En utvärdering av VIFF i praktiken

SAMUEL LIDÉN BORELL
och JOEL PETTERSSON

KTH Datavetenskap
och kommunikation

Examensarbete
Stockholm, Sverige 2010

En utvärdering av VIFF i praktiken

S A M U E L L I D É N B O R E L L
o c h J O E L P E T T E R S S O N

Examensarbete i datalogi om 15 högskolepoäng
vid Programmet för datateknik
Kungliga Tekniska Högskolan år 2010
Handledare på CSC var Mikael Goldmann
Examinator var Mads Dam

URL: www.csc.kth.se/utbildning/kandidatexjobb/datateknik/2010/
liden_borell_samuel_OCH_pettersson_joel_K10046.pdf

Kungliga tekniska högskolan
Skolan för datavetenskap och kommunikation

KTH CSC
100 44 Stockholm

URL: www.kth.se/csc

Referat

Det har snart gått 30 år sedan teorin kring *Secure Multiparty Computation* föddes genom Yao:s klassiska exempel med två miljonärer som ensamma ville ta reda på vem av dem som var rikast – utan att avslöja något om sina tillgångar för varandra. Fram till för bara ett par år sedan har ämnet uppehållit sig just inom teorin, men i och med en nyligen initierad, fullskalig tillämpning har Secure Multiparty Computation tagit första steget ut i verkligheten. Ur detta nyligen initierade projekt utvecklades så småningom ramverket VIFF, *Virtual Ideal Functionality Framework*, i syfte att erbjuda en utökningsbar grund att bygga nya, liknande tillämpningar på. Målet med vårt projekt är att utvärdera det ramverket utifrån ett praktiskt perspektiv. För att ta reda på om VIFF är tillräckligt användbart och prestandastarkt skrev vi ett eget program som utnyttjar ramverket och jämförde detta med en alternativ implementation. Våra resultat visar att det VIFF-baserade programmet förvisso är långsammare, men ramverket har andra styrkor så som god dokumentation och en bra abstraktionsnivå.

Abstract

An evaluation of VIFF in practice

It has been almost thirty years since the *Secure Multiparty Computationn* theory was founded on Yao's classic example, which describes two millionaires who wanted to settle which one of them was the wealthiest — without giving anything away about their own assets. Until just a couple of years ago this subject has remained theoretical, but a recently initiated, large-scale application has taken Secure Multiparty Computation into the real world. From this application the framework VIFF, *Virtual Ideal Functionality Framework*, evolved, aiming at offering an extensible foundation to build new, similar applications upon. The purpose of our project is to evaluate this framework from a practical perspective. To be able to tell whether or not VIFF is sufficiently usable and performes well enough, we wrote an own program based on the framework and compared this with an alternative implementation. The results that we obtained show that the program based on VIFF assuredly was slower, but the framework does also have strengths such as good documentation and a fine level of abstraction.

Innehåll

Kapitel 1

Inledning

I denna rapport presenteras ett examensarbete inom datalogi som utförts på Kungliga Tekniska Högskolan i Stockholm. Examensarbetet behandlar delar inom kryptografin, närmare bestämt området *Secure Multiparty Computation* med inriktning mot praktisk tillämpning av detta. Den praktiska tillämpningen bygger på ramverket *Virtual Ideal Functionality Framework* som tillhandahåller en plattform – i programmeringsspråket Python – för beräkningar som är förenade med det i rapporten behandlade området.

1.1 Viktiga begrepp

SMC (Secure Multiparty Computation) En typ av beräkningsproblem där ett givet antal parter tillsammans vill bestämma värdet av en gemensam funktion. Funktionen kräver indata från var och en av deltagarna, men ingen av dessa är villig att avslöja sin information för någon annan.

VIFF (Virtual Ideal Functionality Framework) Ett ramverk skrivet i programmeringsspråket Python för programmering av lösningar till SMC-problem.

Twisted Ett Python-ramverk för nätverksprogrammering, vilket bland annat kan användas för att implementera klient- och serverprogram.

Deltagare (eller part) En organisation eller person som bidrar med indata till ett beräkningsproblem. En deltagare är inte nödvändigtvis pålitlig, utan kan också vara en så kallad motståndare.

Körning En enskild exekvering av ett program. Som en del av arbetet har upprepade körningar gjorts på flera olika program för att mäta körtiderna för programmen.

Motståndare En motståndare (eng. *adversary*) är ett teoretiskt ondsint väsen vars syfte är att hindra deltagare i ett kryptosystem från att nå sina mål, vilket i SMC huvudsakligen gäller sekretess och korrekthet i en beräkning.

En motståndares drag kan alltså handla om att försöka komma över andras privata information eller agera för att förvanska resultatet av beräkningen.

Pålitlig tredje part (eng. *Trusted Third Party, TTP*) En tredje part i en beräkning, med minst två andra parter, som utför alla beräkningar korrekt och förser alla andra parter med den information de ska ha. En pålitlig tredje part kan exempelvis implementeras som en central server.

Tröskelvärdesschema En metod för att dela upp ett informationsstycke så att det krävs ett visst antal delar för att kunna återskapa det. Utan det krävda antalet delar går det inte att härleda något som helst om det ursprungliga informationsstycket, härav begreppet *tröskelvärde*.

Distribuerad hashtabell En distribuerad hashtabell är en lista som lagras decentraliserat över flera klienter, där varje listelement tilldelas ett hashvärde. Denna typ av datastruktur tillåter snabb uppslagning av de element som har ett visst hashvärde. Exempelvis används distribuerade hashtabeller i fildelningsprotokollet BitTorrent för att hitta klienter som har delar av en fil man önskar ladda ner. Där beräknas hashvärdena utifrån filernas innehåll och elementen är IP-adresser till de klienter som har någon del av motsvarande fil.

UPnP En uppsättning nätverksprotokoll avsedda för att underlätta ihopkoppling av datorer i nätverk, både i hemmet och i företagsmiljöer. UPnP dras dock med vissa säkerhetsproblem som gör det olämpligt att användas i större sammanhang där säkerheten är viktig [7].

1.2 Bakgrund

Den senaste tidens utveckling av Internet och även större lokala nätverk har gjort att det idag är möjligt för olika parter oberoende av geografisk position, att utföra gemensamma beräkningar tillsammans. I många sådana situationer finns det ingen anledning att dölja den ingående datan för varandra. Dessa typer av beräkningar kan exempelvis röra befolkningsantal eller länders BNP[1]. I exemplet med BNP motsvaras ingående data av de enskilda företagens omsättning, vilken inte behöver skyddas då den ändå är tillgänglig offentligt.

Det är å andra sidan också tänkbart med situationer där de inblandade parterna inte betror varandra med den indata som krävs från var och en för att beräkningen ska gå att genomföra. Hit hör till exempel resultatberäkningen av ett val där ingen av deltagarna vill avslöja vem de har röstat på, men alla vill veta vem som har erhållit flest röster. Detta samtidigt som en kontroll görs av att endast röstberättigade personer deltar och då med högst en röst vardera.

Problemet med rösträkning löses i praktiken genom att en myndighet får utföra själva beräkningarna. En sådan myndighet – i Sverige kallad Valmyndigheten – är

[1]Bruttonationalprodukten summerar det totala värdet på ett lands produktion av varor och tjänster, vanligtvis under loppet av ett år.

exempel på en så kallad pålitlig tredje part (eng. *Trusted Third Party, TTP*) som kan användas för att lösa beräkningsproblem av denna karaktär. Dock är det inte alltid rimligt att lösa sådana problem på det viset. Flera anledningar är tänkbara, men mest uppenbart är exempelvis att ingen pålitlig tredje part kan utses eller att en sådan skulle vara alltför kostsam. Det är bland annat i situationer likt dessa som Secure Multiparty Computation (SMC) får en mycket intressant praktisk tillämpning. Med SMC kan nämligen den pålitliga tredje parten helt avskaffas.

1.2.1 Aktuellt forskningsläge

Forskning inom området har pågått ända sedan början av 1980-talet då problemet för första gången framlades av Yao [13]. Sedan dess har teorin utvecklats i olika riktningar, men utan att SMC har fått något större praktiskt genomslag [3], trots många teoretiska bevis av kraftfulla egenskaper. Till exempel visar Ben-Or et al. [1] och Chaum et al. [4] hur alla problem som en Turing-maskin kan programmeras att lösa också kan beräknas med SMC. Vegge [11, s. 70] identifierar dålig prestanda, funktionalitet och skalbarhet som betydande hinder hittills.

Den första storskaliga och praktiska tillämpningen av SMC genomfördes inte förrän 2008 av forskningsprojektet SIMAP [3]. Projektet syftade till att lösa ett nationellt problem i Danmark rörande produktion av sockerbetor och försäljningen av dessa till Nordic Sugar (tidigare Danisco) – landets enda tillverkare av socker från sockerbetor. Den danska sockerbetsproduktionen regleras genom kontrakt med Nordic Sugar och är föremål för handel bönderna emellan. Tidigare har endast parvisa sådana utbyten skett trots vetskapen om att en central marknad skulle öka den totala vinningen [9]. Bidragande orsaker till detta har varit ömsesidig misstro och ovilja att offentliggöra sina bud [3], vilket är två problem som kan lösas med hjälp av SMC.

I januari 2008 användes så en applikation sprungen ur SIMAP som effektivt löste problemet med produktionsrättigheterna [9]. Samma beräkning upprepades därefter under sommaren 2009 – då med VIFF som grund. Beräkningen som omfattade produktionsrättigheter motsvarande 250 000 ton sockerbetor behandlades under loppet av ungefär 15 minuter på tre bärbara datorer sammanlänkade i ett lokalt nätverk. Kontrakten kunde således utbytas till optimala priser utan att någon part behövt lämna ut känslig information eller att en förmodat dyr pålitlig tredje part anlitats.

1.2.2 Syfte

Med bakgrund av föregående avsnitt kan VIFF betraktas som funktionsdugligt; ramverket tillåter utveckling av ett program som löser beräkningsproblem av typen SMC. Precis som andra ramverk – oavsett område och miljö – tjänar VIFF till att erbjuda ett abstraktionslager, i detta fall bland annat för att kapsla in den bakomliggande tillämpade matematiken. Förhoppningsvis kan detta leda till att fler problem kan lösas med hjälp av SMC utan ingående teoretiska kunskaper i ämnet.

Examensarbetet som denna rapport beskriver syftar till att reda ut om VIFF är tillräckligt användbart och prestandastarkt för att det ska vara möjligt.

1.2.3 Sammanhang

Om ett ramverk som VIFF visar sig vara till praktisk nytta och det får till följd att fler får upp ögonen för SMC kan det tänkas få direkta samhälleliga konsekvenser. Antag till exempel att utvecklingen går så långt att fullt datoriserade val baserade på SMC införs runt om i världen. Då skulle ingen behöva bekymra sig för att landets valmyndighet utövar valfusk, vilket indirekt skulle innebära en stärkt demokrati i dessa länder. Detta förutsätter givetvis att systemets utvecklare och operatörer är pålitliga, vilket förslagsvis skulle kunna främjas av transparens så som öppen källkod.

Trots den lovvärda utveckling av SMC som ovan givna framtidsvision antyder är det först nyligen som implementationer av SMC för verklighetsrelaterade problem har börjat diskuteras i tidigare forskning. Flera av dessa återges i kortform under 1.4. För att ta ett exempel behandlar Vegge [11] i sin avhandling teorin för både SMC och VIFF samt implementerar även ett par exempelprogram, men utan att gå närmare in på detaljer kring ramverkets praktiska nytta.

1.3 Problemformulering

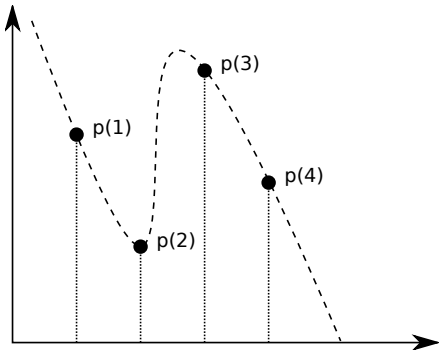
Vi avser jämföra VIFF med den traditionella lösningen med en central server som en pålitlig tredje part. Jämförelsen fokuserar på utveckling av tillämpningar utifrån ett praktiskt perspektiv, där tonvikten läggs på användbarheten för utvecklaren, i form av enkelhet och effektivitet samt prestandan hos de färdiga implementationerna.

1.4 Litteratursammanfattning

1.4.1 How to Share a Secret

Shamir [8] beskriver en mycket fundamental del av SMC, nämligen så kallade tröskelvärdesscheman (eng. *threshold scheme*). I vad som kallas ett (k,n)-tröskelvärdesschema delas ett informationsstycke upp i n delar, där innehavet av k delar är tillräckligt för att kunna återskapa informationsstycket. Innehavet av $k-1$ delar, eller färre, ger däremot ingen information *alls* om informationsstycket.

Det tröskelvärdesschema som Shamir presenterar bygger på polynom av grad $k-1$ i en ändlig kropp, där delarna och det hemliga värdet utgörs av punkter på polynomet. Detta illustreras i Figur 1.1. Som känt kan man återskapa ett polynom om och endast om minst k punkter är kända, genom interpolation. När polynomet har återskapats kan man erhålla det hemliga värdet. Detta utgör grunden till schemat.



Figur 1.1. Ett tredjegradspolynom som har interpolerats från fyra punkter. I Shamirs tröskelvärdesschema hade de fyra delarna utgjorts av de markerade punkterna, och det hemliga värdet hade utgjorts av konstanttermen (d.v.s. polynomvärdet i $p(0)$).

1.4.2 Realizing Secure Multiparty Computation

I sin masteruppsats beskriver och implementerar Vegge [11] två tillämpningar i VIFF – rankning och röstning. Den förstnämnda tillämpningen låter författarna till en artikel rangordna varandra efter arbetsinsats för att erhålla en ordnad lista med alla författare sorterade efter arbetsinsats. Den andra tillämpningen är ett webbaserat system för anonym röstning.

Utöver tillämpningarna återger Vegge [11] även teorin bakom tröskelvärdesscheman och SMC. Då tröskelvärdesscheman redan har beskrivits tidigare sammanfattar vi endast SMC-delen av hans verk. Vegge delar in SMC i tre faser: inmatning, beräkning och avslut. I inmatningsfasen väljer deltagarna Shamir-polynom med slumpmässigt valda koefficienter, men med sin egen hemliga indata som konstantterm. Därefter skickar varje deltagare ut unika punkter till de andra deltagarna, som nu skulle ha möjlighet att återskapa polynomet (och därmed även den hemliga indatan) om tillräckligt många av dem utbytte punkter med varandra. Detta görs dock inte förrän i avslutsfasen; först gör deltagarna beräkningar på datan i beräkningsfasen. Denna fas beskrivs inte här, men Vegge visar emellertid hur addition och multiplikation kan utföras av deltagarna.

1.4.3 Sannolikhetsteori och statistikteori med tillämpningar

I denna bok behandlar Blom et al. [2] grunderna till sannolikhets- och statistikteorin. Boken täcker många olika områden inom dessa ämnen, men i vårt arbete har vi främst använt oss av kapitel 12 om intervallskattning. En intervallskattning innebär att man givet ett antal mätvärden, med okända störningar, bestämmer ett intervall som det sanna värdet med en viss sannolikhet ligger i. Ett sådant intervall kallas ett konfidensintervall och sannolikheten att värdet ligger på intervallet kallas konfidensgraden. Vi utnyttjar också stora talens lag som säger att “det aritmetiska medelvärdet av flera oberoende s.v. [stokastiska variabler] med samma väntevärde

μ ligger nära μ, bara antalet är tillräckligt stort" [2, s. 128].

1.5 Dokumentets struktur

Rapporten är, utöver denna inledning, indelad i följande kapitel:

- **Kapitel 2: Översikt av VIFF**
  I detta kapitel beskrivs VIFFs funktion och förutsättningar. Vidare presenteras arkitekturen som en utvecklare som vill använda VIFF måste följa.
- **Kapitel 3: Metod**
  Här beskrivs bland annat hur vi valde tillämpning att implementera och hur vi erhöll mått på användbarheten och prestandan. Vidare ges webblänkar till vår källkod i slutet av detta kapitel.
- **Kapitel 4: Resultat**
  Detta kapitel innehåller våra resultat i form av tabeller och grafer, samt kommentarer till dessa.
- **Kapitel 5: Diskussion**
  Här diskuteras VIFF i förhållande till en pålitlig tredje part (i form av en central server) utifrån de resultat vi erhöll.
- **Appendix A: Datorspecifikationer**
  Här redovisas specifikationerna för datorerna som testkörningarna utfördes på.

Kapitel 2

Översikt av VIFF

VIFF innehåller komponenter som är tänkta att förenkla skapandet av program vars instanser kan kommunicera med varandra i syfte att utföra en säker gemensam beräkning – Secure Multiparty Computation. Med säker menas i detta avseende att sådana program uppfyller något som kallas för en *Virtual Ideal Functionality*, vilket i själva verket är en virtuell pålitlig tredje part. Tanken är att ett VIFF-baserat programs beteende ska vara omöjligt att särskilja från program som använder sig av en verklig pålitlig tredje part. Om så är fallet kommer dessa båda att vara säkra precis samtidigt.

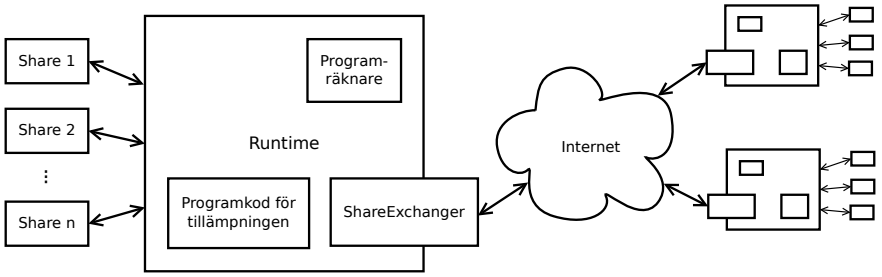
Låt oss därför betrakta situationen med en verklig pålitlig tredje part. Varje deltagare skickar sin indata till denna via en säker kanal. När all indata har tagits emot utförs beräkningen varpå resultatet skickas ut till var och en av deltagarna, återigen via säkra kanaler. Eftersom den pålitliga tredje parten som tar emot indatan och utför beräkningen per definition inte är korrupt måste detta protokoll vara säkert. Således kommer även protokollet med en virtuell pålitlig tredje part – vilket utnyttjas i VIFF – att vara säkert.

Själva ramverket VIFF är dock endast säkert under vissa förutsättningar, vilket innefattar att en rad allmänna säkerhetsaspekter är uppfyllda. Bland dessa kan nämnas att:

- Motståndaren får bara vara kapabel att korrumpera upp till en viss andel av deltagarna. Hur stor andelen är varierar beroende på vilken typ av problem som beräknas, men i regel gäller att majoriteten måste vara ärlig.

- Motståndaren får inte ha obegränsade beräkningsresurser. Detta eftersom de bakomliggande protokollen förlitar sig på att vissa problem kräver mycket stora beräkningsresurser för att lösa och alltså inte är informationsteoretiskt säkra (eng. *information theoretic secure*).

- Motståndaren är antingen passiv eller aktiv, beroende på vilket protokoll som används En passiv motståndare kan övervaka nätverkstrafiken men följer protokollet, medan en aktiv dito kan avvika från det aktuella protokollet.



Figur 2.1. En förenklad modell av hur VIFF fungerar. Hos varje deltagare finns ett `Runtime`-objekt som utökats med tillämpningsspecifik kod.

2.1 Arkitektur

En mycket central del i VIFF är `Runtime`-klassen, som synes i Figur 2.1. Den abstraherar bort all nätverkskommunikation och tillhandahåller grundläggande funktionalitet för att dela information mellan deltagarna. Varje deltagare skapar sitt eget `Runtime`-objekt som i sin tur skapar dubbelriktade kommunikationskanaler till de övriga deltagarna. Dessa kanaler är asynkrona i förhållande till varandra och dessutom kan oberoende beräkningar utföras parallellt. För att denna form av kommunikation ska vara möjlig använder VIFF så kallade *programräknare* (eng. *program counters*), som används för att tilldela varje beräkning ett unikt nummer som skickas med varje gång data skickas mellan deltagarna.

Strukturen på ett typiskt VIFF-program är att det först skapas ett `Runtime`-objekt som sedan anropar en funktion som utför de tre faserna i SMC: inmatning, beräkning och avslut. Det ska dock poängteras att det är möjligt att lösa delproblem inuti ett större SMC-problem, detta hanterar VIFF genom att skapa en ny programräknare för att kunna spåra delproblemets beräkningar. Metoderna som används i inmatningsfasen implementeras inte av `Runtime`-klassen utan av subklasser till den, och i denna rapport kommer vi att utgå från subklassen `PassiveRuntime`.

I inmatningsfasen kan metoden `shamir_share()` användas för att göra hemlig indata tillgänglig för beräkningar enligt Shamirs metod som beskrevs i litteratursammanfattningen. Denna metod returnerar `Share`-objekt som implementerar ett fåtal matematiska operationer i en kropp, vars storlek kan styras genom en parameter till `shamir_share()`-metoden. Objekten översätter operationerna på dem helt transparent till motsvarande operationer med SMC. Detta sker dessutom asynkront för att öka prestandan hos VIFF.

Till sist, när det slutliga resultatet av beräkningarna ska erhållas kan metoden `open()` anropas på `Runtime`-objektet med de `Share`-objekten som svarar mot resul-

tatet. När denna metod anropas hos deltagarna delas deras aktuella `Share`s ut till de andra deltagarna, och när lika många deltagare som krävs av tröskelvärdesschemat har gjort anropet kan resultatet beräknas genom att använda Shamirs metod med polynominterpolation. Detta görs automatiskt av `open()`-metoden.

2.2 Begränsningar

2.2.1 Beslutsfattning under körning

Eftersom `Share`-objekten implementerar ett tröskelvärdesschema går det inte att härleda någon som helst information från dem förrän de har öppnats. Detta får även konsekvensen att det inte går att fatta beslut baserat på beräkningarnas resultat under programmets körning, om man vill behålla resultatvärdena hemliga. VIFF implementerar emellertid en jämförelseoperator som har definierats till att ha värdet 0 eller 1 beroende på utfallet. Denna operator kan simulera en beslutsfattning genom att både den kod som skulle köras vid sant respektive falskt utfall tillåts köra, men att resultaten adderas enligt följande:

$$b \cdot r + (1 - b) \cdot s.$$

Här är r och s resultaten från koden för respektive utfall, och b värdet från operatorn (som antingen är 0 eller 1). Summan blir alltså antingen r eller s, beroende på operatorvärdet.

2.2.2 Kontakt mellan deltagare

Precis som i alla andra decentraliserade system på Internet, behöver deltagarna i ett SMC-problem hitta varandras IP-adresser för att kunna kontakta varandra. Denna funktionalitet finns tyvärr inte inbyggd i VIFF. En enkel lösning på problemet är att införa en central server som förser varje deltagare med en lista över de övriga deltagarna. Om det inte är möjligt finns det decentraliserade lösningar såsom att använda en distribuerad hashtabell (eng. *Distributed Hash Table*). Båda dessa lösningar medför tyvärr vissa säkerhetsproblem som måste beaktas. Med en central server vilar säkerheten onekligen i serverägarens händer och goda vilja. Men även en distribuerad lösning behöver analyseras utifrån de säkerhetskrav man ställer. Urdaneta et al. [10] sammanfattar ett antal problem med just distribuerade hashtabeller. Som tur är ger VIFF ett skydd mot eventuella motståndare som utger sig för att vara någon av deltagarna genom användning av certifikat. Detta löser dock inte de fall där motståndaren hindrar en deltagare från att delta, vilket skulle kunna utnyttjas i problem där deltagandet är frivilligt men påverkar resultatet, såsom i omröstningar.

Kapitel 3

Metod

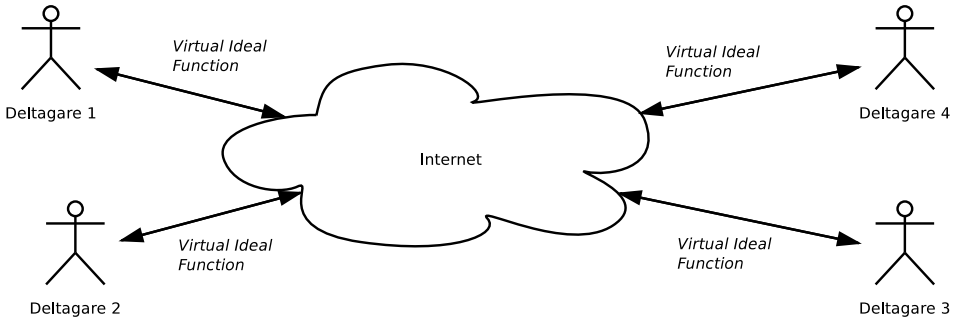
Vårt syfte med arbetet är, som tidigare nämnts, att jämföra användningen av VIFF med en verklig pålitlig tredje part i form av en central server. För att kunna göra dessa jämförelser implementerade vi ekvivalenta program i VIFF och som en central server. Vi jämförde dels prestandan (körtiden) hos programmen och dels användbarheten för oss som utvecklare. Med användbarhet avser vi hur mycket allmän logik som ramverket bidrar med och som kan utnyttjas av utvecklaren för att öka sin produktivitet. Detta bör avspegla sig i antalet rader kod som denna behöver skriva, vilket vi också mätte.

3.1 Val av tillämpning

Vi valde att låta funktionen som skulle beräknas vara relativt enkel, bland annat för att inte riskera att råka ut för onödiga fördröjningar i tidsplanen. Tillämpningen som vi alltså implementerade på två olika sätt består av beräkningar av summor – mer specifikt medelvärden. Ett möjligt scenario är att en grupp personer vill beräkna sin genomsnittliga längd eller vikt utan att avslöja sin egen längd eller vikt för någon annan. Problemet involverar endast en elementär operation, addition, som finns inbyggd i VIFF. Divisionen kan göras i efterhand utan någon hemlig data.

3.1.1 VIFF

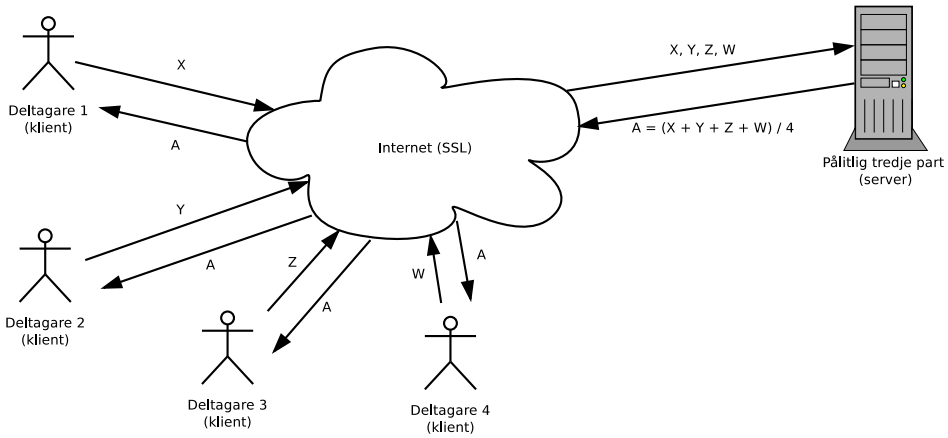
Tillvägagångssättet i vår implementation med VIFF är, på ett övergripande plan, det följande. Till att börja med delar varje deltagare ut portioner (`Share`-objekt) av sina hemliga tal till de övriga. Var och en summerar därefter ihop sammanhörande delar. Sedan kombineras alla portioner ihop igen för att på det viset öppna de resulterande gemensamma summorna. Slutligen utför var och en av deltagarna divisionerna av summorna med deltagarantalet (som inte är hemligt), vilket ger medelvärdena. I Figur 3.1 illustreras hur deltagarna kommunicerar direkt med varandra utan någon tredje parts inblandning, vilket förklarar varför hanteringen av de hemliga talen som distribuerade portioner är nödvändig. I detta arbete användes VIFF version 1.0, vilket i skrivande stund är den senaste versionen.



Figur 3.1. Deltagarna kommunicerar direkt med varandra över säkra kanaler. Tillsammans följer de ett protokoll som tillåter säker beräkning av en gemensam funktion genom en virtuell tredje part.

3.1.2 Central server

Vår referensimplementation involverar en central server som agerar pålitlig tredje part. Förfarandet i denna modell beskrivs av Figur 3.2, som visar hur var och en av deltagarna skickar in sin hemliga indata till en och samma server – den pålitliga tredje parten. Det är hos denna som själva beräkningarna sker, vilka i vårt fall är uträkningarna av summorna. När beräkningarna är klara skickas resultatet ut till var och en av deltagarna.



Figur 3.2. Deltagarna lämnar sin hemliga indata till en pålitlig tredje part som sedan beräknar och delar ut resultatet till var och en.

Servern och motsvarande klient implementerades i samma programmeringsspråk och ramverk som VIFF är implementerat i, nämligen Python respektive Twisted.

3.2 Mätningar

3.2.1 Användbarhet

Användbarheten uppmättes genom att jämföra antalet *fysiska* rader kod som vi själva var tvungna att skriva för att få de båda implementationerna (med VIFF respektive en verklig pålitlig tredje part) att fungera. Varje rad i ett programs källkod som varken är tom eller enbart utgör en kommentar kan, något förenklat, sägas vara en fysisk kodrad.

Vi valde att använda verktyget **SLOCCount**[1] för att räkna antalet rader kod, vilket med standardinställningar följer definitionen av *fysiska kodrader* som ges i Park et al. [6, s. 60–65]. Den kan sammanfattas med att en fysisk kodrad innehåller minst ett tecken som varken är ett tomrums- (eng. *whitespace*) eller kommentartecken samt att raden slutar med ett nyrads- eller slut-på-fil-tecken [12].

3.2.2 Prestanda

För att mäta prestandan skrev vi ett separat program som körde våra prototypprogram och mätte körtiden (skillnaden mellan tidpunkten före och efter körningen) hos varje deltagare. En instans av detta program kördes även på en separat server, som koordinerade körningarna hos deltagarna. Med den uppställningen erhölls separata mätvärden från varje deltagare, vilket motverkade systematiska fel i mätningarna. Noteras bör dock att separationen av mätvärdena även är nyttig i problem där deltagarna har olika roller, då mätvärdena har olika betydelse.

Med ovanstående mätmetod ingår även starttid för programmen och framförallt de externa komponenter som utnyttjades, bland annat VIFF och Twisted. Men i vissa tillämpningar är det troligt att deltagarna låter sina program köra hela tiden eller i långa perioder, inte minst om samma sorts problem ska lösas regelbundet med olika indata. Därför mätte vi även körtiden då inget problem alls löstes. Detta avspeglar starttiden, som kan räknas bort i de tillämpningar där den är av liten betydelse. Vidare har processorns användningstid mätts för att ge en bild av hur programmens processoranvändning förhåller sig till antalet indatavärden.

För att undvika att slumpmässiga fel vid enskilda tillfällen skulle påverka vårt resultat gjorde vi ett flertal körningar, och erhöll ett medelvärde. Vidare beräknade vi ett konfidensintervall, med 95 procents konfidensgrad, för våra mätvärden. I den beräkningen antog vi att mätvärdena var oberoende och normalfördelade.

Körningarna utfördes på datorer med de specifikationer som beskrivs i Bilaga A. Mätningarna av körtiden gjordes med funktionen `time()` i `time`-biblioteket[2] i Python, vilket använder funktionen `gettimeofday()`[3] som tidkälla under UNIX och liknande operativsystem. Den sistnämnda funktionen får i sin tur sin data[4] från

[1] `http://www.dwheeler.com/sloccount/`

[2] `http://docs.python.org/library/time.html`

[3] Se manualsidan för gettimeofday, som bland annat ingår i paketet manpages-dev i Ubuntu.

[4] `http://lkml.org/lkml/2008/4/10/172`

processorns så kallade Time Stamp Counter[5], som har en upplösning motsvarande processorns klockfrekvens, men avrundar denna information till närmaste mikrosekund. Vidare mätte vi processorns användningstid med kommandot `time` på dator 3, som visar sekunder och hundradelar. Detta gav körtidsmätningarna en precision på en mikrosekund, och mätningarna av processorns användningstid en precision på 10 millisekunder.

Svårigheter

Det finns vissa generella problem vid prestandamätningar av datorprogram, och mer specifikt program som kommunicerar över Internet. Främst gäller att överföringshastigheter och svarstider mellan datorer på Internet kan variera med tiden. Vidare gäller att starttiden för ett program på en dator varierar beroende på om programmet redan finns inläst i datorns primärminne (beroende på cachning [5, s. 383 – 401]). Normalt gäller att ett program stannar kvar i primärminnet efter att det har körts, men då primärminnet är begränsat kan operativsystemet frigöra programmet från minnet till fördel för något annat program.

Problemet med varierande prestanda på Internet kan avhjälpas genom att körningarna utförs på ett sådant sätt att programmen som ska jämföras körs i genomsnitt under lika förutsättningar. Genom att upprepade gånger köra programmen växelvis kommer genomsnittet för de båda programmen att närma sig genomsnittet för hela perioden enligt stora talens lag. Alltså närmar sig förutsättningarna (överföringshastigheterna och svarstiderna) för programmen samma värden, och därmed är problemet löst. Som en avvägning mellan tidsåtgång och precision valde vi att göra 10 körningar per program och konfiguration.

Å andra sidan är det tänkbart att operativsystemet under körningen av ett program frigör delar av andra program från primärminnet. Som följd skulle det efterföljande programmet starta långsammare, vilket skulle kunna snedvrida resultaten. Eftersom detta problem kan uppstå vid växelvis körning, föregicks varje körning av en kort körning med minimal indata, för att tvinga operativsystemet att läsa in programmet i primärminnet. Genom att dessa två metoder kombinerades bör resultaten inte ha förvanskats av de nämnda svårigheterna.

3.3 Källkod

Källkoden till våra implementationer och mätprogram med tillhörande hjälpprogram och dokumentation finns tillgänglig på följande webbadresser, i form av ett bläddringsbart källkodsförråd respektive en paketerad version:

`http://viff-dkand.gotdns.org/`
`http://www.csc.kth.se/~joelpet/viff-dkand/`

[5] `http://en.wikipedia.org/wiki/Time_Stamp_Counter`

Kapitel 4

Resultat

I detta kapitel presenterar vi resultaten av radräkningen och prestandamätningarna som beskrevs i det föregående kapitlet.

4.1 Användbarhet

Resultatet av mätningen av antalet fysiska kodrader, enligt den tidigare givna definitionen, presenteras i Tabell 4.1. I den ser vi att det krävdes ungefär 85 % fler rader för oss att implementera lösningen med en central server än vad det krävdes för VIFF-implementationen. Uttryckt annorlunda bestod det VIFF-baserade programmet av 54 % så många rader kod som den centrala server-implementationen.

	VIFF	**Central server**
Antal rader	42	78
Varav signifikanta	22	71

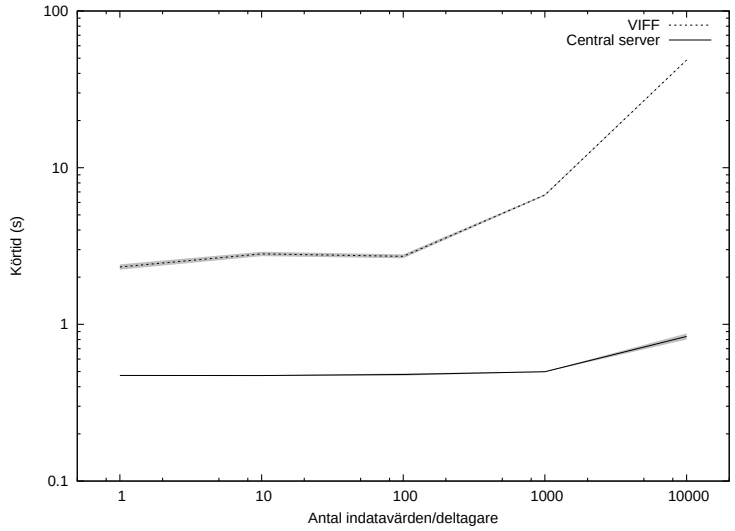
Tabell 4.1. Jämförelse av kodrader mellan implementationen med VIFF respektive en central server som pålitlig tredje part. Med en signifikant rad avses en rad som inte är ett `import`-direktiv och inte enbart läser in inställningar från kommandoraden.

Av de 42 kodraderna i VIFF-implementationen bestod 9 av `import`-kommandon, medan 11 rader hanterade inläsning av programargument från kommandoraden. Dessa användes huvudsakligen av oss för att styra prestandamätningen och var alltså inte centrala för programmets funktionalitet. De resterande raderna benämns *signifikanta rader* i tabellen.

I centrala server-implementationen gjordes alla `import`-direktiv på 3 rader i serverdelen och på 4 rader i klientdelen, således totalt 7. Inga extra rader användes för att hantera kommandoradsargument.

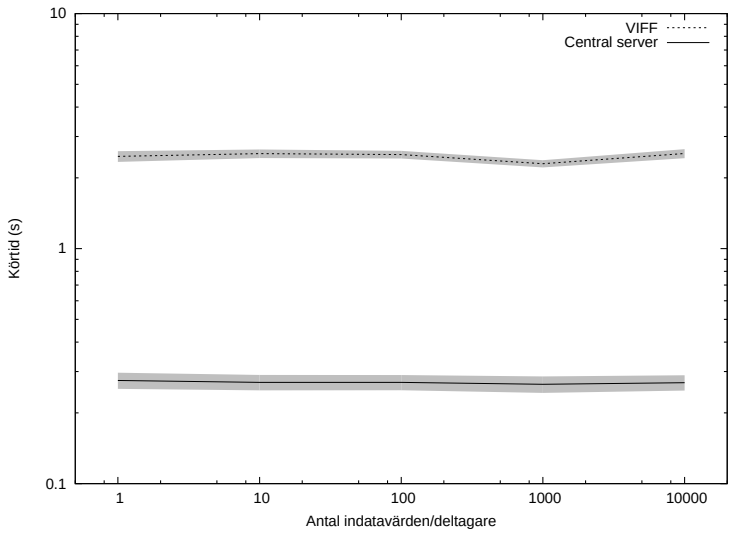
4.2 Prestanda

Körtiderna för medelvärdesproblemet presenteras i Figur 4.1. Dessutom mättes körtiderna för det tomma problemet som saknar både indata och beräkningar, vilket redovisas i Figur 4.2. Linjerna anger medelvärdet och de gråa fälten indikerar ett 95-procentigt konfidensintervall. Notera särskilt att logaritmiska skalor används för båda axlarna i graferna. Trots att VIFF bara ser ut att vara 2-3 gånger långsammare i graferna så är alltså skillnaderna större än så. Man kan dock avläsa det asymptotiska beteendet precis som i en vanlig graf. Som synes ökade körtiden för medelvärdesproblemet linjärt eller svagt exponentiellt i VIFF-implementationen vid indatastorlekarna över 100. Vidare löstes det tomma problemet på konstant tid, vilket var väntat eftersom det inte fanns någon indata. Utöver att resultaten tjänade som en kontroll av mätprogrammet så visar det även starttiden för de båda teknikerna, vilket motiverades i sektion 3.2.2. Slutligen visade mätningen av processorns användningstid på ett linjärt förhållande mellan tiden och antalet indatavärden. Detta presenteras i Figur 4.3.



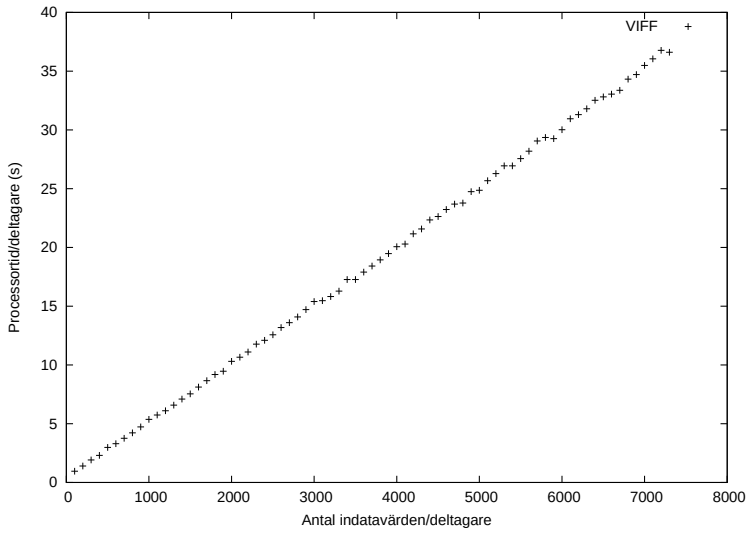
Figur 4.1. Körtider för beräkning av ett antal medelvärden.

Ett par problem påträffades vid mätningarna. Till att börja med gick det inte att köra med 100 000 indatavärden i VIFF som vi hade önskat göra, eftersom flera av datorena fick slut på primärminne. Därför slutar graferna vid 10 000 indatavärden. Vidare tog vissa körningar betydligt mycket längre tid än de övriga körningarna. En trolig förklaring till detta kan vara att någon av datorerna tillfälligt tappade kontakten mot Internet några gånger. Detta problem avhjälptes genom att filtrera bort



Figur 4.2. Körtider för det tomma problemet.

körningar vars körtid avvek mer än 50 % från medianen. De bortfiltrerade körningarna utgjorde som mest 10 % av det totala antalet körningar för någon kombination av program och indatastorlek. Dessutom förekom störningarna med både VIFF och med en central server. Därför kan resultaten ändå anses vara representativa.



Figur 4.3. Processorns användningstid för deltagare nummer 4 vid olika antal indatavärden. Dessa körningar gjordes lokalt på dator 3, med fyra deltagare.

Kapitel 5

Diskussion

Det finns, sammanfattningsvis, många situationer där *Secure Multiparty Computation* är mycket användbart – för att inte säga oundvikligt. Hittills har större delen av all forskning inom området handlat om att utveckla de teoretiska delarna. VIFF är ett intressant försök att realisera den teorin. Vi kommer i resterande del av detta kapitel att diskutera resultaten av de experiment och mätningar som vi har utfört i syfte att jämföra VIFF med en traditionell lösning där en central server agerar pålitlig tredje part.

5.1 Användbarhet

I jämförelsen av antalet rader kod mellan VIFF- och centrala server-implementationen visade det sig att den förstnämnda krävde nästan bara hälften så många kodrader för att uppnå samma resultat. Om man räknar bort kod som inte är kritisk för programmets kärnfunktionalitet, till exempel hantering av kommandoradsargument, blir det mindre än hälften så många rader kod.

Den stora skillnaden i antal kodrader visar sig huvudsakligen bero på att VIFF kapslar in och abstraherar bort mycket av nätverkslogiken. I båda fall hanteras den av ramverket Twisted, men i VIFF sköts det alltså till stor del "under huven" – till skillnad från centrala server-implementationen där all sådan kod måste skrivas explicit.

Även om antalet kodrader inte nödvändigtvis står i direkt korrelation med användbarhet ger detta ändå en fingervisning om ramverkets användbarhet. Det ska dessutom poängteras att VIFF bidrar med ytterligare en dimension i jämförelsen eftersom beräkningarna sker utan behov av en pålitlig tredje part. En fördel med att det inte krävs en server är att endast ett program behöver underhållas, istället för två – klient och server. All kod finns därmed dessutom samlad på ett och samma ställe.

En faktor som med stor säkerhet påverkar användbarheten av ett ramverk är kvaliteten på dess dokumentation. I fallet med VIFF bedömer vi den som god utifrån de erfarenheter som programmeringen med ramverket i detta projekt har

givit oss. Samma sak gäller för det lite mer mogna nätverksramverket Twisted, det vill säga att vi uppfattar dokumentationen som utförlig och omfattande. Twisteds dokumentation erbjuder dessutom en genomgång av ramverket i form av att ett typprogram byggs upp från att bara vara ett tunt skelett till att bli ett högst intressant program. På vägen beskrivs och förklaras varje ny komponent som införs. En sådan genomgång av VIFF hade varit mycket nyttig i syfte att underlätta för nya utvecklare som vill använda ramverket.

Ett generellt problem med den här typen av beräkningsprogram är att de måste konfigureras för att deltagarna ska få kontakt med varandra. Detta gäller i viss grad för centrala server-implementationen, men framför allt för implementationen med VIFF. Den senare måste ju hålla reda på samtliga övriga deltagares nätverksadresser, medan en klient i centrala server-implementationen bara behöver ställa in serverns nätverksadress. I ramverket ingår dock komponenter både för att skapa och läsa in konfigurationsfiler, vilket underlättar förfarandet. Problemet som beskrivs i avsnittet *Kontakt mellan deltagare* kvarstår emellertid.

Utöver konfiguration av programmet hos var och en av deltagare krävs även konfiguration av datorernas nätverksmiljö för att nätverksdelen ska fungera. Varje deltagare behöver lyssna på en i förväg överenskommen port, vilket kan kräva att denna måste öppnas i en eventuell brandvägg hos deltagaren. Problemet hade kunnat lösas genom användning av automatisk portkonfiguration med UPnP, men eftersom detta skulle medföra vissa säkerhetsbrister bör UPnP inte betraktas som en slutgiltig lösning.

5.2 Prestanda

Mätningarna visar att VIFF är flera gånger långsammare än motsvarande implementation med en central server. Detta är naturligt då den sistnämnda implementationen gör både enklare beräkningar och dessutom färre överföringar på nätverket. Här antyder mätningen av processorns användningstid att det inte är processorn som är flaskhalsen, vilket innebär att prestandaproblemet snarare härrör från att deltagarna tvingas vänta på data från varandra. En bidragande faktor till väntetiderna lär vara svarstiderna på Internet.

Emellertid domineras körtiderna för båda implementationerna av starttiden vid färre än 1 000 indatavärden i medelvärdesproblemet. Inte heller i dessa fall var det processorn som var flaskhalsen, men processorns användningstider utgjorde däremot en större andel. Vidare kan det finnas fördröjningar i mätprogrammet, vilket kan ha förlängt körtiderna något, och då felaktigt fått processoranvändningen att se förhållandevis mindre ut. Processoranvändningen vid uppstart beror rimligtvis på komplexiteten hos ramverken Twisted och VIFF samt att de är skriva i Python som är ett interpreterat programmeringsspråk.

Eftersom resultaten baserar sig på mycket enkla program är de troligen inte representativa för komplicerade tillämpningar. Särskilt ger de inget underlag för hur VIFF hanterar problem som kräver beräkningar i flera steg, eller beräkningar som

involverar andra operationer än addition, såsom multiplikation och olikhetsjämförelser. Detta är föremål för vidare studier av vilka resultatet vore mycket intressant att jämföra med det som vi presenterar i denna rapport.

5.3 Slutsatser

VIFF är, avslutningsvis, ett bra steg på vägen i arbetet med att överföra SMC till verkligheten; det är byggt på resultat av modern forskning, väldokumenterat och dessutom i hög grad användbart. Vi ser dock konfigurationen av programmen hos deltagarna och hur de från början ska hitta varandra på ett säkert sätt som betydande bekymmer. Beroende på tillämpningsområde är det möjligt att även prestandan innebär ett hinder. Våra undersökningar slutar dock vid ett enkelt summeringsproblem – här hoppas vi att framtida undersökningar kan ta vid!

Litteraturförteckning

[1] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 1–10. ACM, 1988.

[2] Gunnar Blom, Jan Enger, Gunnar Englund, Jan Grandell, and Lars Holst. *Sannolikhetsteori och statistikteori med tillämpningar*. Studentlitteratur, 2005. ISBN 978-91-44-02442-2.

[3] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael Schwartzbach, and Tomas Toft. Multiparty computation goes live. Cryptology ePrint Archive, Report 2008/068, 2008. `http://eprint.iacr.org/`.

[4] D. Chaum, C. Crépeau, and I. Damgard. Multiparty unconditionally secure protocols. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, page 19. ACM, 1988.

[5] Sibsaknar Haldar and Alex A. Aravind. *Operating Systems*. Dorling Kindersley India Pvt. Ltd., 2009. ISBN 978-81-317-1548-2.

[6] R.E. Park et al. Software size measurement: A framework for counting source statements, September 1992.

[7] K. Selén. UPnP security in Internet gateway devices. *Publications in Telecommunications Software and Multimedia*, 2006.

[8] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11): 612–613, 1979.

[9] VIFF Development Team. Applications — viff v1.0-b2b8e4a74cd6 documentation, 2010. `http://viff.dk/doc/applications.html`.

[10] Guido Urdaneta, Guillaume Pierre, and Maarten van Steen. A survey of DHT security techniques. *ACM Computing Surveys*, 2009. `http://www.globule.org/publi/SDST_acmcs2009.html`.

[11] Håvard Vegge. Realizing secure multiparty computations. Master's thesis, Norwegian University of Science and Technology, June 2009.

[12] David A. Wheeler. Sloccount user's guide, August 2004. `http://www.dwheeler.com/sloccount/sloccount.html`.

[13] Andrew C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual IEEE Symposium on Foundations of computer science*, volume 23, pages 160–164. Citeseer, 1982.

Bilaga A

Datorspecifikationer

I Tabell A.1 redovisas specifikationerna för datorerna som testkörningarna utfördes på. Bandbredden och svarstiden som anges i tabellen är uppmätta genom **Bredbandskollen TPTEST**[1] och anges i Mbit/s respektive millisekunder.

	Server	**Deltagare 1**	**Deltagare 2**	**Deltagare 3**	**Deltagare 4**
Operativsystem	Ubuntu 9.04 Server Edition	Ubuntu 9.10	Ubuntu 9.10	Ubuntu 10.04	Debian Squeeze/Sid
CPU	AMD Sempron™ Processor 2600+	Intel® Core™ 2 CPU 6400	Intel® Core™2 Duo CPU T7250	Intel® Pentium® 4	Intel® Celeron® M
Klockfrekvens	1,607 GHz	2,13 GHz	2,00 GHz	3,0 GHz	900 MHz
RAM	1 GiB	8 GiB	1 GiB	1 GiB	1 GiB
Nätverkstyp	Ethernet	Ethernet	Ethernet	Ethernet	Trådlös (802.11g)
Bandbredd, ned-/uppströms	89,88/10,96	154,00/14,20	110,33/15,61	13,6/0,8	13,7/0,9
Svarstid	2 ms	4 ms	5 ms	28 ms	30 ms

Tabell A.1.

[1] `http://www.bredbandskollen.se/`