Proofs that Yield Nothing But their Validity and a Methodology of Cryptographic Protocol Design

(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

In this paper we demonstrate the generality and wide applicability of *zero-knowledge proofs*, a notion introduced by Goldwasser, Micali and Rackoff. These are probabilistic and interactive proofs that, for the members x of a language L, efficiently demonstrate membership in the language without conveying any additional knowledge. So far, zero-knowledge proofs were known only for some number theoretic languages in $NP \cap Co\text{-}NP$.

SUMMARY OF OUR RESULTS

Under the assumption that encryption functions exist, we show that all languages in NP have zero-knowledge proofs. That is, it is possible to demonstrate that a CNF formula is satisfiable without revealing any other property of the formula. In particular, without yielding neither a satisfying assignment nor properties such as whether there is a satisfying assignment in which $x_1 = x_3$ etc.

The above result allows us to prove two fundamental theorems in the field of (two-party and multi-party) cryptographic protocols. These theorems consist of automatic and efficient transformations that, given a protocol that is correct with respect to an extremely weak adversary, output a protocol correct in the most adversarial scenario. Thus, these theorems imply a powerful methodology for developing secure two-party and multi-party protocols.

Work done while first author was at the Laboratory for Computer Science, MIT; and the third author was at the Mathematical Sciences Research Institute, UC-Berkeley. Work was partially supported by an IBM Postdoctoral Fellowship, NSF Grants DCR-8509905 and DCR-8413577, and an IBM Faculty Development Award.

We also demonstrate that zero-knowledge proofs exist "independently of cryptography and number theory". Using no unproved assumptions, we show that both graph isomorphism and graph non-isomorphism possess zero-knowledge interactive proofs. The mere existence of an interactive proof for graph non-isomorphism is interesting, since graph non-isomorphism is not known to be in NP and thus did not possess so far any efficient proofs.

1. INTRODUCTION

It is traditional to view NP as the class of languages whose elements posses short 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.

0272-5428/86/0000/0174$01.00 © 1986 IEEE

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 and 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). In other words, we will implicitly assume that there exist interesting languages (say in $PSPACE$) which are not in BPP, and be interested in proof systems for such languages.

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 cryptographic 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. Such proofs should be carried out without yielding any secret knowledge. In particular cases, zero-knowledge proofs were used to design secure protocols [FMRW, GMR, CF]. However, in order to demonstrate the generality of this tool (and to utilize its full potential) one should have come with general results concerning the existence of zero-knowledge proof systems. Until now, no such general results were obtained.

In this paper, we present general results concerning zero-knowledge proof systems. In particular, we show how to give zero-knowledge proofs to every NP-statement. A general methodology for designing secure cryptographic protocols follows. Its core is a compiler which, making primary use of the above result, translates protocols correct in a weak adversary model to protocols correct in the most adversarial environment.

1.1 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) *Validity 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.

An important example of an interactive proof system is presented in section 2.1.

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: As is the case with NP, the conditions imposed on acceptance and rejection are not symmetric. Thus the existence of an interactive proof for the language L *does not* imply its existence for the complement of L.

Remark 3: The above "definition" follows the one of Goldwasser, Micali and Rackoff [GMR]. A different definition due to Babai [B], restricts the verifier to generate random strings, send them to the prover, and evaluate a deterministic polynomial-time predicate at the end of the interaction. Demonstrating the existence of proof systems is easier when allowing the verifier to flip private coins (i.e. [GMR] model), while relating interactive proof systems to traditional complexity classes seems easier if one restricts oneself to Babai's model. Surprisingly, these two models are equivalent, as far as language recognition is concerned [GS] (see Sec. 1.3).

Remark 4: 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.

Remark 5: Without loss of generality, we assume that the last message sent during an interactive proof is sent by the prover. (A last message sent by the verifier has absolutely no effect.)

1.2 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.
(For every algorithm A, let $p_A(x)$ denote the probability that A outputs 1 on input x and an element chosen according to the probability distribution $D(x)$. Similarly, $p_A'(x)$ is defined (w.r.t. D'). The distribution ensembles $D(\cdot)$ and $D'(\cdot)$ are *polynomially-indistinguishable* if for every probabilistic polynomial-time algorithm A, $p_A(x)-p_A'(x) \leq |x|^{-c}$, for every constant $c>0$ and sufficiently long x. This notion appeared in [GM] and in [Y1].)

Remark 6: It is not difficult to see that if a language L has a zero-knowledge proof system in which only one message is sent, then $L \in BPP$. Thus, the non-triviality of the interaction is a necessary condition for the non-triviality of the notion of zero-knowledge.

1.3 Previous results concerning interactive proof systems

Let Q be a polynomial. Denote by $IP(Q)$ the class of languages L such that membership of $x \in L$ can be proved through

a general interaction consisting of $Q(|x|)$ message exchanges. Similarly, let $AM(Q)$ denote languages proven through the restricted type interaction in which the verifier only tosses "public coins" (i.e. Babai's Arthur-Merlin framework). Babai [B] showed that for every polynomial Q, $AM(Q+1)=AM(Q)$. This means that his finite level hierarchy collapses. (Note that this does not imply the collapse of the unbounded level hierarchy! For more details see [AGH].) Goldwasser and Sipser [GS] showed that, for every polynomial Q, $IP(Q) \subseteq AM(Q+2)$. This means that from a complexity theoretic point of view, the $IP(\cdot)$ hierarchy and the $AM(\cdot)$ hierarchy essentially coincide. Both the above results say nothing about the preservation of zero-knowledge by the transformations.

The bounded level IP hierarchy is related to the polynomial-time hierarchy by Babai's proof that $AM(2) \subseteq \Pi_2^P$ and that $AM(2) \subseteq NP^B$ for almost all oracles B.

Several Number Theoretic languages, not known to be in BPP, have been previously shown to have zero-knowledge proof systems. The first language for which such a proof system has been demonstrated is Quadratic Non-Residuosity [GMR]. Other zero-knowledge proof systems were presented in [GMR], [GHY], [CF] and [G]. All these languages are known to lie in $NP \cap Co\text{-}NP$.

1.4 Organization of the Paper

In Section 2 we present zero-knowledge interactive proofs for graph isomorphism and graph non-isomorphism. We also discuss complexity theoretic implications of the existence of an interactive proof for graph non-isomorphism.

In Section 3 we show how to use any one-way permutation in order to construct a zero-knowledge interactive proof for any language in NP. This result is extended to any language in IP.

In Section 4, we outline the methodological theorems for two-party and multi-party cryptographic protocols.

2. Proofs of Graph Isomorphism and Graph Non-Isomorphism

We start by presenting a (probably non-zero-knowledge) interactive proof for graph non-isomorphism. Next we present a zero-knowledge interactive proof for graph isomorphism, and for graph non-isomorphism. Let us set some common notations.

Let A be a set. Then $Sym(A)$ denote the set of permutations over A. When writing $a \in_R A$, we mean an element chosen at random with uniform probability distribution from the set A.

We will consider undirected graphs, $G(V,E)$. V will denote the vertex set, and E the edge set of the graph G. n will denote the size of the vertex set, and m the size of the edge set (i.e. $n=|V|$, $m=|E|$). The graph $G(V,E)$ will be represented by the set E, in an arbitrary fixed order (e.g. lexicographic).

Two graphs $G(V,E)$ and $H(V,F)$ are *isomorphic* if and only if there exist a permutation $\pi \in Sym(V)$ such that

$$(u,v) \in E \text{ iff } (\pi(u),\pi(v)) \in F.$$

The *graph isomorphism problem* consists of two graphs as input, and one has to determine whether they are isomorphic. The graph isomorphism problem is trivially in NP, is not known to be in Co-NP, and is believed not to be NP-complete.

We say that the graph $H(V,F)$ is a *random isomorphic copy* of the graph $G(V,E)$ if H is obtained from G by picking $\pi \in_R Sym(V)$ and letting

$$F=\{(\pi(u),\pi(v)): (u,v) \in E\}.$$

2.1 An Interactive Proof of Graph Non-Isomorphism

In this subsection we examplify the notion of an interactive proof system by presenting an interactive proof for graph non-isomorphism. The fact that graph non-isomorphism has interactive proofs is interest-

ing as it is not know to be in NP, and thus has not been know previously to have any efficient proofs. Moreover, the existence of an interactive proof for graph non-isomorphism has interesting complexity theoretic consequences.

In the following protocol the prover needs only to be a probabilistic polynomial-time machine with access to an oracle for graph isomorphism.

common input: Two graphs $G_1(V,E_1)$ and $G_2(V,E_2)$.

1) The verifier chooses at random n integers $\alpha_i \in_R \{1,2\}$, $1 \leq i \leq n$. The verifier computes n graphs $H_i(V,F_i)$ such that H_i is a random isomorphic copy of G_{α_i}. The verifier sends the H_i's to the prover.
2) The prover answers with a string of β_i's (each in $\{1,2\}$), such that $H_i(V,F_i)$ is isomorphic to $G_{\beta_i}(V,E_{\beta_i})$.
3) The verifier tests whether $\alpha_i = \beta_i$, for every $1 \leq i \leq n$. If the condition is violated then the verifier *rejects*; otherwise he *accepts*.

Theorem 1: *The above protocol constitutes a (two-move) interactive proof system for Graph Non-Isomorphism.*

proof: If the graphs G_1 and G_2 are not isomorphic, and both prover and verifier follow the protocol, then the verifier always accepts. If on the other hand, G_1 and G_2 are isomorphic then, for each i, we have $\alpha_i \neq \beta_i$ with probability at least 1/2, even if the prover does not follow the protocol. The reason being that in case G_1 and G_2 are isomorphic,

$$Prob(\alpha_i = 1 \mid \text{verifier sent } H_i) = 1/2.$$

The probability that the verifier does not reject two isomorphic graphs is thus at most 2^{-n}.

The above Theorem has interesting implications on the traditional complexity of the graph isomorphism problem. Namely,

Corollary 1: *Graph Isomorphism is in $(NP \cap Co\text{-}NP)^A$, for a random oracle A. Also, Graph Non-Isomorphism can be recognized by a (non-uniform) family of non-deterministic polynomial-size circuits (i.e. non-uniform NP).*

proof: By the Theorem 1, Graph Non-Isomorphism (*GNI*) is in $IP(2)$. Using Goldwasser and Sipser's transformation of $IP(k)$ protocols to $AM(k+2)$ protocols, $GNI \in AM(4)$. By Babai's proof of the finite $AM(\cdot)$ collapse, $GNI \in AM(2) \subseteq NP^A$ for a random oracle A. Finally, it has been pointed out by Mike Sipser that $AM(2)$ is contained in non-uniform NP.

Another interesting corollary concerning graph isomorphism is due to Boppana and Hastad [BH].

Corollary 2 [BH]: *If Graph Isomorphism is NP-Complete then the polynomial-time hierarchy collapses to its second level.*

proof: Boppana and Hastad showed that if $Co\text{-}NP \subseteq IP(k)$ (for some fixed k) then the entire polynomial-time hierarchy collapses to $AM(2) \subseteq \Pi_2^P$. Since Theorem 1 states that graph non-isomorphism is in $IP(2)$, the Corollary follows.

Corollary 2 may be viewed as providing additional support to the belief that Graph Isomorphism is not NP-Complete.

2.2 A Zero-Knowledge Proof for Graph Isomorphism

In this section we examplify the notion of zero-knowledge proof systems by presenting a zero-knowledge proof for graph isomorphism. The fact that graph isomorphism has efficient proofs is apparent, since it is in NP. However, the fact that graph isomorphism can be proved in zero-knowledge, and in particular without demonstrating the isomorphism is interesting.

In the following protocol, the prover needs only to be a probabilistic polynomial-time machine which gets, as an auxiliary input, the isomorphism between the input graphs.

common input: Two graphs $G_1(V,E_1)$ and $G_2(V,E_2)$.

Let ϕ denote the isomorphism between G_1 and G_2. The following four steps are executed n times, each time using independent random coin tosses.

1) The prover generates a graph H, a random isomorphic copy of G_1. This is done by selecting a permutation $\pi \in_R Sym(V)$, and computing $H(V,F)$ such that $(\pi(u),\pi(v)) \in F$ iff $(u,v) \in E_1$. The prover sends the graph $H(V,F)$ to the verifier.
2) The verifier chooses at random $\alpha \in_R \{1,2\}$, and sends α to the prover. (Intuitively, the verifier asks the prover to prove to him that H and G_α are indeed isomorphic.)
3) If $\alpha \notin \{1,2\}$ then the prover halts. If $\alpha=1$ then the prover sends π to the verifier, else the prover sends $\pi\phi^{-1}$.
4) If the permutation received from the prover is not an isomorphism between G_α and H then the verifier stops and *rejects*; otherwise he continues.

If the verifier has completed n iterations of the above steps then he *accepts*.

The reader can easily verify that the above constitutes an interactive proof system for graph isomorphism. Intuitively, this proof is zero-knowledge since whatever the verifier receives is "useless", as he can generate random isomorphic copies of the input graphs by himself. This is easy to see in case the verifier follows the protocol. In case the verifier deviates from the protocol, the situation is much more complex. The verifier may set the α's depending on the graphs presented to him. In such a case it can not be argued that the verifier only receives random isomorphic copies of the input graph. The issue is fairly involved, as we have to defeat a universal quantifier which is not well understood (i.e. all possible deviations from the protocol). We cannot really trust our intuition in such matters, so a formal proof is indeed required.

Theorem 2: *The above protocol constitutes a zero-knowledge interactive proof system for Graph Isomorphism.*

proof's sketch: It is clear that the above prover conveys no knowledge to the *specified* verifier. We need however to show that our prover conveys no knowledge to all possible verifiers, including cheating ones that deviate arbitrarily from the protocol.

Let V^* be an arbitrary fixed program of a probabilistic polynomial-time machine interacting with the prover, specified by the protocol. We will present a probabilistic polynomial-time machine M_{V^*} that generates a probability distribution which is identical to the probability distribution induced on V^*'s tapes during its interaction with the prover. In fact it suffices to generate the distribution on the random tape and the communication tape of V^*

Our demonstration of the existence of such M_{V^*} is constructive: given an interactive program V^*, we use it in order to construct the machine M_{V^*}. The way we use V^* in this construction does not correspond to the traditional notion of (a subroutine) reduction [K, C], but rather to a more general notion of reduction suggested in [AHU, pp. 373-374]. Typically, we will try to guess which isomorphism the machine V^* will ask to check. We will construct the graph H such that we can answer V^* in case we were lucky. The cases in which we failed will be ignored. It is crucial that from the point of view of V^* the case which leads to our success and the case which

leads to our failure look identical. By throwing away the instances where we failed, we only slow down our construction, but we do not change the probability distribution that V^* "sees".

Following is a more detailed description of M_{V^*}. On input G_1 and G_2, the machine M_{V^*} will monitor the execution of the program V^* on this input and will "simulate" the prover to V^*. M_{V^*} will start by choosing and fixing random coin tosses r (random tape) for V^*, and placing r on a special *record tape*. All subsequent coin tosses are for M_{V^*}. (The random tape of V^*, denoted r, will remain fix and V^* is "deterministic" given its random tape r.) Machine M_{V^*} proceeds in n rounds as follows.

1) M_{V^*} chooses at random $\beta \in_R \{1,2\}$ and a permutation $\pi \in_R Sym(V)$. It computes $H(V,F)$ such that $(\pi(u),\pi(v)) \in F$ if and only if $(u,v) \in E_\beta$. M_{V^*} places H on the communication tape of V^*. (Note that H is an isomorphic copy of G_β.)
2) M_{V^*} reads V^* answer from the communication tape of V^*. When V^* answers with $\alpha=\beta$ (lucky for M_{V^*}), machine M_{V^*} places π on the communication tape of V^*, appends (H,α,π) to its record tape, and proceeds to the next round. If $\alpha \notin \{1,2\}$ (V^* is obviously cheating) then the machine M_{V^*} appends (H,α) to its record tape and stops outputting its record tape. If $\alpha+\beta=3$ (unlucky for M_{V^*}) then M_{V^*} is going to repeat the current round. This is done by "rewinding" V^* to its configuration at the beginning of the current round, and by repeating Steps 1 and 2 with new random choices. (V^* configuration consists of the contents of its tapes, the positions of its heads and its internal state.)

If all rounds are completed then M_{V^*} outputs its record and halts. It should be noted that, for each repetition of the ith round, $Pr(\beta=1 \mid H^{(i)})=1/2$, where $H^{(i)}$ is the list of graphs send to V^* so far (this includes the graph sent in the current repetition of round i, but does not include graphs after which V^* was rewound). Therefore, $Pr(\beta=\alpha(r,H^{(i)}) \mid H^{(i)})=1/2$, where $\alpha(r,H^{(i)})$ is V^*'s answer on random tape r and communication tape $H^{(i)}$. It is left to the reader to verify that the ith round is repeated j times with probability at most 2^{-j}. Machine M_{V^*} stops and outputs its record tape after n rounds were completed or after encountering an improper $\alpha \notin \{1,2\}$. In the first case the machine outputs a sequence of n triples of the form (H,α,π), where π is an isomorphism between H and G_α. It is left to the reader to verify that in both cases, M_{V^*} outputs the right probability distribution. ∎

Remark 7: In the above proof, the probability distribution output by the simulator (M_{V^*}) is *identical* to the distribution during an interaction between V^* and the prover. This is more than required by the definition of zero-knowledge, which only requires that these distributions be polynomially-indistinguishable. We call a proof system for which such a result (i.e. identical distributions) is demonstrated a *perfect zero-knowledge* proof system.

Remark 8: *Serial execution v. parallel execution: the case where the intuition fails?* Although one's intuition may insist that the above zero-knowledge protocol, remains zero-knowledge even when executed in parallel instead than serially, we do not know how to prove this statement. We even doubt this intuition, and will explain why in the full version of this paper.

2.3 Zero-Knowledge Proof of Graph Non-Isomorphism

The interactive proof for graph non-isomorphism presented in section 2.1 is probably not zero-knowledge: a user interacting with the prover may use the prover in order to test to which of the given graphs (G_1 and G_2) is a third graph G_3 isomorphic. The way to fix this flaw, is to let

the verifier first "prove" to the prover that he "knows" an isomorphism between his query graph H and one of the input graphs. The modified protocol and the proof that it constitutes a zero-knowledge interactive-proof system, are omitted from this extended abstract. We get

Theorem 3: *There exist a zero-knowledge interactive proof system for Graph Non-Isomorphism.*

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 $x=y$.

We begin by presenting a zero-knowledge interactive proof for graph 3-colourability. Using this interactive proof and the power of NP-Completeness, we present zero-knowledge proofs for every language in NP. Finally, we show that "everything that is efficiently provable" can be proved in zero-knowledge.

3.1 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 ϕ ($\phi: V \rightarrow \{1,2,3\}$). Let $n=|V|$, $m=|E|$. For simplicity, let $V=\{1,2,...,n\}$.

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 a permutation $\pi \in_R Sym(\{1,2,3\})$, and random r_v's, computes $R_v = f(\pi(\phi(v)),r_v)$ (for every $v \in V$), and sends the sequence $R_1, R_2, ..., R_n$ to the verifier.
2) The verifier chooses at random an edge $e \in_R E$ 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 condition 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 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 $(1-m^{-1})^{m^2} = \exp(-m)$. Thus, the above protocol constitutes an interactive proof system for 3-colourability. Proving that the above protocol is zero-knowledge is even more involved that the proof of Theorem 2.

Proposition 4: *If $f(\cdot,\cdot)$ is a secure probabilistic encryption, then the above protocol constitutes a zero-knowledge interactive proof system for 3-colourability.*

proof's sketch: As in the proof of Theorem 2, we will present a machine M_{V^*} for every interactive machine V^*. Typically, we will try to guess which edge the machine V^* will ask to check. We will encrypt an illegal colouring of G such that we can answer V^* in case we were lucky. The cases in which we failed will be ignored. It is crucial that from the point of view of V^* the case which leads to our success and the case which leads to our failure are polynomially indistinguishable.

The machine M_{V^*} monitoring V^*, starts by choosing a random tape r for V^*. M_{V^*} places r on its record tape and proceeds in m^2 rounds as follows.

1) M_{V^*} picks an edge $(u,v) \in_R E$ and a pair of integers $(a,b) \in_R \{(i,j): 1 \leq i \neq j \leq 3\}$ at random. M_{V^*} chooses random r_i's and computes $R_i = f(c_i, r_i)$, where c_i is 0 for $i \in V - \{u,v\}$, $c_u = a$ and $c_v = b$. M_{V^*} places the sequence of R_i's on the communication tape of V^*.

2) M_{V^*} reads e from the communication tape of V^*. If $e \notin E$ (V^* obviously cheats) then M_{V^*} appends the R_i's and e to its record tape, outputs the record tape, and stops. If $e \neq (u,v)$ (unlucky for M_{V^*}) then M_{V^*} rewinds V^* to the configuration at the beginning of the current round, and repeats the current round with new random choices. If $e = (u,v)$ (lucky for M_{V^*}) then M_{V^*} proceeds as follows: First, it places (a, r_u) and (b, r_v) on the communication tape of V^*. Second, it appends the R_i's, e, (a, r_u) and (b, r_v) to its record tape; and finally, it proceeds to the next round.

If all rounds are completed then M_{V^*} outputs its record and halts. A technical lemma (to be stated and proved in the final paper) guarantees that the three possible "answers" of the verifier (i.e. $e \notin E$, $e \in E - \{(u,v)\}$ and $e = (u,v)$) occur with essentially the same probability as in the interaction of V^* and the real prover. Thus, the probability that the simulation of a particular round requires more than $k \cdot m$ rewinds is smaller than 2^{-k}, and M_{V^*} terminates in polynomial time. The only difference between the probability distribution of the true interactions and the distribution generated by M_{V^*} is that the first contain probabilistic encryptions of colourings while the second contains probabilistic encryptions of mostly 0's. However, a second technical lemma (postponed to the final paper) asserts that this difference is indistinguishable in probabilistic polynomial-time.

Remark 9: The above protocol needs m^2 rounds. In the final version of our paper we will present two alternative ways of modifying the above protocol so to get a four-round zero-knowledge protocol for graph 3-colorability. In both modifications the idea is to have the verifier commit himself to all his queries (i.e. which edge he wants to check for each copy of the coloured graph) before the prover sends to the verifier the corresponding coloured graphs. The two modifications differ by the manner in which the verifier commits to his queries. One modification is based on the intractability of factoring. The second modification is based on a relaxation of the definition of a proof system so that the prover is also restricted to polynomial-time (and his "computational advantage" over the verifier consists of an auxiliary input). This relaxation is natural in the cryptographic applications.

3.2 Zero-Knowledge Proofs for all NP

Incorporating the standard reductions into the protocol for graph 3-colourability, we get

Theorem 5: *If $f(\cdot,\cdot)$ is a secure probabilistic encryption, then every NP language has a*

zero-knowledge interactive proof system.

Slightly less obvious is the proof of the following Theorem 6 that adapts Theorem 5 to a cryptographic scenario in which all players are bounded to efficient computation. What is needed is to notice that the standard reductions transform *efficiently* also the solution to the instances.

Theorem 6: *If there exists a secure probabilistic encryption, then every language in NP has a zero-knowledge interactive proof system in which the prover is a probabilistic polynomial-time machine that gets an NP proof as an auxiliary input.*
(Namely, in case the common input x is in the language L, the polynomial-time prover gets an NP proof that $x \in L$ as an auxiliary input.)

Remark 10: The number of computational steps required by both parties in the above interactive proof is bounded by $O(T^2(n)\cdot F(n)\cdot \log^4 n)$, where $n = |x|$ is the length of the common input x, $T(n)$ is the number of steps required by a non-deterministic machine to accept x, and $F(n)$ is the number of steps require to encrypt a bit when the security parameter is n.

A positive use of NP-Completeness

So far NP-completeness have mostly had a "negative" utility: it was (and is) the most practical way to give evidence to the infeasibility of a problem. Here we want to point out a "positive" use of NP-completeness: its primary role in deriving the general results of Theorems 5 and 6 (i.e. zero-knowledge proofs of every NP statement) from Proposition 4 (i.e. a zero-knowledge proof of a particular NP-Complete problem).

An Example: Verifiable Secret Sharing

Due to its generality, Theorem 6 has a dramatic effect on the design of cryptographic protocols. Let us first demonstrate this point by using Theorem 6 to present a simple solution to a problem which until recently was considered very complex: *Verifiable Secret Sharing*. The more general implications of Theorem 6, are outlined in Section 4.

The notion of a verifiable secret sharing was presented by Chor, Goldwasser, Micali and Awerbuch [CGMA], and constitutes a powerful tool for multi-party protocol design. 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 $1 \leq l < u \leq n$):

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)).

Following the first implementation presented in [CGMA], improvements in efficiency and "tolerance" appeared in [FM, AGY, F]. These solutions are conceptually complicated, and rely on specific properties of particular encryption functions.

Assuming the existence of *arbitrary* one-way permutations, we present a conceptually simple solution allowing $u = l+1 \leq n$. Our scheme combines Theorem 6 with Shamir's (non-verifiable) secret sharing [Sha]. To share a secret $s \in Z_p$ recognizable through $r = g(s)$, the sender proceeds as follows: First, the sender chooses at random a l-degree polynomial over Z_p^* and evaluates it in n fixed points (these are the pieces in Shamir's scheme). Next, the sender encrypts the ith piece using the Public encryption algorithm of the ith receiver, and sends all encrypted secrets to all receivers. Finally, the sender

provides each receiver with a zero-knowledge proof that the encrypted messages correspond to the evaluation of a single polynomial over Z_p^*, and that applying g to the free term of this polynomial yields r (note that this is a NP statement).

3.3 Everything Efficiently Provable Can Be Proven in Zero-Knowledge

We now generalize Theorem 5 to show that not only NP is in zero-knowledge, but also "probabilistic NP" is. Namely,

Theorem 7: *If there exists a secure probabilistic encryption, then for every fixed k every language in $IP(k)$ has zero-knowledge proof systems.*

proof's sketch: Using the results of [GS] and [B], it suffices to demonstrate zero-knowledge proof systems for languages in $AM(2)$. The intuitive idea is to let the verifier send random coins and then let the prover prove that "he could have convinced the verifier with respect to these coins", which is an NP statement! To oblige the verifier to send random coins and not strings of his choice, coin flipping into the well [Blu] is used. It has to be proven however, that the substitution of certified random coins by coin flips into the well preserves zero-knowledge. ■

Recently, Ben-Or extended Theorem 7 and showed that every language which has an interactive proof system, has a zero-knowledge one [Ben]. As above, the result of Goldwasser and Sipser [GS] is used to restrict attention to languages in AM. This time we can not use Babai's result [B], since the number of interactions is unbounded. The idea is to first execute the AM protocol in an encrypted form (only the messages of the prover need to be encrypted and this does not disturb the verifier who only toss coins), and next have the prover convince the verifier in zero-knowledge that the encrypted interaction corresponds to an accepting interaction in the original AM protocol.

The following question was raised by Leonid Levin: Let M be a probabilistic polynomial-time interactive machine having access to a machine P_1 which is able to prove that $x \in L$ via an arbitrary *predetermined* interactive proof system. Can M prove that $x \in L$ to another machine V_2 in a zero-knowledge manner? Clearly the answer is negative if M first interact with P_1 and only later interact with V_2 (hint: P_1 may use a zero-knowledge proof system). However, M is allowed to interleave its interactions in an arbitrary manner. Theorem 6 answers Levin's question positively for the case that P_1 sends M an NP-proof. (In fact this was the motivation for his question.) It is easy to answer Levin's question positively for the case that P_1 interacts with M via an AM protocol. Recently, using a result of Yao [Y2], we have answered this question positively also for the general case (of IP protocols).

3.4 Related Results

Using the intractability assumption of *quadratic residuosity*, Brassard and Crepeau have discovered independently (but subsequently) zero-knowledge proof systems to all languages in NP [BC1]. These proof systems heavily rely on *particular properties of quadratic residues* and do not seem to extend to arbitrary encryption functions.

Recently, Brassard and Crepeau showed that *if factoring is intractable* then every NP language has a perfect zero-knowledge interactive proof system [BC2]. It should be stressed that the protocol they proposed constitutes an interactive proof provided that factoring is intractable. In other words, the validity of the interactive proofs depends on an intractability assumption; while in this paper and in [BC1] the validity of the proofs do not rely on such an assumption.

Independently, Chaum [Cha] discovered a protocol which is very similar to the one in [BC2]. Chaum also proposed an interesting application of such "perfect zero-knowledge proofs". His application is to a setting in which the verifier may have infinite computing power while the prover is restricted to polynomial-time computations (see also [CEGP]). In such a setting it makes no sense to have the prover demonstrate properties (as membership in a language) to the verifier. However, the prover may wish to demonstrate to the

verifier that he "knows" something without revealing what he "knows". More specifically, given a CNF formulae, the prover wishes to convince the verifier that he "knows" a satisfying assignment in a manner that would yield no information which of the satisfying assignments he knows. A definition of the notion of "a program knowing a satisfying assignment" can be derived from [GMR].

4. A Methodology of Cryptographic Protocol Design

Assuming the existence of arbitrary encryption functions, we will present extremely powerful methodologies for developing secure two-party and multi-party protocols. These methodology consists of efficient "correctness and privacy preserving" transformations of protocols from a weak adversary model to the most adversarial model. These (explicit) 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 cryptographic 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 any 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 the final version of this paper we will *define* the notions of a "correctness preserving compiler" and a "privacy preserving compiler". Both notions will be 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 participating in the original protocol P, he could also compute when following the transformed protocol P', properly. "Preserving privacy" means that whatever a set of dishonest players can compute after participating in P', the corresponding players in P can compute from their joint local "histories" 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 cryptographic two-party-protocol P' that achieves the same goals even if one of the players deviates from P' in any arbitrary but polynomial-time way.

The proofs of the above Theorems make primary use of Theorem 6 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 the 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 Improvement

Theorem A constitutes a procedure for automatically constructing fault-tolerant protocols, the goal of which is to compute a predetermine 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. Furthermore, for some functions f it seems to be difficult to write a distributed specification (protocol for honest players) which offers the maximum degree of privacy. Recently, assuming the exist of an arbitrary secure encryption scheme, we found a polynominal-time algorithm which on input a *Turing machine* specification of a n-ary function f, outputs a protocol for n honest players which offers the maximum possible privacy. Namely, 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. Essential ingredients in the algorithm are the "circuit encoding" of Barrington [Bar], a modification of the two-party protocol of Yao [Y2], and a general implementation of a variant of Oblivious Transfer using any encryption function. Details will appear in a forthcoming paper [GMW].

The algorithm claimed above can also be applied to any Turing machine specification of a probability distribution (depending on n variables). Equivalently, one can view the algorithm as a compiler that on input a n-party protocol (for honest players) outputs a fault tolerant n-party protocol, for computing the same distributed input-output relation, which offers the maximum degree of privacy. This compiler, which may increase the privacy present in the input protocol, improves on and uses as a subroutine the compiler of Theorem A (which only preserves the privacy present in the input). The compiler of Theorem A, in turn, improves on and uses as subroutine the compiler of Chor, Goldwasser, Micali and Awerbuch [CGMA].

ACKNOWLEDGEMENTS

We are very grateful to Benny Chor and Shafi Goldwasser for many discussions concerning methodologies for Cryptographic Protocol Design. We also wish to thank Baruch Awerbuch, Manuel Blum, Mike Fischer, Leonid Levin, Albert Meyer, Yoram Moses, Michael Rabin, Charlie Rackoff, Ron Rivest, and Mike Sipser for many helpful discussions concerning this work.

REFERENCES

[AHU] Aho, A.V., J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley Publ. Co., 1974.

[AGH] Aiello, W., S. Goldwasser, and J. Hastad, "On the Power of Interaction", these proceedings.

[AGY] Alon, N., Z. Galil, and M. Yung, "A Fully Polynomial Simultaneous Broadcast in the Presence of Faults", manuscript, 1985.

[B] Babai, L., "Trading Group Theory for Randomness", *Proc. 17th STOC*, 1985, pp. 421-429.

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

[Ben] Ben-Or, M., private communication, 1986.

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

[BH] Boppana, R., and J. Hastad, "Does Co-NP Have Short Interactive Proofs?", in preparation, 1986.

[BC1] Brassard, G., and C. Crepeau, "Zero-Knowledge Simulation of Boolean Circuits", manuscript, presented in *Crypto86*, 1986.

[BC2] Brassard, G., and C. Crepeau, "Non-Transitive Transfer of Confidence: A Perfect Zero-Knowledge Interactive Protocol for SAT and Beyond", these proceedings.

[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.

[Cha] Chaum, D., "Demonstrating that a Public Predicate can be Satisfied Without Reveal-

ing Any Information About How", manuscript, presented in *Crypto86*, 1986.

[CEGP] Chaum, D., J.H. Evertse, J. van de Graaf, and R. Peralta, "Demonstrating Possession of a Discrete Logarithm without Revealing It", manuscript, presented in *Crypto86*, 1986.

[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.

[CF] Cohen, J.D., and M.J. Fischer, "A Robust and Verifiable Cryptographically Secure Election Scheme", *Proc. 26th FOCS*, pp. 372-382, 1985.

[C] Cook, S.A., "The Complexity of Theorem Proving Procedures", *Proc. 3rd STOC*, pp. 151-158, 1971.

[F] Feldman, P., "A Practical Scheme for Verifiable Secret Sharing", manuscript, 1986.

[FM] Feldman, P., and S., Micali, in preparation, 1985.

[FMRW] Fischer, M., S. Micali, C. Rackoff, and D.K. Wittenberg, "An Oblivious Transfer Protocol Equivalent to Factoring", in preparation. Preliminary versions were presented in *EuroCrypt84* (1984), and in the *NSF Workshop on Mathematical Theory of Security*, Endicott House (1985).

[GHY] Galil, Z., S. Haber, and M. Yung, "A Private Interactive Test of a Boolean Predicate and Minimum-Knowledge Public-Key Cryptosystems", *Proc. 26th FOCS*, 1985, pp. 360-371.

[GJ] Garey, M.R., and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, New York, 1979.

[G] Goldreich, O., "A Zero-Knowledge Proof that a Two-Prime Moduli Is Not a Blum Integer", unpublished manuscript, 1985.

[GMW] Goldreich, O., S. Micali, and A. Wigderson, "How to Automatically Generate Correct and Private Fault-Tolerant Protocols", in preparations.

[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.

[GS] Goldwasser, S., and M. Sipser, "Arthur Merlin Games versus Interactive Proof Systems", *Proc. 18th STOC*, 1986, pp. 59-68.

[K] Karp, R.M., "Reducibility among Combinatorial Problems", *Complexity of Computer Computations*, R.E. Miller and J.W. Thatcher (eds.), Plenum Press, pp. 85-103, 1972.

[L] Levin, L.A., "Universal Search Problems", *Problemy Peredaci Informacii 9*, pp. 115-116, 1973. Translated in *problems of Information Transmission 9*, pp. 265-266.

[Sha] Shamir, A., "How to Share a Secret", *CACM*, Vol. 22, 1979, pp. 612-613.

[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", these proceedings.