# Secure Multiparty Computation Goes Live*

Peter Bogetoft[§], Dan Lund Christensen[¶], Ivan Damgård[‡], Martin Geisler[‡], Thomas Jakobsen[¶], Mikkel Krøigaard [‡], Janus Dam Nielsen[‡], Jesper Buus Nielsen[‡], Kurt Nielsen[†], Jakob Pagter[¶], Michael Schwartzbach[‡], and Tomas Toft[††]

[†] Inst. of Food and Resource Economics, University of Copenhagen
[‡] Department of Computer Science, University of Aarhus
§Dept. of Economics, Copenhagen Business School
¶The Alexandra Institute
†† CWI Amsterdam and TU/e

**Abstract.** In this note, we report on the first large-scale and practical application of multiparty computation, which took place in January 2008. We also report on the novel cryptographic protocols that were used.

## 1 Introduction and History

In multiparty computation (MPC), we consider a number of players $P_1, \ldots, P_n$, who initially each hold inputs $x_1, \ldots, x_n$, and we then want to securely compute some function $f$ on these inputs, where $f(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$, such that $P_i$ learns $y_i$ but no other information. This should hold, even if players exhibit some amount of adversarial behavior. The goal can be accomplished by an interactive protocol $\pi$ that the players execute. Intuitively, we want that executing $\pi$ is equivalent to having a trusted party $T$ that receives privately $x_i$ from $P_i$, computes the function, and returns $y_i$ to each $P_i$ [1]. With such a protocol we can - in principle - solve virtually any cryptographic protocol problem. The general theory of MPC was founded in the late 80-ties [16, 3, 7]. The theory was later developed in several ways – see for instance [21, 18, 8]. An overview of the theoretical results known can be found in [6].

Despite the obvious potential that MPC has in solving a wide range of problems, we have seen virtually no practical applications of MPC in the past. This is probably in part due to the fact that direct implementation of the first general protocols would lead to very inefficient solutions. Another factor has been a general lack of understanding in the general public of the potential of the technology. A lot of research has gone into solving the efficiency problems, both for general protocols [11, 17, 9] and for special types of computations such as voting [4, 12].

A different line of research has had explicit focus on a range of economic applications, which are particularly interesting for practical use. This approach was taken, for instance, by two research projects that the authors of this paper have been involved in: SCET (Secure Computing, Economy and Trust)[2] and SIMAP (Secure Information Management and Processing),[3], which has been responsible for the practical application of MPC described in this paper. In the economic field of mechanism design the concept of a trusted third party has been a central assumption since the 70's [15, 19, 10]. Ever since the field was initiated it has grown in momentum and turned into a truly cross disciplinary field. Today, many practical mechanisms require a trusted third party and it is natural to consider the possibility of implementing such a party using MPC. In particular, we have considered:

- Various types of auctions that involves sealed bids for different reasons. The most well-known is probably the standard highest bid auction with sealed bids, however, in terms of turnover another common variant is the so called double auction with many sellers and buyers. This auction handles scenarios where one wants to find a fair market price for a commodity given the existing supply and demand in the market.

* This work was sponsored by the Danish Strategic Research Council.

[1] This "equivalence" can be formalized using, for instance, Canetti's Universal Composability framework[5].

[2] see http://sikkerhed.alexandra.dk/uk/projects/scet

[3] see http://sikkerhed.alexandra.dk/uk/projects/simap

– Benchmarking, where several companies want to combine information on how their businesses are running, in order to compare themselves to best practice in the area. The benchmarking process is either used for learning, planning or motivation purposes. This of course has to be done while preserving confidentiality of companies' private data.

When looking at such applications, one finds that the computation needed is basically elementary arithmetic on integers of moderate size, say around 32 bits. More concretely, quite a wide range of the cases require only addition, multiplication and comparison of integers. As far as addition and multiplication is concerned, this can be handled quite efficiently by well-known generic MPC protocols. What they really do is actually operations modulo some prime $p$, because the protocols are based on secret sharing over $\mathbb{Z}_p$. But by choosing $p$ large enough compared to the input numbers, we can avoid modular reductions and get integer addition and multiplication.

This is efficient because each number is shared "in one piece" using a linear secret sharing scheme, so that secure addition, for instance, requires only one local addition by each player. Unfortunately, this also implies that comparison is much harder. A generic solution would express the comparison operation as an arithmetic circuit over $Z_p$, but this would be far too large to give a practical solution, because the circuit would not have access to the binary representation of the inputs. So instead one must develop special purpose techniques for comparison. One example of this is the constant-round and unconditionally secure comparison protocol from [13].

## 2 Contributions of this Paper

In this paper, we report on the first large-scale practical experiment with using MPC to implement a secure auction. We explain the application scenario and the reasons why multiparty computation turned out to be a good solution. We describe the system that was implemented and report on how it performed. Finally, we explain in detail the cryptographic protocols used and prove their security. In doing so, we propose a logarithmic round comparison protocol that is much more practical than the one from [13] for numbers of realistic size.

## 3 The Application Scenario

In this section we describe the practical case in which our system has been deployed. In [1], preliminary plans for this scenario and results from a small-scale demo were described.

In Denmark, several thousand farmers produce sugar beets, which are sold to the company Danisco, the only sugar beets processor on the Danish market. Farmers have contracts that give them rights and obligation to deliver a certain amount of beets to Danisco, who pay them according to a pricing scheme that is an integrated part of the contracts. These contracts can be traded between farmers, but trading has historically been very limited and has primarily been done via bilateral negotiations.

In recent years, however, the EU drastically reduced the support for sugar beet production. This and other factors meant that there was now an urgent need to reallocate contracts to farmers where productions pays off best. It was realized that this was best done via a nation-wide exchange, a double auction.

*Market Clearing Price* Details of the particular business case can be found in [2]. Here, we briefly summarize the main points while more details on the actual computation to be done are given later. The goal is to find the so called *market clearing price* (*MCP*), which is a price per unit of the commodity that is traded. What happens is that each buyer specifies, for each potential price, how much he is willing to buy at that price, similarly sellers say how much they are willing to sell at each price.[4] All bids go to an auctioneer, who computes, for each price, the total supply and demand in the market. Since we can assume that supply grows and demand decreases with increasing price, there

[4] In real life, a bidder would only specify a small number of prices, namely those where the quantity he wants to trade changes, and by how much. The quantities to trade at other prices then follow from this.

is a price where total supply equals total demand, and this is the price we are looking for. Finally, all bidders who specified a non-zero amount to trade at the market clearing price get to sell/buy the amount at this price.

*Privacy of Bids using Secure MPC* A satisfactory implementation of such an auction has to take some security concerns into account: Bids clearly reveal information, e.g., on a farmer's economic position and his productivity, and therefore farmers would be reluctant to accept Danisco acting as auctioneer, given its position in the market. Even if Danisco would never misuse its knowledge of the bids in the ongoing renegotiations of the contracts (including the pricing scheme), the mere fear of this happening could affect the way farmers bid and lead to a suboptimal result of the auction. On the other hand, the entitled quantities in a given contract are administrated by Danisco (and adjusted frequently according to the EU administration) and in some cases the contracts act as security for debt that farmers have to Danisco. Hence running the auction independently of Danisco is not acceptable either. Finally, the solution of delegating the legal and practical responsibility by paying e.g. a consultancy house to be the trusted auctioneer would have been a very expensive solution.

The solution decided on was to implement an electronic double auction, where the role of the auctioneer would be played by a three-party multiparty computation done by representatives for Danisco, DKS and the SIMAP project. A three party solution was selected, partly because it was natural in the given scenario, but also because it allowed using efficient information theoretic tools such as secret sharing, rather than (much) more expensive cryptographic methods such as homomorphic encryption.

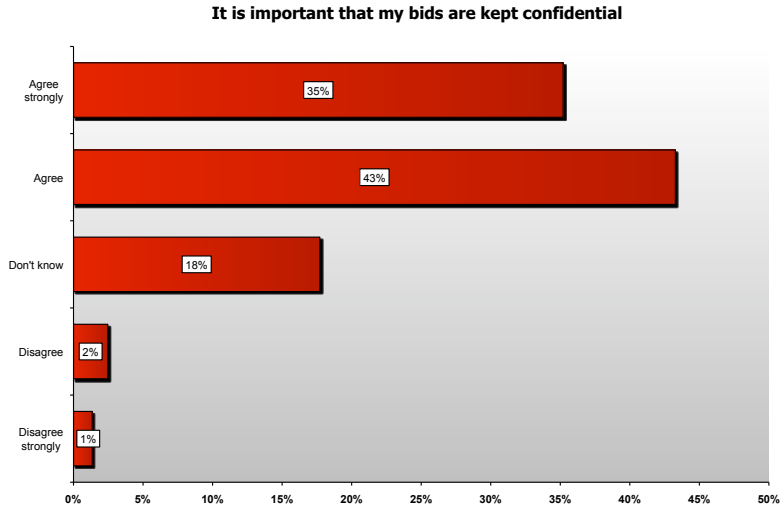


**Fig. 1.** Farmer's confidentiality expectations. (Numbers from survey based on questions asked to the farmers after they had submitted their bids.)

*Motivation* It is interesting to ask what motivated DKS and Danisco to try using such a new and untested technology? One important factor was simply the obvious need for a nation-wide exchange for production rights, which had not existed before, so the opportunity to have a cheap electronic solution –secure or not– was certainly a major reason. We do believe, however, that security also played a role. An on-line survey carried out in connection with the auction showed that farmers do care about keeping their bids private (see tables in Fig. 1 and Fig. 2). And if Danisco and DKS would have tried to run the auction using conventional methods, one or more persons would have had to have access to the bids, or control over the system holding the bids in cleartext. As a result, some security policy would have had to be agreed, answering questions such as: who should have access to the data and when? who has responsibility if data leaks, and what are the consequences?

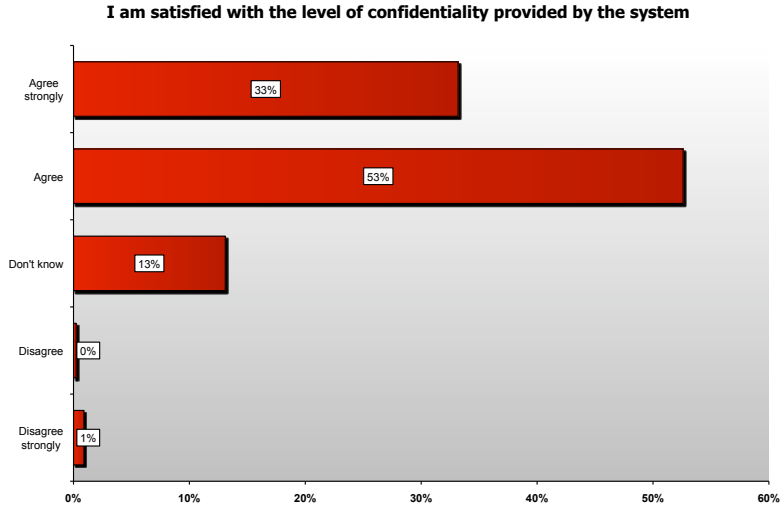


**Fig. 2.** Farmer's satisfaction with the provided level of confidentiality. (Numbers from survey based on questions asked to the farmers after they had submitted their bids.)

Since the parties have conflicting interests, this could have lead to very lengthy discussions, possibly bringing the whole project to a halt. In an interview with the involved decision markers from Danisco and DKS (the sugar beet growers association) these confidentiality issues were well recognized. Using a consultancy house as mediator would have been more expensive as mentioned, and the parties would still have had to agree on whether the mediator's security policy was satisfactory. As it happened, there was no need for this kind of negotiations, since the multiparty computation ensured that no one needed to have access to bids at any point [5].

Note that multiparty computation with security in the semi-honest model already achieves the property that one can "choose not to know", and we therefore decided that a solution with semi-honest security for implementing the auctioneer would be sufficient. Note, however, that active attacks from bidders participating in the auction could still be a valid concern. Although we estimated that this was not a significant risk in our particular case, we show below protocols that protect against malicious bidders.

One might also ask if the full power of multiparty computation was actually needed? Our solution ensures that no single player has any sensitive information, and it might seem that one could solve the problem more efficiently in a similar trust model using a trick often used in voting protocols: one party $P_1$ receives the bids in encrypted form from the bidders, however, the bids are encrypted with the public key of another party $P_2$. Then $P_1$ sends the encryptions, randomized and in permuted order to $P_2$ who decrypts the bids and computes the market clearing price. While this achieves some security because P2 does not know who placed which bids, we have to remember that bids contain much more information than what is conveyed by the result (the market clearing price), e.g., one can see the quantities people were willing to buy or sell at other prices than the clearing price. In principle, this type of information is highly valuable for a monopsonist as Danisco in order to exercise its market power, e.g., in terms of setting the price of an extension or a reduction of the total processing capacity. To what extend such a situation is relevant in practice is not easy to answer. Our conclusion was that using full-blown multiparty computation is a better solution because it frees us from even having to consider the question.

[5] In an interview with the decision makers, they recognized that this fact made it easy to communicate the security policy to the farmers.

## 4 The Cryptographic Protocols

Abstracting away some less relevant details, the scenario we have is the following: Some input clients $I_1, \ldots, I_m$ deliver inputs to a multiparty computation, that is to be executed by servers $P_1, \ldots, P_n$. In the types of cases we are interested in, $m$ is very large and variable while we think of $n$ as a small constant. In our concrete case, we had $n = 3$ and $m$ was about 1200.

The input from client $I_i$ is an ordered list of non-negative integers $\{x_{ij} |\ j = 1, \ldots, P\}$, where index $j$ refers to one of the $P$ possible prices per unit, in increasing order. Such a list is called a *bid*. A bid can be a *sell* bid in which case the list is non-decreasing, or a *buy* bid in which case it is non-increasing. For a buy bid, $x_{ij}$ is the quantity the bidder wants to buy at the $i$'th price per unit, similarly for sell bids, the elements of which we will denote by $y_{ij}$. Due to the practical constraints it must be possible to deliver these inputs non-interactively (and securely) to the servers.

The secure computation consists of computing the total demand and supply at each price, namely

$$d_j = \sum_i x_{ij}, \quad s_j = \sum_i y_{ij}, \quad j = 1, \ldots, P \ ,$$

and to finally find the index $j_0$ for which $d_{j_0} - s_{j_0} = 0$, or rather an index where the difference is as close to 0 as possible. Since quantities are specified in units of fixed size, we cannot expect to find a price where supply exactly meets demand. This also means that there has to be agreed rules for how one handles cases where we must live with a price where supply is larger than demand or vice versa. Such rules were agreed for our concrete case, but the details of this are outside the scope of this paper.

In any case, since supply increases and demand decreases with increasing price, we can find the index we are looking for by binary search over the indices $1, \ldots, P$: We start by comparing $d_{P/2}$ to $s_{P/2}$. If the result is that $d_{P/2}$ was larger, then $j_0 \geq P/2$, else $j_0 < P/2$. Depending on the result, we do a similar comparison in the middle of the top or bottom half of the interval. Continuing in this way, we can find $j_0$ using secure comparisons between $d_j$ and $s_j$ for $\log_2 P$ values of $j$.

Note that it is secure to make the comparison results public: we want $j_0$ to be public anyway, and from this, the result of the comparison between $d_j$ and $s_j$ already follow for any $j$. Finally $j_0$ is made public, as well as $x_{ij_0}, y_{ij_0}$ for all $i$, i.e., the quantity each bidder said he would buy or sell at the market clearing price.

It will therefore be sufficient to design a protocol that (in the given scenario) implements the ideal functionality in Fig. 3.

**Functionality $\mathcal{F}$:**

1. On input $\texttt{Input}(x_1, \ldots, x_P)$ from an input client $I_j$, where $x_1, \ldots, x_P$ is a list of integers where each number is at most $\ell$ bits long, for some fixed $\ell$, and where the list is either increasing or decreasing. The ideal functionality $\mathcal{F}$ stores the numbers in uniquely named registers and notifies all players and the adversary that an input list has been received from $I_j$ along with the names of the registers in which the numbers are stored.
2. On input $C = A + B$, where $A, B, C$ are names of registers of $\mathcal{F}$, $\mathcal{F}$ adds the numbers in $A$ and $B$ and stores the result in $C$.
3. On input $C = A \times B$ where $A, B, C$ are names of registers of $\mathcal{F}$, $\mathcal{F}$ multiplies the numbers in $A$ and $B$ and stores the result in $C$.
4. On input $\texttt{ConstantMult}(a, B)$ where $a \in \mathbb{Z}_p$ and $B$ is a register, $\mathcal{F}$ multiplies the number in $B$ by $a$ and stores the result in $B$.
5. On input $\texttt{Compare}(A, B)$, $\mathcal{F}$ sends 1 to all servers if the number in $A$ is larger than the number in $B$ and 0 otherwise.
6. On input $\texttt{Open}(A)$, $\mathcal{F}$ sends the number stored in register $A$ to all servers.
7. On input $\texttt{RandomBit}(A)$, $\mathcal{F}$ chooses a random 0/1 value and places it in register $A$.

**Fig. 3.** The ideal functionality $\mathcal{F}$ implemented by our protocols.

We will assume a static and passive adversary who may corrupt any number of input clients and any minority of the servers. We show below that we can allow active corruption of the clients at the expense of some efficiency. In our concrete case, however, we have estimated that the risk of active attacks from clients was too small to motivate paying the loss in efficiency – see more details below. We assume secure point-to-point channels between the servers, this can be implemented with standard tools.

Our implementation will be based on standard Shamir secret sharing among the $n$ servers, using a prime field $\mathbb{Z}_p$ where $p$ is chosen such that its bit length is $\ell + \kappa$, where $\kappa$ is a parameter that controls the statistical security of the comparison protocol. In our concrete case $\ell$ was 32 and $p$ was 65 bits long).

We set $t = \lfloor (n-1)/2 \rfloor$, so a number is secret shared by choosing a random polynomial $f$ of degree at most $t$ with $f(0) = x$, and the shares of $x$ are then $f(1), \ldots, f(n)$. By $[x]$ we denote a set of shares of the number $x$, suppressing for readability the random coins used in the sharing.

Let $\mathcal{F}'$ be the functionality that is the same as $\mathcal{F}$, but does not have the comparison command. In the following we will first describe how to implement $\mathcal{F}'$, and then show how to implement $\mathcal{F}$ based on $\mathcal{F}'$.

**Setting up Public Keys** Our implementation assumes that public/secret key pairs have been set up by the servers before the computation starts, and that the public keys are available to the clients. More precisely, for every maximal unqualified set $A$ of servers (i.e., $|A| = t$), we need that all servers not in $A$ have a secret key $sk_A$, and the public key $pk_A$ is available to all players (input clients $I_j$ and servers $P_i$). This can be accomplished in our scenario by having one server in the complement of $A$ generate $pk_A, sk_A$, send $sk_A$ to all servers not in $A$ and $pk_A$ to all players.

**Non-Interactive Input** The first issue is now how to implement the command where a client inputs numbers $x_1, \ldots, x_P$. The naive solution of simply secret sharing each $x_i$ and encrypt each share under the corresponding server's public key has the problem that it would expand the data a client needs to send by a multiplicative factor of at least the number of servers.

Instead, we propose a variant of a non-interactive VSS technique from [14]. We describe it here for simplicity in our concrete case where $n = 3$. In this case the key set-up above becomes the following: we need 3 key pairs $(pk_i, sk_i), i = 1, 2, 3$, and server $i$ has the two keys $sk_j$ where $j \neq i$. Now let $f_i(x), i = 1, 2, 3$ denote polynomials of degree at most 1 satisfying that $f_i(0) = 1, f_i(i) = 0$. One can now communicate a list of numbers $x_1, \ldots, x_P$ in $\mathbb{Z}_p$ to the servers in encrypted form as follows:

1. Choose keys $K_1, K_2, K_3$ for a pseudorandom function (PRF) $F$ that takes an index $j$ as input and produces output in $\mathbb{Z}_p$.[6]
2. Output encryptions $E_{pk_i}(K_i), i = 1, 2, 3$.
3. For $j = 1, \ldots, P$, compute and output

$$y_j = F_{K_1}(j) + F_{K_2}(j) + F_{K_3}(j) + x_j \bmod p \ .$$

Each server $P_a$ can now process such an encryption and compute a Shamir share of each number:

1. Decrypt the two ciphertexts $E_{pk_i}(K_i)$ where $i \neq a$.
2. Compute your share $share_{a,j}$ of $x_j$ as follows: $share_{a,j} =$

$$y_j - F_{K_1}(j) f_1(a) - F_{K_2}(j) f_2(a) - F_{K_3}(j) f_3(a)$$

bearing in mind that since $f_a(a) = 0$, it does not matter that you don't know $K_a$.

It is straightforward to see that if we define the polynomial $g_j$ as $g_j = y_j - F_{K_1}(j) f_1 - F_{K_2}(j) f_2 - F_{K_3}(j) f_3$, then indeed $\deg(g) \leq 1$, $g_j(0) = x_j$ and $g_j(a) = share_{a,j}$ so that a valid set of shares has indeed been computed.

[6] One can e.g. use a PRF $F'$ with output in $\{0,1\}^{\lceil \log_2(p) \rceil + \kappa}$, interpret the output as a number $y \in \{0, \ldots, 2^{\lceil \log_2(p) \rceil + \kappa} - 1\}$ and let $F(x) = F'(x) \bmod p$.

Generalizing this to an arbitrary number of servers and Shamir sharing with threshold $t$ is straightforward: we use the general key set-up above with a key pair $(pk_A, sk_A)$ for every set of servers of size $t$, and $sk_A$ is given to all servers not in $A$. We then use the polynomials $f_A$ of degree at most $t$ where $f_A(0) = 1$ and $f_A(i) = 0$ for all $i \in A$. Of course, this does not scale well to large $n$, but we will not need this in our application.

This method has a number of advantages:

1. Except for an additive overhead depending on the number of servers, the encrypted list is the same size as the list itself.
2. Assuming the decryption algorithm of the public key system is deterministic, the decryption process always results in consistent shares of *some* list of values.
3. If a server loses its secret keys, they can be reconstructed with help from the other servers.
4. We only need communication from clients to servers. This is very convenient in a practical setting where we can control the configuration of the (relatively few) servers, but not the (many) clients — some might e.g. sit behind a firewall making it hard to send data from the servers to the clients.

**Addition and Multiplication** After the input phase, all values are shared using polynomials of degree $\leq t$. We thus can implement addition and multiplication using well known standard protocols and assuming as invariant that all numbers that $\mathcal{F}$ would store in a register are in the real protocol secret shared among the players. The addition command on input $[a], [b]$ is done by having servers locally add their shares of $a$ and $b$, clearly $[a] + [b] = [a + b]$ since the sharing is linear. Likewise, multiplication by a constant is done by having each server multiply his share by the public constant. Multiplication is done by having server $P_i$ multiply his shares of $a, b$: $d_i = a_i b_i$. He then forms shares $[d_i]$ and sends them to the servers. Finally, all servers compute $[ab] = \sum_i \lambda_i [d_i]$, where the $\lambda_i$ are Lagrange interpolation coefficients that are chosen to reconstructing $g(0)$ from $g(1), \ldots, g(n)$ for a polynomial $g$ of degree $\leq 2t$. Since $2t < n$ it is possible to compute such $\lambda_i$.

**Random Bits** For the `RandomBit`, we borrow a trick from [13]: All servers secret share a random value, and add all shares locally, to form a sharing $[u]$ of a random unknown $u$. We then compute $[v] = [u^2 \bmod p]$ and open $v$. If $v = 0$ we start over, otherwise we publicly compute a square root $w$ of $v$, say we choose the smallest one. We compute $w^{-1}[u] \bmod p$ which will be 1 with probability $1/2$ and $-1$ with probability $1/2$. Therefore, $[(w^{-1}u + 1)2^{-1} \bmod p]$ will produce the random shared binary value we wanted.

**Lemma 1.** *If the encryption used is semantically secure and the PRF used is secure, then the above protocol implements $\mathcal{F}'$ securely against a static, passive adversary corrupting any number of clients and at most $t$ servers.*

*Proof.* We must provide a simulator that can, by only interacting with $\mathcal{F}'$, on behalf of the corrupted parties, simulate any adversary's view of the real life protocol. The simulator first generates key pairs $(pk_A, sk_A)$ as described above, sends the public keys to the adversary as well as those secrets that are to be known by corrupt players.

We first show how to simulate the input operation. If the client sending input is corrupt then since he follows the protocol by assumption, the simulator can compute the input that is encrypted by monitoring the computing done by the client. The simulator sends these inputs to $\mathcal{F}'$. When $\mathcal{F}'$ says that inputs were received from an honest input client, the simulator generates an encrypted list of input numbers following the protocol, using 0 for all input numbers. It sends this as the simulated message from the client. The other commands are simulated in the standard way: when an honest server secret shares a value, the simulator generates (up to) $t$ uniform field elements to simulate shares of corrupt players. When a sharing is opened, the simulator is given the value to open by $\mathcal{F}'$ and it completes the set of shares already known to the adversary to a complete set consistent with the value to open.

To argue that the simulation of the input command is indistinguishable from the real protocol, we note that it is clearly perfect for the case of a corrupt client, as we run $\mathcal{F}'$ on the input shared

by the corrupted client. For simulation of an honest client, assume some set of $t$ servers $A$ is corrupt, let REAL denote the view of these corrupted parties in the real protocol, let SIM denote their view in the simulation, and consider a variant of the real process $\text{HYB}_1$ where all encryptions under $pk_A$ are replaced by encryptions of zero. Likewise, we construct $\text{HYB}_2$ by replacing in the simulation all encryptions under $pk_A$ by encryptions of zero. Assuming semantic security, $\text{HYB}_1$ is computationally indistinguishable from the real process, and $\text{HYB}_2$ is computationally indistinguishable from the simulation. In proving this we use, of course, that the adversary does not know $sk_A$. Namely, if there exists an environment and adversary for which one could distinguish, we could break semantic security: We get $pk_A$ from an oracle as well as encryptions that are either encryptions of zeros or encryptions of the messages that would normally be used. A successful distinguisher now breaks semantic security. In both $\text{HYB}_1$ and $\text{HYB}_2$, the use to the PRF can be replaced by oracle access to the function without changing anything. We can then form two new processes $\text{HYB}'_1, \text{HYB}'_2$ by replacing the PRF oracle by a random oracle. This leads to indistinguishable processes by security of the PRF. Finally note that $\text{HYB}'_1 = \text{HYB}'_2$ because the only part that may now depend on the input is the number $y_j$. But this is in one case $x_j + r \bmod p$ where $r$ is uniform in $\mathbb{Z}_p$ and independent of $x_j$ and in the other case $0 + r \bmod p$. This gives, of course the same distribution, so our conclusion now follows from transitivity of indistinguishability.

Finally, the simulation of the commands other than input is perfect by standard arguments.

In the protocol above, we have assumed that the numbers in bids have the correct form, in particular they are significantly smaller than $p$. Assuming only passive attacks, one does not have to check for this, but one may still ask if we could protect efficiently against malicious clients?

**Input without Trusting the Clients** The method described above produces consistently shared numbers no matter what the client does, but in principle allows a client to send numbers that are too large, possibly causing the computation to fail. We can protect against this as well, namely we would fix the size of the pseudorandom values $F_{K_i}(j)$ to be $\ell + \kappa$ bits, choose the length of $p$ to be $2(\ell + \kappa + \log T)$ bits where $T$ is the number of maximal unqualified sets $A$, and otherwise do the same protocol as above to send inputs.

Each $y_j$ in the message sent by the client should be a sum of $T$ pseudorandom values and the actual secret to be shared. By choice of the size of $p$, this sum will not involve any reduction modulo $p$, if $y_j$ is correctly constructed. So we can demand that each $y_j$ is at most a $\kappa + \ell + \log T$ bit number and reject the input otherwise. Even if a $y_j$ is not correctly constructed, this guarantees that the secret we end up getting shares of will be of form $y_j - \sum_A F_{K_A}(j)$, and must therefore be numerically much smaller than $p$, in fact it must be in the interval $[-2^{\kappa+\ell+\log T}..2^{\kappa+\ell+\log T}]$. One can easily see that once we know such a constraint on the numbers we work with, the comparison protocol we show later can be used, indeed the only assumption it makes is that the numbers to compare are sufficiently smaller than $p$. The servers can therefore check that the input numbers are positive and increasing or decreasing as required.

Finally, the public-key encryption used must be chosen ciphertext secure in order to cope with malicious input clients, and each plaintext encrypted must include an identification of the intended receiver.

Changing the protocol as described here costs us an increase in size of $p$ which implies a general loss of efficiency, an increase in size of data, and extra work to check the form of bids. On the other hand, to actually cheat, a bidder would have to write his own client program and convince the server side that the normal client was still used. For our concrete case, we estimated that the risk of bidders cheating in this way was too small to motivate the extra cost of protecting against it.

As an aside, we note that it can be shown that sending bids that are not increasing or decreasing cannot be to a bidders advantage and so this is in any case a minor concern.

### 4.1 Adding Secure Comparison

It remains to describe how to compare numbers securely. We show how to do this assuming access to the functionality $\mathcal{F}'$. Then this, the results from the previous section and the UC composition

theorem gives us the desired implementation of $\mathcal{F}$. Recall that numbers to compare are assumed to be of length at most $\ell$ bits, and the prime used for secret sharing is $\ell + \kappa$ bits long.

In the description of the protocol below, we refer to arithmetic on objects written as $[d]$. In this protocol, where we assume access to $\mathcal{F}'$, this should be understood as referring to a register held by $\mathcal{F}'$, containing the number $d$. In the actual implementation $[d]$ would be a secret-sharing of $d$.

We will need an operator on bit-pairs, $\diamond$, defined as

$$\binom{x}{X} \diamond \binom{y}{Y} = \binom{x \wedge y}{x \wedge (X \oplus Y) \oplus X},$$

where $\wedge$ denotes the Boolean AND operator. Note that if we have $[a], [b]$, where $a, b$ are guaranteed to be 0/1 values, then $[a \oplus b]$ can be computed using operations from $\mathcal{F}'$, as $[a] + [b] - 2[ab]$. So we can assume that $\oplus$ on binary values is available, as if it was an operation implemented in $\mathcal{F}'$, and so $\diamond$ can also be implemented. It is easy to verify that $\diamond$ is associative.

**Comparison protocol:**
Input: $[d], [s]$. Output: 1 if $d \geq s$, 0 otherwise

1. For $i = 0, \ldots, \ell + \kappa + 1$, call `RandomBit` to generate $[r_i]$ for random $r_i \in \{0, 1\}$. Compute $[r] = \sum_i 2^i [r_i]$.
2. Compute $[a] = 2^{\ell+\kappa+1} - [r] + 2^\ell + [d] - [s]$. Open $a$, and compute the bits $a_i$ of $a$.
3. Our goal is now to compute the $\ell$'th bit of $a + r = 2^{\ell+\kappa+1} + 2^\ell + d - s$. Note that we have $a$ and $[r_i]$'s available. Compute

$$\binom{[z]}{[Z]} = \binom{[a_{\ell-1} \oplus r_{\ell-1}]}{[a_{\ell-1}]} \diamond \cdots \diamond \binom{[a_0 \oplus r_0]}{[a_0]} \diamond \binom{0}{0}.$$

   Now $Z$ is the carry bit at position $\ell$ when doing the addition $a + r$.
4. Compute $[res] = a_\ell \oplus [r_\ell] \oplus [Z]$, open and output $res$.

**Fig. 4.** The comparison protocol implementing the command `Compare` given $\mathcal{F}'$.

The comparison protocol is given in Fig. 4. Some intuition on the protocol: when comparing values $d$ and $s$, it is easy to see that the comparison result follows from the $\ell$'th bit of $2^\ell + d - s$ (counting the bits from zero). This bit is extracted in two steps: First the problem is transformed to one where the binary representation of involved numbers is available. This transformed instance can then be solved easily.

**Lemma 2.** *When given access to functionality $\mathcal{F}'$, the above comparison protocol implements the comparison operation with statistical security in $O(\log \ell)$ rounds.*

*Proof.* Once we note that none of the additions or subtractions we do can cause reductions modulo $p$ because of the size for $p$ that we have chosen, it should be straightforward that the protocol outputs the correct result, if indeed $Z$ is the $\ell$'th carry bit from the addition of $a$ and $r$, as claimed. To see this, note that the computation of carry-bits can be perceived as follows. If $a_i \neq r_i$, then the present carry-bit $c_i$ is propagated on up, $c_{i+1} = c_i$. However, if $a_i = r_i$, then the next carry-bit is set to their value, $c_{i+1} = a_i = r_i$. The goal is therefore to determine the value of $a_i$ at the most significant (left most) bit-position $i < \ell$ where $a_i = r_i$. Now, looking at the definition of $\diamond$, one can verify that it outputs the $y$-pair when $x = 1$, otherwise the $x$-pair is output, and hence $Z$ indeed ends up being the desired carry bit. Note that the $\binom{0}{0}$ on the right is added to handle the case where $a_i \neq r_i$ for all $i$.

As for the round complexity, note that since $\diamond$ is associative, the expression in Step 3 can be evaluated in a standard tree-like fashion which will take $O(\log \ell)$ rounds since there are $\ell + 1$ operands and the $\diamond$ operation executes in constant-round.

Finally, note that an execution of the protocol can be simulated by choosing a uniform $\kappa + \ell + 2$ bit number $r$ and outputting $2^{\kappa+\ell+1} + 2^{\ell} - r$ to play the role of $a$. Note that this is the only actual communication in the protocol since everything else happens internally in $\mathcal{F}'$. Since $d, s$ are only $\ell$-bit numbers this simulation has statistical distance $2^{-\kappa}$ from the real distribution of $a$.

In [13] a (more complicated) constant-round comparison was proposed. However, our solution is much more practical for the size of numbers in question: The diamond operator executes in three rounds, so only $3\log(32) = 15$ rounds are required for its repeated application. This implies less than 20 rounds overall. In comparison, the solution in [13] requires more than 100 rounds, and though more efficient constant-rounds solutions have been proposed, these are nowhere near as efficient as the present for the input sizes in question.

Lemmas 2 and 1 and the UC composition theorem now immediately imply

**Theorem 1.** *If the encryption used is semantically secure and the PRF used is secure, then the protocol for implementing $\mathcal{F}'$ together with the comparison protocol securely implement $\mathcal{F}$ against a static, passive adversary corrupting any number of clients and at most $t$ servers.*

**A Trick to Improve Efficiency** We can do the computation involving binary values in the comparison more efficiently by adding to $\mathcal{F}'$ a command that, given a register $[r_i]$ containing a binary value, produces a new register containing the same binary value, but now interpreted as an element in $GF(2^8)$, denoted $[r_i]_{256}$. The $\oplus$ operation is now simply addition in $GF(2^8)$. The idea behind this is of course that we will implement $[r_i]_{256}$ as sharing over the field $GF(2^8)$ so that secure $\oplus$ becomes only a local addition and so is much faster than before. This reduces the diamond operator to a single round implying only $\log(32) = 5$ rounds for the repeated application and less than 10 rounds overall.

This only leaves the question of how to do the conversion. We do this by having each server produce $[s_j], [b_j]_{256}$ for a random bit $b_j$, and random $\kappa$-bit number $s_j$, chosen such that its least significant bit is $b_j$. It is now (statistically) secure to open $r_i + \sum_j s_j$. The least significant bit of this number equals $r_i \oplus b_1 \oplus \cdots \oplus b_n$. Adding this bit to the shares of $[b_1 \oplus \cdots \oplus b_n]_{256}$ produces $[r_i]_{256}$.

We leave the (straightforward) formal proof that this is secure to the reader.

## 5 The Auction Implementation

In the system that was deployed, a web server was set up for receiving bids, and three servers were set up for doing the secure computation. Before the auction started, public/private key pairs were generated for the computation servers, and a representative for each involved organization stored the private key material on a USB stick, protected under a password.

Each bidder logged into the webserver and an applet was downloaded to his PC together with the public keys of the computation servers. After the user typed in his bid, the applet secret shared the bids, and encrypted the shares under the server public keys. Finally the entire set of ciphertexts were stored in a database by the webserver.

As for security precautions on the client side, we did not explicitly implement any security against cheating bidders, as mentioned and motivated in the previous section. Moreover, we considered security against third-party attacks on client machines as being the user's responsibility, and so did not explicitly handle this issue.

After the deadline for the auction had passed, the servers were connected to the database and each other, and the market clearing price was securely computed, as well as the quantity each bidder would buy/sell at that price. The representative for each of the involved parties triggered the computation by inserting his USB stick and entering his password on his own machine.

The system worked with a set of 4000 possible values for the price, meaning that the market clearing price could be found using about 12 secure comparisons.

The bidding phase ran smoothly, with very few technical questions asked by users. The only issue was that the applet on some PC's took up to a minute to complete the encryption of the bids. It is not surprising that the applet needed a non-trivial amount of time, since each bid consisted of 4000

numbers that had to be handled individually. A total of 1229 bidders participated in the auction, each of these had the option of submitting a bid for selling, for buying, or both.

Fig. 5 shows how much time is spent on the used computation for different sizes of input (we did not time the real auction, as we did not find it appropriate to compute anything else than what was told to the participants. Timings where however performed in exactly the same setup, except that random bids where used). Both the timing runs as well as the actual auction used three Dell laptops (Latitude D630) with 4GB RAM (Java being allocated 1500MB on each machine), Intel Centrino Dual Core (2.2GHz) processor, running Windows XP Pro, and connected through an Ethernet LAN using a 100Mbps switch.

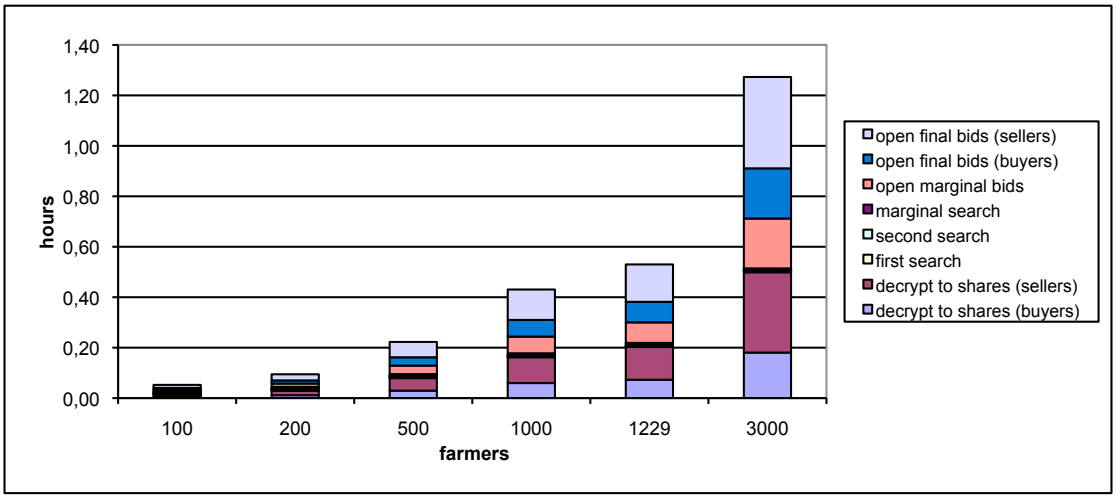


**Fig. 5.** Timings

The computation takes place in eight stages:

**decrypt to shares (buyers)** the servers shares of buy bids are decrypted
**decrypt to shares (sellers)** the servers shares of sell bids are decrypted
**first search** an upper bound on the MCP is located
**second search** a lower bound on the MCP is located
**marginal search** bids which may help resolve the MCP if the upper and lower bound do not match are located
**open marginal bids** bids which may help resolve the MCP if the upper and lower bound do not match are opened
**open final bids (buyers)** for each bidder the buying bids to be realised based on the MCP are opened
**open final bids (sellers)** for each bidder the selling bids to be realised based on the MCP are opened

As can be seen most of the time is spent on decrypting shares of the individual bids, which is not surprising, as the input to the computation, with e.g. 1229 bids, consist of about 9 million individual numbers. Compared to the theoretical insight which suggests that the major overhead is in the communication involved in the actual computation (i.e. comparisons), this is however an interesting observation.

The actual computation was done January 14, 2008 and lasted about 30 minutes. Most of this time was actually spent decrypting. As a result of the auction, about 25 thousand tons of production rights changed owner. To the best of our knowledge, this was the first large-scale and genuinely practical application of multiparty computation.

## 6 Conclusion

How successful have we been with the auction system, and does the technology have further potential in practice?

Other than the fact that the system worked and produced correct results, it is noteworthy that about 80% of the respondents in an on-line survey said that it was important to them that the bids were kept confidential, and also that they were happy about the confidentiality that the system offered. Of course, one should not interpret this as support for the particular technical solution we chose, most farmers would not have any idea what multiparty computation is. But it is nevertheless interesting that confidentiality is seen as important. While it is sometimes claimed that ordinary people do not care about security, we believe they sometimes do care – in particular if money is involved, and even more so if other parties are involved with interests that clearly conflict with yours. For instance, given the history of the sugar beet market, there is little doubt that "confidentiality" for the farmers include confidentiality against Danisco. Danisco and DKS have been satisfied with the system, and at the time of writing, the auction has already been run successfully a second time.

During the experiment we have therefore become convinced that the ability of multiparty computation to keep secret *everything* that is not intended to be public, really is useful in practice. As discussed earlier, it short-circuits discussions and concerns about which parts of the data are sensitive and what common security policy one should have for handling such data.

It is sometimes claimed that the same effect can be achieved by using secure hardware: just send all input data privately to the device which then does the computation internally, and outputs the result. Superficially, this may seem to be a very simple solution that also keeps all private data private. Taking a closer look, however, it is not hard to see that the hardware solution achieves something fundamentally different from what multiparty computation does, *even if one believes that the physical protection cannot be broken*: note that we are still in a situation where some component of our system –the hardware box– has access to *all* private data in cleartext. If we had been talking about an abstract ideal functionality, this would not be a problem. But a real hardware box is a system component like any other: it must be securely installed, administrated, updated, backed up, etc. This must be done under a security policy that all parties can agree on, and parties must trust that the administrator adheres to the policy. It may be time-consuming, expensive or even impossible to reach agreement on all this if parties have conflicting interests. We believe that a much more natural use of secure hardware is for each party in a multiparty computation to use it in order to improve *his own* security, i.e., to make sure that the protocol messages is the only data his system leaks.

Another standard alternative to MPC is to pay a trusted party such as a consultancy house to do the computation. We said earlier that the parties in our scenario decided against this because it would have been much more expensive. One could claim, of course, that this was only because the alternative was to have a research team do the whole thing for free – and that hence the experiment does not show that MPC is commercially viable. While the experiment has certainly not produced a business plan, we wish to point out that an MPC based solution only has to be developed once and costs can then be amortized over many applications. In some cases one may not even need to adapt the system – for instance, in the case of the sugar beet auction it is very likely that the same auction will be run once a year for some time to come.

In conclusion, we expect that multiparty computation will turn out to be useful in many practical scenarios in the future.

## References


1. P. Bogetoft, I. Damgård, T. Jakobsen, K. Nielsen, J. Pagter and T. Toft: *A Practical Implementation of Secure Auctions based on Multiparty Integer Computation.* Proc. of Financial Cryptography 2006, Springer Verlag LNCS.
2. P. Bogetoft, K. Boye, H. Neergaard-Petersen, K. Nielsen: *Reallocating sugar beet contracts: Can sugar production survive in Denmark?*, European Review of Agricultural Economics 2007 (34):, pp. 1–20.
3. M. Ben-Or, S. Goldwasser, A. Wigderson: *Completeness theorems for Non-Cryptographic Fault-Tolerant Distributed Computation*, Proc. ACM STOC '88, pp. 1–10.
4. R. Cramer, R. Gennaro and B. Schoenmakers: *A Secure and Optimally Efficient Multi-Authority Election Scheme*, Proc. of EuroCrypt 1997
5. R. Canetti: *Universally Composable Security*, The ePrint archive, www.iacr.org.
6. R. Cramer and I. Damgård: *Multiparty Computation, an Introduction*, in Contemporary Cryptology, Advanced courses in Mathematics CRM Barcelona, Birkhäuser.
7. D. Chaum, C. Crépeau, I. Damgård: *Multi-Party Unconditionally Secure Protocols*, Proc. of ACM STOC '88, pp. 11–19.
8. R. Cramer, I. Damgård and U. Maurer: *Multiparty Computations from Any Linear Secret Sharing Scheme.* In: Proc. EUROCRYPT '00.
9. R. Cramer, I. Damgård, S. Dziembowski, M: Hirt and T. Rabin: *Efficient Multiparty Computations With Dishonest Minority*, Proceedings of EuroCrypt 99, Springer Verlag LNCS series.
10. P. Dasgupta, P. Hammond, E. Maskin: *The Implementation of Social Choice Rules: Some General Results on Incentive Compatibility*, Review of Economic Studies 1979 (46):, pp. 27–42.
11. I. Damgård and J.B. Nielsen: *Universally Composable Efficient Multiparty Computation from Threshold Homomorphic Encryption*, Proc. of Crypto 2003, Springer Verlag LNCS.
12. I. Damgård, M. Jurik: *A Generalisation, a Simplification and Some Applications of Paillier's Probabilistic Public-Key System.* Public Key Cryptography 2001: 119-136
13. I. Damgård, M. Fitzi, E. Kiltz, J.B. Nielsen, T. Toft: *Unconditionally Secure Constant-Rounds Multiparty Computation for Equality, Comparison, Bits and Exponentiation.* Proc. of TCC 2006, pp. 285-304, Springer Verlag LNCS.
14. I, Damgård and R. Thorbek: Non-Interactive Proofs for Integer Multiplication, proc. of EuroCrypt 2007.
15. A. Gibbard: *Manipulation of Voting Schemes: A General Result*, Econometrica 1973 (41):, pp. 587–601.
16. O. Goldreich, S. Micali and A. Wigderson: *How to Play Any Mental Game or a Completeness Theorem for Protocols with Honest Majority*, Proc. of ACM STOC '87, pp. 218–229.
17. R. Gennaro, M. Rabin, T. Rabin, *Simplified VSS and Fast-Track Multiparty Computations with Applications to Threshold Cryptography*, Proc of ACM PODC'98.
18. M. Hirt, U. Maurer: *Complete Characterization of Adversaries Tolerable in General Multiparty Computations*, Proc. ACM PODC'97, pp. 25–34.
19. R.B. Myerson: *Incentives Compatibility and the Bargaining Problem*, Econometrica 1979 (47):, pp. 61–73.
20. J.D. Nielsen and M.I. Schwartzbach: *A domain-specific programming language for secure multipartycomputation*, Proceedings of Programming Languages and Security (PLAS), 2007, ACM press
21. T. Rabin, M. Ben-Or: *Verifiable Secret Sharing and Multiparty Protocols with Honest majority*, Proc. ACM STOC '89, pp. 73–85.