Privacy for the Stock Market[⋆]

Giovanni Di Crescenzo

Telcordia Technologies Inc.
445 South Street, Morristown, NJ, 07960
giovanni@research.telcordia.com

Abstract. We investigate the problem of performing Stock Market operations, such as buying or selling shares of a certain stock, in a *private* way, which had recently been left open.
We present a formal definition for a private stock purchase protocol, addressing several privacy and security concerns on usual on-line stock market operations. According to our definition, a client would not reveal how many shares she is buying or selling (not even which of these two cases is happening), and what price she is offering for those. We then present an efficient protocol meeting this definition, based on the hardness of the decisional Diffie–Hellman problem. Our protocol requires no interaction between the clients, can be executed in a constant number of rounds between the clients and the server, and requires several technical contributions, such as a new and efficient zero-knowledge protocol for proving sum-related statements about encrypted values, which is of independent interest.

1 Introduction

The overwhelming expansion of the internet is today being accompanied with a large increase of financial activities and transactions that are conducted online. A few minutes navigation on the internet allows to realize the existence of electronic cash systems, payment protocols, auctions, lotteries, digital casinos and gambing systems. The sometimes crucial importance and often large interest around such transactions raises several concerns about the security and the privacy of the information that users and organizations are willing to use on a network.

In this paper we consider a important financial transaction: buying and selling shares of a particular stock on the Stock Market. Such transactions seem to have received not enough attention from the security and privacy literature, and, in fact, an assessment of the privacy problems deriving from these transactions and the construction of a protocol which addresses them had been left as an open problem. We present a formal definition for what it means for a stock purchase protocol to be secure and private, and present an efficient protocol which allows to privately purchase and sell shares of a certain stock.

[⋆] Copyright 2001, Telcordia Technologies, Inc. All Rights Reserved.

Our Model and Definition. We consider a model composed of several clients who intend to purchase or sell shares of a particular stock, and a server, taking care of such operations and of the current share price and current share amount. While the server is assumed to behave honestly (or, more precisely, as a honest-but-curious party), the clients may behave in a malicious way. Therefore, we ask that clients are allowed to perform their operations without revealing to all other clients private information such as how many shares they are buying or selling, which prize they offer for those, and not even whether they are buying or selling. In fact, even the server cannot derive any information about these operations other than what he needs to update the current share amount (namely, the sum of all shares bought/sold) and the current share price (namely, some prespecified function of external information and all private inputs of the clients). Still, at the end, each client who behaves honestly should obtain from the server a certificate for that particular transaction, despite the behavior of all dishonest clients.

Our Results. We present a protocol that satisfies all the mentioned properties and can be implemented in a constant number of rounds. In the important case in which the function that updates the share price is linear in the private amounts of the clients, our protocol has an efficient implementation based on the hardness of the Decisional Diffie Hellman problem. We also present a general solution for arbitrary functions, which uses any 2-party secure function evaluation protocol for computing the same function. Even in this case our solution can be implemented in a constant number of rounds; moreover, a variant of our solution also keeps the function private from the client. Some technical contributions include novel and efficient zero-knowledge protocols for proving sum-related statements about encrypted values, that may find applications somewhere else. In particular, we show a protocol for proving in zero-knowledge that given three ciphertexts, encrypted according to the El-Gamal cryptosystem, the first plaintext is the modular sum of the remaining two. Although we only present an efficient implementation of it, we note that our protocol can be implemented by only assuming the existence of any oblivious transfer protocol.

Related Results . Our model of a honest-but-curious server and several clients who do not need to interact has been already used in several investigations on the topic of auctions [9,4,23,24,16] (other related investigations on auctions which do not use a trusted party have been done in [26]). In particular, the paper [16] posed the open question of investigating privacy in stock market operations. The problem of anonymity in stock market operations has been investigated in [18]. More generally, our model and solution can be considered as belonging to the area of designing efficient protocols for specific multi-party private computation problems (as, for instance, for threshold cryptography [7]).

Organization of the Paper. In Section 2 we present some background on number theory and various cryptographic primitives. In Section 3 we present a detailed definition of the requirements that a private stock purchase protocol has to satisfy, and a high-level description of our solution. The description of the main two subcomponents of our private stock purchase protocol, is divided into two sections: in Section 4 we present the capital update (sub)protocol and in Section 5 we present the price update (sub)protocol.

2 Preliminaries

In this section we review some background notions and protocols as: oblivious transfer (OT), zero-knowledge proofs, the El-Gamal encryption scheme and an OT protocol based on it.

The El Gamal Public-Key Encryption Scheme [12]. Let p be a prime such that $p-1$ has a large prime factor q, and let g be a generator of a subgroup G of Z_p of order q. The key generation algorithm of the El Gamal public-key encryption scheme consists in uniformly choosing $s \in Z_p$ and computing $h = g^s \bmod p$, publishing (p, q, g, h) as a public key and keeping s as a secret key. The encryption algorithm consists of uniformly choosing $r \in Z_p$ and returning (u, v), where $u = g^r \bmod p$ and $v = h^r m \bmod p$, and m is the message. The decryption algorithm consists of outputting m, computed as $m = v/u^r \bmod p$. The semantic security of this proof system is equivalent to the difficulty of deciding the Diffie–Hellman problem.

Oblivious Transfer. The notion of Oblivious Transfer (OT) protocol was introduced by Rabin [21]. Informally, an OT protocol can be described as a game between two polynomial time parties Alice and Bob, where Alice wants to send a message to Bob in such a way that with probability 1/2 Bob will receive the same message Alice wanted to send, and with probability 1/2 Bob will receive nothing. Moreover, Alice does not know which of the two events really happened. There are other equivalent formulations of Oblivious Transfer (see, e.g., [5]), such as 1-out-of-2 OT, in which Alice has two messages m_0, m_1, Bob has one bit c and at the end Bob will receive m_c, without receiving any information about m_{1-c} and without Alice guessing c. This primitive has found numerous implementations and applications in the cryptographic literature. For our constructions, we can use any oblivious transfer protocol based on the difficulty of deciding the Diffie–Hellman problem (a two-round protocol for this task appears in [19]).

Conditional Oblivious Transfer. The notion of conditional oblivious transfer was introduced in [10], where applications were given to the problem of timed-release encryption, or 'sending information to the future'. Informally, a conditional oblivious transfer protocol is the following generalization of ordinary (1-out-of-2) oblivious transfer: Alice and Bob also have a private input (call those x_A and x_B, respectively), and a public predicate ρ such that if $\rho(x_A, x_B) = 1$ (resp., $\rho(x_A, x_B) = 0$) then Bob receives m_0 (resp., m_1), without learning any additional information about the message he has not received. In [10] a conditional oblivious transfer was given for the predicate 'greater than or equal to', based on the intractability of deciding quadratic residuosity. We remark that the same techniques used in [10] for this protocol can be used to construct an implementation based on the difficulty of deciding the Diffie–Hellman problem.

Zero-Knowledge Proof Systems. Informally, zero-knowledge proof systems [15] are interactive protocols allowing a possibly infinitely powerful prover to convince a polynomial time verifier that a statement (e.g., the membership of a string x to a language L) holds without revealing any additional information that the verifier could not compute alone before running the protocol. Now we expand on the definition of such protocols. First of all, an *interactive proof system* for a language L is an interactive protocol satisfying the two requirements of *completeness* and *soundness.* The completeness requirement says that if the prover and the verifier follow the protocol, then the verifier has to accept with probability very close to 1. The soundness requirement says that if the verifier follow the protocol, then no matter which arbitrarily powerful strategy is used by the prover, the verifier accepts with probability very close to 0. Then, a *zero-knowledge proof system* for a language L is an interactive proof system for L satisfying the additional requirement of *zero-knowledge.* This requirement states that for any probabilistic polynomial time strategy used by the verifier, there exists an efficient algorithm S, called the simulator, such that for all $x \in L$, the following two distributions are "indistinguishable": 1) the output of S on input x, and 2) the messages seen by the verifier when interacting with the prover on input x (including the verifier's random tape). According to the specific formalization of indistinguishability, we obtain different variants of the zero-knowledge requirement, called *computational, statistical* and *perfect.* A zero-knowledge *argument* [3] is a zero-knowledge proof system for which the soundness is only required to hold under polynomial-time adversaries.

3 Private Stock Purchase Protocol: Definition and Solution Sketch

In this section we present a definition and a high-level view of our solution for the main protocol of interest in this paper: a private stock purchase protocol.

3.1 A Formal Definition

We start by presenting the players, the phases and the (sub)protocols involved in an execution of such protocol, and then describe the requirements that a private stock purchase protocol has to satisfy.

Players. The players involved in a private stock purchase protocol are the *clients*, denoted as $C_1, C_2, \ldots, C_n$ and a *server*, denoted as S. A client C_i is any individual that intends to buy stock shares. The server S is the machine (or the individual) that takes care of handling the stock shares; including, for instance, selling the shares, updating the number of shares, updating the share price.

Connectivity. Although potentially all clients are connected among them and to the server through some communication link, for practical purposes, we are especially interested in protocols where each client only interacts with the server, and not necessarily at the same time.

Phases. Generally speaking, the lifetime of a stock purchase payment protocol is divided into a large number of time intervals of fixed and known length. In each of these intervals, a set of clients registers with the servers, requests a number of stock shares at a certain offered price, the share price is consequently updated, the number of available shares is properly updated as well, and clients are eventually given the requested number of shares. Since the execution of the protocol is conceptually the same in each interval, from now on we concentrate our study on a single, generic, time interval, and, for simplicity, refer to the protocol executed in this interval as the private stock purchase protocol. In this protocol we distinguish four phases. A first phase, called the *registration phase*, consists of each individual registering herself as a client by committing to the number of stock shares that she is willing to buy or sell and to a price they she is willing to pay for each of them, which we will call the 'offered price'. A second phase, called the *price update phase*, contains an interactive protocol in which each client interacts with the server; at the end the price of the stock shares is updated as a known and polynomial-time computable function of the amounts of shares and the offered prices that clients committed to in the first phase. A third phase, called the *capital update phase*, contains an interactive protocol in which each client interacts with the server; at the end the number of stock shares is eventually updated by subtracting the number of shares bought and by adding the number of shares sold; we call the resulting number the 'new price'; note that the shares for which the offered price was lower (higher) than the new price are not sold to (bought by) the client. In the fourth phase, called the *certification phase*, each client finally obtains from the server some certification of the transaction, namely, a certificate that a certain number of shares (if any) have been bought or sold by that client.

Protocols. Each phase contains an interactive protocol, where the interaction is between all clients and the server only (i.e., there is no interaction between the clients). Specifically, the *registration protocol* is executed between each client and the server, and at the end returns some keys and parameters that will be used in the rest of the payment protocol. The *price update protocol* is executed between all registered clients and the server, and at the end the server knows the output of the price updating function over the clients' private inputs (namely, the number of shares they want to buy and the intended prices) without learning any new information about such inputs; such output will be the new share price of the stock in the new time interval. The *capital update protocol* is executed between all registered clients and the server, and at the end the server knows a partial sum of all clients' private inputs. Here, a private input contributes a positive (resp., negative) amount if the client wants to buy (resp., sell) that amount of shares at an offered price larger (resp., smaller) than the new price; note that clients intending to buy (resp., sell) shares at an offered price smaller (resp., larger) than the new price will contribute no value to this sum. The *certification phase* is executed by the server who sends a single message to each client who successfully executed the previous phases, containing a certification of the client's acquisition or deposition of shares (if any) for the amount committed to during the registration phase, and the current share price, to later allow verification that the purchase was valid.

Requirements. Let us denote by (b_i, x_i, op_i), for $b_i \in \{0,1\}$ and $x_i \in \{0,1\}^t$, the private input of client C_i, for $i = 1, \ldots, n$, where $b_i = 0$ (respectively, $b_i = 1$) means that the i-th client wants to buy (respectively, sell) x_i stock shares at an offered price op_i. Also, let us denote by cp the current stock price per share, by f the price updating function, by np the new stock price per share at the end of the protocol and by k a security parameter. Finally, let us denote by $cert_i$ the time-stamped certificate eventually issued by S to client C_i at the end of the protocol. A *private stock purchase protocol* for function f and for n clients has to satisfy the following four requirements:

Correctness. If S and all clients $C_1, \ldots, C_n$ follow their protocol then with probability 1 at the end of the private stock purchase protocol the following holds: (1) $np = f((b_1, x_1, op_1), \ldots, (b_n, x_n, op_n))$, and (2) each client receives a certificate $cert_i$ containing its private input (b_i, x_i, op_i) and the new price np, which is verifiable to be valid by S.

Security against Clients. If S follows its protocol, then for all i and for all probabilistic polynomial time algorithms $C'_1, \ldots, C'_{i-1}, C'_{i+1}, \ldots, C'_n$, the probability that at the end of the private stock purchase protocol the client C_i does not output $cert_i$, a valid certificate associated with input (b_i, x_i, op_i) and new price np, is exponentially small (in k). Moreover, for any coalition of clients running in probabilistic polynomial time, the probability that at the end of the private stock purchase protocol, they are able to convince S to have a valid certificate $cert$ associated with an input different from all $(b_i, x_i, op_i), np$, for $i = 1, \ldots, n$, is negligible (in k).

Privacy against Clients. Let $i_1, \ldots, i_j \in \{1, \ldots, n\}$; if S follows its protocol, then for any probabilistic polynomial-time algorithms $C'_{i_1}, \ldots, C'_{i_j}$, the distribution of the view of such clients during an execution of the entire protocol is independent from the value of (b_i, x_i, op_i), for each $i \in \{1, \ldots, n\} \setminus \{i_1, \ldots, i_j\}$.

Privacy against the Server. Assume S follows its protocol; for any polynomial time strategy s_1 used by S at the end of the protocol, there exists a polynomial time strategy s_2 that can be used by S before the protocol starts, such that the probability that s_1 allows S to obtain some information about $(b_1, x_1, op_1), \ldots, (b_n, x_n, op_n)$ differs by the probability that s_2 allows S to do the same before the protocol only by a negligible (in k) amount, where both probabilities are conditioned by the fact that $np = f((b_1, x_1, op_1), \ldots, (b_n, x_n, op_n))$.

Remarks. We note that typically the first and the fourth phase of a private stock purchase protocol would require standard registration, certification and verification protocols to be executed, also varying according to non-cryptographic issues deriving from the specific application setting; instead, the second and third phases are supposed to contain the main cryptographic novelties of the protocol. We also note that private stock purchase protocols are very much related to private multi-party computation [27,14], for which no agreement on the 'right notions' of security or privacy has been reached yet, after several research efforts. Finding the 'right notions' of privacy and security for private stock purchase protocols is therefore beyond the scope of this work. Still, we believe that the above definition (following most principles in the current best definitions of private multi-party computation) describes a satisfactory notion of security and privacy for the application of interest in this paper and that the protocol that we present would essentially satisfy alternative notions, eventually claimed to be the 'right notion'. Let us point out however two main differences between the setting considered here and private multi-party computation. In terms of connectivity among the participants, here we only consider solutions in which the clients only talk to the server, and do not need to talk to each other. In terms of adversarial setting, here the server is assumed to be honest-but-curious (rather than being possibly malicious); moreover, we require and achieve security against arbitrary coalitions of up to all-but-one malicious clients (rather than only bounded-size coalitions). These differences are motivated by practical considerations and significantly differentiate our investigation from those in private multi-party computation. In particular, as a consequence, protocols given in the literature within the area of private multi-party computation do not solve the problems considered in this paper (and vice versa).

3.2 A High-Level View of Our Solution

As done for the definition, we describe our solution as divided into four phases: a registration phase, a price update phase, a capital update phase and a certification phase. Recall that we are describing a generic interval of the lifetime of the stock purchase protocol; therefore, we can assume that permanent information such as the current share price and the capital (or the number of available shares) are publicly available. Moreover, here and in the rest of the paper, for simplicity, we will assume that the parties are connected through private channels (that can be implemented, for instance, using a non-malleable encryption scheme).

Registration Phase. First of all the server publishes two public keys: one for an encryption scheme, and one for a signature scheme. Now, a client that wants to take part in the stock purchase protocol makes a commitment to its private input (representing the amount of shares to be bought or sold and the offered price for those) and sends this commitment to the server. Then the server signs such a commitment and sends the resulting signature to the client. Now, the client publishes his commitment, the signature received from the server, a public-key of an encryption scheme, and standard information such as his identity.

Implementation of this Phase. We require that the commitment by the client is implemented as an El-Gamal encryption of the input to be committed to (this choice is for efficiency and compatibility with the remaining protocols in the paper). No particular implementation is required for the signature scheme.

Price Update Phase. In this phase the server and all clients who successfully completed the previous phase run a protocol, called UPDATE, and described in Section 5, which has the following properties. At the end, the server obtains the value output by an evaluation of a known function over all private inputs of clients, (such value being the new share price) but no other information about all private inputs of the clients. Even for this protocol each client only interacts with the server and is also guaranteed that no coalition of malicious clients can receive any information about her private input. At the end of this phase, the server publishes the new share price.
Implementation of this Phase. We note that the efficiency of the implementation of this protocol may depend on how complicated the function for updating the share price is. In the particular case in which the function is linear (which is really an important case since it captures typical functions such as the average), we can use a simple extension of the protocol SUM also used in the following phase; the resulting implementation would therefore be efficient and constant-round. In order to cover the more general case of an arbitrary (and therefore, non-linear) function, in Section 5, we present an implementation for protocol UPDATE, by reducing the private computation of a function in this model to the private computation of a function in the two-party model (which is of independent interest). We note two attractive properties of our solution: it can be run in a constant number of rounds, and it reveals no information about function f to the clients (although we did not explicitly require this property in the definition of previous section, we believe it may still be an interesting property to achieve). The protocol UPDATE can be implemented under the assumption of the hardness of deciding the Diffie–Hellman problem (or, more generally, of the existence of any oblivious transfer).

Capital Update Phase. In this phase the server and all clients who successfully completed the previous two phases run a protocol, called PSUM, and described in Section 4, which has the following properties. At the end, the server obtains a partial sum of the private inputs of clients. Specifically, clients who intended to buy (resp., sell) shares contribute a positive (resp., negative) amount if their offered price was larger (resp., smaller) than the new price. In this protocol, each client only interacts with the server and is guaranteed that no additional information about her private input is revealed to the server or to any coalition of malicious clients (precisely, the server only obtains the partial sum of the share amounts and the other clients obtain no information at all). At the end of this phase, the server publishes the new capital (or amount of available shares).

Implementation of this Phase. In Section 4 we present an efficient and constant-round implementation for protocol PSUM, under the assumption of the hardness of deciding the Diffie–Hellman problem.

Certification Phase. In this phase the server only interacts with all clients who successfully completed the previous three phases and sends to each of them another signature of her commitment, of the new price, and of a special message indicating that she has completed her transaction.
Implementation of this Phase. No specific implementation for the signature scheme is required.

Properties of Our Protocol. Given the simplicity of the registration and certification phase, and the stated properties of subprotocols PSUM and UPDATE, it is not hard to verify that the protocol described in this section satisfies the definition of Section 3.

4 A Private Stock Purchase Protocol: Capital Update Phase

In this section we present the protocol that will be executed by the participants in the capital update phase. At the beginning of this phase each client C_i has already committed to her desired amount of shares to be sold and bought and to the offered price for each of those shares. Then the goal of this phase is to update the capital of stock shares, by transferring to the server S a partial sum of these committed amounts of shares. Specifically, each client who intended to buy (resp., sell) some amount of shares will contribute a positive (resp., negative) amount to the final sum if her offered price op_i was larger (resp., smaller) than the new price. The new capital is then obtained by the server by subtracting the final sum from the current capital.

More precisely, the protocol we would like to construct, called PSUM, is run by a server S and n clients $C_1, \ldots, C_n$, where each client C_i has, as private input, an integer x_i, that may be positive or negative (let $b_i = 1$ denote a negative sign for x_i and $b_i = 0$ denote a positive one) and an integer op_i. We require that at the end of the protocol S obtains a partial sum of the clients' committed amount of shares (i.e., the value $z = \sum_{i \in T} (-1)^{b_i} x_i$, where $T = \{i : ((b_i = 0) \wedge (op_i \geq np)) \vee ((b_i = 1) \wedge (op_i < np))\} \subseteq \{1, \ldots, n\}$). We also require that the server's view is independent on the values of the x_i's, given the value z of the final sum, and that each coalition of clients, no matter how it behaves, receives a view that is independent on the other clients' private inputs.

The description of this construction is divided as follows. First, in Section 4.1 we describe a zero-knowledge protocol for proving a sum-related statement on El-Gamal encryptions. Then, in Section 4.2 we show how to use this protocol in order to obtain protocols for proving more elaborated statements. In Section 4.3 a protocol for privately computing the sum of several El-Gamal encryption, thus solving a slightly simpler version of our problem. Finally, in Section 4.4 we extend the protocol of previous section to privately computing a partial sum, as defined above.

4.1 A Novel Zero-Knowledge Protocol

In this section we present a zero-knowledge protocol for the language EG-SUM, defined as follows. Given (p, q, g, h) such that $p = 2q + 1$, p, q are primes, g generates a subgroup of order q, and h is a member of this subgroup, the language EG-SUM is the set of tuples $((u_1, v_2), (u_2, v_2), (u, v))$ for which there exist r_1, r_2, r, m_1, m_2, m such that

1. $u_1 = g^{r_1} \bmod p$, $v_1 = h^{r_1} m_1 \bmod p$;
2. $u_2 = g^{r_2} \bmod p$, $v_2 = h^{r_2} m_2 \bmod p$;
3. $u = g^{r} \bmod p$, $v = h^{r} m \bmod p$, $m_1 + m_2 = m \bmod p$.

We now describe an efficient perfect zero-knowledge argument (A,B) for language EG-SUM, where the soundness property holds under the assumption that computing discrete logarithms is hard. The proof system is efficient in two ways: first, it does not require reductions of the statement to be proven to an NP-complete statement; second, the prover A, given values r_1, r_2, r, m_1, m_2, m, can run in probabilistic polynomial time.

An Informal Description. The zero-knowledge protocol we propose uses the cut-and-choose technique of [15,11]. A first idea in constructing our protocol is that of combining the linearity of the equation to be proved (namely, that $m_1 + m_2 = m \bmod p$) with the fact that the encryption function of the El-Gamal cryptosystem satisfies some (weak) sum-homomorphism property (specifically, if encryptions (u_1, v_1) of m_1 and (u_1, q_2) of d_1 are computed using the same randomness, then the pair $(u_1, v_1 + q_2 \bmod p)$ is an encryption of $(d_1 + m_1) \bmod p$). Using this idea alone, a (still incorrect) protocol could consist of showing that either the equality $d_1 + d_2 = d \bmod p$ holds, or the equality $(m_1 + d_1) + (m_2 + d_2) = (m + d) \bmod p$ holds, for random values d_1, d_2. This protocol reveals the randomness used to encrypt d_1, which is the same as that used to encrypt m_1, and therefore is not zero-knowledge. To solve this problem, we use the fact that the encryption function of the El-Gamal cryptosystem is product-homomorphic with respect to componentwise product modulo p, (specifically, given encryptions (u_1, v_1) of m_1 and (c, e) of a, then the pair $(cu_1 \bmod p, ev_1 \bmod p)$ is an encryption of $am_1 \bmod p$). Using this property, we can modify the protocol so that it consists of showing that either the equality $am_1 + am_2 = am \bmod p$ holds, or the equality $(am_1 + d_1) + (am_2 + d_2) = (am + d) \bmod p$ holds, where the encryptions of d_1, d_2, d have to be computed using the same randomness as the encryption of am_1, am_2, am, respectively. This modified protocol still does not work since all these encryptions, when sent by the prover to the verifier, would not be simultaneously secure. To solve this last problem, we hide the encryptions by committing to them using a non-interactive commitment protocol that is also sum-homomorphic (namely, given commitments com_1 of m_1 and com_2 of m_2, it is possible to efficiently compute a commitment com of m). An example of a commitment scheme that satisfies this property is the one in [20]; this scheme, in the honest-receiver version, can be implemented in one round of communication, is perfectly-secure (namely, not even a computationally-unbounded receiver can obtain any information about the committed value), and is computationally-binding (namely, assuming that computing discrete logarithms is hard, the committer can reveal the committed value in a unique way). Moreover, given commitments com_1 of m_1, com_2 of m_2, the value $com = com_1 \cdot com_2 \bmod p$ is a commitment to $m_1 + m_2 \bmod p$. In the description of our proof system, we will refer to this scheme as Pedersen's commitment scheme.

A More Formal Description. We proceed by first describing an atomic protocol (A,B), having soundness error 3/4.

1. A uniformly chooses $r_1', r_2', r', a, d_1, d_2, d \in Z_p$ such that $d_1 + d_2 = d \bmod p$;
   A computes the following El-Gamal-encryptions:
   an encryption (u_1', v_1') of am_1 using $r_1' + r_1$ as randomness;
   an encryption (u_2', v_2') of am_2 using $r_2' + r_2$ as randomness;
   an encryption (u', v') of am using $r' + r$ as randomness;
   an encryption (p_1, q_1) of d_1 using $r_1' + r_1$ as randomness;
   an encryption (p_2, q_2) of d_2 using $r_2' + r_2$ as randomness;
   an encryption (p, q) of d using $r' + r$ as randomness;
   A computes two commitments using Pedersen's commitment scheme: com_1 of $(u_1', v_1', u_2', v_2', u', v')$ and com_2 of $(p_1, q_1, p_2, q_2, p, q)$;
   A sends com_1, com_2 to B
2. B uniformly chooses $b \in \{1, 2, 3\}$ and sends it to A
3. If $b = 1$ then
   A decommits com_1 as $(u_1', v_1', u_2', v_2', u', v')$ and sends r_1', r_2', r', a to B;
   using the above, B checks that $(u_1'/u_1, v_1'/v_1), (u_2'/u_2, v_2'/v_2), (u'/u, v'/v)$ are all encryptions of a;
   if $b = 2$ then
   A decommits com_2 as $(p_1, q_1, p_2, q_2, p, q)$ and sends $r_1 + r_1', r_2 + r_2', r + r'$;
   using the above, B checks that com_2 is correctly decommitted, decrypts (p_1, q_1) as d_1, (p_2, q_2) as d_2, and (p, q) as d, and checks that $d_1 + d_2 = d \bmod p$;
   if $b = 3$ then
   using com_1, com_2, A computes commitment com to $(u_1' + p_1, v_1' + q_1, u_2' + p_2, v_2' + q_2, u' + p, v' + q)$;
   A decommits com as $(u_1' + p_1, v_1' + q_1, u_2' + p_2, v_2' + q_2, u' + p, v' + q)$;
   A computes $z_1 = am_1 + b_1, z_2 = am_2 + b_2, z = am + b$ and sends $(z_1, z_2, z,_1 + r_1', r_2 + r_2', r + r')$ to B
   B checks that com is correctly computed from com_1, com_2, that com is correctly decommitted, that $u_1' + p_1$ is an encryption of z_1, $u_2' + p_2$ is an encryption of z_2, $u' + p$ is an encryption of z, and that $z_1 + z_2 = z \bmod p$.
   if at least one verification is not satisfied then B returns: REJECT else B returns: ACCEPT.

We now show that the above protocol satisfies the three properties of completeness, soundness and perfect zero-knowledge.

Completeness. Assume that the input is in the language; then, A, who is given r_1, r_2, r, m_1, m_2, m, can meet B's verifications with probability 1.

Soundness. Note that any tuple $((u_1, v_1), (u_2, v_2), (u, v))$ can be written as a triple of El-Gamal encryptions for some m_1, m_2, m. Therefore, if the input is not in the language it must happen that $m_1 + m_2 \neq m \bmod p$. Thus, let us assume that the latter inequality holds. As a consequence either (a) the inequality $(am_1 + d_1) + (am_2 + d_2) \neq (am + d) \bmod p$ holds, or (b) the inequality $d_1 + d_2 \neq d \bmod p$ holds, for any $a, d_1, d_2, d \in Z_p$. By the binding property of the Pedersen commitment scheme, we have that: in case (b), A cannot meet the question $b = 2$, and, in case (a), A cannot meet at least one of the questions $b = 1$ and $b = 3$. Therefore the probability that A can cheat is at most 2/3 plus the probability that he can cheat in any of the decommitment, which is negligible (assuming the hardness of computing discrete logarithms); therefore, the overall probability that A can cheat is at most, say, 3/4.

Perfect Zero-Knowledge. We construct a simulator S that, using a potentially dishonest verifier B$'$, generates a transcript having distribution computationally indistinguishable from that of a transcript generated after a real execution of the protocol between A and B$'$. The algorithm S uses the usual trial-and-error strategy, with rewinding. Specifically, S randomly chooses $b' \in \{1, 2, 3\}$ and computed a simulated transcript assuming that the challenge b sent by B$'$ is equal to b'; if yes, S outputs the computed transcript; if not, S rewinds B$'$ and tries again. The computation of a simulated transcript for each value of b is done as follows. In the case $b = 1$ it is easy to efficiently simulate the second message from A and the commitment com_1; the commitment com_2 is simulated as a commitment to a random value of the same length. The case $b = 2$ is analogue to the case $b = 1$. In the case $b = 3$ it is easy to efficiently simulate the second message from A and the commitment com; the commitment com_1 is simulated as a commitment to a random value of the same length, and the commitment com_2 is computed as com/com_1. By using the perfect security property of Pedersen's commitment scheme we can show that the simulation is perfect.

Remark. We remark that the soundness error of the above protocol can be decreased to exponentially small by running several parallel repetitions of it, and then having the verifier commit to his random bits by using a discrete-log based information-theoretically secure commitment scheme (see, e.g., [2]) and give a 3-round witness-indistinguishable proof of knowledge of the discrete log of the message sent during the execution of Pedersen's commitment scheme. The resulting protocol is a perfect zero-knowledge argument for language EG-SUM that has exponentially small soundness error and can be implemented in a constant number of rounds.

4.2 More Zero-Knowledge Protocols

We show how to use the protocol in previous section to obtain zero-knowledge protocols for more elaborated statements, that will be used later, in the construction of our capital update protocol.

Linear Equalities over El-Gamal-Encrypted Values. We note that in the protocol in previous section, for simplicity, we have considered the case of the equality between a value and two addends. However, the same technique naturally extends to the case of n addends, for any n. More generally, the same technique can be used to give an efficient and constant-round perfect zero-knowledge argument for the language n-EG-LIN1, defined as follows. Given (p, q, g, h) such that $p = 2q + 1$, p, q are primes, g generates a subgroup of order q, and h is a member of this subgroup, the language n-EG-LIN1 is the set of tuples $((\alpha_1, u_1, v_1), \ldots, (\alpha_n, u_n, v_n), (\alpha, u, v))$ for which there exist $r_1, \ldots, r_n, r$ and $m_1, \ldots, m_n, m$ such that

1. $u_i = g^{r_i} \bmod p$, $v_i = h^{r_i} m_i \bmod p$, for $i = 1, \ldots, n$;
2. $u = g^r \bmod p$, $v = h^r m \bmod p$, $\alpha_1 m_1 + \ldots + \alpha_n m_n = \alpha m \bmod p$.

We also note that by combining this protocol with techniques in [8], we obtain a protocol for proving any monotone formula over membership statements to language n-EG-LIN1.

Linear Equalities with Unencrypted Known Term. In our main construction we will need a zero-knowledge protocol for a language similar to language n-EG-LIN1, the only difference being in that the known term of the linear equality is in clear (rather than encrypted). By simply encrypting the known term and revealing the randomness used to compute this encryption, one can use the same protocol, thus obtaining an efficient and constant-round perfect zero-knowledge argument for the language n-EG-LIN2, defined as follows. Given (p, q, g, h) such that $p = 2q + 1$, p, q are primes, g generates a subgroup of order q, and h is a member of this subgroup, the language n-EG-LIN2 is the set of tuples $((\alpha_1, u_1, v_1), \ldots, (\alpha_n, u_n, v_n), \alpha, m)$ for which there exist $r_1, \ldots, r_n, m_1, \ldots, m_n$ such that

1. $u_i = g^{r_i} \bmod p$, $v_i = h^{r_i} m_i \bmod p$, for $i = 1, \ldots, n$;
2. $\alpha_1 m_1 + \ldots + \alpha_n m_n = \alpha m \bmod p$.

Linear Equalities with Encryptions under Different Public Keys. Another variation over language n-EG-LIN1 that we will need in our main construction is that in which the addends in the linear equality are decryptions of El-Gamal ciphertexts computed according to different public keys (but using the same parameters; namely, the same prime p and generator g). A protocol for this variation can be obtained by using multiple applications of the protocol in Section 4.1, as follows. For each ciphertext c_i computed according to a different public key, the prover computes a ciphertext c_i' computed according to a single, fixed, public key, and sends c_i' to the verifier. Then the prover proves that the plaintext associated with c_i' and the plaintext associated with c_i are the same, for each i (note that this can be proved by using a simplified version of the protocol in Section 4.1). Finally, the prover proves that the linear equality holds by using all ciphertexts c_i' that are computed according to the same public key, and therefore she can use the protocol for language n-EG-LIN2. This gives an efficient and constant-round zero-knowledge protocol for language n-EG-LIN3, defined as the set of tuples $((\alpha_1, u_1, v_1, h_1), \ldots, (\alpha_n, u_n, v_n, h_n), (\alpha, u, v, h))$ for which there exist $r_1, \ldots, r_n, r, m_1, \ldots, m_n, m$ such that

1. $u_i = g^{r_i} \bmod p$, $v_i = h_i^{r_i} m_i \bmod p$, for $i = 1, \ldots, n$;
2. $u = g^r \bmod p$, $v = h^r m \bmod p$, $\alpha_1 m_1 + \ldots + \alpha_n m_n = \alpha m \bmod p$.

4.3 Privately Computing the Sum of Encrypted Values

In this section we describe a protocol for privately computing the sum of *all* share amounts committed by clients, regardless of whether their offered price was larger or smaller than the new stock price. We call this protocol SUM.

Description of Protocol SUM. Let p be a prime given by S to each of the clients, that is much larger than any of the x_i's (e.g., $|p| > 2n|x_i|$ for any i would suffice). Assume that at the beginning of the protocol each client C_i has published a public key pk_i generated using algorithm KG and an encryption c_i of private input (b_i, x_i, op_i), and let $y_i = (-1)^{b_i} x_i \bmod p$. Then protocol SUM goes as follows:

1. Each client C_i writes his input as $y_i = s_{i,1} + \cdots + s_{i,n} \bmod p$ for $s_{i,j}$'s chosen randomly in Z_p and such that the equality holds;
2. each client C_i encrypts each $s_{i,j}$ according to algorithm E and using the j-th client's public key, thus obtaining encryptions $c_{i,j}$, for $j = 1, \ldots, n$;
3. each client C_i sends all $c_{i,j}$ to S together with a zero-knowledge proof that the encryptions $c_{i,1}, \ldots, c_{i,n}$ have been correctly computed; that is, proving that $y_i = s_{i,1} + \cdots + s_{i,n} \bmod p$ (using the zero-knowledge protocol from Section 4.2 for language n-EG-LIN3);
4. for $i = 1, \ldots, n$, server S verifies the proof from client C_i; if this proof is rejected, client C_i is discarded and the computation continues with the remaining clients; if this proof is accepted, S sends all encryptions $c_{1,j}, \ldots, c_{n,j}$ to client C_j, for $j = 1, \ldots, n$;
5. for $j = 1, \ldots, n$, client C_j decrypts all encryptions $c_{1,j}, \ldots, c_{n,j}$ as $s_{1,j}, \ldots, s_{n,j}$ and sends $t_j = s_{1,j} + \cdots + s_{n,j} \bmod p$ to S along with a zero-knowledge proof that t_j has been correctly computed; that is, proving that $t_j = s_{1,j} + \cdots + s_{n,j} \bmod p$ (using the zero-knowledge protocol from Section 4.2 for language n-EG-LIN2);
6. for $j = 1, \ldots, n$, server S verifies the proof from client C_j; if this proof is rejected, client C_j is discarded and the computation continues with the remaining clients;
7. S computes *sum* as the sum modulo p of all t_j's corresponding to clients C_j which have not been discarded from the protocol. If $|sum| \leq |x_1|$ then S returns $(0, sum)$ else S returns $(1, p - sum)$.

Properties of Protocol SUM. We show that protocol SUM satisfies several properties, such as correctness, security against clients, privacy against clients and privacy against server (although we have not exactly defined such properties in this context, their semantic meaning is along the lines of the definition of private stock purchase protocols and will be made clearer in the following discussion). We also show that SUM can be implemented in a constant number of rounds.

Correctness. First of all we note that it is possible to implement protocol SUM, as described above, since it is possible to implement the zero-knowledge protocols in steps 3 and 5 because of the protocols proposed in Section 4.2. Moreover, we note that if all parties follow their protocol then the output of server S is exactly equal to the sum of the clients' private inputs.

Security against Clients. Here we consider the case of clients who may deviate from the protocol and try to compromise the server's final computation. We see from the construction of protocol SUM that clients always have to provide proofs of correctness of their computations to the server (specifically, in both step 3 and step 5), or they are discarded from the execution. Therefore, at the end of protocol SUM, the server is always able to compute the sum of the private inputs of the clients who have not been discarded.

Privacy against Clients. Here we consider the case of clients who may deviate from the protocol and try to obtain information from the other clients' private inputs. We see from the construction of protocol SUM that each private input of a client is shared among all the clients using an n-out-of-n secret sharing (implemented using sum modulo p of the n values) and therefore even a coalition of $n-1$ values does not obtain any information at all (namely, even if clients are not computationally limited) from the values sent by the server in step 4.

Privacy against Server. Here we consider the question of whether the view of the server reveals any information at all about the client's private inputs (other than their sum). We see that in step 3 of protocol SUM the server only obtains encryptions of shares of the clients' private inputs, along with zero-knowledge proofs of correctness of the computation of such shares, and therefore, since the encryption scheme used is assumed to be semantically secure, no information is revealed to the server in this step. Then we note that in step 5 of protocol SUM the server only obtains values $t_1, \ldots, t_n$, along with a zero-knowledge proof of correctness of their computation, and we can see that the distribution of such values is that of n random values in Z_p such that their sum modulo p is equal to the sum of all the y_i's.

Round-Complexity. The number of rounds of protocol SUM is constant provided the zero-knowledge proofs in step 3 and 5 can both be executed in a constant number of rounds. This fact has been established already in Section 4.1.

4.4 Privately Computing the Partial Sum of Encrypted Values

In this section we show how to extend the protocol of previous section for privately computing a sum of values into computing a 'partial' sum of values. The resulting protocol can be directly used as a capital update protocol in our private stock purchase protocol.

More specifically, recall that we denote by (b_i, x_i, op_i) the private input to client C_i, where $b_i \in \{0,1\}$ is the sign denoting whether C_i wants to buy or sell the share amount x_i and op_i is the offered price for each of these shares. Moreover, by np we denote the new share price computed at the end of the price updating phase. We note that the protocol SUM can be used by the server to privately compute the value $\sum_{i=1}^{n}(-1)^{b_i}x_i$; however, this value does not take into account the offered prices committed by the clients. In other words, in our capital update protocol, we would like the server to retrieve the sum of the x_i's only for those clients whose offered prices are valid (i.e., larger than the new price np if they are buying shares or smaller otherwise). Therefore, we need to modify the protocol SUM into a protocol for computing a partial sum; namely, a sum over all clients satisfying the above property (i.e., S will be able to compute $sum = \sum_{i \in T}(-1)^{b_i}x_i$, where $T = \{i : ((b_i = 0) \wedge (op_i \geq np)) \vee ((b_i = 1) \wedge (op_i < np))\} \subseteq \{1, \ldots, n\}$).

Our protocol PSUM uses as a tool a 'conditional oblivious transfer' [10]. More formally, this is a protocol used by S to transfer to client C_i one of two strings s_0, s_1 such that client C_i will obtain s_0 if $op_i \geq np$ or s_1 otherwise, without S learning any information about the value of op_i, including whether C_i received s_0 or s_1.

Description of Protocol PSUM. This protocol is executed between each client C_i and the server S. The basic idea of this protocol is that client C_i will create two ciphertexts, one with plaintext equal to 0 and one with plaintext equal to $-x_i$. Server S will help C_i select one of the two based on the inequality $op_i \geq np$ and on the value of b_i without obtaining any information about these, so that later C_i can contribute to the final sum ciphertexts with associated plaintexts $x_i, 0$ if his offered price if valid (namely, if $op_i \geq np$ and $b_i = 0$ or $op_i < np$ and $b_i = 1$) or ciphertexts with associated plaintexts $x_i, -x_i$ otherwise. Note that effectively client C_i is contributing to the final sum her share amount if her offered price is valid or zero otherwise. The actual protocol we describe below has some additional technical complication for two reasons: first, an El-Gamal encryption of 0 is not secure (therefore, we split it into two encryptions of values which sum up to 0); second, we need to protect the server from possible malicious behavior from the client.

We can assume that in the following description all encryptions and decryptions will be computed according to the El-Gamal public-key cryptosystems. Then the protocol PSUM goes as follows:

1. Client C_i uniformly chooses r_1, computes $z = -x_i - r_1 \bmod p$ and an encryption c_1 of z, and sends c_1, r_1 to S;

2. S uniformly chooses r_2, s_1, s_2, computes $r_3 = r_1 - r_2 \bmod p$, $s_3 = -s_1 - s_2 \bmod p$, encryptions d_j of s_j, for $j = 1, 2, 3$, and encryptions c_l of r_l, for $l = 2, 3$;
3. S transfers to C_i strings $a_0 = (c_1, c_2, c_3)$ and $a_1 = (d_1, d_2, d_3)$ using a conditional oblivious transfer based on the condition ($op_i \geq np$ AND $b_i = 0$) OR ($op_i < np$ AND $b_i = 1$).
4. let a_b, for some $b \in \{0, 1\}$, be the string obtained by C_i at the end of the execution of the conditional oblivious transfer subprotocol;
5. C_i decrypts the 3 ciphertexts in a_b, encrypts the obtained plaintexts using independently chosen random strings, thus obtaining triple $v = (e_1, e_2, e_3)$ and sends it to S;
6. S sends a_0, a_1 to C_i;
7. C_i sends to S a zero-knowledge proof that the plaintexts associated with v are either the same as the plaintexts associated with a_0 or the same as those associated with a_1;
8. all clients and S run the protocol SUM, where client C_i contributes to the final sum with the plaintexts associated with ciphertexts x_i, e_1, e_2, e_3.

5 A Private Stock Purchase Protocol: Price Update Phase

In this section we present the protocol that will be executed by the participants in the price update phase. We consider the sufficiently general case in which the next share price can be a function of the previous share price and of the private inputs of the clients in the most recent time interval.

Specifically, the protocol we would like to construct, called UPDATE, is run by a server S and n clients $C_1, \ldots, C_n$, where each client C_i has, as private input, an integer x_i, that may be positive or negative (let $b_i = 1$ denote a negative sign for x_i and $b_i = 0$ denote a positive one). Both S and the clients have the description of a circuit computing function f as a common input. We require that at the end of the protocol S obtains the output of an application of function f over the clients' private inputs (i.e., the value $z = f((b_1, x_1), \ldots, (b_n, x_n)))$; that the server's view is independent on the values of the clients' inputs, given that z is the output of function f over those, and that each coalition of clients, no matter how it behaves, receives a view that is independent on the other clients' private inputs. We note that if f is a linear function of the x_i's, then protocol UPDATE can be constructed by performing minor modifications to the protocol PSUM in Section 4. This would give a very efficient construction for the entire private stock purchase protocol. In the rest of this section, we deal with the case f is an arbitrary (and thus possibly non-linear) function. In achieving generality, our construction loses the attractive efficiency properties of protocol PSUM; in particular, our protocol builds over a general protocol for 2-party secure computation [27].

The description of this construction is divided as follows. First, in Section 5.1 we recall a protocol for 2-party secure computation [27,6] and then in Section 5.2 we show how to adapt this scheme to our model.

5.1 A 2-Party Secure Computation Protocol

The problem of 2-party secure computation, first considered by Yao in the influential paper [27], asks whether two parties Alice and Bob, having private inputs x and y, respectively, can compute a value $z = f(x, y)$, for some public function f, without revealing any additional information about their private input. Recently, other protocols have been proposed (e.g., [25,6]); here we recall an abstracted version of the protocol in [6], which makes our construction easier to describe.

The 2-Party Secure Protocol in [6]. We describe the case in which both Alice and Bob are honest since the case in which both can be malicious is dealt with using well-known techniques from [14] (i.e., by compiling the honest case with each party proving in zero-knowledge that the messages she is sending have been correctly computed according to the protocol's instructions). The honest version of this protocol combines Yao's construction [27] with oblivious transfer. Yao's construction consists of three procedures: an algorithm C that Bob uses to construct an encrypted circuit, an interactive protocol T between Alice and Bob, and an algorithm E that Bob uses to evaluate $f(x, y)$. More precisely, algorithm C outputs an encrypted version of function $f(\cdot, y)$, including a pair of k-bit strings for each input bit x_i. In order to compute $f(x, y)$, one of these two k-bit string is necessary for each bit x_i of x (which one of the two strings it depends on the value of x_i). In [6] oblivious transfer is used by Bob to transfer to Alice the appropriate k-bit string according to the value of x_i, without Bob revealing to Alice any information on the other string and without Alice revealing to Bob any information on the value of x_i. The rest of the computation proceeds as in Yao's protocol and will stay unchanged in our protocol as well. The oblivious transfer protocol used by Bob to Alice could be the one given in Section 2, or even an abstraction of it, as we now describe. We can consider an oblivious transfer as the following protocol between a sender and a receiver. The receiver publishes two channels such that he can read messages received over only one of them, but the sender cannot tell which one; then the sender sends each of the two messages through each of the channels.

5.2 The Adaptation to Our Setting

We now consider the possibility of adapting the protocol in [6,27] to our setting. Recall that from a communication standpoint, in our setting we would like clients not to talk to each other and that the server is assumed to behave honestly. Moreover, clients want their inputs to be private not only against the server but also against any coalition of other possibly malicious clients.

Description. We now describe the intuitions behind our adaptation. If we could consider all clients together as a single participant Alice and the server as participant Bob, then any 2-party secure protocol would be enough since Bob does not obtain any information about Alice's private input, and in the end Bob obtains the output of the function of Alice's input. However, the setting at hand is more complicated since Alice's input is in fact distributed among the various clients, who should not communicate. One fix to the lack of communication is to ask help from the server; indeed, since the server is honest, he might as well help the clients share their private inputs somehow. Even sharing the private inputs has to be done carefully, since the privacy requirements that our protocol has to satisfy ask that each client keeps her input private even if all other clients behave maliciously. In our solution we have each client send to all other clients, through the server, some information which is enough to allow other clients to play as Alice but still does not reveal any information about all other client's inputs. Specifically, using the oblivious transfer abstraction at the end of Section 5.1, they will send a channel that is readable (without sending the other, unreadable channel); therefore, the other client will not be able to understand which private bit this channel is associated with, but she will still be able to use it to run the oblivious transfer protocol. Finally, an execution of the 2-party protocol has to be executed for each client, and at the end the server checks that all outputs received by these executions are the same. We note that each of these executions can be run in parallel and therefore the resulting protocol can still be executed in a constant number of rounds. The correctness, security and privacy properties follow from the related properties of the 2-party protocol.

Acknowledgments

Remarks from Bob Hettinga and Ronald Rivest helped in further generalizing the real-life setting captured by our solution.

References

1. M. Bellare and S. Micali, *A Non-Interactive Oblivious Transfer Protocol and its Applications*, in Proceedings of "Advances in Cryptology – CRYPTO'88", Lecture Notes in Computer Science, Springer Verlag.
2. M. Bellare, S. Micali, and R. Ostrovsky, *Perfect Zero-Knowledge in Constant Rounds*, in Proceedings of 22th Annual ACM Symposium on Theory of Computing (STOC'90).
3. G. Brassard, C. Crépeau, and D. Chaum, *Minimum Disclosure Proofs of Knowledge*, Journal of Computer and System Sciences, vol. 37, no. 2, 1988, pp. 156–189.
4. C. Cauchin, *Efficient Private Bidding and Auctions with an Oblivious Third Party*, in Proc. of ACM Conference on Computers, Communications and Security, 1999, Springer Verlag.
5. C. Crépeau, *Equivalence between Two Flavors of Oblivious Transfer*, in Proceedings of "Advances in Cryptology – CRYPTO'87", Lecture Notes in Computer Science, Springer Verlag.
6. C. Cachin, J. Camenish, J. Kilian, and J. Muller, *One-Round Secure Computation and Secure Autonomous Agents*, in Proceedings of ICALP 2000, Springer Verlag.
7. A. De Santis, Y. Desmedt, Y. Frankel, and M. Yung, *How to Share a Function Securely,* in Proceedings of 26th Annual ACM Symposium on Theory of Computing (STOC'87).

8. A. De Santis, G. Di Crescenzo, G. Persiano, and M. Yung, *On Monotone Formula Closure of SZK,* in Proceedings of 35th Annual IEEE Symposium on Foundations of Computer Science (FOCS'94).
9. G. Di Crescenzo, *Private Selective Payment Protocols,* in Proceedings of Financial Cryptography 2000, Springer Verlag.
10. G. Di Crescenzo, R. Ostrovsky, and S. Rajagopalan, *Conditional Oblivious Transfer and Timed-Release Encryption,* in Proceedings of "Advances in Cryptology – EUROCRYPT'99", Lecture Notes in Computer Science, Springer Verlag.
11. U. Feige, A. Fiat, and A. Shamir, *Zero-Knowledge Proofs of Identity*, in Journal of Cryptology, vol. 1, n. 2, pp. 77–94, 1988.
12. T. El Gamal, *A Public key Cryptosystem abd a Signature scheme based on Discrete Logarythms*, in Proceedings of "Advances in Cryptology – CRYPTO'84", Lecture Notes in Computer Science, Springer Verlag.
13. M. Franklin and M. Reiter, *The Desing and Implementation of a Secure Auction Service*, in IEEE Transactions on Software Engineering, vol. 22, n. 5, pp. 302–312, 1996.
14. O. Goldreich, S. Micali, and A. Wigderson, *How to Play any Mental Game*, in Proceedings of 19th Annual ACM Symposium on Theory of Computing (STOC'87).
15. S. Goldwasser, S. Micali, and C. Rackoff, *The Knowledge Complexity of Interactive Proof-Systems*, in SIAM Journal on Computing, vol. 18, n. 1, 1989.
16. M. Harkavy, D. Tygar and H. Kikuchi, *Electronic Auctions with Private Bids*, in Proceedings of 3rd USENIX Workshop on Electronic Commerce, 1998.
17. M. Jakobsson and A. Juels, *Addition of El-Gamal Plaintexts*, in Proceedings of "Advances in Cryptology – ASIACRYPT 2000", Lecture Notes in Computer Science, Springer Verlag.
18. P. MacKenzie and J. Sorensen, *Anonymous Investing: Hiding the Identities of Stockholders*, in Proceedings of Financial Cryptography 1999, Springer Verlag.
19. M. Naor and B. Pinkas, *Efficient Oblivious Transfer Protocols*, in Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms (SODA 2001).
20. T. Pedersen, *Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing*, in Proceedings of "Advances in Cryptology – CRYPTO'91", Lecture Notes in Computer Science, Springer Verlag.
21. M. Rabin, *How to Exchange Secrets by Oblivious Transfer*, TR-81 Aiken Computation Laboratory, Harvard, 1981.
22. A. Shamir, *How to Share a Secret*, in Communications of the ACM, vol. 22, pp. 612–613, 1979.
23. K. Sako, *An Auction Protocol Which Hides Bids of Losers*, in Proceedings of Public-Key Cryptography 2000, Springer Verlag.
24. K. Sakurai and S. Miyazaki, *A Bulletin-Board based Digital Auction Scheme with Bidding Down Strategy,* in Proceedings of 1999 International Workshop on Cryptographic Techniques and E-Commerce.
25. T. Sander, A. Young, and M. Yung, *Cryptocomputing in* NC^1, in Proceedings of 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS'99).
26. S. Stubblebine and P. Syverson, *Fair on-line Auctions without Special Trusted Parties*, in Proceedings of Financial Cryptography 1999, Springer Verlag.
27. A.C. Yao, *Protocols for Secure Computations*, in Proceedings of 23th Annual IEEE Symposium on Foundations of Computer Science (FOCS'82).