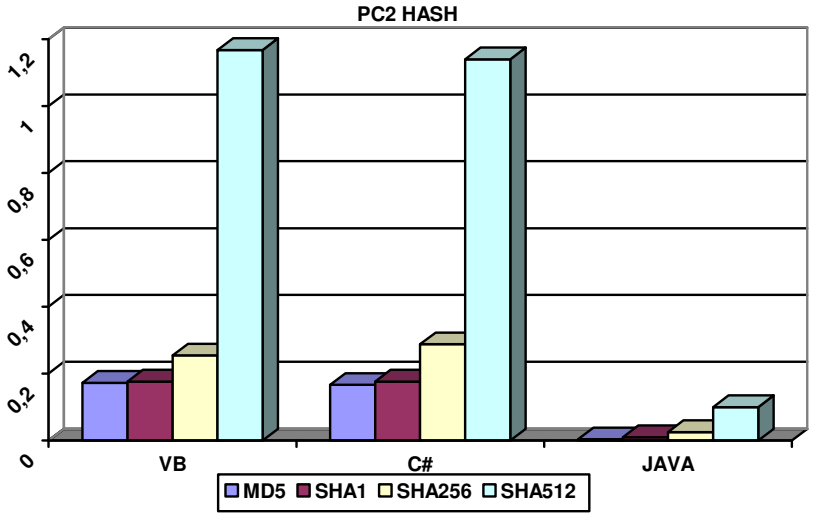

Figura 4.5.10 Algoritmi hash platforma PC2

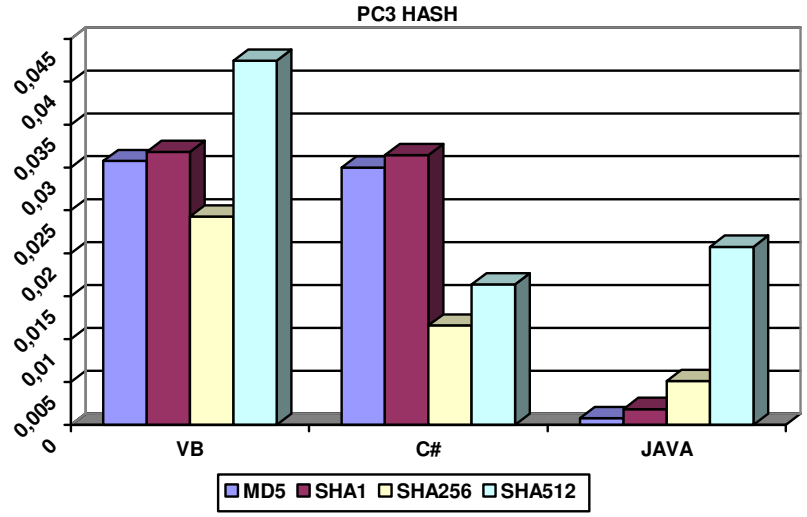

Figura 4.5.11 Algoritmi hash platforma PC3

Se poate observa că şi în cazul algoritmilor hash sau HMAC, platforma PC2 obţine timpi foarte mari de computaţie în comparaţie cu celelalte două.

În figura 4.5.12 se prezintă compararea algoritmului SHA256 sub Visual Basic, C# şi Java pe cele trei platforme. Algoritmul a fost ales la întâmplare.

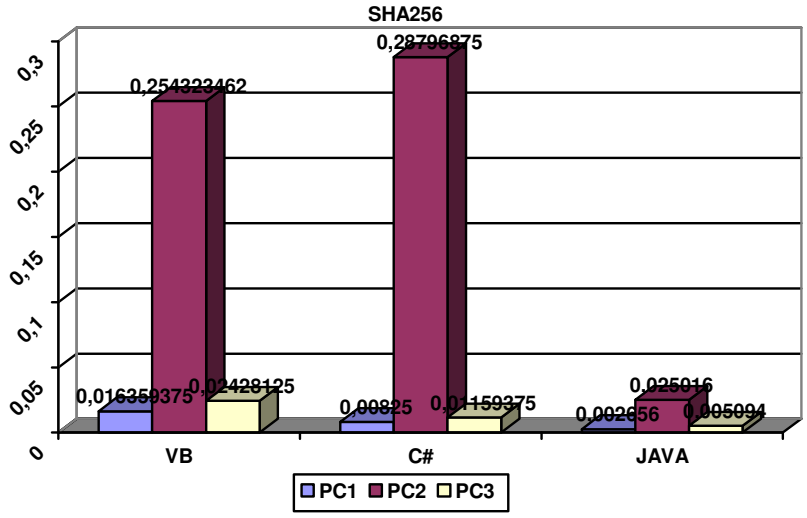

Figura 4.5.12 SHA256

4.6. Concluzii

În acest capitol sunt prezentate rezultatele obţinute în urma rulării aplicaţiilor de criptare descrise în capitolul 3.

S-a realizat o analiză critică a rezultatelor obţinute şi au fost prezentate imagini şi diagrame în care se compară rezultatele între ele.

Rularea testelor din acest capitol a fost necesară pentru a putea crea o bază de date cu performanţele primitivelor criptografice. Numărul de teste efectuate a fost mare şi rezultatele obţinute au ajutat la crearea unei imagini de ansamblu asupra timpilor necesari rulării algoritmilor criptografici. Numărul total de platforme pe care au fost rulate testele a fost de opt, fiind acoperite mai multe familii de procesoare şi mai mulţi producători de memorii. Numărul mare de platforme îşi are ca justificare feedback-ul bogat primit în urma testelor. Numărul mare de primitive criptografice alese are ca scop centralizarea performanţelor acestora. S-a încercat alegerea mai multor clase de primitive criptografice şi, din fiecare clasă, acele primitive care au reprezentat sau reprezintă standarde, sau care sunt folosite foarte des. Din cadrul sistemelor de operare s-au ales două familii mari: Windows şi Unix. Din cadrul fiecărei familii au fost folosite unul sau două sisteme de operare. Nu au fost alese alte sisteme de operare din cauza problemelor de portabilitate a codurilor sursă. Ca limbaje de programare au fost alese .NET (C# şi Visual Basic) şi Java acestea fiind cel mai des folosite, iar sub Linux s-au utilizat bibliotecile oferite de OpenSSL. Necesitatea unui volum mare de benchmark-uri a reieşit din analizele lucrărilor de acest gen în care testele efectuate acopereau o arie mică de variabile. Fie numărul de primitive criptografice testate era mic, fie platformele pe care s-au efectuat testele a fost într-un număr redus, mediile de dezvoltare folosite pentru scrierea aplicaţiilor erau insuficiente, sau numărul datelor de intrare a fost prea mic. A apărut necesitatea rulării unui număr mare de benchmark-uri care să acopere toate insuficienţele constatate şi astfel realizarea unui volum de teste cu caracter unic şi care să cuprindă cât mai multe variabile. Prin testele din acest capitol s-a reuşit acest lucru.

Testele au fost efectuate în două etape. În prima etapă s-a realizat evaluarea performanţei algoritmilor care au fost rulaţi pe fişiere de dimensiune de 1 MB şi 100Mb. A doua etapă a fost mai amplă şi s-a concretizat în rularea aplicaţiilor pe fişiere de 1 GB, 2GB, 3 GB, ...10 GB. O etapă separată a constat în testele efectuate pe date din memorie.

Comparaţiile efectuate au vizat, în primă fază performanţele platformelor [TOM10/3]. Dintre cele cinci platforme testate, în cazul volumelor mari de date, se poate spune că în cazul algoritmilor HMAC, platforma PC2 obţine cele mai bune rezultate. Pentru fişierele mai mici (de până la 3 GB), se întâmplă ca platforma PC1 să obţină timpi sensibil mai mici pentru HMACSHA512 şi HMACMD5. Platforma cu cele mai slabe performanţe este platforma PC3. Aceasta este şi cea mai slabă din punctul de vedere al caracteristicilor tehnice. Un alt aspect este acela că platforma PC1 a obţinut în mai multe cazuri timpi similari pentru cei patru algoritmi HMAC testaţi atunci când au avut acelaşi fişier de intrare. Această situaţie a apărut cu predilecţie în cazul sistemului de operare UNIX. Acest aspect poate fi justificat datorită faptului că, în cazul platformei PC1 timpii de acces la fişiere au fost relativ mari şi au afectat timpii de computaţie pentru rularea primitivelor criptografice. În cazul celorlalte platforme nu s-a observat un astfel de comportament. În schimb, platforma 4 are rezultate medii în comparaţie cu celelalte platforme, după cum se poate observa şi din figura 4.3.2.2. În cazul sistemelor de operare Windows pentru

acest sector de teste se observă că platforma PC5 obţine rezultate sensibil mai bune decât restul platformelor. Acest comportament nu este constant deoarece în anumite etape, fără să se respecte un tipar PC5, este întrecută ca performanţă de PC2 sau PC1, sau chiar de ambele în acelaşi timp. Aceste analize au fost cuprinse şi în [TOM10/3] unde s-a discutat despre comportamentul primitivelor criptografice în cazul volumelor mari de date. Totodată, în [TOM10/3], s-au prezentat factorii care afectează performanţele primitivelor pentru datele de mari dimensiuni. În [TOM10/1] s-a realizat o analiză a performanţelor algoritmului AES în funcţie de platformele computaţionale. Un studiu asemănător s-a realizat în [TOM10/2] pentru algoritmii simetrici de criptare.

În cazul criptărilor simetrice, putem spune că platforma PC2 obţine cele mai bune rezultate sub Unix. În cazul sistemului de operare Windows, pentru algoritmul AES, în limbajul C#, începând de la fişierele de 6 GB în sus platforma PC2 este întrecută de platforma PC1 [TOM10/2]. Situaţia nu se repetă şi în cazul limbajului Visual Basic. În cazul Visual Basic se păstrează, în general, acelaşi comportament ca şi la OpenSSl sub Unix. Platforma PC5 a obţinut în aceste teste performanţele cele mai slabe având timpi foarte mari. O explicaţie ar consta în aceea că Level 2 Cache al platformei PC5 este de 3 MB, la fel ca şi PC1 şi mai mic decât PC2, care are 4 MB. Acest lucru, alături de faptul că PC5 are două hard discuri în RAID de mirroring (cu oglindire = copierea datelor pe mai multe discuri) pot să conducă la performanţe mai slabe şi decât PC3 [TOM10/4]. Acest comportament al lui PC5 a fost similar şi în cazul funcţiilor HMAC, dar performanţa a variat, după cum s-a specificat în paragraful anterior [TOM10/1].

Analizând rezultatele obţinute pentru funcţiile hash putem spune că platformele au comportament similar cu cel din cazul funcţiilor HMAC.

Când analizăm performanţele în ceea ce priveşte mediile de dezvoltare, se poate spune că, în cazul fişierului de 6 GB pentru algoritmul DES, atunci când am comparat platforma PC2 cu platforma PC3, am obţinut rezultate contradictorii. Pentru OpenSSL DES obţine cel mai scurt timp pe platforma PC2 şi cel mai lung timp pe platforma PC3 (fig. 4.4.12). C# şi Visual Basic se situează ca performanţe între valorile stabilite de OpenSSL pentru cele două platforme, dar dacă, în C#, PC2 obţine timp mai slab decât PC3, în Visual Basic rezultatele sunt aproape egale ca valoare, dar PC2 are timp mai bun decât PC3. În figura 4.4.1 se poate observa că C# obţine cel mai bun timp în cazul AES pentru fişierele de 1 MB şi 100 MB. Este urmat de Visual Basic şi apoi de OpenSSL. Acest comportament este caracteristic platformei PC2 [TOM10/2]. În figura 4.4.2 se poate observa comportamentul AES pe toate cele 10 fişiere şi pe cele trei medii de dezvoltare. După cum se poate remarca la fişierele de mici dimensiuni, performanţă sporită au algoritmii în C#. La fişierele de dimensiuni mari până la dimensiunea de 5 GB C#, aceasta creşte în timp uniform şi are perioadă de computaţie mai mică decât VB sau OpenSSL. Începând de la fişierele de 6 GB se observă o creştere mult mai mare decât la VB sau OpenSSL. Singura creştere uniformă pe toate cele zece fişiere de mari dimensiuni o are OpenSSL, dar timpii obţinuţi nu sunt mai buni decât cei scoşi de VB respectiv C# [TOM10/1]. În figura 4.4.4 se poate observa că pentru funcţia HMAC SHA256 pentru fişierul de 4 GB în cazul platformei PC1 Visual Basic şi OpenSSL au timpi apropiaţi, pe când C# are performanţe aproape de două ori mai slabe. În tabelul 4.2.1.1 şi 4.2.1.2 se poate observa că platforma PC5 sub C# obţine pentru HMACSHA1 timpi de computaţie mult mai mari decât celelalte primitive HMAC, acest rezultat fiind anormal. În cazul celorlalte fişiere (1GB...10GB) rezultatele obţinute pentru HMAC sunt în ordine crescătoare: HMACMD5, HMACSHA1, HMAC SHA256 HMAC SHA512. Comportamentul clasic este prezentat în figura 4.2.2.1 pentru PC5 şi PC2. Timpii

obţinuţi sunt tot sub C# [TOM10/3]. Se poate observa că PC3 în acest caz nu respectă tiparul, dar pentru alte fişiere revine la normal. Funcţiile hash au comportament similar cu cele ale funcţiilor HMAC. În multe cazuri, MD5 şi SHA1 ca de altfel şi HMAC MD5 şi HMAC SHA 1 au timpi similari. Cel mai mare timp îl obţine de obicei SHA512, respectiv HMAC SHA512, cum de altfel este şi normal. În general, se poate observa că nici în cadrul mediilor de dezvoltare nu se poate stabili un comportament uniform pe întreaga plajă a valorilor de intrare şi nici pentru platformele testate. O analiză din punct de vedere al mediilor de dezvoltare s-a realizat în [TOM10/4] pentru algoritmul AES.

O altă analiză este cea din punctul de vedere al algoritmilor. În cazul algoritmilor HMAC, ordinea normală performanţelor ar fi: HMACMD5, HMACSHA1, HMACSHA256 HMACSHA512 [SCH03]. Această ordine se păstrează în majoritatea testelor, dar sunt cazuri precum cele din tabelele 4.1.2.3 (PC5 şi PC3), 4.1.24 (PC5), 4.1.2.8 (PC5 şi PC3), 4.1.2.10 (PC5 şi PC3), 4.2.1.1 (PC5 şi PC3), 4.2.1.2 (PC5), figura 4.1.2.1 etc., în care această ordine nu se respectă demonstrând că, fie sistemele Windows nu sunt în totalitate multitasking, şi după o perioadă de timp cedează procesorul altui task afectând astfel timpul în mod aleator, fie că sistemul Kubuntu întâmpină probleme în a citi fişierele de mari dimensiuni de pe o partiţie NTFS (figura 4.3.2.2). Funcţiile hash au comportament similar cu HMAC (fig. 4.4.6,fig. 4.4.7 etc.). Analizând algoritmii simetrici putem spune că sub Kubuntu ordinea performanţelor a fost următoarea: AES, DES, 3 DES, aceasta fiind uniformă pe platformele testate. Sub Windows acest caracter uniform nu se mai păstrează (figura 4.4.2 , fig. 4.4.12, fig 4.4.13, fig. 4.4.14, etc.).

O categorie separată de teste a fost realizată aplicând primitivele criptografice pe date stocate în memorie, de mici dimensiuni. În acest caz, mediile de dezvoltare au fost JAVA, C# şi Visual Basic fiind testate pe trei platforme diferite. Se observă că procesorul Atom al platformei 2 obţine rezultatele cele mai slabe (cu excepţia 3 DES sub C#), făcându-l pe acesta o alegere proastă pentru folosirea intensă a aplicaţiilor criptografice în comparaţie cu celelalte platforme. Acelaşi comportament reiese şi în cazul algoritmilor hash sau HMAC, unde platforma PC2 obţine timpi foarte mari de computaţie în comparaţie cu celelalte două. Ca şi performanţe cea mai rapidă platformă a fost PC3 urmată de PC1 iar pe ultimul loc clasat este PC2. Acest comportament nu este uniform şi există excepţii precum cele din fig. 4.5.5. Din figurile 4.5.6., 4.5.7., 4.5.8, 4.5.9 se poate trage concluzia că singurul limbaj de programare care păstrează acelaşi caracter de creştere a timpilor pentru primitivele criptografice testate este Java [TOM10/3]. Totodată, ca limbaj de programare, Java a avut, în general, cei mai buni timpi de computaţie pe cele trei platforme. C# şi Visual Basic au avut comportament asemănător între ele, deşi performanţele obţinute au fost mai bune în unele teste la C#, iar în altele la Visual Basic (tabel 4.5.1, 4.5.2, 4.5.3).

Rularea unui număr foarte mare de teste îşi are justificarea în faptul că majoritatea testelor efectuate în lucrări similare au un număr foarte mic de variabile de intrare (număr mic de platforme, număr mic de medii de dezvoltare, număr mic de valori de intrare). Rezultatele obţinute sunt corecte, dar în general puţine sau insuficiente pentru a se putea crea o imagine de ansamblu. În testele noastre am avut un număr mare de variabile, un număr mare de teste care au dus la un număr mare de rezultate. Numărul mare de variabile de intrare şi-au pus amprenta asupra rezultatelor. Acestea au fost în anumite situaţii atipice. Având un număr foarte mare de teste care au acoperit multe platforme, sisteme de operare, limbaje de programare şi date de intrare, putem spune că, în final, s-a putut obţine o imagine de ansamblu asupra performanţelor oferite de primitivele criptografice.

După cum s-a constatat analizând rezultatele obţinute în urma testelor efectuate, procesoarele oferă rezultate care variază în funcţie de mărimea fişierelor de intrare, a algoritmilor, a capacităţii memorie, dar şi a limbajului folosit sau a sistemului de operare. Pentru testele efectuate cu datele din memoria RAM s-a concluzionat că, în general, Java a avut timpi de computaţie sensibil mai mici decât Dot Net. Singurele segmente unde Java scade în performanţă faţă de Dot Net sunt algoritmii simetrici. Între C# şi Visual Basic nu au fost diferenţe mari, aceştia având timpi apropiaţi şi comportament similar (a se vedea fig 4.5.1-4.5.12). La aceste teste, cel mai slab procesor a fost Atom N270. Legat de testele realizate pe fişiere putem spune că C# şi Visual Basic au performanţe aproape identice, începând de la 6 Gb până la 10 Gb, în cazul algoritmului AES. Până în 6 Gb C# are un uşor câştig în performanţă (fig 4.4.15). OpenSSL are acelaşi comportament cu Visual Basic până la 9 Gb, peste acest prag, timpul de computaţie al OpenSSL creşte brusc. Acest comportament este caracteristic platformei PC3 care a fost una din cele mai lente din lotul ales. În cazul algoritmilor hash (în special SHA256), pe platforma PC2 s-a observat că Visual Basic şi OpenSSL au scos timpi apropiaţi pe întreaga plajă de fişiere testate, iar C# a obţinut performanţe mult mai scăzute (fig 4.4.16). Platforma PC2 a obţinut, în general, cele mai bune rezultate în toate seturile de teste efectuate. Surpriza testelor a fost platforma PC5, care, în Windows 2000, a obţinut timpi foarte slabi în Dot Net, deşi configuraţia hardware este cea mai bună din cele cinci platforme. Un factor decisiv în aceste rezultate l-au avut cele două hard disc-uri în RAID pentru mirroring, nefiind un RAID de viteză. Dacă în cazul platformelor de test se remarcă un câştigător cu uşurinţă, nu putem spune acelaşi lucru despre limbajele de programare şi sistemele de operare. În acest din urmă caz, nu s-a putut stabili clar care din sistemele de operare, respectiv limbajele de programare, are cele mai bune performanţe. Fiecare limbaj de programare testat obţine performanţe mai bune în detrimentul celorlalte pe anumite sectoare de teste. Acesta este surclasat de celelalte fie în acelaşi test, fie în alte teste.

În urma analizei realizate între sistemele de operare, se poate spune că sistemul de operare Unix oferă stabilitate mai bună în ceea ce priveşte păstrarea caracterului crescător pe cele zece fişiere testate. Sistemul de operare Windows nu a oferit o astfel de stabilitate pe întregul set de fişiere în testele efectuate, fiind cazuri în care primitivele criptografice au avut nevoie de mai mult timp în cazul fişierelor mai mici decât în cazul fişierelor mai mari. Unix poate fi considerat mai stabil decât Windows, dar din punctul de vedere al performanţei, niciunul din cele două sisteme de operare nu este un câştigător.

O situaţie similară este oferită atunci când comparăm rezultatele obţinute de mediul Visual Studio şi Java. În acest caz, toate mediile de dezvoltare fiind testate pe Windows, se poate spune că Java respectă caracterul indicat de [CISCO SEC] şi prezentat în tabelul 2.1, în care se specifică comportamentul firesc al primitivelor criptografice (preformanţa), pe când în mediile Visual Basic şi C# apar situaţii în care timpii necesari unor primitive criptografice sunt mai mari decât ale altora deşi nu este normal.

În urma analizelor efectuate asupra tuturor rezultatelor, s-a ajuns la concluzia că rezultatele testelor erau afectate de prea multe variabile, deoarece platformele ofereau o varietate de configuraţii şi s-a constatat necesitatea unei schimbări ce poate consta în testarea unei platforme care să poată aduce performanţe sporite în comparaţie cu rezultatele obţinute în acest capitol [TOM10/6]. Existau mai multe direcţii printre care enumerăm: implementarea unui algoritm într-un limbaj, astfel încât aplicaţia finală să ruleze pe 64 de biţi, implementarea unui algoritm care să beneficieze de paralelizare şi să poată rula pe mai multe nuclee sau procesoare

simultan şi implementarea unui algoritm pe un procesor neconvenţional care să ajute procesorul dedicat la operaţia de criptare (un coprocesor criptografic).

A fost aleasă ultima variantă, în care s-a optat pentru un procesor video. Această alegere poste fi motivată prin faptul că folosirea plăcii video şi pentru alte scopuri decât cele pentru care placa a fost special concepută prezintă un caracter de noutate. Un alt motiv constă în faptul că autorul a considerat că această soluţie este superioară unui procesor clasic.

5. Propunerea unor soluţii de accelerare a AES pe GPU-CUDA

Dezvoltarea procesoarelor clasice se apropie de limitele tehnologice. Astfel, pentru o mai mare performanţă a procesorului, producătorii măresc frecvenţa, dar mărind frecvenţa creşte temperatura şi totodată consumul. O soluţie la această problemă a fost oferită de procesoarele cu multi nucleu. Inserarea mai multor nuclee oferă un spor de performanţă, dar presupune programe software special concepute pentru acele procesoare. O alternativă la procesoarele multinucleu ar fi Grid Computing (folosirea de compomente IT din sisteme informatice şi de comunicaţii din domenii administrative diferite pentru un scop comun), dar această alternativă este complexă şi presupune un număr mare de calculatoare/ resurse. Aceste două soluţii se pot dovedi costisitoare. Un procesor multinucleu este mai scump decât o placă video mediocră şi nu poate oferi performanţele pe care un GPU le poate oferi [HAR08]. Grid Computing presupune folosirea de resurse ale mai multor sisteme informatice, ceea ce se va reflecta în costuri mai mari.

O soluţie la problemele ridicate mai sus este oferită de procesoarele grafice (Graphical Processing Unit, GPU). Un GPU conţine mai multe nuclee şi poate rula în paralel mai multe thread-uri (fire de execuţie).

Din toţi algoritmii testaţi în această lucrare, pentru capitolul 5 a fost ales algoritmul de criptare simetrică AES, acesta fiind standardul pentru criptarea simetrică în acest moment. Ca şi în testele precedente ne vom orienta pe teste cu date stocate în memorie şi date stocate pe hard disc (fişiere de mari dimensiuni).

Această încercare de a implementa şi adapta un algoritm criptografic pe un procesor grafic îşi are justificare în faptul că, teoretic, este mai ieftin de folosit procesorul grafic cu rol de coprocesor criptografic pentru a elibera procesorul de task-uri (sarcini) decât achiziţia unui procesor criptografic dedicat, mult mai scump.

Implementarea unui algoritm a cărui complexitate computaţională constă în operaţii simple (a se vedea tabelul 2.2.1) pe o placă grafică special concepută pentru operaţii complexe şi operaţii în virgulă flotantă constituie punctul de plecare al cercetării ştiinţifice a acestei teze.

O altă justificare ar fi aceea că procesorul grafic poate fi folosit cu succes în calitate de coprocesor criptografic în aplicaţii de tip web pe parte de server ce necesită autentificare şi criptare a datelor în timp real pentru a elibera din încărcarea procesorului. Totodată, poate fi folosit şi pentru criptarea unui flux mare de date precum streaming media (flux de date video sau audio-video de la un producător către un client) [HAR08].

5.1. Stadiul actual

În ultimii ani, din cauza evoluţiei lente a procesoarelor, dezvoltatorii de aplicaţii consumatoare de putere de calcul s-au orientat şi către alte tipuri de procesoare. Astfel, au fost luate în calcul procesoarele grafice. Acestea au fost iniţial dezvoltate şi proiectate pentru randări 3D (generarea unor imagini 3D pe baza unui model prin intermediul unui program), codare video (compresie video) şi pentru motoarele jocurilor (software destinat creării şi dezvoltării de jocuri pentru calculator). Aplicaţii ca cele amintite necesită o putere mare de calcul şi procesoarele calculatoarelor

personale nu puteau oferi acest lucru. Producătorii de plăci video au dezvoltat procesoare grafice din ce în ce mai puternice, oferind posibilitatea dezvoltatorilor de aplicaţii să îşi scrie propriile aplicaţii care folosesc coprocesarea pentru CPU şi pe GPU. Există programe, fie comerciale, fie open source, care folosesc procesorul video pentru coprocesare. De exemplu, player-ul video Mirillis Splash Pro ajută un calculator cu procesor mai slab să redea filme High Definition 1080p (filme de înaltă rezoluţie) fără să suprasolicite procesorul prin folosirea GPU-ului. În [NVI09] sunt prezentate plăcile video ale producătorului NVIDIA care suportă mediul de dezvoltare CUDA capabil să accelereze task-uri. Din aceste produse fac parte GeForce şi Quadro, care se pot instala într-un calculator personal care posedă slot PCI Express. O categorie aparte o reprezintă produsele TESLA. Unul din aceste produse are patru procesoare grafice cu 240 de nuclee (960 de nuclee) având 16 GB memorie cu lăţimea de bandă de 408 GB/sec. Cei de la NVIDIA spun că acest produs este primul cu procesoare multinucleu ce oferă performanţa de un teraflop. Aceştia oferă în [CUD09-1], [CUD09-2], [CUD09-3], [CUD09-4], [CUD09-5], [CUD09-6], [CUD09-7] documentaţie completă pentru mediul de dezvoltare, instrucţiuni, ghid de instalare, programare. O categorie nouă de procesoare grafice sunt ION pentru calculatoarele portabile. Acestea suportă şi ele CUDA.

În [YON08], autorul porneşte de la reuşita lui Cook [COO05] din 2005 de a implementa un algoritm criptografic pe un GPU, analizează performanţele îmbunătăţite folosind DirectX şi OpenGL şi, în urma cercetării făcute de acesta, concluzionează că un procesor Intel Core 2 Quad (QX6850) are capacitate de 96 GFLOP, pe când o placă video NVIDIA GEFORCE 8800GTX are capacitate de 330 GFLOP. De asemenea, algoritmul AES are o performanţă de 4.5 Gbps, iar DES de 2.8 Gbps pe placa video amintită.

Michael Kipper, în [KIP09], tratează subiectul implementării algoritmului AES pe GPU şi concluzionează că GPU are o performanţă de 14.5 ori mai mare decât în cazul unui procesor. Tot el afirmă că tentative de crack prin brute-force (spargerea codului prin folosirea repetată pe post de cheie a tuturor posibilităţilor) a lui AES sunt încă nefezabile la o accelerare de performanţă de acest ordin. Într-o lucrare similară, Luken Brandon, [LUK09], tratează subiectul criptării AES şi DES folosind accelerarea unui GPU. Testele s-au efectuat pe valori de intrare de până la 100 MB, iar performanţele obţinute au fost următoarele: AES este de 3.75 ori mai rapid decât pe un CPU, iar DES de 4.5 ori mai rapid decât pe un CPU.

Manavski, Svetlin prezintă în [MAN07] rezultatele obţinute la testarea compatibilităţii CUDA pentru procesoare grafice NVIDIA în accelerarea hardware pentru criptare folosind AES. Cele mai bune performanţe obţinute au fost în cazul algoritmului AES 128 pentru fişier de intrare de 8 MB, performanţa fiind de 8.28 Gbps, iar algoritmul fiind mai rapid de 19,60 ori decât în cazul CPU-ului. În figura 5.1.1 este prezentată performanţa lui AES 256, figura fiind preluată din [MAN07].

O altă implementare AES pe GPU este realizată în [TAK07]. Autorul realizează două teste: „Vertex Program" şi „Fragment Program". Implementarea este în OpenGl bazată pe limbaj de asamblare. Folosind ca platformă de test un procesor Pentium 4 3 Ghz 2MB level 2 cache, 1 GB RAM şi placa video GeForce 8800 GTS 640 MB, pentru valori de intrare cuprinse între 4KB şi 16 MB obţine performanţe între 10Mbps şi 95 Mbps. În teste a fost folosit modul CBC ce este paralelizabil doar la decriptare. În final, concluzionează că pentru o performanţă mai bună se poate folosi modul CTR care ajută la paralelizare şi simplifică şi partea de decriptare fiind folosit acelaşi cod.

În [BOS09] sunt prezentate mai multe optimizări ale AES pentru AVR microcontroller pe 8 biţi, Cell şi pe GPU (8800 GTX şi GTX295). Folosind ca mediu

de dezvoltare CUDA pentru NVIDIA obţine performanţe de 59.6 Gbps (GTX295) şi 14.6 Gbps (8800 GTX) pentru criptare şi 52.4 Gbps (GTX295) şi 14.3 Gbps (8800 GTX) pentru decriptare. Implementările din [BOS09] folosesc doar tabelele de căutare ca principală metodă de optimizare. Nu se specifică în această lucrare pentru ce valoare de intrare s-au obţinut performanţele prezentate. Modul folosit este CTR, dar în acest caz codul de criptare fiind identic cu cel de decriptare, nu ar trebui să existe diferenţe mari între performanţele obţinute la criptare şi la decriptare.

[ZAJ06] prezintă o metodă de a modifica MixColumns pentru AES în care concluzionează că este mai rapid decât cel original, dar nu prezintă modalitatea de testare şi nici sporul de performanţă obţinut.

În [BRO05] s-a realizat o accelerare AES pe FPGA cu o performanţă notabilă la acea vreme, de 2 Gbps. Dezavantajul unei astfel de implementări o constituie achiziţionarea unui FPGA. Această performanţă a fost întrecută în anul 2007, atunci când [MAN07] a obţinut 8.28 Gbps, performanţă cu AES pe o placă video asemănătoare celei folosite în testele actuale. [PAR04] prezintă o implementare AES pe FPGA ce obţine performanţe de 21.56 Gbps. O implementare asemănătoare cu [PAR04] este cea realizată în [GOO05], în care sunt prezentate rezultate notabile (25 Gbps), dar fiind o implementare AES pe FPGA prezintă pe lângă dezavantajele din [BRO05] şi necesitatea unor echipamente cu număr mare de porţi (1-2 milioane). O altă lucrare este [BIE05], în care se propune o paralelizare AES pe sistem multiprocesor. În testele realizate s-au folosit 64x Itanium2 1.5GHz (SGI Altrix 3700) cu 16 procesoare şi rezultatele obţinute au arătat performanţe de 5.554 ori mai rapide pentru criptare , respectiv 12.357 ori mai rapide pentru decriptare. [LEE10] prezintă o implementare de procesor pentru accelerarea AES cu performanţe de 5.8 Gbps. [BIA09] realizează o implementare AES folosind CUDA în care se obţin performanţe de peste 10Gbps. [JAC10] realizează o verificare similară celei propuse în această lucrare, dar face comparaţie doar între CPU şi GPGPU fără a testa şi pe alte platforme. Concluzia la care s-a ajuns a fost că procesoarele performante actuale nu sunt suficiente pentru a atinge performanţele obiectivelor propuse, iar găsirea unei soluţii pentru accelerarea AES pe GPU este extrem de dificilă. [www6] propune o implementare AES pe CUDA, dar pentru paralelizare, foloseşte modul ECB ca şi ceilalţi autori. În [HOD05] se propune un coprocesor criptografic bazat pe tehnologia 0.18-µm CMOS. În urma testelor efectuate se obţine o performanţă de 3.84Gbps, iar acest procesor suportă moduri precum ECB, OFB, CBC. [STA03] prezintă o implementare Rijandael pe FPGA în care reuşeşte să obţină o performanţă de 18.5 Gbps. O altă implementare este prezentată de [CHO03] care, folosind un FPGA de 10 $, obţine performanţe de 166 Mbps. În [ATA07/01] şi [ATA07/02] se vorbeşte despre necesitatea identificării unor soluţii pentru coprocesarea în cazul criptării. O astfel de soluţie a fost oferită de [KAK07] prin propunerea unui coprocesor criptografic numit HSSec ce este capabil să folosească SHA1 şi SHA512 ca funcţii hash şi AES pe 128 de biţi ca algoritm simetric de criptare. Performanţele obţinute în urma testelor au fost de 1 Gbps.

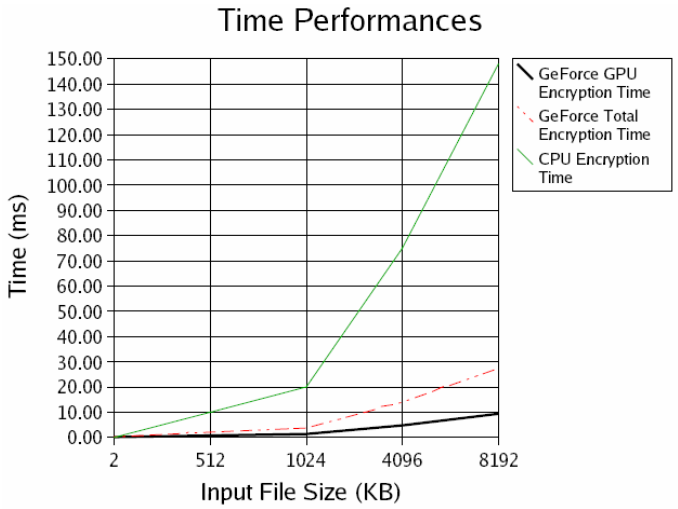

Figura 5.1.1 Performanţa lui AES 256 - preluare[MAN07]

5.2. GPU - Placa video. Arhitectură şi specificaţii

În zilele noastre, plăcile video oferă performanţe computaţionale remarcabile şi sunt folosite deja în multe domenii care necesită putere mare de calcul şi paralelizare. Procesoarele video sunt proiectate mai mult pentru procesarea datelor decât pentru data caching şi flow control. Aceasta înseamnă că task-urile sunt terminate mai repede. [CUD09-2]

Nvidia 8800 GT este bazat pe nucleul G80, care a fost primul nucleu de 65nm Nvidia. Acesta conţine 754 de milioane de tranzistori şi 128 procesoare [HSH08]. 8800GT funcţionează la o frecvenţă de 600MHz, având o memorie de 512 MB pe 256 de biţi la o frecvenţă de 900MHz, iar frecvenţa internă a nucleului este de 1,5Ghz.

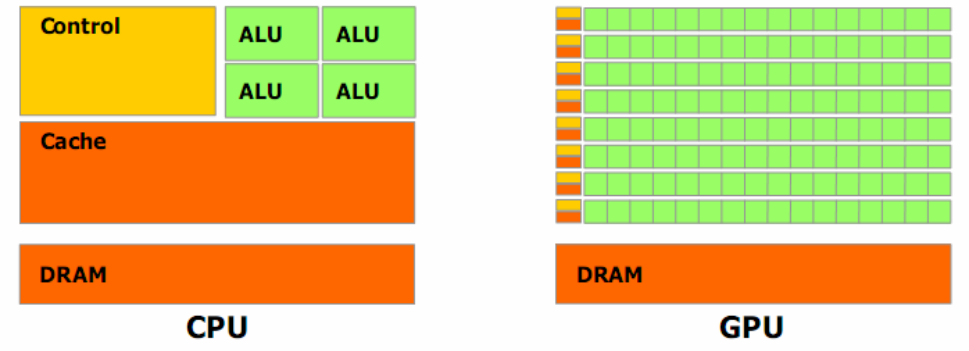

Figura 5.2.1 GPU este orientat pentru procesarea datelor [CUD09-2]

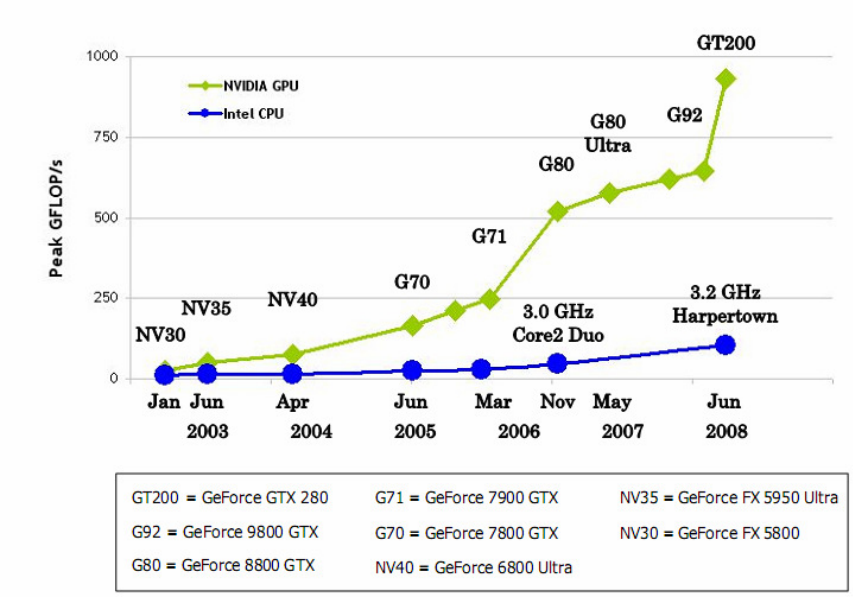

Figura 5.2.2 CPU VS GPU. Operaţii în virgulă mobilă [CUD09-2]

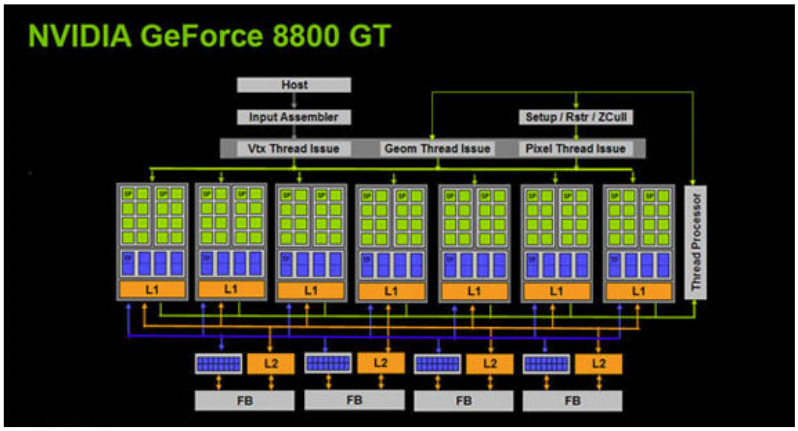

Figura 5.2.3 Nvidia 8800 GT. Arhitectură [NVI10]

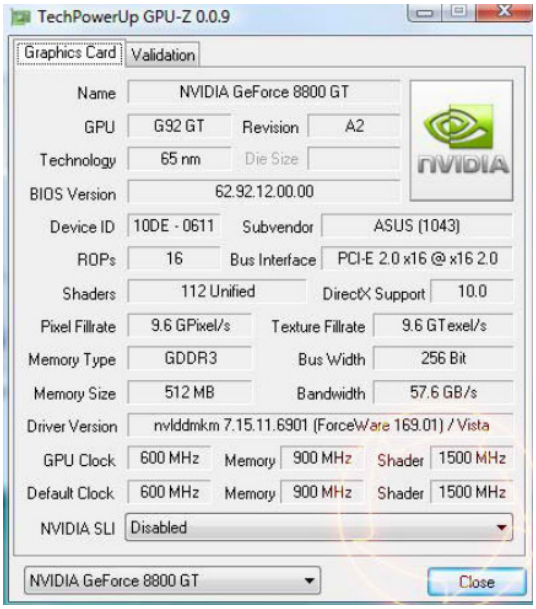

Figura 5.2.3 Nvidia 8800 GT specificaţii tehnice [NVI10]

Procesoarele G80 sunt compatibile DX10 şi aparţin primei generaţii de procesoare grafice care suportă date de tip întreg şi operaţii pe biţi. Acest lucru reprezintă un aspect important pentru criptografie [HAR08].

G80 este alcătuit din 16 multiprocesoare care sunt cuprinse pe un singur chip. Fiecare multiprocesor este alcătuit din 8 unităţi aritmetico-logice (ALU), ce sunt controlate de o singură unitate de instrucţiune singulară în mod SIMD (Single Instruction Multiple Data). Unitatea de instrucţiune comandă o singură instrucţiune către unităţile aritmetico-logice la fiecare patru cicluri de ceas [HAR08]. Acest lucru oferă fiecărui multiprocesor o capacitate de 32 SIMD. Astfel, fiecare multiprocesor are registre de 32 de biţi de memorie, memorie partajată, memorie cache constantă. Toate celelalte tipuri de memorie sunt localizate în memoria globală (în afara chipului) [HAR08].

	8800 GT	**8800GTS**	**8800 GTX**
Memorie	512 MB 256 bit GDDR3	640 MB 384 bit GDDR3	768 MB 384 bit GDDR3
Nume de cod	G92-200	G92-400	G80
Frecvenţă	600 Mhz	650 MHz	612 Mhz
Lăţime de bandă GB/s	57.6	62.1	86.4
Procesoare	128	128	128

Tabel 5.2.1 Comparare 8800 GT vs 8800 GTS vs 8800GTX

5.3. CUDA

CUDA (Compute Unified Device Architecture), a fost introdusă în noiembrie 2006 ca o nouă arhitectură paralelă de computaţie, cu set nou de instrucţiuni şi model de programare paralelă [CUD09-2]. CUDA oferă un mediu software care permite

dezvoltatorilor să folosească C ca limbaj de programare de nivel înalt. Pe lângă acest limbaj mai sunt suportate şi alte limbaje de programare.

Când se programează sub CUDA, procesorul video este văzut ca un dispozitiv computaţional capabil să execute un număr mare de thread-uri în paralel. Operează ca un coprocesor pentru CPU. Dacă se poate izola o zonă din program care se execută de mai multe ori, dar în mod independent şi folosind date diferite, aceasta poate fi transcrisă într-o funcţie ce se execută pe GPU ca mai multe thread-uri diferite [ROS07]. Pentru a atinge acest punct, funcţia foloseşte setul de instrucţiuni al GPU-ului şi programul este încărcat în GPU.

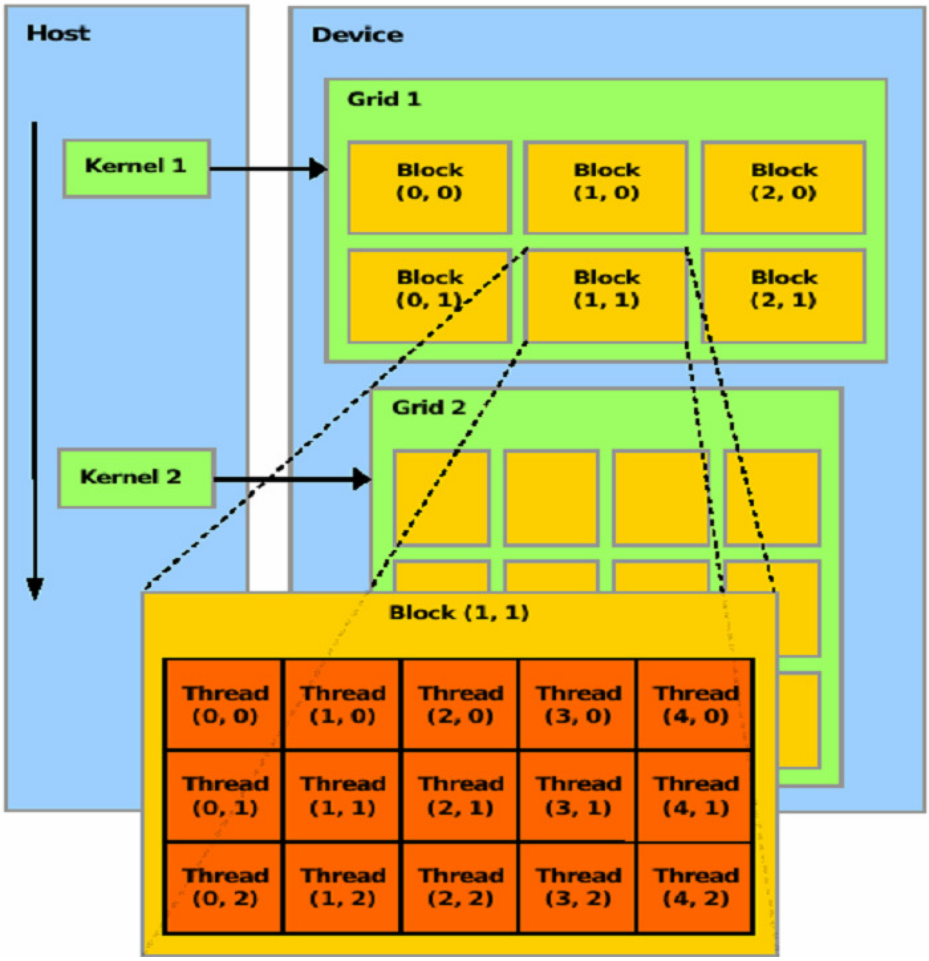

Figura 5.3.1 Dozarea thread-urilor [ROS07]

În figura 5.3.1 se poate observa cum GPU este tratat ca un coprocesor atunci când o funcţie (kernel) este apelată, iar kernelul va rula pe un grid, care, în funcţie de apelare poate să aibă o configuraţie predefinită. Numărul de blocuri dintr-un grid şi numărul de thread-uri dintr-un bloc sunt configurabile.

Toate datele, cheia de criptare şi S-BOX se vor afla în memoria globală, iar variabilele locale se găsesc în registre (fig. 5.3.2).

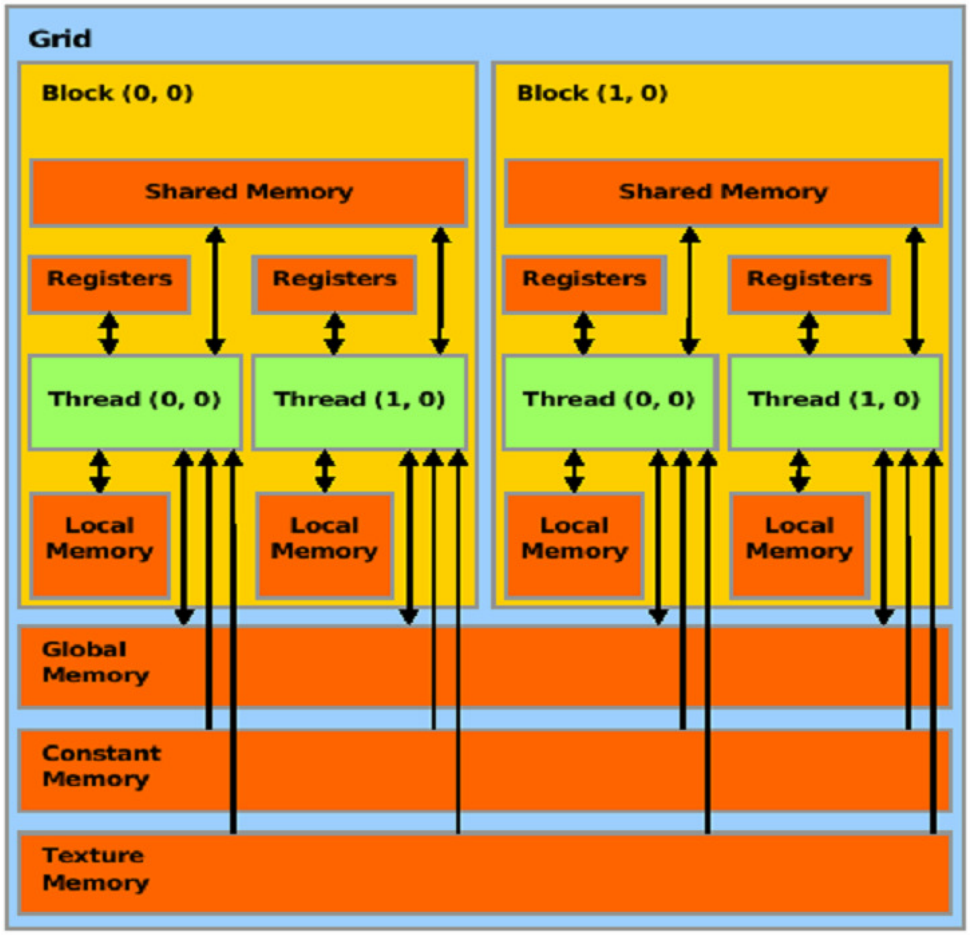

Figura 5.3.2 8800 Memorie [ROS07]

În CUDA, thread-urile în timpul execuţiei pot accesa datele din spaţii de memorie multiple. Fiecare thread are o memorie locală privată. Fiecare bloc are o zonă de memorie partajată accesibilă tuturor thread-urilor din acel bloc şi din aceeaşi durată de viaţă. Toate thread-urile au acces la memoria globală.

În CUDA structura memoriei se împarte în mai multe tipuri. Memoria partajată şi registrele sunt cele mai rapide şi aparţin doar unui thread sau bloc. Memoria locală şi globală sunt cele mai lente şi sunt accesibile de un thread sau de oricare bloc [LUK09]. Memoria constantă poate avea aceeaşi viteză ca cea partajată, dar acest lucru nu e garantat.

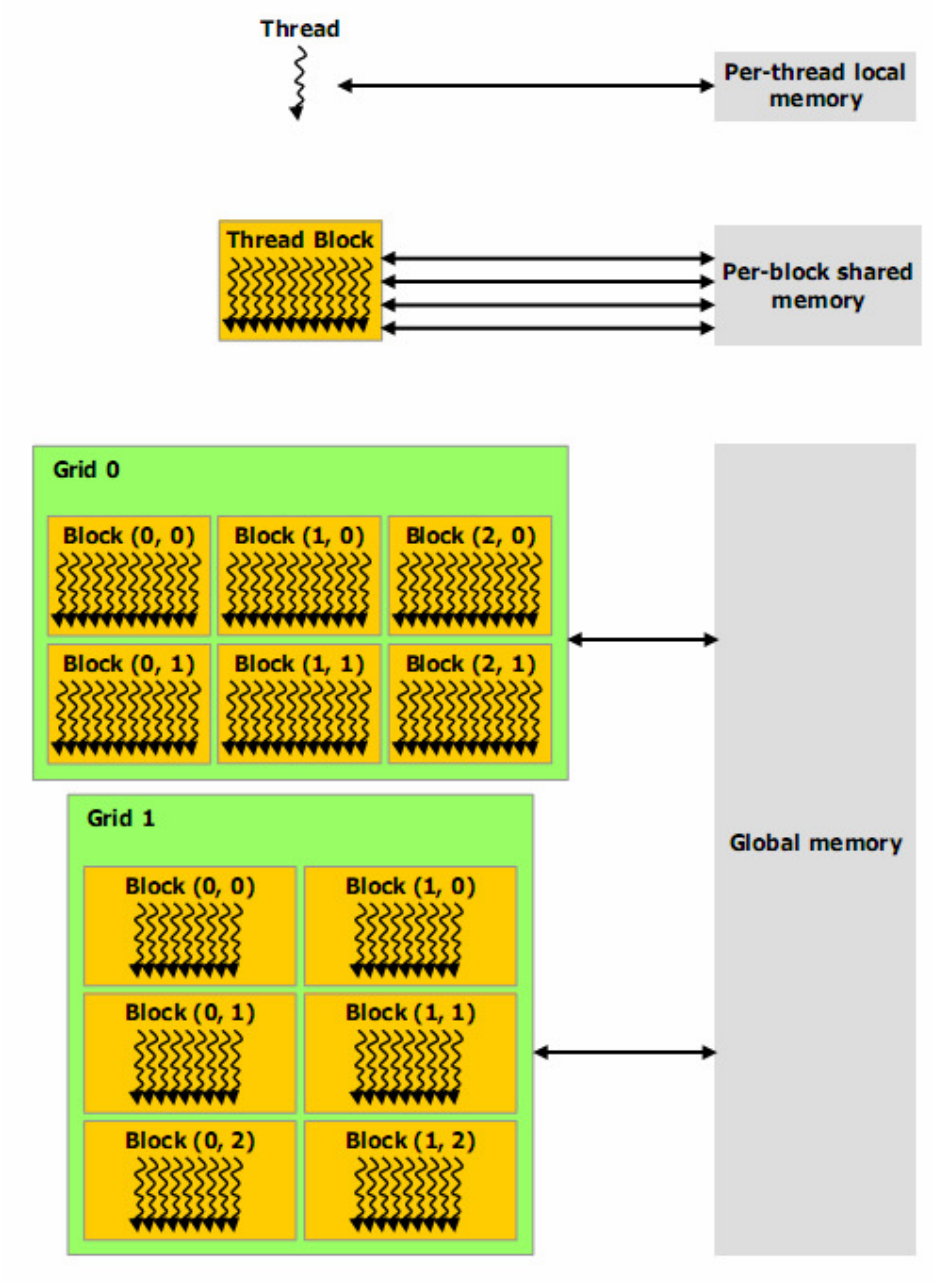

Figura 5.3.3 Ierarhia memoriei [CUD09-2]

Limbajul C pentru CUDA permite programatorului să definească funcţii (kernel), care atunci când sunt apelate, se execută pe mai multe thread-uri CUDA, de mai multe ori în paralel, spre deosebire de funcţiile C clasice.

Un kernel poate fi executat de mai multe blocuri de thread-uri, astfel încât numărul total de thread-uri să fie egal cu numărul maxim de thread-uri per bloc înmulţit cu numărul de blocuri. Aceste blocuri multiple sunt organizate în griduri uni sau bidimensionale (fig. 5.3.4).

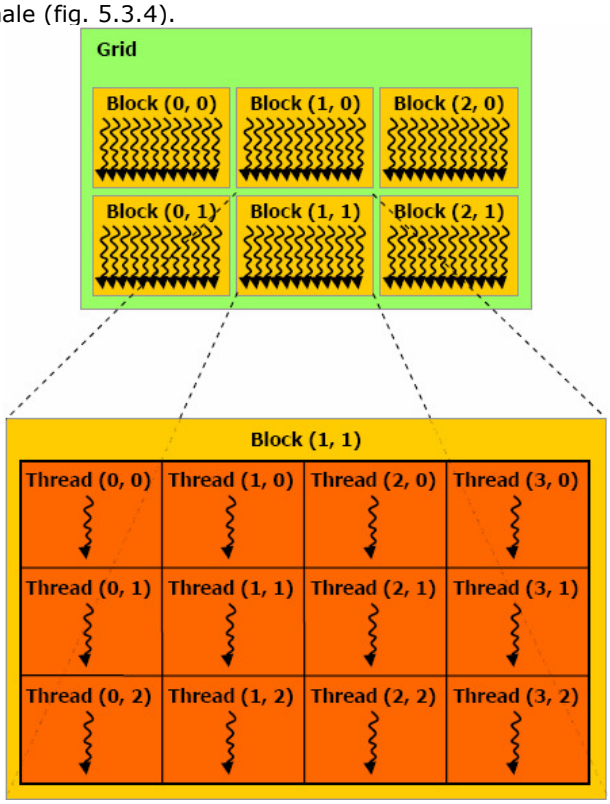

Figura 5.3.4 Grid cu blocuri de thread-uri [CUD09-2]

Modelul CUDA de programare presupune că atât CPU cât şi GPU au propriul DRAM. Programul administrează spaţiile de memorie vizibile kernel-ului prin apeluri CUDA. Aceasta include alocare şi eliberare de memorie precum şi transferul între memorii [CUD09-2].

Dacă un program are şi secvenţe seriale şi paralele, atunci se poate opta ca secvenţele paralele să fie rulate pe GPU, iar cele seriale pe CPU ca în figura 5.3.5.

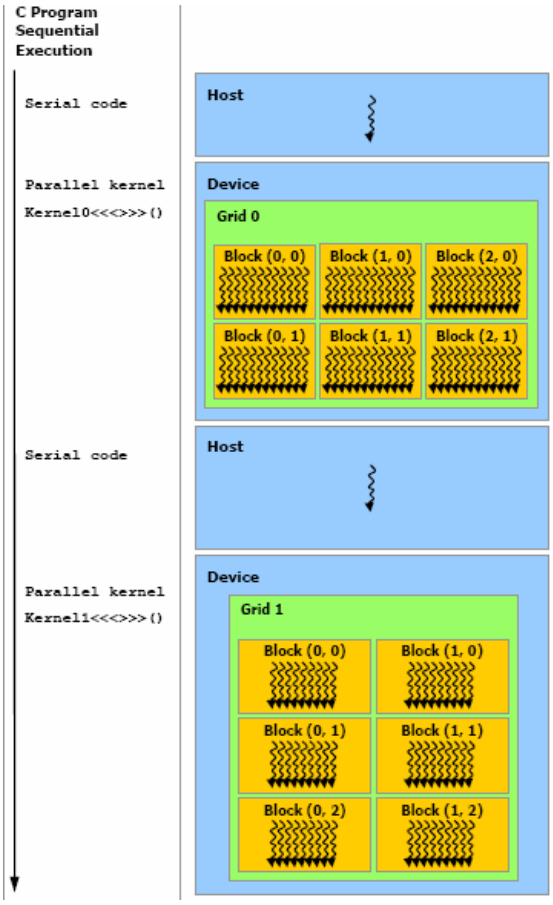

Figura 5.3.5 Secvenţe seriale şi paralele ale unui program C [CUD09-2]

Toate thread-urile(fire de execuţie) unui bloc vor rula pe un singur multiprocesor. Acest lucru permite thread-urilor dintr-un bloc să partajeze date cu alte thread-uri în interiorul blocului prin memoria partajată. Comunicarea interbloc nu este posibilă, deoarece nu este disponibilă o soluţie de sincronizare destinată acestui scop [HAR08].

5.3.1. Instalare CUDA pe Kubuntu 10.04[UBU10]

1. Pentru instalarea CUDA se deschide adresa: http://www.nvidia.com/object/cuda_get.html.
2. Se selectează sistemul de operare. În cazul de faţă, Linux 32 bit (Kubuntu 10.04). Se downloadează driver-ul CUDA, Driver-ul plăcii video, toolkit şi SDK. Acestea sunt:
   NVIDIA-Linux-x86-256.53.run
   cudatoolkit_3.2_linux_32_rhel5.3.run
   cuda-sdk-linux-3.2.run

cudadriver_3.2_linux_32_256.53.run
În funcţie de numărul versiunii actualizate, aceste nume pot să difere

3. Pentru actualizarea driverului NVIDIA trebuie oprit serverul X Windows. Pentru oprirea lui X Windows se foloseşte una din comenzile:
   sudo /etc/init.d/gdm stop
   sudo /etc/init.d/kdm stop
4. Se foloseşte Ctrl+Alt+F1 pentru o nouă sesiune şi se loghează.
5. Se dezinstalează driverele NVIDIA vechi şi cele care au strânsă legătură cu placa video. Se foloseşte comanda:

sudo apt-get remove nvidia-glx-new nvidia-kernel-common ubuntu-restricted-extras linux-restricted-modules-2.6.24-19-generic linux-restricted-modules-2.6.24-22-generic linux-restricted-modules-2.6.24-23-generic linux-restricted-modules-generic

În funcţie de versiune, anumite caractere/denumiri pot diferi.

6. Se instalează utilitarele de dezvoltare
   sudo apt-get install build-essential libglut3-dev
7. Se rulează comanda
   sudo sh NVIDIA-Linux-x86-256.53.run, din folderul în care a fost downloadat fişierul.
8. Se dezactivează driver-ul default „nv”:

sudo vi /etc/default/linux-restricted-modules-common
Se editează DISABLED_MODULES="" astfel DISABLED_MODULES="nv"

9. Se reporneşte sistemul: sudo reboot
10. În această etapă, serverul X ar trebui să pornească fără probleme şi să folosească noul driver Nvidia.
11. Se instalează CUDA Toolkit: sudo sh cudatoolkit_3.2_linux_32_rhel5.3.run
12. Se adaugă directoarele CUDA în calea default şi în variabilele de sistem adăugând în ~/.bashrc liniile:

CUDA
export PATH=$PATH:/usr/local/cuda/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda/lib
Se reporneşte sistemul.

13. Se instalează CUDA SDK
    sh cuda-sdk-linux-3.2.run
14. Dacă totul a decurs fără erori ar trebui să puteţi folosi CUDA.

5.4. Adaptare AES pe GPU- CUDA

În încercarea de a implementa şi optimiza algoritmul AES pe procesorul grafic, trebuie avute în vedere următoarele aspecte [ROS07]:

Optimizarea AES pentru CPU trebuie regândită deoarece aceasta are la bază tabele lookup care, teoretic, la GPU ar încetini procesul. Pentru a putea afirma acest lucru cu exactitate, implementarea a fost realizată în două etape: într-o etapă s-a implementat AES folosind tabele lookup, iar în altă etapă s-a implementat AES folosind calculele.

Procesoarele video au fost gândite pentru calcul paralel, şi nu sunt adecvate pentru algoritmi criptografici, care în general se bazează pe blocul anterior pentru criptare (serializare).

Implementarea algoritmului trebuie să se bazeze mai mult pe complexitatea aritmetică decât pe operaţii de memorie.

Avantajele folosirii unei plăci video care suportă mediul de programare CUDA, constau în faptul că aceasta foloseşte şi operaţii de shiftare, oferă flexibilitate la accesul memoriei (scriere/citire), se pot defini date direct în GPU fără a mai fi necesară o operaţie în plus de copiere [ROS07].

În cazul procesoarelor clasice, optimizarea algoritmului AES se baza foarte mult pe înlocuirea operaţiilor aritmetice cu tabele de căutare (tabele lookup). Spre deosebire de CPU, un GPU poate face mai multe operaţii într-un ciclu decât CPU-ul. Conform [ROS07] când se analizează accesul la memorie GPU are latenţe mai mari decât CPU, deoarece CPU stagnează execuţia pe durata acestui proces.

În testele efectuate de [STE06], s-a ajuns la concluzia că, în cazul unui număr mare de operaţii, GPU operează mai repede decât dacă se folosesc tabele şi se accesează memoria.

În cazul rulării algoritmului AES pe un procesor grafic, având în vedere faptul că se doreşte obţinerea unei performanţe mai bune decât pe un procesor clasic, şi că procesorul video este special proiectat pentru rularea mai multor sarcini în paralel, putem spune că trebuie alese, din toate modurile de operare posibile ale AES, doar cele care sunt pretabile la paralelizare. Acestea sunt ECB şi CTR. Trebuie să specificăm faptul că şi decriptarea în modul CBC este paralelizabilă, dar în acest caz doar unul din cei doi paşi ai criptării (al doilea - decriptarea) este paralelizabil. Modurile care pot fi folosite pentru AES sunt: Electronic Codebook ECB (fig. 5.4.3), Cipher Block Chaining CBC (fig. 5.4.4), Cipher Feedback CFB (fig. 5.4.5), Propagating Cipher Block Chaining PCBC (fig. 5.4.6), Output Feedback OFB (fig. 5.4.7), Counter Mode CTR (fig. 5.4.2). În figurile menţionate sunt prezentate doar modurile folosite la criptarea datelor.

Dintre cele două moduri paralelizabile a fost ales cel din urmă. Alegerea acestuia a fost făcută plecând de la ideea că modul CTR oferă o securitate sporită în comparaţie cu ECB. Dezavantajele modului ECB sunt acelea că acelaşi bloc de text în clar, în urma criptării va avea acelaşi bloc de text cifrat şi nu ascunde tiparele bine. De aceea, ECB nu este recomandat în protocoale criptografice deloc. Deşi în testele din capitolul patru s-a folosit ca mod de criptare ECB, aceste teste s-au realizat doar cu rol statistic. Ţinându-se cont că soluţiile propuse în acest capitol se doresc a avea utilitate practică s-a optat pentru modul CTR. Cunoscându-se modul de lucru *Counter mode,* putem spune că nu mai este nevoie şi de implementarea funcţiei de decriptare, deoarece se foloseşte acelaşi cod ca şi pentru criptare. Pentru teste a fost implementată doar funcţia de criptare, iar dacă se doreşte şi decriptare se mai aplică o dată funcţia de criptare.

Counter mode sau CTR se poate defini astfel:

$K_i := E(K, Nonce \| i), ptr.i = 1,...k$ şi $C_i := P_i \oplus K_i$ [FER03]

CTR foloseşte o metodă simplă de a genera cheile necesare. Concatenează câte un nonce (un număr sau şir de biţi folosiţi o singură dată) cu valoarea curentă a counter-ului (contor) şi îl criptează într-un singur bloc (fig. 5.4.2). Desigur un nonce trebuie să fie mai mic decât un singur bloc, deoarece trebuie să fie suficient loc şi pentru valoarea i din counter. O facilitate oferită de CTR este uşurinţa de a accesa părţi aleatoare din textul în clar. Principalul avantaj oferit de CTR este că se poate paraleliza pentru aplicaţii de mare viteză [FER03]. Principalul dezavantaj este că folosirea CTR-ului poate supune algoritmul la atacuri specializate de tip Hardware Fault Attack [TIR04].

O schemă simplificată pentru implementarea algoritmului prin paralelizare poate fi cea din figura 5.4.1.

Expansiunea cheii se face pe CPU, iar apoi criptarea propriu-zisă se va face pe GPU. Expansiunea cheii presupune operaţii ce se execută în mod serial şi de aceea se alege ca acest proces să fie rulat pe procesor, deoarece nu se doreşte încărcarea GPU-ului cu sarcini seriale.

Pentru a încerca o optimizare vom seta toate thread-urile să folosească memoria globală. Astfel, se poate realiza o grupare pentru accesul la memorie. Accesările datelor fuzionează pentru a permite GPU-lui să efectueze citiri de memorie mai rapide/mari [KIP09]. Accesul la memoria globală se face, în faza iniţială, înainte de procesarea datelor. Acestea sunt mutate în memoria partajată de unde sunt accesate mult mai rapid. Dacă fiecare thread încarcă date în memoria partajată din cea globală, pe care e posibil să nu le folosească, este necesară în această etapă o sincronizare înainte de folosirea memoriei partajate. Sincronizarea este necesară şi pentru scrierea înapoi în memoria globală [KIP09].

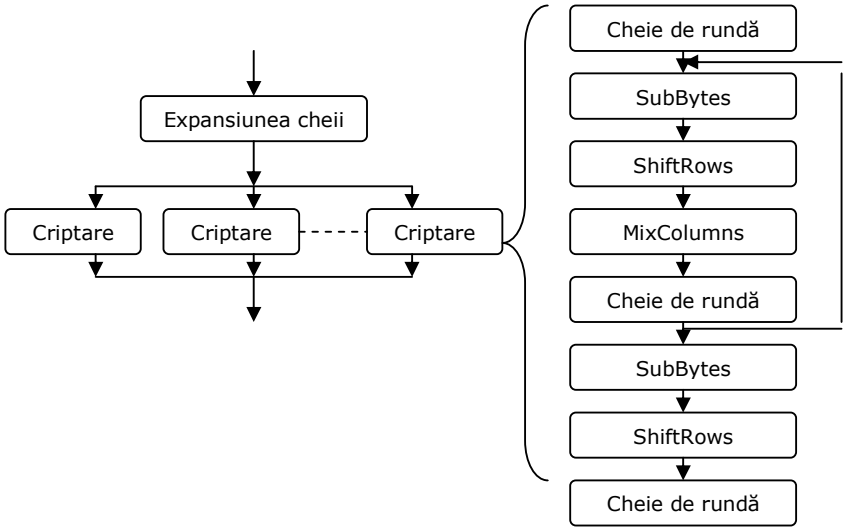

Figura 5.4.1 Paralelizare AES pe GPU

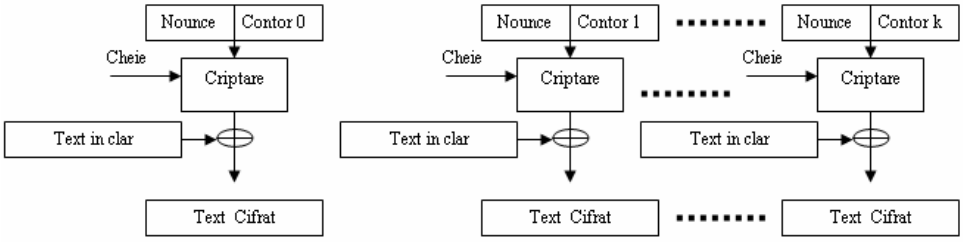

Figura 5.4.2 Counter mode

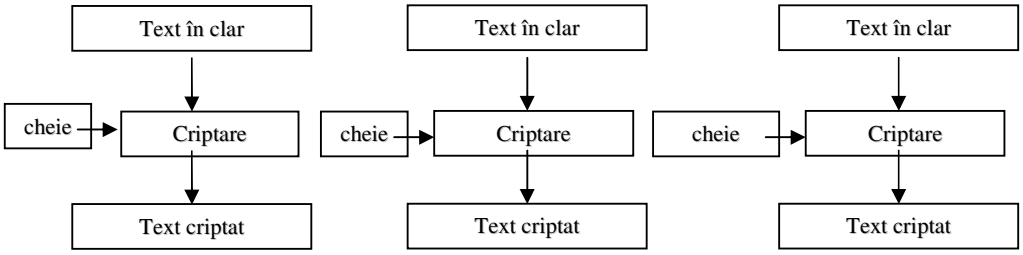

Figura 5.4.3 Electronic Codebook ECB mode

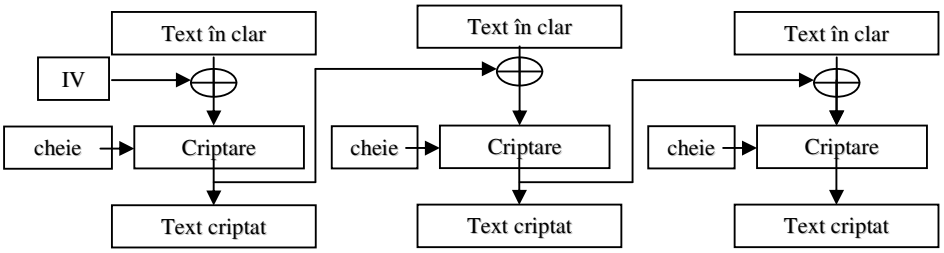

Figura 5.4.4 Cipher-block Chaining CBC mode

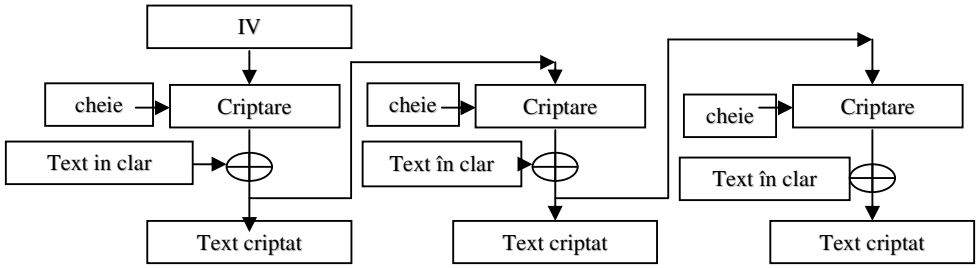

Figura 5.4.5 Cipher Feedback CFB mode

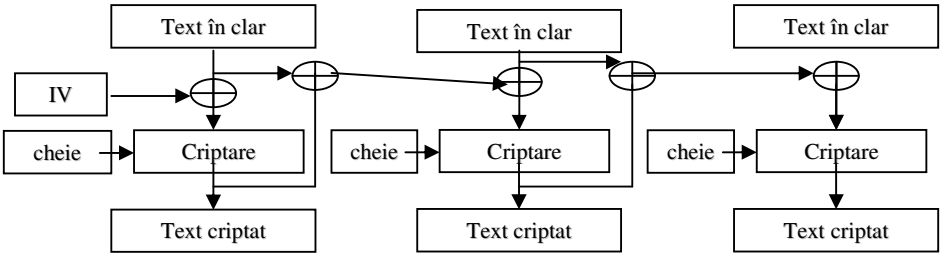

Figura 5.4.6 Propagating Cipher-block Chaining PCBC mode

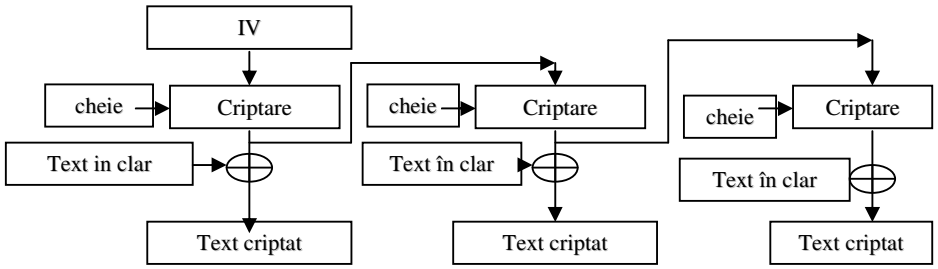

Figura 5.4.7 Output Feedback OFB mode

O posibilitate de implementare a algoritmului AES în C pentru CUDA, este aceea prin care, în mod similar optimizării AES pentru CPU, se folosesc tabele de căutare (lookup tables) de dimensiunea 16x16 bytes. Aceste tabele având valori constante, pot fi încărcate în memoria partajată a GPU-ului pentru a putea fi accesate de thread-uri. Pentru dimensiuni mici de criptare, CPU-ul va avea performanţe mai bune decât GPU-ul [KIP09]. Totuşi, în testele efectuate de [ROS07], s-a dovedit că pentru volume mari de criptare performanţele mai bune se obţin dacă se calculează valorile în loc să se efectueze căutări în tabele. Fiind accesate de toate thread-urile există posibilitatea încărcării tabelului în memoria globală de unde poate fi accesată de orice thread.

O altă soluţie ar fi folosirea memoriei constante pentru stocarea S-BOX-ului şi a cheilor de rundă. Avantajul în această alegere este că poate fi asignată încă din faza de design şi poate fi accesată şi de CPU [LUK09].

În [KIP09] s-a realizat şi o implementare paralelă a algoritmului AES pe CPU, dar trebuie specificat faptul că un CPU nu are nici pe departe posibilitatea rulării unui număr foarte mare de thread-uri asemenea unui GPU. De aceea, paralelizarea pe CPU nu va avea acelaşi rezultat ca şi pe GPU.

În dezvoltarea programului AES în C pentru CUDA vom urmări două direcţii: una este aceea în care vom încerca folosirea puterii de calcul oferită de GPU pentru toate operaţiile AES, iar cea de-a doua este aceea în care vom folosi tabele de căutare stocate în memorie. Deoarece se foloseşte CUDA, s-a putut opta pentru implementarea soluţiilor de optimizare a lui AES pe CPU, şi anume folosirea tabelelor de căutare. Geforce 8800 GT/G80, spre deosebire de predecesorii acestuia, este un procesor scalar şi nu mai este necesară combinarea instrucţiunilor în operaţii vectoriale pentru a obţine putere computaţională maximă. Totodată abilitatea lui G80 de a executa operaţii logice XOR pe 32 de biţi oferă teoretic un spor de performanţă acestei soluţii.

În figura 5.4.8 se prezintă modul de implementare şi testare al celor două soluţii.

În urma obţinerii rezultatelor se va vedea care din aceste două metode este mai rapidă pentru procesorul grafic ales.

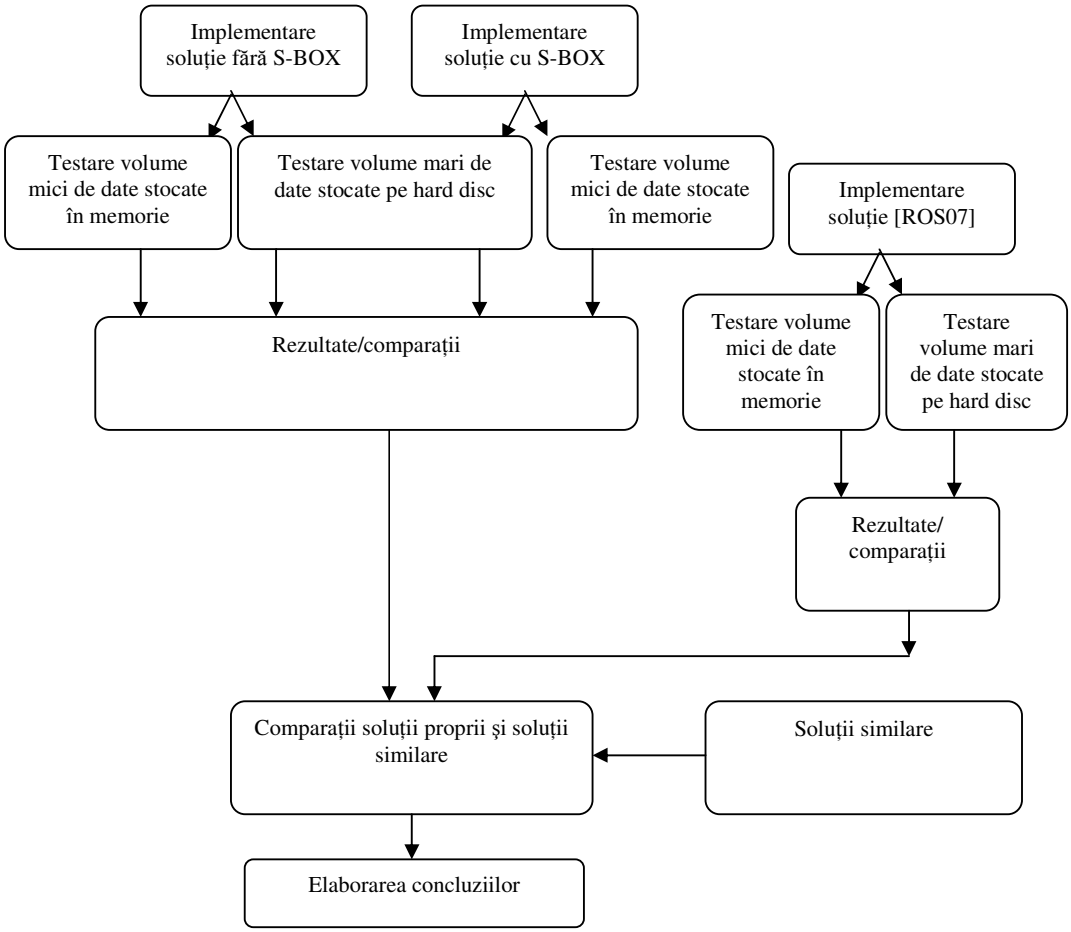

Figura 5.4.8 Implementarea şi testarea soluţiilor propuse

5.4.1. Complexitatea matematică AES

În algoritmul AES toţi bytes sunt interpretaţi ca elemente în câmp finit. Elementele în câmp finit pot fi adunate şi înmulţite, dar operaţiile diferă de cele folosite la numere [FIPS197].

Adunarea a două elemente în câmp finit se realizează prin adunarea modulo 2 (sau operaţia XOR- SAU EXCLUSIV) a coeficienţilor corespunzători puterilor polinoamelor celor două elemente. Adunarea poate fi considerată ca adunare modulo 2 a biţilor corespunzători din fiecare byte implicat în adunare. Adunarea în câmp finit se reprezintă astfel: $\oplus$.

Multiplicarea în câmp finit este reprezentată prin $\bullet$, iar în reprezentarea polinomială, corespunde cu multiplicarea a două polinoame modulo polinom ireductibil de grad 8. În cazul AES, polinomul ireductibil este $x^8+ x^4 +x^3 + x + 1$

Dacă avem două numere în hexa „57" şi „83", atunci „57" $\oplus$ „83"= „D4", cu alte cuvinte 01010111 $\oplus$ 10000011 =11010100, iar sub formă polinomială acest lucru se scrie $(x^6+x^4+x^2+x+1) \oplus (x^7+x+1)= x^7+x^6+x^4+x^2$[FIPS197]. Prin acest exemplu s-a descris operaţia de adunare modulo 2 (sau XOR) unde $1 \oplus 1=0$, $1 \oplus 0=1$ şi $0 \oplus 0=0$.

În cazul înmulţirii modulo f(x) pentru valorile din exemplul anterior „57" şi „83" vom avea „57" $\bullet$ „83"= „C1" deoarece

$(x^6+x^4+x^2+x+1) \bullet (x^7+x+1)= x^{13}+x^{11}+x^9+x^8+x^7+ x^7 + x^5 + x^3 + x^2 + x + x^6 + x^4 + x^2 +x+ 1 = x^{13}+x^{11}+x^9+x^8 + x^6 + x^5 + x^4 + x^3 + 1$ modulo $f(x)= x^7 + x^6 + 1$ [FIPS197].

Spre deosebire de operaţia de adunare, în cazul înmulţirii nu mai există operaţii simple la nivel de byte. Operaţia de înmulţire definită mai sus este asociativă, iar dacă avem un polinom b(x) de grad mai mic decât 8 atunci $b^{-1}(x)$ este inversul lui b(x). Dacă b(x)a(x) + f(x)c(x)=1 şi a(x) $\bullet$ b(x) mod f(x) =1, atunci putem scrie că $b^{-1}(x)=$ a(x)mod f(x) şi astfel a(x) $\bullet$(b(x)+c(x))=a(x) $\bullet$ b(x)+a(x) $\bullet$ c(x) [FIPS197].

Înmulţirea cu x este obţinută reducând rezultatul modulo $x^8+ x^4 +x^3 + x + 1$. Dacă polinomul este de grad maxim 7, atunci rezultatul multiplicării este deja în forma redusă.

Polinoamele cu patru termeni ce au coeficienţi în $GF(2^8)$ sunt diferite de cele prezentate anterior cu diferenţa că, aceste polinoame au coeficienţi elemente în câmp finit, de exemplu bytes în loc de biţi.

Pentru această situaţie, presupunem că avem două polinoame $a(x)= a_3x^3 + a_2x^2 + a_1x + a_0$ şi $b(x)= b_3x^3 + b_2x^2 + b_1x + b_0$ multiplicarea este realizată în doi paşi. În primul pas se obţine polinomul c(x) astfel: $c(x)= c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0$, unde[FIPS197]:

$c_0=a_0 \bullet b_0$

$c_1=a_1 \bullet b_0 \oplus a_0 \bullet b_1$

$c_2=a_2 \bullet b_0 \oplus a_1 \bullet b_1 \oplus a_0 \bullet b_2$

$c_3=a_3 \bullet b_0 \oplus a_2 \bullet b_1 \oplus a_1 \bullet b_2 \oplus a_0 \bullet b_3$

$c_4=a_3 \bullet b_1 \oplus a_2 \bullet b_2 \oplus a_1 \bullet b_1$

$c_5=a_3 \bullet b_2 \oplus a_2 \bullet b_3$

$c_6=a_3 \bullet b_3$

În pasul doi, rezultatul c(x) este redus modulo polinom de grad 4. În cazul AES acest polinom este x^4+1. Produsul modular al lui a(x) şi b(x) este d(x) şi notat prin $d(x)=a(x) \otimes b(x)$. $d(x)= d_3x^3 + d_2x^2 + d_1x + d_0$, unde[FIPS197]:

$d_0=(a_0 \bullet b_0) \oplus (a_3 \bullet b_1) \oplus (a_2 \bullet b_2) \oplus (a_1 \bullet b_3)$

$d_1=(a_1 \bullet b_0) \oplus (a_0 \bullet b_1) \oplus (a_3 \bullet b_2) \oplus (a_2 \bullet b_3)$

$d_2=(a_2 \bullet b_0) \oplus (a_1 \bullet b_1) \oplus (a_0 \bullet b_2) \oplus (a_3 \bullet b_3)$

$d_3=(a_3 \bullet b_0) \oplus (a_2 \bullet b_1) \oplus (a_1 \bullet b_2) \oplus (a_0 \bullet b_3)$

Aceste tipuri de operaţii alături de operaţii pe bytes de tipul xtime() sunt folosite ca operaţii interne în cazul algoritmului AES. După cum se poate vedea, acestea nu sunt operaţii complexe şi consumatoare de putere de calcul la fel cum sunt cele din algoritmii asimetrici.

5.4.2. Dezvoltarea programului AES in C pentru CUDA

După cum s-a scris mai sus, CUDA execută funcţii, numite kernel-uri, în thread-uri, care sunt organizate în blocuri. Blocurile sunt, la rândul lor, organizate în griduri. Toate thread-urile unui grid execută acelaşi kernel. Thread-urile dintr-un bloc se execută simultan şi pot comunica între ele [LUK09].

Pentru început, se definesc constantele care au strictă legătură cu modul de operare al AES pe GPU sub CUDA. Astfel, este necesar să definim numărul de blocuri şi numărul de thread-uri ce urmează să fie folosite în program. Numărul maxim de thread-uri / bloc pentru acest model de GPU şi în general este de 512 [CUD09-2]. Numărul de blocuri care se poate defini este recomandat să fie multiplu de 14; 14 este numărul de multiprocesoare pe care îl are placa video folosită în aceste teste. Fiecare multiprocesor are 8 procesoare [CUD09-2]. Pentru a cunoaşte ce bloc rulează în mod curent în grid-ul folosit şi ce thread rulează în mod curent în acel bloc se folosesc variabilele predefinite *blockIdx.x* şi *threadIdx.x*.

Pentru început se fac declaraţiile :

```
__device__ const unsigned char s_box[256];
```

Apoi se introduc funcţiile GPU

```
void _ShiftRows (uchar *state);
void _MixSubColumns (uchar *state);
void _AddRoundKey (unsigned *state, unsigned *key);
void _Encrypt (uchar *in, uchar *expkey, uchar *out);


extern "C" void gpu_init(int argc, char** argv);
extern "C" double gpu_encrypt_string(char* in, int in_len, char* key, char* out);
extern "C" void ExpandKey(uchar *key, uchar *expkey);
```

Se iniţializează vectorul CTR init ce se stochează în memoria partajată pentru un acces mai facil:

```
ctr_init_vector[1] = ctc_iv[7] << 24 | ctc_iv[6] << 16 | ctc_iv[5] << 8 | ctc_iv[4];
ctr_init_vector[0] = ctc_iv[3] << 24 | ctc_iv[2] << 16 | ctc_iv[1] << 8 | ctc_iv[0];
```

Calcularea blocului CTR folosind blocul de iniţializare şi *global block number* este prezentată mai jos. Apoi se criptează acest bloc şi se aplică XOR între rezultat şi datele din intrare:

```
calcCTR((unsigned int*)out, ctr_init_vector,
                    (gl_bl_no & 0x00000000ffffffffLL),
                    (gl_bl_no & 0xffffffff00000000LL) >> 32);
aes_encrypt(&context, out, out);
xor_block((unsigned long*)out, (unsigned long*)in, (unsigned long*)out);
gl_bl_no++;
```

Cheia pentru criptare este citită dintr-un fişier astfel:

```
int read_key (const char* filename, unsigned char* buffer, unsigned short length,
int* error)
{
  FILE* inputfile = NULL;
  int read_bytes = 0;
```

```
    if ((inputfile = fopen(filename, "r")) == NULL)
         { *error = errno;
            return -1;  }
    if ((read_bytes = fread(buffer, 1, length, inputfile)) < length)
         {   *error = read_bytes;
            return -1;  }
    return 0;
  }
```

După ce este citită este convertită şi salvată într-un buffer:

```
  for (int i=length*2-2; i>=0; i-=2)
  { char hexbyte[3] = {key[i], key[i+1], '\0'};
     int decimal = 0;
     if (sscanf(hexbyte, "%x", &decimal) != 1) {
       return -1;//error
     }
     buffer[length-1-i/2] = (unsigned char)decimal;
```

Astfel, cheia citită din fişier, în format hexadecimal este convertită în format decimal şi salvată într-un buffer pentru a putea fi folosită. Cheia, în format hexadecimal, salvată în fişier, este de 32 de bytes (128 biţi). Conversia este făcută de aşa natură încât primul byte să fie cel mai semnificativ.

Pentru testele în care folosim tabele de căutare prezentăm mai jos tabelul S-BOX [www8]

```
  // S-Box directă
  __device__ const unsigned char s_box[256] =
  {
     0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5,
     0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
     0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0,
     0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
     0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc,
     0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
     0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a,
     0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
     0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0,
     0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
     0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b,
     0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
     0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85,
     0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
     0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5,
     0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
     0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17,
     0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
     0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88,
     0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
     0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c,
     0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
     0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9,
     0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
     0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6,
     0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
```

```
    0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e,
    0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
    0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94,
    0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
    0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68,
    0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16
};
```

În cazul CPU-ului, o optimizare a lui AES constă în înlocuirea transformărilor SubBytes, MixColumns şi ShiftRows în tabele [STE06]. În cazul procesoarelor video există posibilitatea ca o căutare în tabel să fie mai lentă decât procesarea efectivă a operaţiilor aritmetice.

În etapa următoare se prezintă modul de aplicare a CTR unui bloc de intrare. Blocul de intrare şi cel de ieşire sunt de 16 bytes.

Măsurarea timpului se face prin o metodă clasică:

```
clock_t start = clock();
=>criptare AES CUDA;
clock_t end = clock();
printf("Criptarea s-a realizat în %d/%d secunde.\n", end-start, CLOCKS_PER_SEC);
```

Principalele probleme întâmpinate în dezvoltarea, compilarea, rularea şi testarea programelor au fost următoarele:

- Feedback-ul oferit de compilatorul CUDA nvcc este foarte slab şi eliminarea erorilor a fost foarte dificilă în cazul programelor .cu. De asemenea, debuging-ul a fost foarte dificil, deoarece erorile specificate de compilator nu erau deloc detaliate.
- În implementarea programelor CUDA a trebuit gândit că GPU este un coprocesor şi nu procesorul, astfel, faţă de programarea clasică sub C a trebuit schimbat punctul de vedere.
- În cazul alocării de prea multă memorie, programul se oprea din rulare fără a indica o eroare elocventă, aspect care a încetinit procesul de dezvoltare.

5.4.3. Makefile

Pentru compilarea programelor şi obţinerea aplicaţiei care poate fi rulată se foloseşte un script ce compilează întregul proiect pentru a putea fi rulat. În cele ce urmează, se prezintă modul de realizare a acestui script cunoscut sub numele de Makefile.

Pentru început se vor seta căile către folderele necesare:

- Calea de instalare a CUDA

```
CUDA_INC_DIR = $(CUDA_DIR)/cuda/include
CUDA_LIB_DIR = $(CUDA_DIR)/cuda/lib
CUDA_COMMON_LIB_DIR = $(CUDA_DIR)/common/lib/linux
CUDA_SDK_INC_DIR = $(CUDA_DIR)/common/inc
```

- Bibliotecile CUDA şi alte opţiuni

```
DYNAMIC_LIBS = -lcudart -lm -lrt -lcutil
STATIC_LIBS =
INCS = -I$(CUDA_INC_DIR) -I$(CUDA_SDK_INC_DIR)
LIBS = -L$(CUDA_LIB_DIR) -L$(CUDA_COMMON_LIB_DIR) $(DYNAMIC_LIBS)
```

- Compilatorul CUDA şi parametri ce vor fi transmişi compilatorului

```
CUDA_DIR      = /opt/cuda-3.2
CUDA_COMPILER = $(CUDA_DIR)/cuda/bin/nvcc
GPU_COMP_FL = -O2 --compiler-options -fno-strict-aliasing -use_fast_math
```

- Compilatorul C/C++ şi parametri ce vor fi transmişi compilatorului

```
CC= gcc
CPP = g++
CPP_FL = -O2 -Wall -fno-strict-aliasing
CC_FL= -O2 -Wall -fno-strict-aliasing
LINKER_FLAGS      = -fPIC
```

Se vor folosi reguli explicite pentru a obţine crearea unei ţinte, sau pentru a o recrea. În cadrul regulilor explicite folosim linii de dependenţă precum :*%.o : %.c* sau *%.o : %.cu*. Prin „:" determinăm ca la fiecare rulare a *Makefile* să fie verificate toate fişierele cu extensia .o care depind de fişierele cu extensia .c sau .cu, iar de câte ori acestea se modifică se vor compila din nou în fişiere obiect.

În *Makefile* se poate folosi cod din shell, ca de exemplu echo, pentru afişarea unui mesaj.

Vom folosi variabila TINTA în care vom păstra numele fişierului executabil obţinut în urma rulării makefile:

```
TINTA = gpuaes
```

Alte variabile utile sunt $< şi $@. Prima reprezintă „indiferent de dependenţe" iar cea de-a doua reprezintă ţinta.

Se selectează folderul sursă ca fiind folderul rădacină curent:

```
SRC_DIRS := .
```

Se selectează fişierele sursă după regula următoare:

```
SRC_FILES := $(foreach DIR, $(SRC_DIRS), $(wildcard $(DIR)/*.c))
```

Se setează fişierele obiect după regula următoare:

```
OBJS := $(patsubst %.c, %.o, $(SRC_FILES))
```

Se selectează folderul sursă pentru fişierele CUDA ca fiind folderul rădăcină curent:

```
CUDA_SRC_DIRS := .
```

Se selectează fişierele sursă CUDA după regula următoare:

```
CUDA_SRC_FILES := $(foreach DIR, $(CUDA_SRC_DIRS), $(wildcard $(DIR)/*.cu))
```

Se setează fişierele obiect CUDA după regula următoare:

```
CUDA_OBJS := $(patsubst %.cu, %.o, $(CUDA_SRC_FILES))

all : $(TINTA)
	@ finalizare cu succes
%.o : %.c
	$(C_COMPILER) $(GCC_COMPILER_FLAGS) -o $@ -c $<
%.o : %.cu
	$(CUDA_COMPILER) $(GPU_COMP_FL) -o $@ -c $(INCS) $<
$(TINTA) : $(OBJS) $(CUDA_OBJS)
	$(CPP) $(GCC_COMPILER_FLAGS) $(LINKER_FLAGS) -o $@ $(LIBS) $^ $(STATIC_LIBS)
	cp $(TINTA) precompiled/

Compilarea surselor CUDA:
nvcc-link : $(OBJS) $(CUDA_OBJS)
	$(CUDA_COMPILER) -o $(TINTA) $(LIBS) $^ $(STATIC_LIBS)
```

```
	cp $(TINTA) precompiled/
Rularea aplicaţiei
prog: .runprog
	./$(TINTA) -e -K test/keyfile.dat -i test/plaintext.txt -o test/encrypted.dat
```

5.4.4. Rezultate obţinute

În urma rulării soluţiilor prezentate în subcapitolele anterioare s-au obţinut rezultatele orientate în două direcţii: cazul în care s-au folosit tabele lookup şi cazul în care s-au efectuat calculele.

Dimensiunile de intrare au fost aceleaşi ca şi în cazul testelor efectuate în capitolul 3.

Astfel, testele s-au efectuat în 2 etape: criptarea datelor din memorie şi criptarea datelor de dimensiuni mari stocate pe hard disc. La fiecare etapă s-au testat 3 tipuri de algoritmi: cel implementat în [ROS07], algoritmul în care AES foloseşte tabele de căutare stocate în memoria globală şi algoritmul în care AES foloseşte valori calculate fără a mai efectua căutări în tabele.

5.4.4.1. Etapa 1. Criptarea datelor aflate în memorie

Dezvoltarea aplicaţiei a fost gândită în mod similar celor descrise în capitolul 3.

Pentru început, se generează o valoare aleatoare ce va fi folosită pentru criptare. Codul este asemănător celui descris în 3.4.1.

Dimensiunile de intrare au fost aceleaşi ca şi în cazul testelor efectuate în capitolul 3. Practic, pentru început, se generează o valoare aleatorie care va fi folosită pentru criptare. Această valoare este de dimensiune 128 de biţi, exact cât este şi blocul AES.

În această secţiune se testează algoritmul AES pentru date aflate în memoria GPU-ului fiind simulată criptarea datelor „on the fly”. Se rulează algoritmii „în gol” folosind recursivitatea astfel: *buffer = algoritm_criptare(buffer+ valoare aleatoare)*. Numărul de iteraţii ales a fost 1.000.000 ca şi în cazul algoritmilor testaţi în capitolul 3.4.

Se măsoară timpul necesar de rulare, iar apoi se împarte la numărul de iteraţii, obţinându-se în acest mod timpul mediu de rulare al algoritmului pentru o iteraţie.

Rezultatele obţinute au fost comparate cu rezultatul obţinut de algoritmul dezvoltat în [ROS07]. Astfel, în tabelul 5.4.4.1.1 avem trei valori. Una este cea obţinută în urma rulării algoritmului implementat cu tabele de căutare (SBOX), a doua este cea obţinută prin calculul efectiv al operaţiilor pentru etapa SubBytes nefolosind tabele de căutare la fel ca şi în [www7]. În prima coloană este trecută valoarea obţinută rulând algoritmul descris în [ROS07].

Pentru algoritmul care foloseşte tabele S-BOX am obţinut rezulatul 0,00116233 ms. Această valoare nu cuprinde timpul de generare a valorii aleatorii, ci doar datele care sunt stocate în memoria partajată şi criptate, depuse înapoi şi apoi recriptate de 1.000.000 de ori. Timpul de criptare obţinut nu este afectat de aducerea datelor din memoria CPU-ului, deoarece sunt stocate şi preluate direct din memoria GPU-ului. Ţinând cont că absolut toate thread-urile au nevoie să acceseze S-BOX-ul, acesta a fost stocat în memoria globală pentru a putea fi disponibil pentru orice thread.

În cazul algoritmului ce nu foloseşte S-BOX, timpul obţinut este de 0,00121234 ms şi este mai mare decât cel care foloseşte SBOX. La fel ca şi în cazul algoritmului

anterior datele sunt preluate din memoria partajată (GPU) şi depuse tot acolo pentru a fi preluate de următoarea iteraţie.

Comparând cele două rezultate obţinute, pentru datele din memorie de mici dimensiuni se poate trage concluzia că, la fel ca în cazul optimizării AES pentru CPU, cel mai rapid AES este cel care foloseşte valorile S-BOX stocate în memorie.

Comparând-ul cu timpul cel mai bun obţinut pe CPU care a fost tot pe platforma aceasta, care are în dotare placa video folosită în această etapă, putem spune că timpul obţinut de GPU este mai bun. Pe CPU, cel mai bun timp a fost obţinut în Java şi acesta a fost de 0,156324 ms. Sporul de performanţă în acest caz a fost de cca. 134 de ori. Practic AES pe GPU a fost cu 0,155162 ms mai rapid decât în cazul procesorului. În cazul AES fără SBOX sporul de performanţă în acest caz a fost de cca. 129 de ori. În tabelul 5.4.4.1.2 se poate observa că în cazul platformei 3 de test, pentru testele efectuate pe CPU timpii obţinuţi în Visual Basic şi C# sunt şi mai mici decât timpul obţinut în Java pe platforma PC1.

GPU Nvidia 8800		
[ROS07]	**AES cu SBOX**	**AES fără SBOX**
0,00118259	0,00116233	0,00121234

Tabel 5.4.4.1.1 Comparaţie rezultate date aflate în memorie(ms).

PC1		
VB	**C#**	**Java**
0,16534	0,15681	0,15632
PC2		
VB	**C#**	**Java**
0,2118	0,2118	0,40544
PC3		
VB	**C#**	**Java**
0,03266	0,03186	0,20374

Tabel 5.4.4.1.2 Valori obţinute CPU date aflate în memorie(ms).

Analizând cele trei rezultate obţinute în tabelul de mai sus putem spune că una din implementări este mai rapidă decât cea descrisă în [ROS07], iar cealaltă este mai lentă. Timpii nu sunt foarte diferiţi şi sporul de performanţă dintre algoritmi se poate observa cel mai bine în figura 5.4.4.1.1.

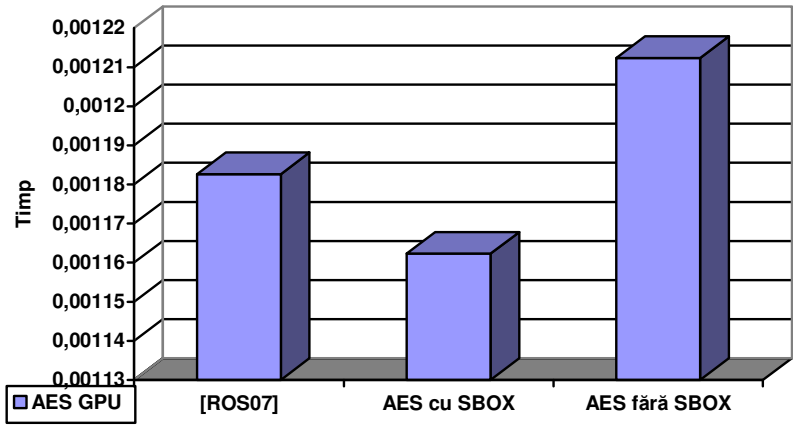

Figura 5.4.4.1.1 Criptarea datelor aflate în memorie

Preluând rezultatele din capitolul 4.5 pentru această platformă şi analizând figura 5.4.4.1.2, putem concluziona că adaptarea AES folosind CTR pentru a benefica de paralelizare este mult mai rapidă decât varianta standard AES, ce foloseşte bibliotecile limbajelor de programare (VB,C#şi Java) pentru CPU. Din rezultate s-a tras concluzia că AES CUDA este de până la 134 de ori mai rapidă decât AES CPU(JAVA). Desigur, Java având cel mai bun rezultat, este de înţeles că algoritmii VB şi C# sunt şi mai lenţi decât Java, iar AES CUDA este mai rapid cu peste 134 de ori.

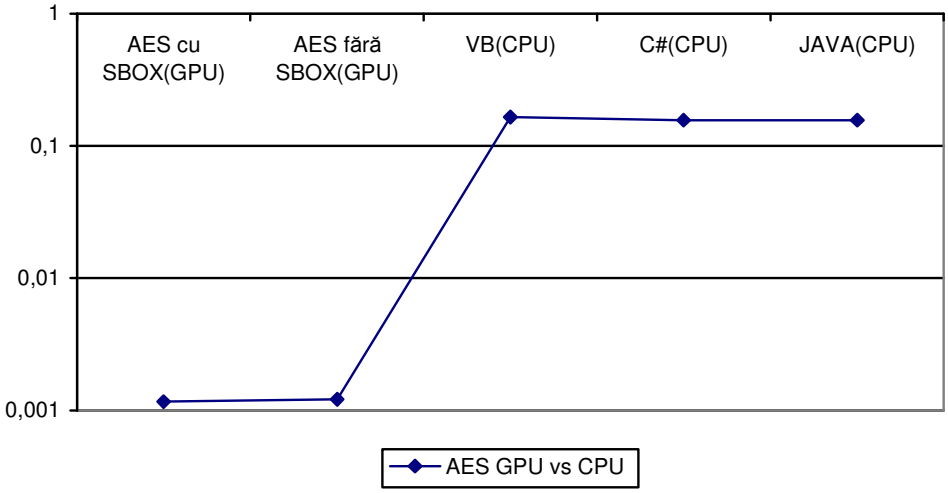

Figura 5.4.4.1.2 Comparaţie GPU – CPU

Rezultatele obţinute în această etapă au dus la concluzia că AES cu S-BOX obţine o performanţă de 13.12 Mbps iar AES fără S-BOX obţine o performanţă de 12.59

Mbps. Aceste performanţe sunt slabe în comparaţie cu cele obţinute de alţi autori în lucrări similare. În [TAK07] autorul obţine performanţe de cca. 10-20 Mbps pentru valori de 4 KB şi de 20-35 pentru valori de 16 KB. Explicaţia pentru aceste valori slabe este că valoarea de intrare este de 16 KB, la fel de mare ca dimensiunea blocului AES (128 de biţi) şi cum numărul de blocuri criptate este 1, în acest caz performanţa este afectată de overhead-ul (suprasarcină) necesar pornirii kernelului. În urma acestor rezultate, a apărut necesitatea alegerii unei valori de intrare mai mari. S-a ales ca dimensiune de intrare o valoare de 10 MB. În acest caz timpii obţinuţi sunt prezentaţi în tabelul 5.4.4.1.3.

AES cu SBOX	AES fără SBOX
914,2342	817,20711

Tabel 5.4.4.1.2 Valori obţinute GPU date aflate în memorie 10 MB(ms).

Aceste două rezultate prezintă un salt de performanţă fantastic faţă de dimensiunea de 16 KB. Rezultatele obţinute în această etapă au dus la concluzia că AES cu S-BOX obţine o performanţă de 10,68 Gbps iar AES fără S-BOX este mai rapid având o performanţă de 11,95 Gbps. [ROS07] a obţinut timpul de 1832,34 ms având o performanţă de 5,3 Gbps. Comparând aceste rezultate cu rezultatul obţinut în urma rulării algoritmului [ROS07] putem spune că sporul de performanţă este notabil. Nu se poate face comparaţia cu CPU, deoarece în cadrul testelor efectuate pe valori din RAM nu s-a criptat volum de 10 MB din memorie. Comparaţia în acest caz se face doar pentru valorile de 16KB.

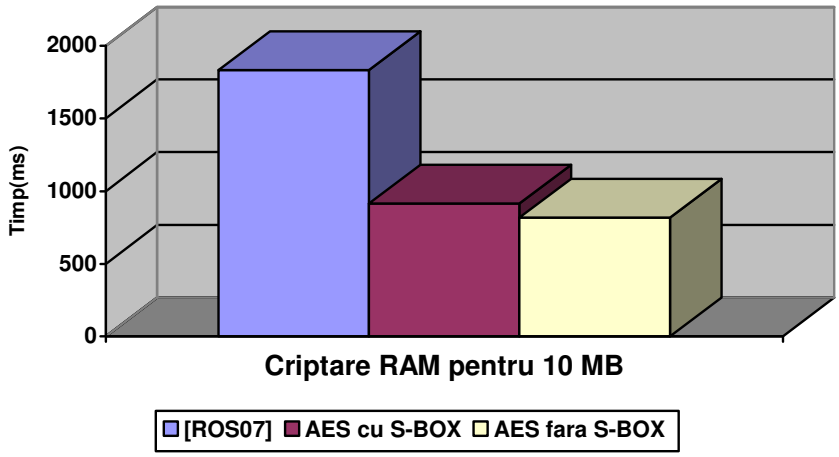

Figura 5.4.4.1.2 Comparaţie GPU – CPU pentru 10 MB

5.4.4.2. Etapa 2. Criptarea datelor stocate pe hard disc

În etapa a doua, testele s-au efectuat pentru date de mari dimensiuni. În mod similar celor din capitolul 4 au fost criptate fişiere de dimensiunea 1GB, 2 GB...10 GB. Într-o primă fază au fost testate trei fişiere cu dimensiunea de 100 MB, 1 GB şi 10 GB. Testând şi fişierul cu dimensiunea de 5 Gb s-a constatat că rezultatele obţinute erau în contradicţie cu cele obţinute pe testele 100 MB, 1 Gb şi 10 GB. Atunci s-a luat hotărârea de a efectua testele pe întreaga plajă 1GB, 2 GB...10 GB.

Rezultatele acestor teste sunt prezentate în tabelul 5.4.4.2.1 şi sunt exprimate în secunde. Şi în această etapă s-a realizat o comparaţie între valorile obţinute prin rularea algoritmilor pe GPU şi cei pe CPU din capitolul 4.

Legat de valorile obţinute pentru cele două implementări (cu SBOX şi fără SBOX), se poate spune că trendul acestora nu este unul crescător uniform. Pentru fişierele de până în 3 GB varianta AES cu SBOX are timpi mai buni, urmând ca de la 4 GB până la 8 GB AES fără SBOX să aibă timpi mai buni, iar pe final pentru fişierele de 9GB şi 10GB AES cu SBOX să obţină din nou rezultate mai bune. Luând în considerare şi rezultatele obţinute în capitolul 5.4.4.1, am putea afirma că AES cu SBOX este mai rapid pentru dimensiuni mai mici de 3 GB, iar pentru valori de peste 3 GB cel mai performant este AES fără SBOX. În contradicţie cu această afirmaţie vin rezultatele obţinute pentru fişiere mai mari de 9 Gb unde situaţia a fost răsturnată. Ca factori care puteau să intervină în această situaţie, putem spune că sistemul de operare UNIX are probleme de manipulare a fişierelor de mari dimensiuni stocate pe partiţie NTFS. Acelaşi lucru poate fi remarcat şi în figura 4.4.15, unde, în cazul OpenSSL sub UNIX, s-a observat o creştere bruscă de timp la fişierele de 9 GB şi 10 GB.

Analizând rezultatele din cele două tabele cu valorile de mai jos putem încerca să facem o diferenţiere de performanţă între rezultatele obţinute pe CPU şi cele obţinute pe GPU. Fiind multe rezultate obţinute pe mai multe platforme şi nerespectând o liniaritate ideală am ales să calculăm sporul de performanţă în două etape. Am ales din tabelul 5.4.4.2.1 şi din tabelul 5.4.4.2.2 timpul cel mai bun obţinut dintre cele trei variante testate pentru fiecare dimensiune de fişier din plajă. Această analiză este reflectată în tabelul 5.4.4.2.3. Se poate observa că cel mai bun timp obţinut de GPU comparat cu cel mai bun timp obţinut de CPU are un raport de cca. 17 (valoarea cea mai mare). Această situaţie se întâlneşte pentru fişier de 100Mb, unde GPU-ul este de 17 ori mai rapid decât CPU-ul. În altă ordine de idei, dacă luăm cel mai slab rezultat obţinut de CPU şi îl comparăm cu cel mai bun rezultat obţinut de GPU, atunci putem spune că GPU-ul este de cca. 40 de ori mai rapid decât CPU-ul. În afară de aceste două valori care se obţin pentru fişierul de 100 MB, raportul de performanţe nu a mai depăşit decât într-un singur caz valoarea de 10. Pentru situaţia când au fost comparaţi cei mai buni timpi obţinuţi de GPU şi cei mai buni timpi obţinuţi de CPU pentru fişierele de peste 1 GB dimensiune, s-a tras concluzia că cel mai bun raport între GPU şi CPU este de cca. 5.5. În situaţia în care se compară cei mai buni timpi obţinuţi de GPU şi cei mai slabi timpi obţinuţi de CPU pentru fişierele de peste 1 GB se ajunge la concluzia că cel mai bun raport între GPU şi CPU este de cca. 10.6.

Algoritm / Fişier	[ROS07]	AES cu SBOX	AES fără SBOX
100 MB	0,517	0,1435	0,835
1 GB	6,015	8,3556	10,344
2 GB	45,514	35,436	40,653
3 GB	72,048	50,660	51,588
4 GB	80,0260	59,3220	55,1460
5GB	85,5820	75,8590	70,5650
6 GB	92,7550	88,7670	81,6450
7 GB	102,3660	93,6430	89,8940
8 GB	110,725	102,276	101,863
9 GB	119,056	109,236	112,034
10 GB	125,646	120,434	121,470

Tabel 5.4.4.2.1 Rezultate timpi de computaţie algoritmi GPU.

Algoritm / Fişier	VB	C#	OpenSSL
100 MB	4,867000	2,437500	5,835000
1 GB	50,148000	23,734375	64,344000
2 GB	69,854000	72,687500	128,653000
3 GB	217,921500	155,421875	194,588000
4 GB	263,921500	196,703125	265,146000
5GB	312,000000	255,609375	326,565000
6 GB	474,264000	679,140625	408,645000
7 GB	435,609000	732,487500	507,894000
8 GB	578,796500	589,921875	529,863000
9 GB	606,439000	789,625000	602,034000
10 GB	615,750000	812,812500	656,470000

Tabel 5.4.4.2.2 Rezultate timpi de computaţie algoritmi CPU.

	GPU best vs. CPU worst	GPU best vs CPU Best
100 MB	40,662021	16,986063
1 GB	10,697257	3,945865
2 GB	3,630573	1,971272
3 GB	4,301648	3,067941
4 GB	4,808073	3,566952
5GB	4,627861	3,622325
6 GB	8,318215	5,005144
7 GB	8,148347	4,845807
8 GB	5,791326	5,201722
9 GB	7,228615	5,511315
10 GB	6,749029	5,112759

Tabel 5.4.4.2.3 Comparare CPU VS GPU. Spor de performanţă

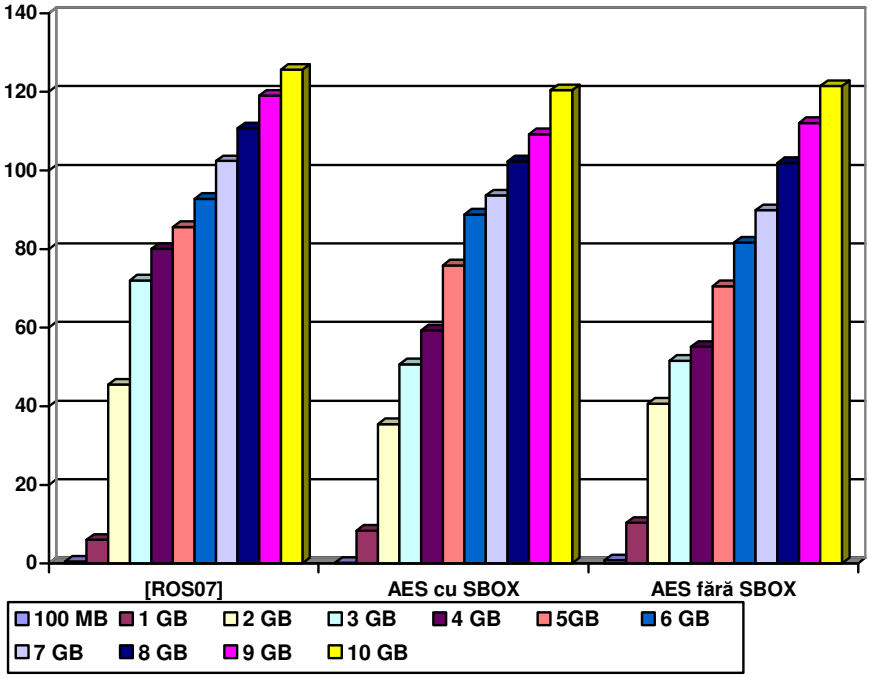

Figura 5.4.4.2.1 Criptarea datelor stocate pe HDD[TOM11]

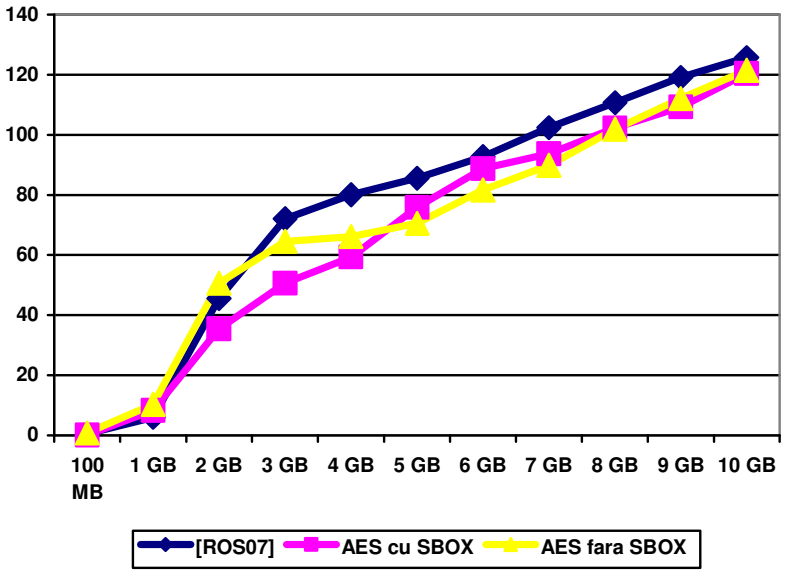

Figura 5.4.4.2.2 Criptarea datelor stocate pe HDD[TOM11]

	AES cu SBOX	AES fără SBOX
100 MB	69,68641	11,97605
1 GB	122,5525	98,99459
2 GB	57,79433	50,37759
3 GB	60,63956	59,54873
4 GB	69,0469	74,27556
5GB	67,49364	72,55722
6 GB	69,21491	75,25262
7 GB	76,54603	79,73836
8 GB	80,09699	80,42174
9 GB	84,36779	82,26074
10 GB	85,02582	84,30065

Tabel 5.4.4.2.4 Performanţe GPU. Mbps

În tabelul 5.4.4.2.4, se poate observa că AES cu S-BOX are cea mai bună performanţă în cazul fişierului de 1 GB. Aceasta este de 122 Mbps. Tot pentru fişierul de 1 GB şi AES fără GPU are performanţa foarte bună fiind a doua din cele obţinute pentru toate fişierele testate, şi-anume de cca. 99 Mbps. La fişierul de 2 GB performanţele celor doi algoritmi scad şi pentru celelalte fişiere umrează un trend ascendent din punctul de vedere al performanţei. În tabelul 5.4.4.2.5 se disting performanţele CPU-ului din toată gama de teste efectuate pentru criptarea fişierelor stocate pe HDD care nu ating performanţele obţinute de GPU în testele echivalente.

Algoritm / Fişier	VB	C#	OpenSSL
100 MB	2,054654	4,102564	1,713796
1 GB	20,41956	43,14417	15,91446
2 GB	29,31829	28,17541	15,91879
3 GB	14,09682	19,76556	15,7872
4 GB	15,51977	20,82326	15,44809
5GB	16,41026	20,03056	15,67835
6 GB	12,95481	9,046727	15,03505
7 GB	16,45512	9,785833	14,11318
8 GB	14,15351	13,88658	15,4606
9 GB	15,19691	11,67136	15,30811
10 GB	16,63013	12,59823	15,59858

Tabel 5.4.4.2.5 Performanţe CPU. Mbps

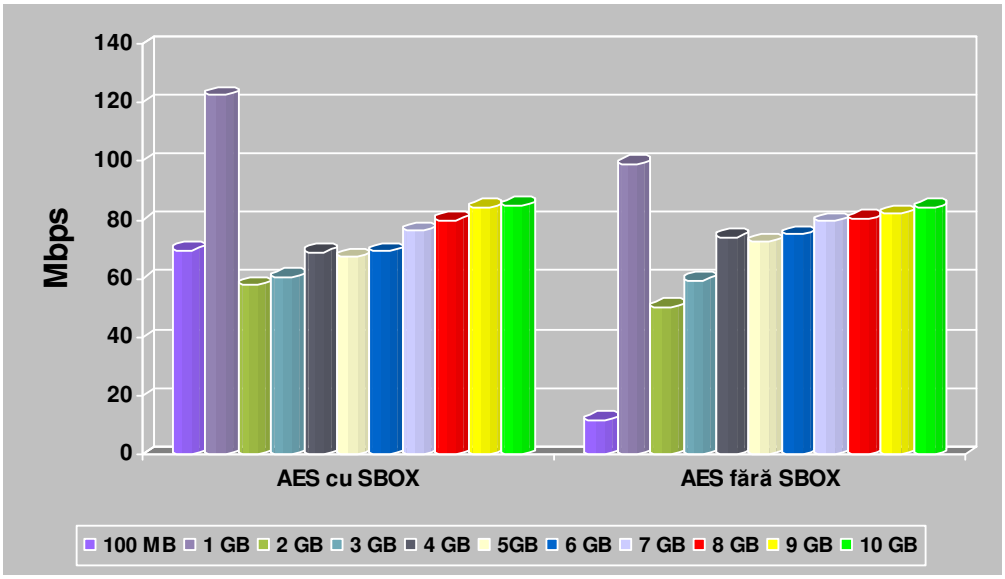

Figura 5.4.4.2.3 Criptarea datelor stocate pe HDD

5.5. OpenSLL

În urma finalizării adaptării algoritmului AES pentru GPU s-a încercat folosirea acestuia într-o aplicaţie de uz general precum OpenSSL.

Pentru a putea folosi algoritmul AES ce rulează pe GPU în OpenSSL sunt necesare câteva modificări în kernel-ul OpenSSL.

Nucleul OpenSSL este scris in limbajul C. Acesta oferă nenumărate funcţii criptografice bazate pe primitive criptografice, iar pe lângă acestea mai dispune şi de nenumărate funcţii auxiliare. Proiectul OpenSSL este creat, dezvoltat şi actualizat de comunitatea online OpenSSL. Aceştia oferă toolkit-ul OpenSSL şi documentaţia sub formă de licenţă Open Source [OSP99]. OpenSSL se bazează şi este derivat din bliblioteca SSLeay dezvoltată de Eric Young şi Tim Hudson.

OpenSSL oferă posibilitatea utilizatorului să folosească accelerator hardware dedicat [ROS07]. Acest lucru se realizează prin folosirea de motoare criptografice care comunică hardware-ului special să cripteze sau să decripteze. Este necesar un set minim de funcţii declarate într-un astfel de motor. Una dintre acestea este funcţia care creează instanţa noului motor criptografic. A doua este cea care setează funcţiile id, nume, iniţializare şi terminare. A treia este cea care returnează lista cu cifrurile suportate[ROS07].

Limbajul C oferă următoarele facilităţi [CRI09]:

- Poate crea mai multe fire de execuţie ce sunt rulate simultan (în cazul GPU)- este un limbaj concurent
- Codul obiect generat în urma compilării este mai eficient, decât în cazul limbajelor de nivel înalt

În urma implementării extensiei OpenSSL care să ofere suport AES pe CUDA trebuie să putem rula o comanda de tipul:

openssl aes-128-ctr-gpu –in numefisierin –out numefisierout –k fişiercheie

unde *aes-128-ctr-gpu* specifică faptul că se foloseşte pentru criptare aes pe 128 de biţi cu modul CTR şi va rula pe GPU,

–in numefisierin –out numefisierout specifică fişierul de intrare şi fişierul de ieşire,

–k fişiercheie specifică exact fişierul din care este citită cheia ce se va folosi.

Deoarece în capitolul 5.4 a fost ales modul CTR, la realizarea acestei extensii se foloseşte acelaşi mod. Pentru paralelizare, aşa cum a fost menţionat şi în capitolul 5.4, se mai putea folosi şi modul ECB, dar care nu oferă o siguranţă sporită în criptarea AES. Spre deosebire de implementarea [ROS07], unde a fost folosit modul ECB, în implementarea noastră s-a folosit CTR. Principalul avantaj al acestui mod este că oferă o securitate sporită. Asemănarea dintre cele două este că ambele pot fi folosite pentru paralelizare.

Pentru ca OpenSSL să recunoască un nou motor, acesta trebuie să fie încărcat de *ENGINE_load_builtin_engines* folosind *ENGINE_load_gpu()*. Asemănător soluţiei [ROS07], se doreşte încărcarea engine-ului automat de către OpenSSL fără a se urmări încărcarea explicită de către aplicaţia ce foloseşte OpenSSL.

Fişierul *aes-128-ctr-gpu.cu,* ca de altfel toate fişierele ce au cod în C pentru CUDA, vor fi compilate cu nvcc.

La fel ca şi în cazul programului dedicat dezvoltat în capitolul 5.4, se foloseşte aceeaşi structură şi pentru extensia OpenSSL.

După cum s-a menţionat în capitolul 5.4, CUDA execută funcţii, numite kernel-uri, în thread-uri, care sunt organizate în blocuri. Blocurile sunt, la rândul lor, organizate în griduri. Toate thread-urile unui grid execută acelaşi kernel. Thread-urile dintr-un bloc se execută simultan şi pot comunica între ele[LUK09]. Această extensie este, în

mare parte, asemănătoare cu cea dezvoltată în 5.4. Pentru început, se definesc constantele care au strictă legătură cu modul de operare al AES pe GPU sub CUDA. Astfel, este necesar să definim numărul de blocuri şi numărul de thread-uri ce urmează să fie folosite în program. Numărul maxim de thread-uri / bloc pentru acest model de GPU, şi în general, este de 512[CUD09-2]. Numărul de blocuri care se poate defini este recomandat să fie multiplu de 14. 14 este numărul de multiprocesoare pe care îl are placa video folosită în aceste teste. Fiecare multiprocesor are 8 procesoare[CUD09-2]. Pentru a cunoaşte ce bloc rulează în mod curent în grid-ul folosit şi ce thread rulează în mod curent în acel bloc se folosesc variabilele predefinite *blockIdx.x* şi *threadIdx.x*.

Ţinând cont de rezultatele obţinute în testele efectuate anterior, s-a ales varianta AES cu tabele lookup, considerându-se că OpenSSL foloseşte, în cea mai mare parte, volume mai mici de date care necesită a fi criptate.

Trebuie menţionat că, din această etapă lipseşte partea de decriptare a datelor, aceasta nefiind necesară, deoarece acelaşi cod sursă folosit la criptare este folosit şi la decriptare.

5.6. Concluzii

În cadrul acestui capitol s-a descris, într-o primă etapă, platforma folosită pentru testele propuse.

S-a prezentat stadiul actual fiind evidenţiate lucrări similare şi s-a punctat necesitatea unei noi implementări care să analizeze două posibilităţi de dezvoltare a lui AES: cu tabele de căutare şi fără tabele de căutare.

S-a evidenţiat modul de adaptare AES pentru C sub CUDA pentru a rula pe GPU cu modificările necesare şi modul de compilare şi rulare al acestuia folosind Makefile.

Tot în acest capitol s-a discutat despre complexitatea computaţională a algoritmului AES prezentându-se operaţiile care stau la baza acestui algoritm pentru a se înţelege mai uşor dificultatea implementării unui algoritm care nu foloseşte operaţii complexe pe un procesor ce a fost conceput să opereze calcule complexe.

S-au realizat două implementări care au fost testate în aceleaşi condiţii cu cele din capitolul 4. S-au prezentat tehnicile folosite, algoritmii, şi particularităţile fiecărei metode alese şi adaptate. Testele au cuprins atât date din memorie cât şi date stocate pe HDD.

S-a prezentat modul în care se poate instala mediul CUDA sub Kubuntu.

În urma rulării testelor au fost obţinute rezultatele ce au fost discutate şi explicate. S-a obţinut o performanţă maximă de 11.95 Gbps pentru algoritmul AES fără S-BOX în cazul în care au fost folosite ca date de intrare valori stocate în memorie cu dimensiunea de 10 MB. Acest rezultat atestă faptul că implementarea folosită este mai performantă decât lucrările similare ale altor autori ce foloseau GPU pentru criptarea simetrică precum [YON08], [MAN07], [TAK07], [LEE10], [BIA09], [HOD05], [BRO05].

În această etapă a cercetării, s-a realizat o adaptare a algoritmului AES pentru procesorul grafic. AES a beneficiat de paralelizare masivă în comparaţie cu posibilităţile de paralelizare oferite de un CPU în acest moment. Testele efectuate au demonstrat că puterea mare de procesare şi lăţimea de bandă a memoriei GPU-ului constituie avantaje în alegerea acestuia ca şi coprocesor criptografic. Diferenţa mare de performanţă a fost obţinută deoarece procesoarele clasice sunt optimizate pentru rularea serială a proceselor şi folosirea de cache de mari dimensiuni şi seturi de instrucţiuni complexe. Pentru a obţine performanţă sporită în comparaţie cu CPU a

fost necesară asigurarea utilizării a tuturor nucleelor GPU-ului prin rularea de instrucţiuni.

Rezultatele obţinute au demonstrat că implementarea pe un procesor video a unui algoritm criptografic aduce un spor de performanţă în comparaţie cu rezultatele obţinute pe procesoarele clasice de pe platformele testate. Sporul de performanţă a fost de 134 de ori mai mare decât pentru procesor, în cazul datelor stocate în memorie, şi până la 17 ori mai mare decât procesorul, în cazul datelor de mari dimensiuni. În testele din urmă, un factor care a afectat performanţa, a fost timpul necesar aducerii datelor de pe hard disc, acestea fiind foarte mari. Performanţa AES pe CUDA a fost cuprinsă între 12 Mbps şi 11,95 Gbps[TOM11].

Analiza efectuată între cele două soluţii alese a fi implementate în această lucrare a dus la următoarele concluzii: implementarea AES cu tabele lookup, în comparaţie cu implementarea AES fără tabele lookup, este mai rapidă pentru datele stocate în memorie şi pentru volumele mici de date stocate pe hard disc [TOM10/5]. Implementarea AES pe CUDA fără tabele lookup s-a dovedit a fi mai rapidă în cazul fişierelor de mari dimensiuni şi a atins performanţe uluitoare în cazul datelor din memorie cu volumul de 10MB. Doar în cazul ultimelor două fişiere cu dimensiuni de 9, respectiv 10 GB această implementare s-a dovedit a fi mai lentă decât prima soluţie, la fel ca şi în cazul figurii 4.4.15 în care AES OpenSSL pentru fişierul de 9 şi 10 GB a avut un timp mult mai mare, creşterea fiind bruscă faţă de celelalte variante testate. Astfel, putem spune că pentru valorile de mari dimensiuni, AES fără tabele lookup este mai rapid decât în cazul în care AES foloseşte tabele lookup. Putem trage concluzia că GPU oferă performanţe mai bune decât CPU dacă nu se foloseşte optimizarea cu tabele lookup. Justificarea este că GPU-ul oferă putere de computaţie mai mare decât viteza de acces la memorie. Explicaţia pentru valorile anormale din figura 4.4.15 şi pentru inversarea situaţiei în cazul AES cu SBOX şi AES fără SBOX poate consta în faptul că fişierele de mari dimensiuni au fost stocate pe o partiţie NTFS şi sistemul Kubuntu 9.04, la fel ca şi în cazul testelor din capitolul 3, a întâmpinat probleme în manevrarea fişierelor de peste 9 Gb[TOM10/6].

În comparaţie cu implementarea efectuată în [ROS07], unde s-a realizat tot o implementare AES pe CUDA, putem spune că doar pentru fişierul de 1 GB a obţinut performanţe mai bune decât implementările realizate în această lucrare. În cazul datelor din memorie, implementarea menţionată se poziţionează ca şi performanţe între cele două realizate. Ca atare, este mai slabă decât AES cu tabele şi mai puternică decât AES fără tabele [TOM10/6]. Pentru datele din memorie de 10 MB această implementare este mai slabă decât cele propuse în această lucrare. În cazul fişierului de 100 MB se repetă situaţia fără a se vedea o modificare[TOM11]. Pentru fişierul de 1 GB, implementarea lui [ROS07] este mai rapidă decât ambele AES (cu şi fără S-BOX). De la fişierul de 2 GB, aceasta a pierdut în performanţă în comparaţie cu cele realizate în această lucrare.

În [TOM10/5] s-a realizat o sinteză a implementărilor realizate pentru volume mici de date stocate în memorie, despre rezultatele obţinute şi despre problemele întâmpinate în diferite etape. În [TOM11] s-au prezentat rezultatele finale obţinute pentru toate implementările.

În figura 5.6.1 se poate observa comparaţia dintre soluţiile propuse şi implementate în această lucrare şi implementările realizate în lucrări similare.

În finalul capitolului s-a realizat o implementare a AES pe GPU sub OpenSSL. Astfel se poate folosi algoritmul sub OpenSSL, beneficiind de accelerarea procesorului video.

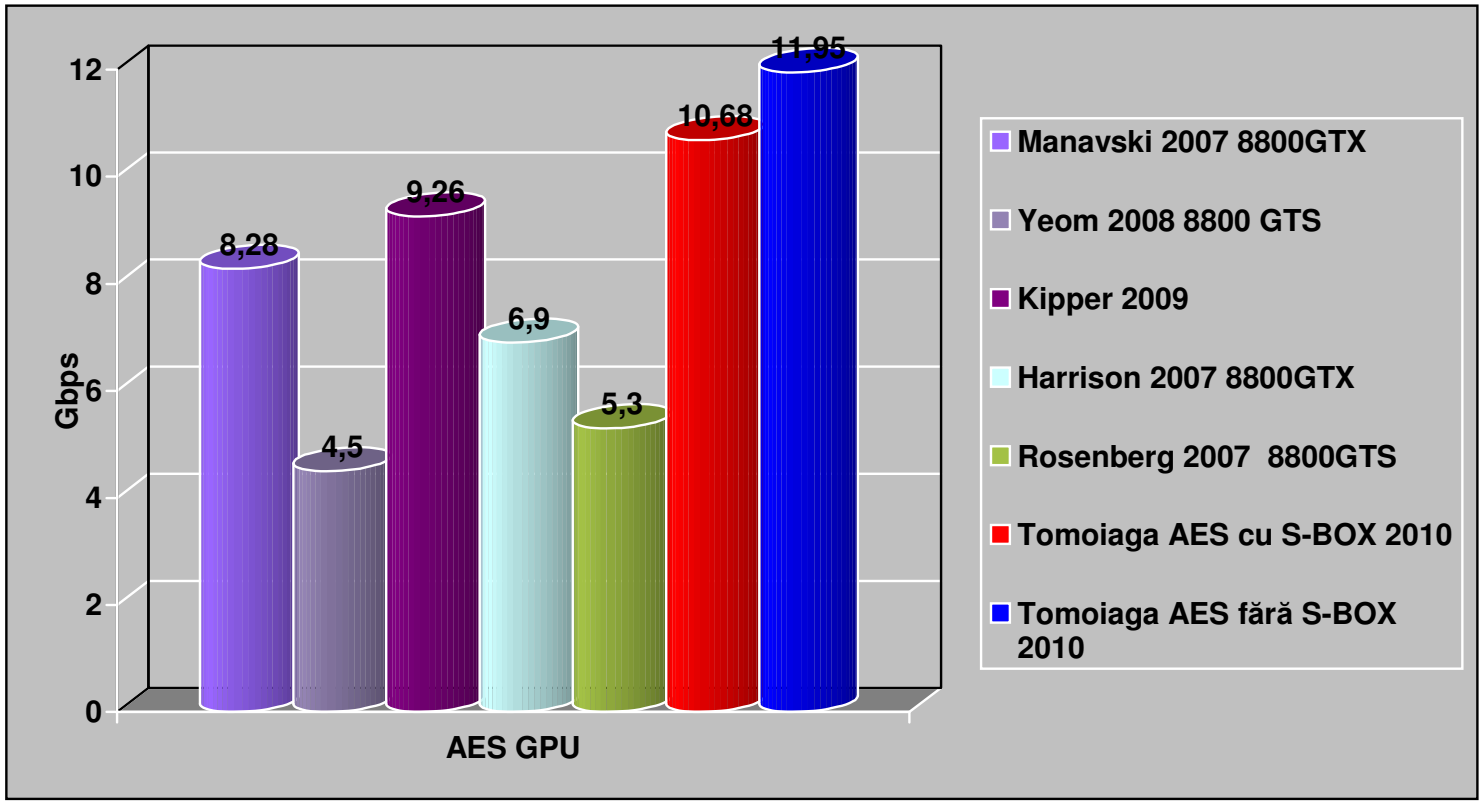

Figura 5.6.1 Comparaţii stadiul actual vs soluţii proprii

6. Concluzii şi cercetare viitoare

Lucrarea de faţă este dedicată cercetărilor din domeniul de mare actualitate al dezvoltării de aplicaţii utilizând procesoare video pe post de coprocesoare pentru rularea acestora şi eliberarea procesoarelor dedicate de sarcini.

Scopul acestei lucrări a fost de a studia posibilitatea folosirii unei soluţii de calcul alternative în criptografie, în speţă folosirea unui GPU în calcule non grafice. Pentru aceasta, s-a început prin a se alege un număr de primitive criptografice. Acestea au fost implementate în mai multe medii de dezvoltare. Aceste implementări au fost rulate pe mai multe platforme de test şi de mai multe sisteme de operare în două etape.

Lucrarea a fost concepută ca o prezentare succesivă a problematicilor ce apar în benchmark-urile legate de primitivele criptografice, sfârşitul fiecărui capitol evidenţiind particularităţi şi concluzii generale, funcţie de specificul considerat.

S-a realizat o analiză a primitvelor criptografice folosite în testele din această teză şi acestea au fost grupate pe categorii în baza unei taxonomii şi în funcţie de modul de operare al fiecăreia.

Se pot evidenţia următoarele direcţii abordate în teză:

- dezvoltarea unor teste pentru verificarea performanţelor primitivelor criptografice pentru procesoarele clasice actuale şi analiza rezultatelor obţinute;
- propunerea unei soluţii de accelerare a unui algoritm criptografic folosind ca suport coprocesarea oferită de plăcile video;
- dezvoltarea aplicaţiei de testare a algoritmului adaptat pentru rularea pe o placă video;
- analiză critică a rezultatelor obţinute.

S-a demonstrat că prin folosirea modurilor corespunzătoare şi prin adaptarea algoritmului AES pentru paralelizare se pot obţine performanţe mult mai bune decât cele oferite de procesoarele actuale.

Modul CTR pentru algoritmul de criptare AES s-a dovedit deosebit de util şi performant în cazul paralelizării şi obţinerii unor timpi de computaţie buni, deşi în majoritatea lucrărilor similare ale altor autori se folosea ECB.

Din studiile şi testele realizate se pot desprinde următoarele concluzii:

- În cazul testelor efectuate pe procesoare clasice rezultatele obţinute nu au condus către un lider de performanţă nici în cazul platformelor, nici al limbajelor de programare şi nici al algoritmilor. Volumul mare de variabile care au putut afecta rezultatele testelor au dus la această concluzie şi au condus către necesitatea testării unei soluţii alternative pentru coprocesare în cazul algoritmilor criptografici;
- Analiza volumului mare de date obţinute în urma testelor şi interpretarea rezultatelor au determinat alegerea ca soluţie a unei nişe în care variabilele ce afectează rezultatele să fie cât mai puţine;
- În cazul testelor efectuate pe procesorul video s-a constatat un spor de performanţă în comparaţie cu rezultatele obţinute în testele precedente. Performanţa obţinută a fost comparabilă cu cele obţinute în lucrări similare şi chiar a depăşit performanţele obţinute pentru lucrările similare.
- Toate testele au fost efectuate în două etape: date stocate în memorie (de mici dimensiuni) şi date stocate pe hard disc(de mari dimensiuni). Rezultatele acestor teste au fost diferite doarece în cazul din urmă timpul

total a fost afectat şi de întârzierea provocată de aducerea/citirea/scrierea datelor de pe hard disc.

De asemenea, s-a efectuat o analiză asupra metricilor de evaluare a performanţei. Această analiză a constituit premisa pentru elaborarea şi rularea testelor ca de altfel şi implementarea algoritmului pentru GPU.

Metodele de accelerare propuse pot fi folosite cu succes şi pe alte plăci grafice care suportă CUDA şi nu numai. Este posibilă adaptarea aplicaţiei şi pentru alte plăci grafice sau multiprocesoare deoarece este folosită paralelizarea.

S-a demonstrat pe baza rezultatelor obţinute că implementările propuse aduc performanţe mai bune decât soluţiile existente şi depăşesc în performanţe, în anumite condiţii, rezultatele obţinute în implementări similare.

Pe baza testelor efectuate s-a putut concluziona că, prin alegerea unui coprocesor capabil să ruleze task-uri în paralel, printr-o alegere corectă a modurilor de lucru ale algoritmului propus şi pe baza mai multor încercări se poate îmbunătăţi performanţa algoritmului, iar procesorul video folosit poate deveni un coprocesor criptografic capabil să ruleze aplicaţii de criptare şi să ajute procesorul, degrevându-l de sarcini.

Concluzia care poate fi desprinsă la finalul lucrării este că, pentru multe aplicaţii, procesoarele grafice constituie soluţii bune, performante şi relativ ieftine de coprocesare. Toate testele efectuate în această teză, analizele şi propunerile făcute sunt factori cheie care trebuie luaţi în considerare la proiectarea unui sistem criptografic în vederea asigurării unor performanţe ridicate.

Pentru elaborarea lucrării a fost utilizată o bibliografie care cuprinde 126 titluri ale unor lucrări de specialitate, multe dintre ele fiind de dată foarte recentă. De asemenea, bibliografia cuprinde 10 lucrări ale căror autor principal este autorul tezei.

Problemele prezentate, precum şi rezultatele obţinute, conferă lucrării şi un caracter practic, deschizând noi perspective cercetărilor în domeniul abordat.

În concluzie, se poate afirma că soluţiile propuse în lucrare, soluţii care pledează pentru utilizarea unor tehnici adecvate vizând produse capabile de procesare paralelă, respectiv soluţii de implementare pentru coprocesare, asigură atingerea unor parametrii de performanţă foarte buni în cadrul diverselor aplicaţii avute în vedere.

6.1. Contribuţii personale

Pornind de la obiectivele declarate ale acestei lucrări, în continuare sunt prezentate principalele contribuţii originale:

- Implementarea a două soluţii de accelerare AES pentru GPU folosind mediul CUDA, soluţii cu performanţe mai bune decât majoritatea celor similare din stadiul actual.
- Elaborarea unei sinteze critice asupra stadiului actual al domeniului benchmark-urilor primitivelor criptografice;
- Elaborarea unui studiu şi a unei analize privind metricile de evaluare a performanţelor şi aspectele ce afectează benchmark-urile;
- Implementarea şi validarea experimentală a algoritmilor propuşi pentru procesoarele clasice;
- Implementarea şi validarea unor metode de optimizare a performanţei unui algoritm criptografic prin adaptarea acestuia pentru rularea pe un procesor grafic;

- O clasificare a algoritmilor criptografici simetrici, funcţii hash şi HMAC care au fost testaţi din punctul de vedere al performanţei pe parcursul acestei lucrări şi descrierea acestor algoritmi;
- Implementarea în diverse medii de programare şi pe diverse platforme, a unui volum mare de teste, cu caracter de unicitate.
- Elaborarea unui studiu critic asupra problematicilor vizând adaptarea algoritmului pentru o placă video, adaptarea pentru paralelizare şi evoluţia procesului de implementare;
- Determinarea analitică şi validarea prin rezultate a performanţei algoritmului implementat şi a metodelor folosite pentru soluţia de accelerare propusă.
- Analiza calitativă şi cantitativă a soluţiilor de criptare testate în prima etapă pe procesoarele clasice cu propunera unei măsuri de accelerare a unui algoritm criptogafic.
- Analiza calitativă şi cantitativă a soluţiilor de criptare testate în a doua etapă pe procesorul video.
- Compararea performanţelor soluţiilor propuse de accelerare cu performanţele unui algoritm propus într-o lucrare similară prin rularea acestuia în acelaşi mediu de test cu soluţiile proprii.
- Elaborarea unei sinteze critice asupra stadiului actual al domeniului coprocesării primitivelor criptografice prin folosirea de plăci video, ASIC şi FPGA.
- Compararea performanţelor soluţiilor propuse de accelerare cu performanţele stadiului actual ale mai multor lucrări ceea ce a dus la concluzia că soluţiile propuse sunt mai performate decât cele din lucrări similare.
- Analiza critică a accesului la memoria GPU-ului în comparaţie cu rapiditatea calculelor realizate de GPU pentru cele două soluţii de accelerare propuse. Această analiză a fost utilă în alegrea soluţiilor finale de accelerare.
- Adaptarea algoritmului implementat pentru GPU şi integrarea lui în OpenSSL, astfel încât OpenSSL să ofere posibilitatea de criptare folosind procesorul video pe post de coprocesor criptografic în criptările care folosesc AES.

6.2. Direcţii de cercetare viitoare

Dintre principalele direcţii de cercetare care pot continua rezultatele obţinute în cadrul acestei teze se pot enumera:

- Implementarea etapei de decriptare pentru algoritmii criptografici simetrici (capitolul 2), testaţi în capitolul 4. Pentru testele efectuate în această etapă a fost necesară doar criptarea. Rezultatele obţinute s-au axat doar pe criptare.
- Algoritmul AES ales a fost AES 128 (dimensiunea blocului 128 biţi, dimensiunea cheii 128 biţi, 10 runde), nefiind implementat şi AES 192 şi AES 256. Acest lucru îşi are justificarea în faptul că implementarea, ajustarea, testarea, debuging-ul pentru AES 128 a reprezentat un proces complex care a necesitat implicare totală şi efort susţinut. Totodată, pentru a putea compara rezultatele obţinute cu implementările altor autori, s-a ales AES 128. Pe viitor, se va încerca şi implementarea AES 192 şi 256.
- Analiza securităţii AES pe GPU realizată pentru implementările din această lucrare din punctul de vedere al puterii consumate de placa video. În [AND08] se vorbeşte despre posibilitatea realizării de atacuri măsurând puterea consumată de un echipament.
- Realizarea unui set de teste folosind aceleaşi implementări, dar oprind Xserverul pentru a putea observa diferenţele care se obţin între cele două abordări. Din testele efectuate în [ROS07] s-a tras concluzia că performanţele mai bune au fost obţinute în urma rulării algoritmilor cu Xserverul pornit.
- Testarea algoritmilor AES adaptaţi pentru CUDA şi pe alte sisteme de operare (Windows ş.a.).
- Paralelizarea AES implementată pentru CUDA şi GPU poate fi extinsă şi la procesoarele cu mai multe nuclee, o paralelizare asemănătoare celei din [BIE05] pentru platformele multi procesor sau celei din [JAC10] pentru procesoarele multi nucleu.

BIBLIOGRAFIE

[AND08] Anderson R. Security Engineering: A Guide to Building dependable Distributed Systems Second Edition, Ed. John Wiley and Sons, 2008

[AHS07] Ahsan S., IT enabled counter terrorism infrastructure: issues and challenges, Int. J. Electronic Security and Digital Forensics, Vol. 1, No. 1, pp.117–124., 2007

[ATA07/01] 1. Atanasiu A. – Secret Sharing Schemes, capitol in Informatics Security Handbook, vol 2 (Ivan I., C. Toma eds), Editura ASE, 2007.

[ATA07/02] Atanasiu, A. – Securitatea informaţiei, vol. 1 (Criptografie), Ed. Infodata, Cluj, 2007.

[BAM01] Bamford, J. Body of Secrets : Anatomy of the Ultra-Secret National Security Agency from the Cold War Through the Dawn of a New Century. New York: Doubleday, 2001

[BAR08] Bardis N., Doukas N., Ntaikos K., „Design and Development of a Secure Militaray Communication based on AES Prototype Crypto Algorithm and Advanced Key Management Scheme", Proceedings WSEAS Transactions Information Science and Applications, 2008 volume 5 ISSN 1790-0832

[BIA09] Biagio A. D., Barenghi A., Agosta G., Pelosi G., "Design of a parallel AES for graphics hardware using the CUDA framework,"in Proceedings of the 2009 IEEE International Symposium on Paral-lel&Distributed Processing, 2009, pp. 1–8.

[BIE05] Bielecki W., Burak D., „Parallelization of the AES Algorithm", Proceedings of the 4th WSEAS International Conference on Information Security, Communications and Computers, pp. 224-228, Tenerife, 2005

[BLU10] BlueKrypt, Cryptographic Key Length Recommendation, http://www.keylength.com/en/6/, 2010

[BOS09] Bos J. W., Osvik D.A., Deian S., „Fast Implementation of AES on Various Platforms", SPEED-CC -- Software Performance Enhancement for Encryption and Decryption and Cryptographic Compilers, 2009, Berlin, ICT-2007-216676

[BRE89] Brewer D.F.C., Nash M.J., THE CHINESE WALL SECURITY POLICY, IEEE SYMPOSIUM ON RESEARCH IN SECURITY AND PRIVACY, 1-3 MAY 1989, OAKLAND, CALIFORNIA. (pp. 206-214).

[BRO05] Brokalakis A., Michail H., Kakarountas A., Milidonis A., Goutis C., „A High-Speed and Area Efficent Hardware Implementation of AES-128 Encryption Standard" Proceedings of the 5th WSEAS International Conference on Multimedia, Internet and Video Technologies,pp. 125-129 Corfu, 2005

[BUL06] Buligu I., Şoavă G., Quality Research by Using Performance Evaluation Metrics for Software Systems and Components, Revista Informatica Economică nr.3(39)/2006 http://revistaie.ase.ro/content/39/I%20Buligiu.pdf

[CHO03] Chodowiec P., Gaj K.,„Very Compact FPGA Implementation of the AES Algorithm", CHES 2003, Proceedings, LNCS Vol. 2779, pp. 319-333, 2003

[CISCO CCNA EXP] Cisco Press CCNA 1-4 Exploration v 4.0, 2009

[CISCO NS] Cisco Press Network Security, 2008

[CISCO SEC] Cisco Press Security, 2009

[CIU03] Ciurescu M., Symmetric encryption/decryption using hash algorithms,2003 http://www.vbforums.com/showthread.php?s=&threadid=232284, Forum

[COO05] D.L. Cook, J. Ioannidis, A.D. Keromytis, and J. Luck. "CryptoGraphics: Secret Key Cryptography Using Graphics Cards." In RSA Conference, Cryptographer's Track (CT-RSA), pp. 334–350, 2005.

[CRI09] Cristea M., Groza B., Sporirea securităţii reţelelor prin protocoale de autentificare rezistente DoS, implementarea unui mecanism de protecţie cu puzzle-uri criptografice, UPT, 2009

[CUD09-1] Cuda Tehnical Training : Introduction to CUDA Programming, 2009 , NVIDIA

[CUD09-2] Nvidia CUDA Programming Guide, 2009 , NVIDIA

[CUD09-3] Nvidia CUDA C Programming, Best Practices Guide, 2009 , NVIDIA

[CUD09-4] CUDA CUFFT Library, 2009, NVIDIA

[CUD09-5] CUDA , Reference Manual 2009, NVIDIA

[CUD09-6] NVIDIA CUDA Development Tools, 2009, NVIDIA

[CUD09-7] NVIDIA CUDA CUBLAS Library, 2009, NVIDIA

[ECR10] ECRYPT II, European Network of Excellence in Cryptology II, Yearly Report on Algorithms and Keysizes, http://www.ecrypt.eu.org, 2010

[FER03] Ferguson N., Schneier N., ”Practical Cryptography”, Wiley Publishing , 2003

[FIPS96] Federal Information Processing Standards Publications Web Site http://www.itl.nist.gov/fipspubs/

[FIPS197] Federal Information Processing Standards Publication 197, 26.11.2001, Advanced Encryption Standard(AES).

[FIPS198-1] Information Processing Standards Publication 198-1, The Keyed-hash Message Authentication Code (HMAC) http://csrc.nist.gov/ publications/fips/ fips198/fips-198a.pdf, 2008

[FLE07] Flechais I., Mascolo C, Sasse M.A., Integrating security and usability into the requirements and design process, Int. J. Electronic Security and Digital Forensics, Vol. 1, No. 1, pp.12–26. 2007

[FRE00] Frei M., Zhou L., Secret Sharing, Course 2000 http://www.cs.cornell.edu/Courses/cs513/2000SP/SecretSharing.html

[GHE05] Gherghe C, D. Popescu: Criptografie, Coduri, Algoritmi, (Ed. Univ. Bucuresti), 2005

[GNU02]The GNU Crypto Project http://www.gnu.org/software/gnu-crypto/, 2002

[GOO05] Good T., Benaissa M., „AES on FPGA: from the fastest to the smallest”, Proceedings of CHES 2005, pp. 427-440, LNCS 3659, Springer, 2005

[GRA04] F. Granelli , “A Novel Methodology for Analysis of the Computational Complexity of Block Cyphers: Rjindael, Camellia and Shacal-2 Compared”, Proceedings of 3rd Conference on Security and Network Architectures (SAR'04), La Londe, France, June 21-25, 2004

[GRO07] Groza B. , Universitatea Politehncă din Timişoara,Introducere în Sistemele Criptografice cu Cheie Publică, 2007 , www.aut.upt.ro/~bgroza

[GRO08] Groza B. , Universitatea Politehncă din Timişoara, ”CONSTRUCŢII CRIPTOGRAFICE HIBRIDE, BAZATE PE TEHNICI SIMETRICE ŞI ASIMETRICE - APLICAŢII ÎN SISTEME DE CONDUCERE”, Teza de Doctorat, 2008, www.aut.upt.ro/~bgroza

[GRO09] B. Groza, D. Pop, I. Silea, "Java Implementation of an Authentication Protocol with Application on Mobile Phones", IEEE-TTTC International Conference on Automation, Quality & Testing, Robotics, AQTR 2009 (THETA 16), pp. 190-195, Cluj-Napoca, Romania, 2009

[GUS95] Gustafson, J.L, HINT: A new way to measure computer performance. Hawaii International Conference on System Sciences, 1995, pp I:392-401.

[HAI03] Haiyong Sie, Arhitectural Analysis of Cryptographic Applications for Network Processors, University of California, 2003

[HAR08] Harrison O.Waldron J., Practical Symmetric Key Cryptography on Modern Graphics Hardware, 17th USENIX Security '08 Symposium, San Jose USA

[HIN06] HINT, www.sc1.ameslab.gov/sc1/HINT-HINT.html, 2006.

[HOD05] Hodjat A., Hwang D., Lai B.C., Tiri K., Verbauwhede I., „A 3.84 Gbits/s AES crypto coprocessor with modes of operation in a 0.18-um CMOS Technology", Proceedings of the 15th ACM Great Lakes Symposium on VLSI 2005, pages 60--63. ACM, ACM Press, April 2005

[HOO05] D. Hook, Beginning Cryptography with Java", Ed. Wiley Publishing, 2005, ISBN-13:978-0-7645-9633-9

[HSH08] Cryptographic hash Algorithm Competition, National Institute of Standard and Tchnology, http://csrc.nist.gov/groups/ST/hash/sha-3/index.html, 2008

[JAC10] Jacquin L., Roca V., „Parallel arithmetic encryption for high-bandwidth communications on multicore/GPGPU platforms", Proceedings of the 4th International Workshop on Parallel and Symbolic Computation, 2010, Grenoble

[JAR10] Jareer A.Q., Walker R., „Performance Evaluation of OpenMP Benchmarks on Intel Quad Core Processors", 14th WSEAS International Conference on Computers, 2010, Corfu,Latest Trends on Computers(Volume 1) , ISSN 1792-4251

[JEB04] Jebelean C., Laboratorul de Structuri de Date si Analiza Algoritmilor, Coduri Huffman,2004, http://www.cs.utt.ro/~calin/resources/sdaa/huffman.ppt

[KAH08] A. Kahate, Cryptographic Algorithms – Impact On Application Performance, http://www.indicthreads.com/1519/cryptographic-algorithms-impact-on-application-performance/, 2008

[KAK07] Kakarountas A., Michail H., „Implementation of a Cryptographic Co-processor", 6th WSEAS International Conference on Information Security and Privacy, Tenerife, 2007

[KAM04] Kaminsky A., Cryptographic One-Way hash Functions, Course, 2004, http://www.cs.rit.edu/~ark/lectures/onewayhash/onewayhash.shtml

[KES07] Kessler G.C., An Overview of Cryptography, http://www.garykessler.net/

[KIP09] Kipper M., Slavkin J., Denisenko D., ," Implementing AES on GPU", University of Toronto, http://www.eecg.toronto.edu/ ~moshovos/CUDA08/ arx/AES_ON_GPU_report.pdf, 2009

[KNU04] J. Knudsen, Java Cryptography, Ed. O'REILLY, 2004

[KUB10] Kubuntu. Download Section http://www.kubuntu.org/getkubuntu/download

[LEE10] Lee R.B., Chen Y.Y., „Processor Accelerator for AES" 2010 IEEE 8th Symposium on Application Specific Processors (SASP), 2010

[LIN04] S.Ling, C.Xing, „Coding Theory", Cambridge University Press 2004.

[LUK09] Luken B., Ouyang M., "AES and DES Encryption with GPU", Proceedings of the ISCA 22nd International Conference on Parallel and Distributed Computing and Communication Systems, pp 67-70, 2009

[MAN07] Manavski Svetlin, *"CUDA Compatible GPU as an efficient Hardware Accelerator for AES Cryptorgraphy" , IEEE International Conference on Signal Processing and Communication, ICSPC 2007, pp. 65–68, Nov. 2007*

[MAO03] Wenbo Mao. Modern Cryprography : Theory an Practice (Hewlett- Packard Profesional Books),Ed. Prentice Hall 2003

[MCA96] Thomas J. McCabe - A Testing Methodology Using the Cyclomatic Complexity Metric, Computer Systems Laboratory, National Institute of Standards and Technology, Gaithersburg, 1996

[MD592] RFC1321 The MD5 Message-Digest Algorithm,1992

[MEL00] H.X.Mel and Doris Baker,Cryptography Decrypted ,Ed. Addison Wesley, 2000

[MEN96] A. Menezes, P. van Oorschot, and S. Vanstone, A. Menezes, P. van Oorschot, and S. Vanstone, CRC Press, 1996 CRC Press, 1996

[MOU07] Mouratidis H., Secure information systems engineering: a manifesto, Int. J. Electronic Security and Digital Forensics, Vol. 1, No. 1, pp.27–41, 2007

[MSA06] Microsoft Academic Program http://www.e-academy.com/

[MSDN] Microsoft Visual Studio 2008/.NET Framework 3.5 MSDN http://msdn.microsoft.com/en-us/library/system.security.cryptography.aspx, 2008

[MSDN08] Microsoft.NET Framework 3.5 Service Pack 1 Download http://www.microsoft.com/downloads/thankyou.aspx?familyId=ab99342f-5d1a-413d-8319-81da479ab0d7&displayLang=en

[NAO95] M. Naor, A. Shamir - Visual Cryptography, în "Advances in Cryptography Eurocrypt 94", A De Santis Ed., Vol. 950 of Lecture Notes in Computer Science, Springer – Verlag, Berlin pp. 1-12, 1995.

[NAO96] M. Naour, A. Shamir. - Visual Cryptography II: improving the contrast via the cover base. Theory of Cryptography Library, report 96-07, 1996

[NSA10]National Security Agency and Central Security Service, NSA Suite B Cryptography,http://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml

[NVI09] NVIDIA Products. Cuda compatible GPU http://www.nvidia.com/object/cuda_learn_products.html

[NVI10] NVIDIA GeForce 8800GT Characteristics. Hardware Heaven Forum http://www.hardwareheaven.com/reviews/8800GTs/whatis.php 17.05.2010

[OSC04] OpenSSl Command Line HowTo http://www.madboa.com/geek/openssl/

[OSP99]Open Source OpenSSL Project http://www.openssl.org/

[PAR04] Parhi K., Zhang X., "An efficient 21.56 Gbps AES implementation on FPGA," in Signals, Systems and Computers. Conference Record of the Thirty-Eighth Asilomar Conference, Nov. 2004, pp. 465–470.

[POC00] Pocatilu P., Cazan D., Ivan I., Evaluarea performantei sistemelor informatice, Revista Informatica Economica, nr. 3 (15)/2000, http://revistaie.ase.ro/content/15/Pocatilu.pdf

[RIJ07] Basic Rijndael encryption code http://snippets.dzone.com/posts/show/3890

[ROS07]Urmas Rosenberg, using Graphic Processing Unit in Block Cipher Calculations, Master's Thesis, http://uraes.sasslantis.ee/

[SCH96] Schneier, B. Applied Cryptography, 2nd ed. New York: John Wiley & Sons, 1996.

[SCH03] Schneier, B ,Niels Ferguson. Schneier's Cryptography Classics Library: Practical Cryptography, 2003

[SCH07] Schneier, B ,Schneier's Cryptography Classics Library: Secrets and Lies (Paperback) 2007

[SCH08] Schneier, B ,Schneier's Cryptography Classics Library: Scnheier on Security 2008

[SHO08] Shostack A., The New School of Information Security Ed. Addison Weskey 2008

[SIN00] Singh S., The Code Book: The Science Of Secrecy from Ancient Egypt to Quantum Cryptography, Ed. Anchor, 2000.

[SOL08] Solga M., Groza B., Evaluarea performanţelor computaţionale pentru funcţii criptografice simetrice şi asimetrice, pe platformele Windows şi Unix, 2008.

[SOR08] Sora I.,Parallel Algorithms http://www.cs.utt.ro/~ioana/calc_par/index.html

[SPE98] SPEC benchmarks, 1998, www.spec.org

[STA03] Standaert F., Rouvroy G., Legat. J, „Efficient Implementation of Rijndael Encryption in Reconfigurable Hardware: Improvements and Design Tradeoffs", CHES 2003, LNCS Vol. 2779

[STE06]Ştefănescu G., Arhitectura sistemelor de clacul, curs, 2006 http://funinf.cs.unibuc.ro/~gheorghe/curs/arhCalc/lec/l02four.pdf

[SZA79] Szabo N., How to Share a Secret http://szabo.best.vwh.net/secret.html

[TAK07] Takeshi Y., „AES Encryption and Decryption on the GPU", GPU Gems 3, http://http.developer.nvidia.com/GPUGems3/gpugems3_ch36.html, 2007

[TIR04] Tirtea R., Deconinck G.," Specifications overview for counter mode of operation. Security aspects in case of faults." Electrotechnical Conference, 2004. MELECON 2004. Proceedings of the 12th IEEE Mediterranean, pag. 769–773 Vol.2, 2004.

[TOM08] Tomoiagă R.D., „Funcţii hash utilizate în criptare simetrică", Simpozionul International "Tinerii si cercetarea multidisciplinara", nr. X, Timisoara,13-14 noiembrie 2008

[TOM09/1] Tomoiagă R., "Proposal of a Model for the Protection of Digital Classified Information", PhD report no. 1, UPT, May 2009

[TOM09/2] Tomoiagă R., Stratulat M., Proposal of a Model for the Protection of Digital Classified Information, 2nd International Conference on Security for Information Technology and Communication, SECIT&C 2009, 19-20 November 2009, Bucureşti, ISBN 978-606-505-283-3

[TOM10/1] Tomoiagă R.D., Stratulat M., „AES behavior on different operating systems or computational platforms", ICCC-CONTI 2010 IEEE International Joint Conferences on Computational Cybernetics and Tehnical Informatics, Timişoara 2010

[TOM10/2] Tomoiagă R.D., Stratulat M., „Evaluation of DES, 3 DES and AES on WINDOWS and UNIX", Proceedings of ICCC-CONTI 2010 IEEE International Joint Conferences on Computational Cybernetics and Tehnical Informatics,Timişoara 2010

[TOM10/3] Tomoiagă R.D., Stratulat M., „The Behaviour of the Cryptographic Primitives in Case of Large Data Volumes", Proceedings of The 8th International Conference on COMMUNICATIONS 2010, Bucureşti

[TOM10/4] Tomoiagă R.D., Stratulat M., „AES Performance Analysis on Several Programming Environments, Operating Systems or Computational Platforms", The Fifth International Conference on Systems and Networks Communications ICSNC 2010, Nice, 2010

[TOM10/5] Tomoiagă R.D., Stratulat M., „AES ON GPU USING CUDA ", Proceedings of European Conference for the Applied Mathematics and Informatics EURO-SIAM, Athens, 2010

[TOM10/6] Tomoiagă R., "Cryptographic Primitives Performance on Windows and Unix Platforms", PhD report no. 2, UPT, January 2010

[TOM11] Tomoiagă R.D., Stratulat M., AES ALGORITHM ADAPTED ON GPU USING CUDA FOR SMALL DATA AND LARGE DATA VOLUME ENCRYPTION, International Journal of Applied Mathematics and Informatics, 2011

[TRA98] Transactions Processing Council, 1998, www.tpc.org

[UBU10] Forum- Installing CUDA in Ubuntu. http://ubuntuforums.org/archive/index.php/t-1112317.html, 17.05.2010

[VBE08] Visual basic . NET Simple Symmetric Encryption/Decription Example. Forum http://www.vbforums.com/showthread.php?t=347276

[WEI08] J. Weiss, „Java Cryptography Extensions: Practical Guide for Programmers", Elsevier 2008

[YON08] Yeom H., Cho Y., Yung M., "High-Speed Implementations of Block Cipher ARIA Using Graphics Processing Units," in Proceedings of the 2008 International Conference on Multimedia and Ubiquitous Engineering (April 24 - 26, 2008). MUE. IEEE Computer Society, Washington, DC, 271-275. 2008.

[ZAJ06] Zajac P., Grosec O., „Searching for a different AES-Class MixColumns operation", Proceedings of the 6th WSEAS international Conference on Applied Computer Science, Tenerife, 2006.

[ZEL05] Zeltser L., Malware: Fighting Malicious Code(Radia perlman Series in Computer Networking and Security), Ed. Pretice Hall, 2005

[ZHA04] Zhanxi Tan, Optimization and Benchmark of Cryptographic Algoritms On Network Processors, Hong Kong University of Science and Techneology, 2004

[ZUG04] Zugravu C., Asigurarea Calitatii in Industria Software, http://facultate.regielive.ro/referate/calculatoare/asigurarea_calitatii_in_industria_software-59516.html, 2004

[www1] http://wiki.hak5.org/wiki/

[www2]http://tools.benramsey.com/md5/

[www3]http://gdataonline.com/seekhash.php

[www4] http://www.md5lookup.com

[www5] http://www.x5.net/faqs/crypto/q103.html

[www6] https://github.com/cbguder/aes-on-cuda

[www7]http://cboard.cprogramming.com/tech-board/98777-aes-subbytes-step.html

[www8] http://osdir.com/ml/debian-bugs-dist/2009-05/msg06123.html

[www9] http://viewvc.coremelt.net/viewvc/it_project/src/cuda

[www10] http://linux.aboutput_bl.com/library/cmd/blcmdl1_time.htm

[www11] http://adi.ro/pub/scoala/Master/SemI/SecDate/Materiale%20CURS%202007/

Glosar

3DES=	Triple DES, Triple Data Encryption Algorithm
AES=	Advanced Encryption Standard
ASIC=	Application-specific integrated circuit
CBC=	Cipher-block chaining
CPU=	Central Processing Unit
COCOMO=	COnstructive COst Model
CTR=	Counter Mode
CUDA=	Compute Unified Device Architecture
DES=	Data Encryption Standard
DRAM=	Dynamic random access memory
FPGA=	Field-Programmeble Gate Array
ECB=	Electronic codebook
GPGPU=	General-Purpose Graphics Processing Units
GPU=	Graphics Processing Unit
HDD=	Hard disk drive
MAC=	Message Authentication Code
NTFS=	New Technology File System
RAID=	Redundant Array of Independent Disks
SBOX=	Substitution-Box
SDK=	Software Development Kit

A1. LISTĂ DE LUCRĂRI PUBLICATE ÎN DOMENIUL TEZEI

A. Volumele unor manifestăru ştiinţifice internaţionale cotate ISI

1. ***AES ON GPU USING CUDA***, Tomoiagă R., Stratulat M., Proceedings of European Conference for the Applied Mathematics and Informatics EURO-SIAM, Athens, 2010 ISSN: 1792-7390 E-ISSN: 1792-7404 ISBN: 978-960-474-260-8 [ISI Web of Knowledge, EI Compendex, ELSEVIER, SCOPUS, IEEExplore]

B. Volumele unor manifestări ştiinţifice internaţionale, indexacte în baze de date internaţionale(BDI)

1. ***AES Performance Analysis on Several Programming Environments, Operating Systems or Computational Platforms,*** Tomoiagă R., Stratulat M., Proceedings of The Fifth International Conference on Systems and Networks Communications ICSNC 2010, Nice(France), 22-27 August 2010, ISBN 978-0-7695-4145-7 [IEEExplore]

2. ***The Behaviour of Cryptographic Primitives in Case of Large Data Volumes***, Tomoiagă R., Stratulat M., Proceedings of 8th International Conference on COMMUNICATIONS 10-12 June 2010, Bucharest, ISBN:978-1-4244-6363-3, [IEEExplore]

3. ***Evaluation of DES, 3 DES and AES on WINDOWS and UNIX,*** Tomoiagă R., Stratulat M., Proceedings of ICCC-CONTI 2010 IEEE International Joint Conferences on Computational Cybernetics and Tehnical Informatics, Timişoara 27-29 May 2010, ISBN 978-1-4244-7431-8 [IEEExplore]

4. ***AES behavior on different operating systems or computational platforms***, Tomoiagă R., Stratulat M., Proceedings of ICCC-CONTI 2010 IEEE International Joint Conferences on Computational Cybernetics and Tehnical Informatics, Timişoara 27-29 May 2010, ISBN 978-1-4244-7431-8 [IEEExplore]

C. Reviste de specialitate din ţară, cu circulaţie internaţională recunoscută, indexate în baze de date internaţionale(BDI). Reviste B+ în clasificarea CNCSIS

1. ***Data flow encryption between terminals with application in robot intrustries***, Tomoiagă R., Revista Robotica & Management, International Journal, Vol 13,No.1, June 2008, ISSN 1453-2069

2. ***AES Algorithm adapted on GPU using CUDA for small data and large data volume encryption***, Tomoiagă R., Stratulat M., International Journal of Applied Mathematics and Informatics, [Scopus], 2011

A2. LISTĂ DE LUCRĂRI PUBLICATE (EXCEPTÂND CELE DIN DOMENIUL TEZEI)

1. ***Smart Home based on an EIB/KNX Network***, Tomoiagă R., Stratulat M., Simpozionul Naţional de Electrotehnică Teoretică National Symposium of Theoretical Electrical Engineering 12-14 OCTOMBRIE 2007, Bucuresti, ISBN 978-973-718-899-1

2. ***Dezvoltarea unui client OPC pentru o Casa Inteligenta bazata pe o retea EIB/KNX,*** Tomoiagă R., Stratulat M., Simpozionul International "Tinerii si cercetarea multidisciplinara" nr. IX , Timisoara, 15-16 noiembrie 2007, pag 109-104, ISSN 1843-6609

3. ***OPC Client for a Smart Home based on an EIB/KNX Network*** , Tomoiagă R., Revista Robotica & Management, International Journal, Vol 12,No.2, December 2007, ISSN 1453-2069

4. ***Proposal of a Model for the Protection of Digital Classified Information***, Tomoiagă R., Stratulat M., 2nd International Conference on Security for Information Technology and Communication, SECIT&C 2009, 19-20 November 2009, Bucureşti, ISBN 978-606-505-283-3

LISTĂ DE FIGURI

LISTĂ DE TABELE