Confidential Benchmarking based on Multiparty Computation*

Ivan Damgård[1], Kasper Damgård[2], Kurt Nielsen[3], Peter Sebastian Nordholt[2], and Tomas Toft[4]

[1] Department of Computer Science, Aarhus University
[2] The Alexandra Institute
[3] Department of Food and Resource Economics, University of Copenhagen
[4] Danske Bank

Abstract. We report on the design and implementation of a system that uses multiparty computation to enable banks to benchmark their customers' confidential performance data against a large representative set of confidential performance data from a consultancy house. The system ensures that both the banks' and the consultancy house's data stays confidential, the banks as clients learn nothing but the computed benchmarking score. In the concrete business application, the developed prototype help Danish banks to find the most efficient customers among a large and challenging group of agricultural customers with too much debt. We propose a model based on linear programming for doing the benchmarking and implement it using the SPDZ protocol by Damgård et al., which we modify using a new idea that allows clients to supply data and get output without having to participate in the preprocessing phase and without keeping state during the computation. We ran the system with two servers doing the secure computation using a database with information on about 2500 users. Answers arrived in about 25 seconds.

1 Introduction

We report on the design and implementation of a system that uses secure multiparty computation (MPC) for credit rating in Danish banks. The use of MPC allows us to create a richer data foundation by merging confidential data from different sources and securely compute relative performance scores (benchmarks) on the joint secret shared data. In close collaboration with Danish banks and a consultancy house, we have developed this confidential benchmarking system that uses linear programming to compute benchmark scores as a complement to traditional credit rating of the banks' customers.

The business case focuses on farmers as a business segment that is particularly challenging for Danish banks. The basic problem is that a large number of farmers have had too high debt ratios and too low earnings for years and the banks have been reluctant

* Financial support from the Center for research in the Foundations of Electronic Markets (CFEM) funded by the Danish Council for Strategic Research and the FP7 EU-project PRACTICE are gratefully acknowledged.

to realise the losses[5]. One would like to avoid a chain reaction that may affect the individual bank's own credit rating and hereby it's ability to lend money on the interbank market. So on the one hand, a number of banks have too many farmers with potential losses as customers and on the other hand, status quo worsens the situation for this group of customers that need to invest to stay competitive. To control the situation and to avoid an escalation, the banks are forced to pick the right farmers among this group of risk-prone customers. This requires more analysis than the traditional credit scores. The banks need to look beyond the debt and find the better performing farmers that are more likely to pay back the loans. To do this one needs accounting and production data from a large range of peer farmers. Since farmers are not required to publish accounting data, a number of banks lack the basic data for sound in-house analysis of their own customers.

However, there is a consultancy house in the market that possesses accounting data from a large number of farmers. This creates an opportunity to score the bank's customers relative to the sector as a whole and not just the banks own portfolio which may not be representative for the business segment.

Hereby, the business case has a characteristic property that it shares with many similar cases: The inputs we need to solve the problem are held by different parties and privacy issues prevent them from pooling the information. In our case, the consultancy house that has the farmers accounting data is of course required to keeps its database confidential. On the other hand, banks are not allowed to give away data on their customers including the identity of its customers. In fact, even ignoring regulations, it would be particularly problematic in our case if the bank were to send data on a customer C in the clear to the consultancy house. If C's data are already in the database, it is very like that the consultancy house could find out which customer the bank wants to evaluate, and this is of course a breach of confidentiality.

Agreeing on a trusted third party who can perform the computation may be both difficult and expensive, hence a different solution is desirable. Secure multiparty computation (MPC) provides a solution – two or more parties can compute any function on private inputs such that the only new information leaked is the intended output of the function [Yao82,GMW87,BOGW88,CCD88]. However, though any function is computable in theory, specialized protocols for concrete problems are typically proposed to achieve acceptable efficiency. This is also the case here, where we implement a secure Linear Program (LP)-solver and demonstrate its applicability.

Conceptually, MPC can be seen as an implementation of a Trusted Third Party (TTP) that receives the input and confidentially computes the result (in this case the result of the benchmarking), while not revealing anything else about the inputs. As such, the MPC approach is analogous to paying a consultancy house to act as a TTP. However, the economic argument for using MPC is that while a consultancy house is likely to charge the parties substantial fees for every analysis, the cost of developing software for MPC only has to paid once and can be amortised over many applications of the system.

Linear programming (LP) is one of the basic and most useful optimization problems known. The goal is to find an assignment to variables, $\boldsymbol{x} = (x_1, \ldots, x_n)$, subject to m

[5] The banks are typically the lenders with the utmost priority in case of default.

linear constraints

$$C \cdot \boldsymbol{x} \leq \boldsymbol{b}$$

maximizing (or minimizing) a linear function, $f(\boldsymbol{x})$. LP is widely used in Operational Research and applied Micro Economics to solve real life problems such as resource allocation, supply chain management or benchmarking as in this paper.

The remainder of this paper is structured as follows. Section 2 describes the application scenario and the linear program to be solved. Section 3 and 4 describe the applied protocol and implementation respectively and concluding remarks are given in Section 5.

2 Application Scenario and Benchmarking Model

Credit rating of a firm is all about estimating the ability that the firm can fulfill its financial commitments based on historical data. Traditional credit rating models such as the original Altman's Z-score aim at predicting the probability that a firm will go into bankruptcy. Using various statistical methods the most relevant explanatory variables are selected and the credit rating model is estimated, see e.g. [Mes97] for a general introduction.

The traditional credit scoring divides customers into groups depending on their overall credit worthiness. While new customers may simply be rejected based on a bad credit score, existing customers that end up with a bad credit score, cannot be rejected without a risk of losses. When larger groups of customers experience a drop in their credit worthiness the banks it self may get exposed by a drop in credit rating. This was indeed the case with the global financial crises that was created over a number of years and ignited by Leman Brothers bankruptcy in 2008. The present problem where a large group of Danish farmers have low credit worthiness, goes back to excessive lending prior to 2008.

In general, farming requires large investments to generate profit. The Danish farmers have historically been highly efficient and to a large extent adapted to the relative high operational costs, not least the high wages (see e.g. [ANB12]). The otherwise successful substitution away from increasing labor costs has resulted in high debt/equity ratios. This challenge the Danish banks with many farmers as customers that on the one hand require large investments to become competitive and on the other hand suffer from high debt/equity ratios. According to BankResearch.dk that continuously evaluate the Danish banks, 7 of the 8 worst scoring banks are among the 30 Danish banks that are most exposed in the agricultural sector in 2014. These 30 banks are all small and medium sized banks and have from 10 to 35% of total loans and guarantees in agriculture or fishery. This development emphasises the fact that when selecting the right farmers for future loans, one needs an analysis that is based on a larger number of comparable farms than what is found in the individual banks' own portfolios.

In close collaboration with the consultancy house that represents the majority of the farmers and selected small and medium sized banks, a prototype software has been developed. The consultancy house has detailed account and production data that are not publicly available. The added security allows us to create a richer data foundation by merging the confidential data from the accounting firm with additional confidential

data from the individual banks. The secure LP solver allows us to conduct state-of-the-art relative performance analysis directly on the richer, though secret data set. The resulting benchmarks is used to evaluate new individual customers as well as the banks' portfolios - in either case the analysis reflects performance relative to the agricultural sector as a whole.

In general terms, benchmarking is the process of comparing the performance/activities of one unit against that of best practice. We apply Data Envelopment Analysis (DEA), which is a so-called frontier-evaluation technique that supports best practice comparisons (or efficiency analysis) in a realistic multiple-inputs multiple-outputs framework. Instead of benchmarking against engineering standards or statistical average performances, DEA invokes a minimum of a priori assumptions and evaluates the performance against that of specific peer units. For these reasons, DEA has become a popular benchmarking approach.

DEA was originally proposed by [CcR78,CcR79], and has subsequently been refined and applied in a large number of research papers and by consultants in a broad sense. A 2008 bibliography lists more than 4000 DEA references, and more than 1600 of these are published in good quality scientific journals [EPT08].

Most often, DEA is used to get *general insight*, e.g. about the variation in performance or the productive development in a given sector. However, DEA have also proven useful in *incentive provision*, e.g. for regulation or as part of an auction market cf. [ABT05,BN08,NT07]. Finally, DEA has also been applied as a direct alternative to traditional credit rating models in predicting the risk of failures see e.g. [CPV04,PAS04,PBS09]. DEA can be formulated as an LP-problem and therefore in general be solved by the Secure LP-solver described in this paper.

To formally define the DEA methodology used for this analysis, consider the set of n observed farms. All the farms are using $k = 1, \ldots, r$ inputs to produce $l = 1, \ldots, s$ outputs, where the *input-output vector* for farm i is defined as: $(x_i, y_i) \in \mathbb{R}_+^{r+s}$. Let x_i^k denote farm i's consumption of the k'th input and y_i^l its production of the l'th output.

The DEA input efficiency score under variable returns to scale (c.f. [BCc84]) for farm i is called θ_i^* and is defined as:

$$\begin{aligned} \theta_i^* = \; & \max \theta_i \\ \text{s.t.} \; & \sum_{j=1}^{n} \lambda_j x_j^k \leq x_i^k, k = 1, \ldots, r \\ & \sum_{j=1}^{n} \lambda_j y_j^l \geq \theta_i y_i^l, l = 1, \ldots, s \\ & \sum_{j=1}^{n} \lambda_j = 1, \lambda_j \geq 0, j = 1, \ldots, n. \end{aligned} \tag{1}$$

The interpretation is that the efficiency score of farm i is the largest possible factor by which farm i can expand all outputs while still maintaining present input consumption. We use the reverse output efficiency score $1/\theta_i^*$ to fix the score between 0 and 1. As an example, the interpretation of a reverse output score of 80 % is that the farm uses

only 80 % of its potential as estimated by comparing to the other farms. (the estimated best practice benchmark). Apart from evaluating individual farmers we use distribution plots to evaluate the individual bank's portfolio of farmers. For further details on the use of DEA, the reader is referred to the textbooks by e.g. [BO11,CST07].

The software described below is in the process of being tested by the end-users i.e. selected banks[6]. The system is able to do several different types of analysis and these have been designed in collaboration with the consultancy house and tested by consultants that are familiar with the evaluated farmers. Here we concentrate on one of the benchmarking analyses that is used on all of the 4 major group of farmers (milk, pig, plant and fur production). The initial data provided by the consultancy house consists of approx. 7500 accounts across the four types of farms in total and provide a representative and sound foundation for the analysis.

The applied benchmarking model reported on in this paper focuses on the farms abilities to transform the basic inputs labour, capital (divided into three sub-groups) and variable inputs into gross output, i.e., profit. The model has been developed, discussed and tested together with the involved consultancy house.

- x_i^1: Labour (wages for paid labor + 450000 DKK to the owner)
- x_i^2: Value of land
- x_i^3: Liquid capital
- x_i^4: Other capital assets
- x_i^5: All variable costs (excluding internal transfer)
- y_i^1: Gross output (including EU subsidies and other income)

The resulting benchmarking scores from the 7500 farmers, supports the basic argument, that additional analysis are required in selecting the best performing farmers among the many with too much debt. Table 1 shows how the benchmarking scores are distributed within segments of the farmers' debt/equity ratios. The result shows that the wast majority have a debt/equity ratio larger than 50% and that farmers with similar debt/equity ratio have widely spread benchmarking scores. However, there is a tendency to a higher benchmarking score for farmers with higher debt/equity scores i.e. among the most risky customers seen from the banks' perspective. Although traditional credit ratings involves other elements than what is captured by the debt/equity ratio, the results do indicate that the suggested benchmarking analysis is able to identify the better performing farmers among many risky customers with too much debt.

We have described how the confidential benchmarking system can be used by banks to evaluate potential new customers, as well as particular all farmers in their existing portfolios. In addition, the situation allows us to provide a different type of analysis, namely a quick stress test of a bank's portfolio: for all the banks that participate, it holds that the 7500 accounts in the database include a significant share of the bank's agricultural customers. So we can give the bank an idea of how well its customers are doing by comparing those that are in the database with the total population. We do this by first having the database *locally* compute the benchmark score for all farmers in the database. This does not require MPC and can be done quite efficiently. Now, only

[6] An early stage demo version of the software has been tested and resulted in valuable feedback for the development of the prototype.

Debt/equity ratio	Number of farmers	Average score	Standard deviation (score)
50%-60%	632	41.6 %	20.6 %
60%-70%	1363	40.0 %	18.6 %
70%-80%	2014	43.0 %	17.2 %
80%-90%	1807	47.8 %	15.6 %
90%-100%	1234	48.3 %	15.1%

Table 1: Debt/equity ratio and distribution of benchmark scores

the bank knows the identity of its customers and this is considered to be confidential information. Therefore the bank uploads a list of id numbers that are secret shared across the two servers, and then, inside a secure computation, we can select the bank's portfolio of farmers and return a summary based on their precomputed scores. Hereby, the initial stress test of the portfolios can be done without delays from the otherwise time intensive LP solving. Note also that since this computation touches every entry in the database, no information is released on which entries belong to customers of the bank in question.

3 Using the SPDZ Protocol for Benchmarking

The scenario in which we want to implement Multiparty Computation (MPC) is composed of the following players: clients, who supply input and get outputs and servers who execute the actual computation. We assume that any number of clients and up to $n-1$ of the n servers may be maliciously corrupted.

In relation to the case outlined in the previous section, a client would typically be a bank who wants to get the score for a certain customer. One special client (who only supplies input) would be the consultancy house who has the database. The servers would be run by different parties with an interest in the system. For the deployment of the present prototype, the two organisations involved in the development of the system (names left out for anonymity), each control one of the two servers involved in the secure computations. Based on discussion with the involved business partners, we expect that the consultancy house and the Danish Bankers Association will control the two servers in a commercial setup.

We use the SPDZ protocol from [DPSZ12] to do the computation. This protocol is indeed capable of general secure computation and can tolerate that all but one of the servers doing the computation are corrupt. Tolerating a dishonest majority actually requires the use of heavy public-key crypto machinery. However, one of the main ideas in SPDZ is to push the use of this into a preprocessing phase that can be done ahead of time (without knowing the inputs), and then use preprocessed material to do the actual computation very efficiently.

However, it is not clear how to integrate the clients. In SPDZ, it is assumed that each player plays both the role of a server and of a client who can supply input and get output. To do this, SPDZ requires that all players take part in the preprocessing stage. But in our scenario, we want to separate the client and server roles and we definitely do

not want to demand from our clients that they do the preprocessing: in our application, it may not even be known who the clients are at preprocessing time. We would also like that the clients do not need to keep state while the computation in running as this simplifies the implementation of client software[7]. We explain our solution below after we explain some more details of the SPDZ protocol:

SPDZ can securely evaluate any arithmetic circuit over a finite field $\mathbb{F}$, and we will assume $\mathbb{F}$ is the field with p elements for a prime p is the following. Each value a that occurs in the computation (as input, output or intermediate result) is represented in a certain format denoted by $\langle a \rangle$. The idea is that each server holds part of the data that represents a. More specifically, each server S_i holds a *share* a_i of a, such that $a = a_1 + \cdots + a_n$ and the a_i's are randomly chosen such that even given $n-1$ of them, a remains unknown. The servers also hold data that can be used to authenticate the value of a if we want to retrieve it in the clear, but the details of this are not important here.

SPDZ includes protocols for operating securely on these representations of field values, i.e., from $\langle a \rangle, \langle b \rangle$, the servers can compute $\langle a + b \rangle$ or $\langle ab \rangle$ without revealing anything about a or b. Similarly we add or multiply by a publicly known constant. The main role of the preprocessing is to supply shared randomness that facilitates secure multiplication, but it can also easily be configured to supply any number of representations $\langle r \rangle$, where $r \in \mathbb{F}$ is a random value that is unknown to all players, this will be very important in the following.

The overall idea of the computation phase in SPDZ is then to first construct representations in the right form of the input values, do the required arithmetic operations to come up with representations of the desired outputs, and then open these to reveal the results to the players who are to receive output.

3.1 Allowing Clients to give Input and get Output

Let us first discuss how to give output to a client C, assuming that the servers have managed to compute a representation of an output value $\langle y \rangle$. As mentioned, this means that each server S_i holds y_i where $y = \sum_i y_i$.

So this may seem easy: each S_i sends y_i privately to C, who adds all values received to get y. However, this will of course not work, even a single malicious server could lie about its value and make C obtain an incorrect result. In the original SPDZ protocol, this type of problem is solved by having the servers collaborate to authenticate the sum of the values supplied. But C cannot take part in and be convinced by such a procedure unless he took part in the preprocessing stage, and we want to avoid this.

So instead we propose to encode the output value in such a way that any modification by malicious servers can be detected by the client[8]. As a first attempt, suppose the

[7] In [JNO14], a generic solution client solution was proposed that works for any MPC protocol, but it requires the client to keep state. In principle, one can always store client state info on the servers, but since our servers are malicious it needs to be authenticated and secret shared or encrypted, and this adds further complications to the implementation.

[8] This is actually the notion of a strong AMD code[CDF+08], the construction we give here is slightly different from previous ones, though, and fits better into our protocol.

servers retrieve a random representation $\langle r \rangle$ from the preprocessing, then they compute securely $\langle w \rangle = \langle yr \rangle$ and finally send all shares to y, r and w privately to C. He can now reconstruct and check that indeed $yr = w$ and will reject the output if not. Recall that if we want to tolerate a dishonest majority of servers, we cannot guarantee that players will always terminate with correct output, as servers may for instance just stop playing. So simply aborting if something is wrong is the best we can do.

One can think of w as an authentication tag and r as a key, so this is an authentication scheme similar to the one already used in SPDZ. As we show below, this will indeed ensure that any attempt to change y will be detected except with probability $1/p$, where we assume that p is large enough that this is negligible. However, there is still a subtle problem: y is supposed to be private, known only to the client. Now, a malicious server could (for instance) change r and leave the other values alone. It is easy to see that then the client will abort if $y \neq 0$ but will accept if $y = 0$. The adversary can observe this and get information on y[9]. A way to solve this problem is to also authenticate the key r in exactly the same way as we authenticated y. It may seem that we are just pushing the problem in front of us, but note that r is guaranteed to be random (contrary to y). So while the adversary can still make a guess at r and get to see if his guess was correct, the probability of guessing correctly is negligible, so we can ignore this possibility.

The protocol is specified in detail in Fig. 1 and we have the following result on its security:

Protocol *Output Delivery*.

Given $\langle y \rangle$ where we want to reveal y to C and only to C.

1. The servers do the following: Retrieve unused random $\langle r \rangle, \langle v \rangle$ from the preprocessed material and compute $\langle w \rangle = \langle yr \rangle, \langle u \rangle = \langle vr \rangle$.
   Each server S_i sends its shares y_i, r_i, w_i, v_i, u_i privately to C.
2. C does the following: compute $y = \sum_i y_i, r = \sum_i r_i, w = \sum_i w_i, u = \sum_i u_i, v = \sum_i v_i$. Check that $w = yr$ and $u = vr$. If not, abort, else output y.

Fig. 1: Protocol for giving output to C

Lemma 1. *The protocol in Fig. 1 satisfies the following. Privacy: if C is honest, the adversary's view of the protocol can be simulated with statistically close distribution without knowing y. Correctness: an honest C will accept a value different from y with probability at most $1/p$.*

Proof. As for correctness, assume for contradiction that that C accepts $y' \neq y$. Let w', r', v', u' be the other values reconstructed by C. Since C accepts we have $w' = y'r'$. Also, by correctness of the original SPDZ protocol, we know that $w = yr$. We can write $y' = y + \alpha, w' = w + \beta, r' = r + \gamma$ where α, β, γ are errors introduced by the adversary

[9] This problem does not occur in the original SPDZ protocol, since there the values that are opened are public.

sending incorrect shares. Inserting this into $w' = y'r'$, we get $w + \beta = (y + \alpha)(r + \gamma)$. Using $w = yr$, this can be simplified to

$$\beta = y\gamma + \alpha r + \alpha\gamma$$

But since $y \neq y'$ implies $\alpha \neq 0$, this equation determines r uniquely, so the adversary must effectively guess r to make C accept an incorrect value. This happens with probability at most $1/p$ since r is unknown a priori.

As for privacy, note that the adversary's view includes the correct shares for corrupt servers of y, r, w, u and v and the share they send to C. The only new information the adversary learns is whether the protocol aborts. We show that from shares and messages sent by corrupt servers, one can predict whether the protocol aborts, except with negligible probability, and hence the adversary's view can be simulated without knowing y. To see this, we reuse the notation from the proof of correctness, and also define $u' = u + \delta, v' = v + \epsilon$. We can then see in the same way as above that C will find that $u' = v'r'$ if and only if

$$\delta = v\gamma + \epsilon r + \epsilon\gamma.$$

Of course, the error terms $\alpha, \beta, \gamma, \epsilon, \delta$ can be computed from the adversaries view and we claim that if they are not all 0, C will abort except with negligible probability. So this gives the prediction we were after (note that, of course, if all error terms are 0, C will accept).

Note first that r plays the role as the authenticated message in the equation $u = vr$, and we already argued that C will abort almost always if the message is changed. It follows that if $\alpha \neq 0$ or $\gamma \neq 0$, C aborts almost always. So assume therefore that $\alpha = \gamma = 0$. Our equations simplify to $\beta = 0$ and $\delta = \epsilon r$. Clearly, C aborts if $\beta \neq 0$. Further, if $\epsilon \neq 0$, we can only get C to accept if $r = \delta/\epsilon$ which happens with negligible probability since r is random. So we can assume that $\epsilon = 0$ and then it follows that $\delta = 0$ as well, since otherwise C always aborts.

Before we consider supplying inputs, note that a client can easily broadcast a message to the servers, by simply sending the same message to them all. Then the servers can compare what they received and abort if there is a disagreement. Again, we cannot avoid the possibility of aborting if the client and a majority of severs are malicious.

Now suppose C wants to contribute input value x. For this, we use a preprocessed random $\langle s \rangle$. We now use the previous protocol to reveal s to (only) C, who can then broadcast $x - s$. The severs can now use a SPDZ subprotocol to add the publicly known value $x - s$ into $\langle s \rangle$ to obtain $\langle x \rangle$. The protocol is specified in detail in Fig. 2.

For the security of the input protocol, note that if C is honest, then the adversary learns nothing about x because s is uniformly random and hence $x - s$ is uniform as well. By the security of the SPDZ protocol the representation produced by the servers will contain the value x intended by C. If C is corrupt, the representation produced will contain a well defined value namely one that is determined by s and the value C broadcasts.

3.2 Using SPDZ for Linear Programming

In order to use the SPDZ protocol for Linear Programming as required in the application, we apply the well known Simplex algorithm. For this we need to do integer arith-

Protocol *Input Supply*.

C holds value x that he wants to supply as input top the servers.

1. The servers do the following: Retrieve an unused random $\langle s \rangle$ from the preprocessed material and use the Output delivery protocol to give s to C.
2. C broadcasts $x - s$ to the servers, and the servers compute $(x - s) + \langle s \rangle = \langle x \rangle$.

Fig. 2: Protocol for C to supply input

metic and comparison on sufficiently large integers. To do this we choose the modulus p large enough compared to the actual data, then we can do additions and multiplications by doing them mod p but avoid overflow.

For divisions, two approaches have been proposed in the literature: Toft [Tof09] suggests using a variant of the Simplex algorithm, where it is ensured that whenever we need to compute a/b, it will always be the case that b divides a. This means we can do the division by multiplying by $b^{-1} \bmod p$ which is much easier than a standard division. The downside of this idea is that the involved numbers (and hence the required modulus size) grows with the size of the Linear Programming problem we solve. Catrina and de Hoogh[Cd10] suggest to use instead fixed point rational numbers throughout. This means we can make do with smaller integers and division is now relatively easy. The catch is that we have to live with rounding errors in the result (where Toft's approach is exact), and that the other arithmetic operations become somewhat more complicated.

In our case we found that the problem size we were dealing with could be made small enough to allow us to use Toft's approach, which is simpler to implement. More specifically, we exploit the fact that one of the parties (the consultancy house) possesses the database with data on all the farmers we are comparing bank customers to. Therefore, prior to the secure computation, we can do a computation locally on the data base, that selects the most efficient farmers. More precisely, if we see the farmers as points in a multidimensional space, we identify those farmers that define the convex hull of all the points. Now, we only need to enter these farmers into the secure computation, since the answer to the linear programming problem will be the same. As a result, we had problems with 8 constraints and between 45 and 70 variables. To avoid overflow, we had to use a modulus p with 512 bits.

We also note that there are several different flavours of Simplex to choose from. In particular, Simplex works by initially setting up a Pivot table containing the input values. This table is then iteratively updated until it contains the solution. There are several different rules for how this update can be done, in particular, we considered Bland's rule and Danzig's rule. To explain the difference in geometric terms, the current state of the computation defines a corner of a polytope in a multidimensional space. The update corresponds to selecting an edge on the polytope, and walk along this edge to obtain a better solution than the one defined by the current position. Bland's rule selects the first edge that will improve the current solution, while Danzig's rule considers all edges and selects the one that will give the largest improvement of the current solution. Clearly Danzig's rule requires more computation per iteration and is less "MPC-friendly" because we need several comparisons which are quite heavy. Nevertheless, in our expe-

rience, the number of iterations you get is so much smaller that Danzig's rule gives us better performance [10]. More details on this can be found in Section 4.2.

Finally, the comparisons are done using ideas from [DFK+06].

Our implementation leaks the number of iterations done in Simplex. This can be partially solved by doing dummy iterations, but this will of course slow down the system. We chose to put priority on getting answers as fast as possible.

4 Prototype and Performance

To demonstrate the potential of doing secure benchmarking a demo application was implemented which was tested by selected Danish banks. After initial positive feedback on the demo from the banks an extended prototype has now been build and a second round of testing at Danish banks is scheduled for the end of October 2015. In this section we will give an overview of the prototype and report on the observed performance.

4.1 Prototype

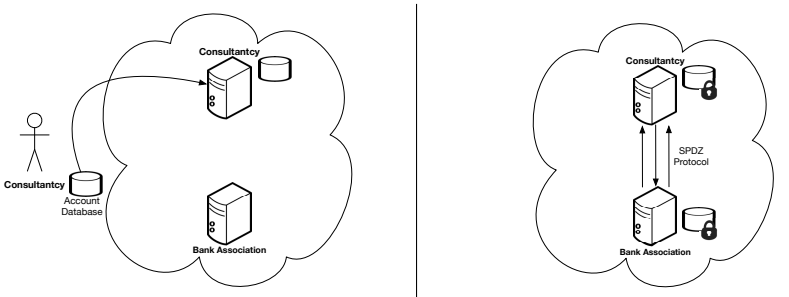
In the prototype secure computation is done between two servers. For convenience, in the prototype setup the two servers are controlled by the authors, however, in a real world setup the servers are assumed to be controlled by two distinct non-colluding parties. Namely, the consultancy house owning the database with the farmers' accounting information and a representative of the banks using the system (e.g., the Danish Bankers Association). The individual banks using the system are regarded as clients, i.e., they simply interact with the system to supply input and read output, but do not directly control the server representing them in the secure computation[11].

To initialise the system the consultancy house uploads its database to its secure computation server, which computes the reduced version of the dataset including only the relevant efficient accounts and adds this to the database. As discussed in Section 3 this corresponds to computing the convex hull of the points defined by the individual farmers. The two servers then run a protocol to secret share the database in the SPDZ internal format, and store the resulting secret shared database. Once the database is uploaded the consultancy house no longer needs to interact with the system and can go offline. This initialisation process is illustrated in Fig. 3.

Once the system is initialized, the banks can login and start submitting analyses to be performed on the system, using a simple web-interface executed in the browser. Since banks are simply clients, once they have submitted an analysis to be performed, they no longer need to interact with the system and can go offline. The secure computation servers will then perform the requested analysis and store the resulting output in the secret shared database. Later the bank can login and request the output of the analysis that has been run. The interactions of a bank with the system are illustrated in Fig. 4.

[10] In theory, Danzig's rule can lead to a cycle, so that the algorithm will not terminate, but this is is rare in practice, and never occurred in our testing.

[11] Alternatively, one could let each bank control their own secure computation server communicating directly with the consultancy house controlled server. This setup up was used for the initial demo system, but the current setup was deemed more scalable as it only requires two secure computation servers.



(a) The consultancy house uploading database

(b) Servers sharing database

Fig. 3: Initialising the secure benchmarking system

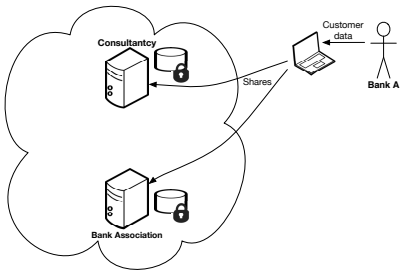
The web-interface used by the banks is connected directly to the two secure computation servers (over https) and runs the input/output protocols described above locally. This means that each server only ever sees the input and output of an analysis in secret shared form. In other words, as long as the bank can trust the two servers to not collude and at least one server to be honest the privacy and correctness of his inputs and outputs are guaranteed.

4.2 Performance

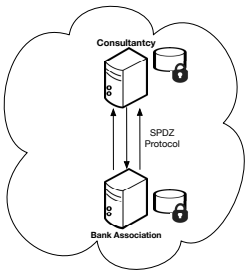
The MPC computation needed for the benchmarking analysis (i.e., the Simplex algorithm as described above) was implemented using the FRESCO framework, a Java framework for secure computation applications which contains an implementation of the SPDZ protocol. Each server is deployed in the cloud on a separate Amazon EC2 instance. Each instance is a standard general purpose *m4.large* instance, with 8GB RAM and 2 cores, running on a 2.4 GHz Intel Xeon processor. The servers are deployed in the same Amazon *availability zone* and *region*, essentially meaning they run in the same datacenter. This is a benefit for performance as it means there is rather low network latency between the servers. This is significant for protocols such as SPDZ with high round-complexity.

It is not entirely clear whether such a set-up would be acceptable in a real business application of the system: One can argue t‘hat having the server of each organization hosted by at the same cloud provider (Amazon in this case) places too much trust in the cloud provider. Namely, since the cloud provider has access to both servers including the secret shared database, a malicious cloud provider could potentially reconstruct all private data.

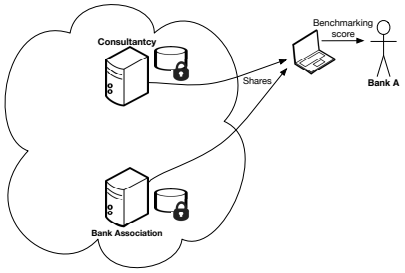
One can reduce these risks by using different clouds for different servers and an organisation could even share its data across 2 servers to reduce the trust in the cloud provider (by requiring the cloud provider collaborates with other cloud providers to break security). The main effect on performance by going to such a solution comes from increased latency of communication. We ran a small number of experiments to see how increased latency affected our prototype, and found that a 10-fold increase in latency



(a) Bank gives input



(b) Servers compute benchmark



(c) Bank reads output

Fig. 4: Interactions with the benchmarking system

gave a 3 fold increase in overall run time. So we estimate that while using different clouds would slow down the system significantly, it would not render it useless.

The prototype allows banks to benchmark a potential costumer's financial data against farmers in four different segments of agriculture: pigs, plants, milk and fur. For these segments the database we benchmark against holds 1786, 2645, 2258 and 358 accounts respectively. After we crop the datasets to only include the relevant efficient accounts, this is reduced to databases of 70, 63, 47 and 45 accounts respectively. In Table 2 we give the average running time in seconds for the analysis on each of the different segments. We see the running time in all segments is around 22-26 seconds even when running on this rather modest hardware, which should be tolerable for a real world system.

Segment	Accounts (reduced)	Av. time (sec)	Std. dev. (sec)
Pigs	70	23	4
Plants	63	22	4
Milk	47	26	8
Fur	45	22	6

Table 2: Time to do analysis for different agricultural segments

Surprisingly we see that we get the longest running times when benchmarking against the relatively small dataset of the milk segment. We also see a rather large standard deviation in the running time of 4-8 seconds (i.e., up to 30% in the milk segment). This variation is not due to secure computation it self but rather a property of the Simplex algorithm used to do the benchmarking analysis. Namely, the fact that the Simplex algorithm depending on the data we analyse may use a varying number of iterations to find a optimal value. Thus for evaluating the efficiency of the underlying Simplex implementation the time pr. iteration is really more relevant. We show these times in Table 3, which shows, as expected, that running times pr. iteration goes up the larger the dataset and datasets of similar size have about the same running time.

Segment	Accounts (reduced)	Av. num of iterations	Av. time pr. iteration (sec)	Std. dev. (sec)
Pigs	70	13	1.80	0.02
Plants	63	12	1.78	0.02
Milk	47	18	1.47	0.03
Fur	45	15	1.46	0.01

Table 3: Time for the a single Simplex iteration on different agricultural segments

The numbers in Table 2 and Table 3 are for our implementation of Simplex using Danzig's rule. As mentioned above we also considered the variation of the Simplex algorithm using Bland's rule. In Table 4 we give times for this variation. As explained above we see that using Bland's results in considerably faster iteration times, roughly around 20% faster than iterations using Danzig's rule. However, for our analysis the increased speed of iterations is cancelled out by having to do many more iterations to finish the analysis. Specifically, using Bland's instead of Danzig's rule increases the average amount of iterations by 33% in the milk segment and more than 150% in the plant segment. While this discourages us from using Bland's rule in our prototype, it may still be worth considering in other applications were if it would cause a less dramatic increase in the number of required iterations. Also, since there is a number of alternative heuristics for the Simplex updating rule it may be interesting to investigate their performance, to see if we could achieve both fast and few iterations.

Segment	Accounts (reduced)	Av. num of iterations	Av. time pr. iteration (sec)	Std dev. (sec)
Pigs	70	23	1.43	0.01
Plants	63	31	1.35	0.01
Milk	47	24	1.19	0.05
Fur	45	23	1.14	0.01

Table 4: Time for Simplex iterations using Blands rule

One may wonder how the performance scales as datasets grow beyond those naturally occuring in the context of this prototype. We did not experiment with this.

However, recall that the dataset we compute on is reduced to only include the efficient accounts. We conjecture that with the applied benchmarking approach (DEA) the number of efficient accounts is, for all practical applications, significantly smaller than the total number of accounts, as is the case in this paper. Thus, even for much larger datasets the reduced dataset including only efficient accounts, will not be much larger than those used in this prototype (given the same LP-problem, i.e., the benchmarking model). The basic arguments for this conjecture are 1) that only the farmers that represent best practice is relevant in the LP-problem and 2) that the number of observations that represent best practice is driven by the dimensionality of the LP-problem (number of inputs and outputs in the analysis), the total number of observations (in this case accounts) and the basic assumption about the technology (restrictions on the λ in the LP-problem). The more dimensions the more efficient observations (the model simply allows for more diversity) and the more observations the better representation of the diversity of the observations.

4.3 Future Directions

Next step is to bring in multiple data-providers. i.e., make also the consultancy house be a client among several clients contributing to the database against which we do our benchmarks. The data providers may add rows (more observations e.g. farms) or columns (more variables). More rows may come directly from the banks as well as other consultancy houses and accounting firms. More columns may e.g. be details about the debt (e.g. from the banks) or additional background information from Danish Statistics to give a few examples.

A more representative dataset (more rows) will improve the quality of the benchmarks and more importantly make the service more user friendly as less additional information is required. A more rich dataset (more columns) will allow for more analysis that better describe best practice and the performance of the farmers in this case.

This general approach with data from multiple sources merged into a larger database using MPC, is computationally more challenging. One challenge is to reduce the joint dataset to only the efficient observations which can no longer be done in the clear, but must be done in MPC.

More work on optimizing performance is also an obvious future direction. More experiments are needed to find the right levels of, e.g., parallelisation and hardware to be used.

5 Conclusion

We have presented a practical implementation of a system for confidentially benchmarking farmers against a large representative population of other farmers, using MPC and linear programming. The prototype has been developed and tested in collaboration with Danish banks and a consultancy house specialised in the agricultural sector. The

system creates new information that helps Danish banks selecting the better performing farmers among the many with bad credit rating.

We have presented some add-ons to the SPDZ protocol making it more useful in a client-sever scenario. The results obtained demonstrate that MPC can be useful in providing data that are useful and could not have been obtained in other ways without violating requirements for confidentiality.

References

[ABT05] Per J Agrell, Peter Bogetoft, and Jørgen Tind. DEA and dynamic yardstick competition in Scandinavian electricity distribution. *Journal of Productivity Analysis*, 23(2):173–201, 2005.

[ANB12] Mette Asmild, Kurt Nielsen, and Peter Bogetoft. Are high labour costs destroying the competitiveness of Danish dairy farmers? Evidence from an international benchmarking analysis. *MSAP Working Paper Series*, 2012.

[BCc84] RD Banker, A Charnes, and William W cooper. Some Models for Estimating Technical and Scale Inefficiencies in Data Envelopment Analysis. *Management Science*, 30:1078–1092, 1984.

[BN08] Peter Bogetoft and Kurt Nielsen. DEA based auctions. *European Journal of Operational Research*, 2008.

[BO11] Peter Bogetoft and Lars Otto. *Benchmarking with DEA, SFA, and R*. SPRINGER, New York, 2011.

[BOGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10, Chicago, Illinois, USA, May 2–4, 1988. ACM Press.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *20th ACM STOC*, pages 11–19, Chicago, Illinois, USA, May 2–4, 1988. ACM Press.

[CcR78] A Charnes, William W cooper, and E Rhodes. Measuring the Efficiency of Decision Making Units. *European Journal of Operational Research*, 2:429–444, 1978.

[CcR79] A Charnes, William W cooper, and E Rhodes. Short Communication: Measuring the Efficiency of Decision Making Units. *European Journal of Operational Research*, 3:339, 1979.

[Cd10] Octavian Catrina and Sebastiaan de Hoogh. Secure multiparty linear programming using fixed-point arithmetic. In Dimitris Gritzalis, Bart Preneel, and Marianthi Theoharidou, editors, *ESORICS 2010*, volume 6345 of *LNCS*, pages 134–150, Athens, Greece, September 20–22, 2010. Springer, Heidelberg, Germany.

[CDF+08] Ronald Cramer, Yevgeniy Dodis, Serge Fehr, Carles Padró, and Daniel Wichs. Detection of algebraic manipulation with applications to robust secret sharing and fuzzy extractors. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 471–488, Istanbul, Turkey, April 13–17, 2008. Springer, Heidelberg, Germany.

[CPV04] Anja Cielen, Ludo Peeters, and Koen Vanhoof. Bankruptcy prediction using a data envelopment analysis. *European Journal of Operational Research*, 154(2):526–532, 2004.

[CST07] William W Cooper, Lawrence M Seiford, and Kaoru Tone. *Data Envelopment Analysis: A Comprehensive Text with Models, Applications, references and DEA-Solver Software*. SPRINGER, second edition, 2007.

[DFK+06] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 285–304, New York, NY, USA, March 4–7, 2006. Springer, Heidelberg, Germany.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany.

[EPT08] A Emrouznejad, B R Parker, and G Tavares. Evaluation of research in efficiency and productivity: A survey and analysis of the first 30 years of scholarly literature in DEA. *Socio-Economic Planning Sciences*, 2008.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229, New York City,, New York, USA, May 25–27, 1987. ACM Press.

[JNO14] Thomas P Jakobsen, Jesper Buus Nielsen, and Claudio Orlandi. A framework for outsourcing of secure computation. In *Proceedings of the 6th edition of the ACM Workshop on Cloud Computing Security*, pages 81–92. ACM, 2014.

[Mes97] L J Mester. What's the point of credit scoring? *Business review*, 1997.

[NT07] Kurt Nielsen and T Toft. Secure Relative Performance Scheme. *Proc. of Workshop on Internet and Network Economics 2007, LNCS 4858*, 2007.

[PAS04] Joseph C Paradi, Mette Asmild, and Paul C Simak. Using DEA and Worst Practice DEA in Credit Risk Evaluation. *Journal of Productivity Analysis*, 21(2):153–165, 2004.

[PBS09] I M Premachandra, Gurmeet Singh Bhabra, and Toshiyuki Sueyoshi. DEA as a tool for bankruptcy assessment: A comparative study with logistic regression technique. *European Journal of Operational Research*, 193(2):412–424, March 2009.

[Tof09] Tomas Toft. Solving linear programs using multiparty computation. In Roger Dingledine and Philippe Golle, editors, *FC 2009*, volume 5628 of *LNCS*, pages 90–107, Accra Beach, Barbados, February 23–26, 2009. Springer, Heidelberg, Germany.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164, Chicago, Illinois, November 3–5, 1982. IEEE Computer Society Press.