Universitat de Lleida

Escola Politècnica Superior

Enginyeria Tècnica en Informàtica de Sistemes

Treball de Final de Carrera

Implementació paral·lela en MPI de l'algorisme de Shanks

Autor : Eduard Jové Bellot

Directors : Josep Maria Miret Biosca i Fransesc Sebé Feixas

Setembre 2009

Índex

1 Objectius del projecte

L'objectiu d'aquest projecte és implementar la versió en paral·lel de l'algorisme de Shanks en l'entorn MPI. L'algorisme de Shanks resol el problema del logaritme discret, problema en el qual basa la seva seguretat la xifra de clau pública ElGamal

En aquest projecte s'han realitzat les següents tasques:

- Un estudi sobre la base teòrica de la criptografia, és a dir, quins tipus de sistemes hi ha i com funcionen.
- Un estudi sobre les principals estructures algebràiques, sobretot l'aritmètica en modular.
- Un estudi teòric sobre la xifra ElGamal i el problema del logaritme discret.
- Un estudi teòric sobre l'algorisme de Shanks i la seva paral·lelització.
- Instal·lar, estudiar i utilitzar la llibreria NTL de nombres grans.
- Instal·lar i utilitzar la llibreria MPI, per a paral·lelitzar l'algorisme de Shanks. Hem fet un estudi de les diferents maneres de realitzar les comunicacions entre processadors.
- Realitzar la implementació de l'algorisme de Shanks seqüencial, és a dir, per ser executat en una màquina monoprocessador, en llenguatge C++.
- Realitzar la implementació de l'algorisme de Shanks paral·lelitzat per ser util·litzat en un cluster de computadors utilitzant les llibreries MPI.
- Execució de les implementacions de l'algorisme de Shanks tant en format seqüencial com paral·lel i un posterior anàlisi dels resultats obtinguts.

2 Criptografia

2.1 Introducció

La criptografia és l'art o ciència de xifrar i desxifrar informació mitjançant tècniques especials. S'utilitza sovint per permetre un intercanvi de missatges que només puguin ser llegits per persones que posseeixen els mitjans per desxifrar-los.

Quan es parla d'aquesta àrea de coneixement com a ciència, es parla de criptologia, que engloba tant les tècniques de xifratge, és a dir, la criptografia pròpiament dita, com les seves tècniques complementàries, entre les quals s'inclou el criptoanàlisi, que estudia mètodes emprats per trencar textos xifrats a fi de recuperar la informació original en absència de les claus.

La finalitat de la criptografia és, en primer lloc, garantir el secret en la comunicació entre dues entitats (persones, organitzacions, etc.). I, en segon lloc, assegurar que la informació que s'envia és autèntica en un doble sentit: que el remitent sigui realment qui diu ser i que el contingut del missatge enviat, habitualment anomenat criptograma, no hagi estat modificat en el seu trànsit.

En l'actualitat, la criptografia no només es fa servir per comunicar informació de forma segura ocultant el seu contingut. Una de les branques de la criptografia que més ha revolucionat el panorama actual de les tecnologies informàtiques és el de la signatura digital: tecnologia que busca associar l'emissor d'un missatge amb el seu contingut de manera que aquell no pugui repudiar-lo posteriorment.

En l'argot de la criptografia, la informació original que s'ha de protegir s'anomena explícita o text en clar. El xifratge és el procés de convertir el text en clar en un galimaties illegible, denominat text xifrat o criptograma. En general, l'algorisme de xifratge (també anomenat xifra) es basa en l'existència d'una clau: informació secreta que adapta l'algorisme de xifratge per a cada ús diferent.

Les dues tècniques de criptografia clàssica més senzilles de xifratge, són la substitució (que suposa el canvi de significat dels elements bàsics del missatge, -les lletres, els dígits o els símbols-) i la transposició (que suposa una reordenació dels mateixos). La gran majoria de les xifres clàssiques són combinacions d'aquestes dues operacions bàsiques.

El desxifratge és el procés que recupera el text en clar a partir del criptograma i la clau. El protocol criptogràfic especifica els detalls de com s'utilitzen els algorismes i les claus (i altres operacions) per aconseguir l'efecte desitjat. El conjunt de protocols, algorismes de xifratge, processos de gestió de claus i actuacions dels usuaris, és el que constitueixen un criptosistema, que és amb el què l'usuari final treballa i interactua.

Hi ha dos grans grups de xifres: els algorismes que utilitzen una única clau tant en el procés de xifratge com en el de desxifratge, i els que fan servir una clau per a xifrar missatges i una clau diferent per desxifrar. Els primers es denominen xifres simètriques, de clau simètrica o de clau compartida, i són la base dels algorismes de xifratge clàssic. Els segons es denominen xifres asimètriques, de clau asimètrica o de clau pública i formen el nucli de les tècniques de xifratge modernes.

En el llenguatge quotidià, la paraula codi es fa servir de forma indistinta amb xifra. En l'argot de la criptografia, però, el terme té un ús tècnic especialitzat: els codis són un mètode de criptografia clàssica que consisteix a substituir unitats textuals més o menys llargues o complexes, habitualment paraules o frases, per ocultar el missatge, per exemple, "cel blau" podria significar «atacar a l'alba». Per contra, les xifres clàssiques normalment substitueixen o reordenen els elements bàsics del missatge (lletres, dígits o símbols). En l'exemple anterior, «rcnm arcteeaal aaa» seria un criptograma obtingut per transposició. Quan es fa servir una tècnica de codis, la informació

secreta sol recopilar-se en un llibre de codis.

Per poder entendre una mica la criptografia, és temps de plantejar quin tipus de problemes resol aquesta. Els principals problemes de seguretat que resol la criptografia són: la privadesa, la integritat, l'autenticació i el no repudi.

La privadesa, es refereix a que la informació només pugui ser llegida per persones autoritzades. Exemples: Si la comunicació s'estableix per telèfon i algú intercepta la comunicació o escolta la conversa per una altra línia podem afirmar que no hi ha privadesa. Si enviem una carta i per alguna raó algú trenca el sobre per llegir la carta, podem dir que s'ha violat la privadesa. En la comunicació per Internet és molt difícil estar segurs de que la comunicació és privada, ja que no es té control de la línia de comunicació. Per tant si xifrem (amaguem) la informació, qualsevol intercepció no autoritzada no podrà entendre la informació confidencial. Això és possible si es fan servir tècniques criptogràfiques, en particular la privadesa s'aconsegueix si es xifra el missatge amb un mètode simètric o asimètric.

La integritat, es refereix a que la informació no pugui ser alterada en el transcurs del seu enviament. Exemples: Quan comprem un bitllet d'avió és molt prudent verificar que les dades són correctes abans d'acabar l'operació. Això es pot realitzar al mateix temps de la compra. Per exemple, si hem comprat un viatge per anar a Amsterdam no volem que degut a una modificació ens acabin donan un bitllet per una altra destinació. Per Internet com la compra es pot fer des de dues ciutats molt distants i la informació ha de viatjar per una línia de transmissió de la qual no es té control, és molt important estar segurs que la informació transmesa no ha estat modificada (en aquest cas es diu que hi ha integritat). Això també es pot solucionar amb tècniques criptogràfiques particularment amb processos simètrics o asimètrics.

L'autenticitat, es refereix a que es pugui confirmar que el missatge rebut hagi estat enviat per qui diu el comandament o que el missatge rebut és el que s'esperava.

Exemples: Si estem envian dades bancaries a una web, ens hem d'assegurar que aquesta és realment qui diu ser i no una web falsa. La tècnica necessària per poder verificar l'autenticitat tant de persones com de missatges és la signatura digital, i d'alguna manera reemplaça a la signatura autògrafa que s'usa comunament.

Per Internet és molt fàcil enganyar a una persona amb qui es té comunicació respecte a la identitat. Resoldre aquest problema és per tant molt important per efectuar una comunicació fiable.

El no repudi, es refereix a que no es pugui negar l'autoria d'un missatge enviat.

Quan es dissenya un sistema de seguretat una gran quantitat de problemes poden ser evitats si s'utilitzen mecanismes per comprovar autenticitat, de garantir privacitat, d'assegurar integritat i evitar el no-repudi.

La criptografia simètrica i asimètrica conjuntament amb altres tècniques, com el bon maneig de les claus i la legislació adequada resolen satisfactòriament els problemes anteriorment plantejats.

2.2 Història

Les primeres civilitzacions van desenvolupar tècniques per enviar missatges durant les campanyes militars, de manera que si el missatger era interceptat la informació que portava no corregués el perill de caure en mans de l'enemic. Possiblement, el primer criptosistema que es coneix fora documentat per l'historiador grec Polibi: un sistema de substitució basat en la posició de les lletres en una taula. També els romans van utilitzar sistemes de substitució, sent el mètode actualment conegut com Cèsar, perquè suposadament Juli Cèsar el va emprar en les seves campanyes (segons alguns autors,

en realitat Juli Cèsar no usava aquest sistema de substitució, però l'atribució té tant arrelament que el nom d'aquest mètode de substitució ha quedat per als annals de la història). Un altre dels mètodes criptogràfics utilitzats pels grecs va ser l'escítala espartana, un mètode de transposició basat en un cilindre que servia com a clau, en el que s'enrotlla el missatge per poder xifrar i desxifrar.

El 1465 l'italià Leon Battista Alberti va inventar un nou sistema de substitució polialfabètica que va suposar un gran avenç de l'època. Un altre dels criptògrafs més importants del segle XVI va ser el francès Blaise de Vigenère que va escriure un important tractat sobre "l'escriptura secreta" i que va dissenyar una xifra que ha arribat als nostres dies associada al seu nom. Durant els segles XVII, XVIII i XIX, l'interès dels monarques per la criptografia va ser notable. Les tropes de Felip II van emprar durant molt de temps una xifra amb un alfabet de més de 500 símbols que els matemàtics del rei consideraven inexpugnable. Quan el matemàtic francès François Viète va aconseguir criptoanalizar aquell sistema per al rei de França, en aquell temps Enric IV, el coneixement mostrat pel rei francès va impulsar una queixa de la cort espanyola davant del papa Pius V acusant a Enric IV d'utilitzar màgia negra per vèncer als seus exèrcits.

Des del segle XIX i fins a la Segona Guerra Mundial, les figures més importants van ser la de l'holandès Auguste Kerckhoffs i la del prussià Friedrich Kasiski. Però és en el segle XX quan la història de la criptografia torna a experimentar importants avenços. Especialment durant els dos aconteixements bèl·lics que van marcar el segle: la Primera Guerra Mundial i la Segona Guerra Mundial. A partir del segle XX, la criptografia fa servir una nova eina que permetrà aconseguir xifres millors i més segures: les màquines de càlcul. La més coneguda de les màquines de xifratge possiblement sigui la màquina alemanya Enigma: una màquina de rotors que automatitza

considerablement els càlculs que s'havia de fer per a les operacions de xifratge i desxifratge de missatges. Per vèncer l'enginy alemany, va ser necessari el concurs dels millors matemàtics de l'època i un gran esforç computacional. No en va, els grans avenços tant en el camp de la criptografia com en el del criptoanàlisi no van començar fins aleshores.

La criptografia actual s'inicia en la segona meitat de la dècada dels anys 70. No és fins la invenció del sistema conegut com a DES (Data Encryption Standard) el 1976 que es dóna a conèixer més àmpliament, principalment en el món industrial i comercial. Posteriorment amb el sistema RSA (Rivest, Shamir, Adleman) el 1978, s'obre el començament de la criptografia en un gran rang d'aplicacions: en transmissions militars, en transaccions financeres, en comunicació via satèl·lit, en xarxes d'ordinadors, en línies telefòniques , en transmissions de televisió etc.

2.3 Criptografia simètrica

La criptografia simètrica es refereix al conjunt de mètodes que permeten tenir comunicació segura entre les parts sempre que anteriorment s'hagin intercanviat la clau corresponent que anomenarem clau simètrica. La simetria es refereix a que les parts tenen la mateixa clau tant per xifrar com per desxifrar.

Aquest tipus de criptografia és coneguda també com criptografia de clau compartida.

Hi ha una classificació d'aquest tipus de criptografia en dues famílies, la criptografia simètrica de blocs (block cipher) i la criptografia simètrica de flux (stream cipher). Encara que amb lleugeres modificacions un sistema de criptografia simètrica de blocs pot modificar per convertir-se en criptografia simètrica de flux, i inversament, però és important veure-les per separat ja que es fan servir en diferents aplicacions.

La criptografia simètrica ha estat la més usada en tota la història. Aquesta ha pogut ser implementada en diferents dispositius, manuals, mecànics, elèctrics i informàtics. La idea general és aplicar diferents transformacions al missatge que es vol xifrar de tal manera que només coneixent una clau es pugui aplicar el procés invers per poder així desxifrar. Encara que no hi ha un tipus de disseny estàndard, potser el més popular és el de Feistel, que consisteix essencialment a aplicar un nombre finit d'interaccions, que finalment dóna com a resultat el missatge xifrat. Aquest és el cas del sistema criptogràfic simètric més conegut, el DES. DES és un sistema criptogràfic que pren com a entrada un bloc de 64 bits del missatge i aquest se sotmet a 16 interaccions, amb una clau de 56 bits. Aquest sistema va ser pres com a estàndard i ha estat un dels més coneguts, usats i estudiats. Actualment la xifra estandard més utilitzada és l'AES.

El principal problema amb els sistemes de xifratge simètric no està lligat a la seva seguretat, sinó a l'intercanvi de claus. Una vegada que el remitent i el destinatari hagin intercanviat les claus poden utilitzar l'algorisme simètric per comunicar-se amb seguretat. Seria molt més fàcil per a un atacant intentar interceptar una clau que provar les possibles combinacions de l'espai de claus.

Un altre problema és el nombre de claus que es necessiten. Si tenim un nombre n de persones que necessiten comunicar-se entre sí, es necessita 1 clau per a cada parella de persones que hagin de comunicar-se de manera privada. Això pot funcionar amb un grup reduït de persones, però seria impossible portar-ho a terme amb grups més grans.

Els criptosistemes simètrics no proporcionen cap mètode perque el receptor del missatge sàpiga amb seguretat que el missatge ha sigut enviat per la persona correcta i que no ha estat modificat.

2.4 Criptografia asimètrica

La criptografia asimètrica és per definició la que utilitza dues claus diferents, una per xifrar que se l'anomena clau pública i una altra per desxifrar que és la clau privada. El naixement de la criptografia asimètrica es va donar per la necessitat d'obtenir d'una manera més pràctica l'intercanvi de les claus. Diffie i Hellman, proposen una forma per fer això, però no va ser fins que el conegut mètode de Rivest Shamir i Adleman, RSA, publicat el 1978, quan pren forma la criptografia asimètrica. El seu funcionament està basat en la impossibilitat computacional de factoritzar nombres enters grans.

Actualment la Criptografia asimètrica és molt usada, les seves dues principals aplicacions són precisament l'intercanvi de claus privades i la signatura digital. Una signatura digital es pot definir com una cadena de caràcters que s'afegeix a un arxiu digital que fa el mateix paper que la signatura convencional que s'escriu en un document de paper ordinari.

En l'actualitat la criptografia asimètrica o de clau pública es divideix en dues grans famílies, segons el problema matemàtic del qual basen la seva seguretat. La primera de les famílies és la que basa la seva seguretat en el Problema de Factorització Sencera PFE, els sistemes que pertanyen a aquesta família són, el sistema RSA, i el de Rabin Williams entre altres. La segona família és la que basa la seva seguretat en el Problema del Logaritme Discret PLD, a aquesta família pertany el sistema de Diffie Hellman DH d'intercanvi de claus i el sistema DSA de signatura digital.

Encara que a les famílies anteriors pertanyen els sistemes asimètrics més coneguts, hi ha un altre tipus de sistemes que basen la seva seguretat en un altre tipus de problema com per exemple en el problema del logaritme discret sobre corbes elíptiques, sobre problemes de retícules i sobre subconjunts de classes de camps numèrics reals i complexos.

El major avantatge de la criptografia asimètrica és que es pot xifrar amb una clau i desxifrar amb l'altra. Malgrat tot aquest sistema té bastants desavantatges:

· Per a un mateix nivell de seguretat es necessita major temps de procés.

· Les claus han de ser més grans que les simètriques.

· El missatge xifrat ocupa més espai que l'original.

El sistema de criptografia sobre corbes elíptiques representa una alternativa menys costosa per a aquest tipus de problemes.

2.4.1 Xifra RSA

En el cas de RSA el problema matemàtic és el de la factorització d'un nombre enter gran n (1024 bits). Aquest nombre enter se sap que és producte de dos nombres primers p, q de la mateixa longitud. Aleshores la clau pública és el nombre n i la privada és p, q. El raonament del funcionament de RSA és el següent:

a) A cada usuari se li assigna un nombre enter n, que funciona com la seva clau pública .

b) Només l'usuari respectiu coneix la factorització de n (o sigui p, q), que manté en secret i és la clau privada.

c) Hi ha un directori de claus públiques.

d) Si algú vol enviar un missatge m a algun usuari llavors agafa la seva clau pública n i amb informació addicional també pública pot enviar el missatge xifrat c, que només podrà desxifrar l'usuari corresponent. El missatge m convertit a nombre (codificació) se sotmet a la següent operació (on e és constant i públic):

$$c = m^e \cdot (mod n)$$

e) Llavors el missatge c pot viatjar sense problema per qualsevol canal insegur.

f) Quan la informació xifrada arriba al seu destí el receptor procedeix a desxifrar el missatge amb la següent fórmula:

$$m = c^d \cdot (mod n)$$

g) Es pot demostrar que aquestes fórmules són inverses i per tant donen el resultat desitjat, (n, e) són públics i actuen com a clau pública. La clau privada és la parella (p, q) o equivalentment al nombre d. La relació que existeix entre d i e és que un és l'invers multiplicatiu de l'altre mòdul $\lambda(n)$, on $\lambda(n) = (p-1) \cdot (q-1)$.

En termes molt generals és així com funciona el sistema RSA.

2.4.2 El Problema de Diffie-Hellman

El problema de Diffie-Hellman està relacionat amb el problema del Logaritme Discret, i és la base d'alguns sistemes criptogràfics de clau pública, com el de Diffie-Hellman i el de ElGamal.

Primerament definirem què és un generador per entendre millor el problema. La resta de conceptes matemàtics estan explicats en el següent capítol.

És diu que g és un element generador del conjunt $\mathbb{Z}_p^*$, amb p primer, si es compleix :

$$\forall\ b \in \mathbb{Z}_p^*\ \exists\ i \text{ tal que } g^i = b$$

El problema de Diffie-Hellman consisteix amb el següent: donat un número primer p, un element generador g de $\mathbb{Z}_p^*$ i els elements g^a i g^b, hem de calcular el número $g^{ab}(mod p)$. La dificultat en aquest cas és que no coneixem ni el valor de a ni el de b. Per obtenir-los hauriem de sobre calcular logaritmes discrets. Veiem com funciona **l'algorisme d'intercanvi de claus de Diffie-Hellman** basat en aquesta idea.

Tenim dos usuaris, A i B, que volen crear una clau compartida a través d'un canal insegur. Primerament es posen d'acord en escollir un numero primer p i un element generador g d'un conjunt $\mathbb{Z}_p^*$. Aquestes dades són públiques, és a dir, les pot conèixer qualsevol usuari del canal. Els passos que seguiran per crear la clau compartida són els seguents:

1. A tria un nombre aleatori x, comprès entre 1 i $p-2$ i envia a B el valor : $g^x(modp)$
2. B tria un nombre aleatori y, comprès entre 1 i $p-2$ i envia a A el valor : $g^y(modp)$
3. A rep el valor $g^y(modp)$ i calcula la clau $K = (g^y)^x(modp)$
4. B rep el valor $g^y(modp)$ i calcula la clau $K = (g^y)^x(modp)$

Com els possibles atacans que escolten el canal no coneixen ni x ni y no poden saber la clau privada K.

2.4.3 Xifra ElGamal

Aquesta xifra està basada en la complexitat del problema Diffie-Hellman . Va ser descrit en 1984 per Taher ElGamal. Actualment s'usa en el programari lliure GNU Privacy Guard (GnuPG), versions de PGP i altres sistemes.

Si desitgem intercanviar un missatge m entre un emissor i un receptor, considerant que el missatge pertany a G, el qual és un grup cíclic finit, l'algorisme segueix els següents passos:

1. L'emissor i el receptor acorden un grup finit G i un generador g de G.

2. El receptor escull una clau privada ($x \in G$) i calcula la clau pública a partir d'ella ($g^x \in G$).
3. L'emissor escull $v \in G$ i calcula $g^v \in G$.
4. L'emissor, amb la clau pública del receptor ($g^x \in G$), calcula: $(g^x)^v$ i $m{\cdot}g^{xv}$ en G .
5. L'emissor envia la parella (g^v , $m{\cdot}g^{xv}$) al receptor.
6. El receptor, per recuperar el missatge original calcula: (g^{vx}) i $(g^{vx})^{-1}$ i obté m calculant $(m{\cdot}g^{vx})(g^{vx})^{-1}$.

Per a que el protocol sigui segur i eficient el grup G i l'element g han de complir les següents condicions :

- Calcular potències d'un element de G ha de ser poc costós.
- El problema del logaritme discret en el subgrup cíclic de G generat per g ha de ser computacionalment intractable.

ElGamal va descriure el protocol anterior sobre el grup multiplicatiu del cos $\mathbb{Z}_p^*$. D'aquesta manera els productes i potències del protocol anterior s'efectuen amb mòdul p, on p és un número primer.

3 Preliminars matemàtics

En aquest capítol explicarem els fonaments matemàtics per introduir el problema del logaritme discret, tot i que en el capítol anterior ja s'hi ha fet referència i s'ha avançat algun concepte matemàtic. A continuació farem un recull de les definicions algebràiques i conceptes matemàtics més importants i mostrarem alguns exemples per tal d'entendre els algorismes que apareixen en capítols posteriors.

3.1 Lleis de composició interna

Una llei de composicó interna o operació interna sobre un conjunt A, és una aplicació del producte cartesià $A \times A$ en A.

Tipus de lleis

Una lei de composició interna $\otimes$ sobre un conjunt A es diu que és :

- Associativa, si $a \otimes (b \otimes c) = (a \otimes b) \otimes c$, $\forall a, b, c \in A$.
- Conmutativa, si $a \otimes b = b \otimes a$, $\forall a, b \in A$.

Element neutre

Es diu que e és un element neutre de $(A, \otimes)$, essent $(A, \otimes)$un cojunt amb una operació interna, si es compleix :
$a \otimes e = e \otimes a = a$, $\forall a \in A$ i $e \in A$. L'element neutre és únic.

Element invertible o simètric

Es diu que a^{-1} és un element invertible de $(A, \otimes)$, essent $(A, \otimes)$ un cojunt amb una operació interna i un element neutre e, si es compleix:

$$a \otimes a^{-1} = a^{-1} \otimes a = e \ .$$

Si $\otimes$ és associativa i té element neutre, tot element simetritzable té un únic simètric.

Propietat distributiva

Donades dues operacions internes $\otimes$ i $\star$ es diu que l'operació $\star$ és distributiva sobre l'operació $\otimes$, si es compleix :

$a \star (b \otimes c) = (a \star b) \otimes (a \star c)$ i $(a \otimes b) \star c = (a \star c) \otimes (b \star c)$, $\forall a, b, c \in A$.

3.2 Estructura de grup

Grup

Donat un conjunt G dotat d'una operació interna $\otimes$, té estructura algebraica de grup, $(G, \otimes)$, si se satisfan les propietats següents:

1. Associativa.
2. Existeix element neutre.
3. Tot element de G té simètric.

Grup abelià

Si a més a més és satisfà la propietat commutativa, és diu que $(G, \otimes)$ és un grup abelià.

Grup finit

Un grup finit G és el que té un nombre finit d'elements.

L'ordre o cardinal d'un grup finit G és el nombre d'elements que té el grup i el denotarem per $|G|$.

Subgrups d'un grup

Partin del grup $(G, \otimes)$ i considerant un subconjunt H de G, direm que $(H, \otimes)$ és un subgrup del grup $(G, \otimes)$ si $(H, \otimes)$ té estructura de grup per si mateix.

3.3 Estructura d'Anell

Anell

Un conjunt A amb dues operacions internes $\otimes$ i $\star$ té estructura d'anell i que denotarem per $(A, \otimes, \star)$ si es compleix :

1. $(A, \otimes)$ és un grup abelià.
2. L'operació $\star$ és associativa.
3. L'operació $\star$ és distributiva respecte de l'operació $\otimes$.

Anell unitari

L'Anell $(A, \otimes, \star)$ es diu que és unitari si té element neutre respecte de l'operació $\star$.

Anell commutatiu

L'Anell $(A, \otimes, \star)$ és commutatiu si l'operació $\star$ compleix la propietat comutativa.

Domini integrat

Es diu que $a \in A - \{0\}$ és un divisor de zero d'un anell $(A, \otimes, \star)$, on 0 és l'element neutre de $(A, \otimes)$, si existeix un element $b \in A - \{0\}$ tal que $a \star b = 0$ o $b \star a = 0$. Un anell commutatiu unitari sense divisors de zero s'anomena domini integrat.

3.4 Estructura de Cos

Cos

Un conjunt K amb dues operacions internes $\otimes$ i $\star$ és un cos si es compleix :

1. $(K, \otimes, \star)$ és un anell unitari.
2. Tot element de K diferent del 0 , on 0 és el neutre de $(K, \otimes)$, és invertible.

Cos commutatiu

Si l'operació $\star$ compleix la propietat commutativa, es diu que $(K, \otimes, \star)$ és un cos commutatiu.

3.5 Exemples d'estructures algebràiques

El grup $\mathbb{Z}_p$

El conjunt $\mathbb{Z}_p$ està format per totes les classes d'equivalencia modul p:

$\mathbb{Z}_p = \{0, 1, 2, ..., p-1\}$

Amb la suma natural compleix les propietats del grup abelià.

El cos $(\mathbb{Z}_p, +, *)$

És el grup $\mathbb{Z}_p$, p primer, amb les operacions de suma i multiplicació. Compleix amb totes les propietats dels cossos.

El comportament de les operacions és el següent :

L' operació suma d'enters forma un grup cíclic, totes les sumes es veuen reduïdes pel mòdul p, el mateix pasa amb l'operació del producte.

El grup multiplicatiu $\left(\mathbb{Z}_p^*, *\right)$

El conjunt $\mathbb{Z}_p^*$ és el conjunt $\mathbb{Z}_p$ però sense el número zero.

$\mathbb{Z}_p^* = \{1, 2, ..., p-1\}$

Compleix amb el producte natural totes les propietats dels grups.

El grup $\mathbb{Z}_p^*$ és cíclic, és a dir, podem escriure tots els elements com a potències d'un element g (generador):

$\mathbb{Z}_p^* = < g > = \{1, g, g^2, g^3, ..., g^{p-2}\}$

Exemple

$(\mathbb{Z}_{11}^*, *) = \{1, 2, 3, ..., 10\}$

Si agafem $g = 2$ com a generador de $(\mathbb{Z}_{11}^*, *)$ aleshores obtenim el següent:

$$\begin{array}{ll} g = 2(mod11) & g^6 = 9(mod11) \\ g^2 = 4(mod11) & g^7 = 7(mod11) \\ g^3 = 8(mod11) & g^8 = 3(mod11) \\ g^4 = 5(mod11) & g^9 = 6(mod11) \\ g^5 = 10(mod11) & g^{10} = 1(mod11) \end{array}$$

Podem dir que l'ordre de $g = 2$ és 10, ja que és la menor potència de 2. Ho denotem : $ord(g) = 10$.

L'ordre de qualsevol element de $\mathbb{Z}_p^*$ divideix a $p-1$, aleshores els possibles ordres d'un element g en $\mathbb{Z}_{11}$ són: $1, 2, 5, 10$.

4 El problema del Logaritme Discret

En aquest capítol definirem el logaritme discret i explicarem per què suposa un problema de difícil solució. El problema del logaritme discret és un dels més utilitzats en criptografia.

També introduirem el problema en el grup $\mathbb{Z}_p^*$, ja que xifres com ElGamal basen la seva seguretat en l'elevada complexitat d'aquest problema.

Anem a fer un breu introducció al problema del logaritme discret, per familiaritzar-nos una mica amb ell.

4.1 Logaritme discret

Donats dos números a i b i el mòdul p, definim el logaritme discret de a en base b i mòdul p a la resolució de l'equació $b^c(modp) = a$, on a, b i p són constants i c és la incògnita i es denota $c = log_b(a)$.

4.2 El problema del logaritme discret generalitzat GDLP

Una descripció més rigurosa del logaritme discret seria la següent.

Sigui (G,*) un grup cíclic finit amb n elements i un generador g. Aleshores anomenem logaritme discret de y en base g a l'enter x comprés entre 0 i $n-1$ que compleix : $y = g^x$ i ho denotem com $x = log_g y$.

El problema del logaritme discret consisteix en, donats g i $y \in G$, trobar la x tal que $g^x = y$.

En un cos finit és molt fàcil elevar un número a una potència degut a l'algorisme de potenciació modular o a altres estràtegies. En canvi, l'operació inversa és molt costosa. Els millors temps de computació coneguts són de tipus subexponencial. El problema es

va fent més difícil com més gran sigui el tamany del grup escollit.

4.3 El problema del logaritme discret sobre $\mathbb{Z}_p^*$

El problema del logaritme discret sobre el grup $\mathbb{Z}_p^*$ (DLP) és un cas particular del problema del logaritme discret generalitzat, on p és un número primer i l'operarció de $\mathbb{Z}_p^*$ és la multiplicació de manera que $\mathbb{Z}_p^*$=$< g >= \{1, g, g^2, g^3, ..., g^{p-2}\}$, on g és l'element generador i y un element $\in \mathbb{Z}_p^*$. El problema en aquest cas serà trobar l'únic enter x comprès entre 1 i $p-2$ que compleixi :

$$g^x = y(modp)$$

Es a dir, calcular :

$$log_g y = x$$

La dificultat de resoldre el problema del logaritme discret és independent del generador escollit, ja que donats g i g' generadors de un grup cíclic G d'ordre n, donat un $g \in G$, i sabent : $x = log_g y$, $x' = log_{g'} y$ i $z = log_g g'$, aleshores tenim que g^x =y =$g'^{x'} = (g^z)^{x'}$.

Com a consecuencia tenim :

$$x = z \cdot x'(modn)$$

i

$$log_{g'} y{=}(log_g y){\cdot}(log_g g')^{-1}(modn)$$

Això significa que cada algorisme que calcula logaritmes en base g, podrà ser utilitzat per calcular logaritmes en base g' també generador de G .

Molts criptosistemes es basen en el problema del logaritme discret en el grup $\mathbb{Z}_p^*$.

4.4 Resolució del problema del logaritme discret

Hi ha tres principals grups d'algorismes per resoldre el problema del logaritme discret.

El primer, format pels que funcionen per grups arbritaris. Entre els quals podríem nombrar la Cerca exhaustiva, Baby Step-Giant Step i Rho de Pollard.

El segon són aquells que funcionen per a grups arbritaris, però són eficients per a grups que cumpleixen certes característiques, un exemple sería l'algorisme de Pohlig-Hellman.

El tercer sería l'Index Calculus, bastant eficient (subexpoenecial), però que solament funciona en determinats grups.

A continuació comentem alguns d'aquests algorismes i en el següent capítol veurem amb més detall l'algorisme de Shanks (Baby Step-Giant Step).

4.4.1 Cerca exhaustiva

Donat un grup G de ordre n i un generador g, el metode de la cerca exhaustiva consisteix en calcular g^0,g^1,g^2,..., fins trobar el valor desitjat, es a dir operem mitjançant la força bruta. Aquest mètode es intractable computacionalment, ja que el temps de cost esperat és de $\mathcal{O}(n)$ operacions en G .

Per tal d'optimitzar més la cerca hem utilitzat l'estrategia de calcular g^i a partir de l'anterior, és a dir :

$$g^i = g^{i-1} * g$$

L'algorisme utilitzat seria el següent :

Entrada : Un primer p, un generador g i un element y $\in$ en $\mathbb{Z}_p^*$.

Sortida : El logaritme discret $x = log_g y(modp)$.

x := 0 ;

A := 1 ;

mentres (A != y)$(modp)$ fer

A := A$*g(modp)$;

x = x + 1 ;

fmentre ;

retorna x ;

4.4.2 Algorisme de Pohlig-Hellman

L'algorisme va ser descobert per Roland Silver, però publicat per primera vegada per Stephen Pohlig i Martin Hellman.

L'algorisme de Pohlig-Hellman funciona per a grups arbritaris, però és més eficient per a grups en els quals el seu ordre és pot descomposar en factors primers petits. L'algorisme es basa en el teorema Xinès del residu.

Anem a explicar l'algorisme en funció del grup format per tenir tots els elements de $\mathbb{Z}_p^*$, que són primers entre si amb p, l'algorisme sería el següent :

Entrada : els enters p, g, y.

Sortida : x, tal que $y \equiv g^x(modp)$.

1. Determineu la factorització de nombres primers de l'ordre del grup :

$$\varphi(p) \equiv p_1 \cdot p_2 \cdot \ldots \cdot p_n$$

2. Amb el teorema de la resta sabem que $x = a_1 \cdot p_1 + b_1$.

Ara hem de trobar el valor b_1, mitjançant la següent equació, utilitzant un algorisme ràpid com Baby Step-Gaint Step :

$$\begin{aligned} y^{\varphi(p)/p_1} &\equiv (g^x)^{\varphi(p)/p_1}(mod p) \\ &\equiv \left(g^{\varphi(p)}\right)^{a_1} \cdot g^{b_1 \varphi(p)/p_1}(mod p) \quad \text{(teorema d'Euler)} \\ &\equiv (g^{\varphi(p)/p_1})^{b_1} \pmod{p} \end{aligned}$$

Quan es compleixi : $g^{\varphi(p)/p_1} \equiv 1(mod p)$, aleshores l'ordre de g és menor que $y^{\varphi(p)/p_1}(mod p)$. En aquest cas hi haurà més d'una solució per a x menor que $\varphi(p)$, pero podem exigir que $b_1 = 0$.

La mateixa operació es realitza per a p_2 fins a p_n .

3. Quan acaben amb les congruencies simultànies, x es pot resoldre utilitzant el teorema Xinès del residu.

La complexitat de temps de l'algorisme de Pohlig-Hellman és $O(\sqrt{n})$ per a un grup d'ordre n, però és més eficaç si el seu ordre és pot descomposar en factors primers petits.

5 Algorisme de Shanks

És l'algorisme en el qual es basa la implementació d'aquest treball de final de carrera. Més conegut com l'algorisme Baby Step-Giant Step va ser proposat per Shanks al 1981 i resol el problema del logaritme discret. Hi ha una millora de temps respecte a la cerca exhaustiva a canvi d'augmentar la memòria utilitzada. Utilitza una taula en la qual hi podem accedir de forma eficient i que la guardarem a memòria. L'algorisme és el següent :

Algorisme (BSGS):

Entrada : Un primer p, un generador g, un element y, una mida de la llista k $\in$ en $\mathbb{Z}_p^*$.

Sortida : El logaritme discret $x = log_g y(mod p)$

1. Crear una llista que contingui les parelles $(y \cdot g^i, i)$, per a $0 \leq i < k$, on el primer valor és la clau

2. j=0

3. Si $g^{k \cdot j}$ es troba a la tupla on el segon component de la parella és a la posició t

Retorna $j \cdot k - t$

4. Sino, incrementa j i torna al pas (3)

Vegem un exemple:

Amb el grup multiplicatiu $\mathbb{Z}_{11}^*$, amb un element y del grup, per exemple $y = 9$ i un element generador g, per exemple $g = 2$. Creem una llista de longidud 3 que contingui els valors $\{y, y \cdot g(mod p), y \cdot g^2(mod p)\}$ o sigui amb els valors {9, 7, 3 }. Per calcular $x = \log_2(9)$ podem fer salts de longitud tres, és a dir, comprovar g, g^4, g^7,... i mirar si el resultat és a la llista, en aquest cas la solució és $2^6 = 9(mod 11)$. Al comprovar $2^7 = 7(mod 11)$, trobariem que 7 és a la llista i a partir d'aquesta podem extreure la solució. Així amb una llista de longi-

tud tres podem reduïr un terç el temps de cerca respecte a la cerca exhaustiva.

A continuació explicarem a més detall els pasos de l'algorisme anterior.

Creació de la taula

Cal una taula on les seves entrades estan formades per dos valors, la creació de la taula seran els "pasos de nen" de l'algorisme. La taula serà de la següent forma :

$(y\ ,\ 0)$
$(y{\cdot}g\ ,\ 1)$
$(y{\cdot}g^2\ ,\ 2)$
...
$(y{\cdot}g^{k-1}\ ,\ k-1)$

Per la creació de la taula utilitzarem la mateixa estrategia que hem fet servir en la búsqueda exhaustiva, es a dir, per calcular $y{\cdot}g^i$, l'obtindrem a partir de $y{\cdot}g^{i-1}$. Hem de tenir clar que hem de guardar el valor $y{\cdot}g^i$ junt amb el valor i. La mida de la taula ideal és $k = \sqrt{p}$. Si volem complir aquest paràmetre, la mida de la taula serà molt gran a mesura que treballem amb números primers molt grans. Per tant haurem de delimitar la taula depenent de la memòria disponible.

Implementació de la taula

La cerca per obtindre l'element desitjat de la taula o simplement per verificar si aquest hi és o no ha de ser eficient, es a dir, les cerques a la taula creada han de tenir un cost $\mathcal{O}(1)$. Una de les estràtegies possibles i que hem utilitzat en aquest treball és la implementacio mitjançant un hash obert amb una llista de nodes a cada posisció de la taula i una funció de disperció h:

$h=$ la clau (mod $Mida\ Taula$)
on la clau és igual a $y \cdot g^i$ i el valor associat és igual a i.

Càlcul del "pas de gegant" i búsqueda a la taula

Una vegada hem creat la taula i en sabem la seva mida hem de calcular el salt que realitzarem a cada iteració que serà igual a g^k .

Seguirem la estratègia que hem utilitzat en la cerca exhaustiva, i en comptes de calcular : g, g^k , g^{2k} , ... calcularem g^k, aleshores per calcular $g^{k \cdot i}$ l'obtindrem a partir de $g^k \cdot g^{k \cdot (i-1)}$.

Mirarem si el resultat és a la taula mitjançant el hash i si hi és ja podem obtenir la resposta. Si la solució és a la posició " i " de la llista i hem fet "j" salts de longitud "k" , aleshores la solució serà : $j \cdot k - i$, on k és la mida de la taula.

Avantatges i desavantatges

El principal avantatge és que millora el cost respecte a la cerca exhaustiva i és un algorisme senzill d'implementar, mentres que el principal desavantatge és que el cost continua sent exponencial i utilitza molta memòria per emmagatzemar la llista.

La mida òptima de la llista teòricament és $k = \sqrt{n}$, essent n l'ordre del grup G amb el que treballem. En les pràctiques implementades ens hem basat en el grup multiplicatiu $\mathbb{Z}_p^*$. Però a la pràctica la mida òptima de k pot venir marcada per les limitacions de la memòria RAM del hardware utilitzat.

Hem de tenir en compte que les cerques a la llista creada han de tenir un cost $\mathcal{O}(1)$.

El cost del algorisme en un grup d'ordre n en temps és $\mathcal{O}(\sqrt{n})$ i si n té una longitud de l bits, el temps de resolució és de $\mathcal{O}(exp(l/2))$, el cost és exponencial respecte a la longitud de la mida del grup.

5.1 Algorisme de Shanks en paral·lel

Aquest algorisme és idèntic a l'anterior, però aprofitarem la programació en paral·lel per distribuir la búsqueda en diferents processadors, així si tenim 8 processadors la búsqueda anirà 8 cops més ràpida que l'algorisme de Shanks sense paral·lelitzar. En la implementació del nostre algorisme cada procès té una còpia de la taula creada. Cada procès busca en un lloc diferent del grup, així si la longitud de la llista és k i el nombre de processos és 8, el primer procès buscara en la posició : g, $g^{8 \cdot k}$, $g^{16 \cdot k}$,... el segon procès buscarà en les posicions g^k, $g^{9 \cdot k}$,... i així succesivament, d'aquesta manera podem fer salts de longitud $k \cdot 8$ a cada iteració.

Hem de tenir en compte que la programació en paral·lel té un cost adicional degut la creació de tantes taules com processos i el cost d'intercanvi d'informació entre processos, de manera que la eficiencia mai serà d'un 100%. L'algorisme utilitzat és el següent per a un processador m d'un total de n processadors :

Algorisme (BSGS).proces(m,n):

Entrada : Un primer p, un generador g, un element y, una mida de la llista k $\in$ en $\mathbb{Z}_p^*$, un número de processador m, número total de processadors n.

Sortida : El logaritme discret $x = log_g y (mod p)$

1. Crear una llista que contingui les parelles$(y \cdot g^i, i)$, per a $0 \leq i < k$, on el primer valor és la clau

2. j=0

3. Si $g^{(n \cdot j + m) \cdot k}$ es troba a la tupla on el segon component de la parella és a la posició t

Retorna $(n \cdot j + m) \cdot k - t$

4. Sino, incrementa j i torna al pas (3)

La paral·letització en MPI ens ha originat el següent problema. No és suficient que un processador trobi la solució, sinó que un cop trobada, el processador ha de comunicar als altres processadors que ja s'ha obtingut la resposta per a que aturin els seus processos. Aquest control s'ha realitzat utilitzant una comunicació entre nodes no bloquejant, on els nodes fan una consulta, cada cert nombre d'iteracions del bucle, per saber si algun node ha trobat la solució i ha d'aturar-se o no, evidentment això tindra un cost que afectarà al rendiment de l'algorisme. Si el processador que ha trobat la solució és el procés pare aquest el mostra per pantalla, però si la solució la trobat un procès fill aquest l'envia al pare, ja que els fills no el poden mostrar.

6 Implementació

En aquest capítol veurem una descripció dels elements principals de software utilitzats per a la implementació dels algorismes, com per l'edició de la mèmoria o per l'execució de diferents proves. També farem una breu descripció dels fitxers de codi creats per l'implementació de l'algosime de Shanks paral·lelitzat.

6.1 Software utilitzat

6.1.1 El llenguatge de programació

El llenguatge de programació utilitzat en la creació de tots els fitxers de codi font és el C++, tant per implementar els algorismes esmentats en l'apartat anterior, com per implementar el hash i utilitzar les diferents llibreries per nombres grans i per la paral·lelització dels algorismes.

6.1.2 Els compiladors

- GNU g++ : és el compilador que s'ha utilitzar per compilar els fitxers de codi font escrits en C++ .
- mpiCC : és el compilador que va inclós a la llibreria MPI de la distribució utilitzada. L'utilitzem per compilar els fitxers de codi font que utilitzen aquesta llibreria.

6.1.3 Les llibreries

La llibreria NTL

NTL és una llibreria portable per a C++ , que ofereix algorismes i funcions per a la manipulació de números grans, vectors, matrius, polinomis,etc.

La part de la llibreria que hem utilitzat és la que es refereix a nombres enters grans, anomenada ZZ. Disposa de totes les rutines per a realitzar totes les operacions aritmètiques bàsiques i algunes de més avançades com les proves de primalitat.

La llibreria MPI

MPI o Message Passing Interface, és un estandard de programació en paral·lel mitjançant pas de missatges, que permet crear programes portables i eficients.

Existeixen implementacions de font oberta, que van permetre el desenvolupament de sistemes paral·lels de baix cost basats en programari lliure. La distritribució que hem utilitzat és la openmpi versió 1.2.5 i és de lliure distribució.

És una biblioteca que inclou interfícies per FORTRAN, C i C++.

L'estandard és extens pel que fa al nombre de rutines que s'especifiquen. Conté al voltant de 129 funcions moltes de les quals tenen nombroses variants i paràmetres. Molts programes paral·lelitzats poden ser escrits utilitzant només 6 funcions bàsiques :

MPI_Init() : Inicia la paral·lelització.

MPI_Finalize() : Finalitza la paral·lelització.

MPI_COMM_SIZE: Determina el número de processos del comunicador.

MPI_COMM_RANK: Determina el identificador del procés en el comunicador.

MPI_SEND: Enviament bàsic de missatge.

MPI_RECV: Recepció bàsica d'un missatge.

Per compilar en C++, normalment, es fan servir la comanda: mpiCC i el funcionament i paràmetres són molt similars al del compilador g++. Un exemple sería el següent :

```
mpiCC -o holamon holamon.cpp .
```

També tenim l'opció de construir archius Makefile. Les implementacions s'encarregaran de definir els mecanismes d'arrencada de processos; per a tals fins, algunes versions coincideixen en l'ús d'un programa especial anomenat mpirun. Aquí mostrem un exemple per a dos processos en paral·lel:

```
mpirun -np 2 -machinefile maquines holamon.
```

6.1.4 LaTeX

LaTeX és un llenguatge i un sistema per a preparació de documents, format per un gran conjunt de macros de TeX, escrites per Leslie Lamport (LamportTeX) el 1984, amb la intenció de facilitar l'ús del llenguatge de composició tipogràfica creat per Donald Knuth. És molt utilitzat per a la composició d'articles acadèmics, tesis i llibres tècnics, atès que la qualitat tipogràfica dels documents realitzats amb LaTeX és comparable a la d'una editorial científica de primera línia. LaTeX és programari lliure sota llicència LPPL.

TeX és un sistema de tipografia desenvolupat per Donald E. Knuth, molt popular en l'ambient acadèmic, especialment entre les comunitats de matemàtics, físics i informàtics. Tex és un llenguatge de baix nivell, ja que les seves comandes són molt elementals, en canvi LaTex és de més alt nivell.

6.1.5 LyX

LyX és un programa gràfic i multiplataforma creat per Matthias Ettrich que permet l'edició de textos utilitzant LaTeX. Hereta totes les seves capacitats (notació científica, edició d'equacions, creació d'índex, etc).

Es tracta d'un processador de textos en el que l'usuari no ha de pensar en el format final del seu projecte, sols en el seu contingut i la seva estructura (WYSIWYM) que significa "lo que veus és el que vols dir", així que pot ésser utilitzat per escriure grans documents amb un format rigurós amb facilitat.

6.2 Fitxers de codi creats

cerca.cpp

En aquest fitxer hem implementat la cerca exhaustiva per resoldre el problema del logaritme discret en un grup multiplicatiu $\mathbb{Z}_p^*$. Utilitzem la llibreria per a enters grans <NTL/ZZ.h>, de la qual destaquem la funció : ZZ PowerMod (const ZZ& g, const ZZ& x, const ZZ& p) per calcular $y = g^x (mod p)$.

La Classe Node

Representa els nodes que formen part del hash. Està formada pels archius node.h i node.cpp. El node es composa dels següents atributs :

Una parella clau, valor on es guarda la informació del node.

Un apuntador a node per apuntar al següent node de la llista.

Un senyal de marca per saber si el node està buit o ple.

La Classe Hash

Està formada pels arxius hash.h i hash.cpp. La implementació del hash està basada en l'estrategia d'un hash obert format per llistes encadenades del tipus node, éstà implementat en memòria dinàmica.

Podem inserir i consultar les parelles (clau,valor) amb un cost $\mathcal{O}(1)$.

La limitació més important és que el tamany màxim del hash ve demilitada pel tamany d'un nombre de tipus long en C++.

cercaHash.cpp

En aquest arxiu implementem la resolució del problema del logaritme discret mitjançant l'algorisme Baby Step - Giant Step. Creem una llista i l'emmagatzemem en un hash creat a partir de la Classe Hash esmentada anterioment.

Utilitzem la llibreria per enters grans <NTL/ZZ.h>.

cercaHashMpi.cpp

En aquest arxiu implementem la resolució del problema del logaritme discret mitjançant l'algorisme Baby Step - Giant Step. Seguim la mateixa estratègia que en l'arxiu anterioment esmentat, però ara introduïm el concepte de la programació en paral·lel utilitzant la llibreria "mpi.h" . L'objectiu és reduïr el temps de càlcul mijançant la utilització de diferents computadors.

generador.cpp

És un generador de números primers a partir d'una entrada que és la longitud en bits del nombre primer p que volem generar, ens genera de forma aleatoria un p primer que és l'ordre del cos multiplicatiu, un g generador, una clau pública y i una clau privada x.

7 Resultats i comentaris

Un cop feta la implementació de l'algorisme de Shanks, tant de manera seqüencial com la implementació en paral·lel, és el moment de posar-los a prova i determinar si la paral·lelització de l'algorsime ha donat els seus fruits. Per determinar-ho, hem realitzat un conjunt de proves. La longitud dels nombres primers s'ha escollit de tal manera que els resultats siguin significatius i que les proves no tinguin un temps d'execució massa llarg. Cada experiment ha estat executat tant en la implementació de l'algorisme seqüencial com el paral·lel i per a cada longitud del nombre primer escollit s'han fet diverses proves. S'han fet proves amb diferents números de processadors,com també amb diferents mides de les taules on es guada l'informació de la llista dels "passos de nen" de l'algorisme de Shanks i diferents mides de hash.

Les execucions de l'algorisme de Shanks han estat realitzades sobre el grup multiplicatiu $\mathbb{Z}_p^*$.

7.1 Resultats i càlculs

Els principals càlculs que hem de realitzar per treure conclusions sobre el rendiment de l'algorisme de paral·lelització són el speedup i l'eficiència. El speedup és el temps en execució seqüencial dividit pel temps d'execució en paral·lel. Mentre que l'eficiencia és el SpeeUp dividit pel nombre de processadors i s'expresa en % .

En el primer bloc de proves hem establert una mida de la llista de 10000 parelles (clau, valor) enmagatzemat en un hash de 5000 posicions. Els resultats són els següents, el temps està expresat en segons.
En aquesta taula és mostra els temps mitjos de les execucions segons el número de bits del primer p i el nombre de processadors

que intervenen en l'execució.
Temps mitjos obtinguts

	Nº processadors			
Nº de bits de p	1	2	4	8
34	2,42	1,68	1,67	1,43
36	4,98	3,28	2,66	1,69
38	22,85	15,58	8,22	4,55
40	87,83	58,6	29,96	15,97
42	135,17	90,6	45,31	24,12

Resultats dels speedup obtinguts

	Nº processadors		
Nº de bits de p	2	4	8
36	1,51	1,87	2,94
38	1,46	2,78	5,02
40	1,49	2,93	5,5
42	1,49	2,98	5,6

Resultats de les eficiències obtingudes

	Nº processadors		
Nº de bits de p	2	4	8
36	76%	46%	36%
38	73%	69%	62%
40	75%	73%	69%
42	75%	74%	70%

En el segon bloc de proves hem establert una mida de la llista de 100000 parelles (clau, valor) enmagatzemat en un hash de 50000 posicions. Els resultats són els següens, el temps està expresat en segons.
En aquesta taula és mostra els temps mitjos de les execucions segons el número de bits del primer p i el nombre de processos que intervenen en l'execució.

Temps mitjos obtinguts

	Nº processadors			
Nº de bits de p	1	2	4	8
36	1,24	0,86	0,76	1,26
38	2,58	1,82	1,27	1,64
40	29,91	19,15	10,21	6,72
42	23,26	16,99	8,58	5,65
44	215	142,04	71,5	43

Resultats dels speeup obtinguts

	Nº processadors		
Nº de bits de p	2	4	8
38	1,41	2,03	1,57
40	1,56	2,93	4,45
42	1,37	2,71	4,11
44	1,51	3	5

Resultats de les eficiències obtingudes

	Nº processadors		
Nº de bits de p	2	4	8
36	71%	51%	20%
38	78%	73%	55%
40	68%	68%	51%
42	76%	75%	62%

7.2 Conclusions

Podem apreciar segons els resultats obtinguts que l'eficiència de l'algorisme de Shanks paral·lelitzat en MPI és d'un 75%, això és degut a que s'utilitza un temps en l'intercanvi de missatges entre processos. Aquest algorisme està implementat mitjançant una comunicació no bloquejant i els processos fan una consulta per saber

si han rebut un missatge cada 1000 iteracions. Hem observat que modificant aquest valor els temps d'execució canvien. Una possible millora en l'eficiència de l'algorsime seria ajustar al màxim aquest valor en funció de la longitud del nombre primer en el qual es busca la solució.

8 Bibliografia

[1] J.J. Ortega, M.A. López i E.C. Garcia. *Introducción a la Criptografía*. Ediciones de la Universidad Castilla-La Mancha,2006.

[2] J. Gimbert i J. M. Miret. *Problemes d'Algebra per a ciències de la computació*. Edicions de la Universitat de Lleida,1997.

[3] Henk C.A. van Tilborg. *Enciclopedia of Cryptography and Secutity*. Eindhoven University of Technology The Netherlands,2005.

[4] J. Pastor i M.A. Sarasa. *Criptografía digital Fundamentos y aplicaciones*. Prensas Universitarias de Zaragoza,1998.

[5] Douglas R.Stinson. *Cryptography Theory and Practice*. Chapman and Hall/CRC,2006.

[6] J.A.Gras. *Ataques al problema del logaritmo discreto y estudio de su seguridad en protocolos criptográficos modernos*. TFC, Universitat de Lleida,2003.

[7] **MPI**. http://www.mcs.anl.gov/research/projects/mpi/

[8] **MPI**. http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf

[9] **NTL**. http://www.shoup.net/ntl/

[10] **LaTeX**. http://www.fceia.unr.edu.ar/lcc/cdrom/Instalaciones/LaTex/latex.html

[11] **LyX**. http://www.lyx.org/