HOW TO PLAY ANY MENTAL GAME

or

A Completeness Theorem for Protocols with Honest Majority

(Extended Abstract)

Oded Goldreich	Silvio Micali	Avi Wigderson
Dept. of Computer Sc.	Lab. for Computer Sc.	Inst. of Math. and CS
Technion	MIT	Hebrew University
Haifa, Israel	Cambridge, MA 02139	Jerusalem, Israel

Abstract

We present a polynomial-time algorithm that, given as a input the description of a game with incomplete information and any number of players, produces a protocol for playing the game that leaks no partial information, provided the majority of the players is honest.

Our algorithm automatically solves all the multi-party protocol problems addressed in complexity-based cryptography during the last 10 years. It actually is *a completeness theorem* for the class of distributed protocols with honest majority. Such completeness theorem is optimal in the sense that, if the majority of the players is not honest, some protocol problems have no efficient solution[C].

1. Introduction

Before discussing how to "make playable" a general game with incomplete information (which we do in section 6) let us address the problem of making playable a special class of games, the *Turing machine games* (*Tm-games* for short).

Informally, n parties, respectively and individually owning secret inputs $x_1,\ldots,x_n$, would like to *correctly* run a given Turing machine M on these x_i's while keeping the maximum possible *privacy* about them. That is, they want to compute $y=M(x_1,\ldots,x_n)$ without revealing more about the x_i's than it is already contained in the value y itself. For instance, if M computes the sum of the x_i's, every single player should not be able to learn more than the sum of the inputs of the other parties. Here M may very well be a probabilistic Turing machine. In this case, all players want to agree on a single string y, selected with the right probability distribution, as M's output.

The correctness and privacy constraint of a Tm-game can be easily met with the help of an extra, trusted party P. Each player i simply gives his secret input x_i to P. P will privately run the prescribed Turing machine, M, on these inputs and publically announce M's output. Making a Tm-game playable essentially means that the correctness and privacy constraints can be satisfied by the n players themselves, without invoking any extra party. Proving that Tm-games are playable retains most of the flavor and difficulties of our general theorem.

2. Preliminary Definitions

2.1 Notation and Conventions for Probabilistic Algorithms.

We emphasize the number of inputs received by an algorithm as follows. If algorithm A receives only one input we write "$A(\cdot)$", if it receives two inputs we write $A(\cdot,\cdot)$ and so on.

RV will stand for "random variable"; in this paper we only consider RVs that assume values in

Work partially supported by NSF grants DCR-8509905 and DCR-8413577, an IBM post-doctoral fellowship and an IBM faculty development award. The work was done when the first author was at the Laboratory for Computer Science at MIT; and the second author at the mathematical Sciences Research Institute at UC-Berkeley.

$\{0,1\}^*$. In fact, we deal almost exclusively with random variables arising from probabilistic algorithms. (We make the natural assumption that all parties may make use of probabilistic methods.)

If $A(\cdot)$ is a probabilistic algorithm, then for any input x the notation $A(x)$ refers to the RV which assigns to the string σ the probability that A, on input x outputs σ. If S is a RV that assigns positive probability only to a single element e, we denote the value e by S. (For instance, if $A(\cdot)$ is an algorithm that, on input x outputs x^3, then we may write $A(2)=8$.) This is in agreement with traditional notation.

If $f(\cdot)$ and $g(\cdot,\cdots)$ are probabilistic algorithms then $f(g(\cdot,\cdots))$ is the probabilistic algorithm obtained by composing f and g (i.e. running f on g's output). For any inputs $x,y,\cdots$ the associated RV is denoted $f(g(x,y,\cdots))$.

Let PA denote the set of probabilistic polynomial-time algorithms. We assume that a natural representation of these algorithms as binary strings is used.

By 1^k we denote the unary representation of integer k.

2.2 Game Networks and Distributed Algorithms

Let us start by briefly describing the communication networks in which games will be played. This is the standard network supporting the execution of multi-party protocols.

Informally, a *game network* of size n is a collection of (interacting) probabilistic polynomial-time Turing machines. Each machine has a private read-only input tape, a private write-only output tape and a private read-write work tape. All machines share a common read-only input tape and a common write-only output tape. The n machines communicate by means of $n\cdot(n-1)$ special tapes. Machine i publically sends messages (strings) to machine j by means of a special tape $i \rightarrow j$ on which only i can write and that all other machines can read. There is a common clock whose pulses define time intervals $1,2,\ldots$. Messages are sent at the beginning of a time interval and are received within the same time interval. We stress, though, that our result is largely independent from the specific communication mechanism, and also holds for "less equipped" communication networks.(1)

A probabilistic distributed algorithm S running in a game network of size n is a sequence of programs $S=(S_1,\ldots,S_n)$, where S_i is the program of the ith Turing machine in the network. We denote by PDA the class of all probabilistic polynomial-time distributed algorithms.

Let $S\in PDA$ run in a game network of size n with common input CI and (respective) private inputs $x_1,\ldots,x_n$. Then $HS(x_1,\ldots,x_n,CI)$ denotes the RV consisting of the *public history*, that is the sequence of all messages sent in an execution of S; $HS_i(x_1,\ldots,x_n,CI)$ denotes the RV consisting of the *private history* of machine i, that is the sequence of the internal configurations of machine i in an execution of S; for $T\subset\{1,\ldots,n\}$, $HS_T(x_1,\ldots,x_n)$ denotes the vector of the private histories of the members of T in an execution of S; and $OS_i(x_1,\ldots,x_n,CI)$ denotes the RV consisting of the private output of machine i in an execution of S.

2.3 Adversaries

We consider two interesting types of adversaries (faulty machines) in a game network: passive ones (a new notion) and malicious ones (a more standard notion).

A *passive adversary* is a machine that may compute more than required by its prescribed program, but the messages it sends and what it outputs are in accordance to its original program. (Passive adversaries may be thought as machines who only try to violate the privacy constraint. They keep on running their prescribed programs correctly, but also run, "on the side", their favorite polynomial-time program to try to compute more than their due share of knowledge. In an election protocol, a passive adversary may be someone who respects the majority's opinion -and thus does not want to corrupt the tally- and yet wants to discover who voted for whom.)

A *malicious adversary* is, instead, a machine that deviates from its prescribed program in any

(1) For instance, there may be only one communication tape. In this case, digital signatures can be used to authenticate the sender. In case that not all machines may read all communication tapes, Byzantine agreement can be used to simulate the fact that all processors agree on what message machine i has sent to machine j at time t. The common clock may be replaced by local clocks that don't drift "too much". The quite tight synchrony of the message delivery can be replaced by a feasible upper bound on the time it takes a message to be delivered and so on.

possible action. That is, we allow the program of such a machine to be replaced by any fixed probabilistic polynomial-time program. (Malicious adversary not only have a better chance of disrupting the privacy constraint, but could also make the outcome of a Tm-game vastly different than in an ideal run with a trusted party.)

We allow machines in a game network to become adversarial in a *dynamic fashion*, during the execution of a protocol. We also allow adversarial machines (of either type) to undetectedly cooperate. Adversarial machines are not allowed, however, to monitor the private tapes or the internal state of good machines.

We believe the malicious-adversary scenario to be the most adversarial among all the natural scenarios in which cryptography may help.

Jumping haed, we will show that all Tm-games are playable with any number of passive adversaries or with $< n/2$ malicious adversaries.

2.4 Indistinguishability of Random Variables

Throughout this paper, we will only consider families of RVs $U=\{U_k\}$ where the parameter k ranges in the natural numbers. Let $U=\{U_k\}$ and $V=\{V_k\}$ be two families of RVs. The following notion of computational indistinguishability expresses the fact that, when the length of k increases, U_k becomes "replaceable" by V_k in the following sense. A random sample is selected either from U_k or from V_k and it is handed to a "judge". After studying the sample, the judge will proclaim his verdict: 0 or 1. (We may interpret 0 as the judge's desicion that that the sample came from U_k; 1 as the desicion that the sample came from V_k.) It is then natural to say that U_k becomes "replaceable" by V_k for k large enough if, when k increases, the verdict of *any* computationally bounded judge becomes "meaningless", that is essentially uncorrelated to which of the two distributions the sample came from.

To formalize the notion of computational indistinguishablity we make use of nonuniformity. Thus, our "judge", rather than polynomial time Turing machine, will be a *poly-size family of circuits.* That is a family $C=\{C_k\}$ of Boolean circuits C_k with one Boolean output such that, for some constants $c,d>0$, all $C_k \in C$ have at most k^c gates and k^d Boolean inputs. In order to feed samples from our probability distributions to such circuits, we will consider only *poly-bounded* families of RVs. That is families $U=\{U_k\}$ such that, for some constant $e>0$, all RV $U_k \in U$ assigns positive probability only to strings whose length is exactly k^e. If $U=\{U_k\}$ is a poly-bounded family of RVs and $C=\{C_k\}$ a poly-size sequence of circuits, we denote by $P(U,C,k)$ the probability that C_k outputs 1 on input a random strings from U_k. (Here we assume that the length of the strings that are assigned positive probability by U_k equals the number of Boolean inputs of C_k.)

Definition (Computational indistinguishability): Two poly-bounded families of RVs U and V are *computationally indistinguishable* if for all poly-size family of circuits C, for all constants $f>0$ and all sufficiently large $k \in N$,

$$|P(U,C,k)-P(V,C,k)|<k^{-f}.$$

This notion was already used by Goldwasser and Micali [GM] in the context of encryption and by Yao [Y] in the context of pseudo-random generation. For other notions of indistinguishability and further discussion see [GMR].

Remark 1: Let us point out the robustness of the above definition. In this definition, we are handing our computationally bounded "judge" only samples of size 1. This, however, is not restrictive. It should be noticed that two families of RVs $\{U_k\}$ and $\{V_k\}$ are computationally indistinguishable with respect to samples of size 1 if and only if they are computationally indistinguishable with respect to samples whose size is bounded by a fixed polynomial in k.

3. Tm-games With Passive Adversaries

An *Tm-game problem* consists of a pair $(\overline{M},1^k)$, that is, the description of a Turing machine M and an integer k, the security parameter, presented in unary.

Let us now make some simplifications that will expedite our exposition. Without loss of generality in our scenario, we assume that, when $(\overline{M},1^k)$ is the common input in a game network, all private inputs have the same length l and that $T(l)$, the running time of M on inputs of size l, is less than k.

Let $S \in PDA$. We say that S is a *Tm-game Solver for passive adversaries* if, for all Tm-game problems $(\overline{M},1^k)$ given as common input and for all (respective) private inputs $x_1,...,x_n$,

1) (*Agreement constraint*)

   At the end of each execution of S, for all machines i and j, i's private output equals j's private output.

2) (*Correctness constraint*)

   $OS_1(x_1,...,x_n,(\overline{M},1^k)) = M(x_1,...,x_n)$ and

3) (*Privacy constraint*)

   $\forall\ T \subset \{1,...,n\}$ and $\forall A \in PPT$, $\exists B \in PPT$ such that $\{A_k\}$ and $\{B_k\}$ are computationally undistinguishable RVs.

   Here

$$A_k = A(\ (\overline{M},1^k)\ ,\ HS(\ (\overline{M},1^k))\ ,\ HS_T(\ (\overline{M},1^k))\ )$$

and

$$B_k = B(\ (\overline{M},1^k)\ ,\ M(x_1,...,x_n)\ ,\ \{(i,x_i) : i \in T\}\ ).$$

Let us now interpret the above definition.

The agreement constraint

This constraint essentially says that all machines agree on a single, common string as the output of S.

The correctness constraint

This constraint ensures that the output of a game solver S coincides with the one of M. As M may be probabilistic, the equality of the correctness constraint must interpreted as equality between RVs.

The privacy constraint

Notice that passive adversaries appear in the above definition in an implicit way. Algorithm A can be thought as all the members of T being passive adversaries computing after an execution of S. In fact passive adversaries are obliged to send messages according to S and their private history, in an execution of S, is an explicit input to A. Let us stress that the private history of a machine i contains the name i, the private input x_i, and M's output as well. Thus the privacy constraint essentially says that whatever the passive adversaries may compute after executing S, they could also easily deduce from the desired M's output, y, and their own private inputs (which they are entitled to have!). In fact, if they are given y by running S, the passive adversaries will see, in addition to y, only the public history and their own private history. However, whatever they could efficiently compute with this additional input, they could also have computed without it. In other words, S keeps whatever privacy of the inputs of the good parties is not "betrayed" by the value y itself. For instance, if M computes the sum of the x_i's, then the privacy constraint will allow the adversarial players to compute (at the end of S) essentially only the sum of the inputs of the good parties. As for another example, if M is the identity function, then the privacy constraint holds vacuously. Same if the set T is the set of all players.

4. Hints on How to Play Tm-games With Passive Adversaries

At a first glance enforcing both correctness and privacy constraints of a Tm-game appears easy only for special cases of M, say the ones computing a constant function. None-the-less,

Theorem: If trapdoor functions exist, there exists a Tm-game solver for passive adversaries.

In this extended abstract we limit ourselves to give a few indications, in an informal manner, about the proof of the above theorem. Moreover, not to get into further complications, we do not let the set of adversarial machines to be chosen dynamically, during the execution of the protocol, but at its start. (We stress, though, that the adversarial set is still unknown to the good machines). This restriction will be removed in the final paper.

4.1 A New and General Oblivious Transfer Protocol

In [HR], Rabin proposes the beautiful notion of an *Oblivious Transfer* (OT). This is a probabilistic polynomial-time algorithm that allows A(lice), who knows the prime factorization of an integer n, to send it to B(ob), who knows just n, so that B will receive n's factorization with probability 1/2 and A does not know whether or not B received it. Clearly, Rabin's notion of an OT, supposes that factoring is computationally hard. Under this assumption, he proposed a protocol that, if A and B are allowed to be at most passive adversaries, correctly implements an OT. This protocol, however, may not work (i.e. no longer possesses a proof of correctness) if A and B are allowed to be malicious. Using the interactive proof-systems of [GMR], Fischer, Micali, Rackoff and Wittenberg [FMRW] found a protocol that correctly implements OT under the simple (and in this context minimal) assumption that factoring is hard. Rabin's OT has proved to be a very fruitful notion, as exemplified by various applications proposed by Blum [B].

A more general and useful notion of OT has been proposed by Even, Goldreich and Lempel [EGL], the *one-out-of-two* OT. In their framework, A has two messages m_0 and m_1. By using a cryptosystem E, she computes $\sigma_0 = E(m_0)$ and $\sigma_1 = E(m_1)$ and sends σ_1 and σ_2 to B. B chooses one of these encryption, σ_i. A one-out-of-two OT allows B to read the corresponding message m_i, while A will not know which message B has read (whenever m_0 and m_1 are different). This notion achieves the right level of generality and is crucial to what follows. Even, Goldreich and Lempel also proposed the first implementation of a one-out-of-two OT using public-key cryptosystems. Their protocol has the merit of having freed the implementation of an oblivious transfer from the algebraic setting to which it appeared to be confined. Their protocol, though, requires a quite strong set of assumptions even when the adversaries are only passive.

Below, we contribute a new protocol that correctly implements a one-out-of-two OT in presence of passive advarsaries. The existence of trap-door permutations suffices to prove the correctness of our protocol.

Trap-door and One-Way functions

A satisfactory definition of a trap-door permutation is given in [GoMiRi]. Here let us informally say that a family of trap-door permutations f fossesses the following properties:

* It is easy, given an integer k, to randomly select permutations f in the family which have k as their security parameter, together with some extra "trap-door" information allowing easy inversion of the permutations chosen.
* It is easy to randomly select a point in f's domain.
* It is hard to invert f without knowing f's trap-door on a random element in f's domain.

We can interpret the above by saying that a party A can randomly select a pair of permutations, (f, f^{-1}), inverses of each other. This will enable A to easily evaluate and invert f; if now A publicizes f and keeps secret f^{-1}, then inverting f will be hard for any other party. We may write f_k to emphasize that k is the security parameter of our permutation.

Trap-door permutations are a special case of one-way permutations. These are permutations enjoying the three properties above, except that we do not insist that the trap-door information exists.

Random Bits in One-Way Permutations

Our one-out-of-two OT protocol makes use of trap-door functions f hiding a random bit B_f. Here B_f is a polynomial-time computable Boolean function; the word "bit" is appropriate as B_f evaluates to 1 for half of the x's in f's domain.

We say that $\{B_f\}$ is a *random bit* in a family $\{f\}$ of trap-door permutations if $\forall$ predicting algorithm Alg that, on inputs $f = f_k$ and $f(x)$, outputs, in $T(k)$ steps, a guess for $B_f((x))$ that is correct with probability ϵ, $\exists\, Alg'$ that, on inputs f and $f(x)$, outputs x in poly($T(k), \epsilon^{-1}$) expected time.

Thus, being f trap-door, no probabilistic, polynomial-time algorithm given $f_k(x)$, can correctly predict $B_{f_k}(x)$ with probability $> 1/2 + 1/poly(k)$. We might as well flip a coin. Thus, for a one-way permutation f, given $f(x)$ the value of $B_f(x)$ cannot be guessed in polynomial time essentially better than at random.

The notion of a random bit in a one-way permutation was introduced by Blum and Micali [BM] who showed a random bit in the Discrete Logarithm Problem, a well known candidate one-way permutation. Chor and Goldreich show random bits in the RSA function. Do all one-way functions have a random bit? We do not know the answer to this question, but Yao [Y] has shown the next best thing. Namely, that given a one-way (trap-door) permutation f, one can construct a one-way (trap-door) permutation F with a random bit B_F (for a detailed proof of this theorem see [BH]). Levin [L] has actually proved a more general version of this theorem.

Our Protocol

Without loss of generality, we assume that the two messages in the one-out-of-two OT both consist of a single bit.

In our protocol, both A and $B \in PA$. A's inputs are a pair of bits (b_0, b_1) and their corresponding pair of encryptions $(E(b_0), E(b_1))$ where E is a probabilistic encryption algorithm [GM]. The pair $(E(b_0), E(b_1))$ is also an input to B who has an additional private input bit α. It is desired that even if some party is a passive adversary the following two properties hold:

i) B will read the bit b_α, but will not be able to predict the other bit, $b_{\bar{\alpha}}$, essentially better than at random.

ii) A cannot predict α essentially better than at random.

We achieve this by means of the following protocol.

Step 1

A randomly selects (f, f^{-1}), a trap-door function of size k (having a random bit B_f) together with its inverse. She keeps f^{-1} secret and sends f to B.

Step 2

B randomly selects x_0 and x_1 in f's domain and computes $z = f(x_0)$ and sends A the pair

$$(u,v) = \begin{cases} (f(x_0), x_1) & \text{if } \alpha = 0 \\ (x_0, f(x_1)) & \text{if } \alpha\ 1 \end{cases}$$

Step 3

A computes $(c_0, c_1) = (B_f(f^{-1}(u)), B_f(f^{-1}(v)))$. She sets $d_0 = b_0 \; xor \; c_0$ and $d_1 = b_1 \; xor \; c_1$ and sends (d_0, d_1) to B.

Step 4

B computes $b_\alpha = d_\alpha \; xor \; B_f(x_\alpha)$.

First notice that $A, B \in PA$ and that B correctly reads b_α. Property i) is satisfied as B only sees $b_{\bar{\alpha}}$ exclusived-ored with a bit essentially 50-50 unpredictable to him. Thus he cannot correctly guess $b_{\bar{\alpha}}$ essentially better than at random. Let us now show that ii) holds. As f is a permutation, randomly selecting x in f's domain and computing $f(x)$ yealds a randomly selected element in f's domain. Thus (u,v) is a pair of randomly selected elements in f's domain both if $\alpha=0$ or $\alpha=1$. As (u,v) is the only message B sends A, not even with infinite computing power A will find out whether B has read b_0 or b_1.

Notice that the protocol makes use that the adversaries are at most passive in a crucial way. Should in fact B send $(u,v) = (f(x_0), f(x_1))$ in step 2, he will easily read both bits. Thus, we will make use of additional ideas to handle malicious adversaries.

Notice also that we never made use of the encryptions $E(b_0)$ and $E(b_1)$. b_o and b_1 could have been bits in "A's mind." We have added these encryptions for uniformity with the next protocol in which the two messages must appear encrypted. Another reason is that, when we will handle malicious adversaries, we will need these encryptions to define the problem.

It is easy to see that, having solved the single-bit messages case, we have also solved the case of arbitrary messages m_0 and m_1 of equal, known length l. In fact, we can repeat the above protocol l times, so that, if α is 0 (1), B is required at the ith time to learn the ith bit of m_0 (m_1).

4.2 Strengthening Yao's Combined Oblivious Transfer

In [Y2], Yao presented a protocol that we call *combined oblivious transfer* (COT). The protocol involves two parties A and B, respectively owning private inputs a and b and any chosen function g. It possesses the following property: upon termination, A computes $g(a,b)$, while B has no idea of what A has computed. If we think of a ad b as secrets, B appears to obliviously transfering a prescribed combination of his and A's secret to A. Yao implemented COT based on the assumption that factoring is hard, (which yelds, as shown by Blum [B]) a particular trap-door permutation. We strengthen his result by showing that COT can be correctly implemented based on any trap-door permutation. We do this by using the one-out-of-two OT of section 4.1 in Yao's scheme. Let us consider first the case where a and b are bits and g is the Boolean *AND*. Consider figure 1.

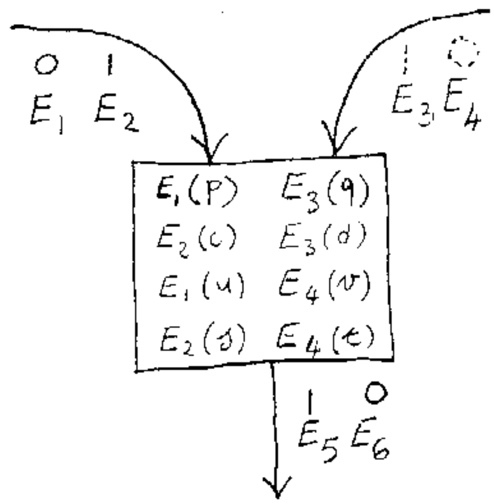

figure 1: A COT *AND*-gate

Here $E_1, \ldots, E_6$ are independently selected encryption algorithms, respectively having decryption keys $D_1, \ldots, D_6$. E_1 and E_2 label the first input-wire, E_3 and E_4 the second input-wire, and E_5 and E_6 the output-wire. Each row in the gate is formed by the encryption of two strings. m and n are two randomly selected strings whose bit-by-bit exclusive-or equals D_5. p and q are two randomly selected strings whose xor equals D_6; so are s and t; so are u and v. The 4 rows have been put in the gate in random order. E_1, E_2 and E_5, E_6 are publically labelled by complementary bits. E_3 and E_4 are each secretely labelled by a bit; more precisely, E_3 is SECRETELY labelled 0 with probability 1/2 and E_4 is labelled with the complement of E_3's bit. (This secrecy is pictorially indicated by drawing E_3 and E_4's bits by a dotted line.) Define the value of a wire to be 0 (1) if one ONLY possesses the decoding algorithm of encryption algorithm labelled 0 (1). Then figure 1 is a or-gate. For instance, assume that both input-wires have value 0. That is, one possesses only D_1 and D_4. Then one is able to decrypt both entries only in the third row. By taking the *xor* of u and v, one easily obtains D_6, but has no idea what D_5 may be. Thus the output-wire has value 0 $= AND(0,0)$.

To COTransfer $AND(a,b)$, B generates a COT *AND*-gate like in figure 1, keeping for himself all decoding algorithms and all the strings in the rows. Then, he gives A the decoding algorithm of the second input-wire that corresponds to the value of b, his own input. Notice that as the association between E_3, E_4 and 0,1 is secret (and E_1, E_2, E_3, E_4 enter symmetrically in the gate rows), this will not betray b at all. Now A will get either D_1 or D_2, according to the value of a, by means of our one-out-of-two OT. Thus, B will not know which algorithm she got. At this point A can easily compute the value of the output-wire. Thus she will be the only one to know $AND(a,b)$.

It is trivial to build a COT *NOT*-gate. Notice that B may also keep secret the corresponding between 0,1 and E_5, E_6.

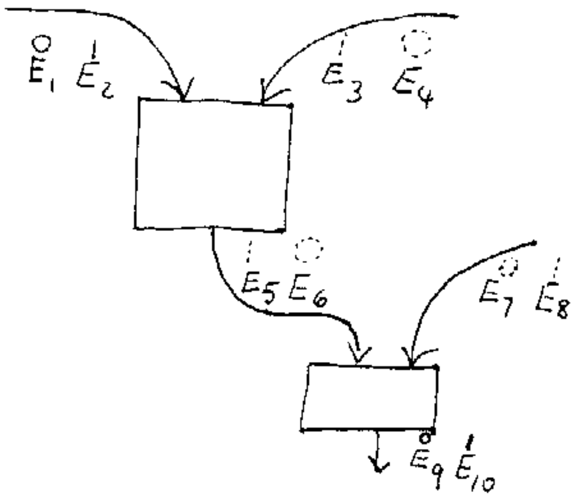

fig. 2

This allows the out-put wire to become an input-wire of another gate. If the encryption algorithms of this second gate are publically labelled 0/1 (see fig. 2), we see that A may evaluate any 2-gates function on her and B's inputs, without knowing intermediate results. Better said, B can "COTransfer" the value of any 2-gates function. By cascading this way COT *AND*-gates and COT *NOT*-gates (which are trivial to design), we can see that B can COTransfer the value of any function, provided that there is an upper bound to the length of A's and B's inputs, (else, the length of the inputs will be betrayed).

4.3 The Tm-game Solver for passive adversaries

Recall that a Tm-game solver wants to compute $M(x_1,\dots,x_n)$ while respecting the privacy constraint. We want to use COT as a subroutine to construct a Tm-solver. This does not appear to be straightforward. For instance, if two parties i and j use COT so that i will compute $g(x_i,x_j)$ for some function g, this would already be a violation of the privacy constraint. Recall also that the Tm-game solver has to be polynomial not only in M's running time, but also in n, the number of players.

We find a way out by making special use of a lemma of Barrington's [Ba] that simulates computation by composing permutations in S_5, the symmetric group on 5 elements. The general picture is the following. First transform the Turing machine M of a Tm-game to an equivalent circuit C in a standard way. The Boolean inputs of C will be $b_1^1,\dots,b_l^1,\dots,b_1^n,\dots,b_l^n$, the bits of the n, l-bit long, inputs of our parties. This circuit C is then transformed to straight-line program as in [Ba]. This straight-line program is essentially as long as C is big. In it,

* 0,1 are encoded by two (specially selected) 5-permutations
* the variables range in S_5 and
* each instruction consists of multiplying (composing) two 5-permutations σ and τ, whre σ (τ) is either a constant, or a variable, or the inverse (in S_5) of a variable.

At the start, each party takes each of his private bits and encodes it by a 5-permutation σ as in [Ba]. Then he divides σ. That is, he selects at random n-1 5-permutations $\sigma_1,\dots,\sigma_{n-1}$ and gives the pair (i,σ_i) to party i (possibly himself). He then sets

$\sigma_n = (\sigma_1 \cdots \sigma_{n-1})^{-1} \cdot \sigma$ and gives (n, σ_n) to party n. Now, inductively, assume that each variable is divided among the parties. That is, for each variable σ, each player i possesses an index permutation pair (x, σ_x) so that $\prod_{x=1}^{x} \sigma_x = \sigma$ and, given only $n-1$ pieces, σ cannot be guessed better than at random. We now want to show that each instruction can be performed (i.e., each party can compute his individual piece of the result) respecting the privacy constraint. There are essentially 3 cases.

Case 1: The instruction is of the form $\sigma \cdot c$, where σ is a variable and c a constant. By induction, each party has a piece of the form (x, σ_x). Then the party owning the piece (n, σ_n) sets his new piece to be $(n, \sigma_n \cdot c)$ and all each party leaves his piece untouched. It is immediately checked that the ordered product of the new pieces is $\sigma \cdot c$ and that privacy has been preserved against $n-1$ passive adversaries.

Case 2: The instruction is of the form $\sigma^{-1} \cdot c$ where, again, σ is a variable and c a constant. It will be enough to show how to compute pieces for σ^{-1} respecting the privacy constraint. To do this, if a party has a piece (x, σ_x), he sets his new piece to be $(n - x + 1, \sigma_x^{-1})$.

Case 3: The instruction is of the form $\sigma \cdot \tau$, where both σ and τ are variables. Then $\sigma \cdot \tau = \sigma_1 \cdots \sigma_n \cdot \tau_1 \cdots \tau_n$, and assume for simplicity that party i possesses piece σ_i and τ_i. Unfortunately, party 1 cannot compute his piece of $\sigma \cdot \tau$ by multiplying his own two pieces. In fact, they are n positions apart in the product and S_5 is not commutative (a fact crucial in Barrington's argument). The idea will then consist of making "partial progress". That is, moving party 1's pieces closer together by "swapping" σ_n and τ_1. This can be correctly accomplished by giving party 1 a piece τ_1' and party n a piece σ_n' so that $\tau_1' \cdot \sigma_n' = \sigma_n \cdot \tau_1$. This way the product of the new (and newly ordered pieces) would remain $\sigma \cdot \tau$. One way of doing this would be of having party 1 and party n tell each other σ_n and τ_1. However this would violate the privacy constraint with respect to a set of $n-1$ passive adversaries. Instead, we use COT in the following way. Party n randomly selects a 5-permutation ρ. Consider now the function g such that, for 5-permutations x,y, and z, $g(x,(y,z)) = w$ where $w \cdot z = y \cdot x$. Let now party 1 (with the role of A and input $a = \tau_1$) and party n (with the role of B and input $b = (\sigma_n, \rho)$) play COT with function g. Set $\tau_1' = g(a,b)$ and $\sigma_n = \rho$.

Then we have made the desired partial progress. In fact, not only the product of the new pieces is unaltered, but we have also respected the privacy constraint. Informally, party n's new piece is a random 5-permutation selected by party n himself and thus cannot give him any information neither about party 1's old piece nor the new one; moreover the transference of $g(a,b)$ is oblivious and thus cannot give party n any knowledge either. On the other side, party 1 is dealt a new piece $g(\tau_1, (\sigma_n, \rho))$ and he knows τ_1. However, as for all x and y, $g(x,(y,\cdot))$ is injective on S_5, and ρ has been randomly and secretely selected by party n, also party 1 does not get any knowledge that he did not possess before! Notice also that during this "swap" we did not create any other pieces. Thus after n "swaps" the only two pieces of party 1 will be in the first two positions in the product and he can thus multiply them together. This product will be party 1's piece for the variable $\sigma \cdot \tau$. It should be verified that the entire walk of party 1 τ-piece towards the left preserves correctness and does not violate the privacy constraint. Essentially because a new random piece is created at each step. This way, after $O(n^2)$ "swaps", and in polynomial time, all parties receive their piece of $\sigma \cdot \tau$.

At the end of the straight-line program, for each output variable γ, each party publicizes his own piece (x, γ_x), the ordered product of these pieces is computed and the output bit recovered so to satisfy both the correctness and the privacy constraint. (A more formal argument will be given in the final paper.)

5. Malicious Adversaries

The complexity of our Tm-game solver greatly increases when up to half of the players is allowed to be malicious and can more powerfully collaborate to try to disrupt the correctness and the privacy constraints. We use essentially all the cryptographic tools developed in the last ten years in the (correct) hope that they would make possible protocol design. Also, the proof of its correctness is rather delicate and unsuitable for an abstract. We will give it in the final paper. Here we only indicate what making playable a Tm-game with malicious advesaries may *mean* and which general ideas are involved in our solution.

As in this case some of the parties may not follow their prescribed programs at all, it is necessary to clarify what a private input is. After all, what stops someone from pretending that his private input is different from what it actually is? To avoid this, we assume that the parties have established their private inputs by announcing correct encodings of them. Their inputs are *by definition* the unique decryption of their respective encodings. Moreover, it shoud be clear that seeking a solution to a Tm-game problem makes sense only if the parties are "willing to play". If, say, one of them "commits suicide", carrying with himself what his private input was, there is very little one can do besides investing exponential time and break his encryption. However we can, loosely speaking, prove that

> Given n players willing to play, less than half of which malicious, all Tm-games are playable.

The above term "willing to play", indicates a technical condition rather than a psychological one. Namely, having successfully completed the *engagement protocol*. After completing this protocol, all players can be forced to play any desired game. The engagement protocol consists of two phases.

1) For each player i, a protocol is performed at the end of which no minority of the players can even predict a bit of i's private input with chances essentially better than $1/2$. However, *it is guaranteed* that any subset of cardinality $>n/2$ can, without the cooperation or even against the actions of other players, easily compute i's private input.

2) The community deals to each player a sequence of encrypted "random" bits so that a) the recipient knows their decryption, b) they appear unpredictable to any minority of the players, but c) they are easily computable by any majority of the players.

We stress that while no one can be forced to complete the engagement protocol (so to become "willing to play"), no one can decide not to complete it because he received a better idea of what the result of the subsequent game may be. Completing the engagement protocol will not give any player (or any small enough group of players) any knowledge about the others' private inputs.

Phase 1 of the engagement protocol consists of a verifiable secret sharing in the sense of Awerbuch, Chor, Goldwasser and Micali [CGMA]. However, we contribute a new protocol both tolerating up to $n/2$ malicious adversaries and using any trap-door function whatsoever. Phase 2 of the engagement protocol is the multy-party version of Blum's coin flipping by telephone. Despite the (deceivingly) similarity with the verifiable secret sharing of phase 1, to implement phase 2 we must make use of a yet unpublished theorem (and algorithm) of ACGM.

We now give a bird's eye view of how to make any Tm-game g playable despite malicious adversaries. On input $M, 1^k$, we first run the engagement protocol, then the passive-adversary playable version of the Tm-game. Here we require all parties to use, as their private inputs, the strings they shared in phase 1 of the engagement protocol and, as a source of randomness, the encrypted random bits each was dealt in phase 2. The key point is that, now, no malicious adversary can deviate from his prescribed program, and thus he becomes a simple passive adversary. In fact, he is required to prove, in zero-knowledge (in the sense of Goldwasser, Micali and Rackoff [GoMiRa]), that each message he sends is what he should have sent being honest, given his private input, his random choices and the messages he received so far. (Here, an essential tool is our recent result that all NP languages possess zero-knowledge proofs [GMW].) If a malicious party, frustrated at not being able to send messages according to a different program, decides to stop, his input and random bits will be reconstructed by the community who will compute his messages when necessary, without skewing the probability distribution of the final outcome.

We would like to stress our new use of NP-completeness. From being our most effective way to prove lower-bounds, it now becomes our most effective tool to construct correct protocols.

6. General Games

Many actions in life, like negotiating a contract, casting a vote in a ballot, playing cards, bargaining in the market, submitting a STOC abstract, driving a car and simply living, may be viewed as participating with others in a game with payoffs/penalties associated with its results. This is not only true for individuals, but also for companies, governments, armies etc. that are engaged in financial, political and physical struggles. Despite the diversity of these games, all of them can be described in the elegant mathematical framework laid out by Von Neumann and Morgenstern earlier in this century. *Game theory*, however, exhibits a "gap", in that it neglected to study whether, or how, or under which conditions, games can be *imple-*

mented. That is, it never addressed the question of whether, given the description of a game, a method existes for physically or mentally playing it. We do fill this gap by showing that, in a complexity theoretic sense, all games can be played.

In this extended abstract we will only informally clarify what and how this is. We start by briefly recalling the ingredients used by game theory to model a n-players game with *incomplete information.*

6.1 Games

Essentially, a game consists of a set S of possible *states*, representing all possible instantaneous descriptions of the game, a set M of possible *moves*, describing all possible ways to change the current state of the game, a set $\{K_1,K_2,...,K_n\}$ of *knowledge functions*, where $K_i(\sigma)$ represents the partial information about state σ possessed by player i, and a function p, the *payoff function*, that, evaluated on the final state, tells the outcome of the game. Without loss of generality, the players make moves in cyclic order and the set of possible moves in any state are the same for all states. Also, WLOG, the game goes on for a fixed number of moves m. With little restriction we do assume that the players make use of recursive strategies for selecting their moves. (The classical model does not rule out selecting moves according to an infinite table.)

Let us now see how a game evolves using, in parenthesis, poker as an example. The game starts by having "NATURE" select an initial state σ_1. (For poker, σ_1 is a randomly selected permutation of the 52 cards; the first $5n$ cards of the permutation representing the players initial hands and the remaining ones the deck.) Player 1 moves first. He does not know σ_1 -nor does anybody else-, he only knows $K_1(\sigma_1)$, his own hand: the first 5 elements of permutation σ_1). Based solely on $K_1(\sigma_1)$, he will select a move μ (e.g. he changes 3 of his cards with the first 3 cards of the deck). This move automatically updates the -unknown!- current state to σ_2. (The new state consists of the cards currently possessed by each player, the sequence of cards in the deck and which cards were discarded by player 1. $K_1(\sigma_2)$ consists of the new hand of player 1 and the cards he just discarded.) Now it is the turn of player 2. He also does not know the current state σ_2, he only knows $K_2(\sigma_2)$. Based solely on this information, he selects his move, which updates the current state, and so on. After the prescribed number of moves, the payoff function p is evaluated at the final state to compute the result of the game. (In poker the result consists of who has won, how much he has won and how much everyone else has individually lost.)

Note that a Tm-game is indeed a game in which the initial state is empty and each player moves only once. State σ_i consists of the sequence of the first i moves. Each player has no knowledge about the current state and chooses his move to be the string x_i, his own private input. The payoff function M is then run on σ_n. (Having probabilistic machines running on the final state, rather than deterministic ones, is a quite natural generalization.)

From this brief description it is immediately apparent that, by properly selecting the knowledge functions, one can enforce any desired "privacy" constraints in a game.

6.2 Playable Games

Game theory, besides an elegant formulation, also suggests to the players strategies satisfying some desired property (e.g. optimality). That is, game theory's primary concern is how TO SELECT MOVES WELL. However, and ironically!, it never addressed the question of how TO PLAY WELL. For a general n-player game, all we can say is that we need $n+1$ parties to properly play it; the extra party being the "trusted party". The trusted party communicates privately with all players. At step t, he knows the current state σ_t of the game. He kindly computes $\alpha=K_{t \bmod n}(\sigma_t)$, communicates α to player t mod n, receives from him a move μ, secretely computes the new state $S_{t+1}=\mu(S_t)$, and so on. At the end, the trusted party will evaluate the payoff function on the final state and declare the outcome of the game. Clearly, playing with the trusted party achieves exactly the privacy constraints of the game description, and at the end each player will get the correct outcome.

Now, the fact that, in general, a n-person game requires $n+1$ people to be played, not only is grotesque, but it also diminuishes the otherwise wide applicability of game theory! In fact, in real life situations, we may simply not have any trusted parties, whether men or public computers. Recently, complaints have been raised about finantial transactions in the stock market. The complaints were about the fact that some parties were enjoying knowledge that was considered "extra" before choosing their move, i.e. before buying stocks. Just another game, the stock market, but one in which you may desire trusting no one!

We are thus led to consider the notion of a *(purely) playable* game. This is a n-person game that can be implemented by the n players without invoking any trusted parties. In general, however, given the specification of a game with complicated knowledge functions, it is not at all easy to decide whether it is playable in some meaningful way. Here, among the "meaningful ways", we also include non-mathematical methods. Yet, the decision may still not be easy.

Poker, for instance, has simple enough knowledge functions (i.e. privacy constraints) that makes it playable in a "physical" way. In it we use cards with equal "back" and "opaque", tables whose top does not reflect light too much, we shuffle the deck "a lot", and we hand cards "facing down". All this is satisfactory as in our physical model (world) we only see along straight lines. However, assume we define NEWPOKER as follows. A player may select his move not only based on his own hand, but also on the knowledge of whether, combining the current hands of all players, one may form a royal flush. NEWPOKER is certainly a game in the Von Neumann's framework but it is no longer apparent whether any physical realization of the game exists, particularly if some of the players may be cheaters.

This is what we perceive lacking in game theory: the attention to the notion of playability. At this point a variety of good questions naturally arises:

> *Is there a model (physical or mathematical) which makes all games playable?*

Or at least,

> *Does every game have a model in which it is playable?*

And if not,

> *Should we restrict our attention to the class of playable games?*

We show that the first question can be affirmatively answered in a computational complexity model.

6.3 A General Result

Theorem: If any trap-door function exists, any game is playable if more than half of the players are honest.

Essentially our result consists of a protocol for simulating the trusted party of an ideal game. That is, if more than half of the players follow our protocol, whatever a player (or a set of players of size less than $n/2$) knows at any step of the game, he would have also known in an ideal execution of the game with a trusted party. In our context the knowledge constraints are satisfied in a computational complexity sense. Namely, any player (or collection of dishonest players) in order tc compute anything more than his due share of the current state, should perform an exponential-time computation. Unfortunaly, we cannot, in this extended abstract, elaborate on the relationship between general games and Tm-games, nor how to pass from solving the latter ones to solve the general case. We'll do this in the final paper.

6.4 A Completeness Theorem For Fault-Tolerant Computation

Our main theorem has direct impact to the field of fault-tolerant computation. This is so as protocols, when properly formalized (which we will do in the final paper), are games with partial information. Thus, as long as the majority of the players is honest, all protocols may be correctly played. Actually, slightly more strongly, the correct way to play a game can be found in a uniform manner. Namely, we exhibit a specific, efficient algorithm that, on input a protocol problem, outputs an efficient, distributed protocol for solving it.

It should be noticed that, before this, only an handful of multi-party protocol problems were given a satisfactory solution (e.g. collective coin flipping and poker over the telephone, secret exchange, voting, and a few others). Moreover the security of some of these solutions crucially depended on the "trap-doorness" of specific functions satisfying some additional, convenient property (e.g. multiplicativity). By contrast, our completeness theorem is proved based on *any* trap-door function (multiplicative or not, associative or not, etc.). That is, we prove that, if public-key cryptography is possible at all, then all protocols problems are (automatically!) solvable if more than half of the players are honest.

7. Recent Developments

Recently, Haber and Micali found a Tm-game solver that is algorithmically much simpler (for instance it does not use Barrington's straight-line programs) but more difficult to prove correct. Also, Goldreich and Vainish found a simpler solution based on a specific assumption, the computational difficulty of quadratic residuosity.

8. Acknowledgements

We are very grateful to Shimon Even, Dick Karp, Mike Merritt, Albert Meyer, Yoram Moses for having doubted the generality of some of our intermediate solutions and having encouraged us to reach the right level of generality. In particular, Albert Meyer contributed the beautiful notion of a Turing-machine game, and Dick Karp steered us toward games with incomplete information as the best avenue to our completeness theorem for protocols.

We also would like to thank Benny Chor, Mike Fischer and Shafi Goldwasser for helpful discussions concerning the issues of this paper.

9. References

[Ba] D. Barrington , *Bounded-Width Branching Programs Recognize Exactly Those Languages in* NC^1, Proc. 18th STOC, 1986 pp 1-5

[Bl] M. Blum, *Coin Flipping by Telephone,* IEEE COMPCON 1982, pp. 133-137.

[BH] R. Boppana and R. Hirschfeld, *Pseudo-Random Generators and Complexity Classes,* To appear in Randomness and Computation, 5th volume of Advances in Computing Research, ed. S. Micali

[BM] M. Blum and S. Micali, *How To Generate Sequences Of Cryptographically Strong Pseudo-Random Bits,* SIAM J. on Computing, Vol. 13, Nov 1984, pp. 850-864

[CG] B. Chor and O. Goldreich, *RSA/Rabin Bits Are 1/2 + 1/poly(log N) Secure,* To appear SIAM J. on Computing.

Earlier version in Proc. FOCS 1984, pp. 449-463

[CGMA]
B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch, *Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults'*, Proc. 26th FOCS, 1985, pp. 383-395

[EGL] S. Even, O. Goldreich, and A. Lempel, *A Randomized Protocol for Signing Contracts,* CACM, vol. 28, No. 6, 1985, pp. 637-647

[FMRW]
M.Fischer, S. Micali, C. Rackoff and D. Witenberg, *A Secure Protocol for the Oblivious Transfer,* In preperation 1986.

[GM] S. Goldwasser, and S. Micali, *Probabilistic Encryption,* JCSS Vol. 28, No. 2, April 1984.

An earlier version (containing other results) was titled *Probabilistic Encryption and How to Play Mental Poker Hideing All Partial Information,*

[GMR]
S. Goldwasser, S. Micali and C. Rackoff, *The Knowledge Complexity of Interactive Proof-Systems,* To appear SIAM J. on Computing (manuscript available from authors).

Earlier version in Proc. 17th Annual ACM Symp. on Theory of Computing, pp 291-304.

[GoMiRi]
S. Goldwasser, S. Micali, and R. Rivest, *A Digital Signature Scheme Secure Against Adaptive, Chosen Cyphertext Attack* To appear in SIAM J. on Computing (available from authors)

Earlier version, titled "*A Paradoxical Solution to The Signature Problem,* in Proc. 25th FOCS, 1984, pp. 441-448

[GMW]
O. Goldreich, S. Micali and A. Wigderson, *Proofs that Yield Nothing but their Validity and a Methodology of Cryptographic Design,* Proc. of FOCS 1986.

[HR] J. Halpern and M.O. Rabin, *A Logic to reason about likehood,* Proc. of 15th STOC, 1983.

[L] L. Leonid, *One-Way Functions and Pseud0-Random Generators,* Proc. 17th STOC, 1985, pp. 363-365

[Y] A.Yao, *Theory and Application of Trapdoor Functions,* Proc. of 23rd FOCS, IEEE, Nov., 1982, pp. 80-91.

[Y2]
A.Yao, *How to Generate and Exchange Secrets,* Proc. 27th STOC, 1986, pp. 162-167