How to Solve any Protocol Problem

by Oded Goldreich, Silvio Micali and Avi Wigderson.

Remark: A version identical in contents, but very different in form has appeared in the proceedings of the *19th STOC*, pp. 218–229, 1987. This version was produced, from an old draft (dating to 1986), by automatic convertion of a old `troff` file into a `latex` file. The output was not carefully checked; hopefully, it does not contain too many typos.

ABSTRACT:

This extended abstract present a general theorem in the field of fault tolerant distributed computing. Following is a simplified description of a special case of this theorem. Loosely speaking, a *protocol problem* is a multi-argument function f and its *solution* is a multi-party fault-tolerant protocol having the following two properties:
(1) Correctness: The protocol allows each party to obtain the value of the function on arguments scattered among all the parties. Namely, the local input of party P_i is x_i, and his local output (obtained by execution of the protocol) is $f(x_1, x_2, ..., x_n)$.
(2) Privacy: Whatever a party (P_i) can efficiently compute after participating in the protocol, he can also efficiently compute from his local input (x_i) and his local output (i.e. $f(x_1, x_2, ..., x_n)$).

In other words, participating in the protocol is equivalent to getting $f(x_1, x_2, ..., x_n)$ from a trusted oracle. For example, if $f(x_1, x_2, ..., x_n) = \sum_{i=1}^{n} x_i$ then a solution is a protocol at the end of which each party gets the sum of the x_i's without gaining any additional knowledge as to how the residual sum is partitioned among his counterparts.

Assuming the existence of secure encryption functions, it will be shown that every protocol problem has a solution with complexity polynomial in the complexity of the problem. Furthermore, we present an efficient algorithm that, on input a Turing machine description of a function, outputs an efficient solution for this problem.

1. Introduction

The general goal of distributed computing is to develop protocols for computing (distributively) functions of local inputs scattered among the processors. If all processors follow their predetermined programs then the existence of such protocols follows immediately from the specification of the corresponding function, and the only challenge is in improving the (message and time) complexities of these protocols. However, the situation is much more complex if some of the processors may deviate from their predetermined program in certain ways. A natural model of such misbehaviour allows *faulty* processors to deviate from their predetemined program in an arbitrary, but polynomial-time manner. When faults are present, it is no longer clear if there exist protocols that are *correct* in the sense that they terminate with each of the non-faulty processors having the value of the function. Furthermore, it is not clear whether correct protocols can offer the maximum possible *privacy* of local inputs allowed by the function (i.e. whether it is possible to restrict what the faulty processors learn about the values of the non-faulty processors, by executing the protocol, to the value of the function).

The main result of this paper is an absulote affermative resolution of the above problem. If secure trapdoor encryption functions exist and as long as more than half of the processors remain non-faulty, *every* function has a *correct* fault-tolerant protocol that offers the maximum degree of *privacy*. The complexity of the protocol is polynomial in the time-complexity of the function. Furthermore, there exist a polynomial-time algorithm that on input a Turing-Machine specification of the function outputs such a protocol.

1.1 Formal Setting

For the sake of simplicity, we consider the special case in which the task is to compute one common output as a function of all local inputs. Let n be an integer, $N = \{1, 2, ..., n\}$, f be an n-ary function computable in polynomial-time, and P be a protocol for n parties the programs of which require only probabilistic polynomial-time computations. In the sequel, we will consider executions of P during which some of the processors become *faulty*. Faulty processors may deviate from P in an arbitrary, but (probabilistic) polynomial-time manner. Furthermore, faulty processors may collaborate with each other. We assume that the set of faulty processors is fixed beforehand (though it is not "known" to the non-faulty processors).

Definition (- sketch): We say that P is a *correct fault-tolerant* protocol for computing f, if for every set $T \subseteq N$ of less than $n/2$ faulty processors either all non-faulty processors locally output $f(x_1, x_2, ..., x_n)$ or they all output a special symbol denoted $\perp$. In case the non-faulty processors output $\perp$ the following holds: there exists a probabilistic polynomial-time algorithm that on input $\{x_i : i \in T\}$ outputs a probability distribution which is polynomially-indistinguishable from the output of the faulty processors in such executions.

Intuitively, P is a correct fault-tolerant protocol for computing f if the (non-faulty) majority can obtain $f(x_1, ..., x_n)$, except for cases in which a minority T based **only** on the values of its local inputs ($\{x_i : i \in T\}$) descides not to take any part in the execution. In this latter case, the non-faulty majority will detect the existence of such a minority (and its identity!). We believe that this notion of "correctness" is the strongest possible one.

Definition (- sketch): We say that the protocol P *offers the maximum degree of privacy* allowed by f, if for every set $T \subseteq N$ of less than $n/2$ faulty processors there exist a probabilistic polynomial-time algorithm such that on input $f(x_1, x_2, ..., x_n)$ and $\{x_i : i \in T\}$ outputs a probability distribution which is polynomially-indistinguishable from the probability distribution consisting of concatenating the outputs of the faulty processors after participating in the protocol P.

Intuitively, P offers the maximum degree of privacy, if whatever a (faulty) minority can compute after participating in an execution of P it could have computed from its local inputs and the value $f(x_1, ..., x_n)$.

1.2 An Example

Before going any further let us provide an example to the notions presented above. Consider the *sum* function

$$f(x_1, ..., x_n) = \sum_{i=1}^{n} x_i$$

A correct fault-tolerant protocol for computing f will not allow faulty parties to "tailor" the *sum* to their liking by "changing their inputs" after seeing the accumalative *sum*. Such parties have only the choice to either participat in the protocol or not participat and be detected as faulty parties. This choice has to be made without any knowledge of the local inputs of non-faulty parties.

Using the notion of verifiable secret sharing [CGMA], it is easy to present a correct fault-tolerant protocol for computing the *sum* function. Let each party verifiably share his input with all other parties, and next (if this phase is completed to the satisfaction of all parties) let each party broadcast his input. The inputs of parties which did not broacast properly are revealed by the honest share holders. However, this protocol does not offer any privacy.

A correct fault-tolerant protocol which offers the maximum degree of privacy allowed by the *sum* function was recently presented by Cohen [Coh]. In the case of the *sum* function "maximum privacy" means that all that a coalition T of faulty processors can efficiently compute after participating in the protocol, can be efficiently computed from their local inputs ($\{x_i : i \in T\}$) and the sum of all local inputs (i.e. $\sum_{i \in N} x_i$). Equivalently, all they learnt about the local inputs of the other processors is their sum $\sum_{i \in N} x_i$, and this of course can not be avoided.

The ideas suggested by Cohen [Coh] do not extend to any other function (except *multiplication*), since they heavily rely on the homomorphism of the secret sharing scheme with respect to the function f. To the best of our knowledge, no other non-trivial protocols which are correct and offer maximum privacy were know until now.

1.3 Our Result

In this paper we show that *all* polynomially computable functionsd have correct polynomial-time fault-tolerant protocols which offer the maximum possible privacy. Furthermore, these protocols can be efficiently generated from the Turing-Machine description of the function. Namely,

Main Theorem: *Assume that there exists a secure public-key encryption scheme. Then there exists a polynomial-time algorithm that on input a Turing machine program for polynomially-computable function f, outputs a correct (polynomial-time) fault-tolerant protocol for computing f in a manner offering the maximum degree of privacy allowed by f.*

The result extends to the generation of probability distributions (i.e. the input may be a probabilistic polynomial-time program), to the computation of different values by each party, and to playing any recursive partial information game. For more details see section 4.

1.4 The Construction

The proof of the Main Theorem is constructive. The construction is in two steps. First, we show how, given a Turing-Machine specification of the function f, one can efficiently generate a protocol for *semi-honest* parties that offers the maximum degree of privacy allowed by f. The semi-honest parties may deviate in their internal computation from the protocol, but the messages they send are in accordance with the protocol. Next, we show how to efficiently compile protocols for semi-honest parties into fault-tolerant protocols, in a manner which preserves both correctness and privacy (as long as the number of faulty processors is smaller than $n/2$).

The core of the first step of our construction is a method for computing through a Boolean circuit, while manipulating the intermediate values distributively in a secret sharing manner. An important ingredient in this method is a generalization of a Theorem due to Yao [Y2]. This generalization states that, for every polynomial-time computable function f, there exist a two-party cryptographic protocol for correctly computing f in a manner which offers the maximum degree of privacy allowed by f, as long as the parties are semi-honest.

The second step of our construction makes primary use of the result that all NP languages have zero-knowledge interactive proofs and that the prover in these interactions may be a probabilistic polynomial-time machine which gets a "witness for membership" as an auxiliary input [GMW]. Other essential ingredients are the notions of verifiable secret sharing and simultaneous broadcast proposed and first implemented by Chor, Goldwasser, Micali and Awerbuch [CGMA].

2. How to Generate Maximum Privacy Protocols for Semi-Honest Parties

Given a Turing Machine program and its inputs, we can easily constract an functionally equivalent polynomial-size Boolean circuit. Without loss of generality, we assume that this circuit has maximum fanin 2, and contains only AND and NOT gates.

Our original construction made use of Barrington's result [Bar] that Boolean circuits can be simulated by straightline programs in permutation groups. The construction was carried out on such straightline programs. Instead, using ideas of Stuart Haber, we present here a simpler construction which views Boolean circuits as straightline programs over $GF(2)$. An AND gate is simulated by multiplication, while a NOT gate is simulated by adding the constant 1. Throughout the rest of this section arithmetic will be in $GF(2)$.

We now present a protocol by which n semi-honest parties can evaluate a straightline program over $GF(2)$ with each of the inputs known only to one party. The protocol will offer the maximum degree of privacy allowed by the original function (Turing Machine) f. That is, for each coalision C of a subset of the parties, whatever C can compute efficiently after the termination of the protocol, can be efficiently computed from $\{x_i : i \in C\}$ and $f(x_1, x_2, ..., x_n)$. Note that here we do not require that $|C| < n/2$.

The protocol starts by each party sharing each of his input bits with all other parties, using a secret sharing scheme in which exactly n pieces are needed in order to reconstract the secret. More specifically, to share his bit b, the "owner" chooses at random n bits $b_1, b_2, ..., b_n$ satisfying $b = \sum_{i=1}^{n} b_i$. Next, the "owner" uses the public-key of each party to sends to him secretly the corresponding piece of b (i.e. party i gets $E_i(b_i)$, where E_i is ith public encryption function).

At this point, the parties hold pieces allowing to obtain the values of all the input lines in the straightline program. Our purpose is to allow the parties to hold pieces allowing to obtain the value of all output lines in the straigthline program. To this end, the parties will scan the straightline program sequentially, generating pieces for a new line from the pieces of the previous lines. We distinguish between three cases:

1) The new line is an input line. In this case, the parties already have the pieces allowing to obtain the value of this line.

2) The new line is obtained by adding the costant 1 to some previous line (say line L). In this case, one of the parties (say the first party) adds the constant 1 to his piece of line L, resulting in his piece of the current line. All other parties let their piece of line L be their piece of the current line.

3) The new line is obtained by multiplying two lines, denoted L_1 and L_2. This case is the most complex one. Let c_i be ith piece of line L_1, and d_i be his piece of line L_2. We need to compute pieces of $(\sum_{i=1}^n c_i) \cdot (\sum_{j=1}^n d_j)$. The idea is to let each party i compute by himself $b_{I,i} = c_i \cdot d_i$, and each pair $i < j$ of parties execute a two-party protocol such that guarantees the following:
(1) Party i ends with $b_{i,j}$ and party j ends with $b_{j,i}$.
(2) $b_{i,j} + b_{j,i} = c_i \cdot d_j + c_j \cdot d_i$
(3) The protocol does not leak knowledge about $b_{i,j} + b_{j,i}$.
Now party i lets $\sum_{j=1}^n b_{i,j}$ be his piece of the current line. Note that

$$(\sum_{i=1}^n c_i) \cdot (\sum_{j=1}^n d_j) = (\sum_{i=1}^n c_i \cdot d_i) + \sum_{1 \le i < j \le n} (c_i \cdot d_j + c_j \cdot d_i) = \sum_{i=1}^n \sum_{j=1}^n b_{i,j}$$

Assuming that factoring is intractable, a protocol satisfying the above three conditions is implicit in Yao's work [Y2]. In fact, using a new "Oblivious Transfer" protocol (which in turn uses ideas from [EGL]), a protocol satisfying the above conditions exists under more general condition: the existence of trapdoor encryption functions.

The proof that the above n party protocol offers the maximum possible privacy, is omitted from this extended abstract.

3. The Preserving Compiler: from Semi-Honest to Fault-Tolerant Protocols

We now present a compiler, that on input a protocol for n semi-honest parties, outputs a fault-tolerant protocol preserving the correctness and privacy of the original protocol. The protocol output by the compiler consists of two phases. In the first phase, each of the parties verifiably shares his private input in a way allowing only a majority of the parties to reconstruct it. The second phase consists of a simulation of the execution of the original protocol, using zero-knowledge proofs to convince that the simulation is proper. If a party stops responding in the second phase the non-faulty parties will detect this, reconstruct his private input, and continue on his behalf. Thus, faulty parties may "suspend" the execution of the protocol only during the first phase. But in such a case, they do so obliviously of the private inputs of the non-faulty parties, and furthermore they will be detected.

The notion of a verifiable secret sharing, presented by Chor, Goldwasser, Micali and Awerbuch [CGMA], plays a central role in phase one of our protocols. Loosely speaking, a *verifiable secret sharing* is a $n+1$-party protocol through which a *sender* (S) can distribute to the *receivers* (R_i's) *pieces* of a secret s recognizable through an a-priori known "encryption" $g(s)$. The n pieces should satisfy the following three conditions (with respect to $1 \leq lu \leq n$):

1) It is infeasible to obtain any knowledge about the secret from any l pieces;

2) Given any u messages the entire secret can be easily computed;

3) Given a piece it is easy to verify that it belongs to a set satisfying condition (2).

The notion of a verifiable secret sharing differs from Shamir's secret sharing [Sha], in that the secret is recognizable and that *the pieces should be verifiable as authentic* (i.e. condition (3)). Assuming the existence of *arbitrary* one-way permutations, a conceptually simple solution allowing $u = l + 1 \leq n$ was presented in [GMW].

During the first phase of the compiled protocol each party will generate a "certified random input" and will use verifiable secret sharing with a threshold of $n/2$ to share both his private input and his certified random input. The certified random input is generated using a distributive coin flip protocol. Such a protocol is easily constructable using the abstraction of simultanous broadcast (see Chor, Goldwasser, Micali and Awerbuch [CGMA]). As long as a majority of the processors are non-faulty, the string generated so are guaranteed to be indistinguishable from truely random strings. Implementing simultanous broadcast was reduced to verifiable secret sharing in [CGMA].

The second phase of the compiled protocol consists of a "certified simulation" of the original protocol. Note that the messages sent in the original protocol are computed in polynomial-time when given the private and random inputs of their sender and all the messages he has received. Using the result in [GMW], the sender can prove in zero-knowledge that the message he sends was properly computed.

The proof that the compiled protocol preserves the correctness and privacy of the original protocol, is omitted from this extended abstract.

4. Concluding Remarks

In the previous sections, we have restricted ourself to the special case where the parties wish to compute one common output. More generally, they may wish to compute simultaneously various different functions of the local inputs. That is, there are n predetermined functions, denoted f_1, f_2,..., f_n; and party i wishes to have $f_i(x_1, x_2, ..., x_n)$. Letting $f(x_1, x_2, ..., x_n)$ be the concatenation of $E_i(f_i(x_1, x_2, ..., x_n))$, for $1 \leq i \leq n$, the general case reduces to the special case discussed in the previous sections. Another generalization of our result is for the generation of probability distributions parametrized by the sequence of local inputs, instead of functions of this sequence.

Throughout the previous sections, we have restricted the functions to be computed in polynomial-time. The reason for this was that twofold: we wanted to end with a polynomial-time protocol, and we wanted to defeat all polynomial-time adversaries. It is obvious that a stronger result holds for all functions, when allowing the protocol to run in time polynomial in the complexity of the function and requiring that it defeats all adversaries running in time polynomial in the run-time of the protocol.

An even more general result, which we obtain using essentially the same techniques, can be phrased as demonstrating "how to play any mental game". In the rest of this abstract we explain what we mean by this.

How to Play any Mental Game (from the Introduction to the STOC version)

Many actions in life such as 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 does not study whether, or how, or under which conditions, games can be *implemented*. 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.

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. 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 a *transcenndental entity* 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 but rather knows $K_1(\sigma_1)$, which is his partial information of the state σ_1 (In poker the first player initially knows his own hand: the first 5 elements of permutation σ_1.) Based solely on $K_1(\sigma_1)$, player 1 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 to player 1) 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 is 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 general multi-party protocol falls into this framework as a very special case. It is 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, where M is the Turing machine computing the function f that for which the parties which to compute $f(x_1, ..., x_n)$. 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. In particular, *maximum privacy* is enforced by having $K_i(\sigma)$ be $f(\sigma)$ if σ is a sequence of n strings and $\perp$ otherwise.

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;* namely how will the modification of the global state by the move be effected. 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 $t \bmod n$, 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 diminishes 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. The stock market is indeed just a game, 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 based on whether combining the current hands of all players, results in a set of cards containing a royal flush. The knowledge functions are defined in a minimal manner allowing such moves; namely, a party only knows his own hand and whether a royal flush is contained in the cards held by all players. 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 interesting 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, or should we restrict our attention to the subclass of playable game?*

We show that the first question can be affirmatively answered in a computational complexity model. Namely, *every game in which the players make computable moves is playable.*

ACKNOWLEDGEMENTS

We are very grateful to Shimon Even, Mike Merritt, Dick Karp, Albert Meyer and Yoram Moses for questioning the "completeness" of our preliminary results. In particular, Dick Karp suggested games with incomplete information as (probably?) the most general formulation for our results. We are also grateful to Benny Chor and Shafi Goldwasser for helpful discussions concerning the issues of this paper. Finally, we wish to thank Stuart Haber for suggesting a simplification to our original construction.

REFERENCES

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

[Blu] Blum, M., "Coin Flipping by Phone", *IEEE Spring COMPCOM*, pp. 133-137, February 1982.

[BD] Broder, A.Z., and D. Dolev, "Flipping Coins in Many Pockets (Byzantine Agreement on Uniformly Random Values", *Proc. 25th FOCS*, 1984, pp. 157-170.

[CGMA] Chor, B., 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.

[Coh] Cohen, J.D., "Secret Sharing Homomorphisms: Keeping Shares of a Secret Secret", technical report YALEU/DCS/TR-453, Yale University, Dept. of Computer Science, Feb. 1986. Presented in *Crypto86*, 1986.

[DH] Diffie, W., and M.E. Hellman, "New Directions in Cryptography", *IEEE Trans. on Inform. Theory*, Vol. IT-22, No. 6, November 1976, pp. 644-654.

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

[GMW] Goldreich, O., S. Micali, and A. Wigderson, "Proofs that Yield Nothing But their Validity and a Methodology of Cryptographic Protocol Design", *Proc. 27th FOCS*, 1986, pp. 174-187.

[GM] Goldwasser, S., and S. Micali, "Probabilistic Encryption", *JCSS*, Vol. 28, No. 2, 1984, pp. 270-299.

[GMR] Goldwasser, S., S. Micali, and C. Rackoff, "Knowledge Complexity of Interactive Proofs", *Proc. 17th STOC*, 1985, pp. 291-304.

[R] Rabin, M.O., "How to Exchange Secrets by Oblivious Transfer", unpublished manuscript, 1981.

[Y1] Yao, A.C., "Theory and Applications of Trapdoor Functions", *Proc. of the 23rd IEEE Symp. on Foundation of Computer Science*, 1982, pp. 80-91.

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