# An MPC-based Privacy-Preserving Protocol for a Local Electricity Trading Market

Aysajan Abidin, Abdelrahaman Aly, Sara Cleemput, and Mustafa A. Mustafa

KU Leuven, ESAT-COSIC and iMinds,
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
{firstname.lastname}@esat.kuleuven.be

**Abstract.** This paper proposes a decentralised and privacy-preserving local electricity trading market. The market employs a bidding protocol based on secure multiparty computation and allows users to trade their excess electricity among themselves. The bid selection and trading price calculation are performed in a decentralised and privacy-preserving manner. We implemented the market in C++ and tested its performance with realistic data sets. Our simulation results show that the market tasks can be performed for 2500 bids in less than four minutes in the "online" phase, showing its feasibility for a typical electricity trading period.

**Keywords:** Secure Multiparty Computation, Local Electricity Trading Market, Smart Grid, Renewable Energy Source, Security and Privacy.

## 1 Introduction

The Smart Grid (SG) is an electricity grid supporting bidirectional communication between components in the grid. An important component of SG is Smart Meters (SMs) which allow real-time grid management [1]. Potential benefits of SG include improved grid efficiency and reliability, and seamless integration of Renewable Energy Sources (RESs), e.g., solar panels, into the grid. When these RESs generate more electricity than their owners need, the excess electricity is fed back to the grid. Currently, users get some compensation from their suppliers for such excess electricity at a regulated (low) price. However, users with such excess electricity may be interested in selling directly to other users at a competitive price for monetary gains. Enabling that would also incentivise more users to own RESs. To address this, a local electricity market that allows RES owners to trade their excess electricity with other households in their neighbourhood has been proposed in [2]. However, such a market has user privacy risks, since users' bids/offers reveal private information about their lifestyle [3].

There are various proposals for an electricity trading market that allows users to trade with each other or suppliers [4, 5]. However, none of these addresses the privacy concerns. The security and privacy concerns in such a local market have been analysed in [2], and initial ideas (without a concrete solution) for designing one has been proposed in [6]. In this work, we not only propose a concrete secure and privacy-preserving solution for such a local market for trading electricity, but also test it in realistic scenarios.

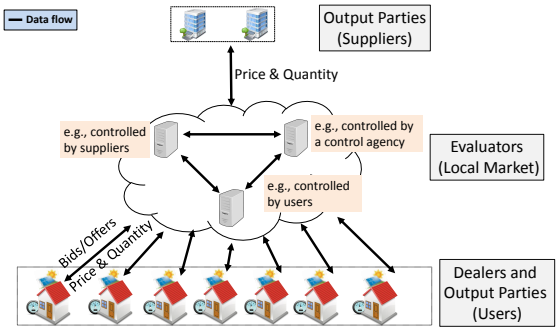


**Fig. 1.** A local MPC-based market for trading electricity from RESs.

**Contributions.** Our contributions are: (i) a concrete decentralised and privacy-preserving protocol for a local electricity trading market using MPC, (ii) a security and complexity analysis of our protocol, and (iii) an implementation, evaluation and analysis of the protocol using realistic bidding data sets.

## 2 Preliminaries

**System Model and Market Overview.** As shown in Fig. 1, a local electricity market comprises the following entities: RESs, SMs, users, suppliers and computational servers. The market operation, as proposed in [2], consists of:

- **Bid Submission:** Prior to each trading period, users submit their bids to the market to inform the market how much electricity they are willing to sell or buy during the trading period and for what price per unit.
- **Trading Price Computation:** The local market performs a double auction trading and generates the supply and demand curve. The intersection of these two curves is used to determine the trading price, amount of electricity traded, as well as which users will trade on the market.
- **Informing Users/Suppliers:** The market informs (i) the users about the amount of electricity they traded and the trading price, and (ii) the suppliers about the amount of electricity agreed to be traded by their respective users.

**Threat Model and Assumptions.** Users and suppliers are malicious. They may try to modify data sent by SMs in an attempt to gain financial advantage or influence the trading price on the market. Computational servers are honest-but-curious. They follow the protocol specifications, but they may attempt to learn individual users' bids. External entities are malicious. They may eavesdrop data in transit and/or modify the data in an attempt to disrupt the market. In addition, we make the following assumptions: (i) each entity has a unique identity, (ii) SMs are tamper-evident, (iii) all entities are time synchronized, (iv) the communication channels between entities are secure and authentic, and (v) users are rational, i.e., they try to buy/sell electricity for the best possible price.

**Functional and Privacy Requirements.** Our protocol should meet the following functional requirements: (i) the local market should receive users' bids, calculate the trading price, and inform the users and suppliers of the market outcome, (ii) each user should learn if their bid was accepted and the vol. of electricity they traded, as well as the trading price, and (iii) each supplier should learn the amount of electricity traded by their customers on the market in each trading period. It should also satisfy the following privacy requirements [2, 6]: (i) confidentiality of users' bids and amount of electricity traded, (ii) users' privacy preservation, i.e., RES and/or trading user identity and location privacy, and trading session unlinkability, and (iii) minimum data disclosure.

**Security Definition under MPC.** MPC allows any set of mutually distrustful parties to compute any function such that no party learns more than their original input and the computed output, i.e., parties $p_1, ..., p_n$ can compute $y = f(x_1, ..., x_n)$, where $x_i$ is the secret input of $p_i$, in a distributed fashion with guaranteed correctness such that $p_i$ learns only $y$. MPC can be achieved using secret sharing schemes [7, 8], garbled circuits [9] and homomorphic encryption [10].

On the security notion: a secure protocol over MPC discloses to an adversary the same information as if the computations were carried out by a trusted (non-corruptible) third party. This definition allows a variety of adversarial and communication models offering various security levels: perfect, statistical or computational. Seminal results prove that any functionality can be calculated with perfect security against active and passive adversaries [7, 8] under the arithmetic circuit paradigm. Other relevant recent contributions in the area include [11, 12]. Note that any oblivious functionality built in this way would be as secure as the underlying MPC protocols used for its execution. Finally, note that under this scenario, functionality, also referred to as sub-protocols, like the ones used in this work, can be used for modular composition under the hybrid model introduced by Canetti [13]. We make use of the following existing functionality:

- **Secure Comparison:** Methods for secure comparison using MPC offer either perfect or statistical security and are constructed under the same assumptions [14]. Moreover, mechanisms as [15] by Catrina and de Hoogh introduced inequality tests at constant complexity.
- **Secure Sorting:** Secure Sorting using MPC can be achieved by sorting networks and other data-oblivious mechanisms, including the randomize shellsort from Goodrich [16]. Moreover, Hamada et al. [17] introduced a technique to facilitate the use of comparison sorting algorithms. This technique consists of randomly permuting the vector before sorting, so that the results of some of the intermediate secure comparisons can be made public.
- **Secure Permutation:** Leur et al. [18] analysed various permutation mechanisms, like the use of vector multiplication by a permutation matrix and sorting networks. Czumaj et al. [19] proposed alternatives for obliviously permuting a vector in (almost) $\mathcal{O}(n \times log(n))$, when $n$ is the vector size.

**Table 1.** Notation.

| Symbol | Meaning |
|---|---|
| $t_i$ | $i$-th time slot |
| $[q]_j$ | electricity volume in absolute terms from the $j$-th bid |
| $[p]_j$ | unit price enclosed in the $j$-th bid |
| $[d]_j$ | binary value corresponding to the $j$-th bid: 1 indicates a demand bid and 0 a supply bid |
| $[s]_j$ | unique supplier identifier $s \in \{1, .., \|S\|\}$ where $S$ is the set of all suppliers. Moreover, $s$ is encoded in a $\{0, 1\}$ vector, i.e, $[s]_{jk} \leftarrow 1$ on the $k$-th position corresponds to the suppliers unique identifier, and $[s]_{jk} \leftarrow 0$ otherwise, for all $j \in B$. |
| $[b]_j$ | bid's unique identifier from the $j$-th bid |
| $[\phi]$ | volume of electricity traded on the market for period $t_i$ |
| $[\sigma]$ | market's trading price (price of the lowest supply bid) for $t_i$ |
| $[a]_i$ | binary value: 1 indicates the bid $i$ was accepted, 0 otherwise |
| $[S]^{\phi}$ | set of the volume of electricity traded by supplier affiliation where $[s]_i^{\phi}$ stands for the summation of all the accepted bids from users affiliated to the supplier $i$, for all $i \in S$ |

**Notation.** Square brackets denote encrypted or secretly shared values. Assignments that are a result of any securely implemented operation are represented by the infix operator: $[z] \leftarrow [x] + [y]$. This extends to any operation over securely distributed data since its result would be of a secret nature too. Vectors are denoted by capital letters. For a vector, say $B$, $B_i$ represents its $i$-th element and $|B|$ its size. The bids originated by SMs are considered as the initial input data. Each bid is a tuple $([q], [p], [d], [s], [b])$ and $B$ is the vector of all bids. We assume that (i) all bid elements belong to $\mathbb{Z}_M$, where M is a sufficiently large number so no overflow occurs, and (ii) the number of bids (or at least their upper bound) is publicly known. Any other data related to the bid is kept secret. If the protocol admits a single supply and demand bid per SM, the computation of this upper bound is trivial. Markets could opt for enforcing all SMs to submit a bid regardless of whether they participate or not in the market. Let $\top$ be a sufficiently big number such that it is greater than any input value from the users but $\top << M$. In this scenario, non-participating SMs would have to replace their input values by $[0]$ and $[\top]$ accordingly. Table 1 lists the notations.

## 3 Privacy-preserving Protocol for Electricity Trading

In our protocol, users submit their private inputs to a virtualized entity consisting of multiple computational servers that function as evaluators. The number of evaluators depend on the application, and it could be as many as the number of parties involved in the computation. However, this is costly in terms of performance. In our setting, we assume three computational parties: one comes from the RES owners, another from the suppliers and a third one from a local control agency. Depending on the underlying MPC protocol, some randomizations might be precomputed in an "offline" phase by a trusted dealer who is not directly involved at any level of the computations [11]. The amount and purpose of the randomly generated numbers depend on such MPC primitives and the security model used by the market. Our protocol consists of five steps:

**Agorithm 1:** Smart Market Clearance.

**Input**: Vector of $n$ bid tuples $B = ([q], [p], [d], [s], [b])$
**Output**: Clearance price $[\sigma]$, volume of traded electricity $[\phi]$, vector of accepted bids $[A]$ of size $|B|$, vector of aggregated volume traded by supplier $S^{\phi}$ of size $|S|$

```
for i ← 1 to n do
    [δ] ← [δ] + [q]_j × [d]_j;
end
[ν] ← [0];
[S^φ] ← {0_1, ..., 0_|S|};
[A] ← {0_1, ..., 0_|B|};
for k ← 1 to n do
    [c] ← [ν] < [δ];
    [σ] ← ((1 − [d]_j) × [c]) × ([p]_j − [σ]) + [σ];
    [φ] ← ((1 − [d]_j) × [c]) × [q]_j + [φ];
    for k ← 1 to |S| do
        [s]^φ_k ← ([s]_jk × ((1 − [d]_j) × [c]) × [q]_j + [s]^φ_k;
    end
    [a]_j ← [c];
    [ν] ← [ν] + [c] × [q]_j;
end
```

**Preprocessing for trading period $t_i$.**

1. `Bidders:` Before the start of $t_{i-2}$, each user prepares and sends his bid to the computational parties. If a linear secure secret sharing scheme (e.g., [20]) is used, each user generates as many shares as the number of computational parties, and sends each of the shares to a different computational party.
2. `Evaluators:` To randomly permute the bidders' input, upon reception, each share is multiplied with a column of a randomized permutation matrix which was precomputed "offline". This is still performed before the start of $t_{i-2}$.

**Evaluation for trading period $t_i$.**

3. `Evaluation:` The evaluation is performed at $t_{i-2}$. In this phase, the trading price and traded volume are computed, and accepted and rejected bids are identified, in a data-oblivious fashion. Algorithm 1 gives a detailed overview of our secure auction evaluation. It calculates the trading price $[\sigma]$, the volume of electricity traded $[\phi]$ and the vector of adjudicated demand and supply bids $[A]$. It does it by obliviously calculating the aggregation of the demand bids $[\delta]$, and then iterating over the set of all bids in $B$ using their volume to match $[\delta]$. To access the vector of accepted supply bids, it is enough to compute $[A]_j \times (1 - [d]_j) \times [b]_j$. To find the vector of accepted demand bids, it is sufficient to calculate $(1 - [A]_j) \times ([d]_j) \times [b]_j$.

**Inform Bidders and Suppliers (before the end of period $t_{i-2}$).**

4. `Bidders:` To hide the order of the bids, the vector of all bids $[B]$, together with the associated vector $[A]$, are shuffled again. Then, the evaluators use the `open` operation of the underlying MPC primitive on $[\sigma]$ (for $t_i$) and $[b]_j$, for all $j \in B$. Each evaluator sends the shares corresponding to the tuple $B_{b_j}$ to the bidder that originated the bid identified by $b_j$. The bidder then reconstructs the shares and learns if his bid was accepted or rejected.
5. `Suppliers:` Evaluators send the shares of the volume aggregation $S^{\phi}_j$, for all $j \in S$, to the corresponding supplier. Suppliers also learn the market trading price. Both, bidders and suppliers are informed of the results at $t_{i-2}$.

**Correctness and Complexity.** The general goal of the protocol is to find the trading price and to identify the accepted and rejected bids. Any supply bid below the trading price, and any demand bid above this price is automatically accepted and vice versa. The market equilibrium can be identified when the price of a given supply allocation surpasses the price of the next cheapest available demand allocation. In other words, when supply equals demand, the market equilibrium can be identified if the price of supply is at least the price of demand.

In our protocol, we proceed to sort all bids regardless of whether they are demand or supply bids. Following Algorithm 1, we then proceed to identify and select bids until the aggregated demand ($[\delta] \leftarrow \sum_i^{|B|} [q]_i \times [d]_i$) is matched (note that to maintain secrecy we iterate over the set of all bids), choosing the bids in ascending order of price. If a supply bid is selected, this implies that there is no supply bid that could be allocated to reduce $[\delta]$, and hence is not part of the market clearance. Using $[d]_i$ cancels the supply bid's effect over $[\delta]$, and provides us with sufficient tools to identify it. The opposite occurs when a demand bid is selected. At the end of Algorithm 1, the bids used to reduce $[\delta]$ can be identified, which correspond to all the supply and demand bids with prices below and above the trading price, respectively. From this, the set of accepted and rejected bids follows. The trading price is set to the price of the last selected supply bid. The protocol complexity grows linearly with the number of bids, which is the main factor influencing the performance. The number of suppliers rarely varies over time, and is of limited size. The complexity of Algorithm 1 is $\mathcal{O}(|B| \times |S|)$. Note that secure vector permutation can be achieved in $\mathcal{O}(n \times log(n))$, where $n$ is the size of the vector (the vector of the Bids $[B]$, in our case). Moreover, the sorting methods used by our secure market can achieve $\mathcal{O}(n \times log(n))$.

**Security Analysis.** The MPC mechanisms used in protocol steps `1-5` constitute a unique arithmetic circuit (addition and multiplication) with no leakage, making privacy straight forward. Moreover, the protocol can be computed with perfect security on the information theoretic model against passive and active adversaries under Canetti's hybrid model [13] by using available MPC protocols such as BGW [7]. We refer the reader to [21] for a complete set of proofs of security and composability for BGW. Indeed, results in BGW [7] and CDD [8] showed that any function can be computed using MPC with the aforementioned security levels by providing secure addition and multiplication under an arithmetic circuit paradigm. There are also promising results on more restricted models, e.g., dishonest majority [11] with computational security. Moreover, there exist privacy-preserving sub-protocols (arithmetic circuits) for sorting, comparison and vector permutation over MPC that can be used, and that provide the same security guarantees with no leakage. These are integrated into a single arithmetic circuit in a modular fashion, i.e., our protocol. Thus, the security of our protocol readily follows. In other words, the order of the operations (multiplications and additions) is predetermined beforehand by the publicly available circuit, i.e., our protocol simulation can be achieved by invoking the corresponding simulators of the sub-protocols used, and/or atomic operations in its predefined order.

## 4 Experimentation and Discussion

We executed our experimentation using the BGW-based MPC Toolkit [22] which includes all the underlying crypto primitives and sub-protocols we report, together with our own introduced code. The library was compiled with NTL (Number Theory Library) [23] that itself was compiled using GMP (GNU Multiple Precision Library). These two libraries are used for the modulo arithmetic that is used by the underlying MPC protocols. Each instance of the prototype comprises two CPU threads: one manages message exchanges and the other executes the protocol. Moreover, each instance required little more than 1 MB of allocated memory during our most memory demanding test.

**Data Generation.** We generated the data using a realistic data from Belgium. First we picked a time slot and date, i.e., between 13:00h and 13:30h on 5-th of May 2016, during which 2382 MW solar electricity was generated in Belgium by Solar Panels (SPs) with total capacity 2953 MW [24], i.e., on average each SP produced electricity approximately equal to 81.66% of its capacity. The average electricity consumption data of a Belgian household for the same time slot was 0.637 kW [25], so for each user we generated a random consumption data for this slot with mean equal to 0.637 kW, standard deviation equal to 0.20 and variance equal to 0.04. Then, we randomly chose 30% of the users to have installed SPs at their homes, and to each of the SPs we randomly assigned 2.3, 3.6 or 4,7 kW electricity generation capacity. After that, we randomly generated the electricity output of each SP during this time slot with a mean equal to the SP's capacity multiplied with the efficiency factor for the time slot, i.e., 81.66%, standard deviation equal to 0.20 and variance equal to 0.04. Once we generated the electricity consumption and generation data for each user with a SP, we simply subtracted the latter from the first value to find the amount of each user's excess electricity.

We assumed that there are 10 suppliers in the market and randomly assigned one to each user. We set the retail electricity sell price of the suppliers to 0.20 €/kWh and the retail buy price to 0.04 €/kWh. For the bid price selection, we divided the retail electricity sell and buy price difference into nine ranges each including several (overlapping) prices, e.g., range 2 includes three prices: 0.04, 0.05 and 0.06 €/kWh, whereas range 7 includes four prices: 0.17, 0.18, 0.19 and 0.20 €/kWh. Then, for each user, depending on how much excess electricity she has for sell (or wants to buy), we picked randomly one of the prices from the appropriate price range. For selecting the appropriate price range we assumed that if users have a lot of excess electricity, they would choose a lower asking price, but if they have a little, they would ask for a higher price. In summary, for each user we generated: unique user ID, amount of electricity for the bid, bid price, supply or demand bid indicator, and ID of the user's contracted supplier.

**Security.** Our security target was to build a prototype for the classic scenario of semi-honest adversaries under the information theoretic model (private authenticated channels) and threshold corruption. This is achieved by the underlying

**Table 2.** List of Primitives used by secure prototype.

| Primitive | Protocol |
|---|---|
| Sharing | Shamir Secret Sharing [20] |
| Multiplication | Gennaro et al. [26] |
| Inequality Test | Catrina and Hoogh [15] |
| Random Bit Generation | Damgård et al. [14] |
| Sorting: QuickSort | Hamada et al. [17] |
| Permutation: Sorting Network | Lai et al. [18] |

BGW primitives and Shamir Secret Sharing (honest majority). This is a necessary configuration to achieve perfect security as long as the adversary does not corrupt more than halve of the parties. However, the prototype offers statistical security on the size of its input given that it uses the same comparison method as in [15]. The security of such method depends on input parameters $l$ and $k$, $l$ is the bit-size of the numbers and $k$ a security parameter. Under the assumption that the channel is perfect, this task is decoupled from the prototype operation.

**Characteristics, Environment and Setting.** Our prototype was built in C++ following an object oriented approach, with modularity and composability in mind. It has an engine that separates communication and cryptographic tasks. Table 2 shows the list of the sub-protocols we used. We executed our tests on a single 64-bit Linux server with 2*2*10-cores with Intel Xeon E5-2687W microprocessors at 3.1GHz and 25 MB of cache available, and with memory of 256 GB. All our tests were performed under a 3-party setting, with two available cores for each instance. We ran our tests starting with a baseline of a realistic scenario with 100 bids and then monotonically increased the number of bids to 2500. Each test scenario was repeated 10 times to reduce the impact of the noise.

**Results.** Our prototype requires bit randomization for the comparison methods. The task of generating such values could be executed beforehand, in an "offline" phase. The "online" phase would execute the remaining tasks and utilize the randomization values generated during the "offline" phase. For a case with 2500 bids, the prototype took 678.50 sec. for either sending or waiting for other parties' messages (as our prototype is synchronous) and 215.52 sec. for other computational tasks (crypto primitives). Hence, ca. 75% of the computational time was for transmission related tasks. We have also measured the computational cost at every test instance. Table 3 shows a more complete break down of our results. From these results we can conclude the following.

- The 2500-bids instance total time on the "online" phase is less than 4 minutes, and less than 15 minutes with the "offline" phase included, which is still less than a typical trading period of 30 minutes.
- The asymptotic behaviour on the growth of the computational time seems to adjust to the behaviour included in the complexity analysis.
- The performance of the prototype could be improved by the use of techniques such as, PRSS [27], to reduce the cost of generating random bits. Moreover, other optimizations can be put in place based on the experimental setting.

**Table 3.** Overall Results

| Bids | Com. Rounds | Comparisons | CPU Time (sec) | On-line Phase (sec) |
|---|---|---|---|---|
| 100 | $\approx 1.40 \cdot 10^5$ | 965 | 2.96 | 1.01 |
| 500 | $\approx 1.96 \cdot 10^6$ | 14628 | 40.40 | 11.35 |
| 1000 | $\approx 7.03 \cdot 10^6$ | 53508 | 147.76 | 39.80 |
| 1500 | $\approx 15.61 \cdot 10^6$ | 118956 | 320.79 | 86.14 |
| 2000 | $\approx 26.97 \cdot 10^6$ | 208132 | 562.50 | 145.78 |
| 2500 | $\approx 43.15 \cdot 10^6$ | 330912 | 894.01 | 235.82 |

– During our tests ca. 95% of the computational time was spent on sorting the bids. As suppliers are not involved in this, their influence on the computational costs is limited, i.e., our prototype can be adjusted to scenarios with larger supplier sets without much overhead.

## 5 Conclusions

We proposed a privacy-preserving protocol for a local market that allows users to trade their excess electricity among themselves. Our protocol employs a bidding scheme based on MPC, and the bid selection and the trading price calculation are performed in a decentralised and privacy-preserving manner. We also implemented the protocol in C++ and tested its performance with realistic data. Our simulation results show its feasibility for a typical electricity trading period of 30 minutes as the market tasks are performed (for 2500 bids) in less than 4 minutes in the "online" phase. Future work will include balancing suppliers' accounts based on the electricity traded by users without violating users' privacy.

**Acknowledgments.** This work was supported by KIC InnoEnergy SE via KIC "SAGA" project, European Commission FP7 project "EKSISTENZ" grant number: 607049, and the European Commission through the ICT programme under contract FP7-ICT-2013-10-SEP-210076296 (PRACTICE).

## References

1. Farhangi, H.: The path of the smart grid. IEEE Power and Energy Magazine **8**(1) (January 2010) 18–28
2. Mustafa, M.A., Cleemput, S., Abidin, A.: A local electricity trading market: Security analysis. In: IEEE PES ISGT-Europe. (2016) 1–6
3. Hart, G.W.: Nonintrusive appliance load monitoring. Proceedings of the IEEE **80**(12) (1992) 1870–1891
4. Lee, W., Xiang, L., Schober, R., Wong, V.W.S.: Direct electricity trading in smart grid: A coalitional game analysis. IEEE Journal on Selected Areas in Communications **32**(7) (July 2014) 1398–1411
5. Tushar, W., Yuen, C., Smith, D.B., Poor, H.V.: Price discrimination for energy trading in smart grid: A game theoretic approach. IEEE Transactions on Smart Grid **PP**(99) (2016) 1–12

6. Abidin, A., Aly, A., Cleemput, S., Mustafa, M.A.: Towards a local electricity trading market based on secure multiparty computation. http://securewww.esat.kuleuven.be/cosic/publications/article-2664.pdf (2016)
7. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: STOC, ACM (1988) 1–10
8. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols. In: STOC, ACM (1988) 11–19
9. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: STOC, ACM (1987) 218–229
10. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: EUROCRYPT '99. Volume 1592 of LNCS., Springer (1999) 223–238
11. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: CRYPTO '12. Volume 7417 of LNCS., Springer (2012) 643–662
12. Bendlin, R., Damgård, I., Orlandi, C., Zakarias, S.: Semi-homomorphic encryption and multiparty computation. In: EUROCRYPT '11. Volume 6632 of LNCS., Springer (2011) 169–188
13. Canetti, R.: Security and composition of multiparty cryptographic protocols. Journal of Cryptology **13**(1) (2000) 143–202
14. Damgård, I., Fitzi, M., Kiltz, E., Nielsen, J.B., Toft, T.: Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In: TCC 2006. Volume 3876 of LNCS., Springer (2006) 285–304
15. Catrina, O., de Hoogh, S.: Secure multiparty linear programming using fixed-point arithmetic. In: ESORICS 2010. Volume 6345 of LNCS., Springer (2010) 134–150
16. Goodrich, M.T.: Randomized shellsort: A simple data-oblivious sorting algorithm. J. ACM **58**(6) (December 2011) 27:1–27:26
17. Hamada, K., Kikuchi, R., Ikarashi, D., Chida, K., Takahashi, K.: Practically efficient multi-party sorting protocols from comparison sort algorithms. In: ICISC '12. Volume 7839 of LNCS., Springer (2003) 202–216
18. Laur, S., Willemson, J., Zhang, B.: Round-efficient oblivious database manipulation. In: ISC 2011. Volume 7001 of LNCS. Springer (2011) 262–277
19. Czumaj, A., Kanarek, P., Kutylowski, M., Lorys, K.: Delayed path coupling and generating random permutations via distributed stochastic processes. In: SODA '99, SIAM (1999) 271–280
20. Shamir, A.: How to share a secret. Commun. ACM **22**(11) (1979) 612–613
21. Asharov, G., Lindell, Y.: A full proof of the bgw protocol for perfectly secure multiparty computation. Journal of Cryptology (2015) 1–94
22. Aly, A.: Network Flow Problems with Secure Multiparty Computation. PhD thesis, Universté catholique de Louvain, IMMAQ (2015)
23. Shoup, V.: Ntl: A library for doing number theory (2001)
24. www.elia.be/en/grid-data/power-generation/Solar-power-generation-data
25. http://vreg.be/nl/verbruiksprofielen-elektriciteit
26. Gennaro, R., Rabin, M.O., Rabin, T.: Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In: PODC '98, ACM (1998) 101–111
27. Cramer, R., Damgård, I., Ishai, Y.: Share conversion, pseudorandom secret-sharing and applications to secure computation. In: TCC 2005. Volume 3378 of LNCS. Springer (2005) 342–362