Practical Two-Party Computation based on the Conditional Gate

Berry Schoenmakers*[1] and Pim Tuyls[2]

[1] Dept. of Mathematics and Computing Science, TU Eindhoven,
P.O. Box 513, 5600 MB Eindhoven, The Netherlands.
`berry@win.tue.nl`
[2] Philips Research Labs
Prof. Holstlaan 4, 5656 AA Eindhoven, The Netherlands
`pim.tuyls@philips.com`

Abstract. We present new results in the framework of secure multi-party computation based on homomorphic threshold cryptosystems. We introduce the *conditional gate* as a special type of multiplication gate that can be realized in a surprisingly simple and efficient way using just standard homomorphic threshold ElGamal encryption. As addition gates are essentially for free, the conditional gate not only allows for building a circuit for any function, but actually yields efficient circuits for a wide range of tasks.

1 Introduction

Homomorphic threshold cryptosystems provide a basis for secure multiparty computation in the cryptographic model [FH96,JJ00,CDN01,DN03]. For a given n-ary function f, one composes a circuit C of elementary gates that given encryptions of $x_1, \ldots, x_n$ on its input wires, produces an encryption of $f(x_1, \ldots, x_n)$ on its output wire. The elementary gates operate in the same fashion. The wires of the entire circuit C are all encrypted under the same public key; the corresponding private key is shared among a group of parties. It is customary to distinguish addition gates and multiplication gates. Addition gates can be evaluated without having to decrypt any value, taking full advantage of the homomorphic property of the cryptosystem. Multiplication gates, however, require at least one threshold decryption to succeed even for an honest-but-curious (passive) adversary. To deal with a malicious (active) adversary, multiplication gates additionally require the use of zero-knowledge proofs.

While the result of [FH96] covers the case of a passive adversary only, an interesting feature is that it covers both the two-party case ($n = 2$) and the multiparty case ($n > 2$) in a *uniform* way. The later papers [JJ00,CDN01,DN03] do cover an active adversary, but only consider the multiparty case. In the present paper, we are particularly interested in extending the use of homomorphic threshold

* Work done partly while visiting Philips Research Labs

cryptosystems to the two-party case. We observe that solutions based on homomorphic threshold cryptosystems can be used just as well in the two-party case. To cover fairness, however, an additional protocol is needed that allows two parties to jointly decrypt the outputs in a gradual fashion. We present such a protocol by showing how to adapt the decryption step of a homomorphic threshold cryptosystem.

A major advantage of secure multiparty computation based on homomorphic threshold cryptosystems is the fact that it results in particularly efficient solutions, even for active adversaries. The communication complexity, which is the dominating complexity measure, is $O(nk|C|)$ bits for [JJ00,CDN01,DN03], where n is the number of parties, k is a security parameter, and $|C|$ is the number of gates of circuit C. A more detailed look at the performance of these solutions reveals, however, that there is considerable room for improvement in several respects.

It is assumed in [JJ00,CDN01,DN03] that the shared key for the homomorphic threshold cryptosystem used in the multiparty protocol is already given. As a consequence, the communication complexity of $O(nk|C|)$ bits does not include the communication needed for the distributed key generation (DKG) protocol of the underlying threshold cryptosystem. However, the performance of the DKG protocol is an issue since we envision a system supporting *ad hoc contacts* among a large group of peer users, where any pair of users may decide to engage in a secure two-party computation for a dynamically agreed upon function. For example, “profile matching” is an application in which two users jointly test whether some function of their (personal) profiles exceeds a given threshold, without divulging any further information on their profiles. In this scenario, it is unreasonable to assume that each pair of users shares a specific key pair for the underlying threshold cryptosystem. Instead, each time two users want to perform a two-party computation, they would need to run the DKG protocol first.

In this respect, an advantage of the Mix and Match approach of [JJ00] is its applicability to any discrete log setting, whereas [CDN01,DN03] depend critically on an RSA-like setting (e.g., using Paillier’s cryptosystem). The advantage is that DKG protocols for discrete log based cryptosystems are efficient and relatively simple (see [Ped91,GJKR99]). In particular, DKG can be achieved practically for free in the two-party case. This contrasts sharply with the known protocols for distributed generation of a shared RSA modulus. Briefly, for the two-party case (without a helper party), Gilboa [Gil99] reports a communication complexity of about 42MB (or 29MB for a slightly optimized version) for generating a shared 1024-bit RSA modulus, while covering passive adversaries only. And, for the multiparty case, the results of [ACS02] show what is currently achievable, also covering passive adversaries only.

Interestingly, it is actually possible to combine the benefits of a discrete log setting and an RSA-like setting, as demonstrated recently in [DJ03]. To this end, one uses an amalgam of the ElGamal cryptosystem and the Paillier cryptosystem (such a combination has also been presented in the full version of [CS02]). A system supporting ad hoc contacts may then be set up by jointly generating a single RSA modulus (between as many parties as deemed necessary, e.g., using a robust version of [ACS02]). A discrete log based DKG protocol will suffice to generate a shared key between any two users. We do note however that the security of the resulting system relies on both a discrete log related assumption and a factoring related assumption, which is undesirable from a theoretical point of view.

In this paper, we will focus on a solution for which the security depends on the standard decisional Diffie-Hellman (DDH) assumption. As a consequence our protocols can be implemented using elliptic curves, for which the security is assumed to be exponential as a function of the security parameter rather than sub-exponential (as for RSA, for example). The Mix and Match approach of [JJ00] is also secure under DDH, but we note that the resulting protocol for evaluating multiplication gates is—despite its conceptual simplicity—quite inefficient. We will show how to evaluate multiplication gates in a much simpler way, such that the computational effort decreases by at least one order of magnitude (that is, a ten-fold speed-up is achieved, see Section 3.3). On the other hand, a disadvantage of our approach is that, in general, the round complexity is $O(nd)$ for n parties and circuit depth $d = d(C)$, versus $O(n+d)$ for Mix and Match. For two-party computation, however, the round complexity is $O(d)$ in both cases, and more generally for small n the gain in computational efficiency outweighs the increased round complexity.

The basis of our approach is formed by the *conditional* gate, a special multiplication gate which we show to be efficiently implementable under DDH. Basically, a conditional gate allows us to efficiently multiply two encrypted values x and y, as long as x is restricted to a two-valued domain, e.g., $x \in \{0, 1\}$. We emphasize that the value of y is not restricted, e.g., we may have $y \in \mathbb{Z}_q$, where q is a large prime. This property can be exploited when designing circuits for specific functions. For example, from the formula $(y'_0, y'_1) = (y_0 - x(y_0 - y_1), y_1 + x(y_0 - y_1))$, with $x \in \{0, 1\}$, one sees that a conditional swap gate, swapping any two values in $\mathbb{Z}_q$ depending on the value of x, can be obtained using a single conditional gate. We will indicate that, using ElGamal, one cannot expect to achieve a multiplication gate for which both inputs are unrestricted. Note, however, that the result of [CDN01] shows that multiplication of two unrestricted values can be achieved efficiently under a factoring related assumption.

Overview. Throughout the paper we will describe the results in a general setting of n-party computation, $n \geq 2$, although we are mainly interested in the two-party case. In Section 2, we review the basics for homomorphic threshold ElGamal. In Section 3, we introduce the conditional gate as our elementary multiplication gate, and we show how it can be used to achieve xor-homomorphic ElGamal encryption efficiently. In Section 4, we then consider the secure evaluation of arbitrary circuits, following [CDN01], which we extend with a new, non-interactive protocol for achieving private outputs. Furthermore, we propose an efficient protocol for achieving fairness in the two-party case. In Section 5, we show that particularly efficient circuits can be built for basic operations such as integer comparison, paying special attention to Yao's well-known millionaires problem in Section 5.2, for which we obtain a solution requiring $12m$ exponentiations for m-bit integers. Finally, in Section 6, we conclude with future work and give an example of a more advanced application which we call 'profile matching'.

2 Preliminaries on Homomorphic Threshold ElGamal

Discrete Log Setting. Let $G = \langle g \rangle$ denote a finite cyclic (multiplicative) group of prime order q for which the Decision Diffie-Hellman (DDH) problem is assumed to be infeasible: given $g^\alpha, g^\beta, g^\gamma \in_R G$, it is infeasible to decide whether $\alpha\beta \equiv \gamma \pmod{q}$. This implies that the Diffie-Hellman (DH) problem, which is to compute $g^{\alpha\beta}$ given $g^\alpha, g^\beta \in_R G$, is infeasible as well. In turn, this implies that the Discrete Log (DL) problem, which is to compute $\log_g h = \alpha$ given $g^\alpha \in_R G$, is infeasible.

Homomorphic ElGamal Encryption. For public key $h \in G$, a message $m \in \mathbb{Z}_q$ is encrypted as a pair $(a, b) = (g^r, g^m h^r)$, with $r \in_R \mathbb{Z}_q$. Encryption is *additively* homomorphic: given encryptions $(a, b), (a', b')$ of messages m, m', respectively, an encryption of $m + m'$ is obtained as $(a, b) \star (a', b') = (aa', bb') = (g^{r+r'}, g^{m+m'} h^{r+r'})$.

Given the private key $\alpha = \log_g h$, decryption of $(a, b) = (g^r, g^m h^r)$ is performed by first calculating $b/a^\alpha = g^m$, and then solving for $m \in \mathbb{Z}_q$. In general, this is exactly the DL problem, which we assume to be infeasible. The way out is to require that message m is constrained to a sufficiently small set $M \subseteq \mathbb{Z}_q$.[1] In this paper, the cardinality of M will be very small, often $|M| = 2$.

Homomorphic ElGamal encryption is semantically secure assuming the infeasibility of the DDH problem. Throughout the paper, we use $[\![m]\!]$ to denote the set of all ElGamal encryptions of m under some understood public key h, and, frequently, we also use $[\![m]\!]$ to denote one of its elements. More formally, using that $[\![0]\!]$ is a subgroup of $G \times G$, $[\![m]\!]$ is the coset of $[\![0]\!]$ in $G \times G$ containing encryption $(1, g^m)$. Hence, encryptions (a, b) and (a', b') belong to the same coset iff $\log_g(a/a') = \log_h(b/b')$. Lifting the operations on the direct product group $G \times G$ to the cosets, we thus have, for $x, y \in \mathbb{Z}_q$, that $[\![x]\!] \star [\![y]\!] = [\![x + y]\!]$, and $[\![x]\!]^y = [\![xy]\!]$, where $(a, b)^c = (a^c, b^c)$ for $c \in \mathbb{Z}_q$. Hence, $[\![x]\!] \star [\![y]\!]^{-1} = [\![x - y]\!]$. Addition and subtraction over $\mathbb{Z}_q$ and multiplication by a publicly known value in $\mathbb{Z}_q$ can thus be performed easily on encrypted values. These operations are deterministic. Another useful consequence is that any encryption in $[\![x]\!]$ can be transformed into a statistically independent encryption in $[\![x]\!]$ by multiplying it with a uniformly selected encryption in $[\![0]\!]$; this is often referred to as "random re-encryption."

Pedersen Commitment. Given $h' \in G$, a Pedersen commitment to $m \in \mathbb{Z}_q$ is a value $b = g^m h'^r$, with $r \in_R \mathbb{Z}_q$. The commitment is opened by revealing m and r. Pedersen's scheme is unconditionally hiding and computationally binding, under

[1] For intervals M, the Pollard-λ ("kangaroo") method runs in $O(\sqrt{|M|})$ time using $O(1)$ storage.

the assumption that $\log_g h'$ cannot be determined. The commitment scheme is also additively homomorphic, and we will sometimes use $\langle\!\langle m \rangle\!\rangle$ to denote a commitment to message m, where the randomization is suppressed.

Σ-Protocols. We briefly mention a few facts about Σ-protocols. A Σ-protocol for a relation $R = \{(v, w)\}$ is a three-move protocol between a prover and a verifier, where the prover does the first move. Both parties get a value v as common input, and the prover gets a "witness" w as private input, $(v, w) \in R$. A Σ-protocol is required to be a proof of knowledge for relation R satisfying special soundness and special honest-verifier zero-knowledge. See [CDS94] for details.

We need some well-known instances of Σ-protocols. The simplest case is Schnorr's protocol for proving knowledge of a discrete log α, on common input $a = g^\alpha$, and Okamoto's variant for proving knowledge of α, β, on common input $a = g^\alpha h^\beta$. Another basic case is Chaum-Pedersen's protocol for proving knowledge of α, on common input $(a, b) = (g^\alpha, h^\alpha)$, which is a way to prove that $(a, b) \in [\![0]\!]$ without revealing any information on α. Applying OR-composition [CDS94], these basic protocols can be combined into, for instance, a Σ-protocol for proving that $(a, b) \in [\![0]\!] \cup [\![1]\!]$, where the common input is an ElGamal encryption (a, b). The latter protocol thus proves that the message encrypted (which is an element of $\mathbb{Z}_q$) actually is a "bit", without divulging any further information on the message.

For simplicity, we will use the non-interactive versions of these Σ-protocols, which are obtained via the Fiat-Shamir heuristic, that is, by computing the challenge as a hash of the first message (and possibly other inputs). The resulting proofs are known to be secure in the random oracle model; in particular, we will use that these proofs can be simulated.

Threshold ElGamal Decryption. We use a $(t+1, n)$-threshold ElGamal cryptosystem, $0 \leq t < n$, in which encryptions are computed using a common public key h (as above) while decryptions are done using a joint protocol between n parties $P_1, \ldots, P_n$. Each party P_i holds a share $\alpha_i \in \mathbb{Z}_q$ of the private key $\alpha = \log_g h$, where the corresponding value $h_i = g^{\alpha_i}$ is public. As long as more than t parties take part, decryption will succeed, whereas t or less parties are not able to decrypt successfully.

The parties initially obtain their shares α_i by running a secure distributed key generation protocol; see [Ped91,GJKR99] for details. We note that these protocols are practical (the communication complexity is $O(n^2 k)$ bits for security parameter k, where the hidden constant is small). For the two-party case ($t = 1$, $n = 2$), we briefly describe a (non-robust) distributed key generation protocol in the spirit of [GJKR99]. The protocol consists of two steps. In the first step, party P_i, $i = 1, 2$, broadcasts a Pedersen commitment $b_i = g^{\alpha_i} h'^{r_i}$, with $\alpha_i, r_i \in_R \mathbb{Z}_q$ along with a proof of knowledge for α_i, r_i. In the second step, party P_i, $i = 1, 2$, broadcasts r_i along with a proof of knowledge of $\log_g h_i$, where $h_i = b_i / h'^{r_i}$. The joint public key is $h = h_1 h_2$, with private key $\alpha = \alpha_1 + \alpha_2$. Clearly, this protocol is very practical. In many cases, it may even be replaced by the trivial one-round protocol in which both parties broadcast $h_i = g^{\alpha_i}$ and a proof of knowledge of α_i. Although the trivial protocol allows one of the parties to influence the distribution of the public key h slightly, this need not be a problem for the application in which the key is used; see [GJKR03] for more details.

For decryption of (a, b), party P_i, $i = 1, \ldots, n$, produces a decryption share $d_i = a^{\alpha_i}$ along with a proof that $\log_a d_i = \log_g h_i$. Assuming w.l.o.g. that parties $P_1, \ldots, P_{t+1}$ produce correct decryption shares, the message can be recovered from $g^m = b/a^\alpha$, where a^α is obtained from $d_1, \ldots, d_{t+1}$ by Lagrange interpolation. Assuming homomorphic ElGamal, $m \in M$ will hold for some small set M; if such m cannot be found decryption fails. Also, if fewer than $t + 1$ parties provide a correct decryption share, decryption fails.

For later use in the proof of Theorem 1, we note that the threshold decryption protocol can be simulated for any input $(a, b) \in [\![m]\!]$, provided message $m \in \mathbb{Z}_q$ is given as well. Assume w.l.o.g. that parties $P_1, \ldots, P_t$ are corrupted, hence collectively form the adversary. The simulator first extracts the shares $\alpha_1, \ldots, \alpha_t$ of the adversary, by rewinding the proofs of knowledge based on these shares. (The parties prove knowledge of their shares during the distributed key generation protocol.) The simulator then computes $a^\alpha = b/g^m$ from b and m. The simulator then computes the correct decryption shares for the corrupted parties as $a^{\alpha_1}, \ldots, a^{\alpha_t}$, which enables the computation of the decryption shares for the honest parties by Lagrange interpolation on $a^\alpha, a^{\alpha_1}, \ldots, a^{\alpha_t}$. The corresponding proofs of correct decryption are simulated for the honest parties. For the corrupted parties, the decryption shares and the proofs of correct decryption are obtained from the adversary, running it as a black box; possibly some of these shares are wrong and/or some of the proofs fail, but these values are included in the output of the simulator anyway. The simulation is then completed by recovering the message as in the real protocol, possible ending with a decryption failure. As a result, the simulated transcript is consistent with the view of the adversary and statistically indistinguishable of real transcripts.

3 Special Multiplication Protocols

The results of the previous section imply that a function f can be evaluated securely in a multiparty setting if f can be represented as a circuit over $\mathbb{Z}_q$ consisting only of addition gates and simple multiplication gates. Here, an addition gate takes encryptions $[\![x]\!]$ and $[\![y]\!]$ as input and produces $[\![x]\!] \star [\![y]\!] = [\![x + y]\!]$ as output, and a simple multiplication gate takes $[\![x]\!]$ as input and produces $[\![x]\!]^c = [\![cx]\!]$ as output, for a publicly known value $c \in \mathbb{Z}_q$. To be able to handle any function f, however, we need more general multiplication gates for which both inputs are encrypted.

In this section, we consider two special multiplication gates. If no restrictions are put on x or y, a multiplication gate, taking $[\![x]\!]$ and $[\![y]\!]$ as input and producing $[\![xy]\!]$ as output efficiently, cannot exist assuming that the DH problem is infeasible.[2] Therefore, we consider two special multiplication gates, putting

[2] Given g^x, g^y we form encryptions $[\![x]\!]$, $[\![y]\!]$ and feed these into the multiplication gate. The gate would return an encryption $[\![xy]\!]$, which would give g^{xy} upon decryption.

some restrictions on the multiplier x. The first gate requires that the multiplier x is *private*, which means that it is known by a single party. The second gate, referred to as the *conditional gate*, requires that the multiplier x is from a *dichotomous* (two-valued) domain. As a direct application of the conditional gate, we also consider xor-homomorphic encryption based on ElGamal encryption.

3.1 Multiplication with a Private Multiplier

We present a multiplication protocol where the multiplier x is a *private* input rather than a shared input. That is, the value of x is known by a single party P. No restriction is put on the multiplicand y. Multiplication with a private multiplier occurs as a subprotocol in the protocol for the conditional gate and in other protocols further on in the paper.

Given encryptions $[\![x]\!] = (a, b) = (g^r, g^x h^r)$ and $[\![y]\!] = (c, d)$, where party P knows r, x, party P computes on its own a randomized encryption $[\![xy]\!] = (e, f) = (g^s, h^s) \star [\![y]\!]^x$, with $s \in_R \mathbb{Z}_q$, using the homomorphic properties. Party P then broadcasts $[\![xy]\!]$ along with a proof showing that this is the correct output, which means that it proves knowledge of witnesses $r, s, x \in \mathbb{Z}_q$ satisfying $a = g^r$, $b = g^x h^r$, $e = g^s c^x$, $f = h^s d^x$.

For later use, we need to be able to simulate the above protocol. The simulator gets as input $[\![x]\!]$ and $[\![y]\!]$, and a correct output encryption $[\![xy]\!]$, but it does not know x. As a result, the simulator only needs to add a simulated proof of knowledge. The simulated transcript is statistically indistinguishable from a real transcript.

Below, we will also use a variation of the above protocol, where the private multiplier x is multiplied with several multiplicands y_i at the same time. Furthermore, we note that often a slight optimization is possible by using a Pedersen commitment $\langle\!\langle x \rangle\!\rangle = g^x h'^r$ instead of an ElGamal encryption $[\![x]\!] = (g^r, g^x h^r)$ for the multiplier.

3.2 Conditional Gate

Next, we consider a multiplication gate for which the multiplier x is from a dichotomous (two-valued) domain, whereas the multiplicand y is unrestricted. We call it the *conditional gate*, and show how to implement it by an efficient protocol, using just homomorphic threshold ElGamal. We will formulate the conditional gate for the dichotomous domain $\{-1, 1\}$.[3]

Let $[\![x]\!], [\![y]\!]$ denote encryptions, with $x \in \{-1, 1\} \subseteq \mathbb{Z}_q$ and $y \in \mathbb{Z}_q$. The following protocol enables parties $P_1, \ldots, P_n$, $n \geq 2$, to compute an encryption $[\![xy]\!]$ securely. For simplicity, we assume that these parties also share the private

[3] Domain $\{0, 1\}$ or any other domain $\{a, b\}$, $a \neq b$, can be used instead, as these domains can be transformed into each other by linear transformations: $x \mapsto a' + (b' - a')(x - a)/(b - a)$ maps $\{a, b\}$ to $\{a', b'\}$. These transformations can be applied directly to homomorphic encryptions, transforming $[\![x]\!]$ with $x \in \{a, b\}$ into $[\![x']\!]$ with $x' \in \{a', b'\}$.

key of the $(t+1,n)$-threshold scheme $[\![\cdot]\!]$, where $t<n$. The protocol consists of two phases.

1. Let $x_0 = x$ and $y_0 = y$. For $i = 1, \dots, n$, party P_i in turn takes $[\![x_{i-1}]\!]$ and $[\![y_{i-1}]\!]$ as input, and broadcasts a commitment $\langle\!\langle s_i \rangle\!\rangle$, with $s_i \in_R \{-1, 1\}$. Then P_i applies the private-multiplier multiplication protocol to multiplier $\langle\!\langle s_i \rangle\!\rangle$ and multiplicands $[\![x_{i-1}]\!]$ and $[\![y_{i-1}]\!]$, yielding random encryptions $[\![x_i]\!]$ and $[\![y_i]\!]$, where $x_i = s_i x_{i-1}$ and $y_i = s_i y_{i-1}$. If P_i fails to complete this step successfully it is discarded immediately.
2. The parties jointly decrypt $[\![x_n]\!]$ to obtain x_n. If decryption fails because the number of correct shares is insufficient, the entire protocol is aborted. If decryption fails because $x_n \notin \{-1,1\}$, each party P_i is required to broadcast a proof that $s_i \in \{-1,1\}$. Parties failing to do so are discarded, and the protocol is restarted (starting again at phase 1). Given x_n and $[\![y_n]\!]$, an encryption $[\![x_n y_n]\!]$ is computed publicly.

The output of the protocol is $[\![x_n y_n]\!]$. Clearly, if all parties are honest, $x_n y_n = (\prod_{i=1}^n s_i)^2 xy = xy$.

Any party may disrupt the protocol for at most one run of phase 1 by picking a value s_i outside the range $\{-1,1\}$. Note that we do not need to require that each s_i is in $\{-1,1\}$ in phase 1. For instance, parties P_1 and P_2 may cheat by setting $s_1 = 2$ and $s_2 = 1/2$. Since $s_1 s_2 = 1$, this type of "cheating" will go unnoticed in phase 2 if all other parties are honest. However, the security of the protocol is not affected by such "cheating." For $t < n/2$, the protocol is robust, allowing up to t failing parties in total (as the threshold decryption step tolerates up to t failing parties). For $n/2 \le t < n$, the protocol is not robust, but we will see from Theorem 1 below that the adversary does not get an advantage in this case.

The protocol requires a single threshold decryption only. Since $x_n \in \{-1,1\}$ is required to hold, decryption is feasible for the homomorphic ElGamal encryption scheme. As the value of x_n is statistically independent of x, the value of x_n does not reveal any information on x. This is stated in the following theorem, which holds for up to $t<n$ corrupting parties.

Theorem 1. *On input $[\![x]\!], [\![y]\!]$ with $x \in \{-1,1\} \subseteq \mathbb{Z}_q$ and $y \in \mathbb{Z}_q$, the above protocol produces $[\![xy]\!]$, without leaking any additional information on x and y.*

Proof. The soundness of the proofs in phase 1 of the protocol ensures that $x_n = x\prod_{i=1}^n s_i$ and $y_n = y\prod_{i=1}^n s_i$ Since it is checked in phase 2 that $x_n \in \{-1,1\}$, it follows from $x \in \{-1,1\}$ that $\prod_{i=1}^n s_i \in \{-1,1\}$ as well. Therefore, $x_n y_n = xy$.

To argue that no additional information on x and y is leaked we present the following simulation of the protocol. The simulation takes as input encryptions $[\![x]\!], [\![y]\!]$, and $[\![xy]\!]$. Given this information, the simulator is able to generate a complete transcript for the protocol, for which the distribution is exactly the same as in real executions of the protocol. If $[\![xy]\!]$ is not available, it may be replaced by a random encryption in $[\![0]\!]$, as done in [CDN01, Theorem 1]. Since the simulator below does not use the shares of the honest parties to simulate decryptions, the simulated transcripts will be indistinguishable (under DDH) from real transcripts for any adversary controlling up to t parties.

Assume that parties $P_1, \ldots, P_t$ are corrupted, hence collectively form the adversary (the simulator is easily adapted for other sets of corrupted parties). The simulator lets the adversary run phase 1 of the protocol for parties $P_1, \ldots, P_t$, each time rewinding the proofs of knowledge used in the private-multiplier multiplication protocol to extract the values $s_1, \ldots, s_t$; if a party fails to provide a correct proof it is discarded. Subsequently, the simulator runs phase 1 for parties $P_{t+1}, \ldots, P_{n-1}$ as in the real protocol, leaving $[\![x_{n-1}]\!]$, with $x_{n-1} = s_1 \cdots s_{n-1}x$, as intermediate encryption. For party P_n, however, the simulator picks $s'_n \in_R \{-S, S\}$, where $S = \prod_{i=1}^{t} s_i$, and it computes a commitment $\langle\!\langle s'_n x_{n-1} \rangle\!\rangle$ and an encryption $[\![s'_n xy]\!]$, from $[\![x_{n-1}]\!]$ and $[\![xy]\!]$, respectively. Writing $s_n = s'_n x_{n-1}$, the simulator then simulates the private multiplier multiplication protocol for multiplier $\langle\!\langle s_n \rangle\!\rangle$ and multiplicands $[\![x_{n-1}]\!]$, $[\![y_{n-1}]\!]$ and outputs $[\![s'_n]\!]$, $[\![s'_n xy]\!]$, which are the correct outputs since $s'_n = s_n x_{n-1}$ and $s'_n xy = s'_n x_{n-1} y_{n-1} = s_n y_{n-1}$.

The output of phase 1 consists of encryptions $[\![s'_n]\!]$ and $[\![s'_n xy]\!]$. By construction, the simulator is able to perform the decryption in phase 2 itself, producing $s'_n \in \{-S, S\}$ as output. The simulator for the threshold decryption protocol is used for encryption $[\![s'_n]\!]$ using s'_n as an additional input (see Section 2). If decryption fails due to an insufficient number of correct decryption shares, the simulation stops, as in the real protocol. If $S \notin \{-1, 1\}$, decryption fails and a proof that $s_i \in \{-1, 1\}$ is generated for each party P_i, by letting the adversary do this for parties $P_1, \ldots, P_t$ (of which at least one fails), running the real protocol for parties $P_{t+1}, \ldots, P_{n-1}$, and using a simulation for P_n. After discarding the failing parties among $P_1, \ldots, P_t$, the simulation is continued by simulating another run of phase 1.

Finally, the simulator computes the encryption $[\![s'_n s'_n xy]\!] = [\![xy]\!]$, which is clearly the correct output. □

If the total number of parties is large compared to the total number of conditional gates to be evaluated, an alternative way to guarantee robustness is to let the parties use encryptions $[\![s_i]\!]$ instead of commitments $\langle\!\langle s_i \rangle\!\rangle$ in phase 1. Again, if $x_n \notin \{-1, 1\}$ in phase 2, all parties are required to prove that $s_i \in \{-1, 1\}$. Failing parties are discarded and their s_i values are decrypted to correct the value of x_n.

The performance of the protocol is as follows (analyzing the case that no party is cheating). The performance is determined by the communication complexity (in bits) and the round complexity. In phase 1 each party applies the private-multiplier multiplication protocol, broadcasting about 10 values. For decryption each party broadcasts 3 values at the most. Hence, the communication complexity is $O(nk)$ where the hidden constant is very small. In general, the round complexity is $O(n)$, which is high, but in case of two-party computation it is $O(1)$. Also, when many conditional gates are to be evaluated in parallel, one may take advantage of the fact that the order in which parties $P_1, \ldots, P_n$ execute phase 1 of the conditional gate protocol can be chosen arbitrarily.

3.3 XOR-Homomorphic ElGamal Encryption

As a direct application of the conditional gate, we obtain an xor-homomorphic ElGamal encryption scheme. (The converse problem of constructing $(\mathbb{Z}_q, +)$-homomorphic schemes, $q > 2$, from xor-homomorphic schemes, such as the Goldwasser-Micali cryptosystem [GM84], is considered in [KMO01].)

Given $[\![x]\!], [\![y]\!]$ with $x, y \in \{0, 1\}$, $[\![x \oplus y]\!]$ is computed as follows, using one threshold decryption (cf. footnote 2):

1. Publicly convert $[\![x]\!]$ to $[\![x']\!]$ with $x' = 2x - 1 \in \{-1, 1\}$.
2. Apply the conditional gate to $[\![x']\!]$ and $[\![y]\!]$ to obtain $[\![x'y]\!]$.
3. Publicly compute $[\![x - x'y]\!]$, which is equal to $[\![x \oplus y]\!]$.

The work per party is very limited, about 13 exponentiations for each conditional gate. In contrast, the Mix and Match approach of [JJ00] would require each party to mix the 4 rows of a truth table for $x \oplus y$ in a verifiable way (Mix step, requiring 24 exponentiations for blinding the entries and, say, 6×12 exponentiations for the correctness proof, using the efficient protocol of [Gro03]), and perform on average 4 plaintext equality tests to find $[\![x \oplus y]\!]$ given $[\![x]\!]$ and $[\![y]\!]$ (Match step, requiring 4×7 exponentiations). Hence, the conditional gate provides approximately a ten-fold improvement, counting exponentations.

4 Circuit Evaluation

In this section, we briefly describe a protocol for evaluating a given circuit composed of elementary gates. Recall that our elementary gates operate over $\mathbb{Z}_q$, except that the first input of a conditional gate is required to belong to a two-valued domain. It is clear that these elementary gates suffice to emulate any Boolean circuit. Specifically, any operator on two bits $x, y \in \{0, 1\} \subseteq \mathbb{Z}_q$ can be expressed uniquely as a polynomial of the form $a_0 + a_1x + a_2y + a_3xy$ with coefficients in $\mathbb{Z}_q$. Hence, any binary operator can be expressed using at most one conditional gate.

The protocol operates in much the same manner as the protocol for circuit evaluation of, for instance, [CDN01]. For convenience, we assume that the parties $P_1, \ldots, P_n$ evaluating the circuit are exactly the same as the parties for which the $(t + 1, n)$-threshold cryptosystem has been set-up, where $t < n$. The circuit is then evaluated in three phases:

1. The parties encrypt their inputs using the homomorphic cryptosystem $[\![\cdot]\!]$, and the parties are required to provide a proof of knowledge for their inputs, and possibly that the inputs belong to a dichotomous domain.
2. The parties then jointly evaluate the circuit gate-by-gate. Conditional gates at the same depth of the circuit are evaluated in parallel.
3. Finally, the parties jointly decrypt the outputs of the circuit.

As described in the previous section, parties failing at some stage in the protocol are discarded immediately. As long as no more than $t < n/2$ parties fail in total, the protocol will complete and all parties will learn the output. The case of $n/2 \leq t < n$ will be discussed below.

The formal security analysis of [CDN01] can be adapted to show that our protocol is secure against a static, active adversary corrupting at most $t < n$ parties, assuming the intractability of the DDH problem. This follows from the fact that we are able to simulate the multiplication protocols of Section 3 in a statistically indistinguishable manner, provided that the simulator for these protocols is given encryptions of the correct output values. We thus achieve the same level of security as [CDN01]. The important difference is that [CDN01] incorporates a general multiplication gate, for which they need an RSA-like cryptosystem such as Paillier's cryptosystem to get an efficient multiplication protocol, while we incorporate a restricted multiplication gate, for which we have presented an efficient multiplication protocol using the ElGamal cryptosystem.

4.1 Private Outputs

In this section, we propose a new non-interactive protocol for achieving private outputs in the context of secure multiparty computation based on homomorphic threshold cryptosystems. Previous methods require the receiving parties to perform some blinding step, as part of the decryption protocol. For our method, it suffices to know the public keys of the receiving parties.

We need a different method than [CDN01] to deal with private outputs anyway, since their method would require us to decrypt an ElGamal encryption of a random message in $\mathbb{Z}_q$. Suppose $[\![m]\!]$ is an encryption of a private output for party P_j, that is, output m is intended for party P_j only. Briefly, the method of [CDN01] is to let party P_j first blind encryption $[\![m]\!]$ by multiplying it with a random encryption $[\![r]\!]$ for some $r \in_R \mathbb{Z}_q$. The encryption $[\![m+r]\!]$ is then jointly decrypted, resulting in the value $m' = m + r$, from which (only) party P_j is able to compute $m = m' - r$. This method critically depends on the ability to decrypt arbitrary messages. Using an RSA-like cryptosystem, such as Paillier's cryptosystem, this is no problem. Using ElGamal encryption, however, we cannot decrypt $[\![m + r]\!]$ (see Section 2). A first way out is to adapt the ElGamal decryption step to output g^{m+r} instead of $m+r$; the receiving party may divide this value by g^r to obtain g^m from which m may be determined, assuming m is from a small set.

We note however that in general it is undesirable that interaction with the receiving parties is required to produce private outputs. Therefore, we present a protocol for which *no* interaction with the receiving party is required. The protocol runs as follows. Let $(a, b) \in [\![m]\!]$ be an output intended for party P_j and let $h_j = g^{\alpha_j}$ denote P_j's public key. Recall from Section 2 that threshold decryption requires each party P_i to produce the value a^{α_i} along with a proof of correctness. We modify this step as follows, by releasing a^{α_i} encrypted under P_j's public key and adapting the proof of correctness accordingly:

1. Each party P_i outputs an encryption $(c_i, d_i) = (g^{r_i}, h_j^{r_i} a^{\alpha_i})$ with $r_i \in_R \mathbb{Z}_q$ along with a proof that it knows r_i, α_i satisfying
$$h_i = g^{\alpha_i}, \quad c_i = g^{r_i}, \quad d_i = h_j^{r_i} a^{\alpha_i}.$$
2. For decryption, party P_j first uses Lagrange coefficients λ_i to compute the following product for a set of $t+1$ valid shares (c_i, d_i):
$$\prod_i (c_i, d_i)^{\lambda_i} = (g^{\sum_i \lambda_i r_i}, h_j^{\sum_i \lambda_i r_i} a^{\sum_i \lambda_i \alpha_i}).$$
Then P_j decrypts this product using its private key α_j to obtain $a^{\sum_i \lambda_i \alpha_i} = a^{\alpha}$. Party P_j then proceeds to recover $g^m = b/a^{\alpha}$, from which it finds m assuming that m belongs to a relatively small, known subset of $\mathbb{Z}_q$.

The protocol requires only a small amount of additional work compared to the basic protocol for decrypting public outputs, where each party P_i outputs a^{α_i} along with a proof of correctness (cf. step 1), from which anyone is then able to recover $a^{\sum_i \lambda_i \alpha_i} = a^{\alpha}$ using $t+1$ valid shares (cf. step 2).

4.2 Fairness

Recall that t denotes the maximum number of corrupted parties tolerated by the circuit evaluation protocol. For $t < n/2$, that is, the case of a dishonest minority, the protocol achieves robustness. We now extend the protocol to handle the two-party case $t = 1, n = 2$ (which is a special case of a dishonest majority, $n/2 \leq t < n$).

For the two-party case we give up on robustness, since one cannot prevent one of the parties from quitting the protocol prematurely. If a party chooses to do so, however, it should not gain any advantage from it. If a protocol achieves this property, the protocol is said to be *fair*.

An important observation for the above circuit evaluation protocol is that neither party gains any advantage from quitting the protocol in phase 1 or phase 2 of the protocol. In particular, consider the case that party P_2, say, chooses to quit during the threshold decryption step of a conditional gate, for which party P_1 has already produced its decryption share. In that case, only P_2 learns the decrypted value x_n, but this value cannot possibly give P_2 an advantage, as follows from the simulation in the proof of Theorem 1.

Therefore, to achieve fairness, we only need to protect the decryption of the output values. For this purpose, we will apply a protocol similar to that of [BST01]. In [BST01], however, the protocol steps for achieving fairness are intertwined with the original protocol steps, while in our protocol the additional steps for achieving fairness are strictly limited to the decryption of the output values.

Let an encryption (a, b) be given. Recall that $(2,2)$-threshold decryption, requires party P_i to provide $d_i = a^{\alpha_i}$, $i = 1, 2$, along with a proof that this value is correct w.r.t. the public key $h_i = g^{\alpha_i}$ of party P_i. Instead of directly revealing this value, we will release it gradually using the following protocol, where k is a security parameter, $k < \log_2 q$, and $h \in \langle g \rangle$ denotes an additional generator for which $\log_g h$ is unknown to parties P_1, P_2:

1. For $i = 1, 2$, party P_i chooses $\epsilon_{ij} \in_R \{0,1\}, \alpha_{ij} \in_R \mathbb{Z}_q$ for $j = 0, \ldots, k-1$ subject to the condition that $\alpha_i = \sum_{j=0}^{k-1} \alpha_{ij} 2^j$. Party P_i then broadcasts the values $d_{ij} = a^{\alpha_{ij}} h^{\epsilon_{ij}}$, $j = 0, \ldots, k-1$ along with a proof that each $\epsilon_{ij} \in \{0,1\}$ and a proof that $\prod_{j=0}^{k-1} d_{ij}^{2^j} = a^{\alpha_i} h^{\epsilon}$, where $\alpha_i = \log_g h_i$, for some value ϵ.
2. Set $j = k-1$. Parties P_1, P_2 repeatedly execute the following step. For $i = 1, 2$, party P_i broadcasts values $\alpha_{ij}, \epsilon_{ij}$. If these values verify correctly against d_{ij}, the value of j is decremented and the step is repeated if $j > 0$.
3. Once $j = 0$ both parties release ϵ_{i0} along with a proof of knowledge for a witness α_{i0} satisfying $d_{i0} h^{-\epsilon_{i0}} = a^{\alpha_{i0}}$.
4. Both parties are able to recover the missing value a^{α_i}, as follows:

$$a^{\alpha_i} = d_{i0} h^{-\epsilon_{i0}} a^{\sum_{j=1}^{k-1} \alpha_{ij} 2^j}.$$

At each stage of the protocol, either party is at most one bit ahead of the other party. If one sets $k = 80$, for instance, it is clearly infeasible for both parties to compute the missing value a^{α_i} at step 1, as it requires a search over 2^k possible values for $\epsilon_{i,k-1}, \ldots, \epsilon_{i0}$. At each later step, the search space is reduced in size by a factor of two.

The protocol does not leak any information on α_i beyond what is implied by the output values a^{α_i}. The protocol can be run in parallel for decrypting multiple outputs at the same time, and the protocol can be combined easily with our protocol for private outputs presented above.

The above protocol achieves a basic level of fairness. In [Pin03] a strengthened notion of fairness for two-party computation is considered, which also addresses the case where one party may be considerably more powerful than the other party; the timed commitments used to resolve this problem, however, critically depend on the hardness of factoring. (The recent paper [GMY04b] describes a way to cover fairness in a universally composable way, for static adversaries. In particular, the result of [CDN01] is extended to cover fairness as well, using a factoring related assumption to achieve timed commitments.) Apart from this difference, the result of [Pin03] is comparable to our result. The difference is that [Pin03] is based on Yao's garbled circuit approach, while our approach is based on homomorphic threshold cryptosystems. In both cases, however, the changes to make the protocol fair are limited to the output stage, where some form of gradual release is used in combination with a method to ensure that commitments opened during gradual release indeed contain the correct output of the computation.

5 Relational and Arithmetic Operators

In this section we apply our set of elementary gates over $\mathbb{Z}_q$ to obtain efficient circuits for basic operations such as integer comparison and integer addition. In most cases, the inputs are required to be given by their binary representations. We consider the general case, in which the circuits operate on encrypted inputs, producing encrypted outputs, such that they can be used as building blocks in constructing circuits for more elaborate functions, either in a two-party setting or in a general multiparty setting.

5.1 sgn($x - y$)

Below, we present an efficient protocol for comparing two (non-negative) integer values x and y. The inputs are given as sequences of encrypted bits, $[\![x_{m-1}]\!], \ldots, [\![x_0]\!]$ and $[\![y_{m-1}]\!], \ldots, [\![y_0]\!]$, with $x = \sum_{i=0}^{m-1} x_i 2^i$, $y = \sum_{i=0}^{m-1} y_i 2^i$. The output of the protocol consists of an encryption $[\![\mathrm{sgn}(x-y)]\!]$, where sgn denotes the signum function:

$$\mathrm{sgn}\, z = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0. \end{cases}$$

Using that $x_i^2 = x_i$ and $y_i^2 = y_i$ for $x_i, y_i \in \{0,1\}$, the general strategy is now to examine the unique multilinear polynomial p over $\mathbb{Z}_q$ satisfying $p(x_0, \ldots, x_{m-1}, y_0, \ldots, y_{m-1}) = \mathrm{sgn}(x-y)$ for all x, y, $0 \le x, y < 2^m$. The problem that remains is to find an efficient circuit (or, equivalently, an oblivious evaluation order) for the polynomial p.

As a first step, we consider the evaluation $p = s_0$ with

$$s_{m-1} = 0, \qquad s_{i-1} = s_i + (1 - s_i^2)(x_i - y_i).$$

Clearly, this results in the correct output value. However, we cannot evaluate the term s_i^2 by means of a conditional gate since s_i is three-valued ($s_i \in \{-1, 0, 1\}$). This is easily resolved by introducing an auxiliary *binary* sequence v_i, with $v_i = 1 - s_i^2$:

$$\begin{aligned} s_{m-1} &= 0, \qquad & s_{i-1} &= s_i + v_i(x_i - y_i), \\ v_{m-1} &= 1, \qquad & v_{i-1} &= v_i - v_i(x_i - y_i)^2. \end{aligned}$$

Now, it is easy to draw up a circuit using $3m - 2$ conditional gates (using that $s_{m-1} = 0$ and $v_{m-1} = 1$ are publicly known values, hence need not be encrypted).

We note that the bits may also be traversed in the opposite direction, starting at the least significant bit. This results in the following sequence, with s'_m as output:

$$s'_0 = 0, \qquad s'_{i+1} = (1 - (x_i - y_i)^2)s'_i + x_i - y_i.$$

This method only needs $2m - 2$ conditional gates: per iteration, one conditional gate to compute $x_i y_i$ and one to subsequently compute $(1 - (x_i - y_i)^2)s'_i$ with $1 - (x_i - y_i)^2$ as dichotomous multiplier. Here we take full advantage of the fact that the conditional gate does not put any constraints on the multiplicand: whereas a Boolean circuit for $\mathrm{sgn}(x - y)$ requires all intermediate values to be binary, our circuit uses non-binary intermediate values, such as the ternary s'_i's.

5.2 $x > y$

The output of $x > y$ consists of one bit only, which is set to 1 if $x > y$ and to 0 otherwise. Starting at the least significant bit, the output is given by t_m, where

$$t_0 = 0, \qquad t_{i+1} = (1 - (x_i - y_i)^2)t_i + x_i(1 - y_i).$$

This method requires $2m-1$ conditional gates. (For comparison we note that the best known circuit using only logical gates requires $5m$ binary gates, e.g. using the circuit for $Bigger_k(X, Y)$ of [KO02]. Similarly, for computing $Max(X, Y)$ given $Bigger_k(X, Y)$, $2m$ additional gates are required in [KO02], while we can compute the bits of $z = max(x, y)$ by setting $z_i = y_i - t_m(x_i + y_i)$, using only m additional conditional gates.)

We now specialize this solution for $x > y$ to obtain a solution for Yao's basic millionaires problem [Yao82]. In this case, the protocol is run by two parties, providing x and y respectively as *private* inputs. This allows for a much more efficient solution, as the conditional gates can all be replaced by the private-multiplier gates of Section 3.1. The private-multiplier gates can be even optimized slightly by using Pedersen commitments instead of ElGamal encryptions, and using that the multipliers are binary.

The total computational cost of our solution to Yao's millionaires problem, including the cost of the distributed key generation and the decryption of the result, is dominated by the cost of about $2m$ private-multiplier gates (computing $[\![y_i t_i]\!]$ and $[\![x_i(t_i - 2y_i t_i - y_i)]\!]$ as intermediate values), which require 6 exponentiations each, hence $12m$ exponentiations in total (starting at the least significant bit). To the best of our knowledge, this is the most efficient solution to date. Here, we cover the malicious case (unlike many other papers on the millionaires problem, that only deal with the semi-honest case, e.g., [Fis01,NN01,IG03]), but we do not cover fairness. We also note that we do not need an auxiliary trusted party, as in [Cac99], although that paper achieves fairness as well at a relatively low cost. Finally, while most other solutions rely on an RSA-like assumption, our solution is secure under the standard DDH assumption. This is also true for the solution of [KO02], but their solution is much less efficient because their circuits are evaluated using the expensive Mix and Match gates of [JJ00].

5.3 $x = y$

For testing equality of x and y, the following sequence can be used, where the output $u_m = 0$ iff $x = y$:

$$u_0 = 0, \qquad u_{i+1} = (1 - (x_i - y_i)^2)u_i + (x_i - y_i)^2.$$

The order in which the bits are processed is actually irrelevant. This method requires $2m - 1$ conditional gates, returning the output bit in encrypted form.

The socialist millionaires problem, a variant introduced by [JY96], is to evaluate $x = y$ for a two-party setting, where $x, y \in \mathbb{Z}_q$ are the respective *private* inputs, and the output may be public. The currently best solution is due to [BST01], using only $O(1)$ exponentiations, hence without using the binary representations of x and y. We obtain a solution in a similar vein as follows. The parties broadcast $[\![x]\!]$ and $[\![y]\!]$, resp., and jointly form $[\![r]\!]$, where $r \in_R \mathbb{Z}_q$ and neither of the parties knows r. Using the private multiplier gate, the parties then compute $[\![(x-y)r]\!]$, which is jointly decrypted to obtain $g^{(x-y)r}$ (rather than obtaining $(x-y)r$). If $g^{(x-y)r} = 1$, then w.v.h.p. $x = y$, otherwise $x \neq y$.

5.4 $x + y$ and $x * y$

Given $[\![x]\!], [\![y]\!]$, one obtains $[\![x+y]\!]$ directly using the homomorphic property. A nice application of the conditional gate is that $[\![xy]\!]$ can also be computed efficiently, if we assume that x is given in binary form.

Given $[\![x_{m-1}]\!], \ldots, [\![x_0]\!]$ and $[\![y]\!]$, where $y \in \mathbb{Z}_q$, one computes $[\![xy]\!]$ using that $xy = \sum_{i=0}^{m-1} x_i(y2^i)$. This method requires only m conditional gates, whereas a standard Boolean circuit would require $O(m^2)$ bit multiplications.

6 Concluding Remarks

We envision a practical system supporting ad hoc contacts among a large group of peer users. Since efficient DKG protocols for $(2,2)$-threshold ElGamal are easily achieved, our results show that *any* pair of users is able to engage in a two-party computation for evaluating some dynamically agreed upon function. For example, the circuits of the previous section lead to solutions for tasks of practical interest, such as profile matching, allowing two users with profiles (length m bit vectors) x and y, resp., to evaluate $\Delta(x,y) > T$, where $\Delta(x,y) = \sum_{i=1}^{m} x_i y_i$ is an example similarity measure and T is a threshold.

Further research is required for a full comparison with some recent approaches to secure (two-party) computation. For instance, an interesting approach is presented in [GMY04a], which is based on committed oblivious transfer instead of homomorphic threshold encryption. The amount of work per gate is comparable to the work for a conditional gate, but the hidden constants for their approach are larger than in our case. This is partly due to the fact that their solution is designed to be universally composable, but remains true if their solution is 'downgraded' to a protocol for static adversaries; per gate, one party uses 5 bit commitments and proves a number of relations for these commitments, followed by a $\binom{4}{1}$ oblivious transfer. For a full comparison with [GMY04a], our solution needs to be 'upgraded' to a universally composable one, e.g., following the approach of [DN03]. This would provide an interesting alternative, as the extension of [GMY04a] to the multiparty case requires *each pair* of parties to run their basic two-party protocol for each multiplication gate, while with our approach the parties run a single joint protocol for each conditional gate.

A well-known alternative to the gate-by-gate approach, is Yao's garbled circuit approach for two-party computation. The Fairplay system is designed to evaluate the practical merits of the garbled circuit approach, including some optimizations that will pay off for sufficiently large circuits [MNPS04]. We expect a trade-off showing that the garbled circuit approach is best for large circuits whereas a gate-by-gate approach is best for small circuits, or rather circuits for which the number of inputs is proportional to the total number of gates.

Acknowledgements We thank the anonymous referees for their helpful comments.

References

[ACS02] J. Algesheimer, J. Camenisch, and V. Shoup. Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In *Advances in Cryptology—CRYPTO '02*, volume 2442 of *Lecture Notes in Computer Science*, pages 417–432, Berlin, 2002. Springer-Verlag.

[BST01] F. Boudot, B. Schoenmakers, and J. Traoré. A fair and effcient solution to the socialist millionaires' problem. *Discrete Applied Mathematics*, 111(1–2):23–36, 2001. Special issue on Coding and Cryptology.

[Cac99] C. Cachin. Efficient private bidding and auctions with an oblivious third party. In *6th ACM Conference on Computer and Communications Security*, pages 120–127. ACM press, 1999.

[CDN01] R. Cramer, I. Damgård, and J.B. Nielsen. Multiparty computation from threshold homomorphic encryption. In *Advances in Cryptology—EUROCRYPT '01*, volume 2045 of *Lecture Notes in Computer Science*, pages 280–300, Berlin, 2001. Springer-Verlag.

[CDS94] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology—CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187, Berlin, 1994. Springer-Verlag.

[CS02] R. Cramer and V. Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In *Advances in Cryptology—EUROCRYPT '02*, volume 2332 of *Lecture Notes in Computer Science*, pages 45–64, Berlin, 2002. Springer-Verlag.

[DJ03] I. Damgård and M. Jurik. A length-flexible threshold cryptosystem with applications. In *ACISP 2003*, volume 2727 of *Lecture Notes in Computer Science*, pages 350–364, Berlin, 2003. Springer-Verlag.

[DN03] I. Damgård and J.B. Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In *Advances in Cryptology—CRYPTO '03*, volume 2729 of *Lecture Notes in Computer Science*, pages 247–264, Berlin, 2003. Springer-Verlag.

[FH96] M. Franklin and S. Haber. Joint encryption and message-efficient secure computation. *Journal of Cryptology*, 9(4):217–232, 1996.

[Fis01] M. Fischlin. A cost-effective pay-per-multiplication comparison method for millionaires. In *Progress in Cryptology – CT-RSA 2001*, volume 2020 of *Lecture Notes in Computer Science*, pages 457–471, Berlin, 2001. Springer-Verlag.

[Gil99] N. Gilboa. Two party RSA key generation. In *Advances in Cryptology—CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 116–129, Berlin, 1999. Springer-Verlag.

[GJKR99] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *Advances in Cryptology—EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 295–310, Berlin, 1999. Springer-Verlag.

[GJKR03] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure applications of Pedersens distributed key generation protocol. In *Cryptographers' Track RSA 2003*, volume 2612 of *Lecture Notes in Computer Science*, pages 373–390, Berlin, 2003. Springer-Verlag.

[GM84] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[GMY04a] G. Garay, P. MacKenzie, and K. Yang. Efficient and universally composable committed oblivious transfer and applications. In *Proc. 1st Theory of Cryptography Conference (TCC 2004)*, volume 2951 of *Lecture Notes in Computer Science*, pages 297–316, Berlin, 2004. Springer-Verlag.

[GMY04b] J. Garay, P. MacKenzie, and K. Yang. Efficient and secure multi-party computation with faulty majority and complete fairness, 2004. Submitted. Available at http://eprint.iacr.org/2004/009/.

[Gro03] J. Groth. A verifable secret shuffle of homomorphic encryptions. In *Public Key Cryptography—PKC '03*, volume 2567 of *Lecture Notes in Computer Science*, pages 145–160, Berlin, 2003. Springer-Verlag.

[IG03] I. Ioannidis and A. Grama. An efficient protocol for Yao's millionaires' problem. In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03)*, page 6 pages. IEEE, 2003.

[JJ00] A. Juels and M. Jakobsson. Mix and match: Secure function evaluation via ciphertexts. In *Advances in Cryptology—ASIACRYPT '00*, volume 1976 of *Lecture Notes in Computer Science*, pages 162–177, Berlin, 2000. Springer-Verlag.

[JY96] M. Jakobsson and M. Yung. Proving without knowing: On oblivious, agnostic and blindfolded provers. In *Advances in Cryptology—CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 186–200, Berlin, 1996. Springer-Verlag.

[KMO01] J. Katz, S. Myers, and R. Ostrovsky. Cryptographic counters and applications to electronic voting. In *Advances in Cryptology—EUROCRYPT '01*, volume 2045 of *Lecture Notes in Computer Science*, pages 78–92, Berlin, 2001. Springer-Verlag.

[KO02] K. Kurosawa and W. Ogata. Bit-slice auction circuit. In *ESORICS 2002*, volume 2502 of *Lecture Notes in Computer Science*, pages 24–38, Berlin, 2002. Springer-Verlag.

[MNPS04] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay—a secure two-party computation system. In *Proceedings of Usenix Security 2004, August 9–13*, 2004. To appear.

[NN01] M. Naor and K. Nissim. Communication complexity and secure function evaluation. In *Mixer II, October 9, 2001, NEC Research Institute, Princeton, New Jersey*, DIMACS Mixer Series, 2001.

[Ped91] T. Pedersen. A threshold cryptosystem without a trusted party. In *Advances in Cryptology—EUROCRYPT '91*, volume 547 of *Lecture Notes in Computer Science*, pages 522–526, Berlin, 1991. Springer-Verlag.

[Pin03] B. Pinkas. Fair secure two-party computation. In *Advances in Cryptology—EUROCRYPT '03*, volume 2656 of *Lecture Notes in Computer Science*, pages 87–105, Berlin, 2003. Springer-Verlag.

[Yao82] A. Yao. Protocols for secure computations. In *Proc. 23rd IEEE Symposium on Foundations of Computer Science (FOCS '82)*, pages 160–164. IEEE Computer Society, 1982.