ZERO-KNOWLEDGE AND THE DESIGN OF SECURE PROTOCOLS

(an Exposition for Engineers)

by

Oded Goldreich

Technical Report #480

December 1987
Zero-Knowledge and the Design of Secure Protocols
(an Exposition for Engineers)

Oded Goldreich
Dept. of Computer Sc.
Technion
Haifa, Israel

ABSTRACT

A communication protocol is called zero-knowledge if no "computational-knowledge" is revealed through its execution. Namely, whatever a party can efficiently compute after a zero-knowledge interaction, he can also efficiently compute by himself, when assuming the validity of some claim. The only effect of a zero-knowledge protocol is in convincing this party in the validity of that claim.

Recent results demonstrate the existence of zero-knowledge proofs for a wide class of claims, specifically for claims corresponding to the complexity class NP (the class of claims which have "short" and efficiently verifiable proofs). That is, it is possible to prove every claim in this class in a manner yielding no further knowledge over the validity of the claim. A Methodology for developing secure protocols for arbitrary purposes follow. The methodology consists of efficient compilers which transform a protocol designed to be secure in a semi-reliable model into a protocol secure in the most unreliable model.

In this paper, we survey the above development, and consider the efficiency of the solutions. In particular, we introduce measures of efficiency for zero-knowledge proofs, and single out a particular proof which has efficiency advantages over all other known proofs.
1. INTRODUCTION

We are rapidly heading into the era of electronic business communication. In the near future, we can expect to see communication networks through which various automatic business transactions take place. These transactions may be subject to very complex specifications. Furthermore, due to the large scope of users in these networks, one may no longer assume that the participants in transaction protocols are honest and follow their prescribed programs. In fact, some participants may deviate from their prescribed programs in order to gain illegitimate advantage over their counterparts. In light of this danger, the design and analysis of "secure protocols" is of major theoretical and practical importance. A "secure" protocol should guarantee that honest participants get the correct result, while ensuring that cheating participants do not get any knowledge that they are not supposed to get. The first condition, called "correctness", is a "minimum" requirement on the knowledge attained by participating in the execution of the protocol. The second condition, called "privacy", is a "maximum" requirement on the knowledge gained by executing the protocol. Together, these conditions ensure that the protocol exactly meets its specifications (i.e. is secure).

The design of secure protocols, even for particular purposes, is an extremely complex task. Coping with this problem at its full generality was considered until recently to be infeasible. This belief was contradicted by recent developments which are based on the introduction of the concept of zero-knowledge proofs [GMR] and the demonstration of such proofs for all efficiently verifiable claims [GMW]. Zero-Knowledge proofs have the remarkable property of being both convincing and yielding nothing except that the validity of their corresponding assertion. Such proofs are a very powerful tool for the design of secure fault-tolerant protocol. Typically these protocols must cope with the problem of distrustful parties convincing each other that the messages they are sending are indeed computed according to their predetermined local program. These proofs should be carried out without yielding any secret knowledge. Zero-knowledge proofs are tailored exactly for this purpose. In particular cases, zero-knowledge proofs were used to design secure protocols [FMRW, GMR, CF]. General results concerning zero-knowledge proofs, were obtained in [GMW], and have general implications on the design of secure protocols.

The general result in [GMW] is a method for constructing zero-knowledge proofs for any efficiently verifiable claim. No attempt was made in [GMW] to improve the efficiency of the method. Subsequently, alternative equivalent methods have been suggested, some of the seeming more efficient. However, no investigation of this issue has appeared until now. In this paper, we introduce natural efficiency measures for zero-knowledge proofs, and single out a method for constructing zero-knowledge proofs which is the most efficient one (with respect to all these measures).

Organization

In section 2 we present the definition of zero-knowledge proofs, demonstrate how to prove all NP statements in zero-knowledge, and discuss related efficiency considerations (subsection 2.4). In Section 3 we present a methodology for constructing secure protocols, using the above result as a primary tool.
Underlying Thesis

Throughout this paper, we associate efficient computations with those realizable in probabilistic polynomial-time. In terms of language recognition problems, probabilistic polynomial-time correspond to the complexity class $BPP$: the set of languages $L$ such that there exists a probabilistic polynomial-time Turing Machine $M$ such that if $x \in L$ then $\text{Prob}(M(x)=1) \geq 2/3$, while if $x \notin L$ then $\text{Prob}(M(x)=1) \leq 2/3$.

2. ZERO-KNOWLEDGE PROOFS

The complexity class $NP$ consists of languages whose elements possess short and easily verifiable proofs of membership. A "proof that $x \in L$" is a witness $w_x$ such that $P_L(x,w_x)=1$ where $P_L$ is a polynomially computable Boolean predicate associated to the language $L$ such that $P_L(x,y)=0$ for all $y$ if $x$ is not in $L$. The witness must have length polynomial in the length of the input $x$, but needs not be computable from $x$ in polynomial-time. A slightly different point of view is to consider NP as the class of languages $L$ for which a powerful prover may prove membership in $L$ to a polynomial-time deterministic verifier. The interaction between the prover and the verifier, in this case, is trivial: the prover sends a witness (proof) and the verifier computes for polynomial time to verify that it is indeed a proof.

This formalism was recently generalized by allowing more complex interaction between the prover and the verifier and by allowing the verifier to toss coins to be convinced by overwhelming statistical evidence [GMR, B]. The prover has some computational advantage over the verifier and for the definition to be interesting one should assume that this advantage is crucial for proving membership in the language (otherwise the verifier can do this by itself).

A fundamental measure proposed by Goldwasser, Micali and Rackoff [GMR] is that of the amount of knowledge released during an interactive proof. Informally, a proof system was called zero-knowledge if whatever the verifier could generate in probabilistic polynomial-time after "seeing" a proof of membership, he could also generate in probabilistic polynomial-time when just told by a trusted oracle that the input is indeed in the language. In other words, zero-knowledge proofs have the remarkable property of being both convincing and yielding nothing except that the assertion is indeed valid.

Besides being a very intriguing notion, zero-knowledge proofs promise to be a very powerful tool for the design of secure fault-tolerant protocols. General results concerning zero-knowledge proofs, were obtained in [GMW1], and are surveyed in this section. These results show how to give zero-knowledge proofs to every NP-statement, and have a dramatic effect on the design of fault-tolerant protocols (to be discussed in Section 3).

2.1 What is an Interactive Proof

An interactive proof system for a language $L$ is a protocol (i.e. a pair of local programs) for two probabilistic interactive machines called the prover and the verifier. Initially both machine have access to a common input tape. The two machines send messages to one another through two communication tapes. Each machine only sees its own tapes, the common input tape and the communication tapes. In particular, it follows that one machine cannot monitor the internal computation of the other machine nor read the other's coin tosses, current state, program etc. The verifier is bounded to a number of steps which is polynomial in the
length of the common input, after which he stops either in an accept state or in a reject state. At this point we put no restrictions on the local computation conducted by the prover.

We require that, whenever the verifier is following his predetermined program, \(V\), the following two conditions hold:

1) **Completeness of the interactive proof system:** If the common input \(x\) is in \(L\) and the prover runs his predetermined program, \(P\), then the verifier accepts \(x\) with probability \(\geq 1 - \|x\|^{-c}\), for every constant \(c > 0\). In other words, the prover can convince the verifier of \(x \in L\).

2) **Soundness of the interactive proof system:** If the common input \(x\) is NOT in \(L\), then for every program \(P^*\), run by the prover, the verifier rejects \(x\) with probability \(\geq 1 - \|x\|^{-c}\) (for every constant \(c > 0\)). In other words, the prover cannot fool the verifier.

**Remark 1:** Note that it does not suffice to require that the verifier cannot be fooled by the predetermined prover (such a mild condition would have presupposed that the "prover" is a trusted oracle).

**Remark 2:** The ability to toss coins is crucial to the non-triviality of the notion of an interactive proof system. If the verifier is deterministic then interactive proof systems coincide with \(NP\).

### 2.2 What is a Zero-Knowledge Proof

Intuitively, a zero-knowledge proof is a proof which yields nothing but its validity. This means that for all practical purposes, "whatever" can be done after interacting with a zero-knowledge prover, can be done when just believing that the assertion he claims is indeed valid. (In "whatever" we mean not only the computation of functions but also the generation of probability distributions.) Thus, zero-knowledge is a property of the predetermined prover. It is the robustness of the prover against attempts of the verifier to extract knowledge via interaction. Note that the verifier may deviate arbitrarily (but in polynomial-time) from the predetermined program. This is captured by the formulation appearing in [GMR] and sketched below.

Denote by \(V^*(x)\) the probability distribution generated by a machine \(V^*\) which interacts with (the prover) \(P\) on input \(x \in L\). We say that the proof system is **zero-knowledge** if for all probabilistic polynomial-time machines \(V^*\), there exists a probabilistic polynomial-time algorithm \(M_{V^*}\) that on input \(x\) produces a probability distribution \(M_{V^*}(x)\) such that \(M_{V^*}(\cdot)\) and \(V^*(\cdot)\) are polynomially-indistinguishable (see below) for \(x \in L\).

For the sake of self-containment, we recall the definition of polynomial-indistinguishable ensembles [GM, Y1]. Let \(D_1(\cdot)\) and \(D_2(\cdot)\) be two ensembles (i.e. two sequences of probability distributions). That is, for every \(i \in \{1,2\}\) and \(x \in \{0,1\}^*\), \(D_i(x)\) is a probability distribution over 0-1 strings. Typically, there exists a polynomial \(p(\cdot)\), such that the support of \(D_i(x)\) will consist only of strings with length \(\leq p(\|x\|)\). For every algorithm \(A\), let \(p_i^A(x)\) denote the probability that \(A\) outputs 1 on input \(x\) and an element chosen according to the probability distribution \(D_i(x)\). The ensembles \(D_1(\cdot)\) and \(D_2(\cdot)\) are polynomially-indistinguishable if for every probabilistic polynomial-time algorithm \(A\), for every constant \(c > 0\) and for all sufficiently long \(x\)'s

\[
p_A(x) - p_{A'}(x) \leq \|x\|^{-c}.
\]

Polynomially-indistinguishable probability distributions should be considered equal for all practical purposes, since any application running in polynomial-time and using either distributions, demonstrates essentially the same behaviour. It follows that polynomially-indistinguishability of \(V^*(x)\) and \(M_{V^*}(x)\)
suffices for saying that nothing substantial is gained by interacting with the prover, except of course conviction in the validity of the assertion $x \in L$.

Remark 3: It is important to stress that in our model of interactive proofs the internal computations of the parties are no assumed to be simultaneous. Thus, one machine can not infer the number of steps taken by its counterpart from the "delay" between the communications. In real applications, such inference may be possible, and an easy modification of the definition of zero-knowledge will be adequate.

2.3 All Languages in NP Have Zero-Knowledge Proof Systems

In this section we assume the existence of secure encryption schemes (in the sense of Goldwasser and Micali [GM]). Such schemes exist if unapproximable predicates exist [GM]. The existence of unapproximable predicates has been shown by Yao to be a weaker assumption than the existence of one-way permutations [Y1]. An encryption scheme secure as in [GM] is a probabilistic polynomial-time algorithm $f$ that on input $x$ and internal coin tosses $r$, outputs an-encryption $f(x, r)$. Decryption is unique: that is $f(x, r) = f(y, s)$ implies $x = y$.

Following [GMW1], we begin by presenting a zero-knowledge interactive proof for graph 3-colourability. The language $\text{Graph 3-Colourability}$ consists of the set of graphs, the vertices of which can be coloured using three colours such that no two adjacent vertices are assigned the same colour. Using this interactive proof and the fact that Graph 3-Colouatability is NP-Complete, we present zero-knowledge proofs for every language in NP.

A Zero-Knowledge Proof for Graph 3-Colourability

The common input to the following protocol is a graph $G(V, E)$. In the following protocol, the prover needs only to be a probabilistic polynomial-time machine which gets a proper 3-colouring of $G$ as an auxiliary input. Let us denote this colouring by $\phi : (\phi : V \rightarrow \{1, 2, 3\})$. Let $n = |V|$, $m = |E|$. For simplicity, let $V = \{1, 2, \ldots, n\}$.

Protocol 1

The following four steps are executed $m^2$ times, each time using independent coin tosses.

1) The prover chooses a random permutation of the 3-colouring, encrypts it, and sends it to the verifier. More specifically, the prover chooses at random a permutation $\pi \in \text{Sym}((\{1, 2, 3\}))$, and random $r_v$'s, computes $R_v = f(\pi(\phi(v)), r_v)$ (for every $v \in V$), and sends the sequence $R_1, R_2, \ldots, R_n$ to the verifier.

2) The verifier chooses at random an edge $e \in E$ and sends it to the prover. (Intuitively, the verifier asks to examine the colouring of the endpoints of $e \in E$.)

3) If $e = (u, v) \in E$ then the prover reveals the colouring of $u$ and $v$ and "proves" that they correspond to their encryptions. More specifically, the prover sends $(\pi(\phi(u)), r_u)$ and $(\pi(\phi(v)), r_v)$ to the verifier. If $e \notin E$ then the prover stops.

4) The verifier checks the "proof" provided in step (3). Namely, the verifier checks whether $R_u = f(\pi(\phi(u)), r_u)$, $R_v = f(\pi(\phi(v)), r_v)$, $\pi(\phi(u)) \neq \pi(\phi(v))$, and $\pi(\phi(u)), \pi(\phi(v)) \in \{1, 2, 3\}$. If either condi-
tion is violated the verifier rejects and stops. Otherwise the verifier continues to the next iteration. If the verifier has completed all $m^2$ iterations then it accepts.

The reader can easily verify the following facts: When the graph is 3-colourable and both prover and verifier follow the protocol then the verifier always accepts. When the graph is not 3-colourable and the verifier follows the protocol then no matter how the prover plays, the verifier will reject with probability at least $(1 - m^{-1})^m = \exp(-m)$. Thus, Protocol I constitutes an interactive proof system for 3-colourability. Clearly, Protocol I yields no knowledge to the specified verifier, since all he gets is a sequence of random pairs. The proof that Protocol I is indeed zero-knowledge (with respect to any verifier) is much more complex, and is omitted. We get

**Proposition 1:** If $f(\cdot, \cdot)$ is a secure probabilistic encryption, then Protocol I constitutes a zero-knowledge interactive proof system for 3-colourability.

**Zero-Knowledge Proofs for all NP**

For every language $L$ in NP, there exist an efficient transformation of instances of the language $L$ to instances of 3-colourability. This transformation is called a reduction, and is guaranteed by the fact that 3-colourability is NP-complete (see [GI]). Incorporating the standard reductions into Protocol I, we get

**Theorem 2:** If $f(\cdot, \cdot)$ is a secure probabilistic encryption, then every NP language has a zero-knowledge interactive proof system. Furthermore, the prover in this system may be a probabilistic polynomial-time machine that gets an NP-proof as an auxiliary input.

**Proof:** For every language $L \in \text{NP}$ the protocol incorporates a fixed reduction of $L$ to 3-colourability. Each party computes the 3-colourability instance from the common input, and then the prover proves to the verifier that this instance is 3-colourable (using Protocol I). It is important to note that the NP-proof for membership in $L$ is efficiently transformable into an NP-proof of 3-colourability of the resulting graph.

**2.4. Efficiency Measures for Zero-Knowledge Proofs**

Following the presentation of a zero-knowledge interactive proof for Graph 3-Colourability, many researchers have presented "direct" zero-knowledge proofs for a variety of NP-Complete languages. A partial list of such languages include Hamiltonian Circuit [zero-knowledge proof communicated by M. Blum], Modular Knapsack [A. Shamir], Maximum Clique [M. Rabin], 3-Exact-Cover [N. Linial and L. Lovasz], Satisfiability [J. Benaloh and others], Circuit Value [D. Chaum and others]. All these zero-knowledge proofs are equivalent in the sense that each of them implies Theorem 2 (and thus implies all the others). However, some of these zero-knowledge proofs seem more "efficient" than others.

In order to compare the efficiency of zero-knowledge proofs for NP-Complete languages, we consider the zero-knowledge proof obtained by applying the construction of (the proof of) Theorem 2 to these interactive proofs. That is, we consider the cost of particular zero-knowledge proofs as design blocks for the
construction of zero-knowledge proofs for an arbitrary language in NP. (We consider here only zero-knowledge proofs which rely on the existence of arbitrary encryption functions.)

Two standard efficiency measures are:

1) The computational complexity of the proof (i.e. number of steps taken by either or both parties).
2) The communication complexity of the proof. Here one may consider the number of interactions, and/or the total number of bits exchanged.

A non-standard measure for the efficiency of a zero-knowledge proof is its tightness. Intuitively, tightness is the ratio between the time it takes the simulator to simulate an interaction with the prover over the time the interaction takes the verifier. Formally, the knowledge-tightness (tightness) of a zero-knowledge proof is a function $t : N \rightarrow N$ (from integers to integers) such that for all polynomial-time ITMs $V^*$

$$\frac{\text{time}(M_{V^*}(x))}{\text{time}(V^*(x))} < t(|x|),$$

where $\{M_{V^*}(x)\}_{x \in L}$ is polynomially indistinguishable from $\{<P,V^*>(x)\}_{x \in L}$, and $\text{time}(A(x))$ is the number of steps taken by machine $A$ on input $x$.

The definition of zero-knowledge only guarantees that tightness does not have to grow faster than a polynomial. However, the definition does not guarantee that tightness can be bounded above by a particular polynomial. It is easy to see that the knowledge tightness of Protocol 1 is $m$ (the number of edges in the graph). Protocols with constant knowledge tightness and a number of iterations which is logarithmic exist for all languages in NP. Suggestions are due to Benaloh, Blum, Chaum, Impagliazzo, Rabin, Shamir and possibly others. The most efficient suggestion (concurrently in all measures) is a protocol which uses the circuit value problem. This suggestion originates in [Cha]. A modification is presented in the sequel.

The Circuit Value Problem

A non-deterministic circuit is a directed acyclic graph, with vertices marked by the following symbols. A vertex of in-degree zero is called a leaf and is marked by a literal (a Boolean variable or its negation). Variables are either input variables or auxiliary variables (simulating non-determinism). With no loss of generality, all non-leaves have in-degree two. Non-leaves are marked by either $\land$ or $\lor$. Once values are assigned to all variables, computation in the circuit is carried out in the obvious way (e.g. if the ingoing edges of a vertex marked by $\land$ carry values 1 and 0 respectively, then the out-going edges of the vertex carry the value 0). A non-deterministic circuit accepts an input if there exists an assignment of the auxiliary variables such that, assigning the input bits to the corresponding input variables, the circuit evaluates to 1.

Any language in $NP$ can be recognized by a family of polynomial-size non-deterministic circuits. If the original Turing Machine accepts inputs of length $n$ in time $t(n)$ then the $n$-th circuit, which accepts inputs of length $n$, has size $t(n) \cdot \log_2 t(n)$ [PF]. (Constructing a circuit of size $t^2(n)$ is obvious.) To determine whether a non-deterministic circuit accepts a particular input, one may consider the auxiliary circuit obtained by assigning the input bits to the input variables and leaving the auxiliary variables unspecified. The circuit value problem then consists of whether there exists an assignment under which the auxiliary circuit evaluates to 1. It is clear from the above, that we may consider the circuit value problem instead of considering $NP$. 

- 7 -
A Zero-Knowledge Proof for the Circuit Value Problem

The language we consider now is the set of circuits which evaluate to 1, under some assignment to their variables. Let \( \epsilon(\cdot) \) be a desired bound on the error probability of the interactive proof, as a function of the input length, \( n \). Typically, \( \epsilon^{-1}(n) \) grows faster than any polynomial in \( n \).

Protocol 2

The interactive proof proceeds in \( \log_2 \epsilon^{-1}(n) \) rounds, each consisting of the following steps.

1A) The prover chooses randomly and independently a bit \( \sigma_e \) for each edge \( e \) in the circuit. This bit will serve as the encoding of the (zero) value on that edge. The prover encrypts the encoding of all edges and sends it to the verifier. The "output edge" (on which the circuit's result appears) is always encoded by 0.

1B) For each internal vertex, the prover constructs an encoded truth table as follows. First, take the truth table corresponding to the gate assigned to the vertex, and substitute each value in it by the corresponding encoding. (E.g. if edges \( e_1 \) and \( e_2 \) are joined into edge \( e_3 \) by an AND gate, then the original truth table has rows 000, 010, 100, 111, and the encode truth table has rows \( \sigma_1,\sigma_2,\sigma_3, \sigma_1,\sigma_2,\sigma_3, \sigma_1,\sigma_2,\sigma_3, \sigma_1,\sigma_2,\sigma_3 \).) Second, randomly permute the rows of the table. The prover encrypts the encodings of each row of each truth table and sends them to the verifier.

1C) The prover takes an assignment under which the circuit evaluates to 1, and computes all intermediate results. The prover encodes these results by using the edge value encodings, encrypts all these computation values encodings and sends them to the verifier.

2) The verifier answers with a randomly chosen bit \( b \).

3A) If \( b=0 \) then the prover proves the consistency of the edge and vertex encodings. To this end, the prover "opens" the encryptions of all edge encodings and of all rows in all encoded truth table. The verifier may readily check that these are consistent with the vertices markings.

3B) If \( b=1 \) then the prover proves that the encoded computation evaluates to 1. This is done by "opening" the encryption of the computation encodings and also "opening" the encryptions of the row in each truth table which corresponds to these values. (E.g., let \( \tau_1,\tau_2,\tau_3 \) be the encoded computation values on edges \( e_1, e_2, e_3 \), resp., such that \( e_1 \) and \( e_2 \) are the ingoing edges of vertex \( v \) and \( e_3 \) is an outgoing edge of \( v \). Then the prover opens the encryption of a row in the encoded truth table of vertex \( v \) which reads \( \tau_1,\tau_2,\tau_3 \).)

Proposition 1: If \( \epsilon(\cdot) \) is a secure probabilistic encryption, then Protocol 2 constitutes a zero-knowledge interactive proof system for the circuit value problem.

Efficiency Analysis of Protocols 1 and 2

Let \( n \) denote the size of the circuit. The number of messages exchanged in Protocol 2 is \( O(\log \epsilon^{-1}(n)) \). Using Protocol 1 and incorporating the standard reductions (we get \( n=\Theta(m) \)), we see that the number of rounds in the resulting protocol is \( \Theta(n^2) \). A finer analysis yields \( \Theta(n \cdot \log \epsilon^{-1}(n)) \) rounds in that protocol. The
length of the messages exchanged in the two protocols is about the same, thus in terms of communication complexity Protocol 2 is more efficient by a factor of $n$. The reader can easily verify that the computational complexity of the two protocols relate in the same manner.

Protocol 2, has knowledge tightness 2. Recall that the knowledge tightness of Protocol 1 is $m$ - the number of edges in the graph.

Conclusions

Protocol 2 is superior, in all three efficiency measures, to Protocol 1. We thus recommend that Proposition 1' be used to generate zero-knowledge proofs for any NP statement. In special cases, specially designed zero-knowledge proofs may even be more efficient, especially if one relies on special encryption functions. For an example see [Bh2].

3. A METHODOLOGY FOR THE DESIGN OF SECURE PROTOCOLS

In this section, we present an extremely powerful methodology for designing correct fault-tolerant protocols. The methodology consists of efficient "correctness and privacy preserving" transformations of protocols from a weak adversary model to the most adversarial model. These transformations are informally summarized as follows.

Informal Theorem A: There exist an efficient compiler transforming a protocol $P$ designed for $n=2t+1$ honest players, to a fault-tolerant protocol $P'$ that achieves the same goals even if $t$ of its $n$ players are faulty. Faulty players are allowed to deviate from $P'$ in an arbitrary but polynomial-time way.

In the formal statement of the corresponding Theorem, we avoid talking about "achieving goals". The "goal of a protocol" is a semantic object that is not well understood. Instead, we make statements about well understood syntactic objects: the probability distribution on the tapes of interactive machines. In particular, we define the notions of a "correctness preserving compiler" and a "privacy preserving compiler". Both notions are defined as relations between the probability distribution on the tapes of interactive machines during the execution of protocol $P$ (in a weak adversarial environment) and the distribution on these tapes during the execution of $P'$ (in a strong adversarial environment). Loosely speaking, "preserving correctness" means that whatever a party could compute after properly participating in the original protocol $P$, he could also compute when following the transformed protocol $P'$. "Preserving privacy" means that whatever a set of dishonest players can compute after participating in $P'$, the corresponding players in $P$ can compute when sharing their "knowledge" after participating in $P$. Similarly we formalize the following

Informal Theorem B: There exist an efficient compiler transforming a two-party protocol $P$ that is correct in a fail-stop model, to a two-party fault-tolerant protocol $P'$ that achieves the same goals even if one of the players deviates from $P'$ in an arbitrary but polynomial-time way.

The proofs of the above Theorems make primary use of Theorem 2 to allow a machine to "prove" to other machines that a message it sent is computed according to the protocol. In addition, these proofs make innovative use of most of the cryptographic techniques developed in recent years. Essential ingredients in the proof
of Theorem A are the notions of verifiable secret sharing and simultaneous broadcast proposed and first implemented by Chor, Goldwasser, Micali, and Awerbuch [CGMA]. An essential ingredient in the proof of Theorem B is Blum’s “coin flipping into the well” [Blu].

Further Improvements

Theorem A constitutes a procedure for automatically constructing fault-tolerant protocols, the goal of which is to compute a predetermined function of the private inputs scattered among the players. This procedure takes as input a distributed specification of the function (i.e. a protocol for honest players), not the function itself. It is guaranteed that this procedure will output a fault-tolerant protocol for computing this very function (i.e. the “correctness” condition) and that the “privacy” present in the specification will be preserved. Thus, the degree of privacy offered by the output fault-tolerance protocol depends on the specification, and not on the function to be computed. Recently, an efficient protocol generator that, on input a Turing Machine program of a function, output a protocol offering the privacy of the function, was presented [GMW2]. The protocol output by the generator has the following (“maximum privacy”) property: at the termination of the protocol, each subset of players can compute from their joint local history only whatever they could have computed from their corresponding local inputs and the value of the function.

ACKNOWLEDGEMENTS

We wish to thank Silvio Micali and Avi Wigderson for collaboration in research exposed in this paper.

REFERENCES


