

Multiparty Computation Goes Live*

Anonymous Submission to ACM CCS 2008

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.

Categories and Subject Descriptors

C.2.0 [Computer-Communication Networks]: General—*security and protection*

General Terms

Theory, Security, Algorithms, Performance, Experimentation

Keywords

Protocol, Multiparty computation

1. INTRODUCTION AND HISTORY

Multiparty computation (MPC) is an extremely general subject, and a protocol enabling general secure multiparty computation is a very strong tool that can – in principle – solve almost any cryptographic protocol problem.

In MPC, we consider a number of players P_1, \dots, P_n , who initially each hold inputs x_1, \dots, x_n , and we then want to securely compute some function f on these inputs where $f(x_1, \dots, x_n) = (y_1, \dots, y_n)$ 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 π that the players execute. Intuitively, we want that executing π is equivalent to 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 . This “equivalence” can be formalized using, for instance, Canetti’s Universal Composability framework[5].

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

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 problem, both for general protocols [11, 17, 9] and for special types of computations such as voting [4, 12].

The authors of this paper have been involved in two research projects SCET (Secure Computing, Economy and Trust) and SIMAP (Secure Information Management and Processing) that have also aimed at improving the efficiency of multiparty computation, but this time with an explicit focus on a range of economic applications, which we believe are particularly interesting for practical use. In the economic field of mechanism design the concept of a trusted third party has been a central assumption since the 70-ties [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. In particular, we have considered:

- Various types of auctions. This is not limited to only standard highest bid auctions with sealed bids but also includes, for instance, variants with many sellers and buyers, so called double auctions - essentially scenarios where one wants to find a fair market price for a commodity given the existing supply and demand in the market.
- 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, we found that the com-

putation needed is basically elementary arithmetic on integers of moderate size, typically around 32 bits. More concretely, quite a wide range of the cases require only addition, multiplication and comparison of integers. The known generic MPC protocols can usually handle addition and multiplication very efficiently. What they really do is actually operations modulo some prime p , because the protocols are based on secret sharing over Z_p . So 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 we developed special purpose techniques for comparison. While comparison in constant-round with unconditional security is possible[13], we propose in this paper a logarithmic round solution that is much more practical for numbers of realistic size.

The SIMAP project goes a step further and has additionally developed a domain specific programming language `smcl` [20]. This language allows you to express the desired computation, and specify which information should be available to which players at any given time. Such a program can then be compiled to code that will run on the players’ machines and execute the appropriate protocols. The SIMAP project has been responsible for the application of MPC described in this paper.

2. 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, which is the only sugar producing company on the Danish market. Farmers have contracts that give them production rights, that is, a contract entitles a farmer to produce a certain amount of beets per year and deliver them to Danisco. These contracts can be traded between farmers, but trading has historically been very limited and has been done only 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. Details of the particular business case can be found in [2]. Here, we briefly summarize the main points: the goal is to find the so called *market clearing price*, 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 price¹. 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 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.

This could in principle be implemented with a single trusted party as the auctioneer. However, in our scenario, we have some additional security concerns implying that this is not a satisfactory solution: 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 future price negotiations, 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, contracts in some cases act as security for debt that farmers have to Danisco, and hence the farmers’ organization DKS running the auction independently would not be acceptable for Danisco. Finally, the common solution of delegating the legal and practical responsibility by paying e.g. a consultancy house to be the trusted auctioneer would be a very expensive solution.

It was therefore decided to implement an electronic double auction, where the role of the auctioneer would be played by a 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.

3. THE AUCTION SYSTEM

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, other than verifying their identity. This is in part because the method used for encrypting bids (described below) implicitly gives some protection: it is a variant of the non-interactive VSS based on pseudorandom secret sharing pre-

¹In 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.

sented in [14]. Using this method, an encrypted bid is either obviously malformed, or is guaranteed to produce consistently shares values. This means that the only cheating that is possible, is to submit bids that are not monotone, i.e., bids where, for instance, the amount you want to buy does not decrease with increasing price, as it should. It is easy to see that this cannot be to a bidders advantage. As a final word on the client-side security, 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 computation was based on standard Shamir secret sharing over a field $GF(p)$ where p was a 65 bit prime. Standard protocols with passive security were used for addition and multiplication, while a protocol described in more detail below was used for secure comparison. We settled for passive security because our most important goal was to avoid that any party would need access to bids in cleartext at any point, and passive security already achieves this.

The system worked with a set of 4000 possible values for the price, meaning that after the total supply and demand has been computed for all prices, the market clearing price could be found using binary search over 4000 values, which means 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 1200 bidders participated in the auction, each of these had the option of submitting a bid for selling, for buying, or both.

The actual computation was done January 14, 2008 and lasted about 30 minutes. Most of this time was spent on decrypting shares of the individual bids, which is not surprising, as the input to the computation consisted of about 9 million individual numbers.

As a result of the auction, about 25.000 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.

4. THE CRYPTOGRAPHIC PROTOCOLS

Abstracting away some of the details, the scenario we have is the following: a large number of clients deliver inputs to a multiparty computation, that is to be executed by n servers, where n is usually quite small (3 in our case). The

input from client i is an ordered list of nonnegative integers $\{x_{ij} | j = 1..P\}$, 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 to the servers, but we assume that public keys for the servers are available to the clients.

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

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

and to finally find the index j_0 for which $d_{j_0} - s_{j_0} = 0$, or at least is as close as possible to 0. Since supply increases and demand decreases with increasing price, this can be done by binary search over the indices $1..P$. In other words, 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 OK 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 follows 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 will buy or sell.

We chose to implement this based on standard Shamir secret sharing among the n servers, where we assume honest majority and a passive adversary. We used a prime field Z_p where p is chosen appropriately for the size of input numbers we expect (in our case p was 65 bits). By $[x]$ we denote a set of shares of the number x .

4.1 Submitting bids

The first issue is how the clients should supply the input numbers in shared form. The naive solution of simply secret sharing each x_{ij} and encrypt each share under the server's public key has the problem that servers have no way to check that the sharings created are consistent, unless we apply zero-knowledge proof techniques, which would have been completely impractical for the amounts of data we deal with. Cheating clients may not be a serious concern in our case, where users have a common interest in the auction functioning properly, but a more immediate practical problem is that the straightforward method 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 $n = 3$. We create 3 key pairs $(pk_i, sk_i), i = 1, 2, 3$ for a secure public-key cryptosystem, and give to server i the two keys sk_j where $j \neq i$. Now let $f_i(x), i = 1, 2, 3$ denote the polynomial 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, \dots, x_P in Z_p to the servers in encrypted form as follows:

1. Choose keys K_1, K_2, K_3 for a pseudorandom function

F that takes an index j as input and produces output in Z_p .

2. Output encryptions $E_{pk_i}(K_i), i = 1, 2, 3$.
3. For $j = 1..P$, compute and output

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

Each server ℓ 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 $i \neq \ell$.
2. Compute your share $share_{\ell,j}$ of x_j as follows: $share_{\ell,j} =$

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

bearing in mind that since $f_\ell(\ell) = 0$, it does not matter that you don't know K_ℓ .

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 $g_j(\ell) = share_{\ell,j}$ so that a valid set of shares has indeed been computed.

Generalizing this to an arbitrary number of servers and Shamir sharing with threshold t is straightforward: One associates a key pair (pk_A, sk_A) to every set of servers of size t , and sk_A is given to all servers not in A (the above is the special case with $t = 1$). Likewise, we use the polynomial f_A of degree at most t where $f_A(0) = 1$ and $f_A(i) = 0$ for all $i \in A$.

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, even if the client does not follow the protocol.
3. If a server loses its secret keys, they can be reconstructed by talking the other servers.

We assume below that the numbers in bids are significantly smaller than p , i.e., they have bit length less than half that of p . The method described above in principle allows a bidder to send numbers that are too large, possibly causing the computation to fail. It is possible to protect against this as well, namely by limiting the size of the pseudorandom values $F_{K_i}(j)$ so they are, for instance, half as long as p . Since the number to be shared is the sum of some pseudorandom values and a public value, we can now bound its size. However, the pseudorandom values are now no longer uniform mod p and therefore, to protect the privacy of honest bidder, they must be significantly longer than the integers to be shared. If honestly chosen bids contain 32 bit integers, a typical choice would be to have 64 bit long pseudorandom values and so p should be 129 bit. This increase in size of p implies a general loss of efficiency and an increase in size of data. On

the other hand, to actually cheat, a bidder would have to write his own client program, and we estimated that the risk of bidders cheating in this way was too small to motivate the extra cost of protecting against it.

4.2 Secure Comparison

It remains to describe how to compare the d_j and s_j 's, i.e. determine the branching conditions for the binary search. As noted above, the difficulty lies in the fact that the values are secret shared. Only abstract field arithmetic over Z_p is present, thus specifically there is no access to the binary representation of the numbers. A purely arithmetic solution is not feasible, so a different approach must be taken. As noted above, for efficiency, the comparison considers numbers of bounded size compared to the 65 bit modulus, p . With $\ell = 32$ bit d_j and s_j 's it is possible to perform the integer arithmetic needed for the comparison protocol.

When comparing values d and s , it is easy to see that the comparison result follows from the ℓ '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 is then solved, and from the solution, we compute the solution to the original problem.

4.2.1 Transforming the problem

We start by the servers generating $[r]$ for a random, secret shared value $r \in Z_p$ unknown to all, along with secret sharings $[r_i]$ of its binary representation, $r_{\ell+1+\kappa}, \dots, r_0 \in \{0, 1\}$, where κ is a security parameter controlling the statistical security of the protocol. We use $\kappa = 30$ in the implementation, as we must have $\ell + 1 + \kappa < \lceil \log p \rceil = 64$. Generating r is done by first generating bits randomly in shared form over Z_p , and constructing a sharing of r from this by taking a linear combination. This is done exactly as in [13]. We then convert each $[r_i]$ into a sharing over $GF(2^8)$, denoted $[r_i]_{256}$. We go to $GF(2^8)$ for efficiency reasons; xor on shared values will be used repeatedly in the following and in $GF(2^8)$ this is simply addition.

The conversion is done by having each server produce $[s_j], [b_j]_{256}$ for a random bit b_j , and random κ -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 $r_i \oplus b_1 \oplus \dots \oplus b_n$. Adding this bit to the shares of $[b_1 \oplus \dots \oplus b_n]_{256}$ produces $[r_i]_{256}$.

In the next step, the servers compute and reveal

$$a = (2^{\ell+1+\kappa} - r) + (2^\ell + d - s)$$

which is statistically secure since r was chosen to be random and κ bits longer than d, s . This computation can be seen as occurring over the integers, as the bit-lengths of the numbers prevent overflow.

We now have the transformed instance of our problem: given the public a and the set $[r_i]_{256}$, compute the ℓ 'th bit of

$$a + r = 2^{\ell+1+\kappa} + 2^\ell + d - s$$

It is straightforward to see that this bit will also give the solution to the original problem, but we are now in a more

favorable situation: a is public and the bits of the binary representation of r are secret shared.

4.2.2 Solving the binary problem

The desired result - the ℓ 'th bit of $-a+r$ - is simply the xor of a_ℓ , r_ℓ , and the ℓ 'th carry-bit c_ℓ , produced while adding a and r .

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 a_i at the most significant (left most) bit-position $i < \ell$ where $a_i = r_i$.

Consider now a list of ℓ pairs of bits, $\begin{pmatrix} a_i \oplus r_i \\ a_i \end{pmatrix}$. We can compute the answer using an operator on bit-pairs, \diamond , defined as

$$\begin{pmatrix} x \\ X \end{pmatrix} \diamond \begin{pmatrix} y \\ Y \end{pmatrix} = \begin{pmatrix} x \wedge y \\ x \wedge (X \oplus Y) \oplus X \end{pmatrix}.$$

It can be verified that this discards the x -pair when $x = 1$, otherwise the y -pair is discarded. We can therefore get the answer by computing

$$\begin{pmatrix} a_\ell \oplus r_\ell \\ a_\ell \end{pmatrix} \diamond \dots \diamond \begin{pmatrix} a_0 \oplus r_0 \\ a_0 \end{pmatrix}.$$

This computation is done over $GF(2^8)$, using the sharings $[r_i]_{256}$, since then \oplus does not cost communication and multiplications costs less communication since shares are shorter than those from Z_p ².

As \diamond is associative, we can do the \diamond operation on all ℓ pairs and hence find the comparison result for d_j and s_j in $\mathcal{O}(\log(\ell))$ rounds, using $\mathcal{O}(\ell)$ secure multiplications.

In [13] a (more complicated) constant-round solution was proposed. However, our solution is much more practical for the size of numbers in question: The diamond operator executes in a single round, so only $\log(32) = 5$ rounds are required for its repeated application. This implies less than 10 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 size in question.

5. 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 of course important what users think. In this connection, we can note the results of an on-line survey that was conducted simultaneously with the bidding phase. Here, about 80% of the respondents 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. We find that, in particular, the fact that confidentiality is seen as important is very interesting. Also

²In practice, we append a pair $\begin{pmatrix} 0 \\ 0 \end{pmatrix}$ on the right to handle the case where $a_i \neq r_i$ for all i .

Danisco and DKS have been satisfied with the system, and plan to run the auction again in coming years.

In judging the further potential of multiparty computation, it is important to ask what motivated, at the end of the day, 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. 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 system and when? who has responsibility if data leaks, and what are the consequences? Since the parties have conflicting interests, this could have lead to very lengthy discussions, possibly bringing the whole project to a halt. Alternatively, the parties might have found a solution in collaboration with, e.g., a consultancy house as mediator, but this would have been a more expensive solution, 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.

Our conclusion is that the ability of multiparty computation to keep secret *everything* that is not intended to be public, really is useful in practice, because 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 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. Again, 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 releases.

We therefore expect that multiparty computation will turn out to be useful in many practical scenarios in the future.

6. REFERENCES

- [1] (Author names removed for this anonymous submission) *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] Cramer, Gennaro and 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] Cramer and 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. Nielsen: *Universally Composable Efficient Multiparty Computation from Threshold Homomorphic Encryption*, Proc. of Crypto 2003, Springer Verlag LNCS.
- [12] Ivan Damgård, Mads Jurik: *A Generalisation, a Simplification and Some Applications of Paillier's Probabilistic Public-Key System*. Public Key Cryptography 2001: 119-136
- [13] Ivan Damgård, Matthias Fitz, Eike Kiltz, Jesper Buus Nielsen, Tomas Toft: *Unconditionally Secure Constant-Rounds Multi-party Computation for Equality, Comparison, Bits and Exponentiation*. Proc. of TCC 2006, pp. 285-304, Springer Verlag LNCS.
- [14] Damgård and 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] (Author names removed for this anonymous submission) *A domain-specific programming language for secure multiparty computation*, 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.