A Cost-Effective Pay-Per-Multiplication Comparison Method for Millionaires

Marc Fischlin

Fachbereich Mathematik (AG 7.2)
Johann Wolfgang Goethe-Universität Frankfurt am Main
Postfach 111932
60054 Frankfurt/Main, Germany

marc@mi.informatik.uni-frankfurt.de
http://www.mi.informatik.uni-frankfurt.de/

Abstract. Based on the quadratic residuosity assumption we present a non-interactive crypto-computing protocol for the greater-than function, i.e., a non-interactive procedure between two parties such that only the relation of the parties' inputs is revealed. In comparison to previous solutions our protocol reduces the number of modular multiplications significantly. We also discuss applications to conditional oblivious transfer, private bidding and the millionaires' problem.

1 Introduction

Yao's famous millionaires' problem [19, 20] consists of two millionaires trying to compare their riches but without disclosing their assets. General secure two-party protocols computing the greater-than function $\mathrm{GT}(x, y) = [x > y]$ provide a solution to this problem [11, 20, 10].[1] Unfortunately, these protocols are rather inefficient. Recently, Boudot et al. [2] proposed a quite efficient protocol for the *socialist* millionaires' problem in which both parties test their inputs for equality only, i.e., compute the function $\mathrm{EQ}(x, y) = [x = y]$ securely. This scheme needs a (quite large, but) constant number of modular exponentiations, and improves the protocol of Jakobsson and Yung [15] requiring $\Theta(k)$ modular exponentiations for security parameter k.

Related to the millionaires' problem is the problem of non-interactive crypto-computing for the greater-than function which we address in this paper. A non-interactive crypto-computing protocol [18] is a two-party protocol where the client encrypts his input y and sends the encryption to the server. The server inattentively evaluates a secret circuit C on this encrypted input and returns it to the client. The client extracts the circuit's output $C(y)$ but learns nothing more about the circuit C. In the case of the millionaires' problem think of the server's circuit as computing the greater-than function with partially fixed input:

[1] The predicate $[x > y]$ stands for 1 if $x > y$ and 0 otherwise —interpreting bit strings x, y as numbers.

$C(\cdot) = \mathrm{GT}(x, \cdot)$. Keeping the circuit secret implies that nothing about the input x except for $[x > y]$ is leaked.

As an application of such non-interactive crypto-computing consider a company that offers groceries or airline tickets over the Internet. The company tries to optimize its profit but is willing to sell an airline ticket above the breakeven point y. Hence, they ask the customer how much he would pay for the ticket, and if the customer's offer x exceeds y then they clinch the deal. This, of course, requires that y is hidden or else the customer will simply set x to $y + 1$. On the other hand, the customer, too, would like to keep his offer x secret unless the deal is made and he has to reveal x anyway.

Using a non-interactive crypto-computing protocol the company plays the client and publishes its bound y by encrypting it. The customer computes the circuit for the greater-than function with his bid x and returns it to the company (along with a commitment of x). The company decrypts and verifies that $x > y$ and, if so, declares that it sells and asks the customer to reveal his bid and to decommit. If the customer then refuses or decommits incorrectly then the company may blacklist him, at least for a certain time. Or, the customer is obliged to attach a signature to the commitment of x which binds him to his offer in case of a dispute.

In the example above, a 'clever' customer could simply find out the corporation's bound y and buy for $x = y + 1$ by bidding $1, 2, 3, \ldots$ until the company announces the deal. The most obvious countermeasure against such a behavior is to allow the customer to bid only once within a certain period (say, a couple of minutes for stocks, a week for flight tickets, etc). Alternatively, the offerer may raise the bound y with each failing bid of the customer. Details depend on the application.

General non-interactive crypto-computing protocols for various classes of circuits appear in [18, 1]. If one applies the general result in [18, 1] to compute the GT-function with the straightforward circuit (this circuit is also optimal when applying [18, 1], as far as we know) and using the quadratic-residuosity bit-encryption of Goldwasser and Micali (see [12] or Section 2.1), then this requires at least n^4 modular multiplications in $\mathbb{Z}_N^*$ for inputs $x, y \in \{0, 1\}^n$. The general solutions in [5, 16, 4] involve $\Omega(n)$ modular exponentations with constants larger than 3 (and on the average therefore at least $4.5kn$ multiplications[2] for security parameter k). Hence, as opposed to [18, 1] and, as we will see, to our solution, the computational complexity of the latter protocols depends on the length of security parameter and grows whenever we switch to larger k. For these protocols, $k = 160$ seems to be an appropriate choice today.

In contrast to the general approach in [18] our starting point is the logic formula describing the GT-function. As we will see, this formula can be converted into a protocol that utilizes the homomorphic operations supported by the Goldwasser-Micali system. For instance, we take advantage of the AND-

[2] Neglecting preprocessing. Yet, preprocessing is not always applicable in these protocols. Futhermore, Naor et al. [16] accomplish the constant 3 in the random oracle model only; otherwise the bound becomes 5.

homomorphic variant of the Goldwasser-Micali scheme presented in [18]. To best of our knowledge this is the first application of this AND-homomorphic encryption scheme.

Our protocol takes at most $6n\lambda$ modular multiplications in $\mathbb{Z}_N^*$ for the server and $2n$ for the client (once an RSA-modulus N has been generated and neglecting the less expensive effort for GM-decoding), where n is the bit length of each input x, y and λ determines the error of the protocol (the error is $5n \cdot 2^{-\lambda}$). Note that the number of multiplications in our protocol does not depend on k, but rather on the input length n and the absolute error parameter λ.

Allowing a small error of, say, 2^{-40}, for inputs of 15 bits our solution requires about $4,000$ multiplications for the server instead of $15^4 \approx 50,000$ as in [18, 1] and $\geq 10,800$ for [5, 16, 4] for $k = 160$. If we have 20-bit inputs (which occurs if we compare time data, for instance) then for error 2^{-40} our protocol needs $6,000$ multiplications for the server compared to $20^4 = 160,000$ in [18, 1] or $\geq 14,400$ in [5, 16, 4].

Finally, we discuss consequences to related protocols. The first observation is that our protocol can also be applied to functions that reduce to the greater-than function, e.g., any comparison function that can be described by $\mathrm{COMP}_{a,b}(x, y) = [ax+b > y]$ for public constants a, b. In particular, the greater-or-equal-to function equals $\mathrm{GT}(x + 1, y)$. This possibly increases the number of multiplications since the bit size of $ax + b$ might be larger than n bits, yet the number of multiplications in our protocol grows quasi linear with the bit size for fixed error level.

Another interesting application of our non-interactive protocols are *conditional* oblivious transfer protocols. Introduced by Di Crescenzo et al. [7], with such a protocol, instead of obliviously transferring a bit b to a receiver with probability 1/2, the sender transfers the bit given that a predicate over additional private inputs x (of the sender) and y (of the receiver) is satisfied. We devise such a protocol for the greater-than predicate that, in contrast to the solution in [7], keeps the server's input x secret. That is, the receiver learns the predicate $[x > y]$ but nothing else about x, and moreover gets the bit b if and only if $x > y$. The sender, on the other side, does not learn anyhing about y and in particular does not come to know if b has been transferred. It is worth mentioning that we are not aware if the general non-interactive crypto-computing protocol in [18] for the greater-than function can be used to derive such an oblivious transfer protocol. See Section 4 for details.

As for further implications, we have already mentioned the connection to the millionaires' problem and we elaborate further in Section 4.2. Aiming at a similar problem, we show how to construct improved private-bidding protocols with an oblivious third party [3]. These are protocols where two parties compare their bids. For this, a third party helps to compute the result and thereby guarantees fairness. Yet, the third party remains oblivious about the outcome of the bidding.

2 Preliminaries

We denote by $\mathbb{Z}_N$ the ring of intergers modulo N and by $\mathbb{Z}_N^*$ the elements in $\mathbb{Z}_N$ relatively prime to N. Let $\mathbb{Z}_N^*(+1)$ denote the subset of $\mathbb{Z}_N^*$ that contains all elements with Jacobi symbol $+1$. By $\mathrm{QR}_N \subset \mathbb{Z}_N^*(+1)$ and $\mathrm{QNR}_N \subset \mathbb{Z}_N^*(+1)$ we refer to the quadratic residues and non-residues, respectively. See [14] for number-theoretic background.

In the sequel we write $\mathrm{Enc}(b, r)$ for the output of the encryption algorithm Enc for bit b and randomness r. Let $\mathrm{Enc}(b)$ denote the corrresponding random variable, and $\mathrm{Enc}_0(b)$ a fixed sample from $\mathrm{Enc}(b)$ (where the random string r is irrelevant to the context). We write $c \leftarrow \mathrm{Enc}(b)$ for the process of encrypting a bit b randomly and assigning the result to c.

We sometimes switch between bit strings and numbers in a straightforward way. That is, for a bit string $x = x_1 \dots x_n$ we associate the number $\sum x_i 2^{i-1}$ and vice versa. In particular, x_n is the most significant bit and x_1 the least significant one.

2.1 Goldwasser-Micali Encryption Scheme

In [12] Goldwasser and Micali introduced the notion of semantic security for encryption schemes and presented such a semantically-secure scheme based on the quadratic residuosity assumption. Namely, the public key consists of an RSA-modulus $N = pq$ of two equally large primes p, q, and a quadratic non-residue $z \in \mathbb{Z}_N^*(+1)$. To encrypt a bit b choose a random $r \in \mathbb{Z}_N^*$ and set $\mathrm{Enc}(b, r) := z^b r^2 \bmod N$. If and only if $b = 1$ then this is a quadratic non-residue. And this can be recognized efficiently given the secret key, i.e., the factorization p, q of N. In contrast, deciding quadratic residuosity without knowledge of the factorization is believed to be hard, i.e., the quadratic-residuosity-assumption says that infeasible to significantly distinguish between 0- and 1-encryptions given only N and z.

Let us recall some useful facts about the GM-encryption scheme. First, the GM-scheme has nice homomorphic properties which allow to compute the exclusive-or of two encrypted bits and to flip an encrypted bit. Second, it is rerandomizable, i.e., given a ciphertext of an unknown bit b and the public key only, one can generate a uniformly distributed ciphertext of b.

- xor-property: $\mathrm{Enc}_0(b) \cdot \mathrm{Enc}_0(b') = \mathrm{Enc}_0(b \oplus b') \bmod N$
- not-property: $\mathrm{Enc}_0(b) \cdot z = \mathrm{Enc}_0(b \oplus 1) \bmod N$
- rerandomization: $\mathrm{Rand}(\mathrm{Enc}_0(b)) := \mathrm{Enc}_0(b) \cdot \mathrm{Enc}(0) \bmod N$ is identically distributed to $\mathrm{Enc}(b)$

Another important property of the GM-system is that it can be turned into an AND-homomorphic one over $\{0, 1\}$ (cf. [18]): Let k be a security parameter and λ a sufficiently large function such that $2^{-\lambda}$ is small enough; we will discuss the choice of λ afterwards. To encrypt a bit b we encode $b = 1$ as a sequence of λ random quadratic residues (i.e., as λ GM-encryptions $\mathrm{Enc}(0)$), and $b = 0$ as a sequence of λ random elements from $\mathbb{Z}_N^*(+1)$ (i.e., as λ GM-encryptions $\mathrm{Enc}(a_i)$ for random bits $a_1, \ldots, a_{\lambda(k)}$). We denote this encryption algorithm by $\mathrm{Enc}^{\mathrm{AND}}$ and adopt the aforementioned notations $\mathrm{Enc}^{\mathrm{AND}}(b), \mathrm{Enc}^{\mathrm{AND}}(b,r), \mathrm{Enc}_0^{\mathrm{AND}}(b)$.

The decryption process takes a sequence of λ elements from $\mathbb{Z}_N^*(+1)$ and returns 1 if all elements are quadratic residues, and 0 otherwise (i.e., if there is a quadratic non-residue among those elements). Note that there is a small probability of $2^{-\lambda}$ that a 0-bit is encrypted as a sequence of λ quadratic residues, and thus that decryption does not give the desired result. Choosing λ sufficiently large this almost never happens. In practice setting λ to 40 or 50 should be sufficient.

Next we explain how $\mathrm{Enc}^{\mathrm{AND}}$ supports the AND-operation (with some small error). Given two encryptions $\mathrm{Enc}_0^{\mathrm{AND}}(b)$ and $\mathrm{Enc}_0^{\mathrm{AND}}(b')$ of bits b, b' we claim that the componentwise product $\bmod N$ is an encryption of $b \wedge b'$ (except with error $2^{-\lambda}$ over the choice of the randomness in the encryption process). Clearly, this is true if at least one of the sequences represents a 1-encryption, because multiplying this sequence with the other one does not change the quadratic character of the elements in the other sequence. If $b = b' = 0$, though, the quadratic non-residues in both sequences can accidentally cancel out. But again this happens with probability $2^{-\lambda}$ only.

A crucial observation for our protocol is that we can embed a basic GM-encryption into an AND-homomorphic one. This embedding is done as follows: given $\mathrm{Enc}_0(b)$ first flip the encapsulated bit b by multiplying the encryption with z. Then, generate a sequence of λ basic encryptions by letting the i-th sample be either $\mathrm{Rand}(z\,\mathrm{Enc}_0(b))$ or $\mathrm{Enc}(0)$ with probability 1/2. If $b = 1$, and therefore $z\,\mathrm{Enc}_0(b) \in \mathrm{QR}_N$, then the result is identically distributed to $\mathrm{Enc}^{\mathrm{AND}}(1)$. For $b = 0$ we generate a sequence of random quadratic residues and random non-residues (since $z\,\mathrm{Enc}_0(b) \in \mathrm{QNR}_N$), identically distributed to $\mathrm{Enc}^{\mathrm{AND}}(0)$.

2.2 Non-Interactive Crypto-Computing

We have already outlined the problem of non-interactive crypto-computing protocols in the introduction. We give a very succinct description; for a more formal definition see [18]. Also, our definition merely deals with honest-but-curious parties, i.e., parties that follow the prescribed program but try to gain advantage from listening. The general case of dishonest parties is discussed afterwards.

Recall that a non-interactive crypto-computing protocol consists of two parties, the client (with input y) and the server (possessing a circuit C), such that the client sends a single message to the server and receives a single message as reply. The following holds:

- completeness: for any input y and any circuit C the honest client is able to extract the value $C(y)$ from the answer of the honest server.
- (computational) privacy for the client: the client's message for input y is not significantly distinguishable from a message generated for any other input y' of the same length

– (perfect) privacy for the server: the distribution of the server's answer depends only on the circuit's output $C(y)$ (where y is the message that the honest client sends encrypted).

A dishonest client could cheat by sending incorrect encryptions or system parameters. In case of the Goldwasser-Micali scheme the client should therefore send also a non-interactive proof of correctness for the modulus N and the non-redisue z; in practice these parameters are likely to be certified by some trusted authority anyway, and an additional correctness proof is redundant. Moreover, it is easy to see that, once the parameters' correctness is approved, the server can check that the client sends n encrypted bits by verifying that the n transmitted elements belong to $\mathbb{Z}_N^*(+1)$.

As for dishonest servers, Sander et al. [18] suggest that the server publishes a pair $(y_0, C(y_0))$. The client may now ask about several input pairs where each pair consists of encryptions of y and y_0 in random order. It can then be checked that the server answers consistently. Yet, the client must now also prove that each pair equals encryptions of y, y_0 for the *same* y.

We stress that in some settings neither party seems to gain any noticeable advantage by deviating from the protocol. Recall the flight ticket example. As explained, the company as the client in the crypto-computing protocol essentially cannot cheat if it uses certified system parameters. And, since it wants to sell its product, it is likely to announce the deal if it later decrypts and finds out that the bid is high enough; since nothing else than this relationship is leaked about the bid, there does not seem to be a point in delaying the deal to wait for better offers. Similarly, if the customer sends garbage the company may blacklist him.

3 Non-Interactive Crypto-Computing for Comparison

In this section we present our protocol for non-interactively computing the greater-than function. As discussed before, we only deal with the case of honest-but-curious client and server. Clearly, a number x is greater than another number y if, for some i, we have $x_i = 1$ and $y_i = 0$ and $x_j = y_j$ for all more significant bits for $j = i+1, \ldots, n$. More formally,

$$[x > y] \quad :\Longleftrightarrow \quad \bigvee_{i=1}^{n} \Big(x_i \wedge \neg y_i \wedge \bigwedge_{j=i+1}^{n} (x_j = y_j) \Big) \tag{1}$$

Note that $x_j = y_j$ can be written as $\neg(x_j \oplus y_j)$ and that both operations $\oplus, \neg$ can be easily implemented for the basic GM-scheme.

The disjunction in Expression (1) is an exclusive OR, i.e., only one implicant is satisfiable simultanously. This suggests the following strategy to compute the formula: we process each implicant individually and compute $x_i \wedge \neg y_i \wedge \bigwedge_{j=i+1}^{n}(x_j = y_j)$ using the basic and the AND-homomorphic GM-system, respectively, for $\oplus, \neg$ and AND. This is done by performing the computations for $x_j = y_j$ and $\neg y_i$ with the basic GM-encryption scheme, and by embedding the results and the encryption for x_i into the extended system and computing the AND. Then we permute the resulting n encryptions of the implicants and output them. If and only if $[x > y]$ then there exists a random AND-GM-encryption of 1, and this appears at a random position. Otherwise we have a sequence of random 0-encryptions. The protocol is given in Figure 1.[3]

In Figure 2 we reduce the number of multiplications. This is done by first noting that one can re-use the encryption for $\bigwedge_{j=i+1}^{n}(x_j = y_j)$ from stage i for stage $i-1$. That is, for $i = n, n-1, \ldots, 1$ we compute and store the product P_i^{AND} representing $\bigwedge_{j=i+1}^{n}(x_j = y_j)$. Then we merely compute the AND of this stored encryptions with the ciphertext for $x_i = y_i$. Moreover, if we know x explicitly then we can compute $x_i = y_i$ directly by $\mathrm{Enc}_0(y_i) \cdot z^{1-x_i}$ instead of first encrypting x and then computing $\mathrm{Enc}_0(y_i) \cdot \mathrm{Enc}_0(x_i)$. Finally, we remark that we only need to rerandomize one of the encryptions of X_i^{AND}, $\bar{Y}_i^{\mathrm{AND}}$ and P_i^{AND} in order to rerandomize each product $T_i^{\mathrm{AND}} = X_i^{\mathrm{AND}} \cdot \bar{Y}_i^{\mathrm{AND}} \cdot P_i^{\mathrm{AND}}$. In Figure 2 this is done for X_i^{AND}, whereas for $\bar{Y}_i^{\mathrm{AND}}$ and E_i^{AND} we choose the fixed quadratic residue $1 \in \mathrm{QR}_N$ instead of a random GM-encryption $\mathrm{Enc}(0)$ when embedding.

Some easy but important observations follow:

Lemma 1. *For inputs $x, y \in \{0,1\}^n$ and security and error parameter k and λ the protocol in Figure 1 with the optimized server algorithm in Figure 2 satisfies:*

- *for $x \leq y$ the T_i^{AND} are all random AND-encryptions $\mathrm{Enc}^{AND}(0)$ of 0.*
- *for $x > y$ there exists exactly one uniformly distributed i for which T_i^{AND} represents a random 1-encryption $\mathrm{Enc}^{AND}(1)$; for all other $j \neq i$ we have random 0-encryptions $\mathrm{Enc}^{AND}(0)$ for T_j^{AND}.*
- *the evaluation takes only $6n\lambda$ multiplications in $\mathbb{Z}_N^*$ for the server in the worst case, and $5n\lambda$ multiplications on the average (over the random choices in the encryption process).*
- *the error is at most $5n \cdot 2^{-\lambda}$.*

We thus derive the following result:

Theorem 1. *The protocol in Figure 1 and Figure 2 constitutes a non-interactive crypto-computing protocol for $\mathrm{GT}(x,y) = [x > y]$ such that for inputs $x, y \in \{0,1\}^n$ and security parameter k and error parameter λ the client has to perform $2n$ modular multiplications (plus the number of multiplications to generate a GM-instance N, z for security parameter k in a preprocessing step) and the server has to carry out at most $6n\lambda$ multiplications. The error of the protocol is at most $5n \cdot 2^{-\lambda}$.*

[3] We remark that the fact that we do not have to compute the disjunction explicitly supports our improved protocol. Otherwise we would have to compute the OR of AND-encryptions which we do not know how to do without blowing up the number of multiplications like in [18].

Fig. 1. Non-Interactive Crypto-Computing for GT

security parameter k, error parameter λ

Client's algorithm I:

- generate GM-instance N, z for security parameter k
- encrypt input y bit-wise: $Y_i \leftarrow \mathrm{Enc}(y_i)$ for $i = 1, \ldots, n$
- send $N, z, Y_1, \ldots, Y_n$ to server

Server's algorithm:

- receive $N, z, Y_1, \ldots, Y_n$ from client
- encrypt input x bit-wise: $X_i \leftarrow \mathrm{Enc}(x_i)$ for $i = 1, \ldots, n$
- compute encryptions of $e_i = [x_i = y_i] = \neg(x_i \oplus y_i)$:
  for all $i = 1, \ldots, n$ compute $E_i = Y_i \cdot X_i \cdot z \bmod N$
- embed E_i into extended encryptions E_i^{AND}:
  for all $i = 1, \ldots, n$ set $E_i^{\mathrm{AND}} := (E_{i,1}^{\mathrm{AND}}, \ldots, E_{i,\lambda}^{\mathrm{AND}})$, where $E_{i,j}^{\mathrm{AND}} \leftarrow \mathrm{Rand}(zE_i)$ or $\mathrm{Enc}(0)$, the choice made by a fair coin flip.
- embed encryptions X_i and $\bar{Y}_i$ of x_i and $\neg y_i$ into encryptions X_i^{AND} and $\bar{Y}_i^{\mathrm{AND}}$:
  for all $i = 1, \ldots, n$ set $X_i^{\mathrm{AND}} := (X_{i,1}^{\mathrm{AND}}, \ldots, X_{i,\lambda}^{\mathrm{AND}})$, where $X_{i,j}^{\mathrm{AND}} \leftarrow \mathrm{Rand}(zX)$ or $\mathrm{Enc}(0)$, the choice made by a fair coin flip.
  for all $i = 1, \ldots, n$ set $\bar{Y}_i^{\mathrm{AND}} := (\bar{Y}_{i,1}^{\mathrm{AND}}, \ldots, \bar{Y}_{i,\lambda}^{\mathrm{AND}})$, where $\bar{Y}_{i,j}^{\mathrm{AND}} \leftarrow \mathrm{Rand}(Y_i)$ or $\mathrm{Enc}(0)$, the choice made by a fair coin flip.
- compute terms $t_i := [x_i \wedge \neg y_i \wedge \bigwedge_{j=i+1}^{n} x_j = y_j]$:
  for $i = 1, \ldots, n$ let $T_i^{\mathrm{AND}} = X_i^{\mathrm{AND}} \cdot \bar{Y}_i^{\mathrm{AND}} \cdot \prod_{j=i+1}^{n} E_j^{\mathrm{AND}} \bmod N$
- randomly permute $T_1^{\mathrm{AND}}, \ldots, T_n^{\mathrm{AND}}$ and return them to the client

Client's algorithm II:

- receive n sequences of λ elements from $\mathbb{Z}_N^*$ from server
- if there exists a sequence of λ quadratic residues then output '$x > y$', else output '$x \leq y$'.

4 Applications

In the previous section, we have shown how to compute the function $\mathrm{GT}(x, y) = [x > y]$ with few modular multiplications. Here, we discuss several applications of this result.

4.1 Conditional Oblivious Transfer

With an oblivious transfer protocol [17] a sender hands with probability 1/2 a secret bit to a receiver such that the sender remains oblivious about the fact whether the receiver has actually learned the bit or not. As for a *conditional* oblivious transfer [7], the random choice is replaced by a predicate evaluation depending on some additional private inputs of both parties. For example, in [7] such a protocol has been used to derive a time-release encryption scheme where

Fig. 2. Optimized Non-Interactive Crypto-Computing for GT

security parameter k, error parameter λ

Optimized server algorithm:

- receive $N, z, Y_1, \ldots, Y_n$ from client
- embed input x into extended encryptions X_i^{AND}:
  for all $i = 1, \ldots, n$ let $X_i^{\mathrm{AND}} := (X_{i,1}^{\mathrm{AND}}, \ldots, X_{i,\lambda}^{\mathrm{AND}})$, where $X_{i,j}^{\mathrm{AND}} \leftarrow \mathrm{Enc}(z^{1-x_i})$ or $\mathrm{Enc}(0)$, the choice made by a fair coin flip.
- embed $[x_i = y_i]$ into extended encryptions E_i^{AND}:
  for all $i = 1, \ldots, n$ set $E_i^{\mathrm{AND}} := (E_{i,1}^{\mathrm{AND}}, \ldots, E_{i,\lambda}^{\mathrm{AND}})$, where $E_{i,j}^{\mathrm{AND}} := Y_j \cdot z^{x_i} \bmod N$ or $1 \in \mathrm{QR}_N$, the choice made by a fair coin flip.
- compute extended encryptions P_i^{AND} of $p_i = \bigwedge_{j=i+1}^{n} [x_j = y_j]$:
  for $i = n-1, \ldots, 1$ let $P_i^{\mathrm{AND}} := P_{i+1}^{\mathrm{AND}} \cdot E_{i+1}^{\mathrm{AND}} \bmod N$ where $P_n^{\mathrm{AND}} := (1, \ldots, 1)$.
- embed encryptions $\bar{Y}_i$ of $\neg y_i$ into encryptions $\bar{Y}_i^{\mathrm{AND}}$:
  for all $i = 1, \ldots, n$ set $\bar{Y}_i^{\mathrm{AND}} := (\bar{Y}_{i,1}^{\mathrm{AND}}, \ldots, \bar{Y}_{i,\lambda}^{\mathrm{AND}})$, where $\bar{Y}_{i,j}^{\mathrm{AND}} = Y_i$ or $1 \in \mathrm{QR}_N$, the choice made by a fair coin flip.
- compute terms $t_i := [x_i \wedge \neg y_i \wedge \bigwedge_{j=i+1}^{n} [x_j = y_j]$:
  for $i = 1, \ldots, n$ let $T_i^{\mathrm{AND}} := X_i^{\mathrm{AND}} \cdot \bar{Y}_i^{\mathrm{AND}} \cdot P_i^{\mathrm{AND}} \bmod N$
- randomly permute $T_1^{\mathrm{AND}}, \ldots, T_n^{\mathrm{AND}}$ and return them to the client

a trusted party releases a secret only if a predetermined release time has expired. In this case, the predicate is given by the greater-than function and the private inputs are the release time and the current time, respectively.

Since the current time is publicly known anyway in the setting of time-release encryption, the conditional oblivious transfer scheme in [7] does not hide the private input of the sender. In some settings, though, this information may be confidential and should be kept secret. Our non-interactive crypto-computing protocol provides a solution. We stress that we do not know how to construct such a scheme with the non-interactive crypto-computing protocol from [18] for the greater-than function.

The outset is as follows. The sender possesses a bit b which is supposed to be transferred to the receiver if and only if the sender's input x is greater than the receiver's input y. The receiver generates an instance of the GM-system and sends N, z (together with a proof of correctness, if necessary) and a bit-wise GM-encryption of his private input y to the sender. Then the sender computes Formula (1) on these encryptions and his private input x. Recall that this evaluation yields n sequences of λ bits for $x, y \in \{0,1\}^n$. Exactly if $x > y$ then there is a sequence with quadratic residues exclusively; otherwise all sequences contain random entries from $\mathbb{Z}_N^*(+1)$. Now, for each $i = 1, \ldots, n$ the sender splits the bit b to be transferred into λ pieces $b_{i,1}, \ldots, b_{i,\lambda}$ with $b = b_{i,1} \oplus \ldots \oplus b_{i,\lambda}$. In addition to the T_i^{AND}'s send $(zT_{i,j}^{\mathrm{AND}})^{b_{i,j}} \cdot r_{i,j}^2 \bmod N$ for random $r_{i,j}$'s. That is, the receiver also gets the bit $b_{i,j}$ if $T_{i,j}^{\mathrm{AND}}$ is a quadratic residue, and a uniformly distributed quadratic residue if $T_{i,j}^{\mathrm{AND}} \in \mathrm{QNR}_N$ (and therefore no information about $b_{i,j}$). In other words, the receiver learns all random pieces $b_{i,1}, \ldots, b_{i,\lambda}$ for some i —and thus b— if $x > y$, and lacks at least one random piece for each i if $x \leq y$ (unless an encryption error occurs, which happens with probability at most $5n \cdot 2^{-\lambda}$).

Bottom line, if $x > y$ then the receiver gets to know the sender's bit b, whereas for $x \leq y$ the bit b is statistically hidden from the receiver. Furthermore, the receiver does not learn anything about the sender's private input x except the predicate value $[x > y]$. On the other hand, the sender gets only an encryption of the receiver's input y and does not know if b has been transferred. Hence, this is a conditional oblivious transfer protocol that keeps the private inputs secret.

4.2 Private Bidding and the Millionaires' Problem

In a private bidding protocol [3] two parties A and B compare their inputs x, y with the support of an active third party T. Although the third party T is trusted to carry out all computations corectly it should remain oblivious about the actual outcome of the comparison. Furthermore, T does not collude with either of the other parties. Note that such a bidding protocol immediately gives a fair solution to the millionaires' problem in presence of an active third party.

An efficient protocol for private bidding has been presented in [3]. The solution there requires the trusted party to compute n exponentiations, whereas A and B have to compute n encryptions. Yet, for technical reasons, the basic GM-system is *not* applicable, and the suggested homomorphic schemes need at least one exponentiation for each encryption. Also, the protocol there requires potentially stronger assumptions than the quadratic residuosity assumption.

Security of a bidding protocol in [3] is defined in terms of secure function evaluation against non-adaptive adversaries [6, 9]. Namely, an adversary corrupting at most one of the parties at the outset of the protocol does not learn anything about the other parties' inputs beyond what the compromised party should learn about the outcome of the bidding, i.e., for corrupted A or B the adversary merely learns the function value $\mathrm{GT}(x, y)$ and for compomised T it learns nothing about x, y at all. It is assumed that all parties are mutually connected via private and authenticated channels; this can be achieved using appropriate cryptographic primitives. A formal definition of secure bidding is omitted from this version.

The Honest-But-Curious Case. We first discuss the idea of our protocol in the honest-but-curious case. Also, to simplify, suppose that always $x \neq y$. The trusted party T publishes a GM-instantiation N, z such that neither A nor B knows the factorization of N. Party A sends an encryption of his input x under N, z to B who answers with an encryption of y. Then one party, say A, sends a random bit b and two random strings that are used to compute the non-interactive crypto-computing evaluation procedure on both encryptions of x, y; one time the parties inattentively compute $\mathrm{GT}(x, y)$ and the other time they evaluate $\mathrm{GT}(y, x)$. Note that both parties obtain the same strings by this since they use the same encryptions and identical random strings. Each party sends both strings in random order according to bit b to the third party. The third party decrypts the strings if and only if it receives the same strings from A and B. T computes the decision bits $[x > y], [y > x]$ for $b = 0$ or $[y > x], [x > y]$ for $b = 1$ —but does not know which of the cases has occured— and returns the bits to each party. A and B can then decide whether $x > y$ or $y < x$.

Let us briefly discuss that this protocol is secure against honest-but-curious adversaries. First note that if A and B are honest then T learns nothing (in a statistical sense) about x and y. The reason is that T merely gets random answers of the non-interactive crypto-computing protocol for $\mathrm{GT}(x, y)$ and $\mathrm{GT}(y, x)$ in random order (and exactly one of the predicates is satisfied by assumption about inequality of x and y).

Assume that an adversary sees all internal data and incoming and outgoing messages of A, but does not have control over A and, in particular, cannot bias the random bits used for the crypto-computation. This, however, means that unless an error occurs in the crypto-computation the adversary gets only the information about $[x > y]$ from the trusted party, and a secure encryption of y from B. Hence, with very high probability the adversary does not learn anything in a computational sense from listening to the execution. Similarly, this follows for a dishonest B.

Fig. 3. Private Bidding with an Oblivious Third Party

Trusted party T publishes a GM-instantiation N, z and random quadratic non-residues w_A and w_B.

- A and B jointly generate two random strings each of $n\lambda$ bits and a bit b using a coin-flipping protocol with N, w_A.
- A sends bit-wise GM-encryptions $\mathrm{Enc}(x)$ under N, z to B and gives a zero-knowledge proof of knowledge based on N, z, w_A; A also sends a sufficient number of random elements from $\mathbb{Z}_N^*$.
- Vice versa, B sends an encryption $\mathrm{Enc}(y)$ under N, z to A and proves in zero-knowledge with N, z, w_B that he knows the plaintext.
- Each party computes the server's evaluation procedure for $\mathrm{GT}(x, y)$ and $\mathrm{GT}(y, x)$ on the encryptions with the predetermined elements from $\mathbb{Z}_N^*$ and submits the result to the trusted party (in random order according to bit b).
- If and only if both incoming values are equal then the trusted party decodes both sequences and returns the bits.
- Both parties output the result $\mathrm{GT}(x, y)$.

The Malicious Case. In order to make the protocol above secure against actively cheating adversaries we have to ensure that the parties do not choose the encryptions in an adaptive manner. Formally, we need to extract the encrypted values from the dishonest party A or B and for this purpose add an interactive zero-knowledge proof of knowledge to the encryptions; this proof of knowledge can be carried out in three rounds with the data N, z, w_A, w_B published by T. Details are postponed to Appendix A.

Also, we have to ensure that the non-interactive crypto-computation is really based on truly random bits: biased bits may increase the probability that the result of an AND of encryptions is incorrect and thus the outcome might reveal some information about x or y, respectively. This is solved by using a coin-flipping protocol in which one party commits to a random string a and the other publishes a random string b and the outcome is set to $a \oplus b$. Again, the reader is refered to Appendix A for details. It is important to notice that we only need random bits for the embedding of basic GM-encryptions into AND-encryptions. That is, only the choice whether we encode the i-th component as $\mathrm{Enc}(d)$ or as $\mathrm{Enc}(0)$ when embedding a basic GM-encryption $\mathrm{Enc}_0(d)$ must be made at random. Hence, we only need $n\lambda$ random bits for each evaluation; the necessary elements from $\mathbb{Z}_N^*$ for embedding can be announced by one of the parties.

We present an informal argument why this scheme is secure; a formal proof is deferred from this version. Say that the adversary corrupts party T. Then the same argument as in the honest-but-curious case applies: T only sees two honestly generated outcomes of crypto-computations for $\mathrm{GT}(x, y)$ and $\mathrm{GT}(y, x)$ in random order.

Consider the case that the adversary corrupts A. We have to present a simulator that is allowed to query an oracle $\mathrm{GT}(\cdot, y)$ *once*, and outputs a protocol execution which is indistinguishable from a true execution with honest T and B (with secret input y). Roughly, the simulator extracts the input x^* from the adversary's proof of knowledge for the encryption and simulates T's and B's behavior by the zero-knowledge property.[4] The simulator then queries the oracle about x^* to obtain $\mathrm{GT}(x^*, y)$. Given this bit the simulator finally outputs T's answer; this is possible since it knows the order of the transmitted crypto-computations and both parties must send the same crypto-computation for $\mathrm{GT}(x^*, y)$ and $\mathrm{GT}(y, x^*)$. Additionally, since the computation involves truly random bits because of the coin tossing, the result of the crypto-computation is correct with very high probability. In this case, T's decoding would be identical to the simulator's output for $\mathrm{GT}(x^*, y)$.

The case that the adversary corrupts B is analogous. Hence, we obtain a secure constant-round private-bidding protocol with an active oblivious third party; the protocol requires at most $19n\lambda + 2\lambda$ modular multiplications for each party, where n is the length of the bids and λ determines the error.

Above we presumed that $x \neq y$. It is not hard to see that our non-interactive crypto-computing protocol in Section 3 can be modified to a scheme which computes $\mathrm{EQ}(x, y)$ and where the server's answer is identical distributed to the one for $\mathrm{GT}(x, y)$ (for the same instance N, z). Therefore, if one alters the

[4] At first glance, it seems that the simulator could choose N on behalf of T such that it knows the factorizaton p, q of N and such that it can extract x^* directly by decoding. Yet, the simulator has to present a fake encryption of y pretending to be B, and we were not able to prove this to be indistinguishable from a correct encryption of y if the simulator knows p, q. Therefore, we take the detour using a proof of knowledge.

bidding protocol by letting A and B passing three crypto-computations for $\mathrm{GT}(x,y), \mathrm{GT}(y,x), \mathrm{EQ}(x,y)$ in random order to T, then one obtains a secure bidding protocol where A and B know which input is bigger (if any), or if the inputs are equal.

Acknowledgements

We thank the anonymous reviewers of RSA 2001 for comprehensive comments.

References

1. D.Beaver: Minimal-Latency Secure Function Evaluation, *Eurocrypt 2000, Lecture Notes in Computer Science, Vol. 1807, Springer-Verlag, pp. 335–350*, 2000.
2. F.Boudot, B.Schoenmakers, J.Traoré: A Fair and Efficient Solution to the Socialist Millionaires' Problem, *to appear in Discrete Applied Mathematics, Special Issue on Coding and Cryptography, Elsevier*, 2000.
3. C.Cachin: Efficient Private Bidding and Auctions with an Oblivious Third Party, *6th ACM Conference on Computer and Communications Security, pp. 120–127*, 1999.
4. C.Cachin, J.Camenish: Optimistic Fair Secure Computations, *Crypto 2000, Lecture Notes in Computer Science, Vol. 1880, Springer-Verlag, pp. 93–111*, 2000.
5. C.Cachin, J.Camenish, J.Kilian, J.Müller: One-Round Secure Computation and Secure Autonomous Mobile Agents, *ICALP 2000, Lecture Notes in Computer Science, Springer-Verlag*, 2000.
6. R.Canetti: Security and Composition of Multiparty Cryptographic Protocols, *Journal of Cryptology, Vol. 13, No. 1, Springer-Verlag, pp. 143–202*, 2000.
7. G.Di Crescenzo, R.Ostrovsky, S.Rajagopalan: Conditional Oblivious Transfer and Time-Release Encryption, *Eurocrypt '99, Lecture Notes in Computer Science, Vol. 1592, Springer-Verlag, pp. 74–89*, 1999.
8. U.Feige, A.Fiat, A.Shamir: Zero-Knowledge Proofs of Identity, *Journal of Cryptology, Vol. 1, No. 2, pp. 77–94, Springer-Verlag*, 1988.
9. O.Goldreich: Secure Multi-Party Computation, *(working draft, version 1.2), available at* `http://www.wisdom.weizmann.ac.il/home/oded/public_html/pp.html`, March 2000.
10. O.Goldreich, S.Micali, A.Wigderson: How to Play any Mental Game —or— a Completeness Theorem for Protocols with Honest Majorities, *Proceedings of the 19th Annual ACM Symposium on the Theory of Computing, pp. 218–229*, 1987.
11. O.Goldreich, S.Micali, A.Wigderson: Proofs That Yield Nothing About Their Validity —or— All Languages in NP Have Zero-Knowledge Proof Systems, *Journal of the ACM, Vol.8, No. 1, pp. 691–729*, 1991.
12. S.Goldwasser, S.Micali: Probabilistic Encryption, *Journal of Computer and System Sciences, Vol. 28(2), pp. 270–299*, 1984.
13. S.Goldwassser, S.Micali, C.Rackoff: The Knowledge Complexity of Interactive Proof Systems, *SIAM Journal on Computation, Vol. 18, pp. 186–208*, 1989.
14. G.Hardy, E.Wright: An Introduction to the Theory of Numbers, *Oxford University Press*, 1979.
15. M.Jakobsson, M.Yung: Proving Without Knowing: On Oblivious, Agnostic and Blindfolded Provers, *Crypto '96, Lecture Notes in Computer Science, Vol. 1109, Springer-Verlag, pp. 186–200*, 1996.

16. M.Naor, B.Pinkas, R.Sumner: Privacy Preserving Auctions and Mechanism Design, *1st ACM Conference on Electronic Commerce, available at* `http://www.wisdom.weizmann.ac.il/~bennyp/`, 1999.
17. M.Rabin: How to Exchange Secrets by Oblivious Transfer, *Technical Report TR-81, Harvard*, 1981.
18. T.Sander, A.Young, M.Yung: Non-Interactive Crypto-Computing for NC^1, *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science (FOCS)*, 1999.
19. A.Yao: Protocols for Secure Computation, *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science (FOCS), pp. 160–164*, 1982.
20. A.Yao: How to Generate and Exchange Secrets, *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 162–167*, 1986.

A Fair Coin-Tossing and Zero-Knowledge Proof of Knowledge for GM-Encryptions

In this section we review the missing sub protocols of Section 4.2: a three-round zero-knowledge [13] proof of knowledge for GM-encryptions [8], and a coin-flipping protocol to generate unbiased random bits. We start with the coin-flipping protocol as we will need it for the proof of knowledge, too.

A.1 Fair Coin-Tossing

Assume that a trusted party publishes a modulus N and a quadratic non-residue $w_A \in \mathrm{QNR}_N$. To generate a single unbiased random bit,

– party A commits to a bit a by sending a random encryption $\mathrm{Enc}(a, s)$ under N, w_A.
– B sends a random bit b.
– A decommits to a by sending a, s to B (who verifies that $\mathrm{Enc}(a, s)$ equals the ciphertext from the first step).
– the random bit c is set to $c = a \oplus b$.

These basic steps can be repeated $2n\lambda + 1$ times in parallel to generate $2n\lambda + 1$ random bits as required in our application.

We show that the protocol can be used to generate an unbiased bit even if one party is dishonest; furthermore, we show that a simulator playing A can bias the outcome to any predetermined value c.

If A is corrupt then c is uniformly distributed because the encryption of a binds perfectly. If B is controlled by the adversary this is accomplished by letting T announce an invalid but correct looking w_A which is a quadratic residue instead of a non-residue; moreover, assume that we know a root v_A of $w_A = v_A^2 \bmod N$. We bias the coin flipping by first choosing a random $c \in \{0, 1\}$ before the protocol starts and by sending an encryption $\mathrm{Enc}(1, s)$ of $a = 1$ under N, w_A. Then, after having received b from the adversary, we decommit to $a' = c \oplus b$ by sending $(0, sv_A)$ for $a' = 0$ and $(1, s)$ for $a' = 1$. It is readily verified that this is a corret decommitment for a'. Conclusively, the coin flip is biased to the previously selected, but uniformly distributed c. On the other hand, under the quadratic residuosity assumption, B cannot distinguish that w_A is quadratic residue.

A.2 Zero-Knowledge Proof of Knowledge

Next, we introduce the proof of knowledge, but we start with a special case of a short challenge. Assume again that the trusted party publishes a modulus N (which can be same as in the coin flipping protocol) and a quadratic non-residue $z \in \mathrm{QNR}_N$ (which is also used for encryption). Party A has published a bit-wise encryption $X_i = \mathrm{Enc}(x_i, r_i)$ of $x \in \{0,1\}^n$ under N, z.

- party A commits to an n-bit string u by sending a bit-wise encryption $U_i = \mathrm{Enc}(u_i, s_i)$ of u under N, z.
- party B sends a random bit c.
- if $c = 0$ then A sends u and the randomness $s_1, \ldots, s_n$ used to encrypt u. If $c = 1$ then A sends $v = u \oplus x \in \{0,1\}^n$ and $t_i = r_i s_i \bmod N$.
- B verifies the correctness by re-encrypting the values with the reveled randomness and comparing it to the given values. Specifically, for $c = 0$ party B checks that $U_i = \mathrm{Enc}(u_i, s_i)$, and verifies that $U_i X_i = \mathrm{Enc}(v_i, t_i) \bmod N$ for $c = 1$.

Under the quadratic residuosity assumption this protocol is computational zero-knowledge, i.e., there exist an efficient simulator that, for any malicious B, imitates A behavior in an indistinguishable manner without actually knowing x. This zero-knowledge simulator basically tries to guess the challenge at the outset and sends appropriate phony values in the first step.

This basic protocol for bit-challenges allows to cheat with probability 1/2 by simply guessing the challenge. But the steps can be repeated independently in parallel for λ in order to decrease the error to $2^{-\lambda}$. However, while this protocol is provably zero-knowledge for logarithmically bounded λ, it is not known to be zero-knowledge for large λ.

Fortunately, the problem is solvable by tossing coins. That is, we generate λ bit-challenges with a coin flipping protocol as described above. This can be interleaved with the proof of knowledge to obtain a three-round protocol. Since the outcome of the coin flips can be chosen a priori if we have a quadratic residue w_A, the protocol becomes zero-knowledge; the zero-knowledge simulator does not even have to guess the challenge bits because it can choose them for himself before the protocol starts.

As for our bidding protocol, we announce independent w_A and w_B for each party: either party that is under control of the adversary gets a quadratic non-residue (to force this party to provide a correct proof of knowledge and to generate truly random bits), whereas the simulator playing the honest party is given a quadratic residue in order to "cheat". For an adversary this is not detectable under the quadratic residuosity assumption.