GAF: A General Auction Framework for Secure Combinatorial Auctions

by

Wayne Thomson

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Master of Science
in Computer Science.

Victoria University of Wellington
2013

Abstract

Auctions are an economic mechanism for allocating goods to interested parties. There are many methods, each of which is an Auction Protocol. Some protocols are relatively simple such as English and Dutch auctions, but there are also more complicated auctions, for example combinatorial auctions which sell multiple goods at a time, and secure auctions which incorporate security solutions. Corresponding to the large number of protocols, there is a variety of purposes for which protocols are used. Each protocol has different properties and they differ between how applicable they are to a particular domain.

In this thesis, the protocols explored are privacy preserving secure combinatorial auctions which are particularly well suited to our target domain of computational grid system resource allocation. In grid resource allocation systems, goods are best sold in sets as bidders value different sets of goods differently. For example, when purchasing CPU cycles, memory is also required but a bidder may additionally require network bandwidth. In untrusted distributed systems such as a publicly accessible grid, security properties are paramount. The type of secure combinatorial auction protocols explored in this thesis are privacy preserving protocols which hide the bid values of losing bidder's bids. These protocols allow bidders to place bids without fear of private information being leaked.

With the large number of permutations of different protocols and configurations, it is difficult to manage the idiosyncrasies of many different protocol implementations within an individual application. This thesis proposes a specification, design, and implementation for a General Auction Framework (GAF). GAF provides a consistent method of implementing different types of auction protocols from the standard English auction

through to the more complicated combinatorial and secure auctions. The benefit of using GAF is the ability to easily leverage multiple protocols within a single application due to the consistent specification of protocol construction.

The framework has be tested with three different protocols: the Secure Polynomial auction protocol, the Secure Homomorphic auction protocol and the Secure Garbled Circuits auction protocol. These three protocols and a statistics collecting application is a proof of concept for the framework and provides the beginning of an analysis designed at determining suitable protocol candidates for grid systems.

Acknowledgments

There are several people who I need to acknowledge who have helped me in a variety of ways while working on this thesis. My family who provided both moral and financial support, and who listened to me patiently while I tried to explain something which sounded meaningless. My supervisors: Kris Bubendorfer and Ian Welch who provided plenty of insight, support and guidance. The members of the distributed systems research group who I have worked with closely over the last few years. The science faculty, especially Celia Simpson who put up with my complicated enrollment due to my marriage and other working commitments.

Finally and most importantly my wife Zakiah who puts up with a lot, especially my grumpyness when I am tired from too much work or study.

Contents

Chapter 1

Introduction

Auctions are an economic mechanism for allocating goods to interested parties. In a standard auction a single good is allocated to a single bidder, but there are more complex auction types such as combinatorial auctions, where a set of goods is allocated between multiple bidders. Each type of auction is called an auction protocol and there are many, including a large number for performing combinatorial auctions. All of the protocols implemented for this thesis are combinatorial auctions, as they are better suited to our target domain.

Auctions are used to distribute resources for many applications, including selling household goods, cars, allocating trucking and bus routes, airport takeoff and landing slots, frequency spectrums and grid computing. Grid computing is the use of distributed networks for the purpose of resource sharing [21]. Grid systems are platforms which allow applications to engage in grid computing. For grid systems, combinatorial auctions are a good solution for resource allocation, as resources such as memory, bandwidth, or storage may have an individual value but subsets of goods may have different values for different bidders. One such system is Nomad [9]: a middle-ware framework in which distributed applications are divided into a series of mobile agents. A marketplace within Nomad uses combinatorial auctions to distribute resources between agents at run-

time. One major concern with using auctions in an environment without a trusted party is that public information can be abused for competitive advantage [12]. Although there are a variety of different protocols using embedded security, it is the special case of privacy preserving combinatorial auctions with bid hiding which are used to solve this issue.

GAF (General Auction Framework) is a framework designed for the development of consistent auction protocols. This thesis develops the design of GAF, and describes a proof of concept prototype. The prototype is used to implement three privacy preserving auction protocols and an application for analysing them. These three protocols provide case studies for the framework but are also potential solutions for grid system resource allocation.

1.1 Objectives

The primary objective of this project is to develop a framework for consistent auction protocol development. The framework will offer services to two developer types: protocol developers who build auction protocols within GAF, and application developers which use a GAF implementation to leverage implemented protocols. It is important to show that GAF is useful from both perspectives and therefore three secure combinatorial auction protocols are implemented within GAF, and also a GAF implementation included within a protocol statistics gathering tool. The data gathered on the different protocols will be analysed and compared for future discussion on compatibility with grid systems.

1.2 Contributions

1. This thesis develops a specification and design of an auction framework called General Auction Framework (GAF) for the structured development of auctions protocols and applications.

2. An implementation of GAF has been built for this project in JAVA providing the base infrastructure for auction protocol development. In addition the work includes an implementation of a sample auction application called AuctionComposer which tests and gathers statistics on auction protocols included as GAF protocols. The application provides a proof of concept for the framework as it runs different types of auctions using GAF.

3. Three case studies of secure combinatorial auction protocols have been implemented in GAF. The first two, which were implemented for this thesis by interpreting the original papers, are Polynomial secret sharing, using dynamic programming [63], and Threshold homomorphic encryption, using dynamic programming [69]. The third is Garbled Circuits [38] which was initially built outside of GAF [40], but has been ported into GAF by the author for this thesis.

4. Explanations and worked examples are provided for the two dynamic programming protocols as both original papers are difficult to understand and missing detail required for implementation.

5. An introductory performance analysis of the three secure combinatorial auction protocols. This analysis is intended to form the beginning of more detailed future work analysing the protocols.

6. A proposed modification of the Polynomial secret sharing auction protocol which significantly improves scalability.

1.3 Thesis Organisation

This thesis is organised as follows. Chapter two provides related work on auction theory and framework development. Chapter three introduces the topic through three auction scenarios before providing a high level description of GAF. Detailed descriptions including worked examples of

the two dynamic programming protocols are provided in chapter four. Chapter five provides implementation information of GAF, the performance recording tool and the three protocols. Chapter six discusses the implementation of GAF, the protocols and a statistics gathering application called AuctionComposer which were all developed for this thesis. Chapter seven contains an evaluation of the protocols and framework followed by chapter eight concluding the thesis and listing future work.

Chapter 2

Related Work

This chapter surveys related work with specific interest in combinatorial auction protocols and framework design. Section 2.1 discusses work related to auctions, including combinatorial auctions and the winner determination problem, combinatorial auction protocols, auction rules, bidding languages and security concerns. The section also provides examples of existing combinatorial auction implementations. Section 2.2 provides work on framework design, including domain analysis, framework development and hotspots (variable framework aspects).

2.1 Auctions

Auctions are used to distribute goods between interested parties. Interested parties are called bidders who bid on auctions controlled by auctioneers. An auctioneer may sell or purchase goods, with the latter type called a procurement auction. Auctions are useful when the value of goods is unknown, facilitating discovery of a market value (called price discovery). Auction theory [29] is the study of auctions, including the study of different types of auctions and their possible use.

2.1.1 Combinatorial Auctions

Combinatorial auctions sell multiple goods in a single auction, where bidders bid on subsets of the total auction goods. When bidding in combinatorial auctions, goods can either be complements or substitutes. Complementary goods are required by a bidder as a complete package (set of goods). If a bidder cannot acquire all goods then there will be no interest in subsets of those goods. Substitutable packages are not required as a whole and so goods can be acquired individually [17].

2.1.2 The Winner Determination Problem (WDP)

The combinatorial auction has had much interest in the last fifty years, of which most study has been spent on creating solutions to the winner determination problem (WDP). The WDP is the problem of determining an optimal allocation of goods to bidders (also called pareto efficient). This is difficult because of the number of potential allocations. As the number of goods increases the potential allocations grow exponentially.

Lehmann et al. explain three optimisation models for solving the WDP with OR and XOR languages: integer linear programming, weighted stable set in graphs, knapsack and matching. The WDP is shown to be NP-Complete which means there is most likely no algorithm which can find a solution within a polynomial time in relation to the size of the input [37]. Under certain conditions however (by restricting some aspect of a protocol such as expressiveness or using approximation) some algorithms allow tractable runtime for some auction instances [34, 37].

Müller attempts to discover what conditions make certain input instances in combinatorial optimisation algorithms solvable in polynomial time [37]. It is demonstrated that restricting bundles which can be bid on or the values which can be represented can allow for polynomial-time solvable solutions. Possible algorithms given were created using integer programs and stable sets in intersection graphs.

Sandholm studies winner determination algorithms which can be proved to find an optimal solution [50]. When using modern search algorithms he claims that only the minority of instances will be slow. Sandholm uses a search tree, discussing construction, searching and improving performance. By removing unnecessary bids (ones which will not win) as early as possible depending on certain criteria, performance can be increased. This can be achieved with search trees by placing cuts on nodes or edges so that unneeded nodes are not traversed. The search algorithms given are:

- Depth-First Search keeps the currently found optimal solution in memory which can be used if the algorithm instance takes too long.
- Depth-First-Branch-and-Bound Search (DFBnB) is a simple algorithm which prunes branches which are not of greater value than the solution in memory.
- A* creates the smallest possible search tree but often runs out of memory.
- Iterative Deepening A* Search (IDA*) uses as little memory as depth-first and DFBnB. It is claimed that IDA* is two orders of magnitude faster than depth-first search.

Sandholm explains his implementation and customisation of a DFBnB algorithm called Combinatorial Auction Branch on Bids (CABOB). CABOB creates a tree on which each parent node is a bid for a bundle and has two children. Given a node and a bid, one child of that node will correspond to the bid being accepted and the other child as not accepted (A Branch on Bids strategy).

When determining the winner of an auction additional constraints may affect the outcome. More so in procurement auctions, auctioneers may specify requirements that bidders must also bid on, such as service quality

aspects. In multi-attribute English auctions [19] bidders offer the auctioneer a profile, which is scored according to the attributes specified by the auctioneer. The winner is the bidder with the highest scoring profile. The difficulty is scoring attributes and relating the score to a price (if supplied), i.e. does the auctioneer prefer a higher score or a lower price?

2.1.3 Auction Protocols

Protocols define the method and basic rules of an auction. Four common auction protocols are known: English, Dutch, Sealed-Bid and Vickrey [14, 29]. The English auction is the standard ascending price auction, the price in Dutch auctions descends from some maximum amount until a bid is made. Bidders in sealed-bid auctions submit a single bid which is not opened by the auctioneer until the auction ends, a vickrey auction is a sealed-bid auction where the winner pays the second highest bid price.

For clarity a distinction is made between protocols and auction rules. Protocols are the different types or models of auction and specify pre-established rules. Within protocols rules may be modified and so a distinction is made: the choice of protocol for an auction is not an auction rule, an auction rule is an addition or modification of pre-established protocol rules.

Combinatorial Auction Protocols

There are many combinatorial auction protocols including the Vickrey Clarke-Groves auction (VCG or GVA) [4], the iterative combinatorial auction [42], the ascending proxy auction [5], the simultaneous ascending auction [17] and PAUSE [32].

The Vickrey Clarke-Groves auction is an extended form of the Vickrey auction. Once winners have been found a discount is applied to the winning bidders' prices (because there are multiple goods, there can be multiple winners, hence there usually is not a single second highest price).

There are several benefits of using VCG auctions [4]: each bidder's dominant (best) strategy is to bid according to their true values, communication and computational cost is reduced, average revenues are no less than in other efficient protocols and additional winner determination rules can be added to the algorithm.

Ausubel [4] claims that in practice VCG is not often used because seller revenue can be low or nothing and it is vulnerable to collusion which reduces seller revenue. By making substitute preferences for goods required, Ausubel shows the problem is removed.

Iterative combinatorial auctions are a class of combinatorial auctions which are run in a similar fashion to English auctions. The auctioneer provides information about the state of the winners or winnings, such as the winning packages of goods and prices in the given round and the bidders submit bids based on this information. Parkes [42] identifies two types of iterative combinatorial auctions: price-based and non-price-based. In price-based auctions, the bidders are expected to provide the auctioneer with the price of their current bid. In non-price-based auctions, the auctioneer expects bidders to supply some other representation of a bid.

The benefit of using an iterative combinatorial auction over a VCG auction is that even when substitute preferences are not used the seller revenue is guaranteed to be reasonable. One potential problem with iterative combinatorial auctions is the amount of computation and communication required to solve the WDP repeatedly, whenever bidders need to be supplied with current auction state. However Sandholm suggests an algorithm to update the WDP instead of rerunning it [50].

Bid communication can also be high, as there must be at least two rounds if the auction is round-based, or enough time for several submissions of bids in a time-based protocol. Examples of implemented iterative combinatorial auction protocols are given in [42].

The ascending proxy auction [5] is an iterative combinatorial auction protocol. Bidders submit package valuations for packages they are inter-

ested in to an intermediary called a proxy bidder. The prices for packages at the start of the auction are set at some minimum amount and raised by a set amount in each round. At the end of each round the auctioneer solves the WDP to find a set of provisionally winning bidders. If the proxy bidder determines a package to be profitable and is not already a provisionally winning bidder, it bids the set amount for that round on behalf of the bidder.

As it is an iterative auction the outcome is as efficient as a VCG auction when substitute bids are present, and more efficient when not present. Bid signaling is a technique where bidders may communicate through bids and this is a concern in iterative combinatorial auctions. For efficiency when proxy bidders bid on behalf on multiple bidders, they can forward only the highest bids, reducing the number of transmissions but more importantly reducing the number of possible allocations. If this mechanism is used, care needs to be taken to avoid corrupt proxy bidders favouring particular bidders.

PAUSE [32] describes an iterative protocol which distributes computation of the WDP (see section 2.1.2) amongst bidders. Auction rounds are spread amongst a number of stages equal to the number of goods. Bidders bid on individual goods in the first stage, but in subsequent stages, bidders offer solutions to the WDP using any combination of disjoint packages made by any of the bidders in previous stages. When creating packages, the maximum size is equal to the round number. This version of an iterative protocol requires little computation from the auctioneer, and bidders are able to verify the outcome by comparing the offered solutions [32]. It is possible that bidders could attempt to represent false bids as other bidder's bids so verification is an important consideration. Although computation is cheap for the auctioneer (proportional to the number of bidders), it is expensive for bidders. Not only do bidders have to generate their valuations, they need to solve the WDP in each round.

Simultaneous ascending auctions, although not true combinatorial auc-

tions, have been used in auctions of related goods with combined revenue totaling over $200 billion [17]. Bids are made on separate goods as if running multiple English auctions so determining the set of winners is computationally cheap, as English auctions are computationally cheap. The problem with this approach is that bidders can win only a subset of complementary goods, which may not be useful as a whole. This is mitigated slightly by allowing bidders to withdraw bids for goods when they cannot win complementary goods. Despite this, in practice this scheme has proven to provide efficient allocations as little resale of goods have been recorded [17]. Simultaneous auctions suffer from low seller revenue with low levels of competition and are also susceptible to bid signaling. To mitigate these problems, concealing bidder identities, setting high reserve prices and offering preferences for small businesses has been suggested [17].

2.1.4 Auction Rules

Auction rules are properties of an auction protocol which may be specified by the owner of the auction and they provide for variability in the auction. Wurman et al. [65] cover some basic ground-defining bidding rules, clearing rules (here called winner determination rules) and privacy rules (which here is a subset of another set of auction rules: security rules).

Bidding Rules

Bidding rules govern who can bid, when bids will be accepted, what constitutes a bid and what goods can be bid on.

Activity rules [3, 6, 42, 65] state how active a valid bidder must be in order to continue bidding in an auction. The rules may stop a bidder from further bidding on particular packages of goods or possibly the entire auction if the bidder is not active enough, or too active.

One type of activity rule requires bidders in iterative combinatorial

auctions to bid on packages they are interested in from the first round. The purpose behind this is to improve price discovery and minimise the number of rounds in the auction. If bidders are allowed to bid on any packages late in the auction they may try to hold off bidding until the end of the auction, when there is no time for competition to grow (called sniping) [5]. Other ways of achieving this are: in multi-unit auctions as prices increase, quantities must not, keeping a total of goods being bid on and ensuring that amount does not increase, recording bidder preferences and ensuring they stay the same [5]. Each of these strategies have related problems and loopholes which need to be considered when designing a protocol. Another example is the auto-extend feature in traditional English online auctions such as that on the auction sites trademe.co.nz and ebay.com.

Activity rules can also be used to reduce retaliatory bidding in combinatorial auctions [18]. For example if bidder A bids on a package bidder B desires, B might retaliate by bidding on a package bidder A desires (forcing the price to be increased). By restricting bidders to bidding on packages they have bid on in previous rounds, a bidder who wishes to retaliate can only do so on packages they are already interested in.

Bid restrictions determine how a bidder can make bids, for example reserve prices define a minimum size of a bid for a particular good or package. This is more complex for combinatorial auctions as sets of goods may have different minimum values than the sum of their individual goods.

Language restrictions determine what language or format the bids must be in, see section 2.1.5. A rounding rule requires bidders to submit rounded bids. This may be required in situations where bidders may try and communicate using bid signaling [42].

Timing rules restrict when bidders can submit bids. For example in iterative round-based auctions, only one bid is accepted per round per bidder. Subsequent bids may be thrown away or replace previous bids.

Bid improvement rules restrict the acceptance of bids based on their

value or intent. Bid improvement rules include bid withdrawal conditions (if allowed), minimum bid increments and reduced bids [18].

An example of a bid improvement rule is the bid dominance rule which prohibits bidders from bidding a lower value than that which they have previously made for the same package [65, 18].

Minimum bid increments restrict bidders to bidding a minimum amount more than has already been bid. The goal of this is to encourage meaningful bidding. It would be pointless and even detrimental for a bidder to bid in increments of a dollar when their valuation and others bidder's valuations are thousands of dollars higher. Not only would this waste time, but also communication and computation [17]. Increments may be specified as a price or a percentage and may either be fixed or offered as quotes by the auctioneer. Quotes are expected to be beaten by the bidder [65].

A reduced bidding rule allows bidders to reduce their previous bids [41]. This allows bidders to refine their bids for packages. For instance in a simultaneous auction where bid withdrawal is not permitted but reduced bidding is and a bidder is interested in one of two goods but not both. If one becomes too expensive the bidder can reduce the amount they are willing to pay for the former and start bidding more on the second.

Withdrawal conditions are used in auctions where benefit is gained from bidders being able to back out of bids. A good example of a protocol which takes advantage of this is the simultaneous ascending auction. By allowing bidders to back out of packages they have bid on, the protocol simulates a combinatorial auction. As bidders can back out of failed attempts at acquiring complementary items (refining their bids), an efficient allocation can be found [17, 42]. Penalties may be introduced to discourage insincere bidding, for example Cramton suggests fining the bidder the difference between the withdrawn bid and the final sale price [17].

Winner Determination Rules

Winner determination rules modify the outcome of the auction in consideration of factors other than prices or valuations. These rules are often domain specific, but generalised rules are:

1. A quantity cap on the number of goods or packages a bidder can win encourages competition [17].

2. Restricting the number of winners allows the auctioneer to only deal with a specified number of winners [34]. For example if signing contracts with multiple vendors is costly, a limited number of winners may be beneficial.

3. Side constraints factor business rules into the WDP. Examples include legal constraints, prior contractual obligations, reputation and delivery time [34]. In these cases bidders either need to supply additional information with their bids, or the auctioneer needs another method of obtaining these types of information.

Security Rules

Security rules are put in place for a variety of reasons for example verification of auction results or bidder privacy. As secure combinatorial auction protocols are a focus of this thesis, auction security is explored in more depth than the other rule types in section 2.1.6.

2.1.5 Bidding Languages

The way that bids are represented in an auction, especially a combinatorial auction, can make a considerable difference to the expressiveness of bidder preferences, the outcome of the auction, and the level of computation and communication required. Two types of languages are discussed here: logical and graphed.

Nisan [39] presents different ways that bids can be represented in combinatorial auctions:

1. Atomic bids allow bidders to submit a set of goods and a valuation of that set.

2. OR bids contain multiple atomic bids of which the bidder is willing to accept any number of disjoint components for the total valuation of the components.

3. XOR bids contain multiple atomic bids of which the bidder is willing to accept one for the valuation of that atomic bid.

4. OR-of-XORs and XOR-of-ORs give a higher level of expressiveness, allowing multiple disjoint XOR bids from a set of XOR bids or a single OR bid from a set of OR bids to be accepted respectively.

5. OR* bids can express everything that OR-of-XORs and XOR-of-ORs can, but is expressed in OR bids. This is achieved by providing each bidder with a personal set of dummy goods. The XOR bid A XOR B XOR C can be represented using dummy item Z as the OR bid AZ OR BZ OR CZ [39].

Nisan gives some extensions: logical notation, budget limits, limits on numbers of goods, using graphs for homogenous goods, allowing ALL and MIN notation and k-OR, which allows a bidder to express a desire for a maximum k number of atomic bids from an OR statement.

Instead of asking bidders for bids, in preference elicitation the auctioneer asks bidders specific questions about packages of goods. By carefully asking bidders different questions, an optimal allocation should be possible with less communication [51]. Questions may be phrased such as ”Do you prefer package A or B?” or What is your valuation for package ”Y?”.

Resource description graphs (RDG) [9, 16, 11] are used to describe resource requirements. The party interested in purchasing resources creates an RDG which is used by bidders to decide whether to bid, what to

bid on and how to formulate their bid. Bidders submit RDGs to the auctioneer which contain some subset of the original graph corresponding to their interests of supply. An RDG is a rooted-directed acyclic graph where the edges correspond to resources and the vertices are used to indicate whether a path will be accepted as a bid. Edges have attached constraints which indicate resource requirements. The bidders RDG edges have corresponding fulfillment values. A sentence is a path which can be accepted as a bid and starts at the root of the graph and ends at an accept state. Accept states match tasks that the agent placing the auction wishes to complete. A grammar is available for conversion to a textual representation for use by humans.

This language can reduce the WDP slightly (see 2.1.2) as bids are limited to the initial RDG [9] and allow bidders to easily determine if their constraints permit them to bid. RDGs were designed for use with NOMAD [10] but may be useful for other systems.

2.1.6 Auction Security

Security properties of an auction protocol are factors which ensure correct performance of an auction. Correct performance includes the correct calculation of the auction allocation, but also non-functional requirements which may be desirable such as privacy. Security properties can be achieved through the use of corresponding auction rules which are implemented within protocols.

Secure Auction Properties

A classification of security properties for sealed-bid auction protocols is given by Peng et al. [43]. The first two properties are required in any auction protocol but the other properties can be useful in different situations and are offered by some protocols.

1. Correctness ensures a correct outcome from submitted bids according to auction rules when participants are honest.

2. Fairness is a property which varies under different auction protocols. The definition given states that no bidder should have knowledge of other bidders valuations before the bids are opened, a bidder cannot modify their bid (which could be assumed to include withdrawals and resubmitting) and a bidder cannot deny their submitted bids. Although somewhat relevant to sealed-bid auctions, the definition can be generalised. A better definition would state that a fair protocol is one that does not allow bidders (trustworthy or otherwise) to deviate from protocol rules. For example a protocol may allow bidders conditional withdrawals which would be fair as long as any bidder can withdraw their bid, as long as they meet the requirements and cannot otherwise.

3. Confidentiality requires that bids are not revealed. For sealed-bid auctions, confidentiality is required until bids are opened. Anonymity is a modified form of confidentiality where bidders identities are kept secret.

4. Privacy in auctions requires losing bidders bids to be kept confidential to all participants including the auctioneer.

5. Public verifiability allows anyone to verify the outcome of the auction.

6. Robustness guarantees a correct outcome when a corrupt participant attempts something malicious.

7. Price flexibility allows bids to be as precise as the participants require.

8. Protocol and Rule flexibility means the outcome will not be changed if the protocol or rules are changed. In addition, winner determina-

tion rules and approximation protocols can affect the outcome of the auction.

There are two requirements of a secure auction protocol. An auction must achieve the first three properties, including the new definition of fairness. The second requirement is that any of the subsequent six properties must hold if specified in the protocol rules.

2.1.7 Secure Auctions and Trust

There is little point in using an auction protocol which theoretically implements one or more of the properties of section 2.1.6 if there is no trust that the authority will carry out the protocol properly. With auctions being financial tools by nature, there is motivation for unscrupulous parties to cheat. A corrupt auctioneer for example in a sealed bid auction could drop or change bids. Techniques for running secure auctions are [14]:

- Pre-existing trust: participants have pre-established trust with the auction facilitator, such as the auctioneer. The main difficulty of using a system which relies on pre-established trust is the difficulty of establishing trust in the first instance. Also, establishing trust is problematic in economies requiring many small scale auctions as many trusted parties will be required. In addition, an auctioneer's motivation may change or become corrupted by another party.

- Reputation services: reputation services can reduce the primary problem of establishing trust initially as the number of pre-existing trusted parties required is reduced. Auction participants can refer to the reputation service in order to verify collective trust of another participant, such as an auctioneer.

- Bid-encryption protocols [14, 43]: protocols using bid-encryption provide a procedure to ensure correct functioning so that no pre-established

trust is required. The advantage of protocols with bid-encryption is that it is not possible for auctioneers to learn or manipulate bids. Peng et al. give two examples: in hash function sealing bidders supply signed digests of their bids, confidentiality is guaranteed until the bidder supplies an unencrypted bid. In encryption sealing, bids are encrypted when submitted. If winning bids can be found without decryption then they are the only bids to be revealed which preserves the privacy of losing bidders.

- Threshold bid-encryption protocols: threshold trust schemes allow for trust dependent on whether a number of auctioneers are trusted. Some secure auction protocols take advantage of such techniques as [55, 56, 58, 66].

2.1.8 Secure Combinatorial Auctions

Combinatorial auction protocols can have additional security issues, such as bid signaling and pseudonymous bidder strategies:

One potential concern in combinatorial auctions and possibly multi-unit auctions is bidder signaling. Signaling is not an issue in single item auctions as there is little purpose and limited strategies. Bidders may encode messages in their bids by adding small values to bids to communicate a message, or retaliating to particular bids. Retaliation can for example be used by a large bidder who bids on a package of goods it is uninterested in because a smaller bidder bid in a package it is interested in (clearly sending the smaller bidder the message that is unhappy about something the smaller bidder did) [18]. The possibility of bidder signaling can be mitigated by rounding [18], concealing bidder identities, proxy auctions [18] or possibly by setting high reserve prices and offering preferences for small businesses have been suggested [17].

Bidders in combinatorial auctions may have a pseudonymous bidder strategy where it is advantageous to bid under multiple identities [67].

Bidders may attempt this to alter VCG payments [67] or to purchase complementary items separately (presumably at a lower price) [41]. Yokoo introduces the concept of a pseudonymous-bid-strategy-proof protocol where the dominant strategy for a bidder is to not use pseudonymous. He proves that there is no pseudonymous-bid-strategy-proof combinatorial auction protocol which has pareto efficiency. Yokoo develops a pseudonymous-bid-proof protocol called the Leveled Division Set (LDS) protocol.

Privacy Preserving WDP Algorithms

A subset of secure auctions is privacy preserving combinatorial auctions, in which the WDP algorithm performs operations on encrypted bids. Bidders submit encrypted bids of which malicious auctioneers can only determine the optimal allocation and its corresponding bids. Some component of a bid may need to be decrypted to find prices for goods and the general approach to this problem is to use threshold encryption to restrict decryption to a minimum. None of the following algorithms provide a method for implementing business rules, though some could be implemented in monetary terms.

Yokoo et al. [68] provides a combinatorial threshold bid-encryption WDP algorithm using dynamic programming and homomorphic threshold encryption. Values encrypted with homomorphic encryption can be multiplied, retaining the product of the encrypted values when decrypted. Vectors of encrypted values, called weights, can be created representing bids, e.g. if $e(z)$ represents z encrypted and z is a number not equal to one, then the vector $e(z), e(z), e(z), e(1), e(1)$ will represent the number three. Notice that the number of values not equal to one represent the total weight. If two vectors are multiplied together, the resulting vector is the greater of the two vectors. An example is shown below.

$$\begin{array}{rr} & \{e(z), e(z), e(1), e(1), e(1)\} \\ \times & \{e(z), e(z), e(z), e(1), e(1)\} \\ \hline & \{e(z^2), e(z^2), e(z), e(1), e(1)\} \end{array}$$

An acyclic directed graph is created containing all the combinations of goods as nodes, while edges represent the highest bid for a bundle of goods (which as shown can be found without revealing any bids). The technique for threshold encryption is not specified, however any mechanism should be possible. The winning allocation is the longest path through the graph. Without threshold encryption and with a trustworthy auctioneer this algorithm provides confidentiality, privacy and increases robustness, however with a corrupt auctioneer all properties fail. These can be restored with threshold encryption, using multiple auctioneers. If the number of corrupt auctioneers is below the threshold then it is not possible to break the confidentiality.

This algorithm by itself does nothing for verifiability, and removes the obvious form of bidders solving the WDP themselves. The scheme has been implemented for combinatorial auctions using a single auctioneer [14]. A modification for verification is provided by Palmer [40].

Polynomial secret sharing for combinatorial auctions [62] is another threshold bid-encryption WDP algorithm. Instead of using homomorphic encryption, bids are represented by polynomials. Threshold properties are provided using polynomial secret sharing [56] which simplifies the design as it matches the bid. Dynamic programming is used and the secure properties hold. This has been implemented for this thesis as explained in [13].

The Secure Generalised Vickrey Auction (SGVA) [63] is another secure combinatorial auction WDP algorithm using homomorphic encryption and threshold encryption. SGVA does not use dynamic programming. Instead the bidders each submit their bids to the auctioneer by adding their bids to a table. The columns of the table represent the different ways that goods can be allocated amongst bidders. The first row represents the total bids for each allocation and the subsequent rows represent the totals without a corresponding bidder. Finding the winners and computing the VCG prices is easy but can be expensive. The number of columns is given by b^g where b is the number of bidders and g is the num-

ber of goods, the number of rows is $b + 1$ and so the table would usually be large, and searching lengthy. The same properties of the two secure dynamic programming schemes hold.

A modified version of SVGA [69] does not distribute encryption amongst multiple auctioneers but amongst the bidders instead. Prices for bundles are calculated first and bidders do not operate on their own submitted bids. With threshold encryption, bidders are no more able to influence the outcome of an auction than auctioneers are in the other schemes. The algorithm still uses homomorphic encryption and dynamic programming. The secure properties still hold but there is no requirement for additional third-parties. A problem not discussed is what to do in the case of not enough bidders. One solution would be to add auctioneers to make up the difference. However systems running many auctions may have to hold a spare number of auctioneers in reserve.

Garbled circuits [38] is a method of securely computing the result of an auction and ensuring no information, other than what is requested is leaked. Programs (circuits) are created by a trusted third party called an Auction Issuer (AI) and used by the auctioneer and bidders. To run an auction, an auctioneer requests a program from the third party (an auction issuer), in the form of a garbled circuit [66]. The auctioneer acts as a proxy for an oblivious protocol for bidders and the AI to encrypt bids suitable for input into the circuit. The auctioneer then passes the encrypted bids through the circuit which returns the winners and prices paid (as specified by the circuit).

Naor claims that circuits can be created to any auction specification required, without exposing private information other than the result. It is a concern that for large-scale systems, using garbled circuits would be too difficult [68] or too large to use (Naor suggests sending the circuit on a CD or DVD). Garbled circuits allow public verifiability while retaining privacy. Naor claimed that it is possible to run combinatorial auctions with garbled circuits and this has been proven to be possible [40].

2.1.9 Combinatorial Auction Implementation

This section on combinatorial auction implementation is divided into three parts: the first part outlines documented combinatorial auction implementations, the second describes a testing framework called CATS and the third describes an implementation which optimises itself by choosing from different WDP algorithms using approximation.

Industry Implementation

It has been proposed that airport take off and landing slots in the United States of America be allocated with combinatorial auctions. Currently airplanes do not have allocated times, but join a queue when taking off and landing. It is believed that purchasing slots in combinatorial auctions will make long-term strategic route planning possible [6]. Ball et al. suggests using a simultaneous multiple round ascending auction for price discovery with XOR bidding for a high level of expressiveness. There is a large number of goods in such an auction, as each takeoff or landing slot must be represented in time as a single good. However, time may not be required if sold in blocks or re-occuring times such as everyday at a particular time.

In the USA, combinatorial auctions are used for procurement of truck shipping routes. Bidders [15] bid for an expected volume of shipping across routes they are interested in for a particular rate. Third party companies hold the auctions on behalf of the companies interested in shipping their goods. Complications in this type of auction come from two directions: Shipping companies are interested in minimising the amount of time their trucks are empty, and the complexity of requirements for the WDP due to the large number of properties for consideration in addition to price. In addition to route rates, purchasers may be interested in the reputation of carrier, location, carrier response times or the size of the carrier. Caplice et al. suggest that different bidding languages may be required depending on WDP requirements.

Experimental Implementation

*i*Bundle [41] is an iterative combinatorial auction. There are three different versions allowing for different types of price increments: *i*Bundle(2) provides the same bid increment to each bidder, *i*Bundle(3) provides different increments for each bidder to support efficient allocations and *i*Bundle(d) switches between the two strategies. The bidding language used is XOR and the WDP algorithm is DFBnB (but is intended to be interchangeable with other algorithms). *i*Bundle returns efficient optimal allocations [41]

Sandholm [50] claims that in practice WDP algorithms have found optimal solutions (when distributions of packaged goods are simple) for hundreds of thousands of items in seconds. With harder distributions he claims optimal solutions for tens of items can be found within a minute. Unfortunately no specifications of the software or hardware are given.

A critique [24] of the Ascending Proxy Auction [5] in regards to implementation shows potential problems: required bidding by non-winning bidders in each round and bidding increments may lead to bidders paying more than their valuations or bidders winning goods which other bidders value more highly. These problems can both occur if the final increment raises a package bid above bidder valuation. Modifications to solve this solution include finding the greatest bids for a package, finding the maximum increment, scaling the increment so that it reduces to a (hopefully) minimal error level in the final round, and adding a rollback phase to the end of the auction for correction [24]. If bidder privacy is important, consideration should be taken as to whether the auctioneer can learn true bidder valuations.

Test Suites

The combinatorial auction suite (CATS) [35] is a package for generating sets of realistic data for combinatorial auctions. Compared with other auction protocol testing models, the CATS generators create realistic bid

packages of (specifically not randomly) related goods [35] . Goods can be marked as substitutes or complements. Several classes of problems are given:

- Paths in Space are groups of goods related by linked points. An example is trucking routes where bidders would be interested in supply routes which have collection and delivery points joined together.
- Proximity in Space considers groups of goods related by proximity. For example, bidders may be more interested in adjacent blocks of land to subdivide or blocks of adjacent frequency bands.
- Temporal Matching considers groups of goods related by time, such as airport takeoff and landing slots.
- Temporal Scheduling considers time constraints on goods. The example given is job-shop scheduling where some jobs must be completed at particular times or at some ranking in a list.
- Arbitrary Relationships are between goods which cannot be specified by a pattern. An example would be different coloured bowling pins, individually they would probably have little value but in sets of the same colour they would have the highest value.

2.2 Framework Design

The purpose of this thesis is to develop an application framework. Developing an application framework is somewhat different from developing an individual application. The development approach must consider a generic perspective across an entire domain, balancing reuse with flexibility. Standard application design objectives still apply; determining system participants and relationships, functionality and flow of control. The

major challenge of framework design, is identifying the fixed and variable aspects and designing how they are integrated. The integration between fixed and variable aspects is important, as flexibility directly relates to what is dictated by the fixed aspects.

Bosch et al. [8] define six activities in framework development; Domain analysis, architectural design, framework design, framework implementation, framework testing and documentation. The definition of a framework design methodology used in this section incorporates the first three activities. Domain analysis captures the functionality of the domain and identifies requirements and framework aspects (fixed and variable). The architectural design develops the underlying architecture of the framework, for example batch sequential, pipes and filters or layers [8, 57]. In the framework design step, the complete framework is designed, including traditional design goals, hotspots (variable aspects) and their integration.

Framework design is an ongoing research topic and there is a large number of publications in the area. The first set of publications outlined focus on domain analysis for frameworks. The second set of publications aimed to provide a full application framework development methodology but did not detail domain analysis effectively.

2.2.1 Domain Analysis for Frameworks

Boone [7] discusses (by example) harvesting domain knowledge from existing applications, but does not include explicit steps or explanations. Aksit et al. [1, 2] provides detailed steps on building 'knowledge graphs' from domain knowledge developed by analysing domain literature. Hotspots can then be determined and are mapped to an object model, but the paper does not provide a methodology for mapping. In addition, Aksit notes that the mapping is not straightforward as concerns may be cross cutting which cannot be directly mapped to objects. Pree [44] identifies hotspots

by facilitating discussion with domain experts using hotspot cards, determining aspects which differ between applications. The problem with this approach is that it is not systematic and as it is essential that the interviewer asks the correct questions, therefore experience and luck are both required. Hot spot integration is guided by the hotspot descriptions and their degree of flexibility (i.e. whether the hotspot implementation is bound at run time or configured during development). Miller et al. [36] have developed heuristics to identify hotspots by abstracting use cases from requirement sets of multiple applications.

There is much work on applying patterns to frameworks, for example [25, 27, 26, 33, 48, 52, 59]. Although patterns are very useful for composing applications and frameworks, by themselves, they do not provide a methodology. Covering all aspects of a framework with patterns, is unlikely and so at least a standard application design methodology would still be required. It can also be difficult for a framework developer as using patterns requires knowledge of the different patterns, a strong understanding of the domain to be able to apply patterns, and experience using patterns with frameworks. Riehle [47] develops frameworks using role models. Role models specify sets of object collaborations in a system which can be mapped to patterns [45, 46]. Role models may provide a useful way to identify patterns, but a mapping methodology for mapping domain knowledge to role models, and filling in any extra functionality outside of the patterns is still required.

2.2.2 Framework Development

A methodology detailed by Schmid [53, 54] develops frameworks using generalisation of application object models. The methodology requires hotspots to be extracted from existing object models but does not explain how. It does however provide a detailed process to generalise the original object models by applying patterns using heuristics to a hotspot specifica-

tion. The hotspot specification is the framework's hotspots classified with set attributes. Koskimies [31] also develops frameworks by generalising an existing application. Instead of using patterns to generalise the application, Koskimies asks the following two questions 'Which concepts of the problem domain exist in variants and should be determined uniformly?' and 'Is it possible to find a concrete concept that can be generalised into a more abstract concept?' The other difference is that once the framework is generalised as much as possible, he builds a framework for each generalisation level, starting at the most abstract. Building many framework variants could obviously get expensive. A similar methodology to Schmid, Sherlock [60, 64] requires domain analysis, and developing a domain terminology vocabulary [28]. The vocabulary is mapped to a UML model and then refined into a framework. It seems that no method is provided to develop the mapping, but is left to experience.

2.2.3 Framework Types

There are two framework types: black-box frameworks require the developer to choose existing hotspot implementation while white-box frameworks require the developer to implement new behaviour. In reality, frameworks will neither be simply black-box or white-box. The two types are more like different framework views [54]. Required adaption should be minimal and reuse of existing implementation should be maximised (without reducing required flexibility) to reduce application developers' required knowledge of framework internals.

2.2.4 Hotspot Dependencies

A hotspot dependency occurs when a hotspot cannot be replaced independently from another [49]. There are four categories of techniques which are commonly used to enforce hotspot dependencies [49]: closely coupled

classes, consistent object creation, data-driven classes and meta-level configuration.

Closely coupled classes contain references to each other in their definitions and so cannot be created independently. Consistent object creation uses factories to generate dependent types consistently without necessarily requiring knowledge of underlying type. The behaviour of data-driven classes is dependent on the data being processed and so dependent hotspots, which will use the same data type to perform consistently. Meta-level configuration is used to enforce dependencies during application construction.

Dependency enforcement for protocol dependent hotspots must be dynamic because binding for all of them is at run time. They cannot be closely coupled as each is independently created within a resource container, potentially in a distributed environment. Closely coupled classes also increase difficulty when trying to replace one implementation with another, especially with a large number of classes and many references to modify. Data-driven classes include the behaviour of each supported type, which limits scalability and there are potentially an unlimited number of possible protocols. There are two types of factories for consistent object creation. Abstract factories create consistent objects of the correct type based on the factory's implementation. State-driven factories create consistent objects, based on the factory's state. Although scalability of state-driven factories can be limited in the same way which data-driven classes can be, the factory can be parameterised with objects created with reflection [49]. This was not possible with data-driven classes because they are defined as including behaviour, although this definition could be adapted. However the change to the structure would be incompatible with the required hotspot subsystem which requires state.

Chapter 3

Introduction to GAF

This chapter provides an introduction to the General Auction Framework (GAF), firstly introducing simple protocols and protocol extensions that are prerequisite to the secure protocols used in this thesis. The examples in sections 3.1.1, 3.1.2 and 3.1.3 build on one another with the first being the standard English auction, followed by a sealed bid auction and a sealed combinatorial auction. The outcome of the sealed combinatorial protocol is computed using an auction graph and this forms the basic component for two of three protocols used as case studies for GAF. Extensions to the combinatorial protocol are presented in section 3.2 which are further requirements of the protocols explored in chapter 4.

The framework itself is introduced in section 3.3 outlining the different components and participants which it is made up of. This work is extended in chapter 5, which provides the full specification for the framework.

3.1 Auction Protocols

Auctions determine the optimal allocation of a set of goods to interested parties called bidders. Bidders are auction participants who offer an amount (their bid) for a set of goods in the auction based on their valuation of the

goods they are interested in. Probably the most widely known auction type is the open outcry, ascending bid English auction. The type of auction is the auction protocol. It defines: the different participants and their behaviour (including participant communication), how goods are offered, winner determination and pricing algorithms, bid representation, the auction closure rule and any protocol specific settings (required and optional). Reasons for choosing between protocols are numerous and outside the scope of this thesis. The reader is directed to auction theory surveys such as [30, 29].

This section explores three different protocols, contrasting the different attributes. The protocols are English, sealed bid first price and combinatorial auctions using directed graphs (an unsecured version of [68]). The English auction is an example of a single good auction with a large amount of dynamic communication. The sealed bid auction is similar in nature but the interaction between participants differs. The combinatorial auction protocol is an extension of the sealed bid protocol for combinatorial auctions. The communication pattern between participants and settings are the same as for sealed bids, but bids are made on multiple goods and the outcome is computed using a graph.

3.1.1 An English Auction

In a standard English auction, only one good is sold (offered) per auction, which is allocated to the bidder with the highest bid at the price of their bid. There are three types of participants: the auctioneer collects bids, determines the winner and informs the participants of the outcome; bidders submit their bids to the auctioneer; observers (optional) watch the entire process. Bids or bidders are not anonymous and all bids are published, so it is publically verifiable.

There are two common forms of bidding in English auctions; either the auctioneer announces prices and bidders inform whether they are still in-

terested or bidders announce offers which the auctioneer records. So bids either represent whether the bidder is still interested or the amount offered. The auction closes when there is no longer any interest at a higher price. Required protocol settings are the minimum bid and auction timeout (the length of time to wait for additional bids before closing the auction). The only optional setting is a reserve price, which is the minimum bid the good will sell at. The reserve price also may be publicised when it is achieved during the auction, or it may be private to the auctioneer.

An Example Auction

Three bidders Adam, Barry and Claude each have valuations for 'Good A' at $\$30$, $\$50$ and $\$65$ respectively. The protocol settings are: a reserve price of $\$30$, bid increment of $\$10$, the auction timeout is 2 minutes, the bidding is by bidder offer and there is no reserve price.

The auctioneer announces to the bidders and observers that bidding is open at $\$30$. The auction proceeds as follows:

1. The auction opens with a reserve of $\$30$.
2. The auctioneer notifies participants that the minimum bid is $\$30$.
3. Adam sends a bid of $\$30$ for 'GOOD A' to the auctioneer.
4. The auctioneer notifies participants that the minimum bid is $\$40$.
5. Claude sends a bid of $\$40$ for 'GOOD A' to the auctioneer.
   (Adam drops out at this point).
6. The auctioneer notifies participants that the minimum bid is $\$50$.
7. Barry sends a bid of $\$50$ for 'GOOD A' to the auctioneer.
8. The auctioneer notifies participants that the minimum bid is $\$60$.
9. Claude bids $\$60$ for 'Good A' to the auctioneer.
   (Barry drops out at this point).
10. The auctioneer notifies participants that the minimum bid is $\$70$.
11. Two minutes pass without any new bids, the auctioneer closes the auction and notifies (publishes) the participants that the last bidder Claude won 'GOOD A' as $\$60$

The order of bidding does not affect the outcome in this scenario. However it may if multiple bidders have the same valuation. In which case, additional bids of the same value need to be dropped. Also note that in this case when the bidder drops out they do not need to notify the auctioneer, as they will no longer be considered.

3.1.2 A Sealed Bid Auction

Like English auctions, sealed bid auctions sell a single good at a time. Bidders submit a single sealed bid which cannot be opened until the auction closes. This is achieved by encrypting bids and providing the keys once the auction is finished. When the auctioneer has the keys, it can publish the result (picking the highest bid). The protocol can provide verification by publishing (instead of just providing the auctioneer with the keys) both the encrypted bids and the keys.

The participants are the same as for English auctions; an auctioneer, the bidders and observers. But unlike English auctions, bidders each only submit one bid. The auctioneer determines the winner after the auction closes, and isn't required to update the participants at any point. Once the auction is closed the bidders publish their keys, the auctioneer decrypts the bids and the result is found and published as per the English auction. Obviously, observers have little to observe until the auction is closed as nothing is known but who has bid.

The auction length is fixed, required protocol settings are the minimum bid, auction length and encryption settings. The only optional setting is a reserve price which will be private until the end of the auction as it cannot be determined whether a bid is greater than the reserve until the decryption key is provided. A public reserve is not useful as a minimum bid serves the same purpose.

An Example Auction

In this scenario, the same three bidders Adam, Barry and Claude have the same valuations for 'GOOD A' as in the English auction example, $\$30$, $\$50$ and $\$65$ respectively. The auction length is set at 5 minutes. There is no minimum bid, but the reserve is set at $\$35$. To demonstrate why verification is important, bids are not published and it uses a trusted auctioneer.

The auctioneer announces to the bidders and observers that bidding is open for 5 minutes with no minimum bid. The auction may proceed as follows:

1. The auction opens for bidding, the auctioneer sets a 5 minute timer.
2. Adam sends an encrypted bid of $\$30$ to the auctioneer.
3. Barry sends an encrypted bid of $\$50$ to the auctioneer.
4. Claude sends an encrypted bid of $\$65$ to the auctioneer.
5. The auctioneer's timer expires and the auctioneer notifies participants that the auction is closed.
6. Adam, Claude and Barry each send the decryption key for their bid to the auctioneer.
7. The auctioneer decrypts the bids, discarding Adam's bid as it is below the reserve. The auctioneer compares the other bids and then informs the participants that Claude has won 'GOOD A' at $\$65$.

The problem with this version of the protocol is that a corrupt auctioneer can easily change the auction outcome. For instance the auctioneer could award the auction to a losing bidder or award it at any price and no participant would be able to prove the auctioneer incorrect. In another version of the auction, bidders publish their bids and keys (once the auction closes) to a board, which allows the scheme to be publicly verifiable. Once the decryption keys have been published, all observers can check the allocation the auctioneer publishes.

3.1.3 A Combinatorial Auction with a Directed Graph

Combinatorial auctions are a type of auction used to sell multiple goods in a single auction of which there are many different protocols [18]. A sealed bid combinatorial auction can be run using a directed graph [68]. Unlike the first two protocols introduced in sections 3.1.1 and 3.1.2, bidders submit valuations for each combination of goods under auction. Bids are eventually added to the graph where vertices represent each of the different combinations of unallocated goods and edges represent allocations. Each path through the graph is a disjoint set of the different combinations, allocating all goods.

The communication pattern is the same as for sealed bid auctions, as the bidders submit their bids and decryption keys once. The auctioneer runs the winner determination algorithm once. Participant behaviour is more complex because the participants have to value sets of goods. The auctioneer must perform some pre-auction setup, generating the different combinations of goods and the graph. Bidders traverse the graph and assign valuations to edges of which the collection of their valuations is their bid. The auction closes after a set time at which point the auctioneer decrypts the bids, adding the highest valuations (over the reserve if used) for each edge to the graph. To determine the optimal solution, the auctioneer finds the highest cost path through the graph from the root node (which contains all of goods, unallocated) to an empty node with all goods allocated. The auctioneer publishes the highest bid for each edge as a winning bid.

An Example Auction

The auction offers two goods, 'GOOD A' and 'GOOD B' with the combinations $\{GOODA, GOODB\}$, $\{GOODA\}$ and $\{GOODB\}$. Two bidders, Adam and Barry have valuations for each combination of $(50, 20, 30)$ and $(0, 40, 15)$ respectively. Note that Barry is interested in one good or the

other but not both. The auction length is five minutes.

The auctioneer generates the three different combinations of goods and builds the auction graph. Each of the combinations becomes a vertex in the graph, directed edges connect each edge 'A' and 'B' where $|A| \geq |B|/2$. Duplicate paths should be removed for optimisation, along with any edges which become disconnected. The example graph provided in Figure 3.1 is optimised, with vertex C removed because edges (A, C) and (C, D) provide the same path as (A, B) and (B, D). The combinations and graph are published for retrieval by bidders. Each bidder iterates through the different goods combinations, finding its valuation for each. The bid is made up of the valuations referenced against the corresponding edges on the graph.

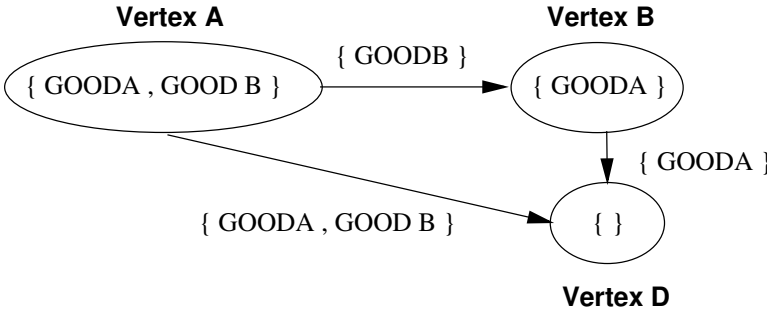

Figure 3.1: Basic optimised dynamic programming graph for two goods. Nodes represent unallocated goods, edges represent allocation of goods to bidders.

The auctioneer publishes the combinations and graph and announces to the bidders and observers that bidding is open (providing the combinations and graph) for 5 minutes with no minimum bid. The auction may proceed as follows:

1. The auction opens for bidding, the auctioneer sets a 5 minute timer.
2. Adam retrieves the combination and graph and forms the bid containing valuations for each edge: v(A, B) = $\$30$, v(B,D) = $\$20$ and v(A, D) = $\$50$. The bid is sent to the auctioneer.

3. Barry retrieves the combination and graph and forms the bid containing valuations for each edge: v(A, B) = $\$15$ and v(B,D) = $\$40$. His bid does not include a valuation for edge (A, D). The bid is sent to the auctioneer.

4. The auctioneer's timer expires and the auctioneer notifies participants that the auction is closed.

5. Adam and Barry each send the decryption key for their bid to the auctioneer.

6. The auctioneer decrypts the bids and adds each edge valuation to the graph where there is no valuation or the current valuation is lower. The graph is provided in Figure 3.2. The crossed out valuations were either dropped or replaced by higher valuations. The auctioneer finds the path from vertex A to D which is (A, B) and (B, D), with Adam winning 'GOOD B' and Barry winning 'GOOD A'. The auctioneer publishes the result.

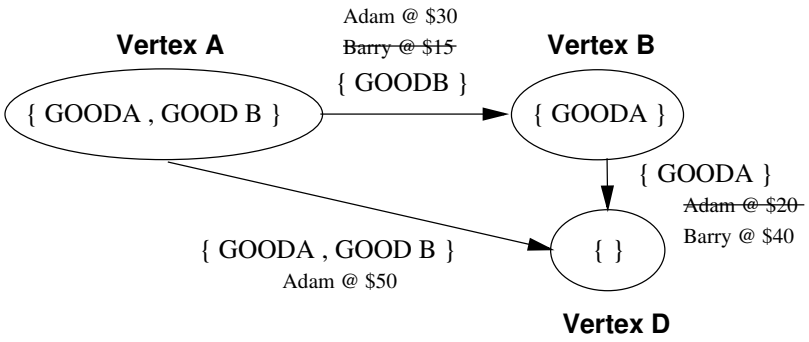

Figure 3.2: Basic optimised graph for two goods with bidder valuations.

3.2 Dynamic Programming for Combinatorial Auctions

The combinatorial auction protocol introduced in the previous section can be adapted to use a method of determining the optimal path, called dynamic programming [62]. The main advantage of dynamic programming is that it reduces the search space by removing paths which can not be optimal, but it is also a prerequisite to the first two secure combinatorial protocols studied in chapter 4.

Dynamic programming techniques solve complex problems by recursively solving subproblems. A graph can be used when each potential solution can be represented by a path on a graph with subproblems as sub-paths between two nodes. The optimal path is found by determining optimal sub-paths until the optimal path between the graph's source and destination node is discovered. The optimal value of a node is the most optimal of the incoming edges with their corresponding source nodes. This is possible because the optimal path between any node on the optimal path and the start node must be optimal. The search space is therefore reduced as not all potential solutions require searching, i.e. paths can be ignored where sub-paths can be discounted.

Dynamic programming with a graph can be used to solve the combinatorial allocation problem by representing allocations of goods to bidders on edges [62]. The optimal edge $f(a, b)$, between two nodes n_a and n_b is the edge between the two nodes with the highest bid. The optimal value of node n_b, $f(b)$, is the sum of the optimal edges from the start node n_0 to n_b, which is equivalent to the highest $f(a) + f(a, b)$ for all incoming edges (a, b). An optimal node value can only be discovered once the optimal values of each source node on incoming edges has been found. Note that the value of the start node $f(0) = 0$.

Example: Four goods $\{g1, g2, g3, g4\}$ are offered to bidders, for the different combinations $\{g1\}$, $\{g2\}$, $\{g1, g2\}$, $\{g3, g4\}$, and $\{g1, g2, g3, g4\}$. An

example auction graph is provided by Figure 3.3. It shows the edges between nodes with the highest bid, with the others trimmed. Two bidders b_0 and b_1 each value a subset of the possible combinations:

$Bidder$	$Bids$				
	$\{g1\}$	$\{g2\}$	$\{g1, g2\}$	$\{g3, g4\}$	$\{g1, g2, g3, g4\}$
b_0	4	2	7	1	9
b_1	1	5	6	–	–

The optimal values for each node are recursively calculated and the optimal path is discovered. For this example all nodes except n_3 have a single incoming edge simplifying evaluation. The optimal values paths for each node are:

$Node$	n_0	n_1	n_2	n_3
$Optimalvalue$	0	1	6	10
$Optimalpath$	$\{0\}$	$\{0, 1\}$	$\{0, 1, 2\}$	$\{0, 1, 2, 3\}$

When $f(0)$, $f(1)$ and $f(2)$ have been determined, $f(3)$ can also be determined. For each of the edges between n_3 and the other nodes, the total values are compared to find the optimal path from n_0 to n_3. The values to compare are 9, 8 and 10 for $\{(0, 3)\}$, $\{(0, 1), (1, 3)\}$ and $\{(0, 1), (1, 2), (2, 3)\}$ respectively. With standard pricing (see section 4.4.3), bidder b_0 is awarded $\{g1, g3, g4\}$ at $\$5$ and b_1 is awarded $\{g2\}$ also at $\$5$.

3.2.1 Edge Traceback

Instead of storing the optimal path during evaluation, an edge traceback step can be added to the process. During edge traceback the graph is traversed backwards, and optimal edges are detected where $f(a) + f(a, b) = f(b)$ from an optimal node n_b.

This is the method used for the two secure schemes SGVA (section 4.2) and polynomials (section 4.1) as the optimal edges cannot be determined during an initial pass through the graph. This technique can also be used

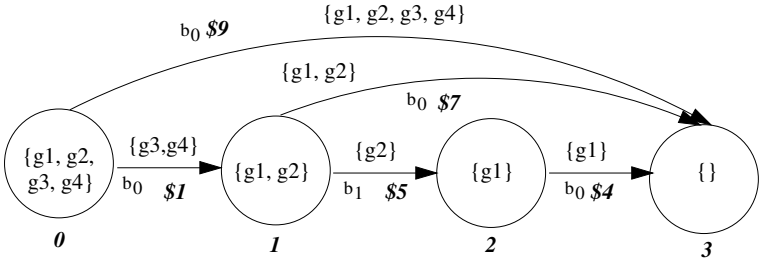

Figure 3.3: An example auction solved using dynamic programming. The highest bid for an allocation between nodes is provided with the bidder and amount bid.

to reduce the memory requirement in large graphs, as the optimal path needs to be stored only once, as opposed to storing an optimal path for each node.

3.2.2 Secure Dynamic Programming

Secure dynamic programming finds the highest cost path without revealing valuations for edges which are not on it. As dynamic programming requires only addition and comparison of values, if these are provided by an encryption scheme then dynamic secure programming is possible.

3.2.3 Graph Generation

Auction graphs can either be generated before the auction is run to be advertised with an auction description, or when bids are received. If an auction is advertised with a graph, the bidders relate bids to edges and the auctioneer adds the bids to the correct edges. If there is a node index, this is simple and efficient. On the other hand, it may require trimming before evaluation to remove edges containing combinations without bids. If the graph is built just before evaluation, the auctioneer can create an optimal graph once, but needs to search through all the bids to find matching combinations.

For the secure auction schemes discussed in this thesis, the graph must be created in advance as separate evaluators are assigned to nodes with specific encryption information. For the remaining part of this thesis, the graph is assumed to be generated in advance, although it does not affect the graph generation technique.

In [62, 68] the graph generation technique used creates a node for each combination of goods under offer. An edge connects node n_a to n_b where the goods from each: $G(a)$ and $G(b)$ respectively, meet the following criteria: $G(b) \subset G(a)$ and $|G(b)| \geq |G(a)|/2$. The set of goods allocated on the edge is the difference between the two nodes $G(a) \setminus G(b)$. This method generates an inefficient graph as duplicate paths and unnecessary nodes are included in the graph, increasing generation and graph traversal cost. An example graph for two goods using this method is shown in Figure 3.4, with an optimised version in Figure 3.5. An optimised graph can be created by trimming the excess edges and nodes from the graph. This however can be memory intensive and time consuming as determining duplicate paths in graphs requires a full traversal of each path, storage for discovered paths, and a search through the storage for each path.

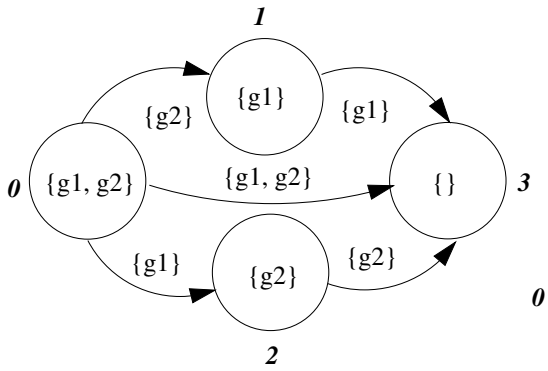

Figure 3.4: Naive graph.

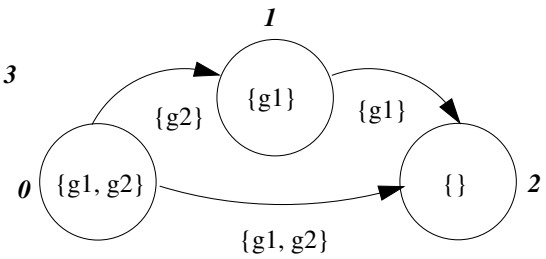

Figure 3.5: Optimal graph example 1.

A graph generation technique is now introduced, which generates an optimal graph in the first instance. To evaluate a combinatorial auction, every possible allocation of goods must be represented by a path through

the graph. Allocations are generated of disjoint allocation subsets. Each allocation is then recursively added to the graph. The source node for the allocation subset is the previous node added, or the node containing all goods if it is the combination's first allocation subset. The unallocated set contains the goods within the destination node, which is created if one containing the goods does not already exist. If there is already an edge between the source and destination nodes, then this allocation is skipped and the process continues with the next allocation subset and the destination node. When the end node is reached, the next allocation is added to the graph and this continues until all allocations have been added.

Example: Three goods are under auction $\{g1, g2, g3\}$. The different allocation subsets are: $\{g1\}$; $\{g2\}$; $\{g3\}$; $\{g1, g2\}$; $\{g1, g3\}$; $\{g2, g3\}$ and $\{g1, g2, g3\}$. The different allocations are shown in Figure 3.6.

The initial graph is created with the start and end nodes containing the sets of goods $\{g1, g2, g3\}$ and $\{\}$ respectively. For the first subset of the first allocation, a node is created for the unallocated goods $\{g2, g3\}$ and the first edge spans from the start node to this new node allocating $\{g1\}$. The second edge placed spans from the node just created to another new node containing $\{g3\}$ allocating $\{g2\}$. The final edge for this allocation spans from the second node to the end node and allocates the remaining good $\{g3\}$. The complete optimal graph is shown in Figure 3.7.

An alternative solution is to store graph templates for frequently used auctions. Optimal templates can be generated in advance and loaded when required.

3.3 An Introduction to GAF

Without a unified approach and reusable code, auction protocol development and system integration requires strenuous repetitive development. With many different auction protocols differing significantly, but with the same goal, auction systems can benefit from a general auction framework.

	Allocations
1	$\{\{g1\},\{g2\},\{g3\}\}$
2	$\{\{g1\},\{g2,g3\}\}$
3	$\{\{g2\},\{g1,g3\}\}$
4	$\{\{g3\},\{g1,g2\}\}$
5	$\{\{g1,g2,g3\}\}$

Figure 3.6: Possible allocations.

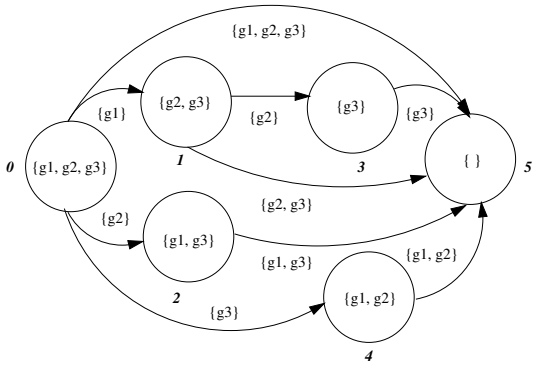

Figure 3.7: Optimal graph example two.

Frameworks define the variable and persistent aspects of a system domain, providing a consistent design and reusable components. GAF is a framework which facilitates reuse of protocol utilities and basic components for protocol developers, as well as allowing protocol integration within different systems for minimum effort.

The varying aspects of a framework are called its hotspots [49, 54], while the persistent aspects are called its frozen spots [54]. Protocol developers build their protocols within a GAF implementation by filling in protocol hotspots. Application developers can then easily include different protocols within applications as well as being able to switch between protocols dynamically. This reduces effort during protocol development, protocol integration and testing. Although it is not possible to make auction protocols within GAF completely plug-and-play for every application without restricting protocols, integration of each protocol should require little to no additional work.

Protocols implemented in GAF are defined in terms of auction participants and resources. Auction resources are distributed components which run an auction, such as an auctioneer or evaluator. Some resources are prescribed but additional types can also be added as required. Figure 3.8 shows the different resources and participants divided between the frame-

work, protocol and application. The framework contains the system interfaces, frozen spot implementations and default hotspot implementations. Protocols implement the majority of the framework hotspots, while an application using GAF implements bidders and auction owners, as well as optional observers and/or verifiers.

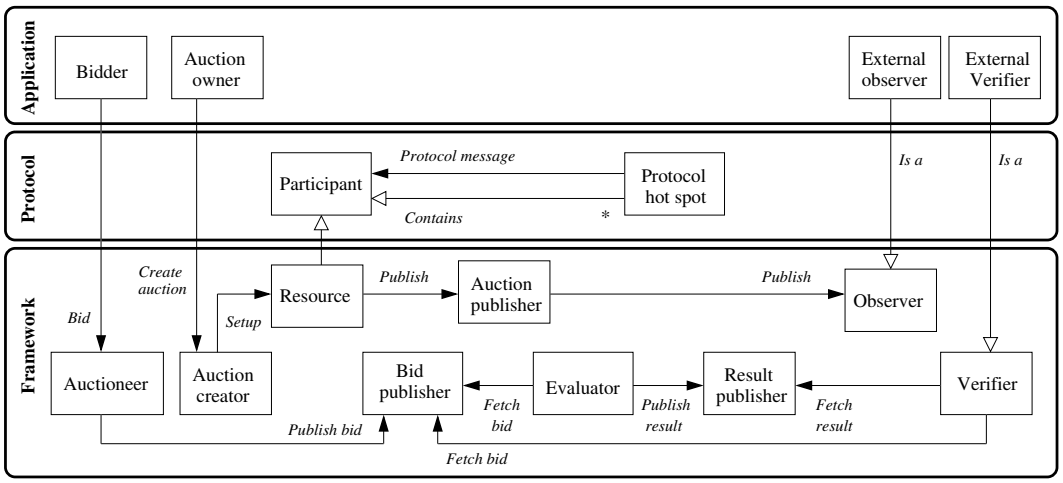

Figure 3.8: GAF overview. For brevity the diagram does not show auctioneer, auction creator, bid publisher, auction publisher, evaluator and result publisher as resources, nor observer and verifier as participants. For a list of framework hotspots see section 5.4.

The different auction activities are divided into six required resources with separate interfaces. The auction creator instantiates the auction, setting up the different resources provided by the application via the owner. It is the auction creator that the auction owner interacts with. An auctioneer is responsible for the collection of bids and auction facilitation such as managing phases throughout the auction. The bid publisher records bids, the result publisher records the result and the auction publisher manages events. It may be the case that a protocol only requires one resource which implements all of the interfaces. However separation of concerns is important as some protocols need to distribute behaviour. For instance, a

protocol implementation may require one auction manager but multiple auctioneers located throughout a distributed system (for example, near different groups of bidders). Another probable example would be a resource implemented by plural servers for redundancy.

Resources communicate using standard interfaces as well as custom events and protocol messages. Although the framework may be used in a variety of different distributed applications, using different communication types (such as Sockets or Remote Method Invocation), the protocol developer ignores the method of communication.

GAF offers several low level services to facilitate auction protocol development. Two examples are authentication and authorisation, both important for auction systems. Resources should be able to identify participants making requests, as well as work out what they are authorised for. Although it has not been developed yet, future development of GAF will include a sub-framework to accomplish this.

As a proof of concept three protocols have been included within GAF; combinatorial auctions using threshold homomorphic encryption [13, 69], combinatorial auctions using threshold polynomials [13, 62], and Garbled circuits for combinatorial auctions [38]. A prototype application using GAF to test and collect statistics for auctions of each type are presented in chapter 7.

Chapter 4

Secure Combinatorial Auction Protocols

There are many secure auction protocols offering different types of security [40]. One such type is privacy-preserving auction protocols. Privacy-preserving auctions maintain bidder privacy for valuations of non-winning bids. This chapter describes three privacy-preserving combinatorial secure auction protocols which have been included as case studies within GAF. The first two protocols were implemented for this thesis, both using dynamic programming with graphs, while the third was implemented by Palmer [40] and wrapped to be used in GAF.

Section 4.1 describes a combinatorial auction protocol, which represents bidder valuations in the degree of a polynomial [13, 62]. The polynomials are divided between several evaluators using a method which prevents bid resolution without evaluator concensus. Section 4.2 describes a similar protocol [14, 69], which represents bids as a vector of values encrypted with a threshold cipher. Similarly to the polynomial protocol, multiple evaluators must agree to decrypt bids. Section 4.3 outlines a protocol where the bid processing uses a garbled circuit of logic gates [38, 40]. This protocol is built in such a way that three parties must collude in order to decrypt bids that should not be decrypted. Section 4.4 discusses

limitations of each of the protocols.

4.1 Case Study One: Polynomial Auction Protocol

Suzuki and Yokoo [62] developed a secure combinatorial auction protocol where bids are encoded in the degree of a polynomial. The protocol is a (t, n) threshold scheme where at least t of n evaluators are required to decrypt bids. Bids are divided between evaluators using well known threshold polynomial techniques [56] into evaluator shares which are the polynomials solved with unique evaluator resolve values. If enough evaluators publish their shares, bids are reconstructed using interpolation by consensus. This required consensus provides the security of the protocol, where decryption should only take part at specific points of the protocol.

Dynamic programming is possible using polynomial encryption, as the largest degree of two polynomials can be found using addition, and the sum of the degree of two polynomials found using multiplication. Each auctioneer finds the total of the graph using their shares with a combination of addition for alternate paths and multiplication to add edge costs together. Auctioneers cooperate during a traceback step to determine polynomial degrees for optimal edges using Lagrange interpolation.

Some aspects of the protocol are not adequately described in the original literature but the scheme is better described in [13, 12] which is reproduced here with more detail. A description is provided with a worked example illustrating the scheme.

4.1.1 Protocol Description

This description divides the scheme into five phases, exploring each with a worked example. The phases are: initialisation; bid submission; evaluator setup, total value determination and optimal path determination.

Phase 1: During initialisation, information is published by the auction initiator and evaluators for bid generation.

When evaluators are created, they each generate a unique resolving value. This value is published to a bulletin board for solving polynomials by bidders and mask publishers. Other information published for bidders includes the details of goods under auction, the different combinations of goods available and an initial auction graph. A constant value c is published for weight resolution as well as the threshold value t.
Example: Ten evaluators $\{e_1, e_2, \ldots e_9, e_{10}\}$ publish resolve values $\{1, 2, 3, \ldots 9, 10\}$. The auction initiator publishes details of two goods $\{g_1, g_2\}$, the three available combinations ($\{g_1\}, \{g_2\}, \{g_1, g_2\}$), an initial graph and the constant value 100. The threshold value and maximum polynomial coefficient are both set as 1 for simplicity. These details are illustrated in Figure 4.1. Two mask publishers are used.

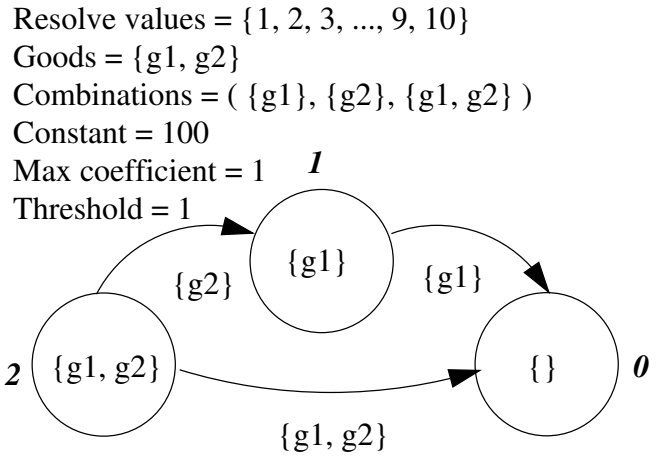

Figure 4.1: An initial graph for two goods.

Phase 2: After the initilisation phase, bidders generate and submit bids to each evaluator.

Bidders calculate their valuations for all combinations of goods they are interested in. Iterating through the edges in the published graph, a weight is generated based on the valuation for each edge's goods combination. To form the weight w a threshold modifier $t(j-i)$ is added to the valuation where j and i are the identification numbers of the source and destination nodes respectively. The threshold modifier increases the number of shares required to interpolate by the threshold t, and it is multiplied by the edge identifier difference so that the total amount added is equal on each path.

A bid is generated for each evaluator for each set of goods that the bidder is interested in: a random polynomial of degree w and constant 0 solved with the evaluator's resolve value. The solved value s is sent with the node identifiers to the evaluator. As each evaluator receives bids, a new edge is created on their copy of the graph from j to i with weight s.
Example: Valuations for two bidders b_0 and b_1 for the three combinations ($\{g_1\}, \{g_2\}, \{g_1, g_2\}$) are shown in Table 4.1. The bidders iterate through the graph, creating weights by adding the threshold modifiers shown in Table 4.2. The polynomials used in this example have coefficients of one, which compromises security by reducing attacker search space but simplifies the example. The random polynomials for b_0 and b_1 are:

$$b_0 = \begin{array}{ll} (2,0) & x^6+x^5+x^4+x^3+x^2+x \\ (2,1) & x^3+x^2+x \\ (1,0) & x^2+x \end{array} \qquad b_1 = \begin{array}{ll} (2,0) & x^4+x^3+x^2+x \\ (2,1) & x^2+x \\ (1,0) & x^2+x \end{array}$$

The two bidders resolve their polynomials with each evaluator's resolve values as shown in table 4.3. and send them to each respective evaluator. The evaluators store their bids on their own copy of the auction graph. The graph maintained by evaluator e_1 is shown in Figure 4.2.
Phase 3: Evaluators setup for evaluation, calculating their shares of the optimal value of each node.

Bidder	$\{g_1\}$	$\{g_2\}$	$\{g_1, g_2\}$
b_0	1	2	4
b_1	1	1	2

Table 4.1: Bidder valuations.

Edge	b_0	b_1
$(2, 1)$	2 + 1 (2 - 1) = 3	2
$(2, 0)$	4 + 1 (2 - 0) = 6	4
$(1, 0)$	1 + 1 (1 - 0) = 2	2

Table 4.2: Bidder edge valuations with added threshold modifier.

	Bidder b_0			Bidder b_1		
Evaluator	$(2, 1)$	$(2, 0)$	$(1, 0)$	$(2, 1)$	$(2, 0)$	$(1, 0)$
$e_1(1)$	$1^3 + 1^2 + 1 = 3$	6	2	2	4	2
$e_2(2)$	$2^3 + 2^2 + 2 = 14$	126	6	6	30	6
$e_3(3)$	$3^3 + 3^2 + 3 = 39$	1092	12	12	118	12
...	...	...	...	...	...	...
$e_9(9)$	$9^3 + 9^2 + 9 = 819$	597870	90	90	7380	90
$e_{10}(10)$	$10^3 + 10^2 + 10 = 1110$	1111110	110	110	11110	110

Table 4.3: Bids for the two bidders b_0 and b_1. The first column for b_0 shows the full calculation.

The optimal value of the auction is the sum of bidder valuations on the greatest path through the graph. As the bids are distributed amongst the evaluators, valuations must be reconstructed by interpolating shares published by evaluators. An evaluator's share of the optimal value is the sum of the paths through their copy of the graph. There are two properties of polynomials that make the calculation possible:

- The degree of a polynomial resulting from the addition of two polynomials is equal to the larger degree of the two.
- The degree of a polynomial resulting from the multiplication of two polynomials is equal to the sum of the degrees.

An evaluator's share is found using dynamic programming and then

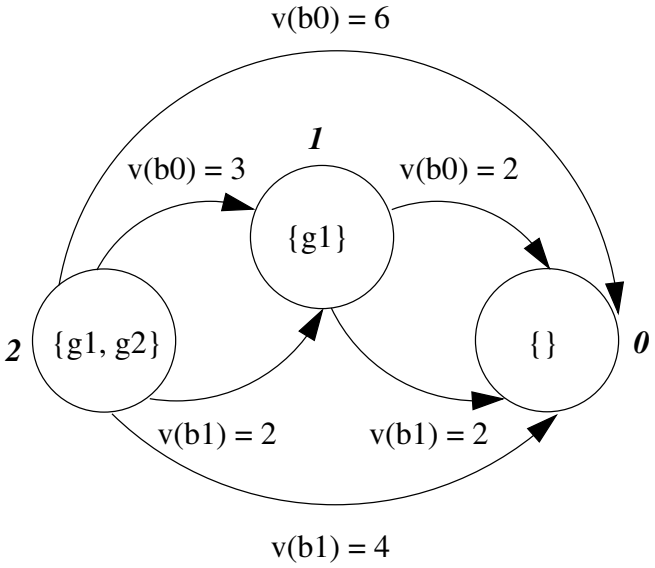

Figure 4.2: Graph maintained by e_1 after receiving bids from b_0 and b_1.

published. The sum of all paths in the graph is the cost of node n_0 denoted $f(0)$. Secure dynamic programming for this protocol works as follows: iteratively calculate the cost of each node from root node n_N (where N is the number of nodes). The node cost $f(x)$ is the sum of the cost of all alternative paths to node n_x from node n_N. The cost of a path from node n_x to n_{x-1} where the nodes are connected by at least one edge is $f(x-1)$ multiplied by the sum of the cost of all alternative edges between the nodes. Note that $f(N) = 0$.

Example: Evaluator e_1 calculates its share of the node costs.

$$f(2) = 0$$
$$f(1) = 3 + 2 = 5$$
$$f(0) = ((2 + 2) * 5) + (4 + 6) = 30$$

Node costs for all evaluators are given in Table 4.4.

Phase 4: Binary search is used to discover the optimal value.

Using Lagrange interpolation, a polynomial can be recovered with a de-

Evaluator	$f(2)$	$f(1)$	$f(0)$
e_1	0	5	30
e_2	0	20	396
e_3	0	51	2436
...	...	...	...
e_9	0	909	768870
e_{10}	0	1220	1390620

Table 4.4: Evaluator node costs.

gree of one less than the number of shares used. The constant value of the polynomial is used as the indicator in a modified form of binary search.

As the threshold modifier is added to each of the edges, the sum of the modifiers is the minimum optimal value. The initial smallest possible optimal value s therefore is $t(N-1)$ and the maximum optimal value m is $E-1$. A recursive binary search calculates the current search value $d = \lfloor (m-s)/2 \rfloor + s$ and requests $d+1$ evaluators publish masked shares. Masks are created by M masking agents who generate a random polynomial of degree $d+1$ and constant c. If the lagrange polynomial using $d+1$ shares has a constant equal to $M * c$, the optimal value is less than or equal to d and m becomes $d-1$. If the constant is not equal to $m*c$ the optimal value is greater than d and s becomes $d+1$. Without being able to directly check whether d is the optimal value, it is found where $d \leq o$ and $d+1 > o$.

The optimal value is different from the actual outcome of the auction as valuations are modified in step two with the threshold modifiers. The total price paid will be $d-(t(N-1))$, i.e. subtract the threshold modifier. Only one evaluator needs to perform and publish the interpolation, although other evaluators may wish to verify that the correct result is published.

Example: The minimum optimal value is 0, the maximum is 9 and initially $d = 4$. Masks are sent to the evaluators from the mask publishers. The masks are polynomials of degree 4 and coefficient 1 (again for sim-

plicity) which are resolved with the respective evaluator's resolve value. Because random coefficients are not used in this example, both mask publishers generate the same masking polynomial $x^4 + x^3 + x^2 + x$ and the evaluators have the following total masks:

e_1	208
e_2	260
e_3	440
...	...
e_9	14960
e_{10}	22420

Each evaluator then adds the total masking value to their share of the optimal value and publishes it. If evaluators e_1, e_2, e_3, e_9 and e_{10} publish their shares then the published shares are:

e_1	$30 + 208 = 238$
e_2	$396 + 260 = 656$
e_3	$2436 + 440 = 2876$
e_9	$768870 + 14960 = 783830$
e_{10}	$1390620 + 222420 = 1413040$

The lagrange polynomial from these five points is $1988x^4 - 15211x^3 + 46185 - 60334x^2 + 28480$. As the constant is not equal to 100, $o > 4$. Table 4.5 shows the binary search for the optimal value which finds that the optimal value is 6.

Phase 5: An optimal path is traced from node n_{N-1} to the root node n_0 using binary search.

Once the optimal value is discovered, an optimal path can be traced to discover an optimal allocation of goods. There can be multiple optimal paths and allocations but this protocol returns the first found. It is possible to modify the protocol to find all of the optimal paths but they would be meaningless without revealing more information (i.e. the bidders or a

d	Polynomial	Constant	
4	$195x^4 - 1293x^3 + 3784x^2 - 4808x + 2360$	2360	$o > 4$
7	$2x^7 + 3x^6 + 5x^5 + 10x^4 + 12x^3 + 8x^2 + 4x + 200$	200	$o \leq 7$
5	$26x^5 - 165x^4 + 747x^3 - 1616x^2 + 1768x - 520$	-520	$o > 5$
6	$3x^6 + 5x^5 + 10x^4 + 12x^3 + 8x^2 + 4x + 200$	200	$o \leq 6$

Table 4.5: Using binary search to find the optimal value.

time stamp which could be used to match bids to bidders). This would contradict the primary goal of privacy preservation.

As the cost of an optimal node n_x is the greatest path cost from n_x to the root node, the cost of an optimal edge added to the cost $f(y)$ of a connected node y is $f(x)$. Starting with $x = N - 1$, and $f(N-1) = o$, an optimal path is found by iteratively evaluating the cost of each edge from x added to the cost of the destination node. Binary search as in step four is used to check edges from x for $f(x)$.

Edge costs will differ from the actual cost and are modified accordingly by subtracting the threshold modifier.

Example: The edges from n_0 are searched and all but $(2, 0)$ are not equal to $f(0)$. Once this edge is found the search ends because the destination is n_2. The actual cost is $6 - (1(2 - 0)) = 4$ and the optimal allocation is published as b_0 wins $\{g_1, g_2\}$ at \$4.

4.2 Case Study Two: Homomorphic Auction Protocol

In a Homomorphic encryption scheme there exists an algebraic operation which is transitive from cipher-text to its corresponding plain-text. The Homomorphic auction protocol [14, 69] encrypts bids as vectors of homomorphic ciphers which are transitive on multiplication. Vectors are con-

structed of a series of encrypted public constant values followed by a series of encrypted 1s. Such vectors can be multiplied together, resulting in a vector of the largest value. This technique allows paths in an auction graph to be compared without complete decryption. With these vectors and other techniques described in this section, the protocol implements secure dynamic programming.

To preserve the privacy of losing bids, the auction protocol uses a threshold homomorphic encryption cryptosystem. Each node in the auction graph has a set of assigned evaluators, each set having a shared public key and a private key for each evaluator. Bids are made for each edge encrypted with the public key of the destination node of the evaluation group. Evaluators collaborate when required to decrypt vectors or vector elements.

4.2.1 Vector Preliminaries

Representation

Homomorphic bid vectors are made from a series of encrypted constants (not equal to one) followed by a series of encrypted 1s. Each element in the vector is encrypted with the public key PK^+ of the relevant evaluator group. The value of the vector is equal to the number of elements not equal to one. For example, a bid of three in a vector with a maximum bid of five and constant 2 is represented by: $PK_0^+(2), PK_0^+(2), PK_0^+(2), PK_0^+(1), PK_0^+(1)$. It is important that the homomorphic cryptosystem used must produce unique ciphertext each time the same value is encrypted so that vectors are indistinguishable.

The maximum bid is the length of the vector which must be decided before bidding begins. Variable length vectors could indicate value.

Comparison

Vectors need to be compared for alternate bidders on edges and alternate paths to nodes. Two vectors cannot be directly compared without decryption, instead a vector encrypting the higher of two values can be computed with multiplication if they have been encrypted with the same key. For instance, if two bidders bid four and two for the same edge then they would be multiplied as follows:

$$\begin{array}{rl} & PK_0^+(3), PK_0^+(3), PK_0^+(3), PK_0^+(3), PK_0^+(1) = 4 \\ \times & PK_0^+(3), PK_0^+(3), PK_0^+(1), PK_0^+(1), PK_0^+(1) = 2 \\ \hline & PK_0^+(9), PK_0^+(9), PK_0^+(3), PK_0^+(3), PK_0^+(1) = 4 \end{array}$$

The result shows that the highest of the two vectors is four. The new vector can now be decrypted, and the value of the higher vector determined without knowledge of what vector it is.

Addition (Left-shifting)

A constant value v can be added to an encrypted vector by left shifting the vector by the value required. v elements are removed from the right side, and v public constants encrypted with the same key as the vectors are added to the left. A modified vector must be 'randomised' by multiplying unchanged elements by an encrypted one. When a modified vector is added to the bulletin board, randomisation prevents participants from counting the number of modified elements.

Cryptosystem

The implementation of homomorphic vector encryption in this thesis uses the ElGamal cryptosystem [55]. The original Elgamal specification can be modified and used as a threshold cryptosystem [56].

4.2.2 Protocol Description

Phase 1: An auction graph is generated as per section 3.2.3. An auctioneer and evaluators are chosen, and the evaluators assigned nodes. Dependent on security requirements, there are n evaluators per node, where a threshold t of them is required to decrypt bids on incoming edges. For each node v, a shared public key ${PK^+}_v$ is generated along with n private keys ${PK^{-1}}_v$, ${PK^{-2}}_v$,..., ${PK^{-n}}_v$ for each evaluator. Information published for bidders are: the graph with public keys mapped to edges, the bid vector length B and a common value C used for encryption.

Example: An optimised graph is shown in Figure 4.3, created for two goods; $\{g1, g2\}$ and three bundles; $\{g1\}$, $\{g2\}$ and $\{g1, g2\}$. The graph is published with $B = 5$ and $C = 3$.

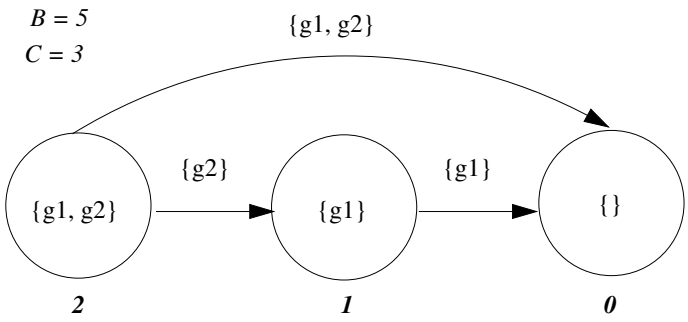

Figure 4.3: An optimised graph for two items.

Phase 2: Bidders value each of the offered bundles, generating bids for each edge containing a bundle they are interested in and submit bids to the auctioneer. A bid contains a homomorphic bid vector encrypted with public key ${PK^+}_v$ of the edge (the shared public key of the destination node) with a reference to the edge. If the bidder's valuation is v, the homomorphic bid vector contains C encrypted, v times followed by 1 encrypted $B - v$ times.

Example: Bidders b_0 and b_1 generate valuations and bid vectors for each

	Edge	Goods	Valuation	Bid Vector
b_0	(n_2, n_0)	$\{g_1, g_2\}$	4	$PK_0^+(3), PK_0^+(3), PK_0^+(3), PK_0^+(3), PK_0^+(1)$
	(n_2, n_1)	$\{g_2\}$	2	$PK_1^+(3), PK_1^+(3), PK_1^+(1), PK_1^+(1), PK_1^+(1)$
	(n_1, n_0)	$\{g_1\}$	1	$PK_0^+(3), PK_0^+(1), PK_0^+(1), PK_0^+(1).PK_0^+(1)$
b_1	(n_2, n_0)	$\{g_1, g_2\}$	2	$PK_0^+(3), PK_0^+(3), PK_0^+(1), PK_0^+(1), PK_0^+(1)$
	(n_2, n_1)	$\{g_2\}$	1	$PK_1^+(3), PK_1^+(1), PK_1^+(1), PK_1^+(1), PK_1^+(1)$
	(n_1, n_0)	$\{g_1\}$	1	$PK_0^+(3), PK_0^+(1), PK_0^+(1), PK_0^+(1).PK_0^+(1)$

Table 4.6: Bids for b_0 and b_1.

of the edges, shown in Table 4.6. The bids are packaged and sent to the auctioneer.

Phase 3: Once bidding has closed, evaluation using dynamic programming begins by recursively discovering the optimal values of each node. The optimal value of a node d, $f(d)$ is the sum of the edges on the highest cost path from the end node n_n to n_v. The optimal value of the graph is the optimal value of the root node n_0 which provides the total of the highest cost path in the graph. An optimal edge value is the optimal value of its source node, added to the highest bid vector for that edge. The highest incoming optimal edge value for each node is computed, which is the decryption of the multiplicative sum of all incoming bid vectors left-shifted by the optimal value of their source node.

The evaluators of a node each decrypt the highest incoming optimal edge value using their private keys, and publish their share. Once t shares have been published, the optimal value of the node can be decrypted by an evaluator of that edge and it is added to all outgoing edges by left-shifting and randomising. These new vectors are published as the incoming edges for their respective destination nodes. Edges for which there are no bids should have a dummy bid of 0 added, so missing edges do not break paths.

Example: As there are no incoming edges for n_2, $f(2) = 0$, the bid vectors

on the outgoing edges from n_2 do not need to be left shifted. The corresponding bid vectors for the destination nodes n_0 and n_1 are multiplied together:

$$\begin{array}{rl} & PK_0^+(3), PK_0^+(3), PK_0^+(3), PK_0^+(3), PK_0^+(1) \\ \times & PK_0^+(3), PK_0^+(3), PK_0^+(1), PK_0^+(1), PK_0^+(1) \\ \hline & PK_0^+(9), PK_0^+(9), PK_0^+(3), PK_0^+(3), PK_0^+(1) \end{array}$$

$$\begin{array}{rl} & PK_1^+(3), PK_1^+(3), PK_1^+(1), PK_1^+(1), PK_1^+(1) \\ \times & PK_1^+(3), PK_1^+(1), PK_1^+(1), PK_1^+(1), PK_1^+(1) \\ \hline & PK_1^+(9), PK_1^+(3), PK_1^+(1), PK_1^+(1), PK_1^+(1) \end{array}$$

These are published as the highest bid vectors from n_2 to n_0 and n_1 respectively. The evaluators for n_1 can now determine the optimal value, as the only incoming optimal edge value has been published. The evaluators each use their private key to decrypt a share of the bid vector. Once t shares have been published, the optimal value is decrypted from the shares to find that $f(1) = 2$. The outgoing edge to n_0 is left shifted by two, randomised and then published. The evaluators of n_0 multiply the two incoming edges together:

$$\begin{array}{rl} & PK_0^+(3), PK_0^+(3), PK_0^+(3), PK_0^+(3), PK_0^+(1) \\ \times & PK_0^+(3), PK_0^+(3), PK_0^+(3), PK_0^+(1), PK_0^+(1) \\ \hline & PK_0^+(9), PK_0^+(9), PK_0^+(9), PK_0^+(3).PK_0^+(1) \end{array}$$

and perform shared decryption, finding that $f(0) = 4$. This value is published as the optimal value of the graph.

Phase 4: The total optimal value is used to trace back the optimal path through the optimal bid vectors. An edge is on the optimal path, if its optimal bid vector is equal to $d = f(v)$ where v is the edge's destination node. It is possible to discover this without decrypting bid values of edges not on the winning path by only checking position d of the optimal edge vector for a value greater than one. The process starts from the root node n_0 and works backwards towards n_n. When an optimal edge is found, its

source node is published and the incoming edges of that node are checked. When n_n is reached, the search ends. The decryption is performed in the same manner as in phase 3, with evaluators publishing shares of the bid vector position.

Example: Position 4 of n_0's incoming optimal bid vectors are checked for a value greater than 1. The evaluators each publish their shares and the decrypted values are found as 3 and 1 for edges $(2, 0)$ and $(2, 1)$ respectively. n_1 is published as not optimal, 2 is published as optimal and because $n_2 = n_n$, the optimal path has been found and it is also published as the single edge $(2, 0)$.

Phase 5: Once the optimal path has been traced back the bidders for each optimal edge can be found. Using the original bid vectors for each of the optimal edges, the same technique used in phase 4 to identify optimal edges is used to identify optimal bid vectors and hence the winning bidders. The difference is that the position to check is the actual value that the edge provides, i.e. $f(destination) - f(source)$.

Example: Position $4 : (4 - 0)$ for both bids on $(2, 0)$ are checked for a value greater than 1. The value from b_0's vector is 4 and the value from b_1's vector is 2. The result is published; $b0$ having won both goods $\{g1, g2\}$ for $4.

4.3 Case Study Three: Garbled Circuits Auction Protocol

Garbled circuits [38, 40] is a mechanism for securely computing the outcome of an auction using two primary parties: an auctioneer and an evaluator. It is designed so that communication between bidders and the auction evaluator is performed through a proxy participant. The proxy transfers required information between the auctioneer and evaluator retaining information which is not required.

A circuit can be built to solve a combinatorial auction while maintaining the privacy of losing parties [40] through an evaluation protocol called verifiable proxy oblivious transfer (VPOT) [38]. This scheme is equivalent to a $(2, 2)$ threshold protocol, where if the auctioneer does not collude with the evaluator, privacy is guaranteed.

The scheme contains three phases. During setup, the auction is advertised and the garbled circuit created. Once the bidding phase is complete the VPOT protocol is run, with the different participants communicating via the proxy. After the VPOT protocol is complete the allocation is calculated by the proxy and then published. There are four parts to the VPOT protocol. The proxy first retrieves commitment values for each of the bidders from the evaluator and sends them to the bidders. The bidders then send bid proofs for the commitment values to the proxy. One portion of the bid proof is kept by the proxy and the other portion is passed to the evaluator who computes the garbled input. The garbled input is sent to the proxy who verifies it before determining the solution using the garbled input and circuit.

4.4 Secure Auction Protocol Limitations

This section comments on the limitations of the secure protocols described in this chapter. The limitations are in regards to the lack of extensibility of each of the first two protocols.

4.4.1 Bid Scaleability

In polynomial auctions the number of bidder valuations which can be represented is limited as the maximum bid is directly related to the number of evaluators. This is because the valuation of an edge or node is discovered using interpolation upon a number of evaluator shares greater than its cost by one. As each edge represents a bid on a set of a least one good,

the graph's maximum length is equal to the number of goods G.

The maximum bundle valuation is further limited by the threshold modifier. The threshold modifier is added to the valuation so that the minimum number of shares required to decrypt is greater than t. If the same threshold modifier was used for all edges then the sum of the modifiers would differ between paths and the optimal value would not be able to be determined. By multiplying the threshold with the difference between two node identifiers, the threshold modifier sum will be equal on each path. The threshold modifier q is $q = t(j - i)$ where j and i are the nodes connected by the edge. As the smallest node identifier is 0 the largest value of $j - i$ is the largest node identifier $N - 1$. The maximum valuation v is therefore $v = E - 1 - t(N - 1)$ and the maximum valuation per good is $\lfloor v/G \rfloor$. The maximum pricing does not have to be divided equally between the goods as long as it is stipulated to bidders in advance. However, no matter how the maximum pricing is divided between goods, it is still all but completely unusable for the average auction.

In comparison the maximum bid in the Homomorphic protocol is the length of the vector, as each element in a bid vector represents one. This provides a more scalable bid representation than for the polynomial scheme. In the Garbled Circuit protocol the possible maximum bid is even greater where it is 2^b where b is the number of bits used. Increasing the number of bits used to represent prices, increases evaluation cost a small amount linearly as can be seen in Figure C.9.

4.4.2 Polynomial Threshold Property

The polynomial protocol is a threshold scheme in the sense that at a minimum t of n evaluators are required to decrypt a bid. However, more often than not, more than t evaluators are required. As the threshold modifier must be evenly distributed amongst edges, the threshold must always be less than $(E-1)/(N-1)$ where E is the number of evaluators and N is the

number of nodes in the graph. This is quite limiting for example with just three goods even a $(2, 8)$ threshold scheme is not possible: Say that with a scaler modifier x from the domain of natural numbers, $2x$ of $8x$ evaluators are required to decrypt bids and the number of nodes is six (an optimised graph). Trying to determine a suitable x, $2x < \frac{8x-1}{5} \equiv 10x < 8x-1$ it can be seen that this is not possible. The problem remains the same as the number of goods increases as the number of nodes will increase exponentially.

The number of evaluators required can be reduced or the threshold increased by modifying the threshold property so that it is proportional to the number of goods instead of the number of nodes in the graph. This reduces the effect of the threshold modifier from an exponential increase in evaluators to a linear increase when increasing the number of goods. The purpose of the threshold modifier is to increase each bid by at least the threshold but keep the total threshold modifier added on each path the same. Instead of multiplying the threshold by the difference in node identifiers to add to a bid, a much more efficient method is to multiply the difference in size between the two nodes. Before where the threshold modifier was $q = t(j-i)$ it would not be $q = t(|n_j| - |n_i|)$ where j and i are the nodes connected by the edge. Although in almost all cases the threshold number of evaluators required to decrypt will decrease, the security of the protocol is not reduced. This modification to the protocol would bring the protocol more inline with what the threshold property was intended to achieve.

With this modification the maximum threshold modifier will become equal to the size of the largest node, and therefore will be equal to the number of goods. The maximum threshold will become $(E-1)/G$ where G is the number of goods. With the same example from above it can be seen that a $(2, 8)$ threshold scheme is now possible for three goods ($6x < 8x-1$), though if the number of goods is increased then the ratio between the threshold number of evaluators and the total number of evaluators must be reduced. However as stated the threshold ratio will be reduced linearly.

Figures 4.4 and 4.5 show the difference in threshold modifiers for a graph of three goods using the two approaches to adding threshold modifiers. As can be seen in Figure 4.5 some bids still require more than t evaluators for decryption. The protocol could be further modified to use additional nodes attached to the end node n_0 which filter incoming edges, increasing the path cost by the amount required for the total threshold modifier to be Gt.

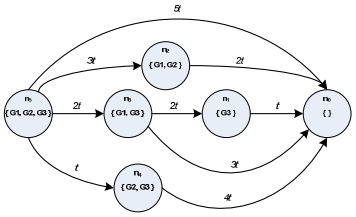

Figure 4.4: Threshold modifier for an optimised three good graph using node identifiers.

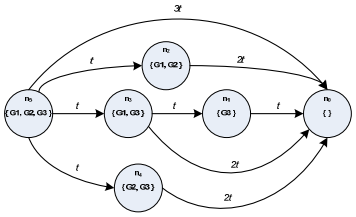

Figure 4.5: Threshold modifier for an optimised three good graph using node size.

In addition to the increased maximum threshold the maximum bid can also be increased. Where the maximum amount for a bid was previously $\frac{E-1-t(N-1)}{G}$, with the modification the maximum bid is $\frac{E-1-tG}{G}$. If the number of evaluators and threshold remains the same between the two ver-

sions of the protocol then with a threshold of three, an increased maximum bid by one for a three good auction with fourty evaluators. With one hundred evaluators, a threshold of three and four goods the maximum bid would be increased from thirteen to twenty-two. Testing and implementation of this modification is left to future work.

4.4.3 GVA Pricing

Generalised Vickrey auctions or commonly called Vickrey-Clark-Grove (VCG) auctions offer a specific pricing scheme for sealed bid combinatorial auctions. GVA auctions are similar to second priced auctions in that the pricing is shown to be strategy-proof [4]. In a strategy-proof auction, there is no incentive for bidders to not bid their true valuations.

GVA winners pay their opportunity cost which is the difference between the sum of the highest bids and the sum of the highest bids without the winner's bids. For example three bidders b_1, b_2 and b_3 bid $(\$4, \$2, \$5)$, $(\$3, \$6, \$7)$ and $(\$2, \$3, \$5)$ respectively for bundles $(\{g_1\}, \{g_2\}, \{g_1, g_2\})$. The optimal allocations are b_1 wins g_1 and b_2 wins g_2. The sum of the highest bids with all bidders is $\$10$, without b_1 it is $\$8$ and without b_2 it is $\$7$. So b_1 pays $\$10 - \$8 = \$2$ for g_1 and b_1 pays $\$10 - \$7 = \$3$ for g_2.

It would not be possible to run GVA priced secure auctions using either the polynomial or homomorphic graph protocols in their current form. The primary problem in both is that determining the winner without detecting the amount they won is currently unsupported. The evaluation procedure would have to be re-run with each winning bidder removed once the winners have been determined. This unfortunately breaks the security properties of the protocol as more than just the winning (GVA) prices are revealed. The highest bids would be revealed as they are required to find the winning bidder during the optimal edge traceback step. It is possible to run secure GVA auctions using homomorphic vectors in a matrix [63] , but the matrix must contain every potential allocation of

goods to bidders. This protocol in comparison is much more expensive than the dynamic programing version and completely unscalable.

To run a GVA auction with the Garbled Circuit protocol, the circuit generator would need to be modified to include the pricing in the circuit. Although this is said to be possible, it is as yet unproven [40]. GVA pricing support can however be added to the polynomial and homomorphic protocols with less modification.

4.4.4 Bidder Preferences

In all three protocols bidders specify valuations for complementary bundles. However, as the protocols are currently defined and implemented, bidders cannot specify substitutable bundles. For instance the protocols do not allow for an auction with four goods $\{g_1, g_2, g_3, g_4\}$ where a bidder is interested in $\{g_1, g_2\}$ or $\{g_3, g_4\}$ but not both. If the bidder provides valuations for both bundles then he will win both if his bids are the highest valuations for the bundles on the optimal path.

For the first two protocols: polynomials and homomorphic, it is possible to modify the auction graph to allow substitutable preferences. Each path containing a substitutable set of bundles must be duplicated for each permutation of bidders specifying the same substitutable set. Bids for bidders who are interested in all bundles are placed on each of the respective edges on the different paths.

Sub-paths can be reused when the bidders on different edges are disjoint, but the number of additional edges required significantly increases as the number of substitutable bundles or bidders increases. Without reusing edges there will be B^D additional edges added to the graph where B is the number of bidders interested in the set of substitutable bundles and D is the number of substitutable bundles. This may add a significant cost to solving the graph although a few small sets may be manageable. An algorithm for supporting substitutable bundles while reusing edges would

then be required but that it is outside of the scope of this thesis.

For example, three bidders have the following subset of valuations in an auction with five goods:

$Bidder$	$\{g_1, g_2\}$	$\{g_3, g_4\}$	$\{g_5\}$
b_1	2	2	2
b_2	3	4	1
b_3	1	2	3

The two first two bidders, b_1 and b_2 are only interested in one of $\{g_1, g_2\}$ of $\{g_3, g_4\}$, while b_3 would like them both. A visual depiction of this is provided by Figure 4.6.

Because information required to bid is tied to the nodes in the graph, the two protocols would also need to be modified to have a preference phase before bidding is opened. Bidders would submit their preferences, the graph could be generated and information sent to the bidders. Then bidders could encrypt and submit their bids.

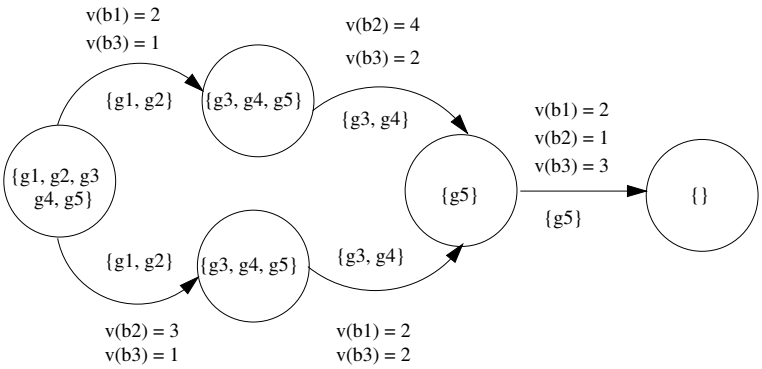

Figure 4.6: A auction sub-graph showing substitutable bids.

4.4.5 Additional Protocol Extensions

Extending the protocols to incorporate different functionality is possible to some extent but will add complexity and cost. Because the prices are

encrypted, extensions which deal with the price are unlikely to be possible. Extensions which deal with goods allocation can be possible by modifying the bid processing mechanism. This is done for example by modifying the graph for the polynomial and homomorphic protocols or modifying circuit generation algorithm for garbled circuits.

For instance, a fair share policy limits the number of goods a bidder can win to a certain quota. To implement this in an auction graph, specific bidders must be assigned to edges in a similar way to that of the last section. The number of paths must be increased for every combination of bundles allocated to bidders with the maximum number of allocations for a bidder on each path less than or equal to the quota. The number of bundles which can be bid on in the auction can be reduced as they must be smaller than the quota, but the cost of the extra paths will be high.

Chapter 5

GAF Specification

This chapter develops the specification for the auction framework I have designed and used to implement the auction protocols in this thesis. The General Auction Framework (GAF) has been developed to facilitate implementation, testing and comparison of auction protocols. GAF defines the general behaviour and interfaces for the different participants required to run auctions. The different aspects of an auction are divided to provide a structured approach to building centralised and distributed auction systems.

Section 5.1 explains the framework design methodology used in this thesis. Requirements for GAF are drawn from domain knowledge in section 5.2.2, analysis of the behaviour of a standard English auction, the secure auction protocols [38, 43, 62, 63, 68, 69], and the different combinatorial auction protocols in [18]. Building on the requirements, section 5.3 provides a domain model (an approach from [2]). The domain model is used to produce a design for each area of adaptability within the framework in section 5.4. Finally, the high level design of GAF is presented in section 5.6 which is developed into the object model provided in chapter 6.

5.1 Framework Design Methodology used in this Thesis

There are two types of framework design methodologies [36]. The first attempts to design a framework without any existing applications or models and the second type generalises a framework from existing applications. The first approach is called an *'a priori'* approach [36], and the second is called an *'a posteriori'* approach. Although framework design literature commonly states that frameworks must be built *a priori*, it is possible for a framework to be successfully built *a posteriori* [36]. The difference is that domain knowledge is inherently built into existing applications while with *a priori* it needs to be extracted from literature or domain experts.

An *a priori* approach is used in this thesis because there are no auction applications or models available to generalise. Most auction applications are industry built and closed source, especially those which provide combinatorial auctions. Domain knowledge is therefore extracted from standard auction examples such as the three given in chapter 3: combinatorial auction protocols [18]; secure combinatorial auctions [38, 43, 62, 63, 68, 69]; and a few auction protocol application prototypes built to collect statistics. Concepts from the domain have been extracted into a series of requirements which are provided in section 5.2, and form the basis of a domain analysis.

The approach used to develop a domain model (including hotspot identification and description) from the requirements and literature is knowledge graphs, a methodology by Askit et al. [1, 2]. This scheme provides the first step identified by Bosch et al. [8] of domain analysis, but how to map the modeled domain to an object model is not adequately described. Instead, the third aspect of framework design is achieved with Schmid's, more formal method [53, 54]. A hotspot specification is built from the hotspots identified in the domain model and then patterns are mapped using Schmid's heuristics. This merging of schemes is reason-

able as the requirement of building a hotspot specification is a list of well defined hotspots which are provided by both schemes. The difference between the two methodologies, in terms of discovering hotspots, is that in Askit's scheme they are identified using graphs of domain knowledge. In Schmid's they are identified from application models, which were themselves distilled from domain knowledge. Therefore without any application models, knowledge graphs is the best method of domain analysis.

The design methodology process is shown in Figure 5.1. It is broken into the three phases identified; building a domain knowledge model, choosing an architecture and designing the framework.

Phase one: Building a Knowledge Model

A domain knowledge model represents the different concerns of a framework's domain [2]. It is an intermediary step, analysed to develop a more complex model. Domain knowledge can be extracted from literature, prior experience or other applications. The GAF requirements identified in section 5.2 provide the basis for this knowledge model. The approach is semi-formal, but this suits the current problem as it is not especially complex.

Step 1. **Identify the top-level knowledge graph.** The top-level knowledge graph represents the basic concepts required in the system. It is not unlike drawing a doodle of a system upon initial investigation. The vertices on the knowledge graph represent concepts required in all application implementations using the framework. The edges represent relationships between concepts.

Step 2. **Refine the top-level knowledge graph into knowledge domains.** This involves splitting each concept (vertex) into behavioural components and identifying potential specialised implementations of each. There is no rigorous formula to do this. The behavioural components are found through an informal analysis of domain literature.

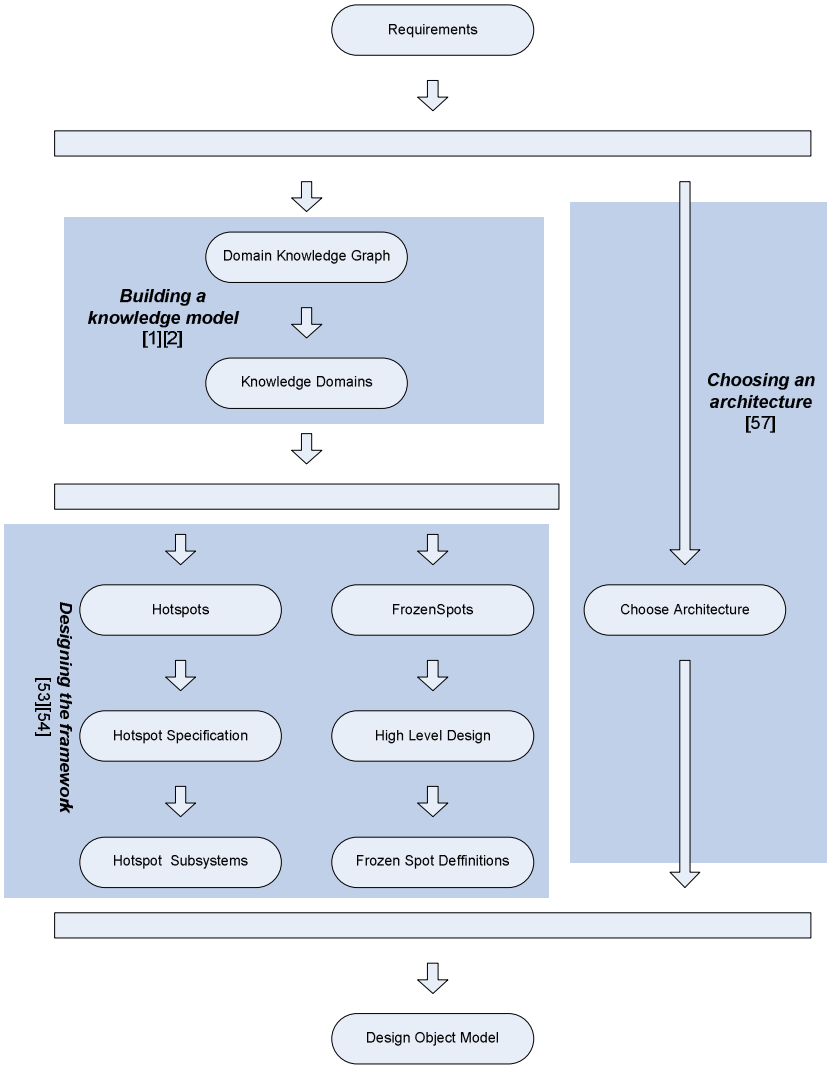

Figure 5.1: Design methodology process.

Step 3. **Define hotspots and hotspot dependencies.** In this step, each knowledge domain is investigated individually to determine which concerns can be implemented as fixed inside the framework. Concerns which may change between applications using the framework become hotspots which must be implemented or chosen when using the framework. The specialisation of concerns identified in the previous step should point towards areas of required adaptability in the framework.

Hotspots may have dependencies between them which need to be identified. Dependencies can be identified by looking for hotspots which cannot be modified independently [49].

Phase Two: Choosing an Architecture

This framework follows a distributed object model using proxies to support different types of communication (see section 5.4).

Phase Three: Designing the Framework

There are two parts to designing a framework; the hotspot integration and the fixed aspect design. The fixed aspect design is developed using standard use case analysis, supported by use cases and sequence diagrams. Hotspot integration is achieved using Schmid's hotspot specification and heuristics [53, 54] by assigning patterns to each hotspot. The hotspot specification for a hotspot contains the following attributes:

1. **Hotspot name**
2. **Short description.**
3. **Examples of hotspot specialisations.**
4. **Bind time:** the point at which specialisation is bound to a hotspot (either at implementation or run time) and whether it can be rebound.

5. **Responsibility:** the different behaviour of the hotspot required by other components in the framework.

6. **Variability:** the aspects of adaptability required. A hotspot is classified as elementary if it contains a single elementary variable aspect. An elementary aspect is one which cannot be decomposed into separate non-dependent sub-aspects. If the hotspot is non-elementary, then the reasons why it is not should be supplied.

7. **Multiplicity:** whether the hotspot can use multiple instances of the hotspot simultaneously. For example, while a graph can only have one underlying structure at a time, a window can have multiple simultaneous decorations.

   The multiplicity is either one (a single instance) or n (n simultaneous alternatives). If the hotspot multiplicity is n it is further characterised by its structure, either chain-structured or tree-structured. If the hotspot is chain-structured, each instance contains a reference to one or zero other instances. If the hotspot is tree-structured then each implementation contains a reference to a set of other instances. When the hotspot implementation is called, all instances are called recursively.

   The multiplicity also defines whether the variability can be provided by parameterisation. For example if the hotspot was storage for a collection, the storage type could be specified using parameters upon creation (e.g. the method used by the Java collection framework to manage different types through generics).

The design for integrating a hotspot within a framework is called a hotspot subsystem, and it is based on these attributes. The design can either be mapped from the variability and multiplicity to a design pattern, or one created if no existing pattern is suitable [54]. Schmid categorises the well known patterns [23] by their multiplicity which narrows the pattern

Non-recursive hot-spot subsystem design patterns:
Interface inheritance, Abstract Factory, Builder, Factory Method, Prototype, Adapter, Bridge, Proxy, Command, Iterator, Mediator, Observer, State, Strategy, Template method, Visitor
$1:1$ recursive hot-spot subsystem design patterns:
Chain of responsibility, Decorator
$1:n$ recursive hot-spot subsystem design patterns:
Composite, Interpreter

Figure 5.2: Hotspot subsystem categories with corresponding design patterns from [54].

search. The variability is then used to pick between the different patterns in the category. The pattern mapping [54] is provided for convenience in Figure 5.2.

5.2 Framework Requirements

The requirements are divided into two concerns: the requirements of different protocols and the requirements of different applications. For instance, protocols may require different communication between participants which is protocol specific, whereas different applications may use different types of communication which is application specific.

Requirements for a framework cannot be arbitrarily decided or related to one particular application. They must be logically drawn from the framework's target domain to suit varying applications. The auction requirements in this section are drawn from domain knowledge, analysing the behaviour of a standard English auction, the secure auction protocols [38, 43, 62, 63, 68, 69] and the different combinatorial auction protocols in [18]. Experience from previously developing two isolated auction protocols has also been incorporated. Section 5.2.1 provides required definitions

before section 5.2.2 sets out the application and protocol specific requirements.

5.2.1 Definitions

- Framework developer: The developer of a GAF implementation.
- Protocol developer: The developer of a protocol implementation to run within a GAF implementation.
- Application developer: The developer of an application which uses a GAF implementation to run auctions.
- Auction participant: An auction participant is anything which performs an operation in the auction, even simply by registering as an observer.
- Framework boundary: The boundary of the framework defines the participants and interfaces of the system. All communication flows and behaviour inside the boundary are defined by the framework (and implemented by the framework or protocol developer). Auction participants can operate both outside and within the GAF boundary but behaviour outside of the boundary is not defined by the framework (except for convenience) and must be designed by the application developer.
- External participant: An external participant is one which sits outside the boundary of the framework. It has an interest in the auction, but does not perform any internal protocol procedures. The majority of the behaviour of external participants is not defined by the framework and is application specific.
- Auction resource: An auction resource is a participant which performs some set of operations to run the auction, for example an auctioneer or evaluator. Behaviour of the auction resources is totally

inside the boundary defined by the framework or protocols within GAF.

5.2.2 Requirements

This section provides the requirements of the auction framework. They are divided into the protocol and application specific requirements. These are the requirements to support multiple auction protocols within an implementation of GAF and those to support using an implementation of GAF within different applications. For reference, each requirement is labeled with a number and a prefix. The prefix is either 'PR' or 'AR' indicating protocol and application specific requirements respectively.

Protocol Specific Requirements:

PR1. **Auction fundamentals:** An auction instance is one occurrence of an auction, with multiple coordinating participants. The framework should provide a systematic method of creating and running auction instances. An auction consists of bidders who bid on goods offered by an auction owner which are collected by an auctioneer. Evaluators solve the auction (finding the optimal allocation of goods) and publish the results.

PR2. **Auction protocols:** The framework should support multiple auction protocols where participants may use different protocols concurrently or consecutively. Each protocol will define the different resources it requires to run.

PR3. **Auction resources:** The framework should support multiple auction resources used to run an auction. Resources may be used for multiple auction instances at a time (auction pooling). There are set resources required for every auction to perform standard tasks. Protocols however may need to define custom resources, for example a

bulletin board to store working data. Some protocol specific setup will be required for most, if not all, resources before the resource can participate.

Some protocols require resource behaviour which would normally be included in a single resource to be implemented by multiple resources. For example, a protocol which requires plural auctioneers to prevent a single rogue auctioneer manipulating the auction. Some protocols may require behaviour which would normally be divided between multiple resources to be included in a single resource. For instance, an English auctioneer usually performs all required activities: receiving bids, determining the winner and announcing the progress at all stages of the auction. To enable this, resources must be able to represent themselves as multiple types dependent on the protocol (note that simple casting is not possible in a distributed environment).

PR4. **Bidder behaviour:** The majority of bidder behaviour is defined by the application using GAF. However, bid generation is protocol dependent and should be implemented by the protocol developer. Note that in this context bid generation does not include valuation of goods. Goods valuation is at least application dependent and may differ between bidders in the same application.

PR5. **Auction owner behaviour:** What the auction owner does before an auction, (for example deciding what goods to offer) is application specific and left to the application developer. The only behaviour defined by the framework is the call to create the auction.

There is protocol specific behaviour which the auction owner will need to perform. For instance, creating a representation of the bids to match the protocol.

PR6. **Participant communication:** Communication order and message con-

tent of non-standard auction behaviour is protocol dependent and should be left to the protocol developer. The ability to send custom messages needs to be facilitated by the framework.

PR7. **Protocol adaption:** For dynamic adaption protocols, participant behaviour should be modifiable using protocol specific auction settings.

PR8. **Auction input and output:** The inputs to an auction instance are the bundles under auction and the settings for the auction instance. A bundle is a subset of goods. Both the bundles and settings which can be used in an auction are protocol specific. For example, in an English auction only one bundle containing a single good can be auctioned. In a combinatorial auction the auction may sell a subset of the different possible combination of the goods, each of which is a bundle.

The result of the auction maps bundles to bidders with prices. There may be multiple results if the protocol cannot decide on the outcome. For instance, a protocol may defer tie-break decisions to the application. The representation of an auction result can be generalised for different protocols but custom data should be able to be provided with a mapping. For example, a sealed bid auction may defer tie-breaks to the application and provide the application with the time each bid was placed.

PR9. **Separation of bidding language and goods representation:** The framework should not enforce bidding language or representation of goods. Both may change with different protocols and settings.

PR10. **Protocol dependencies:** Each instance of an auction will require participants and their behaviour to be compatible. For example, the algorithm used to evaluate an auction should be compatible with bidders' bids. Dependencies should be enforced by the framework.

PR11. **Verification:** Some auction protocols require or allow the auction outcome to be verified. The framework must allow for outcome verification by all participants if the protocol supports it. Verification may require access to any stage of the auction data; the auction settings, the initial bids, any intermediate results and the end result.

PR12. **Observation:** Multiple participants will be interested in observing the auction. What an observer can observe and who is permitted to observe will be dependent on the auction protocol and policies.

Application Specific Requirements:

AR1. **Communication:** The method of communication between participants should not be defined by the framework but left to the application developer.

AR2. **Authentication and authorisation:** Participants must be able to authenticate requests as certain requests will require authorisation. For example a request for auction cancellation must be made by the auction owner.

AR3. **Minimal external interaction:** There should be minimal entry points into the frameworks, reducing the amount of work required for integration with external systems.

5.3 Domain Knowledge Model

GAF maps well to a top-level knowledge graph, as auction protocols for the most part are a set of similar interactions between the same participant types. The top-level knowledge graph is provided in Figure 5.3. Table 5.1 lists all identified concerns with the corresponding requirements referenced.

All vertices except outcome, protocol and policy are participants which inherit from a standard auction participant or auction resource (which in turn inherits from auction participant).

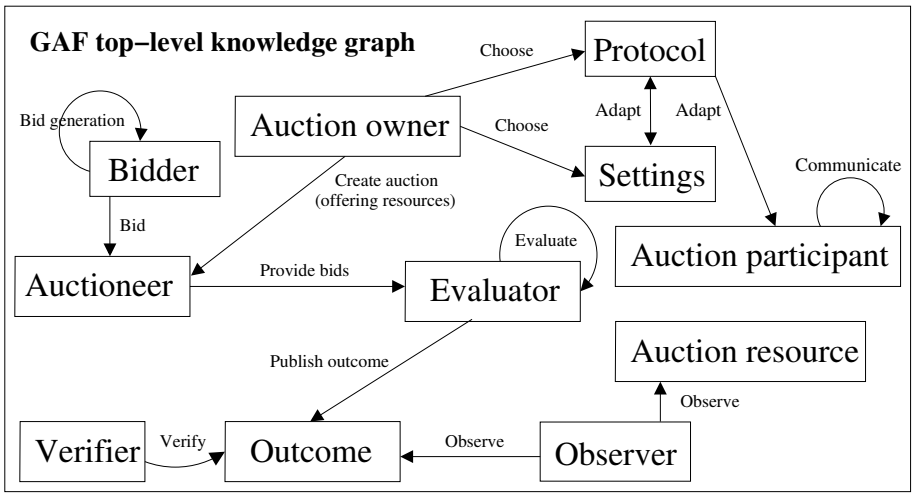

Figure 5.3: **The top-level knowledge graph for GAF.** Although the directed edges appear to imply a chronological ordering, this cannot be assumed. They indicate which concept initiates an interaction but not when.

Each concern (knowledge vertices) in the top-level knowledge graph has been divided into the knowledge domains provided in Figure 5.4. To be more concise, the diagram does not include any protocol specialisation examples. Instead, the specialisation given is "Protocol specific". There are many examples including English, Dutch, Secure Homomorphic and Secure Polynomial (see chapters 3 and 4). Generally the different domains are easily mapped to the requirements and are not discussed. The few that are less obvious are set out below.

- **Auction participant:** Auction identification is introduced from requirement PR1 to allow participation in multiple auction instances. When the participant is sent a message from another participant, the message will need to contain a reference to the auction instance.

Knowledge Vertex	Requirements
Bidder	PR1
Auctioneer	PR1
Verifier	PR11
Auction owner	PR1,PR5
Outcome	PR1
Evaluator	PR1
Protocol	PR2, PR3, PR7
Settings	PR7, PR9
Auction resource	PR3
Observer	PR12
Auction participant	PR1

Table 5.1: Top-level knowledge requirements reference.

Each participant needs to be able to uniquely identify themselves within an auction instance. Identification should be attached to requests and is required for authentication and/or authorisation. Participants need to be sure requests are authentic, and that a participant making a request or providing data is authorised for the related action.

A participant who has a reference to another participant may require a different interface than the one currently held. This would, for example, be required in the case of an auction resource performing the duties of two different resource types. Casting will not always be possible as the participant may be holding a remote reference to the other participant. The participant who needs the reference (interface) will need to request the other to provide a new reference. This is called participant transformation. Before transformation can take place, the types the participant supports will need to be checked for the type the requester requires. Transformation is included in par-

ticipant instead of auction resource to allow for more flexibility as this will be required. For instance, an auction owner who is also an observer.

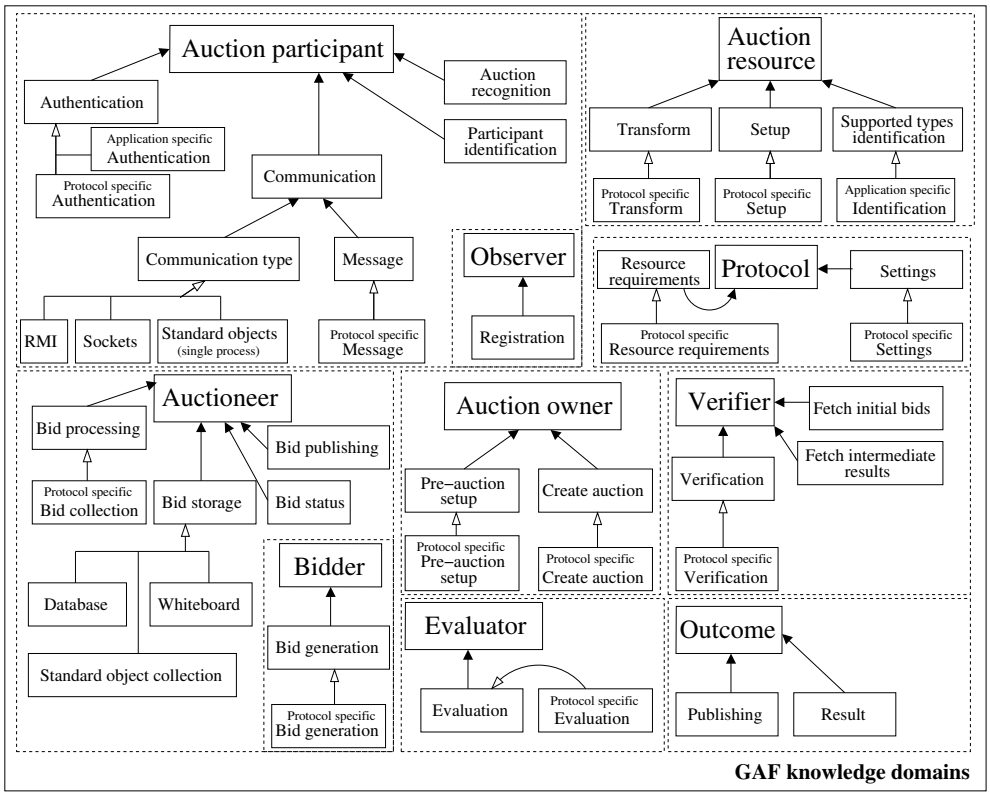

Figure 5.4: Knowledge domains for GAF. Unlike [2], the diagram includes the concerns, not just the domains in each concern. This provides a better overview of all domain knowledge. Because the concerns are included, a distinction is required between domain aggregation and the specialisation of sub-domains. A directed line is used for aggregation, and specialisation is shown by standard UML inheritance notation.

Two patterns in the diagram are important and are noted in the next step. First, each domain with a specialisation is a good candidate for adaptability. Second, because the context is a single auction instance and

Domain parent	Domain	Dependence
Bidder	Bid generation	Protocol
Auctioneer	Bid processing	Protocol
Auctioneer	Bid storage	Application
Auctioneer	Bid status	Protocol
Verifier	Verification	Protocol
Verifier	Fetch data for verification	Protocol
Auction owner	Pre-auction setup	Protocol
Auction owner	Create auction	Protocol
Evaluator	Evaluation	Protocol
Protocol	Resource requirements	Protocol
Protocol	Settings	Protocol
Auction participant	Authentication	Protocol & Application
Auction participant	Communication type	Application
Auction participant	Message	Protocol
Auction resource	Transform	Protocol
Auction resource	Setup	Protocol
Auction resource	Supported types identification	Protocol
Auction resource	Supported types identification	Protocol

Table 5.2: Hotspots identified from knowledge domains.

only one protocol may be used per instance, there are dependencies between each domain with a "Protocol specific" child.

All of the vertices from Figure 5.4 with specialisation are areas of adaptability (either by the protocol or application developer) which are the hotspots for the framework. Hotspot dependencies exist between every vertex with a protocol specific child, as each will have to be compatible within an auction instance. Table 5.2 contains the list of hotspots, identifying the dependences between them.

5.4 Hotspots

Hotspots are the variable aspects of a framework [49, 54], similar to the blanks in a 'fill in the blanks' worksheet. Hotspots are the most important concern in framework development as they support varying applications. Hotspots also separate the tasks of the framework developer and the application developer. The framework developer builds the structure around the hotspots while the application developer fills them in. Once the hotspots in a framework have been filled with existing and/or new implementations, the system should be usable.

In GAF, the two different views represent two different developers. The black-box view is used by the application developer who only requires knowledge of the required input and external interfaces. The white-box view is used by the protocol developer who will need to have an understanding of internal framework aspects.

The method of including a hotspot in the framework design is called its hotspot subsystem [54]. To design the subsystems, each hotspot must be identified, defined and analysed. The hotspot analysis matches the hotspots against design patterns to determine how they can be implemented. The different hotspots in GAF were identified in table 5.2 from section 5.3. Three hotspots are not included to reduce complexity: Protocol policy, Application policy and Authentication. As discussed in section 5.2.2, policies are not included in the design. Because authentication is based solely on policies (both protocol and application), it is also not included.

The next two sections will first define the hotspots and then the hotspot subsystems.

5.4.1 Hotspot Specification

The description of a hotspot is called its hotspot specification [53, 54]. The specification is given by the attributes defined in section 5.1; name, de-

scription, examples, bind time, responsibility, variability and multiplicity.

5.4.2 Hotspot Subsystems

As all GAF hotspots have a multiplicity of one, the hotspots are limited to the first group of patterns. For each of the protocol algorithm hotspots (setup, bid processing, evaluation, create auction, bid generation, verification) the only patterns which are suitable for the variability are interface inheritance and the factory method. All of these, except bid generation require state to be kept by the resources during operations. The only way to use state with factory methods is to pass it to each method when called. Therefore, interface inheritance is chosen for all of the protocol algorithm hotspot subsystems, except for bid generation, which can use the factory method pattern, overloaded in the protocol class.

The bid storage hotspot is quite simple and needs no additional functionality other than what interface inheritance provides.

5.4.3 Hotspot Dependencies

As stated in section 2.2.4, a hotspot dependency occurs when a hotspot cannot be replaced independently from another [49]. In GAF this is the case with all protocol specific hotspots as only one protocol can be used within a single auction instance and the hotspot implementations must be compatible.

5.5 System Architecture

A software architecture describes an underlying design; decomposing the system into components with patterns of interactions between them [57]. The architecture is shown in Figure 5.5. As described in section 5.2.1, there are three developers; the application developer chooses between protocols

Name	Bind time	Parameterisation
Bidder bid generation	Run time	No
Auctioneer bid processing	Run time	No
Auctioneer bid storage	Run time	No
Auctioneer bid status	Run time	No
Verifier verification	Run time	No
Verifier, fetch verification data	Implementation	Yes
Auction owner pre-auction setup	Run time	No
Auction owner create auction	Run time	No
Evaluator evaluation	Run time	No
Protocol resource requirements	Run time	No
Protocol settings	Run time	No
Auction participant authentication	Implementation	Yes
Auction participant communication	Implementation	No
Auction participant message	Implementation	Yes
Auction resource transform	Implementation	Yes
Auction resource setup	Run time	No
Auction resource supported	Run time	No
Auction resource types identification	Run time	Yes

Table 5.3: Hotspot specification summary. The variability and multiplicity of the hotspots are not shown because they are all the same. The variability is elementary and multiplicity is one.

and a few other hotspots, the protocol developer develops implementations of the protocol specific hotspots and the framework developer builds the fixed aspects of the framework and maybe some default implementations of non-protocol specific hotspots.

It is useful to think of the system in terms of three subsystems, each corresponding to one of the different developers. The application is built by the application developer whose operations interact with the infrastructure provided by the framework subsystem. The protocol developer builds a protocol in the protocol subsystem, implementing the different protocol specific hotspots. The protocol subsystem interacts for the most part with services provided by the framework subsystem except for communication with participants if required, which is passed by the application. An application uses GAF to leverage the different protocols implemented within it.

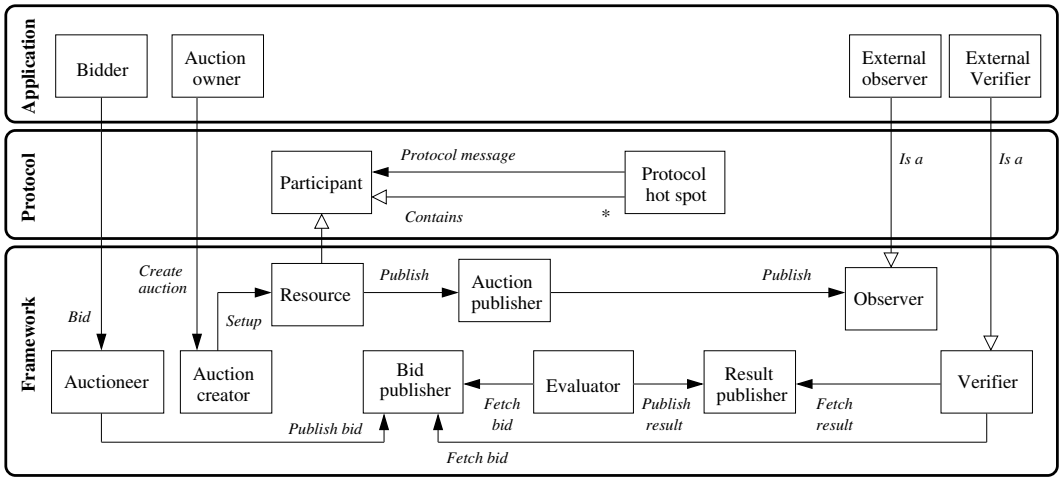

Figure 5.5: GAF system architecture. For brevity the diagram does not show auctioneer, auction creator, bid publisher, auction publisher, evaluator and result publisher as resources, nor observer and verifier as participants.

5.6 High Level Design

A high level design builds a model from the requirements specification, defining the different participants in the system, their behaviour and communication. The high level model is analysed to build a lower level model such as an object model. The design in this section uses the functionality identified in section 5.3, and includes the hotspot subsystems from section 5.4 as well as the framework's frozen spots. Frozen spots are the areas of functionality which are fixed between applications [54].

This section starts by discussing two important concerns; auction phases in section 5.6.1 and communication in section 5.6.2. Participant behaviour is expressed with use cases and sequence diagrams in section 5.6.3 and the patterns used within GAF are presented in chapter 6.

5.6.1 Auction Phases and Events

Auction phases and events are an important part of auctions as they are time based and/or event driven. For instance, the bidding phase in a sealed bid auction usually ends after a fixed period. On the other hand, bidding in English auctions often lasts as long as bidders keep bidding. As events cannot be predetermined and phases may not be able to be predetermined (depending on the protocol), a method for participants to be informed of phases and events is required.

To ensure that messages are consistent and minimal, publishing of events is centralised using the observer pattern [23]. Observers register with an auction publisher and are notified when a resource publishes to it. As different protocols may require events to be sent to different participants, resources will filter by including the participant types to publish in the publish request. One concern with this is that authentication and authorisation is required to stop an observer from falsely registering its part in the auction.

5.6.2 Participant Communication

As requirement AR1 in section 5.2 states, communication must not be defined by the framework. This is difficult as participants within the framework interact with each other and the protocol developer does not know how communication will be implemented. To provide this requirement, the proxy pattern is used [23].

In the proxy pattern, a 'proxy object' stores a reference to its real object and they both share the same interfaces. Behaviour is managed by the real object, but communication is forwarded through the proxy. Proxies are used in GAF to provide a transparent method of implementing different types of communication thereby satisfying requirement AR1.

Use of remote proxies enable protocol developers to specify communication between participants without the concern of how communication is actually implemented.

Resources need to be aware of their proxy, as at some point they will need to pass a reference to themselves to another participant.The resource will need to pass a reference to its proxy for communication to work.

5.6.3 Framework Participants

This section describes each of the possible participants in an auction protocol. Although most of the participants were identified in section 5.2, a full description of each is now provided. Participants identified from the domain knowledge are given first, followed by three which are introduced to separate different concerns. The use cases for the external participants are listed in table 5.4 which are fully described in the appendix. The high level interaction between participants is shown in Figure 5.6.

Participants Identified from Domain Knowledge

- Auction participant: Each participant involved in the framework has four base requirements, they must be able to: be referenced by an

identifier unique to the auction; inform other participants what participant types it supports; and transform into any of them.

- Auction owner: The auction owner initiates an auction instance, interacting with an auction creator. It is the auction owner's responsibility to provide (having selected and negotiated with) all resources except possibly verifiers and observers. The auction owner is provided by the application using the framework and has no unique functionality defined other than a call to create the auction. An auction owner is automatically an auction observer.

- Bidder: Each bidder values a set of goods offered by the auction application and sends bids to the auctioneer if so desired. How a bidder values goods is not defined by the framework. The only behaviour which is, is bid formulation. A bidder is automatically an auction observer.

- Auction resource: Each internal component in the framework is an auction resource. Auction resource behaviour is completely defined by the framework and protocol. There are four types of auction resources from the domain knowledge: auction creator; auctioneer; evaluator and verifier.

  It is important to separate the different auction protocol concerns into different resources as functionality may be divided in distributed applications and with different protocols.

- Auction creator: It is the responsibility of the auction creator to set up the auction as per the specification provided by the auction owner. Interactions between the auction creator and the resources are protocol dependent.

- Auctioneer: The auctioneer facilitates the auction, collects bids, and manages the auction phases.

- Evaluator: The evaluator determines the optimal allocation. Results and possibly partial results (if required for verification) are provided to the result publisher (a resource which is introduced) for retrieval by the other participants. The algorithm used for solving the auction is protocol dependent.

- Auction verifier: An auction verifier may be some subset of the other components participating in an auction, or it may be an external auditor. It may not be possible or required to verify the different aspects of the auction, so the auction verifier is not always required.

  Result verification requires checking that the allocation of goods and pricing matches the partial and final results of the auction, according to the protocol used.

- Auction observer: An auction observer is any component or external object which needs to observe the auction outcome. By subscribing, the observer will be updated of outcome changes. All bidders and the auction owner are mandatory observers.

Introduced Participants

- Bid publisher: The bid publisher provides storage and retrieval capacity for bids. This resource has been introduced to separate the concern of bid collection from bid storage.

- Result publisher: A result publisher stores the final result of the auction. Introducing this resource separates the concern of result storage from bid collection and bid storage. This also allows results to be stored after auctions have been closed without storing other information. As it is possible that the result publisher could store results for an extended period, storage of working results should be separated from result storage.

Participant	Use case
Auction owner	Create auction Cancel auction Adapt auction Open bidding Close bidding
Bidder	Create bid Bid Cancel bid Get bid status
Auction verifier	Retrieve result Retrieve bids
Auction observer	Register interest Unregister

Table 5.4: GAF Use Cases.

- Auction publisher: An auction publisher was introduced in section 5.6.1. Auction publishing follows the observer pattern. Events include the different phases and the result, but also any other protocol specific events such as a bid placed (a bid event) in an English auction.

5.6.4 Frozen Spots

The number of frozen spots are limited as the majority of functionality is protocol specific, so much reuse is achieved using parameterised hotspots. The fixed aspects in the framework are:

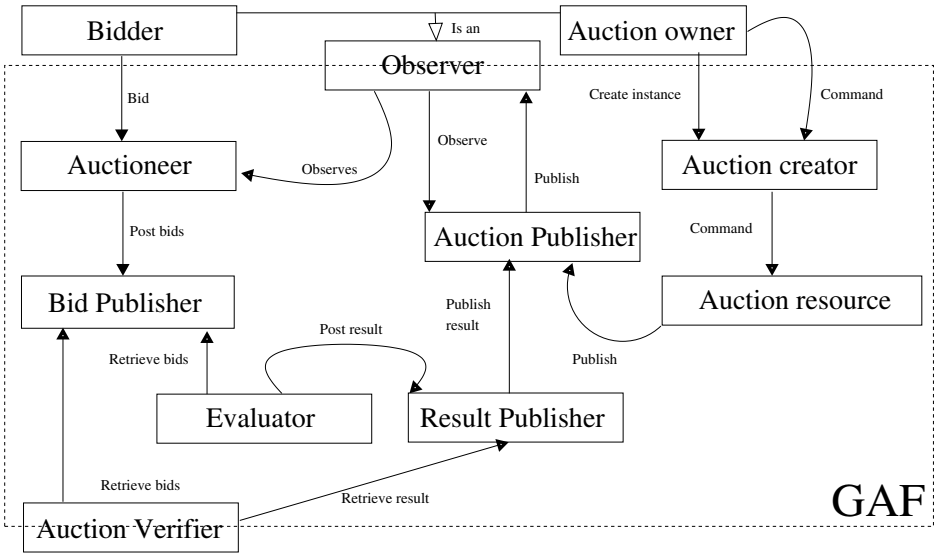

Figure 5.6: High-level GAF design.

Auction Recognition

As participants may be involved in multiple auctions simultaneously, messages must include an auction reference. A participant receiving a message must first check whether it is participating in the auction. If it is not, then the message is discarded, otherwise the requested operation is performed in the context of the auction instance provided by the reference.

Participant Identification

Participants must be able to identify each other to check whether requests are authorised. For instance, a malicious bidder should not be able to cancel an auction. Messages must include a participant reference to be checked against known participants. Participant references for resources should be passed in the auction reference. Other participant references should be passed as required, for example when a bidder bids, its reference can be passed to the auctioneer.

Auction Publishing

Auction resources sends events to be published to the auction publisher (see section 5.6.1). The auction publisher needs to maintain a list of registered observers and allow for registering and unregistering. When the auction publisher recieves an event, it either sends it to all observers or a list specified by a filter list if present.

Chapter 6

Implementation

This chapter details the design and implementation of the development performed for this thesis. As outlined in chapter 5 there are three distinct developers involved with GAF. The framework developer designs and implements the framework determining how to best provide shared functionality and bind hotspot implementations. The protocol developer implements protocol specific hotspots to provide reusable protocols. The application developer leverages GAF in their application to take advantage of different protocols already implemented and binds implementation to any application specific hotspots required. The sections in this chapter correspond to each of the respective views.

Section 6.1 provides a sample of the low level design components and how they have been implemented. The method of including each of the three combinatorial auction protocols within GAF is specified in section 6.2. Section 6.3 discusses an auction application called AuctionComposer which has been built to read and run auction test scenarios defined from XML and output performance statistics with activity logs.

6.1 GAF Implementation Design

The GAF specification provides the framework requirements, it develops definitions of the different components, involved including hotspots and frozen spots. This section presents the high level object-oriented design of GAF and describes several patterns used to implement GAF.

6.1.1 Object Model

The object model is divided into two parts, internal and external behaviour. Internal behaviour occurs within an auction instance as specified by the protocol developer. External behavior is application specific though it may take advantage of protocol developer provided hotspot implementations. For example, generating a bid is external behaviour as bidders determine when and what to bid which is dependent on the application. However, bidders use a bid generation object provided by the protocol developer as bid construction is protocol specific.

This high level object model is presented with three UML diagrams. Figure 6.1 provides the high level system behaviour in a general case using a sequence diagram, Figure 6.2 displays the classes involved in external interactions, and Figure 6.3 presents the classes used in the internal auction process.

High Level Sequence Diagram

Protocol Resource Setup. The high level sequence diagram in Figure 6.1 depicts the life-cycle of an auction-instance providing context to communication between auction participants. It ignores external participant behaviour, for example auction owner setup and instead starts with the call to a 'create auction' method by the auction owner. This assumes that the auction owner has found resources to run the auction, wrapped them in the required proxy, and performed any protocol specific setup. The se-

quence diagram ends with the optimal allocation being published and the auction resources deallocated.

Three important protocol specific hotspots are shown: protocol resource setup; bid processing, and, winner determination. As these hotspots are implemented at a lower level than the diagram depicts, the participant behaviour can not be defined further at this point. In addition, other participant behaviour is protocol specific but the communication pattern will most commonly be equivalent to that in the diagram.

The auction begins with a call by the auction owner to the auction creator to create the auction. Auction creation involves instantiating auction resources and any other protocol specific setup such as generating encryption keys. It is assumed that the auction owner has determined the auction settings and acquired the use of auction resources previously, otherwise the resources may reject the auction creator's request. The auction owner passes back an auction reference to the auction owner including a reference to the auction which the auction owner uses to start the auction (open bidding) when desired. The auctioneer informs observing participants through the auction publisher.

Upon completion of the auction, an evaluator publishes the outcome via the auction publisher who informs each of the participants. When the creator is informed of the outcome, it deallocates the resources.

Bid Processing. During the bidding phase, bidders submit their bids to the auctioneer who processes and stores them as required by the protocol. For example, protocol specific bid processing in an English auction is that the auctioneer must use the auction publisher to inform bidders of the current highest bid. Most commonly the auctioneer may also be the bid publisher and retain bids, but in some cases, for example if bids need to be kept for historical evidence, an additional bid publisher or service may be required.

Winner Determination. The auction then closes and the auctioneer informs observers via the auction publisher. This may be due to a protocol rule such as a timeout or an auction owner command. The auctioneer also requests the evaluators to find the auction solution. Winner determination is driven by a set of auction evaluators but is protocol specific and may require any of the auction participants to take part. The optional solution is provided to the result publisher who may store it and pass it to the action publisher or the auctioneer may directly pass the result to the auction publisher to inform the observers. Once the auction is closed, the auction creator requests deallocation of auction resources.

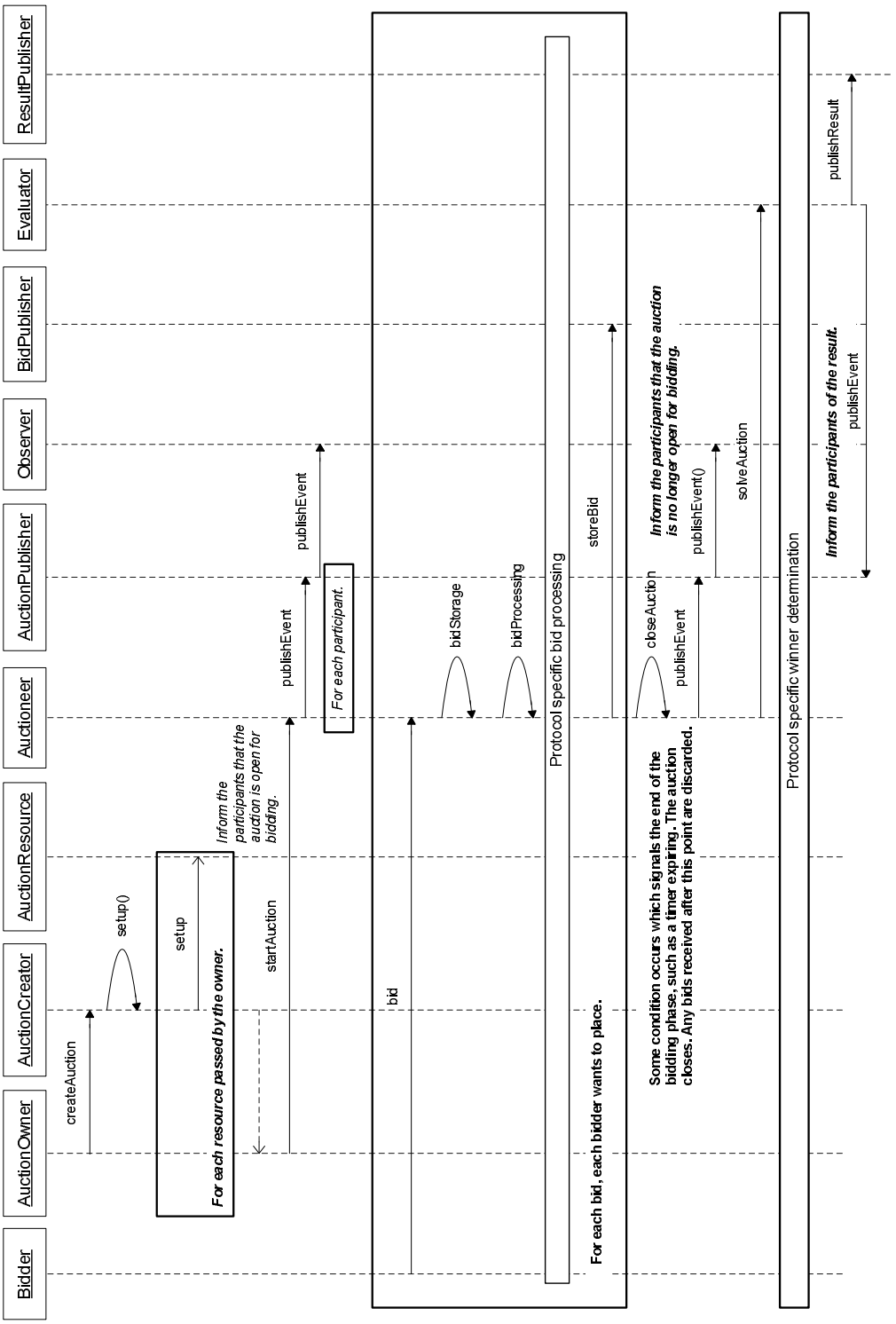

Figure 6.1: System sequence diagram in the general case. Boxes indicate repetitive behaviour, for example, the auction creator requests each auction resource participating performs setup at the start of the auction.

External Class Diagram

There are three areas of interest in the external class diagram (Figure 6.2): auction creation, resource instantiation, and bidding. Auction creation is performed by the auction owner and involves choosing a protocol, setting up the auction and acquiring resources to run it. Resource instantiation is used to create resources, based on the protocol for the auction. This may be performed by the auction owner or by another party, for instance a resource provider.

During auction creation, the auction owner chooses a set of bundles under auction and the protocol for the auction. The protocol is used to retrieve resource requirements and potential auction settings, with some application specific process to determine how the settings should be chosen. Independent of who supplies the resources, they have to be instantiated, which is performed by passing requirements to the resource factory. The resource factory uses reflection to dynamically instantiate the resources based on the classes specified in the resource requirements. Once the owner has the resources for the auction it can create auction instructions to supply to the auction creator. The instructions contain the protocol, settings, bundles of goods and each of the resources wrapped in a proxy (using the proxy factory). An auction reference is returned to the owner once the auction has been created, containing a reference string and the resources performing the auction. The reference may also be subtyped by a protocol for protocol specific data, such as encryption keys.

The application must then provide the auction reference to bidders either directly or through another service such as an auction advertising service. Bidders retrieve a bid generator from the protocol which is contained within the auction reference. Any protocol specific options are created and with the bidder's valuation and mapping to the goods bundle the bidder is interested in, is formed the bid settings. The bid settings are used with the reference and bid generator to create a bids.

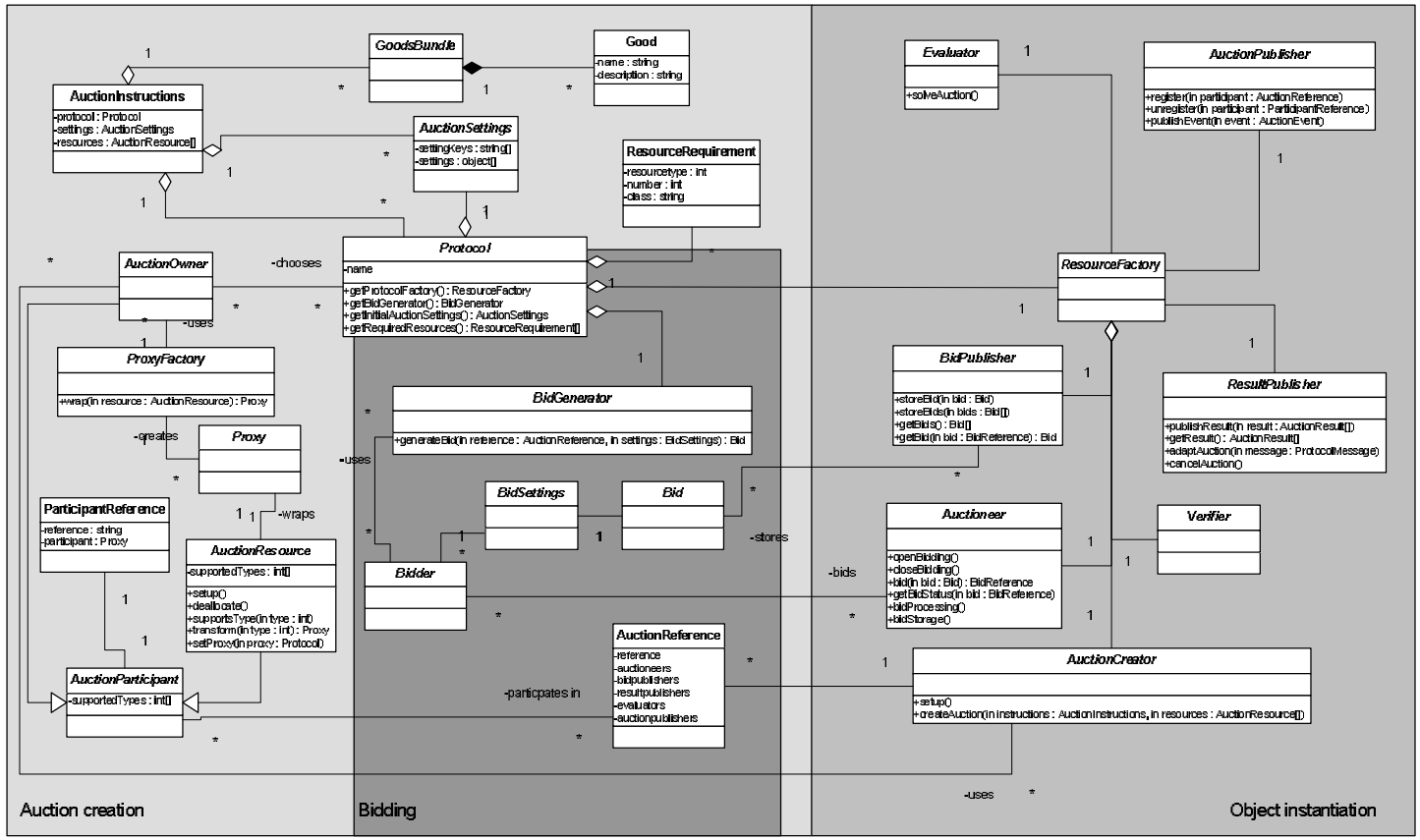

Figure 6.2: External class diagram. For brevity the inheritance relationship between the different resource types and AuctionResource is not shown. Each of the following: AuctionCreator, Auctioneer, Evaluator, AuctionPublisher, BidPublisher, ResultPublisher and Verifier are subclasses of AuctionResource. To be concise, auction and participant references are not shown but they must accompany each message. Setter and getter methods are also not shown.

Internal Auction Class Diagram

Figure 6.3 displays the relationship between resources during an auction instance. The auction publisher publishes auction events to each of the observers registered. Auction resources can also send protocol specific messages to each other, allowing communication for hotspots. When a bidder places a bid, the auctioneer provides the bidder with a bid reference. The bidder can use the reference to request the current status of a bid. What the current status may be is protocol specific, but it may need to retrieve the bid from the bid publisher. The auctioneer creates a bid status object for the bid and returns it to the bidder.

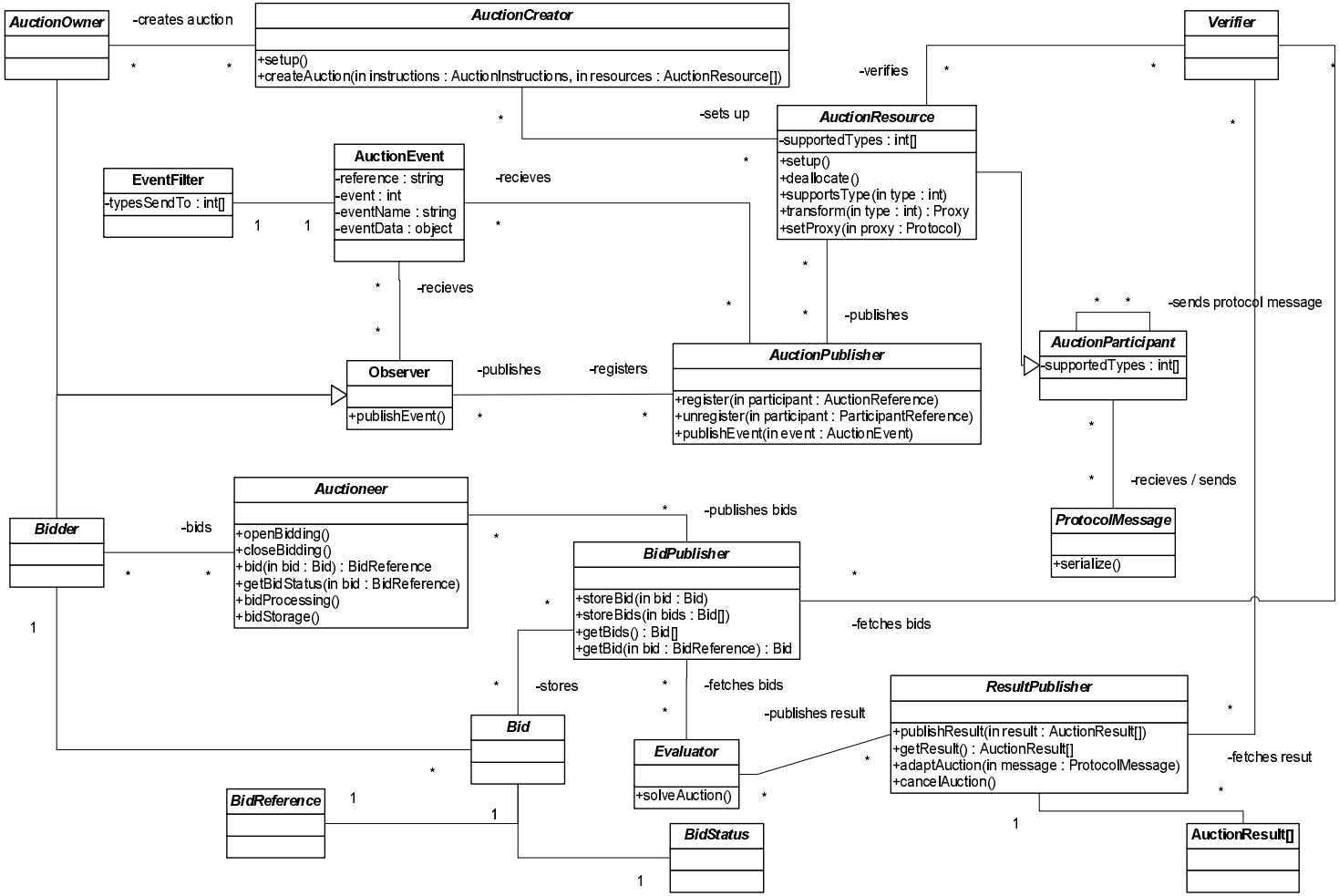

Figure 6.3: Auction class diagram. To save space, setter and getter methods are not shown. An auction reference also contains the protocol and settings for the auction, but the link between these classes is not shown.

6.1.2 Design Patterns

Design patterns are standard solutions to well defined recurring design issues [23]. Several GAF components use design patterns in order to achieve their requirements. The following design patterns are defined in the pattern catalog [23] and are briefly described here with detail about their use within GAF. The first five facilitate hotspot development, providing transparency to either the application or protocol developer. The final pattern facilitates consistent communication within auctions.

- Abstract factory: An abstract factory is a single object which can create families of related objects. It is used to produce a consistent set of concrete classes throughout an auction instance, and the application need only deal with abstract classes.

  An abstract factory is used within GAF to create concrete auction resources in accordance with the protocol in use. An object wishing to instantiate an auction resource, such as a resource provider, retrieves a resource factory and generates the requested resource by parameterising the factory with an auction resource requirement and protocol. The factory uses reflection to instantiate a concrete resource of the requested protocol and type.

- Proxy pattern: As discussed in section 5.6.2, participant communication is abstracted from the framework using the proxy pattern. Each resource is wrapped within a remote proxy which channels participant communication using the type required by the application. A proxy abstract factory specified by the auction owner is used to wrap resources. The auction owner or resource provider passes the resources through the factory, which determines the type of resource and returns a corresponding proxy of the communication proxies type.

- Command: The command pattern is a design for the issuing of ab-

stract commands. The auction owner may need to issue protocol or policy dependant commands to the auction creator. Abstract commands are represented by an object which can be passed between components. The component executes the request without knowing what the concrete class or actual request is.

- Template method: A template method defines a series of operations that are required in an algorithm, but leaves the detail to subclasses. The detail left for subclasses are the hotspots in the framework.

- Observer: An observer can subscribe to a publisher so that when publisher state changes, the observer is updated. For auction components to be aware of the events in the auction they subscribe to the auction publisher, becoming an auction observer.

6.2 Auction Protocol Implementation

Two methods of implementation have been used to include the three different protocols within GAF: the Homomorphic protocol has been implemented completely within GAF resources using a few base libraries. The Polynomial and Garbled circuit protocols have been previously implemented and these implementations have been wrapped within simple GAF resources.

6.2.1 Case Study One: Polynomial Auction Protocol

The Polynomial protocol was first implemented as a stand-alone single threaded application. In order to run polynomial protocols within GAF, each of the different components has to be separated and wrapped within auction resources. Six different auction resources were used: an auction creator, auctioneer, auction publisher, bulletin-board, mask publisher, and a result publisher. The bulletin board and mask publisher are protocol

specific resources which extend from the base type AuctionResource and communicate using proxies.

The auction creator behaves exactly as specified in the GAF specification, retrieving goods and the auction settings from the auction instructions which are: the threshold parameter (the minimum number of evaluators required to decrypt), the maximum polynomial coefficient and the target constant (for decryption resolution). The auction creator generates an auction graph using the provided goods, the maximum bid as described in section 4.4.1, and a resolve value for each of the auctioneers. The settings and generated objects are used to create an auction reference. Using that auction reference, the auction creator instantiates the other resources which have been provided by the auction owner and returns the auction reference to the auction owner.

Bidders in the Polynomial protocol send their evaluators who are the auctioneers. Several polynomial auctioneers are used in each auction instance, each implementing both the auctioneer and evaluator interfaces. When an auctioneer is requested to set up, it instantiates a polynomial evaluator which is an object from the lower level library and a bulletin board translator. The translator is a proxy object which wraps the methods of the polynomial bulletin board and is provided to the evaluator as the original library was not intended for use in a distributed environment. It is able to do this by implementing the base libraries bulletin board interface. The translator stores a reference to the GAF bulletin board auction resource (which is actually the remote proxy) and translates between the two high and low level implementations. Figure 6.4 displays the relationship between GAF and the low level classes and Figure 6.5 shows a wrapped message that the bulletin board translator provides to the low level polynomial evaluator.

There is a single bulletin board which implements only the auction resource interface in order to receive protocol messages. It works much the same as the auctioneer, with auctioneer translators used to communicate

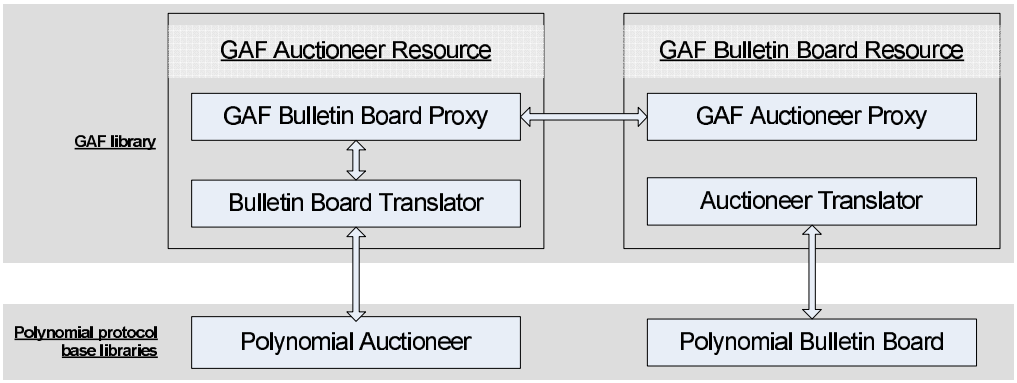

Figure 6.4: Relationship between GAF resources, GAF proxies, GAF translators and base polynomial libraries auctioneer and bulletin classes.

```
public void setOptimalBidder( int step, PolynomialBidder
    bidder ) throws ... {
        if ( this.isDeallocated ) return;
        this.bb.sendProtocolMessage( reference, new
            PolyBBSetOptimalBidderMessage( step, bidder ) )
            ;
}
```

Figure 6.5: A code snippet from the bulletin board translator showing the method which marks a bidder as optimal for a particular step in the protocol. If the resource has been deallocated then the method will return without performing any action, otherwise it will send a protocol message to the bulletin board. Note that each protocol message has its own type in order to store different objects. For instance, the set optimal bidder message contains the optimal bidder and the step the bidder is being set for.

with each of the auctioneers. The auctioneer translators are instantiated, wrapping each auctioneer when the bulletin board is setup. There are several mask publishers, depending on security requirements, which also simply implement the auction protocol interface and send protocol messages as required. The results are sent to the result publisher which then forwards it on to the auction publisher.

6.2.2 Case Study Two: Homomorphic Auction Protocol

All of the behaviour for the Homomorphic auction protocol is specified within the GAF implementation of the protocol as it had not previously been fully implemented. A single threaded application without threshold encryption has been [14], but the implementation would have been too difficult to extend. The Homomorphic protocol is implemented using five auction resources, where all are one of the standard types: auctioneer, evaluator, bid publisher, result publisher and auction publisher.

There is a single auctioneer, which is also the auction creator. The auctioneer when requested, instantiates the auction instance using the goods, the different resources, the key size and the vector length. One of the auction creator's tasks is to assign evaluators to evaluator groups with matching threshold keys. A single public key is provided for each of the evaluator groups, with private keys for each of the evaluators. In a production environment this would require the auctioneer to be a trusted party or the protocol could be modified to perform group key generation.

When requested to find the auction solution, homomorphic evaluators each start an evaluation thread which solves the auction by running through the different steps as specified in section 4.2. The bid publisher and result publisher are used to store the bids by the auctioneer and the results from the evaluators respectively. These are not specifically required by the protocol and could be replaced by simple behaviour within the auctioneer if the protocol was not modified to take advantage of the separa-

tion of concerns. The auction publisher publishes auction events to registered observers.

6.2.3 Case Study Three: Garbled Circuit Auction Protocol

The Garbled Circuit protocol wraps the behaviour of a lower level library in a similar manner that the polynomial protocol does. The auctioneer wraps the garbled auction manager, and the evaluator wraps the garbled auction issuer. Communication in the Garbled Circuit protocol is minimal and so translators are not used, with the communication translated in the resource objects themselves. The bid publisher, result publisher and auction publisher behave exactly the same as in the other two protocols.

6.3 AuctionComposer Design and Implementation

AuctionComposer is intended to be a simple distributed system to test and compare auction protocols. It provides the ability to gather and export auction performance statistics and logged event details. This system's implementation verifies that the framework can be used with an application and shows that the development required to use a variety of pre-implemented protocols is minimal. Auction resources are provided by remote resource providers and auctions are either simulated from existing auction descriptions or by randomly generating auction scenarios. Future extensions will enable the auction composer to generate and run random auctions as well as export auction results which can then be compared either against other protocols or expected solutions.

6.3.1 AuctionComposer Applications

AuctionComposer consists of three applications: an auction resource provider, an auction test manager and the composer application. Each application is able to run on a different host as communication runs through Remote Method Invocation (RMI) [61]. The resource provider generates auction resources based on predefined resource definitions provided when the application is started. The test manager reads test definition files and submits tests to the AuctionComposer application which runs and controls simulations.

AuctionComposer

The AuctionComposer application runs auction simulations, playing the part of the auction owner. It maintains a pool of resources which it has been informed of and pending test simulations which have been requested by an auction test manager. When the application is started, several auction threads are created which monitor the pools of tests and resources, each running an auction at a time. The threads cycle through the pool of simulations attempting to run each test. If there are not enough resources of the required type to run a simulation then the simulation is added back to the pool and attempted at a later point.

For each protocol the composer must support a corresponding test runner which is provided by the protocol class. Test runners are implemented to interact with a protocol specific subclass of auction test which is used to create auction settings. In addition, the test runner interacts with the auction resources to run the auction as required by the protocol.

An auction thread when running an auction starts three services: the aforementioned test runner and two listeners, one listens for auction statistics, the other logging auction events. In order to retrieve auction statistics, a protocol must be set up to publish events of a GAF auction statistic type. The listeners record the events to a file specified within the auction test

definition.

Resource Provider

When the resource provider application is started it is passed as arguments the IP address and port that the auction composer has opened for connections and a file location of XML resource definitions. Resource definitions specify a list of protocol resource types with the corresponding number the provider can maintain simultaneously. When the application starts it reads this file but only instantiates resources as required to the limit configured in the test definition.

Once the resource provider has started it connects to the AuctionComposer application, informing it of the resource types the provider supports. When the composer requests a resource it is instantiated and passed to the composer. The composer will inform the resource provider to deallocate the resource at the end of the auction, at which point the provider can instantiate another when requested.

Auction Test Manager

An auction test manager reads XML auction test scenarios and submits them to an AuctionComposer for simulation. Scenarios are are read by a corresponding protocol specific parser which is included within the test manager and instantiated using reflection on the class name from an attribute in the test file. A snippet of an example test file showing the definition of a polynomial test auction scenario is given in Figure 6.6, where the different attributes are:

- parser: the protocol specific test parser class which parses the XML file. An initial reader will read this before instantiating it using reflection. The remainder of the file is parsed by this protocol specific test reader.

- timeout: A timeout after which the auction is disbanded with the resources being requested to be deallocated.
- biddingPhase: A timeout after which the auctioneer closes the bidding phase and the auction is evaluated.
- maxBidders: The maximum number of bidders to wait for if used in conjunction with the wait for maximum bidders tag.
- waitForMax: A flag which if set informs the auctioneer to close bidding once the maximum number of bidders have placed their bid. This is useful when testing in order to not have to wait for the bidding phase to end as a set number of bidders are expected.
- numEvaluators: The evaluator group size, i.e. the number of evaluators for each node.
- numMaskPublishers: The number of mask publishers to use.
- threshold: The threshold number of evaluators required to decrypt bids.
- constant: The constant value for weight resolution.
- maxCoefficient: The maximum coefficient which can be used for a random polynomial.
- toLog: A flag which indicates that events should be logged to a log file.
- logDir: The directory containing event logs.
- log: The filename of the log.
- requiresStatistics: A flag which indicates whether to log statistics.
- statsDir: The directory containing statistics logs.

- statsLog: The filename of the statistics log.

```
<tests>
        <parser>
                nz.ac.vuw.mcs.dsrg.auctionprotocols.secure.polynomial.testing.XMLPolynomialTestParser
        </parser>

        <auction timeout="900000" biddingPhase="105000" maxBidders="30" waitForMax="true" numEvaluators="21"
              numMaskPublishers="2" threshold="1" constant="5" maxCoefficient="10" toLog="false" logDir="~/
             polynomial/results/" log="bidders_3-log.txt" requiresStatistics="true" statsDir="~/auction/
             polynomial/results" statslog="bidders_3.txt">

                <title>Polynomial-bidders-3, R1</title>
                        <goods>
                        <good id="0" name="GOOD1" description="1"/>
                        <good id="1" name="GOOD2" description="2"/>
                        <good id="2" name="GOOD3" description="3"/>
                </goods>

                <bundles>
                        <bundle id="0" name="G1" description="G1">
                                <good id="0"/>
                        </bundle>
                        <bundle id="1" name="G2" description="G2">
                                <good id="1"/>
                        </bundle>
                        <bundle id="2" name="G3" description="G3">
                                <good id="2"/>
                        </bundle>
                        <bundle id="3" name="G1,G2" description="G1,G2">
                                <good id="0"/>
                                <good id="1"/>
                        </bundle>
                        <bundle id="4" name="G1,G3" description="G1,G3">
                                <good id="0"/>
                                <good id="2"/>
                        </bundle>
                        <bundle id="5" name="G2,G3" description="G2,G3">
                                <good id="1"/>
                                <good id="2"/>
                        </bundle>
                        <bundle id="6" name="G1,G2,G3" description="G1,G2,G3">
                                <good id="0"/>
                                <good id="1"/>
                                <good id="2"/>
                        </bundle>
                </bundles>

                <bidders>
                        <bidder name="Bidder_1" behaviour="nz.ac.vuw.mcs.dsrg.auctionprotocols.secure.
                              polynomial.testing.PolynomialBidBehaviour" >
                                <valuation value="1" goods="0"/>
                                <valuation value="1" goods="1"/>
                                <valuation value="1" goods="2"/>
                                <valuation value="1" goods="0,1"/>
                                <valuation value="1" goods="0,2"/>
                                <valuation value="1" goods="1,2"/>
                                <valuation value="1" goods="0,1,2"/>
                        </bidder>
                        ...
                </bidders>
        </auction>
        ...
</tests>
```

Figure 6.6: Snippet from sample test definition file.

Chapter 7

Evaluation

This chapter empirically evaluates GAF and the three protocols which have been implemented directly within it. Having implemented three protocols in GAF within a testing application allows self-reflection to be made on the framework usability, from both a protocol developer's and an application developer's perspective. The primary objective of this chapter is to determine whether the framework adds overhead to auction protocols and if so, by how much. This evaluation provides a test case of the framework and can be built on for a more detailed auction protocol analysis.

Section 7.1 provides the framework evaluation, section 7.2 evaluates each of the different protocols, and section 7.3 concludes the chapter by comparing the three protocols.

7.1 Framework Evaluation

7.1.1 Framework Overhead

In this section GAF is emperically evaluated for added overhead, by comparing polynomial auctions run independently and within (using simple object proxies) the framework without distributed message handling. For a sense of the RMI overhead incurred when using GAF in a distributed

system, the statistics for the first two tests are compared against polynomial auctions running in GAF on a single machine using RMI proxies.

Due to the nature of the winner determination problem, the single variable with the largest impact on a combinatorial auctions is the number of goods. Therefore this variable is used to compare autonomous polynomial auctions with those run within GAF with the respective proxy types. Auction evaluation time is plotted as this is the most computationally expensive of an auction's various phases and is the primary concern when comparing protocols.

The machine used for these experiments is a Toshiba Satellite M110 1.7GHZ Centrino Duo with 1GB of Memory. All experiments were run 30 times using the Sun 1.5.0 JVM. The machine was running no other applications other than the KDE x-windows operating system and its standard processes.

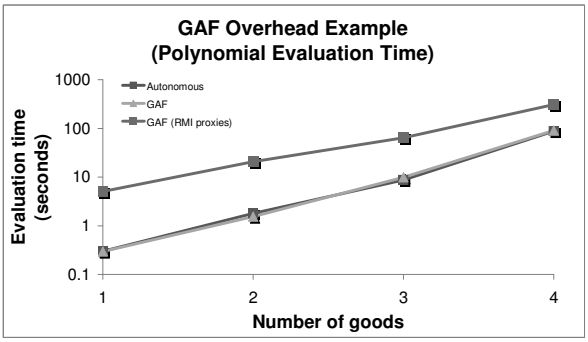

Figure 7.1: Overhead introduced by GAF for the polynomial protocol.

Figure 7.1 shows the comparative performance of the three different tests: autonomous (protocol running independently from GAF), auctions

within GAF, and RMI proxies. This experiment shows that running an auction protocol within GAF adds virtually no overhead to a protocol. The raw data showed an average maximum difference of no more than three seconds. However running the protocols in a distributed system using RMI proxies introduces significant costs which is to be expected as RMI is well known to be a costly communication solution. More interestingly, this experiment shows that the overhead from RMI has less of a relative impact as the number of goods increases. The reduction in communication overhead is due to the increase in evaluation time which outweighs communication costs as more evaluation computation is required.

7.1.2 Framework Usability

There are two different users of a GAF implementation which need to be considered. Protocol developers implement protocol specific auction behaviour conforming to GAF. Application developers leverage GAF's auction mechanism within their application, benefiting from the protocols. This section briefly explores what is required of each developer and the author's experience using GAF.

Protocol Developer

The protocol developer must divide auction behaviour between protocol specific implementations of the different auction resource types. Protocol specific messages are wrapped within standard message objects conforming to GAF. An application using GAF can use the protocol as long as it has been built to create auction parameters for it.

Application Developer

An application developer builds auction support into their application in three parts: auction resource allocation, communication, and protocol support. Resource allocation is the acquisition of auction resources to facili-

tate the auction, communication is the provision of a communication layer for resources to communicate, and protocol support bridges protocol behaviour for the application implementation such as resource configuration and auction instantiation.

The difficulty of distributing resources to the auction is as simple or difficult as the application requires. For the simulation application AuctionComposer described in section 6.3, resources are randomly selected from a set which registered with the central test server upon startup. This is a simple solution for a test application, but other systems would have a variety of mechanisms for choosing, distributing and selecting resources.

Providing a communication layer for resources involves wrapping GAF objects with remote proxies. Implementation of proxies in terms of technology and detail depends on the the implementation of GAF, but the technology for communication would depend primarily on application requirements. Although it is desirable that communication in independent from protocol development there may be some requirements of a protocol which have implications on communication, such as a requirement for secure communication between participants.

Integration for protocol support is required for each protocol. This is required given the variety of interactions between participants, and the different specifications of messages. For example, a user may need to transform auction parameters such as minimum bid into a representation specific to the protocol in question. The complexity of supporting each protocol depends on the complexity of the protocol itself, that is how much translation is required for elements. The auction protocols discussed in this thesis do not require a large amount of specialised parameters, or interactions with external participants so protocol support was relatively simple.

GAF is a relatively small framework and is kept simple by maintaining a small number of simple interfaces. Supporting the protocols implemented in this thesis within GAF was straight forward, implementing a

handful of resources which delegated to auction protocol libraries. It was more difficult to add resource wrappers for communication with RMI, but this was mainly due to difficulities with memory leaks due to Java RMI objects not always being closed. The greatest difficultly was the initial implementation of the secure auction protocol libraries.

As discussed in chapter 6 the protocols for the case studies in this thesis were developed in two different ways. The polynomial and garbled ciruit protocols were developed as a library independently of GAF and wrapped within GAF auction resources. The homomorphic protocol was implemented within GAF objects directly. When developing the homomorphic protocol, this made the development task slightly harder, in that testing the system was more difficult as additional application components were required. In comparison, if the protocol has been developed independently, then wrapped with GAF objects, it is relatively simple if the protocol elements are divided in a way that they can be used as auction resources. One disadvantage of developing a protocol independently is that synchronisation may not be a primary concern from the beginning of development if it is believed that GAF will be responsible for communication. It is important however to include a synchronisation strategy in the development of any distributed system.

7.1.3 Framework Security

GAF does not offer auction protocols running within it any additional security above what they provide themselves. The primary aim of GAF is to facilitate the use of multiple protocols in a systematic way so that protocols can be swapped at runtime with as little individual development for each protocol as possible. Services could, however be added to GAF to facilitate security requirements. One such concern is that currently, requests between resources are not authenticated and so any request would be responded to. There are several ways that GAF could be extended to

include authentication and authorisation of requests, such as a security layer or framework hooks which could be implemented by an application developer. Another concern is that messages between resources are not themselves encrypted. This could also be offered by GAF as a low level optional service which would secure some subset of auction messages.

7.2 Protocol Evaluation

Experiments have been performed for each of the three protocol case studies: polynomial, homomorphic and garbled circuit auctions.

The sensitivity of performance to different choices of variable is explored with respect to the number of goods, the number of bidders and the maximum bid price. Together in terms of scaleability, these primary variables are the most important factors as they determine how flexible the protocols are. These, therefore, provide the basis for how each of the protocols: can be used. The additional variables which are examined are the security properties of the protocols; the maximum coefficient, evaluator threshold and the number of mask publishers for the Polynomial Protocol and the keysize, as well as the evaluator threshold for the Homomorphic protocol.

The different variables may have a differing impact on each of the protocols different phases. The two auction phases which are present in all protocols are bid generation and total evaluation time, however each of the protocols also have protocol specific phases. In the Polynomial and Homomorphic protocols, total evaluation time is comprised of optimal value and optimal path determination. In the Garbled Circuit protocol, as discussed in section 4.3, the phases are structured quite differently, with part of the evaluation being predetermined during the bid submission.

In this chapter each of the phases in the Polynomial protocol are evaluated, but not for the other two protocols. In the Homomorphic protocol only the evaluation phase is evaluated as the protocol phases are so similar

to the Polynomial protocol that the same patterns occur. The Garbled Circuits results are divided into the different phases, however only evaluation is provided within the chapter for the primary variables. The remaining phases are provided in the appendix for the reader's reference. The Polynomial and Garbled Circuit results were generated and evaluated for this thesis, and the Homomorphic results were originally collected by Palmer [40]. The primary variables are evaluated for the Garbled Circuits protocol as a sample of the possible variables.

Experiments have been performed on up to 33 Dell Optiplex GX755 machines, each of which is a Intel Core 2 Duo 2.4GHz with 2GB of memory running NetBSD. All applications have been written in Java 5.0 and each resource was run in a Sun 1.5.0 JVM with a maximum heap space of 256mb. Bids were placed for all bundles (each combination of goods) and randomised from between one and the maximum bid.

Values graphed for each experiment are the average of 30 runs to reduce the effect of outliers. All machines are in a shared laboratory and are available to many users, however the experiments have been performed during off-peak periods to minimise any effect of outliers. Communication between all processes is through the GAF RMI proxies. The set number of goods and bidders where the respective variable is not under test is three and ten respectively. It is not possible to have a consistent maximum bid price because of the different mechanisms used by the protocols. These values are not necessarily a true reflection of the average auction, but they are non-trivial defaults which allow the effect of variables on scaleability to be explored. The default values of the other protocol-specifc variables are provided in each of the relevant sections.

The following experiments have been run for each of the protocols:

Increasing the Number of Goods

The scalability of a combinatorial auction protocol in terms of the number of goods is the most important concern. As the motivation for combinato-

rial auctions is to sell related goods in a single auction, this is the primary concern for flexibility. If the number of related goods requiring sale are greater than that of which the auction protocol can manage, a different protocol would be required, or the goods packaged into bundles. Almost all combinatorial auction protocols and certainly the protocols explored in this thesis will occur an exponential evaluation cost when increasing the number of goods. The simple reason being that as the number of goods increases, the number of potential solutions increase exponentially. The exception to this rule are auctions which use approximation evaluating on only a limited number of potential solutions, such as those in [20, 22].

Increasing the Number of Bidders

The number of bidders a protocol can reasonably support is also an important consideration, second to the number of goods. Increasing the number of bidders should result in a linear cost increase as the number of potential solutions will increase linearly.

Increasing the Maximum Bid

The maximum bid is the third most important performance consideration, as the number of potential bids is important when choosing a protocol for a particular purpose. Because of the different mechanisms for representing bids, the performance of different auctions under different maximum bids will differ.

7.2.1 Case study one: Polynomial Auction Protocol

The independent variables to vary for the Polynomial Protocol are the number of goods under auction, number of bidders participating, maximum bid price, threshold number of evaluators required to decrypt bids, number of mask publishers and the maximum coefficient. The different phases are bid generation, total evaluation time and its component phases:

optimal value and optimal path determination. Evaluation time makes up the majority of time taken to run an auction and this is provided for each experiment. Bid generation time is provided for each of the experiments, except where it is not relevant as bid generation is independent of other bidders. It is also not included in the mask publisher's experiment, as mask publishers are involved only in evaluation. Optimal value and optimal path determination are divided for the number of goods and bidders experiments, as the ratio of work is consistent over the other variables.

The protocol specific default variables for the polynomial protocol that are held constant are:

Variable default	Default value
Maximum bid	5
Threshold	1
Number of evaluators	*Varies according to other variables*
Number of mask publishers	2
Maximum coefficient	10

The default maximum bid and threshold are not sufficient for a real auction. They have been kept low because of the relationship between the maximum bid, the threshold and number of evaluators. This relationship cripples the scheme as was explained in section 4.4.1. A proposed modification to the protocol to reduce the effect of the relationship was provided in section 4.4.2. The number of mask publishers and maximum coefficient is also low, though there are currently no recommendations in the related work for suitable values.

Increasing the Number of Goods

This experiment shows that the number of goods under auction has a signifiant effect on scalability. An increase in goods, as shown in Figures 7.2 and 7.3, corresponds to an exponential increase in evaluation and bid

generation time respectively. Both of these figures grow exponentialy due to the exponential increase in goods combinations requiring evaluation. For this experiment, a maximum number of five goods has been used because a number greater than this requires the JVM heapspace to be increased past 256mb for each evaluator. Increasing the heapspace past 256mb would be unreasonable for the number of auction resources required to run for some of the experiments and, in addition, it would need to be greatly increased with more goods. It can be seen from the graph though that with an additional two goods, evaluation would take well over one thousand seconds per auction. Future work should investigate methods of reducing the memory usage of each protocol and providing a way in which a larger number of goods can be supported, such as caching intermediate results.

In the evaluation phase breakdown shown in figure 7.4, it is apparent that the growth of optimal path determination is less than that of value determination. Optimal path determination, though, is initially higher, due to the number of evaluations required during bidder search. The number of edges to search in optimal path determination is significantly lower than that in optimal value determination, due to the nature of edge traceback with dynamic programming. Therefore as the number of edges increases, the number of decryptions required during optimal value determination begins to outweigh the number of decryptions required when determining optimal edges and bidders.

Increasing the Number of Bidders

The effects of increasing the number of bidders on evaluation time are provided by Figures 7.5 and 7.6, as the total evaluation time and the break down of evaluation phases respectively. Increasing the number of bidders increases the evaluation time a slight amount linearly. This increase is minimal as all bids for a particular bundle are added together and evaluated together until the winning bidders need to be determined.

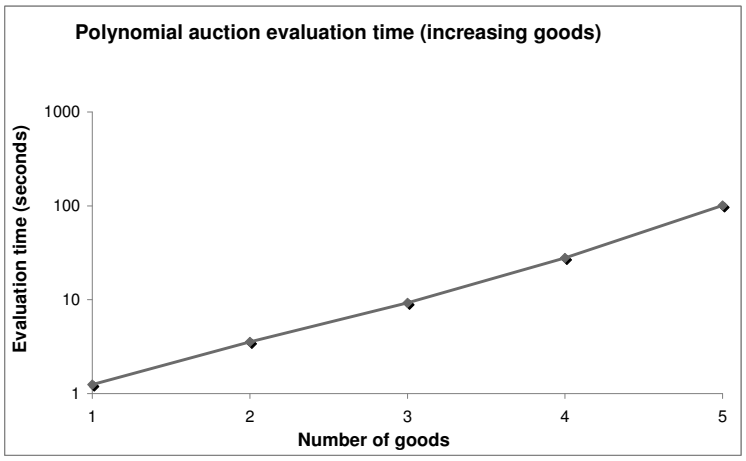

Figure 7.2: Effect of increasing the number of goods on total evaluation time.

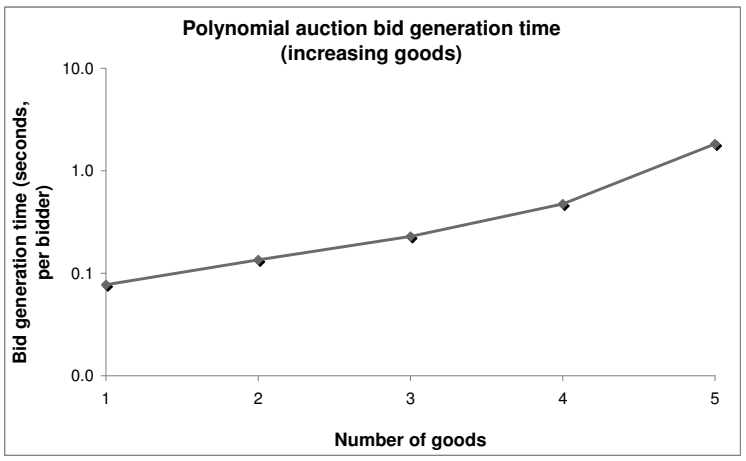

Figure 7.3: Effect of increasing the number of goods on bid generation time.

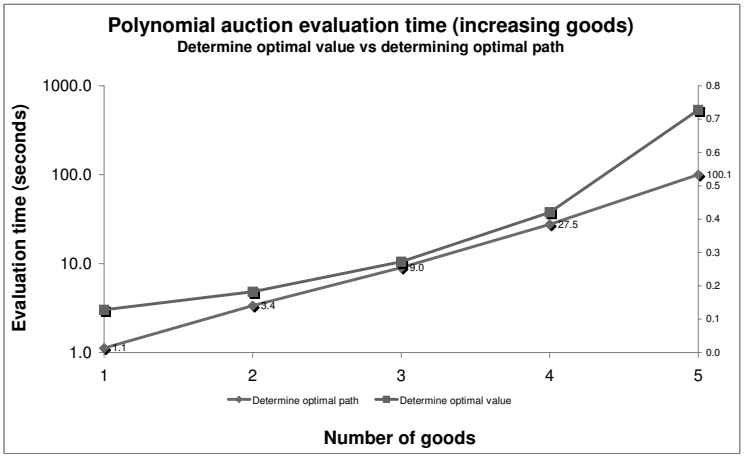

Figure 7.4: Effect of increasing the number of goods, optimal value determination vs optimal path determination.

The difference between optimal path and optimal value determination varies erratically, most likely due to demand of system resources by other users. As the evaluation times are short, they are greatly susceptible to other process demands.

Increasing the Maximum Bid

As shown in figure 7.7, evaluation time for the Polynomial Protocol does not scale well as the size of the maximum bid increases. This is due to the relationship between the number of goods, evaluators and the threshold (see section 4.4.2). For this experiment the number of goods and the threshold have been maintained at the defaults, so to increase the maximum bid the number of evaluators has to be increased. One consequence of this is that the security is reduced as the ratio of threshold to evaluators (t, n).

The exponential increase in evaluation time is due to the large number

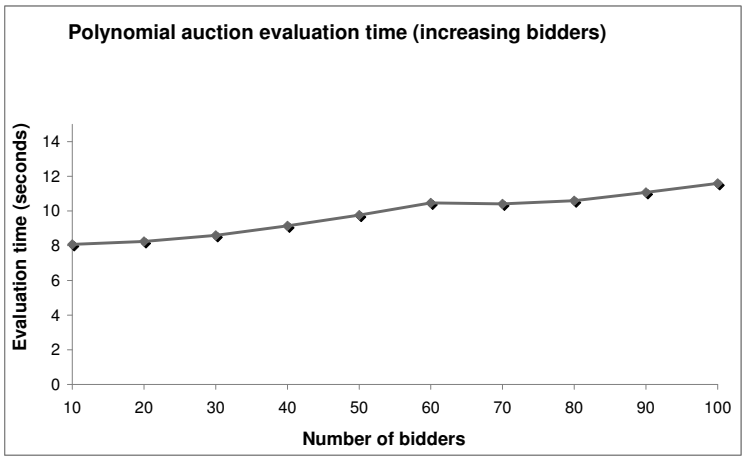

Figure 7.5: Effect of increasing the number of bidders on total evaluation time

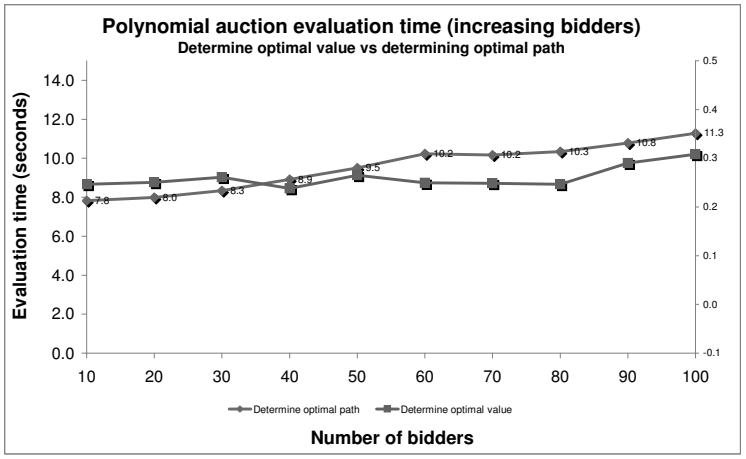

Figure 7.6: Effect of increasing the number of bidders, optimal value determination vs optimal path determination.

of evaluators which have been spread accross a consistent number of machines. Not only does the number of messages and amount of processing increase with the number of evaluators but, as the machines are limited, the resources must contend for system resources.

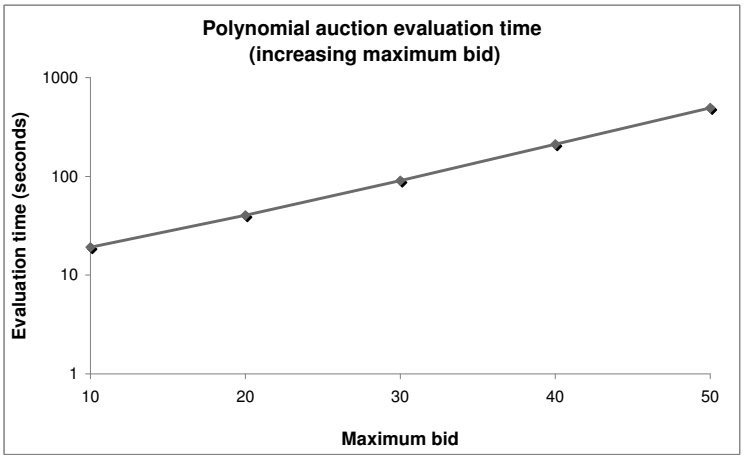

Figure 7.7: Effect of increasing the maximum bid on total evaluation time.

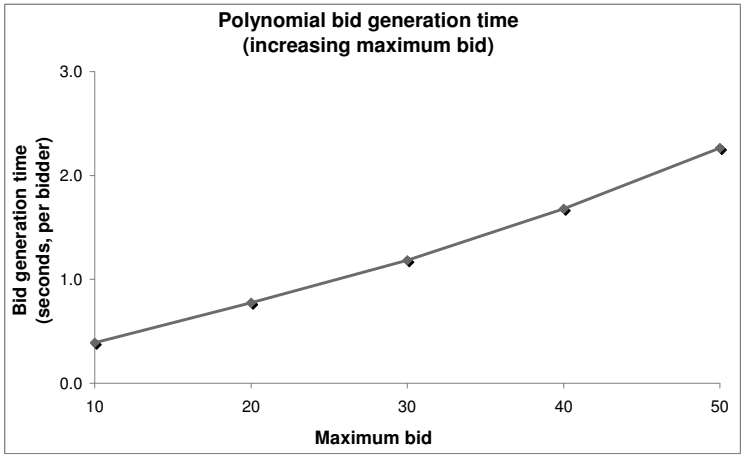

Figure 7.8: Effect of increasing the maximum bid on bid generation time.

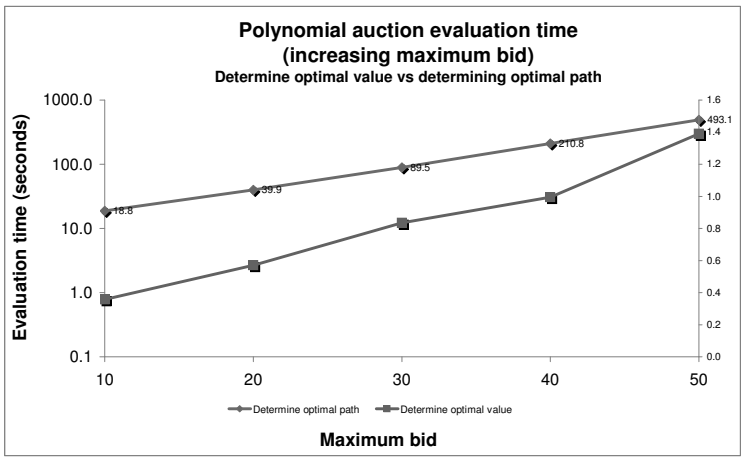

Figure 7.9: Effect of increasing the maximum bid, optimal value determination vs optimal path determination.

Increasing the Threshold Number of Evaluators

Exponential growth occurs when increasing the threshold number of evaluators, as depicted in Figure 7.10. Similarly to increasing the maximum bid, when increasing the threshold the number of evaluators used also has to be increased because of the circular dependency between evaluators, threshold and maximum price. As in the maximum bid experiment, increasing the number of evaluators introduces an exponential cost.

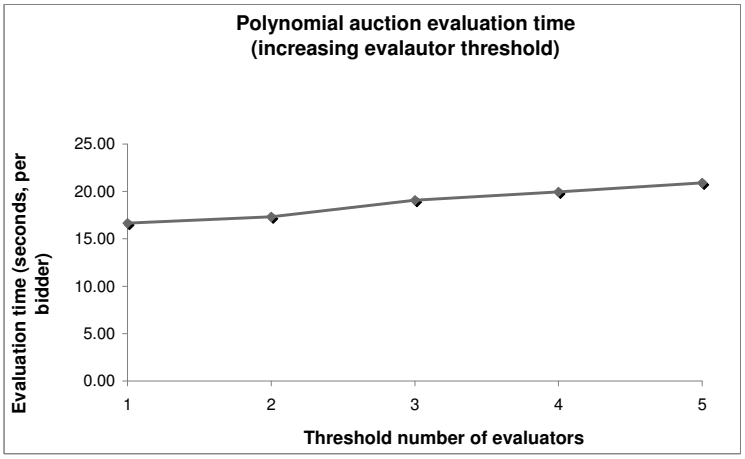

Figure 7.10: Effect of increasing the threshold number of evaluators on total evaluation time.

Increasing the Number of Masking Agents

Figure 7.11 shows the effect of increasing the number of mask publishers on total evaluation time and Figure 7.12 provides the breakdown of evaluation phases. Increasing the number of mask publishers has a linear effect on the total evaluation time. This is because of the time each evaluator must wait for the additional masking values from the added mask pub-

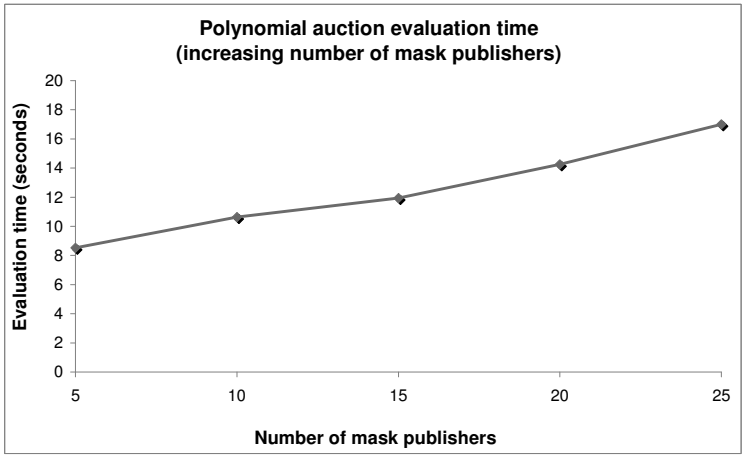

Figure 7.11: Effect of increasing the number of mask publishers on total evaluation time.

lishers. It is not due to the additional masks themselves, as they require only an extra addition operation which is trivial compared to the communication required to send the mask.

The number of mask publishers has a greater effect on determining the optimal path than on determining the optimal value. The minimal effect when determining the optimal value is due to a much smaller number of node evaluations required, compared to the number of bids which must be evaluated when determining the optimal path (hence a reduced number of masks required).

Increasing the Maximum Coefficient

There is no effect on evaluation or bid generation when increasing the maximum coefficient, therefore it is not shown. This is because increasing the maximum coefficient increases the pool that polynomials can be generated from, but polynomial operations remain constant irrespective

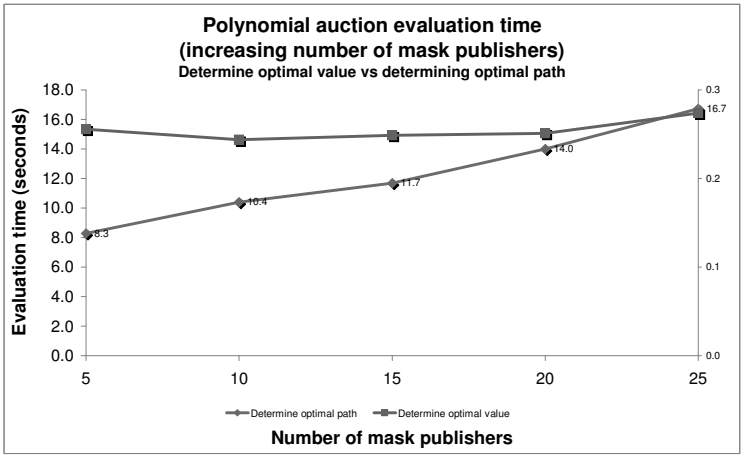

Figure 7.12: Effect of increasing the number of mask publishers, optimal value determination vs optimal path determination.

of polynomial size.

7.2.2 Case study two: Homomorphic Auction Protocol

The Homomorphic Auction Protocol shares the independent variables of the polynomial and Garbled Circuit Protocol: number of goods, bidders and maximum bid. The security variables are the threshold number of evaluators required to decrypt bid vectors and the keysize used for encryption. The different phases are the same as for polynomial auctions and are not explored individually as the phases exhibit the same behavioural patterns.

The experimental results evaluated in this section were produced by Palmer [40]. They do not include exploring the number of evaluators or threshold number of evaluators. Although they would be desirable experiments to perform, as one of the contributions of this thesis was imple-

menting Homomorphic Auctions with threshold encryption they are left to future work to reduce scope.

The following table provides the default protocols' specific variables used for the Homormophic auction tests. The number of evaluators and threshold are three and two respectively to provide a $(2, 3)$ threshold scheme which is equivalent to the Garbled Circuit protocol. The maximum bid is sixteen as this is the minimum bid for Garbled Circuit.

Variable default	Default value
Maximum bid	16
Threshold	2
Number of evaluators	3
Keysize	128

Increasing the Number of Goods

The impact of increasing the number of goods on evaluation time for the Homomorphic Protocol is shown in Figure 7.13. Similar to the effect of increasing the number of goods on the performance of the polynomial auction, increasing the number of goods in the Homomorphic Auction Protocol increases evaluation time exponentially.

Interestingly, the graph shows that evaluation time does not change between one and two goods. This is because actual evaluation is so cheap that the times recorded are equal to the minimum overhead. During evaluation, evaluators submit their calculations to the bulletin board and then wait until other evaluators have published their calculations. The minimum overhead is equal to the time threads spent sleeping while waiting for other evaluators.

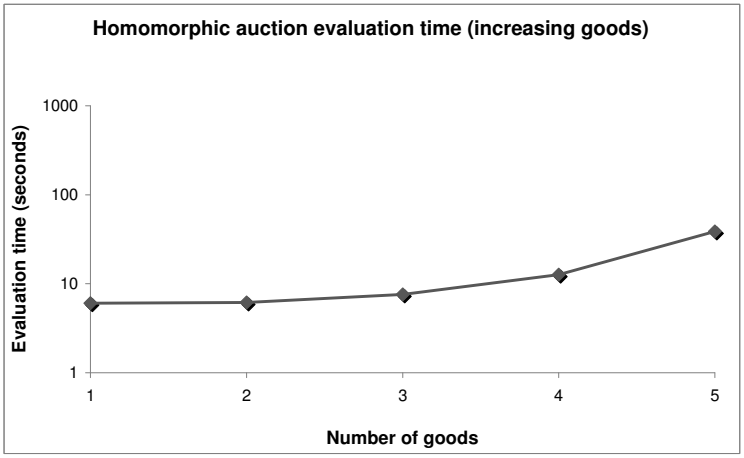

Figure 7.13: Effect of increasing the number of goods on total evaluation time.

Increasing the Number of Bidders

The effect of increasing the number of bidders on evaluation time shown in Figure 7.14 appears to be linear. There is an unusual kink between eighty and one hundred bidders, most likely due to increased load from other processes on the shared machines.

During optimal value determination, edge vectors have been multiplied together and so there is no impact from increasing the number of bidders. During optimal edge traceback, an additional bidder corresponds to an additional vector which will require decryption. Therefore the growth will be equal to the number of edges in the graph multipled by the additional number of bidders for each edge searched.

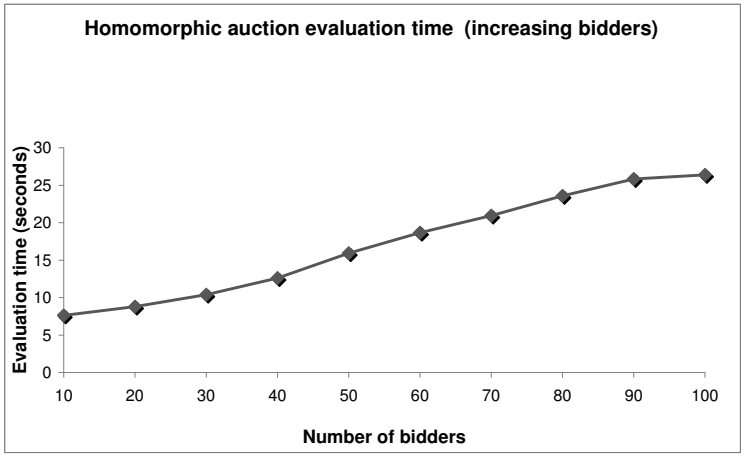

Figure 7.14: Effect of increasing the number of bidders on total evaluation time.

Increasing the Maximum Bid

The maximum bid in the Homomorphic Protocol corresponds directly to the length of the bid vector used for bidding. Figure 7.15 shows that the relationship between the vector length and evaluation time is linear. For each vector element which is added (the vector size increased) an additional decryption is required for each edge. Therefore, during optimal edge traceback, the number of decryptions will be increased by the number of edges, multiplied by the additional vector elements added and the number of bidders for each edge searched. During optimal value determination the effect will be neglible for smaller vector increases as decryptions are only carried out once for each node and multiplication (for masking) is neglible.

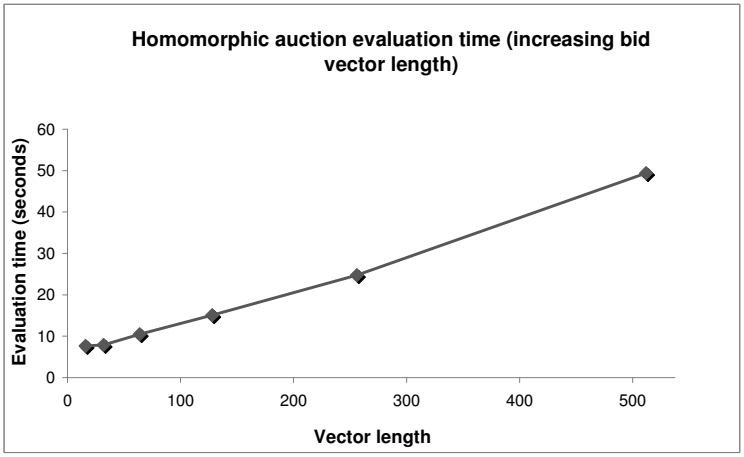

Figure 7.15: Effect of increasing the maximum bid on total evaluation time.

Increasing the Encryption Key Size

The cost of increasing the encryption key size is exponential as shown in figure 7.16. This is due to the greatly increased time to perform operations on the increased ciphertext components [40]. The cost of increasing key size is wide ranging as vectors include many encrypted values and there are many vectors for each bidder. Increasing this variable will also marginally increase time to setup and communications costs, as key generation will be more expensive and the bid vectors will be larger.

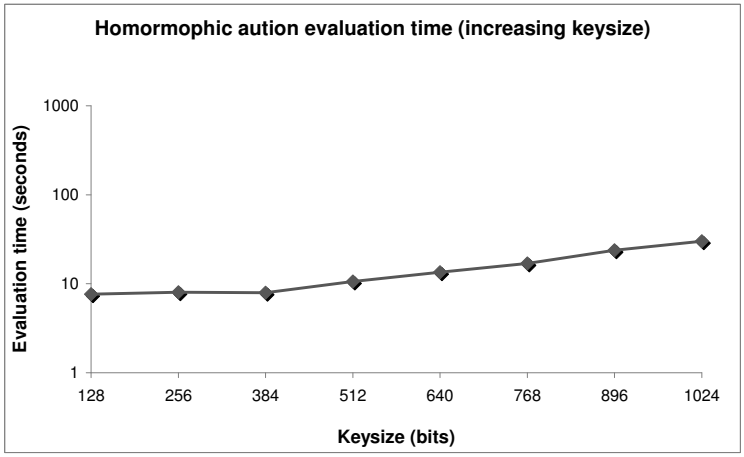

Figure 7.16: Effect of increasing the keysize on total evaluation time.

7.2.3 Case study three: Garbled Circuits Auction Protocol

The Garbled Circuit Protocol has been used as a third proof of concept, with performance statistics collected through GAF similarly to the first two protocols. Although a full collection of results for each variable and phase has been generated for this thesis, only a sample of the primary variables for the auction setup phase is provided, as a full analysis of the Garbled Circuits auction protocol is provided by Palmer [40]. The auction setup phase is the most computationally expensive part of a Garbled Circuit auction, as it includes pre-computation of part of the solution. The remaining graphs are included in appendix C for the reader's reference.

The only protocol specific default variable in the Garbled Circuit protocol is the maximum bid, which was sixteen. Figures 7.17, 7.18, 7.19 depict the effect on evaluation time for the Garbled Circuits protocol for the primary variables goods, bidders and maximum bid respectively. As in the first two protocols, increasing the number of goods increases evalu-

ation cost exponentially, while increasing the number of goods increases the evaluation cost linearly. The effect of increasing the maximum bid on evaluation is minimal and linear, much improved over the other protocols.

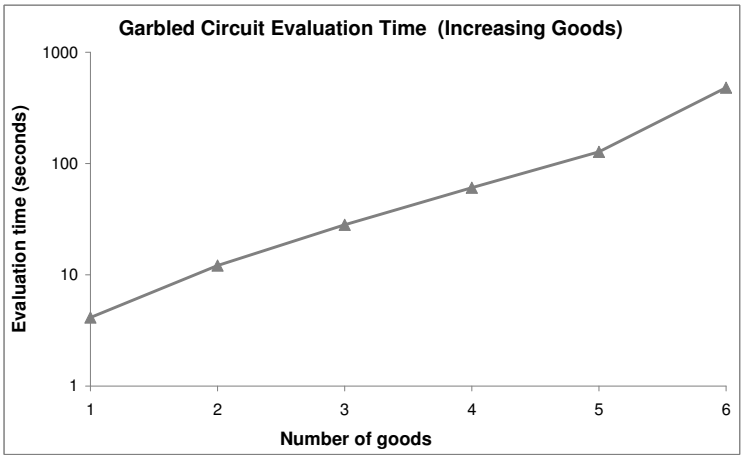

Figure 7.17: Garbled Circuit evaluation time: number of goods.

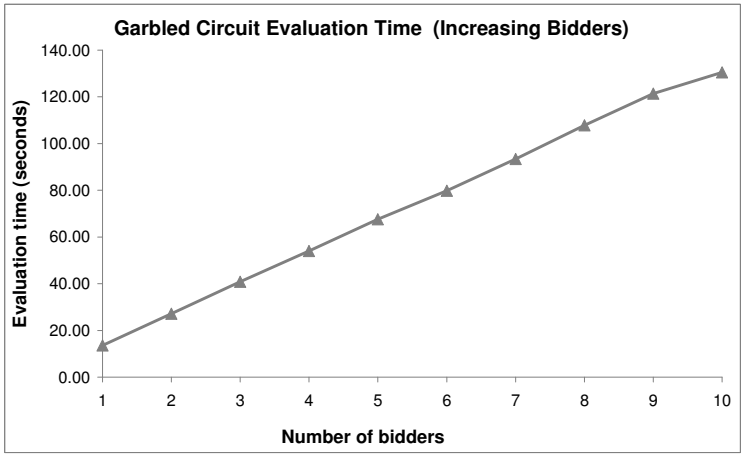

Figure 7.18: Garbled Circuit evaluation time: number of bidders.

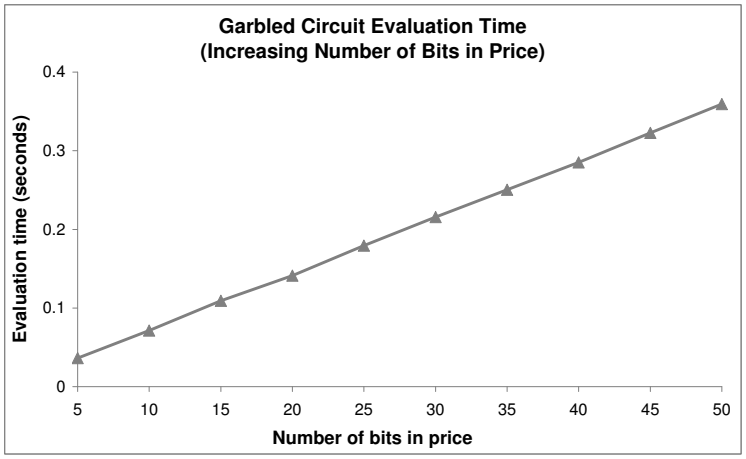

Figure 7.19: Garbled Circuit evaluation time: number of bits (max bid is 2^b where b is the number of bits used.

7.3 Protocol Comparison

A performance comparison of the three protocols under an increasing number of goods and bidders is provided in figures 7.20 and 7.21 respectively. How performance time scales as the maximum bid is increased is not provided as the three protocols are not comparable: the maximum bid for the polynomial protocol is in the tens, the homomorphic protocol in the hundreds; the garbled circuit protocol supports bids in the thousands. The security properties are not compared either, as future work is required to provide comparable values for the different security variables.

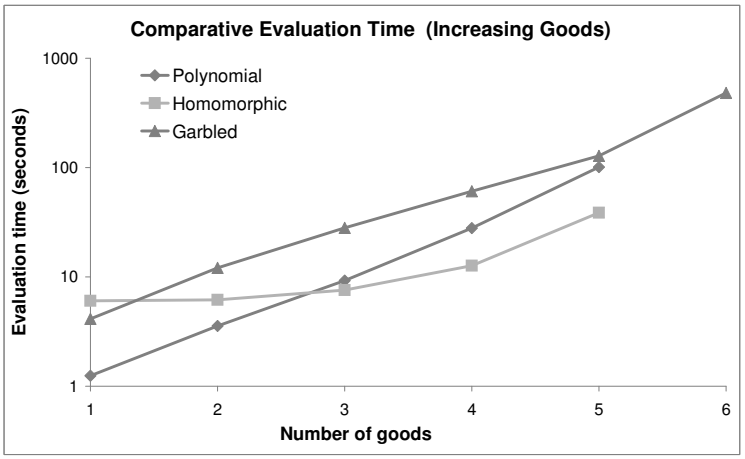

Figure 7.20: Comparative evaluation time: number of goods.

Each of the three protocols have exponential growth for an increasing number of goods, with the most expensive being the Garbled Circuit protocol. The Polynomial Protocol displays exponential growth while the Garbled Circuit protocol fluctuates. The Homomorphic Protocol arcs upwards but without further experiments with a higher number of goods the general performance pattern can not be determined.

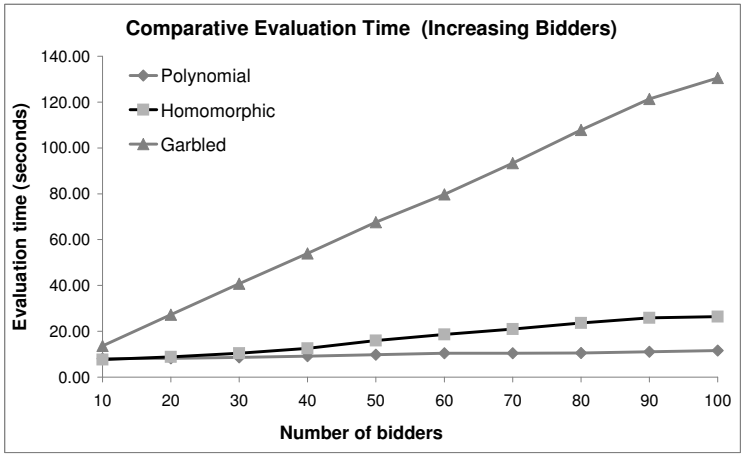

Figure 7.21: Comparative evaluation time: number of bidders.

When increasing the number of bidders, all three protocols have a steady linear increase in evaluation cost, however the garbled circuit protocol has the steepest growth. The Homomorphic and Polynomial protocols are much less affected, with the Homomorphic protocol being slightly more expensive. The reason is in the Garbled Circuit protocol the circuit is greatly increased by the addition of a new bidder, while in the other two protocols the additional multiplication or addition of encrypted values adds little cost. What little additional cost there is in the Homomorphic and Polynomial protocols is due to the added number of bids which need to be decrypted when determining which bidder placed the winning bid. This is minimal as only bids on the optimal path need to be decrypted

Although the Garbled Circuit protocol in these experiments has a higher evaluation cost and is affected the most by an increase in variables, this is artificially so. The figure provided for Garbled Circuits includes circuit generation which is the most costly part of the protocol and can be precomputed [40]. Once this is subtracted from the evaluation cost, Garbled

Circuits is a less costly protocol than the other two. The primary reason is that in the other two protocols, many evaluators must repeatedly perform decryption. In Garbled Circuits, once bidding has closed and the auctioneer has received each bidder's garbled input, the single auctioneer need only run the circuit which is much cheaper than running hundreds of decryptions.

Further work is required in order to run these auction protocols with a comparable security cost. In addition, optimisation is required of the protocols in order to run auctions with a higher number of goods. This would provide for a more thorough comparison.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

Auctions have a diverse range of applications including selling household goods, cars, allocating trucking and bus routes, airport takeoff and landing slots, frequency spectrums and grid system resources. There are also a large number of varying auction protocols to suit different applications, with differing attributes such as the supported number of goods, bidding language, solution algorithm and various types of auction rules. Without a structured approach to implementing auction protocols the developer of an auction application must learn the idiosyncrasies of many protocols. Forcing the use of mismatched protocols into a single application is difficult to implement, difficult to maintain and is not good design practice. This thesis develops the design of an auction framework which provides a structured approach to developing auction protocols and auction applications.

8.1.1 Contributions

The contributions of this thesis as listed in chapter one are:

1. This thesis develops a specification and design of an auction frame-

work called General Auction Framework (GAF) for the structured development of auctions protocols and applications.

2. An implementation of GAF has been built for this project in JAVA providing the base infrastructure for auction protocol development. In addition the work includes an implementation of a sample auction application called AuctionComposer which tests and gathers statistics on auction protocols included as GAF protocols. The application provides a proof of concept for the framework as it runs different types of auctions using GAF.

3. Three case studies of secure combinatorial auction protocols have been implemented in GAF. The first two, which were implemented for this thesis by interpreting the original papers, are Polynomial secret sharing, using dynamic programming [63], and Threshold homomorphic encryption, using dynamic programming [69]. The third is Garbled Circuits [38] which was initially built outside of GAF [40], but has been ported into GAF by the author for this thesis.

4. Explanations and worked examples are provided for the two dynamic programming protocols as both original papers are difficult to understand and missing detail required for implementation.

5. An introductory performance analysis of the three secure combinatorial auction protocols. This analysis is intended to form the beginning of more detailed future work analysing the protocols.

6. A proposed modification of the Polynomial secret sharing auction protocol which significantly improves scalability.

8.2 Future Work

8.2.1 GAF

- For GAF to be of use in production systems, it requires further development, offering additional low level services. One such service is a system for authentication and authorisation. This would enable an application to provide an interface that auction resources can use to identify and authenticate participants as well as determine what requests are permitted. Authorisation of participants needs to be specified partially by the protocol developer as to what protocol actions (such as closing an auction or withdrawing a bid) are permissible and whether there are any attached conditions are protocol specific.

- Further analysis of the framework should be performed which compares the effort of developing a protocol within GAF against developing a protocol individually. This could involve some form of time based analysis or comparing lines of code.

- Additional proxy types could be added to the framework in order to facilitate its usefulness to application developers. For example a secure proxy could be added to be nested inside other proxies offering encryption, or other types of communication technologies such as network sockets. In addition, the framework needs to be modified to allow multiple proxy types running within an application. Another potential requirement is determining whether GAF could be used in a virtually state-less environment such as web services.

8.2.2 Auction Protocol Development

- The auction protocols discussed in this thesis are inflexible. Further work is required to determine how the secure schemes could be modified to allow for modifications to the protocols. This includes

for example GVA pricing and a offering a more extensive range of bidding languages.

- The exponential nature of combinatorial auctions is a limiting factor, which can be mitigated in some instances with protocols which approximate the solution. It should be investigated whether an approximation combinatorial auction protocol can be implemented with privacy preserving properties.

- An extension to the polynomial auction protocol has been offered which requires implementation, testing and analysis.

8.2.3 Auction Protocol Evaluation

The three protocols used as cases studies in this thesis require more evaluation. The Garbled Circuit protocol implementation has been evaluated well in [40], however the Polynomial and Homomorphic protocols require additional work. Three future tasks for providing better auction evaluation are:

- The Homomorphic protocol requires evaluation of performance under differing numbers of evaluators and different threshold sizes. This is important as it is the threshold ratio of evaluators which provide the primary security of the scheme.

- Optimising protocols to enable running auctions of any number of goods. In order to provide a more thorough pattern of increased cost for increased goods, the protocols should support at least ten goods but it would be ideal if they were capable of running a higher number. This would involve adapting the protocols to reduce their memory usage as much as possible, and providing a method for caching the internal calculations.

- For a complete comparison of protocols, a study is required to determine how secure each scheme actually is, including identifying comparable values for each of the security properties. A baseline may be the length of the time required to break bid encryption, or the number of participants required to cheat in order to disrupt or break security requirements. This would also make it possible to make an advised decision when choosing between the protocols, as currently the quality of security of each protocol is relatively abstract.

In addition to these tasks, a formal analysis is required, contrasting each of the different protocols using the identified comparable security requirements. Included in this is determining the primary strengths and weaknesses of each scheme and under what conditions it could be suitably used. These conditions could then be used to determine if one of these protocols could be used for resource allocation in a grid system, or, if not, provide more insight into what is required of a secure combinational auction for grid resource allocation.

8.2.4 AuctionComposer

AuctionComposer needs to be extended in order to provide the ability to compare different protocols in terms of their outcome and test protocols by comparing actual outcome with an expected outcome.

8.2.5 Summary

The General Auction Framework (GAF) which is developed in this thesis provides a structured approach to building auction protocols and applications. GAF's design divides auction behaviour into logical components which all studied auction protocol can be divided into. The framework defines two sets of functionality: frozen spots, which the framework developer provides are fixed behaviour which all protocols use, and hotspots,

which are the variable aspects of GAF. The variable aspects are further divided into two types, firstly protocol hotspots which are implemented by a protocol developer effectively forming the protocol. The second type is application hotspots, which define behaviour which must be implemented by an application developer providing application specific behaviour such as communication.

The framework has been shown to be useful and versatile through the use of case study protocols and a test application. With minimal effort, protocols can be included within GAF and used by applications leveraging the framework. The protocols, being quite different in design, provide good test cases as they show GAF to be flexible enough to manage diverse protocols. As auction resource behaviour is completely defined within protocol specific hotspots it is possible to implement a protocol however a protocol developer wishes. The added value that GAF brings is the standardised interfaces containing the hotspots, the distinction between different auction resources and the frozen spots which form the underlying infrastructure of an auction protocol.

The test application provides both a test case for the framework, as an example application built with GAF, and it is also a useful tool in itself. The application provides the ability to record statistics for any protocol auction built using GAF which publishes GAF statistics events. When a new protocol is required to be integrated within the testing application, the application developer only needs to add a test runner and a test parser. Once complete, XML tests of the corresponding protocol type can be submitted to an AuctionComposer through an auction manager. The AuctionComposer statistics listener will output the results to a file specified in the test XML which can be collated as required.

AuctionComposer currently supports three protocols: secure combinatorial polynomial auctions, secure combinatorial homomorphic auctions, and secure garbled circuit auctions. Combinatorial auctions allocate multiple goods amongst bidders in a single auction. Secure auctions provide

security related auction properties to an auction. One such security property is privacy preservation where valuations of losing bidders are not discovered. All three of the case study protocols are privacy preserving combinatorial auctions.

The first two protocols, which were implemented especially for this thesis, have been fully described with worked examples in this work where in the original papers there was ambiguity. Included in this description is explanation on how to run Homomorphic protocols with threshold encryption, which had previously only been stated as possible. The garbled circuits protocol has not been fully defined as it was not implemented for this thesis and is adequately described in the original literature. In addition to the protocol descriptions, a proposed modification for the Polynomial protocol is given which should improve performance and pricing flexibility. As the number of goods increases, the relative gain from the modification is increased significantly.

Appendix A

GAF Auction Resource Use Cases

Use case name	CreateAuthenticatedRequest
Participating actors	Initiated by AuctionComponent
Flow of events	**1.** Create signature for request. **2.** Attach signature to request.
Entry conditions	• The AuctionComponent has a request to be signed.
Exit conditions	• The AuctionComponent has a signed request.
Quality conditions	

Figure A.1: AuctionComponent use case: CreateAuthenticatedRequest.

Use case name	AuthenticateRequest
Participating actors	Initiated by AuctionComponent Communicates with AuthenticationService
Flow of events	**1.** The AuctionComponent requests any required data from the AuthenticationService. **2.** The AuthenticationService returns the required data. **3.** The AuctionComponent checks whether the request was authentic.
Entry conditions	• The AuctionComponent has a signature to authenticate.
Exit conditions	• The AuctionComponent knows whether the request is authentic or not auction.
Quality conditions	

Figure A.2: AuctionComponent use case: AuthenticateRequest.

Use case name	SetupAuction
Participating actors	Initiated by AuctionOwner Communicates with AuctionCreator
Flow of events	**1.** The AuctionOwner requests the AuctionCreator create an auction instance using specified participants, protocol and policy. **2.** The request is authenticated. **3.** If the request was authentic, the AuctionCreator creates the auction and returns the auction details.
Entry conditions	• The AuctionOwner has selected and successfully negotiated to use an AuctioneerGroup, PolicyManager, BidPublisher, EvaluatorGroup ResultPublisher, AuctionVerifier • The AuctionOwner has chosen an auction protocol and corresponding policies.
Exit conditions	• An auction owner has either had an auction instance created and received the auction details, or an error message was returned.
Quality conditions	• The inherited AuctionComponent use-case CreateAuthenticatedRequest is included in step 1. • The inherited AuctionComponent use-case AuthenticateRequest is included in step 2.

Figure A.3: AuctioneerOwner use case: SetupAuction.

Use case name	WithdrawAuction
Participating actors	Initiated by AuctionOwner Communicates with AuctionCreator
Flow of events	**1.** The AuctionOwner requests the AuctionCreator delete an auction instance. An AuctionReference is attached. **2.** The request is authenticated. **3.** If the request was authentic, the AuctionCreator finds the auction instance process matching the AuctionReference. If no instance was found an error is returned. **3.** If the request was authentic and the auction instance was found, the AuctionCreator attempts to delete the auction and returns either an error or success message. **4.** The AuctionOwner receives a success or error message.
Entry conditions	• The AuctionCreator has previously created an auction instance for the AuctionOwner. • The AuctionOwner posses the AuctionReference for the auction instance.
Exit conditions	• An AuctionOwner has received an error or success message corresponding to whether the auction instance was deleted.
Quality conditions	• The inherited AuctionComponent use-case CreateAuthenticatedRequest is included in step 1. • The inherited AuctionComponent use-case AuthenticateRequest is included in step 2.

Figure A.4: AuctioneerOwner use case: WithdrawAuction.

Use case name	CreateAuction
Participating actors	Initiated by AuctionCreator Communicates with AuctioneerGroup, PolicyManager, BidPublisher, EvaluatorGroup ResultPublisher, AuctionVerifier
Flow of events	**1.** The AuctionCreator requests participation by the auction participants concurrently (AuctioneerGroup, PolicyManager, BidPublisher, EvaluatorGroup ResultPublisher, AuctionVerifier). **2.** The participants verify the requests. **3.** The participants attempt to start auction instance processes. **4.** The participants return error or success messages. **5.** If any of the participants returned a success message the auction instance creation is canceled. **6.** If creation was canceled, any participants who accepted are sent cancellation requests.
Entry conditions	• The AuctionComponent has been informed which participants to use for the auction instance.
Exit conditions	• Either the auction was created and all the participants returned success messages, or at least one participant returned an error. • If creation was canceled, all accepted participants have been informed of auction instance cancellation.
Quality conditions	• The inherited AuctionComponent use-case CreateAuthenticatedRequest is included in step 1. • The inherited AuctionComponent use-case AuthenticateRequest is included in step 2. • The inherited AuctionComponent use-case CreateAuthenticatedRequest is included in step 4.

Figure A.5: AuctionCreator use case: CreateAuction.

Use case name	CancelAuction
Participating actors	Initiated by AuctionCreator Communicates with AuctioneerGroup, PolicyManager, BidPublisher, EvaluatorGroup ResultPublisher, AuctionVerifier
Flow of events	**1.** The AuctionCreator checks whether the policies and auction instance state allow the auction to be cancelled. **2.** The AuctionCreator is informed of whether the auction can be cancelled. **3.** If the auction can be canceled, a cancel request is sent to each participant. An AuctionReference matching the auction instance is included with the request. **4.** The participants verify the requests. **5.** If the requests are authentic, the participants locate the auction instance processes. **5.** If the auction instance processes were located, the participants cancel their instance processes. **6.** The participants inform the AuctionCreator whether the auction instance was cancelled. **7.** The AuctionCreator receives either success messages indicating the participants canceled their processes or messages indicating the auction processes were not found.
Entry conditions	• The AuctionCreator has an AuctionReference for an auction instance to be cancelled.
Exit conditions	• The auction instance has been cancelled.
Quality conditions	• The inherited AuctionComponent use-case CreateAuthenticatedRequest is included in step 3. • The inherited AuctionComponent use-case AuthenticateRequest is included in step 4.

Figure A.6: AuctionCreator use case: CancelAuction.

Use case name	PlaceBid
Participating actors	Initiated by Bidder Communicates with AuctioneerGroup
Flow of events	**1.** The Bidder sends the bid to an AuctioneerGroup. **2.** The AuctioneerGroup authenticates the request. **3.** If the bid is acceptable and the request was authentic the AuctioneerGroup adds the bid to the bid collection. **4.** If the request was authentic, the AuctioneerGroup returns a BidReference to the Bidder. **5.** The Bidder receives the BidReference.
Entry conditions	• The bidder has created a bid for an auction.
Exit conditions	• The bidder's bid has been recorded or rejected.
Quality conditions	• The inherited AuctionComponent use-case CreateAuthenticatedRequest is included in step 1. • The inherited AuctionComponent use-case AuthenticateRequest is included in step 2.

Figure A.7: Bidder use case: PlaceBid.

Use case name	WithdrawBid
Participating actors	Initiated by Bidder Communicates with AuctioneerGroup
Flow of events	**1.** The Bidder sends a BidReference with a request to withdraw the bid to the AuctioneerGroup. **2.** The AuctioneerGroup authenticates the request. **3.** If the request was authentic, the AuctioneerGroup finds the bid matching the BidReference. If no bid was found an error is returned. **4.** If the request was authentic and the bid was found, the AuctioneerGroup attempts to remove the bid and returns either an error or success message. **5.** The Bidder receives a BidStatus or error message.
Entry conditions	• The bidder has a BidReference for the bid to withdraw.
Exit conditions	• If the request was authentic the bidders bid has been withdrawn or the request has been rejected. • If the request was not authentic, an error message was returned.
Quality conditions	• The inherited AuctionComponent use-case CreateAuthenticatedRequest is included in step 1. • The inherited AuctionComponent use-case AuthenticateRequest is included in step 2.

Figure A.8: Bidder use case: WithdrawBid.

Use case name	RequestBidStatus
Participating actors	Initiated by Bidder Communicates with AuctioneerGroup
Flow of events	**1.** The Bidder sends a BidReference with a request for BidStatus to the AuctioneerGroup. **2.** The AuctioneerGroup authenticates the request. **3.** If the bid exists and the request was authentic, the AuctioneerGroup returns a new BidStatus. **4.** The Bidder receives a BidStatus or error message.
Entry conditions	• The bidder has placed a bid matching the BidReference.
Exit conditions	• The bidder received a BidStatus matching the BidReference submitted or an error.
Quality conditions	• The inherited AuctionComponent use-case CreateAuthenticatedRequest is included in step 1. • The inherited AuctionComponent use-case AuthenticateRequest is included in step 2.

Figure A.9: Bidder use case: RequestBidStatus.

Use case name	AddBid
Participating actors	Initiated by AuctioneerGroup Communicates with BidPublisher
Flow of events	**1.** The AuctioneerGroup sends a Bid to the BidPublisher. **2.** The BidPublisher authenticates the request. **3.** If the request was authentic, the BidPublisher adds the bid to the bid collection and returns a BidReference. **4.** The AuctioneerGroup receives a BidReference or error message.
Entry conditions	• The AuctioneerGroup has received a bid from a bidder.
Exit conditions	• The AuctioneerGroup received a BidReference matching the bid submitted or an error.
Quality conditions	• The inherited AuctionComponent use-case CreateAuthenticatedRequest is included in step 1. • The inherited AuctionComponent use-case AuthenticateRequest is included in step 2.

Figure A.10: AuctioneerGroup use case: AddBid.

Use case name	RemoveBid
Participating actors	Initiated by AuctioneerGroup Communicates with BidPublisher
Flow of events	**1.** The AuctioneerGroup checks whether the policies and auction instance state allow the bid to be removed. **2.** The AuctioneerGroup is informed of whether the bid can be removed. **3.** If the bid can be removed the AuctioneerGroup sends a BidReference to the BidPublisher. **2.** The BidPublisher authenticates the request. **3.** If the request is authentic, the BidPublisher locates the bid. **4.** If the bid was located, the BidPublisher removes the bid. **5.** The BidPublisher returns a new BidStatus. **6.** The AuctioneerGroup receives a BidStatus.
Entry conditions	• The AuctioneerGroup has received a BidReference from a bidder for a bid to be withdrawn.
Exit conditions	• The AuctioneerGroup received a BidStatus matching the BidReference.
Quality conditions	• The inherited AuctionComponent use-case CreateAuthenticatedRequest is included in step 1. • The inherited AuctionComponent use-case AuthenticateRequest is included in step 2.

Figure A.11: AuctioneerGroup use case: RemoveBid.

Use case name	SolveAuction
Participating actors	Initiated by EvaluatorGroup
Flow of events	**1.** EvaluatorGroup requests bids from the BidPublisher. **2.** The BidPublisher authenticates the request. **3.** If the request is authentic, the BidPublisher returns the bids. **4.** The EvaluatorGroup uses a protocol specific algorithm to solve the auction (finding the optimal allocation). **5.** The EvaluatorGroup requests the results be published by the ResultPublisher. **6.** The ResultPublisher authenticates the request. **7.** If the request is authentic, the ResultPublisher publishes the result.
Entry conditions	• The auction is ready to be solved, this will be dictated by protocol or policy.
Exit conditions	• The optimal allocation has been found, and the result published to the ResultPublisher.
Quality conditions	• The inherited AuctionComponent use-case CreateAuthenticatedRequest is included in step 1. • The inherited AuctionComponent use-case AuthenticateRequest is included in step 2. • The inherited AuctionComponent use-case CreateAuthenticatedRequest is included in step 5. • The inherited AuctionComponent use-case AuthenticateRequest is included in step 6.

Figure A.12: EvaluatorGroup use case: SolveAuction.

Use case name	VerifyOutcome
Participating actors	Initiated by AuctionVerifier
Flow of events	**1.** The AuctionOwner requests the selected PolicyManager, BidPublisher, AuctioneerGroup, EvaluatorGroup and ResultPublisher to run an auction.
Entry conditions	• The AuctionOwner has selected a PolicyManager, BidPublisher, AuctioneerGroup, EvaluatorGroup and ResultPublisher.
Exit conditions	• The AuctionOwner has either found a full set of objects to run the auction, or SetupAuction has been canceled.
Quality conditions	

Figure A.13: AuctionVerifier use case: Verify outcome.

Use case name	AdaptComponent
Participating actors	Initiated by PolicyManager
Flow of events	**1.** The AuctionOwner requests the selected PolicyManager, BidPublisher, AuctioneerGroup, EvaluatorGroup and ResultPublisher to run an auction.
Entry conditions	• The AuctionOwner has selected a PolicyManager, BidPublisher, AuctioneerGroup, EvaluatorGroup and ResultPublisher.
Exit conditions	• The AuctionOwner has either found a full set of objects to run the auction, or SetupAuction has been canceled.
Quality conditions	

Figure A.14: PolicyManager use case: AdaptComponent.

Use case name	ObserveAuction
Participating actors	Initiated by AuctionObserver
Flow of events	**1.** The AuctionOwner requests the selected PolicyManager, BidPublisher, AuctioneerGroup, EvaluatorGroup and ResultPublisher to run an auction.
Entry conditions	• The AuctionOwner has selected a PolicyManager, BidPublisher, AuctioneerGroup, EvaluatorGroup and ResultPublisher.
Exit conditions	• The AuctionOwner has either found a full set of objects to run the auction, or SetupAuction has been canceled.
Quality conditions	

Figure A.15: AuctioneerOwner use case: ObserveAuction.

Appendix B

Full GAF Hotspot Specification

Bidder, bid generation:

The bidder uses the bid generation hotspot to generate a bid from provided settings and the protocol for the auction instance the bids will be submitted to.

- **Examples:** For an English auction, the bidder just needs to combine the auction reference with the amount to bid. In a homomorphic auction, the bidder needs to encrypt values for combinations of goods using particular keys, and match them to the combinations and the auction reference.
- **Bind time:** The hotspot is protocol specific, so it is rebound at runtime for each auction instance.
- **Responsibility:** The hotspot generates a bid for some provided settings and protocol.
- **Variability:** The single variable aspect is the algorithm the bidder uses to generate the bid including communication between other participants and order of operations. This is elementary so the hotspot is also elementary.

- **Multiplicity:** Only one bid generation hotspot implementation can be used per auction instance. The hotspot can not be parametrised as the algorithm will be completely different for different protocols.

Auctioneer, bid processing:

The auctioneer processes incoming bids from bidders. Each bid may have a protocol specific operation performed on it.

- **Examples:** In an English auction, the auctioneer receives bids checks that they are greater than the reserve price. Bids lower than the reserve price are discarded and the bid reference provided to the bidder informs of this. In the homomorphic auction, the bids are stored immediately, as it is the evaluators that decrypt the bids.

- **Bind time:** The hotspot is protocol specific, so it is rebound at runtime for each auction instance.

- **Responsibility:** The hotspot must process incoming bids.

- **Variability:** The variable aspect is the algorithm the auctioneer uses including communication between other participants and order of operations. Bid references should be generated by this hotspot because they will be protocol specific and directly related to (and can not be separated from) the way bids are processed. For instance the reference passed back to the bidder may contain whether the bid was accepted or not or how it was modified by the bid processor. These are both elementary aspects so the hotspot is also elementary.

- **Multiplicity:** Only one bid processing hotspot implementation can be used per auction instance. The hotspot can not be parametrised as the algorithm will be completely different for different protocols.

Auctioneer, bid storage:

This hotspot stores and retrieves bids for a bidder and manages references that the auctioneer can use to identify bids. All competing bids are recorded in bid storage.

- **Examples:** An auctioneer using a standard hash map could just set the bid for the bid reference generated during bid processing. A database implementation will require operations to interact with a database.
- **Bind time:** Bid storage is application specific and so will be specified during implementation.
- **Responsibility:** The bid storage should provide storage and retrieval capabilities, as well as generate bid references.
- **Variability:** The variable aspect is the type of storage for bids.
- **Multiplicity:** Only one hotspot implementation can be used at a time.

Auctioneer, bid status:

Bid status is protocol dependent but indicates the state that a bid is in, for example 'considering' or 'outbid'. The bid status is provided by the auctioneer to a bidder who requests the status of a bid they have placed.

- **Examples:** In an English auction, the auctioneer checks to see whether the bidders bid is winning and if not then it informs the bidder that it has been outbid. In a sealed bid auction, either the bid is being considered or not.
- **Bind time:** The hotspot is protocol specific, so it is rebound at runtime for each auction instance.

- **Responsibility:** The auctioneer checks the status of the bids and reports the status back to the bider.
- **Variability:** The variable aspect is the algorithm for determiing the status of the bid.
- **Multiplicity:** Only one hotspot implementation can be used at a time.

Verifier, verification:

The verifier performs some protocol specific algorithm to determine that the outcome of the auction was correct.

- **Examples:** A verifier in an English auction would use the sets of bidders and bids and perform the same operation as the auctioneer. The verifier would then determine whether the result was correct. In one version of the sealed bid auction protocol, bidders submit encrypted bids but don't publish the decryption keys until bidding is closed. The verifier decrypts the bids using the published keys, checking that the result found is the same by the evaluator.
- **Bind time:** The hotspot is protocol specific, so it is rebound at runtime for each auction instance.
- **Responsibility:** The hotspot performs some algorithm to verify the outcome of the auction.
- **Variability:** The single variable aspect is the algorithm the verifier uses including communication between other participants and the order of operations. This is elementary so the hotspot is also elementary.
- **Multiplicity:** Only one verification implementation can be used per auction instance.

Verifier, fetch verification data:

The verifier fetches data required for verification.

- **Examples:** In a standard English auction, the verifier can fetch all bid details, including bidder names, valuations and time of bid. In a secure English auction which hides bidder details, the verifier will only be allowed to fetch the valuations.
- **Bind time:** The hotspot is protocol specific, so it is rebound at runtime for each auction instance.
- **Responsibility:** The hotspot fetches the data the caller is permitted to fetch for the protocol and/or participant.
- **Variability:** The single variable aspect is what the verifier is permitted to fetch.
- **Multiplicity:** Only one fetch verification data can be used per auction instance as. The hotspot be generalised for any protocol using parametrisation to limit what can be fetched.

Auction owner, create auction:

The auction owner creates the auction. This hotspot performs any protocol specific pre-auction setup, generating an auction reference.

- **Examples:** In the English auction, the auctioneer uses auction settings and/or polices to define the length of the auction and a reserve price. These are included in an auction reference which is returned to the auction owner. In the secure homomorphic auction the auctioneer has to generate encryption keys for bids and requests the other resources to be set up.
- **Bind time:** The hotspot is protocol specific, so it is rebound at runtime for each auction instance.

- **Variability:** The single variable aspect is the algorithm the evaluator uses including communication between other participants and the order of operations. This is elementary so the hotspot is also elementary.

- **Multiplicity:** Only one auction creation hotspot implementation can be used per auction instance. The hotspot can not be parametrised as the algorithm will be completely different for different protocols.

Evaluator, evaluation:

Evaluators use protocol specific algorithms to determine the optimal allocation of goods to winners.

- **Examples:** The auctioneer in the English auction is the only evaluator. The auctioneer sorts the bids by amount bid and picks the highest as the winner. Multiple evaluators in the secure homomorphic protocol work together to solve the auction. Each evaluator solves it's part of the problem and submits it to shared storage, once the threshold number of evaluators have solved their part of the algorithm the set of winners with corresponding goods can be determined.

- **Bind time:** The hotspot is protocol specific, so it is rebound at runtime for each auction instance.

- **Responsibility:** The hotspot must process incoming bids and pass them to the bid storage hotspots.

- **Variability:** The single variable aspect is the algorithm the evaluator uses including communication between other participants and order of operations. This is elementary so the hotspot is also elementary.

- **Multiplicity:** Only one evaluation hotspot implementation can be used per auction instance. The hotspot can not be parametrised as the algorithm will be completely different for different protocols.

Protocol, resource requirements:

Different protocols require different auction resources types, and when establishing the auction an auction owner must what resources are required.

- **Examples:** In an English auction a single auctioneer is required, but in comparison the Polynomial protocol requires an auctioneer, multiple evaluators, mask publishers, and auction publishers.
- **Bind time:** The hotspot is protocol specific, so it is rebound at runtime for each auction instance.
- **Responsibility:** The protocol must provide a list of the resources required and the number of each, for an auction instance.
- **Variability:** The variable aspect is the algorithm required to determine the type and number of required resources.
- **Multiplicity:** A single auction resource requirement hotspot implementation is bound per auction instance.

Protocol, settings:

Protocol settings store the settings for an auction instance.

- **Examples:** The maximum bid in a sealed bid protocol or the number of evaluators in a secure auction.
- **Bind time:** The settings are bound at runtime during auction setup.
- **Responsibility:** The protocol settings store the settings for the auction instance.

- **Variability:** The variable aspect is the different settings which are stored.

- **Multiplicity:** There is only one auction settings used in an auction instance as it uniquely identifies the parameters for the auction. It can be parametrised by using a simple map of settings to protocol values.

Auction participant, communication:

Different applications will require different methods of communication, some may require multiple types of communication within an application.

- **Examples:** With a single application, single process application simple objects can be passed around, casting can be used to transform participants between types. In a RMI (Remote Method Invocation) application, it is not so simple. Interfaces and skeletons are required by both participants, to transform the participant to another type a new interface and skeleton is required.

- **Bind time:** Communication can be bound by the application at implementation or run time, it will not affect the framework.

- **Variability:** The variable aspect is the type of communication. The concern for the framework is that communication is transparent. Protocol developers must not need to consider communication and framework developers should not need to implement any communication specific behaviour.

- **Multiplicity**: Each participant may use multiple types of communication at once and so, multiple communication implementations may be bound.

Auction resource, setup:

Each resource requires setup for an auction instance it will participate in. The auction owner requests the setup behaviour of the auction creator before starting the auction. During setup the auction creator requests the other resources to be used in the instance to also set themselves up.Once a resource is setup, it is ready to perform its part in the auction.

- **Examples:** In an English auction only the auctioneer is used, it only needs to set up storage. In the secure homomorphic auction, there are multiple resources: for instance the auctioneer and evaluators. Both need to setup different types of storage, but when the auctioneer supplies the auction resource to an evaluator, the evaluator sets references to shared boards, and retrieves or generates its encryption keys.

- **Bind time:** As this hotspot is protocol specific and multiple protocols may be operating at one time within an application, the bind time is at run time and it will be rebound for every auction instance.

- **Responsibility:** Setup will be requested once for each auction instance. The resource will use auction instance parameters and the current policies to perform it's behaviour.

- **Variability:** There are several sub-aspects which will change between resources and protocols: the algorithm the resource uses including communication between other participants and the order of operations, setting up any resources (such as storage or external connections) and pre-auction computation). All of these are bound to the protocol and cannot be decomposed without interdependencies, so the hotspot is elementary.

- **Multiplicity:** Only one setup hotspot implementation can be used per auction instance, per resource type. The hotspot can not be parametrised

as the algorithm will be completely different for different resources and protocols.

Auction participant, authentication and authorisation:

This is outside the scope of this thesis and left to future work.

Appendix C

Garbled Circuit Performance Results

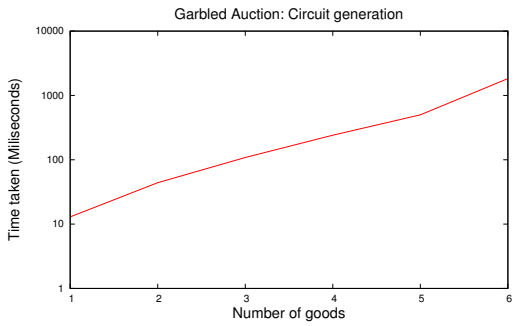

Figure C.1: Effect of increasing the number of goods on circuit generation.

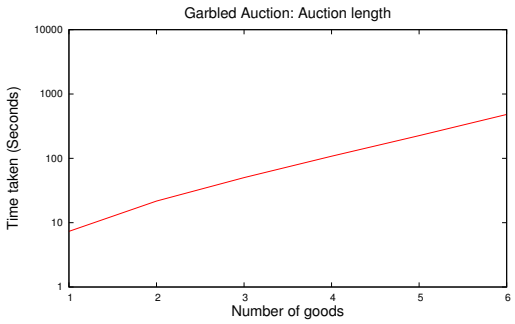

Figure C.2: Effect of increasing the number of goods on auction setup.

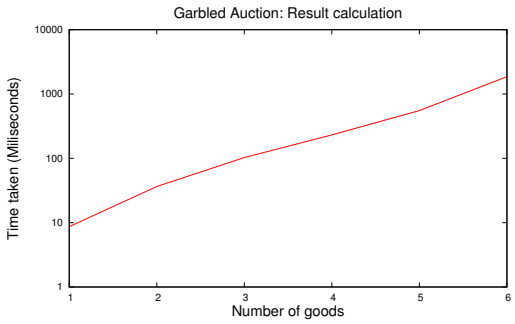

Figure C.3: Effect of increasing the number of goods on evaluation.

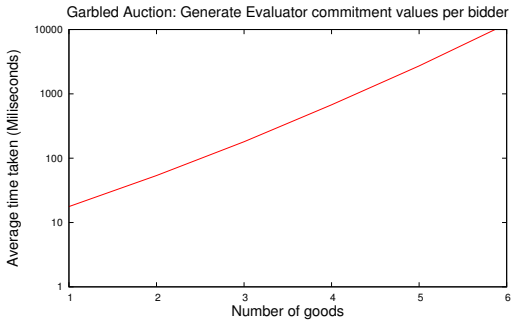

Figure C.4: Effect of increasing the number of goods on evaluator commitment generation.

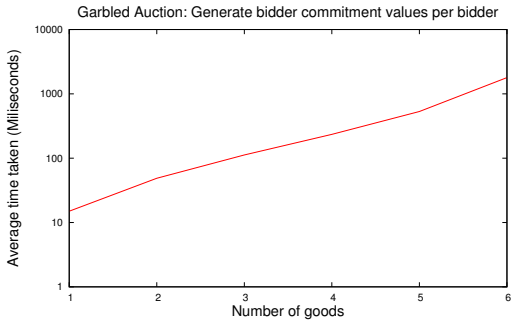

Figure C.5: Effect of increasing the number of goods on bidder commitment generation.

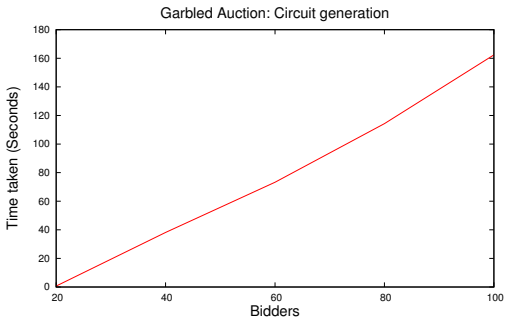

Figure C.6: Effect of increasing the number of bidders on circuit generation.

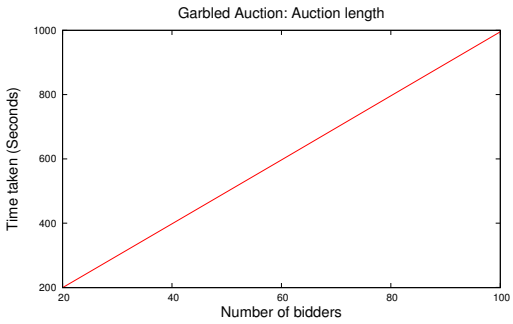

Figure C.7: Effect of increasing the number of bidders on auction setup.

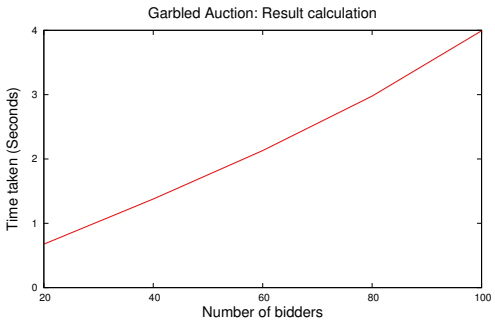

Figure C.8: Effect of increasing the number of bidders on evaluation.

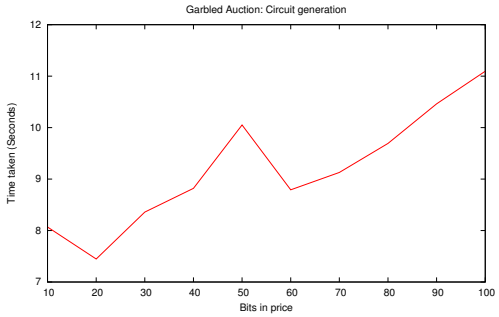

Figure C.9: Effect of increasing the number of bits on circuit generation.

Figure C.10: Effect of increasing the number of bits on setup.

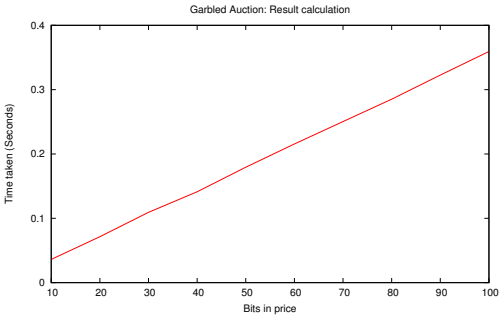

Figure C.11: Effect of increasing the number of bits on evaluation.

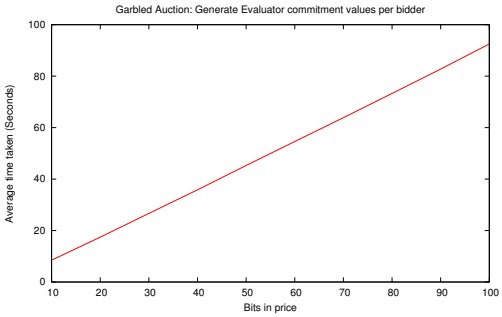

Figure C.12: Effect of increasing the number of bits on evaluator commitment generation.

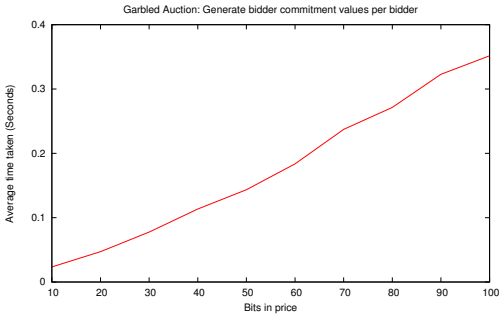

Figure C.13: Effect of increasing the number of bits on bidder commitment generation.

Bibliography

[1] AKSIT, M., MARCELLONI, F., AND TEKINERDOGAN, B. Developing object-oriented framworks using domain models. *ACM Comput. Surv. 32*, 1es (2000), 11.

[2] AKSIT, M., TEKINERDOGAN, B., MARCELLONI, F., , AND BERGMANS, L. *Building Application Frameworks, Object-Oriented Foundations of Framework Design*. Wiley Computer Publishing, 1999, ch. Deriving Frameworks from Domain Knowledge, pp. 169–189.

[3] AUSUBEL, L. M., CRAMTON, P., AND MILGROM, P. *Combinatorial Auctions*. MIT Press, 2006, ch. 5 The Clock-Proxy Auction: A Practical Combinatorial Auction Design, pp. 115–138.

[4] AUSUBEL, L. M., AND MILGROM, P. *Combinatorial Auctions*. MIT Press, 2006, ch. 1 The lovely but lonely Vickrey Auction, pp. 17–40.

[5] AUSUBEL, L. M., AND MILGROM, P. *Combinatorial Auctions*. MIT Press, 2006, ch. 3 Ascending Proxy Auctions, pp. 79–98.

[6] BALL, M. O., DONOHUE, G. L., AND HOFFMAN, K. *Combinatorial Auctions*. MIT Press, 2006, ch. 20 Auctions for the Safe, Efficient, and Equitable Allocation of Airspace System Resources, pp. 507–538.

[7] BOONE, J. *Building Application Frameworks, Object-Oriented Foundations of Framework Design*. Wiley Computer Publishing, 1999, ch. 8 Harvesting Design, pp. 199–210.

[8] BOSCH, J., MOLIN, P., MATTSSON, M., BENGTSSON, P., AND FAYAD, M. E. *Building Application Frameworks, Object-Oriented Foundations of Framework Design*. Wiley Computer Publishing, 1999, ch. Framework Problems and Experiences, pp. 55–82.

[9] BUBENDORFER, K. *NOMAD: Application Participation in a Global Location Service*. PhD thesis, Victoria University of Wellington, 2002.

[10] BUBENDORFER, K., AND HINE, J. H. Auction based resource negotiation in nomad. In *ACSC* (2005), pp. 297–306.

[11] BUBENDORFER, K., KOMISARCZUK, P., AND CHARD, K. Efficient dynamic resource specifications. In *Mobile Data Management* (2005), pp. 281–285.

[12] BUBENDORFER, K., PALMER, B., AND THOMSON, W. *Market Oriented Grid and Utility Computing*. Wiley Computer Publishing, 2008, ch. Trust in Grid Resource Auctions.

[13] BUBENDORFER, K., AND THOMSON, W. Resource management using untrusted auctioneers in a grid economy. In *e-Science* (2006), p. 74.

[14] BUBENDORFER, K., WELCH, I., AND CHARD, B. Trustworthy auctions for grid-style economies. In *CCGRID* (2006), pp. 386–390.

[15] CAPLICE, C., AND SHEFFI, Y. *Combinatorial Auctions*. MIT Press, 2006, ch. 21 Combinatorial Auctions for Truckload Transportation, pp. 539–572.

[16] CHARD, K., KOMISARCZUK, P., BUBENDORFER, K., AND DESAI, A. Fine grained resource reservation and management in grid economics. In *GCA* (2005), pp. 31–38.

[17] CRAMTON, P. *Combinatorial Auctions*. MIT Press, 2006, ch. 4 Simultaneous Ascending Auctions, pp. 99–114.

[18] CRAMTON, P., SHOHAM, Y., AND STEINBERG, R., Eds. *Combinatorial Auctions*. MIT Press, 2006.

[19] DAVID, E., AZOULAY-SCHWARTZ, R., AND KRAUS, S. An english auction protocol for multi-attribute items. In *AMEC* (2002), pp. 52–68.

[20] DOBZINSKI, S., NISAN, N., AND SCHAPIRA, M. Approximation algorithms for combinatorial auctions with complement-free bidders. In *STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing* (2005), ACM, pp. 610–618.

[21] FOSTER, I. T., KESSELMAN, C., AND TUECKE, S. The anatomy of the grid - enabling scalable virtual organizations. *CoRR cs.AR/0103025* (2001).

[22] FUKUTA, N., AND ITO, T. Short-time approximation on combinatorial auctions: a comparison on approximated winner determination algorithms. In *DEECS '07: Proceedings of the 3rd international workshop on Data enginering issues in E-commerce and services* (2007), ACM, pp. 26–33.

[23] GAMMA, E., HELM, R., JOHNSON, R., AND VLISSIDES, J. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., 1995.

[24] HOFFMAN, K., MENON, D., VAN DEN HEEVER, S., AND WILSON, T. *Combinatorial Auctions*. MIT Press, 2006, ch. 17 Observations and Near-Direct Implementations of the Ascending Proxy Auction, pp. 415–450.

[25] JOHNSON, R. E. Documenting frameworks using patterns. In *OOPSLA '92: conference proceedings on Object-oriented programming systems, languages, and applications* (1992), ACM, pp. 63–76.

[26] JOHNSON, R. E. Frameworks = (components + patterns). *Commun. ACM 40*, 10 (1997), 39–42.

[27] JONES, S. R. *Building Application Frameworks, Object-Oriented Foundations of Framework Design*. Wiley Computer Publishing, 1999, ch. 10 A Framework Recipe, pp. 237–266.

[28] KANG, K. C., COHEN, S. G., HESS, J. A., NOVAK, W. E., AND PETERSON, A. S. Feature-oriented domain analysis (foda) feasibility study. Tech. rep., Carnegie-Mellon University Software Engineering Institute, November 1990.

[29] KLEMPERER, P. Auction theory: A guide to the literature. *Journal of Economic Surveys 13*, 3 (1999), 227–86.

[30] KLEMPERER, P. *Auctions: Theory and Practice*. Princeton University Press, 2004.

[31] KOSKIMIES, K., AND MSSENBCK, H. Designing a framework by stepwise generalization. In *Proceedings of the 5th European Software Engineering Conference* (1995), Springer-Verlag, pp. 479–498.

[32] LAND, A., POWELL, S., AND STEINBERG, R. *Combinatorial Auctions*. MIT Press, 2006, ch. 6 PAUSE: A Computationally Tractable Combinatorial Auction., pp. 139–157.

[33] LARSEN, G. Designing component-based frameworks using patterns in the uml. *Commun. ACM 42*, 10 (1999), 38–45.

[34] LEHMANN, D., MÜLLER, R., AND SANDHOLM, T. *Combinatorial Auctions*. MIT Press, 2006, ch. 12 The Winner Determination Problem, pp. 298–317.

[35] LEYTON-BROWN, K., AND SHOHAM, Y. *Combinatorial Auctions*. MIT Press, 2006, ch. 18 A Test Suite for Combinatorial Auctions, pp. 451–478.

[36] MILLER, G. G., MCGREGOR, J., AND MAJOR, M. L. *Building Application Frameworks, Object-Oriented Foundations of Framework Design*. Wiley Computer Publishing, 1999, ch. 13 Capturing Framework Requirements, pp. 310–324.

[37] MÜLLER, R. *Combinatorial Auctions*. MIT Press, 2006, ch. 13 Tractable Cases of the Winner Determination Problem, pp. 319–336.

[38] NAOR, M., PINKAS, B., AND SUMNER, R. Privacy preserving auctions and mechanism design. In *ACM Conference on Electronic Commerce* (1999), pp. 129–139.

[39] NISAN, N. *Combinatorial Auctions*. MIT Press, 2006, ch. 9 Bidding Languages for Combinatorial Auctions, pp. 215–231.

[40] PALMER, B. Verifying privacy preserving combinatorial auctions. Master's thesis, Victoria University of Wellington, 2008.

[41] PARKES, D. C. *i*Bundle: an efficient ascending price bundle auction. In *ACM Conference on Electronic Commerce* (1999), pp. 148–157.

[42] PARKES, D. C. *Combinatorial Auctions*. MIT Press, 2006, ch. 2 Iterative Combinatorial Auctions, pp. 41–77.

[43] PENG, K., BOYD, C., DAWSON, E., AND VISWANANATHAN, K. Five sealed-bid auction models. In *Proceedings of the Australasian information security workshop conference on ACSW frontiers* (2003), vol. 21, Australian Computer Society, pp. 77–86.

[44] PREE, W. *Building Application Frameworks, Object-Oriented Foundations of Framework Design*. Wiley Computer Publishing, 1999, ch. 16 Hot-Spot-Driven Development, pp. 379–392.

[45] RIEHLE, D. Describing and composing patterns using role diagrams. In *Softwaretechniktrends 16, 4 (Dezember 1996)* (1996), pp. 64–80.

[46] RIEHLE, D. A role-based design pattern catalog of atomic and composite patterns structured by pattern purpose. Tech. rep., Ubilab, 1997.

[47] RIEHLE, D., AND GROSS, T. Role model based framework design and integration. In *OOPSLA '98: Proceedings of the 13th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications* (1998), ACM, pp. 117–133.

[48] ROBERTS, D., AND JOHNSON, R. E. Evolving frameworks: A pattern language for developing object-oriented frameworks. In *Pattern Languages of Program Design 3*. Addison Wesley, 1997.

[49] RPING, A. *Building Application Frameworks, Object-Oriented Foundations of Framework Design*. Wiley Computer Publishing, 1999, ch. 14 Managing Class Dependencies, pp. 325–343.

[50] SANDHOLM, T. *Combinatorial Auctions*. MIT Press, 2006, ch. 14 Optimal Winner Determination Algorithms, pp. 337–368.

[51] SANDHOLM, T., AND BOUTILIER, C. *Combinatorial Auctions*. MIT Press, 2006, ch. 10 Preference Elicitation in Combinatoral Auctions, pp. 233–263.

[52] SCHMID, H. A. Creating the architecture of a manufacturing framework by design patterns. In *OOPSLA '95: Proceedings of the tenth annual conference on Object-oriented programming systems, languages, and applications* (1995), ACM, pp. 370–384.

[53] SCHMID, H. A. Systematic framework design by generalization. *Commun. ACM 40*, 10 (1997), 48–51.

[54] SCHMID, H. A. *Building Application Frameworks, Object-Oriented Foundations of Framework Design*. Wiley Computer Publishing, 1999, ch. 15 Framework Design by Systematic Generalization, pp. 353–377.

[55] SCHNEIER, B. *Applied cryptography (2nd ed.): protocols, algorithms, and source code in C*. John Wiley & Sons, Inc., 1995.

[56] SHAMIR, A. How to share a secret. *Commun. ACM 22*, 11 (1979), 612–613.

[57] SHAW, M., AND GARLAN, D. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.

[58] SHOUP, V. Practical threshold signatures. In *EUROCRYPT* (2000), pp. 207–220.

[59] SILVA, A. R., ROSA, F. A., AND GONÇALVES, T. Framework description using concern-specific design patterns composition. *ACM Comput. Surv. 32*, 1es (2000), 16.

[60] SUCCI, G., VALERIO, A., VERNAZZA, T., FENAROLI, M., AND PREDONZANI, P. Framework extraction with domain analysis. *ACM Comput. Surv. 32*, 1es (2000), 12.

[61] SUN MICROSYSTEMS. Remote method invocation (rmi). `http://java.sun.com/javase/technologies/core/basic/rmi/`.

[62] SUZUKI, K., AND YOKOO, M. Secure combinatorial auctions by dynamic programming with polynomial secret sharing. In *Financial Cryptography* (2002), pp. 44–56.

[63] SUZUKI, K., AND YOKOO, M. Secure generalized vickrey auction using homomorphic encryption. In *Financial Cryptography, 7th International Conference* (2003), vol. 2742, Springer, pp. 239–249.

[64] VALERIO, A., SUCCI, G., AND FENAROLI, M. Domain analysis and framework-based software development. *SIGAPP Appl. Comput. Rev. 5*, 2 (1997), 4–15.

[65] WURMAN, P. R., WELLMAN, M. P., AND WALSH, W. E. Specifying rules for electronic auctions. *AI Magazine 23*, 3 (2002), 15–24.

[66] YAO, A. How to generate and exchange secrets. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science* (1986), pp. 162–167.

[67] YOKOO, M. *Combinatorial Auctions*. MIT Press, 2006, ch. 7 Pseudonymous Bidding in Combinatorial Auctions, pp. 163–187.

[68] YOKOO, M., AND SUZUKI, K. Secure multi-agent dynamic programming based on homomorphic encryption and its application to combinatorial auctions. In *AAMAS* (2002), pp. 112–119.

[69] YOKOO, M., AND SUZUKI, K. Secure generalized vickrey auction without third-party servers. In *Financial Cryptography* (2004), pp. 132–146.