In *Discrete Applied Mathematics*, 111 (2001) 23–36. (Special issue on coding and cryptology)

A Fair and Efficient Solution to the Socialist Millionaires' Problem

Fabrice Boudot[a] Berry Schoenmakers[b] Jacques Traoré[a]

[a]*France Télécom R&D, 42 rue des Coutures, BP 6243, 14066 Caen Cedex, France*
e-mail: {fabrice.boudot,jacques.traore}@francetelecom.fr

[b]*Department of Mathematics and Computing Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands*
e-mail: berry@win.tue.nl

Abstract

We present a solution to the *Tiercé problem*, in which two players want to know whether they have backed the same combination (but neither player wants to disclose its combination to the other one). The problem is also known as the *socialist millionaires' problem*, in which two millionaires want to know whether they happen to be equally rich. In our solution, both players will be convinced of the correctness of the equality test between their combinations and will get no additional information on the other player's combination. Our solution is *fair*: one party cannot get the result of the comparison while preventing the other one from getting it. The protocol requires $O(k)$ exponentiations only, where k is a security parameter.

1 Introduction

1.1 Description of the problem

Two players both have decided to back a combination for the coming Tiercé [1]. The players want to know whether they happen to back the same combination, but neither player wants to simply disclose its combination to the other. This problem is called the *Tiercé problem* [19] or the *socialist millionaires' problem* [10]. It is a variant of the *millionaires' problem*, introduced by Yao [17,18], in which two players wish to compare their riches: they wish to know who is richer, but apart from that they do not want to disclose any other information on their riches to each other.

[1] Tiercé is a French betting game in which the winning combination consists of the three first of a horse race (in order).

Formally, the problem is to find an efficient two-party protocol for the secure computation of function $f : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}$ with $f(x,y) = [x = y]$, where $[B] = 1$ if condition B holds, and $[B] = 0$ otherwise. (For the millionaires problem we may take $f(x,y) = [x < y]$, interpreting x and y as nonnegative integers.) The basic security requirements for two-party protocols, and more generally, multi-party protocols have been laid out by Yao [17,18] and by Goldreich, Micali, and Wigderson [9].

Secure computation of $f(x,y)$ thus means (i) that both players will be convinced of the correctness of the result, while (ii) neither player learns more about the other player's input than what is implied by the value of $f(x,y)$. In particular, for $f(x,y) = [x = y]$ this means that the player's do not get any information on each others values if $x \neq y$, except for the fact that the values are different. In addition, we may require the computation to be *fair*, which means that a player can not stop the protocol after getting the result of the comparison, and thereby preventing the other player from getting the result too.

For secure computations it is assumed that the players are committed to their inputs. A secure computation cannot guard against cheating players who do not behave according to their input values, e.g., by using a value $x' \neq x$ instead of x throughout the computation of $[x = y]$. Similarly, a secure computation for $f(x,y)$ does not guard against cases in which the first player tries various values $x_1, \ldots, x_n$ for x to find out more about the value of y: if the second player happens to use the same y in these computations then the first player is allowed to learn everything that is implied by the knowledge of $x_1, \ldots, x_n$ and $f(x_1, y), \ldots, f(x_n, y)$.

For two-party computations it is also known that even for basic functions such as the binary AND and OR no secure computations exist which protect the inputs of both players at the same time in an information-theoretic way. We will therefore consider secure computations of $[x = y]$ for which condition (ii) above holds subject to an intractability assumption, which will be the Decision Diffie-Hellman assumption in our case.

A solution to the millionaires' problem is described by Salomaa in [14] (and also by Schneier [16]). From this solution, we can easily deduce a solution to the *Tiercé problem*. However, such a solution is only efficient when x and y are very small (the complexity is exponential in the size of x and y). Jakobsson and Yung [10] presented a solution with polynomial complexity requiring $O(k)$ exponentiations, where k is a security parameter, using many rounds of interaction. Moreover, all of these protocols fail to meet the fairness requirement.

In this paper we present a protocol with the same properties as the protocol by Jakobsson and Yung but requiring only $O(1)$ exponentiations, and requiring a few rounds of interaction only. In addition, our protocol can be made fair, increasing the cost to $O(k)$, where k is a security parameter.

2 Assumptions and Proofs of Knowledge

2.1 Notations

Let $\mathbb{Z}_n$ denote the residue class ring modulo n and $\mathbb{Z}_n^*$ the multiplicative group of invertible elements in $\mathbb{Z}_n$. We let G_q denote a group of large prime order q, such that computing discrete logarithms in this group is infeasible. For $g, y \in G_q$, $g \neq 1$, we let $\log_g y$ denote the discrete logarithm of y to the base g, which is equal to the unique $x \in \mathbb{Z}_q$ satisfying $y = g^x$. A common construction of G_q is to take the unique subgroup of order q in $\mathbb{Z}_p^*$, where p is a large prime such that $q \mid p - 1$. Finally, we let $h : \{0,1\}^* \to \mathbb{Z}_q$ denote a cryptographic hash function, i.e., h is one-way and collision-resistant and we use $h(a, b)$ to denote the image under h of the concatenation of the strings a and b.

2.2 Assumptions

The security of our protocol involves three standard assumptions in cryptography.

The *Discrete Logarithm* (DL) assumption for group G_q states that it is infeasible to compute $\log_g y$ given random $g, y \in G_q$, $g \neq 1$. Or, more formally, for all constants c and for all sufficiently large q, there exists no probabilistic polynomial time Turing machine which, on input G_q, g, y, outputs $\log_g y$ with probability greater than $1/|q|^c$.

The *Diffie-Hellman* (DH) assumption for group G_q states that it is infeasible to compute g^{ab} given random generators $g, y_1, y_2 \in G_q$, where $a = \log_g y_1$ and $b = \log_g y_2$. Or, more concisely, it is infeasible to compute g^{ab} given g, g^a, g^b for random $a, b \in \mathbb{Z}_q$.

The *Decision Diffie-Hellman* (DDH) assumption for group G_q states that it is infeasible to decide whether $y = g^{ab}$ given random generators $g, y, y_1, y_2 \in G_q$, where $a = \log_g y_1$ and $b = \log_g y_2$. Or, more concisely, it is infeasible to decide whether $c = ab$ (which is equivalent to $g^c = g^{ab}$) given g, g^a, g^b, g^c for random $a, b, c \in \mathbb{Z}_q$.

Our result requires the DDH assumption, which implies the DH assumption, which in turn implies the DL assumption. We will also use the equivalent formulation for DH which states that it is infeasible to compute g^b given g, g^a, g^{ab} for random $a, b \in \mathbb{Z}_q$, and similarly for DDH.

The DDH assumption is equivalent to the semantic security [8] (indistinguishability of encryptions) of the ElGamal cryptosystem [6]. See [2] for a discussion of the Decision Diffie-Hellman problem.

2.3 Non-interactive proofs of knowledge

The following protocols are non-interactive *zero-knowledge* proofs of knowledge, and correspond to well-known interactive *honest-verifier zero-knowledge* proofs of knowledge. The interactive protocols are converted to their non-interactive counterparts using the generic transformation introduced by Fiat and Shamir [7]. This transformation preserves the properties of the original protocol: Bob is convinced by Alice's proof if and only if she holds the secret whose knowledge she proves, and at the end of the protocol Bob will not have learned any information on Alice's secret. In fact, the resulting non-interactive proofs can be proven secure in the random oracle model of [3] (see [13]).

In order to simplify the description of the proofs of knowledge we assume that the verifier (in this case Bob) already knows all the public parameters related to the assertion the prover (in this case Alice) wants to prove. Furthermore, to prevent that Alice and Bob may copy each other's proofs in our protocol for the socialist millionaires' problem, we assume that the inputs to the hash function are appropriately diversified, e.g. by including a string that identifies Alice or Bob and other diversifying information.

2.3.1 Proof of knowledge of a discrete logarithm

Schnorr's protocol [15] allows Alice to prove to Bob that she knows an element $x \in \mathbb{Z}_q$ satisfying $y = g^x$, where y is Alice's public key. Alice randomly selects an integer $r \in \mathbb{Z}_q$, computes $W = g^r, c = h(W)$, and $D = r - xc \bmod q$. Then Alice sends the *proof* (c, D) to Bob. Bob is convinced (accepts the *proof*) if $c = h(g^D y^c)$.

2.3.2 Proof of knowledge of discrete coordinates

Okamoto's protocol [11] extends Schnorr's protocol to the case of two generators g_1, g_2, where $\log_{g_1} g_2$ is unknown to both Alice and Bob. Using the protocol described below, Alice is able to prove to Bob that she knows $x_1, x_2 \in \mathbb{Z}_q$ satisfying $y = g_1^{x_1} g_2^{x_2}$, where y is Alice's public key. Alice randomly selects two integers $r_1, r_2 \in \mathbb{Z}_q$, computes $W = g_1^{r_1} g_2^{r_2}, c = h(W)$, $D_1 = r_1 - x_1 c \bmod q$, and $D_2 = r_2 - x_2 c \bmod q$. Then Alice sends the proof (c, D_1, D_2) to Bob. Bob is convinced if $c = h(g_1^{D_1} g_2^{D_2} y^c)$. In contrast with Schnorr's protocol, this protocol is known to be provably witness-hiding [11].

2.3.3 Proof of equality of two discrete logarithms

Consider the same setting as for Okamoto's protocol. The protocol described below allows Alice to prove to Bob that she knows an element $x \in \mathbb{Z}_q$ satisfying $y_1 = g_1^x$ and $y_2 = g_2^x$, where y_1, y_2 is Alice's public key. Alice randomly selects $r \in \mathbb{Z}_q$, computes $W_1 = g_1^r, W_2 = g_2^r, c = h(W_1, W_2)$, and $D = r - xc \bmod q$. Then Alice sends the *proof* (c, D) to Bob. Bob is convinced (accepts the *proof*) if $c = h(g_1^D y_1^c, g_2^D y_2^c)$. This protocol is due to Chaum and Pedersen [5].

2.3.4 Proof of equality of two discrete coordinates

Similar to the extension of Schnorr's protocol to Okamoto's protocol, we extend the Chaum-Pedersen protocol to the case involving several generators. The protocol described below allows Alice to prove to Bob that she knows x_1, x_{21}, x_{22} satisfying $y_1 = g_1^{x_1} g_2^{x_{21}}$ and $y_2 = g_1^{x_1} g_2^{x_{22}}$, where y_1, y_2 is Alice's public key. Alice randomly selects $r_1, r_{21}, r_{22} \in \mathbb{Z}_q$, computes $W_1 = g_1^{r_1} g_2^{r_{21}}, W_2 = g_1^{r_1} g_2^{r_{22}}, c = h(W_1, W_2)$, $D_1 = r_1 - x_1 c \bmod q$, $D_{21} = r_{21} - x_{21} c \bmod q$, and $D_{22} = r_{22} - x_{22} c \bmod q$. Alice sends the proof (c, D_1, D_{21}, D_{22}) to Bob. Bob is convinced if $c = h(g_1^{D_1} g_2^{D_{21}} y_1^c, g_1^{D_1} g_2^{D_{22}} y_2^c)$. The protocol is easily adapted to the case in which y_2 involves a different pair of generators g_1', g_2' instead of g_1, g_2.

2.3.5 Proof that a coordinate is equal to 0 or to 1

Consider the same setting as for Okamoto's protocol. Suppose that $x_2 \in \{0, 1\}$. The following protocol allows Alice to prove to Bob that she knows x_1, x_2 with $x_1 \in \mathbb{Z}_q$ and $x_2 \in \{0, 1\}$ satisfying $y = g_1^{x_1} g_2^{x_2}$, where y is Alice's public key. In particular no information on x_2 other than the fact that it is in $\{0, 1\}$ is revealed. This protocol is constructed using the technique due to Cramer, Damgård, and Schoenmakers [4].

Suppose $x_2 = v$ with $v = 0$ or $v = 1$. Alice randomly selects $r, c_{1-v}, D_{1-v} \in \mathbb{Z}_q$, computes $W_v = g_1^r, W_{1-v} = g_1^{D_{1-v}} (y/g_2^{1-v})^{c_{1-v}}, c = h(W_0, W_1), c_v = c - c_{1-v} \bmod q$, and $D_v = r - x_1 c_v \bmod q$. Alice sends the proof (c_0, c_1, D_0, D_1) to Bob. Bob is convinced if $c_0 + c_1 = h(g_1^{D_0} y^{c_0}, g_1^{D_1} (y/g_2)^{c_1}) \bmod q$.

3 The protocol

It is instructive to first consider the following naive protocol for computing $[x = y]$ securely. The players are called Alice and Bob. Let x be Alice's input and y be Bob's one. The protocol starts with Alice sending $h(x)$ to Bob, followed by Bob sending $h(y)$ to Alice, where h is a cryptographic hash function as above. If $h(x) = h(y)$, then $x = y$ with overwhelming probability, hence both players may correctly compute the output by evaluating $[h(x) = h(y)]$. However, the protocol is clearly not hiding the player's inputs, in the sense of semantic security [8]. For instance, Bob learns the value of $h(x)$ even if $x \neq y$, which enables him to test candidate values $\tilde{x}$ for x by comparing $h(\tilde{x})$ with $h(x)$. The protocol is also not fair, as Bob may refuse to return anything after receiving $h(x)$ from Alice.

The following protocol allows Alice and Bob to prove to each other that their respective secrets x and y are equal (or not) in such a way that Bob learns nothing about x and Alice learns nothing about y (except for the value of $[x = y]$). The protocol is well suited to the case that x and y are small.

3.1 Parameter generation

Alice and Bob (jointly) generate a group G_q of a large prime order q (at least of size 160 bits), e.g., by taking G_q as a subgroup of $\mathbb{Z}_p^*$ for a large prime p (say of size at least 1024 bits) such that $q \mid p-1$, or, alternatively, by taking a group (of order q) of points on an elliptic curve. They also decide on generators g_0, g_1, g_2 of G_q, for which they don't know $\log_{g_i} g_j$ for $i \neq j$, $0 \leq i, j < 3$. The generation of such numbers does not pose problems. We assume, to simplify the description of the protocol, that x and y are elements of $\mathbb{Z}_q$ (if x or y is larger than q, then we use this protocol to compare $h(x)$ and $h(y)$ instead).

3.2 Development of the protocol

Let k be a security parameter, such that it is computationally infeasible to do 2^k computations in a human-scale time and with human-scale computation resources (nowadays, k is taken equal to 80). We need that $k < |q|$. In the protocol without fairness, k is set to 0.

3.2.1 Step 1

Alice generates $g_a = g_1^{x_a}$ for random $x_a \in \mathbb{Z}_q^*$. Similarly, Bob generates $g_b = g_1^{x_b}$ for random $x_b \in \mathbb{Z}_q^*$. They use Schnorr's protocol to prove knowledge of x_a and x_b, respectively. They also check that $g_a \neq 1$ and $g_b \neq 1$. Let $g_3 = g_1^{x_a x_b} = g_a^{x_b} = g_b^{x_a}$, which can be computed by both Alice and Bob. [2]

3.2.2 Step 2

Alice selects a random element $a \in \mathbb{Z}_q$ and a random number e, $0 \leq e < 2^k$, and computes

$$(P_a, Q_a) = (g_3^a g_0^e, g_1^a g_2^x) \tag{1}$$

Using the protocols of Section 2 she shows that there indeed exists an $a \in \mathbb{Z}_q$ for which she knows $e, x \in \mathbb{Z}_q$ satisfying (1). Depending on whether we want to design a fair or a non-fair protocol, Alice and Bob perform to following:

- (Fair Version) Alice shows that she knows $a, e \in \mathbb{Z}_q$ with $0 \leq e < 2^k$ by choosing random $a_i \in \mathbb{Z}_q$ and $e_i \in \{0, 1\}$, $i = 0, \ldots, k-1$ subject to the condition that $a = \sum_{i=0}^{k-1} a_i 2^i \bmod q$ and $e = \sum_{i=0}^{k-1} e_i 2^i$, and setting $B_i = g_3^{a_i} g_0^{e_i}$, $i = 0, \ldots, k-1$. This kind of splitting has been used before in [1]. Then she proves that each e_i is in $\{0, 1\}$ using the last protocol of Section 2. Since Alice can only know one pair a, e satisfying $P_a = g_3^a g_0^e$, it follows that (P_a, Q_a) is correctly computed by Alice. If Alice finds a', e' with $a' \neq a$ mod$q$ (hence $e' \neq e$) satisfying $P_a = g_3^{a'} g_0^{e'}$ then we have $g_3 = g_0^{(e-e')/(a'-a)}$ hence $\log_{g_0} g_3$ follows which contradicts the DL assumption. Alice sends (P_a, Q_a) and the proofs over to Bob. Bob verifies the proofs and also checks that $P_a = \prod_{i=0}^{k-1} B_i^{2^i}$.

  By symmetry, Bob does the same as Alice, computing P_b, Q_b satisfying

  $$(P_b, Q_b) = (g_3^b g_0^f, g_1^b g_2^y) \tag{2}$$

  where $b \in \mathbb{Z}_q$ and f, $0 \leq f < 2^k$ are randomly chosen.
- (Version without fairness) Alice and Bob set $e = f = 0$, $P_a = g_3^a$ and $P_b = g_3^b$.

3.2.3 Step 3

In this step, Alice and Bob both compute $(P_a/P_b, Q_a/Q_b)$, which will be of the form:

$$(P_a/P_b, Q_a/Q_b) = (g_3^{a-b} g_0^{e-f}, g_1^{a-b} g_2^{x-y}) \tag{3}$$

[2] Another choice for g_3 could be $g_3 = g_a g_b$. However, for this choice the resulting protocol enables any eavesdropper to learn whether $x = y$ or not, while in the present protocol this will only be known to Alice and Bob.

Then Alice produces

$$R_a = (Q_a/Q_b)^{x_a}$$

and a proof that $\log_{g_1} g_a = \log_{Q_a/Q_b} R_a$ to show that R_a is correctly formed. Similarly, Bob produces

$$R_b = (Q_a/Q_b)^{x_b}$$

and a corresponding proof. Now Alice and Bob both know on account of (3) and the definition of g_3 that

$$R_{ab} = R_a^{x_b} = R_b^{x_a} = (Q_a/Q_b)^{x_a x_b} = g_3^{a-b} g_2^{(x-y)x_a x_b}. \tag{4}$$

3.2.4 Step 4

Finally, Alice and Bob fairly disclose the values of e and f. Once these values are released both Alice and Bob (but not anyone else) may determine whether $x = y$ by testing whether

$$P_a/P_b = R_{ab} g_0^{e-f}. \tag{5}$$

On account of (3) and (4) this equality will hold if and only if $x = y$.

To disclose e and f without revealing the values of a and b, Alice and Bob execute the following step for $i = k-1, \ldots, 1$. They send each other the values of a_i, e_i and b_i, f_i, respectively. Bob checks that $B_i = g_3^{a_i} g_0^{e_i}$ and Alice does a similar check for b_i, f_i. Subsequently, they respectively release only e_0 and f_0 (because if they reveal a_0 and b_0, they also reveal to the other party the values of $a = \sum_{i=0}^{k-1} a_i 2^i$ and $b = \sum_{i=0}^{k-1} b_i 2^i$, and consequently the values of g_2^x and g_2^y). Finally, Alice proves that she knows $\log_{g_3} B_0/g_0^{e_0}$ and Bob gives a similar proof for f_0: this convinces the other party that the bits e_0 and f_0 are correct.

This step can be regarded as fair, because if Bob (for example) deliberately aborts the protocol at $i = l$ say, he will be only at most one bit ahead of Alice to test, by exhaustive search, the combinations for the missing bits $e_0, \ldots, e_l$.

3.3 Security

We will consider the security with respect to correctness, privacy, and fairness, where we will consider Bob as the adversary and Alice as the honest party. However, we could reverse these roles as our results are symmetric in nature and hide information both ways.

For the security proofs, we will consider two different kind of attacks: passive attacks and active attacks. In the former case, the (passive) adversary correctly follows the specifications of the protocol. Such adversaries model attacks that take place after the protocol has been completed, and may involve either Alice or Bob. In the latter case, the adversary may be active during the protocol and deviate from it. In particular, he does not necessarily make random choices when it is prescribed in the definition of the protocol. Security against such an adversary means that there is no strategy that increases the amount of information that this adversary learns about the secret of the other party.

Correctness means that at the end of the protocol Alice and Bob are convinced of the validity of the result of the comparison of their respective secrets. This is achieved by the (non-interactive) proofs of knowledge given by Alice and Bob, which show that the values exchanged in the various protocol steps are of the intended form. This implies that test (5) at the end of the protocol is correct.

Privacy (or secrecy) means that the protocol hides the private inputs of Alice and Bob, which are x and y respectively. We have the following results which imply that the protocol is secure against passive attacks.

3.3.1 Security against passive attacks

Clearly, if $x = y$, Bob will learn Alice's secret. If $x \neq y$ we have to show that Bob learns no information about Alice's secret. As the (non-interactive) proofs (of knowledge) used during this protocol are zero-knowledge, the only information learnt by Bob are the following values produced by Alice: $g_a = g_1^{x_a}, P_a = g_3^a g_0^e, Q_a = g_1^a g_2^x, R_a = (Q_a/Q_b)^{x_a}, e$. Since e is released by Alice, and Bob knows x_b, b, and y, Bob essentially learns $g_1^{x_a}, T = g_1^{ax_a} = (P_a/g_0^e)^{x_b^{-1}}, g_1^a g_2^{x-y}, g_2^{(x-y)x_a} = R_a \times (P_b/g_0^f)^{x_b^{-1}}/T$.

Writing $g_1 = g$, $g_2^{x-y} = g^w$, $x_a = u$, $a = v$, and reordering, we may summarize this by saying that for a generator g, Bob essentially learns $g^u, g^{uv}, g^{uw}, g^{v+w}$.

To prove that the protocol is secure against passive attacks that *fully recover* the value of x, we must prove that Bob is not able to compute x from $g^u, g^{uv}, g^{uw}, g^{v+w}$. For this, it is sufficient to prove that he is not able to compute $g_2^{x-y} = g^w$ from $g^u, g^{uv}, g^{uw}, g^{v+w}$. By using the following lemma, we conclude that under the Diffie-Hellman assumption, the protocol is secure against passive attacks that fully recover the value of x.

Lemma 1 *Under the DH assumption it is infeasible to compute g^w from g^u, g^{uv}, g^{uw}, g^{v+w}, for random $u, v, w \in \mathbb{Z}_q$.*

Proof: Suppose we have an oracle computing g^w given g^u, g^{uv}, g^{uw}, g^{v+w}, for random $u, v, w \in \mathbb{Z}_q$. Then we show how to compute g^b given $\alpha = g^a, \beta = g^{ab}$ for random $a, b \in \mathbb{Z}_q$, hence contradicting the DH assumption. The reduction is as follows. Set $\gamma = g^{ac}$, for random $c \in \mathbb{Z}_q$, and give $\alpha, \beta, \gamma/\beta, g^c$ to the oracle. Since this tuple is equal to $g^a, g^{ab}, g^{a(c-b)}, g^c$ the oracle returns g^{c-b}, from which we obtain $g^c/g^{c-b} = g^b$.

To prove that the protocol is secure against passive attacks that only *partially recover* the value of x, we must prove that Bob is not able to decide whether a candidate value $\tilde{x}$ is equal to x (unlike the naive protocol). Writing $g^t = g_2^{\tilde{x}-y}$, it is sufficient for this to prove that Bob is not able to decide whether t is equal to w given $g^t, g^u, g^{uv}, g^{uw}, g^{v+w}$. By using the following lemma, we conclude that under the Decision Diffie-Hellman assumption, the protocol is also secure against this type of passive attacks.

Lemma 2 *Under the DDH assumption it is infeasible to decide whether $t = w$ correctly from g^t, g^u, g^{uv}, g^{uw}, g^{v+w}, for random $t, u, v, w \in \mathbb{Z}_q$.*

Proof: Suppose we have an oracle deciding $t = w$ given $g^t, g^u, g^{uv}, g^{uw}, g^{v+w}$, for random $t, u, v, w \in \mathbb{Z}_q$. Then we show how to decide whether $b = c$ given $\alpha = g^a, \beta = g^{ab}, g^c$ for random $a, b, c \in \mathbb{Z}_q$, hence contradicting the DDH assumption. The reduction is as follows. Set $\gamma = g^{ad}$, for random $d \in \mathbb{Z}_q$, and give $g^{d-c}, \alpha, \beta, \gamma/\beta, g^d$ to the oracle. Since this tuple is equal to g^{d-c}, g^a, g^{ab}, $g^{a(d-b)}$, g^d the oracle will tell whether $d - c = d - b$, from which we decide whether $b = c$.

Note that Lemma 2 may also be interpreted as follows in relation to the semantic security of ElGamal encryption. Informally, an encryption scheme is semantically secure if ciphertexts leak no information about the plaintext, i.e., the scheme is secure against a passive adversary.

Corollary 3 *Let $y = g^x$ denote the public key corresponding to private key $x \in \mathbb{Z}_q$. Suppose that the value of $m^{1/x}$ is given in addition to an ElGamal encryption $(g^r, y^r m)$ of message m, where $r \in \mathbb{Z}_q$ is random. Then ElGamal encryption is still semantically secure.*

Finally, the fairness of the fair version of our protocol is straightforward. Both Alice and Bob are unable to compute the result of the comparison before the beginning of the step 4. Moreover, during the fourth step, Bob's advantage over Alice is at most one bit. So, if Bob decides to abort the protocol and tries to search the remaining bits by exhaustive research, Alice needs no more than twice as much time compared to Bob to compute the same result.

3.3.2 Security against active attacks

So far we have only considered the case of passive adversaries. Now we will extend the arguments developed above to the case of active adversaries. We will show that our protocol remains secure even in this case, provided that g_2 is jointly computed by Alice and Bob. The *random oracle model*, formalized by Bellare and Rogaway [3], will be used to model the behaviour of the hash function h underlying the various non-interactive *zero-knowledge* proofs of our protocol. In this model, the hash function is replaced by an oracle which produces a truly random value (in the range of the function) when queried. For identical queries, the same answers are given. Various cryptographic schemes using hash functions have been proved secure in this model. In particular, Pointcheval and Stern [13] provided security proofs for signature schemes derived from *honest-verifier zero-knowledge* identification schemes.

In our proofs below, all parties (including the adversary) will be modeled by probabilistic polynomial time interactive Turing machines with access to the random oracle. As in the passive case, we will consider two types of attacks: active attacks that fully recover the secret x and active attacks that only partially recover this value. We will prove the following result.

Lemma 4 *Under the Diffie-Hellman assumption and assuming the random oracle model, the (modified) protocol for the socialist millionaires' problem is secure against active attacks that fully recover Alice's secret.*

The above result can be easily extended to the security against attacks that only partially recover the value of x.

Lemma 5 *Under the Diffie-Hellman assumption and assuming the random oracle model, the (modified) protocol for the socialist millionaires' problem is secure against active attacks that only partially recover Alice's secret.*

We will only prove the third lemma. The other lemma can be proved in a similar way, as in the passive case. Also, it suffices to consider the version of the protocol that does not address fairness, as the additions to make the protocol fair are not essential to our argument below.

For our proof, we need to slightly modify the original protocol. We require that Alice and Bob, in addition to g_3, jointly compute g_2. So, Alice generates $g_{a_2} = g_1^{x_{a_2}}$ for random $x_{a_2} \in \mathbb{Z}_q^*$. Similarly, Bob generates $g_{b_2} = g_1^{x_{b_2}}$ for random $x_{b_2} \in \mathbb{Z}_q^*$. They use Schnorr's protocol to prove knowledge of x_{a_2} and x_{b_2}, respectively. They also check that $g_{a_2} \neq 1$ and $g_{b_2} \neq 1$. Let $g_2 = g_1^{x_{a_2} x_{b_2}} = g_{a_2}^{x_{b_2}} = g_{b_2}^{x_{a_2}}$, which can be computed by both Alice and Bob.

We are now assuming that Bob is the active adversary (the analysis of the case in which Alice is the adversary is essentially the same). This means that

Bob will choose his values (x_b, x_{b_2}, b, and y) in a 'clever' way rather than truly random as specified in the protocol description. However, the messages he will send will be in accordance with the protocol. Our security proof is based on a reduction argument; we prove that if an active adversary Bob (viewed as a probabilistic polynomial time Turing machine) can find, with non-negligible probability, x, hence g_2^x, given the information 'seen' during the execution of the protocol, then this adversary can be used to build a probabilistic polynomial time Turing machine which contradicts the DH assumption (we can easily adapt the proof to the DDH setting). Here the probability is taken over the random tapes of Alice and Bob, the random oracles, the public parameters G_q, g_1 and the integer x. For simplicity, we will not write in the sequel the dependencies on the security parameter $|q|$, but when we say that an expression f is non-negligible, this means that f depends on $|q|$ and that there exists a positive integer c such that $f(|q|)$ is larger than $1/|q|^c$ for all sufficiently large $|q|$.

Proof: Let $g_1, \alpha = g_1^{x_a}, \beta = g_1^{ax_a}$ (where x_a and a are random and unknown) be an instance of the DH problem (see also Section 2.2). We want to obtain $\gamma = g_1^a$. We will see how we can use Bob to compute this value. We will 'convert' this instance to an input to our protocol, and exhibit a *simulator* $\mathcal{S}$ (a probabilistic polynomial time Turing machine) capable of simulating the three steps of our protocol in such a way that the adversary Bob cannot distinguish a real interaction with Alice from a simulated one. Bob will be used as a *resettable black box*. In other words, the *simulator* will have control over its tapes, and will have the ability to bring Bob to a halt and restart it in its starting state at any time it wishes. All the simulations will be performed under the random oracle model. $\mathcal{S}$ will play Alice's role and will speak first in the protocol.

3.3.3 Step 1

$\mathcal{S}$ sends α to Bob. Since $\mathcal{S}$ doesn't know x_a, the proof required at this step is simulated. In the random oracle model, where $\mathcal{S}$ has a full control of the values returned by the oracle, this proof can easily be simulated. In order to produce this proof, $\mathcal{S}$ randomly chooses $c \in \mathbb{Z}_q$ and $D \in \mathbb{Z}_q$. $\mathcal{S}$ then defines the output of the random oracle on the input (query) $W = g_1^D \alpha^c$ to be c (which means that $c = h(W)$). Then $\mathcal{S}$ sends the *proof* (c, D) to Bob. Note that $\mathcal{S}$ produces tuples (c, D) with an distribution identical to the one produced by a real prover knowing x_a. This is due to the *honest verifier zero-knowledge* property (*special honest verifier zero-knowledge* in fact [4]) of Schnorr's interactive protocol (see also [12] for the proof that such distributions are the same). Then Bob sends g_b to Alice along with a proof of knowledge (c_b, D_b) (in the random oracle model) of x_b the discrete logarithm of g_b to the base g_1; the *proof* (c_b, D_b) is correct if $c_b = h(g_1^{D_b} g_b^{c_b})$, where c_b corresponds to the answer of the random oracle to the query $g_1^{D_b} g_b^{c_b}$. If this proof is not correct $\mathcal{S}$ aborts the protocol. At this point $\mathcal{S}$ needs to obtain the discrete logarithm x_b in order to carry on with its simulation. By using the technique developed by Pointcheval and Stern [13] and known as the *oracle replay attack* (forking lemma) one can easily obtain this value: if we replay Bob, with the same random tape and a different oracle, Bob will produce, with non-negligible probability and in polynomial time, two valid proofs (c_b, D_b) and $(\dot{c}_b, \dot{D}_b)$ with $c_b \neq \dot{c}_b (\mathrm{mod} q)$ such that

$$g_1^{D_b} g_b^{c_b} = g_1^{\dot{D}_b} g_b^{\dot{c}_b}$$

holds. From this equation, $\mathcal{S}$ can compute $x_b = (\dot{D}_b - D_b)/(c_b - \dot{c}_b) \bmod q$. Hence $\mathcal{S}$ can also compute $g_3 = g_1^{x_a x_b} = \alpha^{x_b} = g_b^{x_a}$, even though it does not know x_a. $\mathcal{S}$ then generates $g_{a_2} = g_1^{x_{a_2}}$ for random $x_{a_2} \in \mathbb{Z}_q^*$ and uses Schnorr's protocol to prove knowledge of x_{a_2} (since $\mathcal{S}$ really knows x_{a_2} this is a real proof not a simulated one). Bob then sends g_{b_2} to Alice along with a proof of knowledge (in the random oracle model) of x_{b_2} the discrete logarithm of g_{b_2} to the base g_1. Again, by using the oracle replay attack, $\mathcal{S}$ can find the value x_{b_2}. Let $g_2 = g_1^{x_{a_2} x_{b_2}} = g_{a_2}^{x_{b_2}} = g_{b_2}^{x_{a_2}}$. So at the end of step 1, Alice knows x_b and x_{b_2}.

3.3.4 Step 2

$\mathcal{S}$ randomly selects an element $d \in \mathbb{Z}_q$ and computes

$$(P_a, Q_a) = (g_3^a, g_1^d)$$

So, we have $P_a = g_3^a = g_1^{a x_a x_b} = \beta^{x_b}$. Again, $\mathcal{S}$ can compute this value since it extracted x_b from Bob. Following the definition of g_2, we have:

$$Q_a = g_1^d = g_1^a g_1^{d-a} = g_1^a g_2^{x_{a_2}^{-1} x_{b_2}^{-1} (d-a)}$$

We put $x = x_{a_2}^{-1} x_{b_2}^{-1} (d - a)$, where the simulator does not know a. Since d is a random element of $\mathbb{Z}_q$, x is uniformly distributed in $\mathbb{Z}_q$ and consequently constitutes a choice that a real Alice could have made. $\mathcal{S}$ sends (P_a, Q_a) to Bob and must also prove that it knows a and x satisfying $(P_a, Q_a) = (g_3^a, g_1^a g_2^x)$. Since $\mathcal{S}$ does not know a, the proof required at this step is simulated. In order to produce this proof, $\mathcal{S}$ randomly chooses $c, D_1, D_2 \in \mathbb{Z}_q$, and then defines the output of the random oracle on input W_1, W_2 with $W_1 = g_3^{D_1} P_a^c$ and $W_2 = g_1^{D_1} g_2^{D_2} Q_a^c$ to be c (hence $c = h(W_1, W_2)$). With overwhelming probability, Bob has not yet already queried the random oracle at this point. Then $\mathcal{S}$ sends the *proof* (c, D_1, D_2) to Bob. Again the *special honest verifier zero-knowledge* property of the interactive protocol underlying this proof of knowledge ensures that $\mathcal{S}$ produces tuples (c, D_1, D_2) with a distribution indistinguishable from one produced by a real prover knowing a and x. Next, Bob sends P_b, Q_b to $\mathcal{S}$

along with a proof of knowledge of $b, y \in \mathbb{Z}_q$ satisfying $P_b = g_3^b$ and $Q_b = g_1^b g_2^y$. Using the oracle replay attack, $\mathcal{S}$ can find b and y (note that $\mathcal{S}$ now knows x_b, x_{b_2}, b and y).

3.3.5 Step 3

In this step, $\mathcal{S}$ and Bob both first compute $(P_a/P_b, Q_a/Q_b)$, which will be of the form:

$$(P_a/P_b, Q_a/Q_b) = (g_3^{a-b}, g_1^{a-b} g_2^{x-y})$$

Then $\mathcal{S}$ computes:

$$R_a = (Q_a/Q_b)^{x_a} = g_1^{dx_a} g_1^{-bx_a} g_2^{-yx_a} = g_1^{dx_a} g_1^{-bx_a} g_1^{-x_{a_2} x_{b_2} y x_a} = \alpha^{d-b-x_{a_2} x_{b_2} y}$$

$\mathcal{S}$ can compute this value since it knows d, b, y, x_{a_2} and x_{b_2}. $\mathcal{S}$ sends R_a to Bob along with a proof that $\log_{g_1} \alpha = \log_{Q_a/Q_b} R_a$. Since $\mathcal{S}$ does not know x_a, the proof required at this step is simulated. To produce this proof,$\mathcal{S}$ randomly chooses $c, D \in \mathbb{Z}_q$. $\mathcal{S}$ then defines the output of the random oracle on input W_1, W_2 with $W_1 = g_1^D \alpha^c$ and $W_2 = (Q_a/Q_b)^D R_a^c$ to be c (hence $c = h(W_1, W_2)$). $\mathcal{S}$ then sends the *proof* (c, D) to Bob. As before, the *special honest verifier zero-knowledge* property of the interactive protocol underlying this proof of knowledge ensures that $\mathcal{S}$ produces tuples (c, D) with a distribution indistinguishable from those that would be produced by a real prover knowing x_a. The *simulator*'s part of the protocol is now complete.

Since the distribution of the simulated views is indistinguishable from that produced by a 'real' Alice (not a simulated one), Bob, after the interaction with $\mathcal{S}$, will be able, as assumed, to find with non-negligible probability g_2^x, which is equal to g_1^{d-a}. Since $\mathcal{S}$ knows d, $\mathcal{S}$ can find $\gamma = g_1^a = g_1^d / g_1^{d-a}$, hence contradicting the DH assumption. Consequently, such an adversary Bob cannot find g_2^x hence cannot fully recover x.

4 Conclusion

We have presented a protocol which allows one to *fairly* check the equality (or the inequality) of two secrets x and y, and this gives an answer to the *Tiercé problem*. Counting the number of exponentiations, the complexity of our protocol is $O(1)$ in its simple version (without fairness) and $O(k)$ in its fair version, where k is a security parameter. However, designing an efficient solution to the *millionaires' problem*, which asks one to test whether $x < y$, remains an open problem.

Acknowledgements

The authors would like to thank Gilles Zémor for stating the *Tiercé problem*, and Marc Girault and Martijn Stam for helpful comments.

References

[1] M. Bellare and S. Goldwasser: *Verifiable partial key escrow.* 4th Annual Conference on Computer and Communications Security, 1997. Earlier version Technical Report CS95-447, Department of Computer Science and Engineering, University of California at San Diego, October 1995.

[2] D. Boneh: *The decision Diffie-Hellman problem.* Third Algorithmic Number Theory Symposium, volume 1423 of Lecture Notes in Computer Science, pages 48–63, Berlin, 1998. Springer-Verlag.

[3] M. Bellare and P. Rogaway. *Random oracles are practical: A paradigm for designing efficient protocols.* 1st ACM Conference on Computer and Communications Security, Fairfax, November 1993.

[4] R. Cramer, I. Damgård, and B. Schoenmakers: *Proofs of partial knowledge and simplified design of witness hiding protocols.* Advances in Cryptology—CRYPTO '94, volume 839 of Lecture Notes in Computer Science, pages 174–187, Berlin, 1994. Springer-Verlag.

[5] D. Chaum and T. P. Pedersen: *Wallet databases with observers.* Advances in Cryptology—CRYPTO '92, volume 740 of Lecture Notes in Computer Science, pages 89–105, Berlin, 1993. Springer-Verlag.

[6] T. ElGamal: *A public key cryptosystem and a signature scheme based on discrete logarithms.* IEEE Trans. Inform. Theory, 31:469–472, 1985.

[7] A. Fiat and A. Shamir: *How to prove yourself: Practical solutions to identification and signature problems.* Advances in Cryptology—CRYPTO '86, volume 263 of Lecture Notes in Computer Science, pages 186–194, New York, 1987. Springer-Verlag.

[8] S. Goldwasser and S. Micali: *Probabilistic encryption.* Journal of Computer and System Sciences, 28(2):270–299, 1984.

[9] O. Goldreich, S. Micali, and A. Wigderson: *How to play any mental game – or – a completeness theorem for protocols with honest majority.* Proc. 19th Symposium on Theory of Computing (STOC '87), pages 218–229, New York, 1987. A.C.M.

[10] M. Jakobsson and M. Yung: *Proving without knowing: On oblivious, agnostic and blindfolded provers.* Advances in Cryptology—CRYPTO '96, volume 1109 of Lecture Notes in Computer Science, pages 186–200, Berlin, 1996. Springer-Verlag.

[11] T. Okamoto: *Provably secure and practical identification schemes and corresponding signature schemes.* Advances in Cryptology—CRYPTO '92, volume 740 of Lecture Notes in Computer Science, pages 31–53, Berlin, 1993. Springer-Verlag.

[12] D. Pointcheval: *Les Preuves de Connaissance et leurs Preuves de Sécurité.* PhD thesis, Université de Caen, France, 1996.

[13] D. Pointcheval and J. Stern: *Security proofs for signature schemes.* Advances in Cryptology— EUROCRYPT '96, volume 1070 of Lecture Notes in Computer Science, pages 387–398, Berlin, 1996. Springer-Verlag.

[14] A. Salomaa: *Public-Key Cryptography.* Springer-Verlag, 1990.

[15] C. P. Schnorr: *Efficient signature generation by smart cards.* Journal of Cryptology, 4(3):161–174, 1991.

[16] B. Schneier: *Applied Cryptography.* John Wiley & Sons, 1996.

[17] A. Yao: *Protocols for secure computations.* Proc. 23rd IEEE Symposium on Foundations of Computer Science (FOCS '82), pages 160–164. IEEE Computer Society, 1982.

[18] A. Yao: *How to generate and exchange secrets.* Proc. 27th IEEE Symposium on Foundations of Computer Science (FOCS '86), pages 162–167. IEEE Computer Society, 1986.

[19] G. Zémor: *Private communication.* 1998.