Multi-Prover Interactive Proofs: How to Remove Intractability Assumptions

Michael Ben-Or* Hebrew University
Shafi Goldwasser† MIT
Joe Kilian‡ MIT
Avi Wigderson§ Hebrew University

Abstract

Quite complex cryptographic machinery has been developed based on the assumption that one-way functions exist, yet we know of only a few possible such candidates. It is important at this time to find alternative foundations to the design of secure cryptography. We introduce a new model of generalized interactive proofs as a step in this direction. We prove that all NP languages have perfect zero-knowledge proof-systems in this model, without making any intractability assumptions.

The generalized interactive-proof model consists of two computationally unbounded and untrusted provers , rather than one, who jointly agree on a strategy to convince the verifier of the truth of an assertion and then engage in a polynomial number of message exchanges with the verifier in their attempt to do so. To believe the validity of the assertion, the verifier must make sure that the two provers can not communicate with each other during the course of the proof process. Thus, the complexity assumptions made in previous work, have been traded for a physical separation between the two provers.

We call this new model the multi-prover interactive-proof model, and examine its properties and applicability to cryptography.

*Supported by Alon Fellowship.
†Supported in part by NSF grant 865727-CCR, ARO grant DAAL03-86-K-017, and US-Israel BSF grant 86-00301, Jerusalem, Israel.
‡Supported by a Fannie and John Hertz Foundation fellowship.
§Supported by Alon Fellowship

1 Introduction

The notion of randomized and interactive proof system, extending NP, was introduced in [GMR] and in [B]. An interactive proof-system consists of an all powerful prover who attempts to convince a probabilistic polynomial-time bounded verifier of the truth of a proposition. The prover and verifier receive a common input and can exchange upto a polynomial number of messages, at the end of which the verifier either accepts or rejects the input. Several examples of interactive proof-system for languages not known to be in NP (e.g graph non-isomorphism) are known.

In [GMW1] Goldreich, Micali and Wigderson show the fundamental result that that if "non-uniform" one-way functions exist (i.e no small circuits exist for the function inverse computation), then every NP language has a computationally zero-knowledge interactive proof system. This has far reaching implications concerning the secure design of cryptographic protocols. It also seems to be the strongest result possible. Results in [F] and [BHZ] imply that if perfect zero-knowledge interactive proof-systems for NP exist, (i.e which do not rely on the fact that the verifier is polynomial time bounded) then the polynomial time hierarchy would collapse to its second level. This provides strong evidence that it will be impossible (and at least very hard) to unconditionally show that NP has zero-knowledge interactive proofs.

In light of the above negative results, it is interesting to examine whether the definition of interactive proofs can be modified so as to still capture the notion of efficient provability and yet allow perfect zero-knowledge proofs for NP, making no intractability assumptions.

This is particularily important from a cryptographic view point, as the possible one-way functions currently considered are very few and almost exclusive to number theory (e.g. integer factorization, discrete logarithm computation and elliptic logarithm computation.) If these were found to be efficiently solvable, the cryptographic consequences of the [GMW] result would be unusable.

1.1 New Model

We extend the definition of an interactive proof for language L as follows: instead of one prover attempting to convince a verifier that x, the input string, is in L, our prover consists of two separate agents (or rather two provers) who jointly attempt to convince a verifier that x is in L. The two provers can cooperate and communicate between them to decide on a common optimal strategy before the interaction with the verifier starts. But, once they start to interact with the verifier, they can no longer send each other messages or see the messages exchanged between the verifier and the "other prover". As in [GMR] the verifier is probabilistic polynomial time, and can exchange upto a polynomial number of messages with either one of the two provers (with no restriction on interleaving the exchanged messages) before deciding to accept or reject string x.[1]

We restrict the verifier to send messages to the prover in a predetrmined order. It can be shown that this is equivalent with respect to language recognition, to a model in which the verifier is free to talk to the provers in any order he wishes. Moreover, the verifier can be forced to send messages to the provers in a predetermined order by using a simple password scheme. Thus, we can work in the easier to deal with synchronous model completely without loss of generality.

The main novelty of our model is that the verifier can "check" its interactions with the provers "against each other". One may think of this as the process of checking the alibi of two suspects of a crime (who have worked long and hard to prepare a joint alibi), where the suspects are the provers and the verifier is the interrogator. The interrogators conviction that the alibi is valid, stems from his conviction that once the interrogation starts the suspects can not talk to each other as they are kept in separate rooms, and since they can not anticipate the randomized questions he may ask them, he can trust his findings (i.e receiving a correct proof of the proposition at hand).

Applying this model in a cryptographic scenario, one may think of a bank customer holding two bank-cards rather than one, attempting to prove its identity to the bank machine. The machine makes sure that once the two cards are inserted they can no longer communicate with each other. In this scenario, the provers correspond to the two cards, and the verifier to the bank machine.

1.2 Results

1.2.1 Perfect Zero Knowledge Multi-Prover Interactive Proofs

We show, that in our extended model all NP languages have a perfect zero-knowledge interactive proof-system, making no intractability assumptions.

The protocol for NP languages proposed, requires the two provers to share either a polynomially long random pad or a function which they can compute but the polynomially bounded verifier can not. It is well known that such functions exist by counting arguments. Most of the burden of the proof lies on one predetermined prover. In fact, the "other" prover sole function is to periodically output segments of the random pad he shares with the "primary prover". The protocol is constant (two) round.

Differently then in the case of the graph non-isomorphism and quadratic non-residousity proof-systems in [GMR], [GMW], parallel executions of the protocol remain perfect zero-knowledge.

More generally, we show that any language which can be recognized in our extended model, can be recognized in perfect zero-knowledge making no intractability assumptions.

Our construction does not assume that the verifier is polynomial time bounded. The assumption that there is no communication between the two

[1] A proof-system for a language in this model is defined in a similar manner to [GMR]. Namely, L has a multi-prover interactive proof-system if there exist a verifier V and provers P1, P2 such that when $x \in L$ the probability that V accepts is greater than 2/3, and when x is not in L then for all P1, P2 the probability that V accepts is less than 1/3.

provers while interacting with the verifier, must be made in order for the verifier to believe the validity of the proofs. It need not be made to show that the interaction is perfect zero-knowledge.

1.3 Language Recognition Power of New Model

It is interesting to consider what is the power of this new model solely with respect to language recognition. Clearly, $NP \subseteq IP$ which in turn is a subset of languages accepts by our extended model. We show that adding more provers than two, adds no more power to the model.

We also show for every language possessing a two prover interactive proof there exists another two prover interactive proof which achieves completeness, i.e. the verifier will always accept strings which are in the language.

Fortnow, Rompel and Sipser [FRS] have shown that two provers can accept any language in IP (one-prover model with polynomial number of rounds) using only a constant number of rounds. They also show that three provers can accept in a constant number of rounds all languages recognized by a multi prover model.

Feige, Shamir and Tennenholtz [FST] look at a model they call the k-noisy oracle model, in which the verifier is interacting with k oracles all of which but one may be dishonest. Based on the assumption that one of the oracles is trusted, they show that P-space langauages can be recognized in a 2-noisy oracle model.

1.4 Open Problem

Whether the two-prover proof-system is actually more powerful with respect to language recognition than the original one-prover interactive proof-system of [GMR],[B], remains an open problem.

Even the simplest case of two-round two-prover proof-system in which the verifier sends the result of his coin tosses first (some to prover 1 and some to prover 2), receives responses (from both provers) on the subsequent round, and then evaluates a polynomial time predicate to decide whether to accept or reject, is not known to lie in PSPACE. Hastad and Mansour [HM] show that resolving this question in the positive will imply that $NP \neq poly(\log) - SPACE$.

2 Definitions

Definition 1: Let P_1, P_2,..., P_k be Turing machines which are computationally unbounded and V be a probabilistic polynomial time Turing machine. All machines have a read-only input tape, a work tape and a random tape. In addition, P_1, P_2,.., P_i share an infinite read-only random tape of 0's and 1's. Every P_i has one write-only communication tape on which it writes messages for V. V has k write-only communication tapes. On communication tape i, V writes messages to P_i. We call $(P_1, P_2, ..., P_k, V)$ a ***k-prover interactive protocol.***

Remark 1: Fortnow, Rompel and Sipser [FRS] remark that the above can be modeled as a probabilistic polynomial time Turing machine V and an oracle p such that queries to p are prefixed always by $1 \leq i \leq k$, corresponding to whether the query is directed to prover i. Each query contains the history of the communication thus far.

We note that although this memoryless formulation is equivalent to the i-prover formulation with respect to language recognition, it is not equivalent when zero-knowledge is considered. In this latter case the provers must be able to check that the history is indeed what is claimed by the verifier, before answering the next query. Since the verifier is not untrusted, the provers can not be memoryless.

Definition 2: Let $L \subset \{0,1\}^*$, We say that L has a ***k-prover interactive proof-system***(IPS) if there exists an interactive BPP machine V such that:

1 $\exists P_1, P_2, ..., P_k$ such that $(P_1, P_2, ..., P_k, V)$ is a k-prover interactive protocol and $\forall x \in L$, prob(V accepts input x) $\geq \frac{2}{3}$.

2 $\forall P_1, P_2, ..., P_k$ such that $(P_1, P_2, ..., P_k, V)$ is a k-prover interactive protocol, prob(V accepts input x) $\leq \frac{1}{3}$.

Remark 2: if L has an k-prover interative proof-system and condition (1) holds for a particular $\hat{P}_1$ $\hat{P}_2$, ...,$\hat{P}_k$, then we say that $(\hat{P}_1, \hat{P}_2, \hat{P}_k, V)$ is a k-prover interactive proof-system for L.

Remark 3: if L has an two-prover interative proof-system, then L has a two-prover interactive proof-systems (P_1, P_2, V) such that for $x \in L$, prob(V accepts x) $= 1$. See Theorem 5.

Remark 4: For convenience, without loss of generality, we assume that every verifier V outputs his coin tosses at the end of his interaction with the P_i's.

Definition 3: Let $IP_k = \{L$ which have k-prover interactive proof-system $\}$.

The following definition of perfect zero-kowledge is identical to the Goldwasser-Micali-Rackoff [GMR] definition of perfect zero-knowledge in the 1-prover model.

Definition 4: Let $(P_1, P_2, ..., P_k, V)$ be a k-prover interactive proof-system for L. Let $View_{P_1,P_2,...,P_k,V}(x)$ denote the verifier's view during the protocol (namely the sequence of messages exchanged between the verifier and the two provers including the last message of the verifier which contains his coin tosses – see remark 4 above). This is a probability space taken over the coin tosses of V and the joint random tape of $P_1, P_2, ..., P_k$. We say that k-prover interactive protocol $(P_1, P_2, ..., P_k, V)$ is *perfect zero-knowledge for* V if there exists a BPP machine M such that $M(x) = View_{P_1,P_2,...,P_k,V}(x)$. We say that L has a *k-prover perfect zero-knowledge proof-system* if there exists provers P_1, P_2,...,P_k such that for all BPP verifiers $\hat{V}$, there exists a probabilistic Turing machine M such that for all x in L, $M(x) = View_{P_1,P_2,...,P_k,\hat{V}}(x)$ and $M(x)$ terminates in expected polynomial time.

3. Statement of our Results

Theorem 1: Every $L \in NP$ has a two-prover perfect zero-knowledge interactive proof-system.

Proposition 1: parallel executions of the perfect zero-knowledge interactive proof-system for NP remain perfect zero-knowledge.

Theorem 2: Every $L \in IP_2$ has a perfect zero-knowledge interactive proof-system.

Theorem 3: Any two party oblivious function computation can be done in this model.

Theorem 4: For all $k \geq 2$, if $L \in IP_k$ then $L \in IP_2$.

Theorem 5: If $L \in IP_2$ then $\exists P_1, P_2, V$ such that (P_1, P_2, V) is a two-prover interactive proof-system for L and for all $x \in L$, Prob(V accepts x) = 1.

3 Key Ideas

A general primitive used in complexity based cryptography (and in particular in the proof that NP is in zero-knowledge under the assumption that one-way functions exist)is the ability to encrypt a bit so that the decryption is unique. In our model, encryption is replaced by a commitment protocol to a bit such that the bit is *equally likely* to be 0 or 1 (information theoretically), and yet the probability that a different bit can be decommited (i.e revealed) is less than $\frac{1}{2}$(this fraction can then be made arbitrarily small using standard techniques). The idea is that one prover is used to commit the bit, and the other to reveal it.

Another important primitive is that of oblivious circuit evaluation. This primitive allows two parties, A and B, possessing secrets i and j respectively, to compute some agred upon function $f(i, j)$ in such a way that A learns nothing, and B learns only $f(i, j)$. The original implementation of this protocol, due to Yao [Yao86a], requires the existence of trapdoor functions. In fact, oblivious circuit evaluation can not be implemented without cryptographic assumptions in the standard two party scenario. However, we show that oblivious circuit evaluation between verifier and 1 prover can be done without assumptions in the two-prover model. The proof relies on a result of [K] reducing oblivious circuit evaluation to a simpler protocol, known as 1-out-of-2 oblivious transfer, which was reduced by [C] to a still simpler protocol, known as oblivious transfer. This last protocol is implemented in the two-prover model.

4 Proof of Theorem 1: How to Commit Bits

We first show that every language in NP has a perfect zero-knowledge two-prover interactive proof-system.

Theorem 1: Every L in NP has a two-prover perfect zero-knowledge interactive proof-system.

Idea of Proof:

Let (P_1, P_2, V) denote a multi-prover protocol which receives as input the graph $G = (\mathcal{V}, \mathcal{E})$. Let P_1 and P_2 share an infinite random pad R such that $R = r_1 r_2 ... r_k ...$ where $r_i \in \{0, 1, 2\}^2$. Let

[2] Alternatively, R can be replaced by the outcome of $f(x)$ where x is the input and $f : \{0,1\}^* -> \{0,1\}^*$ is a function such that for all $x \in \{0,1\}^*$, for all $i < |f(x)|$, the i-th bit of $f(x)$ is equally likely to be 0 or 1 with respect to any probabilistic polynomial time machine. Such functions can be shown to exist by standard diagonalization techniques over all probabilistic polynomial time machines.

$n = |V|$.

Let us quickly review[3] one of the, by now standard proofs ([GMW1], [Bl]) that NP is in zero-knowledge under the assumption that one-way functions exist.

Review: The prover is attempting to convince the verifier that G is Hamiltonian. The prover publicizes an probabilistic encryption algorithm E (as in [GM], [Yao82a])[4] The prover and verifier repeat the following protocol n times:

STEP 1:prover randomly permutes the vertices of graph G (using permutation π) to obtain graph $\hat{G}$ and sends to verifier

- an $n \times n$ matrix $\alpha = \{\alpha_{ij}\}$ where α_{ij} in $E(b_{ij})$ and $b_{ij} = 1$ if edge ij is present in the $\hat{G}$ and 0 otherwise.
- $\beta \in E(\pi)$, i.e an encryption of π.

STEP 2:verifier chooses at random $coin \in \{0,1\}$, and sends $coin$ to the prover.

STEP 3: If $coin = 1$, prover decrypts β and α_{ij} for all $i, j \leq n$ and sends decryptions to verifier. If $coin = 0$, prover decrypts those α_{ij} such that edge ij is in the Hamiltonian path in $\hat{G}$.

STEP 4: If prover is unable to preform step 3 correctly, verifier rejects. Otherwise, after n iterations of steps 1 through 4, verifier accept.

End of Review

Returning to the two prover model, prover P_1 replaces the prover in step 1 of above protocol and prover P_2 replaces the prover in step 2 of above protocol. Algorithm E is no longer a probabilistic encryption algorithm based on the existence of one-way functions as in [GM] or [Yao86a], but rather a commitment algorithm computed as follows.

Let $\sigma_0, \sigma_1 : \{0,1,2\} -> \{0,1,2\}$ be such that

(1) for all i, $\sigma_0(i) = i$,

(2) $\sigma_1(0) = 0, \sigma_1(1) = 2$ and $\sigma_1(2) = 1$.

Let m_k be the k-th bit to be committed to in the protocol.

<u>To commit m_k :</u>

- V chooses at random $c_k \in \{0,1\}$ and sends c_k to P_1.
- P_1 sets $E(c_k, m_k) = \sigma_{c_k}(r_k) + m_k \bmod 3$, where $r_k \in \{0,1,2\}$ is read off the random tape P_1 shares with P_2, and sends $E(c_k, m_k)$ to V.

To reveal the k-th bit committed in the protocol, V and P_2 engage in the following protocol.

<u>To reveal the k-th bit:</u>

- V sends k to P_2.
- P_2 sends V the string r_k.
- V computes $\sigma_{c_k}(r_k)$ and sets m_k to $(E(c_k, m_k) - \sigma_{c_k}(r_k)) \bmod 3$.

Note: P_2 does not know c_k and has never seen $E(c_k, m_k)$.

We prove two properties of the above pair of commit-reveal protocols. First, since P_2 sees neither $E(c_k, m_k)$ nor c_k, but knows exactly what P_1's program is, the probability that P_2 successfully reveals a bit value different than the one P_1 committed to is less than $\frac{1}{2}$.

Claim 1.1: $\forall r \in \{0,1,2\}, m \in \{0,1\}$,

$$\text{prob}(\ \hat{r} \text{ is s.t. } E(c,r,m) = E(c,\hat{r},\overline{m})) \leq \frac{1}{2}$$

Comment: To decrease the probability of successfuly cheating from $\frac{1}{2}$ to $\frac{1}{2^n}$, P_1 preform n commits to m_k and P_2 preforms n reveals correspondingly.

Knowing k, $E(c_k, m_k)$ and c_k gives the verifier no advantage in guessing m_k.

Claim 1.2: $\forall c \in \{0,1\}$,

$$\text{prob}(\ m = 0 | E(c,r,m)) = \text{prob}(\ m = 1 | E(c,r,m)) = \frac{1}{2}$$

Proving now that the altered mutli-prover Hamiltonian cycle protocol constitutes a two-prover interactive proof for the Hamiltonian cycle problem follows directly from [Bl]'s proof and claim 1.

Proving that the protocol is perfect-zero-knowledge is more subtle.

To this end, we exhibit a probabilistic Turing machine M such that

[3] the proof reviewed is from [Bl]

[4] The encryption algorithm E is public. We denote $\gamma \in E(m)$ to mean that there exists string r such that algorithm E using r for his coin tosses, on input m, produces γ. Given γ there exists unique m, r such that E, on coin tosses r and input m outputs γ. To decrypt γ both m, r are revealed.

- for Hamiltonian graphs G, $M(G)$ terminates in expected polynomial time.
- for all $\hat{V}$ such that $(P_1, P_2, \hat{V})$ is a two-prover protocol, and for all Hamiltonian graphs G, $M(G) = View_{P_1,P_2,\hat{V}}$. (where P_1, P_2 are honest provers as specified above.)

WLOG let the number of coin tosses of verifier and prover on input $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $|\mathcal{V}| = n$ be be bounded by polynomial $Q(n)$.

Simulator M program: (tailored after steps 1-4 above in [Bl]'s proof)

STEP 1: M chooses $\rho \in \{0,1\}^{Q(n)}$ at random for the coin tosses to be used by $\hat{V}$. and sets $R = r_1 r_2 \ldots r_k \ldots$, $|R| \in \{0,1\}^{Q(n)}$ where $r_k \in \{0,1,2\}$ are chosen at random. ($\hat{V}(\rho, G)$ will denote the program $\hat{V}$ on input G and coin tosses ρ.) M picks a random permutation π of the vertices of graph G to obtainthe permuted graph $\hat{G}$ and an $n \times n$ random binary matrix MAT. Next, M simulates a commitment protocol to π and MAT as follows. To simulate a commitment protocol to the k-th bit m: M runs $\hat{V}(\rho, G)$ to obtain c, computes $E(c,m) = \sigma_{c_k}(r_k) + m \bmod 3$ for $r_k \in R$, and writes $E(c,m)$ on $\hat{V}(\rho, G)$'s tape.

STEP 2: M continues running $\hat{V}(\rho, G)$ to obtain *coin*.

STEP 3: if *coin* $= 1$, M reveals π (as P_2 would do in real protocol) by writing the appropriate $r \in R$ on $\hat{V}(\rho, G)$'s tape. Revealing MAT to V is more involved, as follows. Let $MAT = \{m_{ij} | 1 \le i,j \le n\}$ and $\alpha = E(c, m_{ij}) = \sigma_c(r) + m_{ij} \bmod 3$ where $r \in R$ is the r used in step 1 to commit m_{ij}. Let $\hat{r}$ be such that $\alpha = \sigma_c(\hat{r}) + \overline{m}_{ij} \bmod 3$. Note that such $\hat{r}$ always exists and since M knows c (differently from P_2 in the real protocol) M can compute it. Set

$$\check{r} = \begin{cases} r & \text{if } m_{ij} = 1 \text{ and } ij \text{ is an edge of } \hat{G}, \\ & \text{or } m_{ij} = 0 \text{ and } ij \text{ is not an edge of } \hat{G} \\ \hat{r} & \text{if } m_{ij} = 0 \text{ and } ij \text{ is an edge of } \hat{G}, \\ & \text{or } m_{ij} = 1 \text{ and } ij \text{ is not an edge of } \hat{G} \end{cases}$$

Then M reveals $\check{r}$ to $\hat{V}(\rho, G)$.

If *coin* $= 0$, M selects n ij entries at random in MAT such that no two entries are in the same column or in the same row. Set

$$\check{r} = \begin{cases} r & \text{if } m_{ij} = 1 \\ \hat{r} & \text{if } m_{ij} = 0 \end{cases}$$

Where again $r \in R$ from step 1 such that $\alpha = E(c, m_{ij}) = \sigma_c(r) + m_{ij} \bmod 3$, and $\hat{r}$ is such that $\alpha_{ij} = \sigma_c(\hat{r}) + \overline{m}_{ij} \bmod 3$. Next, M reveals $\check{r}$ to $\hat{V}(\rho, G)$. Finally, M sets $\check{R}$ to be R with the values of $\check{r}$ substituted for r used to commit the matrix MAT.

STEP 4: M runs $\hat{V}$ to either accept or reject. It then outputs the transcript of its exchanges with $\hat{V}$ followed by $\check{R}$. DONE

It is clear that, M on G operates in polynomial time in the running time of $\hat{V}$. Since $\hat{V}$ is assumed to be probabilistic polynomial time, so is M.

To show that the probability space generated by M is identical to that in $View_{(P_1,P_2,\hat{V})}$, we notice that for fixed ρ (coin tosses of the verifier) and fixed $\check{R}$ (joint random tape of P_1 and P_2) the output of $M(G)$ is identical to $View_{(P_1,P_2,\hat{V})}$. This is so as M actually runs $\hat{V}$ to obtain his moves and therefore $\hat{V}$'s moves are guaranteed to be perfectly simulated, while M itself follows the moves P_1, P_2 would have made on joint random tape $\check{R}$. Since ρ was picked by M at random at step 1, it remains to argue that the probability that $\check{R}$ was chosen by P_1 and P_2 is the same as the probability that $\check{R}$ was output by M. This is trivially true by claim 1.2. ∎

We claim, without proof here, that independent executions of the above protocol for any language $L \in NP$ can be performed in parallel and the resulting protocol will still be a 2-prover perfect zero-knowledge proof-system for L.

In the 1-prover model the question of whether it is possible in general to preform parallel executions of perfect zero-knowledge protocols maintaining perfect zero-knowledge is unresolved. In particular, it is not known how to parallelize the proof-systems for quadratic residuosity and graph isomorphism.

5 Proof of Theorem 4: $IP_k = IP_2$ for all $k \geq 2$

We now show that any k-prover $(P_1, \ldots, P_k, V)$ interactive proof-system for language L can be converted into a 2-prover $(\hat{P}_1, \hat{P}_2, \hat{V})$ interactive proof-system. The idea is as follows.

Verifier $\hat{V}$ tosses all his coins and sends them to prover $\hat{P}_1$. In return, $\hat{P}_1$ sends $\hat{V}$ the entire history of communication that would have occured for theses coin tosses between the real verifier V and the k real provers P_i's. If this is an accepting conversation for V, $\hat{V}$ now uses $\hat{P}_2$ to check the validity of the conversation. This is done by $\hat{V}$ selecting at random an original prover P_i, and simulating with $\hat{P}_2$ the conversation between V and P_i on these coin tosses. If the conversation does not match the conversation sent by $\hat{P}_1$ then $\hat{V}$ rejects, otherwise the protocol is repeated k times (in series) and finally $\hat{V}$ accepts.

Note that the number of rounds in the simulating protocol is k^2t, where t is the number of rounds in the k-prover interactive proof-system. Fortnow, Rompel and Sipser in [FRS] show that for each $L \in IP_2$, there exists a 3-prover IPS for L with only a constant number of rounds.

Theorem 4: Let $k \geq 2$. If $L \in IP_k$ then $L \in IP_2$.
proof: Let L have a k-prover interactive proof-system $(P_1, ..., P_k, V)$. Let $I_k = \{1, 2, ..., k, \$\}$ and r denote the coin tosses made by the verifier. For a $w \in L$, the optimal provers $P_1, ..., P_k$ and the verifier V can be thought of as deterministic functions $P_i : \Sigma^* \to \Sigma^*$ and $V : \Sigma^* \times I_k \times \Sigma^* \to \Sigma^* \cup \{accept, reject\}$ such that $y_j{}^i = P_i(h_{j-1}{}^i \# x_j^i)$ denotes the j-th message of the i-th prover to the verifier, $x_j{}^i = V(r, i, h_{j-1}^1, .., h_{j-1}^k)$ denotes the j-th message of the verifier to the i-th prover, and $h_j{}^i = \# x_1^i \# y_1^i \# ... \# x_j{}^i \# y_j{}^i$ denotes the history of communication as prover i sees it at round j. Let t the total number of rounds, then $V(r, \$, h_t^1, ..., h_t^k) \in \{accept, reject\}$. Let Q be a polynomial such that $|r|, |x_j^i|, |y_j^i| < Q(|w|)$.

We now define provers $\hat{P}_1$ and $\hat{P}_2$ and verifier $\hat{V}$ in the simulating two-prover protocool $\hat{P}_1, \hat{P}_2, \hat{V}$.

On input w,

STEP 1: $\hat{V}$ chooses $r \in \{0,1\}^{Q(|w|)}$ at random, sends r to $\hat{P}_1$.

STEP 2: $\hat{P}_1$ sends $h_t{}^1, ..., h_t{}^k$ to $\hat{V}$ where the h_t^i's are computed according to functions $P_1, ..., P_k$ and V. If $V(r, \$, h_t{}^1, ..., h_t{}^k) = reject$ then $\hat{V}$ rejects and halts. Otherwise $\hat{V}$ picks $1 \leq i \leq k$ at random, sets $j = 1$ and continues.

STEP 3: $\hat{V}$ sends $u_j^i = V(r, i, \hat{h}_{j-1}^i)$ to $\hat{P}_2$, where $\hat{h}_j^i = \# u_1^i \# v_1^i \# ... \# u_j^i \# v_j^i$ for $j \leq t$. if $j = t$ and $\hat{h}_t^i = h_t{}^i$ then $\hat{V}$ accepts and halts , otherwise $\hat{V}$ rejects and halts.

STEP 4: $\hat{P}_2$ sends $v_j^i = P_i(h_{j-1}^i \# u_j^i)$ to $\hat{V}$. Set $j = j + 1$ and GOTO STEP 3.

claim 5.1: $\forall w \in L$,

$$\text{prob(} \hat{V} \text{ accepts } w) = \text{prob(} V \text{ accepts } w)$$

proof: If $\hat{P}_i$ follow the protocol as described above and compute the $h_t{}^i$ according to the functions of the corresponding P_i's, then for every sequence of coin tosses r on which V would accept so would $\hat{V}$.

claim 5.2: if $w \notin L$, prob(V accepts w) $\leq$ (prob(V accepts w) $+ e^{-k}$.

proof: Assume $w \notin L$. Then, the prob($\hat{V}$ accepts w) $\leq$ prob($\hat{V}$ accepts $w | \forall i \leq k \forall j \leq t, y_j^i = \hat{P}_i(h_{j-1}^i)$)+ prob($\hat{V}$ accepts $w | \exists l, j$ s.t. , $y_j^l \neq \hat{P}_l(h_{j-1}^l)$) $\leq$ prob(V accepts w)+ prob($V(r, \$, h_t{}^1, ..., h_t{}^k) =$ *accept*, and
$\exists l \leq k$, s.t. $h_t{}^l \neq \hat{h}_t^l$, but i of step 4 is s.t. $h_t{}^i = \hat{h}_t^i$) $\leq$ prob(V accepts w) $+ (1 - \frac{1}{k})$.

If the above protcol is repeated k^2 independent times, the probability of success is reduced to prob(V accepts w) $+ (1 - \frac{1}{k})^{k^2} \leq$ prob(V accepts w) $+ e^{-k}$.

This completes the proof, and L is indeed in IP_2. ▮

6 Proof of Theorem 5: Completeness

Goldreich, Mansour and Sisper [GMS] showed that any $L \in IP$ has an interactive proof-system for which strings in L are always accepted. We show the corresponding property for any $L \in IP_2$.

Theorem 5: If $L \in IP_2$, then there exists a 2-prover interactive proof-system (P_1, P_2, V) for L such that for all $x \in L$, prob(V accepts) $= 1$.

proof: Suppose (P_1, P_2, V) is a 2-prover interactive proof-system for L such that $\epsilon =$ prob(V accepts $\{w$ not in $L\}$ and the number of coin tosses on input w which V makes is a polynomial $Q(|w|)$. We show a simulating 2-prover interactive proof-system $(\hat{P}_1, \hat{P}_2, V)$ for L which also achieves completenes. The simulation is done in two stages. In stage 1, we use the idea of the completeness proof for the 1-prover interactive proof-system model by Goldreich, Mansour and Sisper in [MGS] (based on Lautman's Lemma) where $\hat{P}_1$ plays the part of both P_1 and P_2. In stage 2, as in the proof of the theorem of section 6, V uses $\hat{P}_2$ to check the validity of stage 1.

Let t denote the number of rounds in (P_1, P_2, V). Again, consider P_1, P_2 and V as determinstic functions as in the proof of theorem of section 6.

Let r denote the coin tosses of the verifier. For $i = 1, 2$, let $h_t^i(r) = \# x_1^i \# y_1^i \# ... \# x_t^i \# y_t^i$ where $x_j^i = V(r, i, h_{j-1}^i(r))$, and $y_j^i = P_i(h_{j-1}^i(r) \# x_j^i)$.

Define $W = \{r | V(r, \$, h_t^1, h_t^2) = accept\}$. Note that for $w \in L$, $\frac{|W|}{2^{Q(|w|)}} \geq (1 - \epsilon)$ and for w not in L $\frac{|W|}{2^{Q(|w|)}} \leq \epsilon$. Lautman[L] shows that $\forall w \in L \exists s_1, ..., s_{Q(|w|)}, |s_i| = Q(|w|), s.t. \forall r, |r| = Q(|w|), \exists l$ s.t. $r \oplus s_l \in W$. We use this in a manner similar to [GMS].

On input w,

STEP 1: $\hat{P}_1$ sends V $s_1, ..., s_{Q(|w|)}$ such that $s_i \in \{0, 1\}^{Q(|w|)}$

STEP 2: $\hat{V}$ sends r to $\hat{P}_1$ where r is randomly selected in $\{0, 1\}^{Q(|w|)}$

STEP 3: $\hat{P}_1$ sends to $\hat{V}$, $h_t^i(s_j \oplus r)$ for $i = 1, 2$ and $1 \leq j \leq Q(|w|)$. (These are the histories of conversations which would have been exchanged in original protocol (P_1, P_2, V) on coin tosses $r \oplus s_j, 1 \leq j \leq Q(|w|)$.)

STEP 4: if $V(r \oplus s_j, h_t^1(r \oplus s_j), h_t^2(r \oplus s_j)) = reject$ for all $1 \leq j \leq k$, then $\hat{V}$ rejects. If $\exists l$ s.t. $V(r \oplus s_l, h_t^1(r \oplus s_l), h_t^2(r \oplus s_l)) = accept$, then goto STEP 5.

STEP 5: $\hat{V}$ chooses $i \in \{1, 2\}$ at random. It then interacts with prover $\hat{P}_2$ in the same way that V and P_i would have on coin tosses $r \oplus s_l$. If this interaction produces exactly the same history string $h_t^i(r \oplus s_l)$ sent by $\hat{P}_1$ in STEP 3 then $\hat{V}$ accepts, otherwise it rejects.

The above protocol is repeated $Q(|w|)s$ times, and the verifier accepts if and only if he accepeted in any of these iterations.

Claim 1: prob(V accepts $|w| \in L) = 1$

proof: if $\hat{P}_1$, and $\hat{P}_2$ follow the program outlined above, follows directly from [L] and [GMS].

Claim 2: prob(V accepts $|w|$ not in $L) \leq \frac{1}{3}$

proof: We now can not assume that $\hat{P}_1, \hat{P}_2$ follow the protocol. Let h_{ij}, for $i = 1, 2$, $1 \leq j \leq Q(|w|)$ denote the strings sent by $\hat{P}_1$ in STEP 3.

prob(V accepts in one iteration $|w \notin L) \leq \sum_l$ prob($\exists l, V(r \oplus s_l, h_{1l}, h_{2l}) = accept | \hat{P}_1, \hat{P}_2$ honest) +prob($\hat{P}_1, \hat{P}_2$ not caught in step 5 but $\exists j, i, h_t^i(r \oplus s_j) \neq h_{ij}) \leq Q(|w|) \cdot \epsilon + (1 - \frac{1}{2Q(|w|)}) = 1 - \frac{1}{2Q(|w|)} + Q(|w|) \cdot \epsilon$

Now, prob(V accepts in $Q(|w|)^3$ iterations $|w \notin L) = (1 - \frac{1}{2Q(|w|)} + Q(|w|) \cdot \epsilon)^{Q(|w|)^3}$ which is less than a 1/3 for ϵ sufficiently small. QED

7 Proof of Theorem 2: Outline

Overview

The proof of Theorem 2 is very long and complicated. The main idea of the proof is the implementation of a technique we call *encrypted conversations.* This is a general technique for transforming proof systems into zero-knowledge proof systems. A protocol that has been transformed using this technique closely mirrors the original protocol. Indeed, all the questions and answers of the transformed protocol can be mapped to questions and answers in the original protocol. However, these questions and answers are all strongly encrypted, in an information theoretic sense, using keys that are known by the provers, but not by the verifier. Because the conversation is so strongly encrypted, the verifier gets no information, so the protocol is zero-knowledge.

Two concerns such a transformation must deal with are

- How can the verifier, who in a strong sense knows little of what has happened in an encrypted conversation, be convinced that the conversation indeed mirrors a valid conversation from the original protocol? Also, how can the verifier be convinced that the unencrypted conversation would indeed have caused the original verifier to accept?
- How can one insure that a malicious verifier cannot subvert the encrypted protocol in order to acquire information in some way?

We deal with the first concern by showing how the provers and verifier can take an encrypted transcript of the first i rounds of a conversation, and compute an encrypted transcript of the first $i + 1$ rounds of a conversation. This is done in such a way that the verifier can verify with high probability that this is the case. We deal with the second concern by insuring that the encrypted conversation, if generated at all, will mirror a conversation between the prover and an honest verifier. Thus, if the verifier follows the simulation, he will only find out whether the original verifier, on a random set of coin tosses, accepted. Since the original verifier accepts with probability 1, this is no information. Furthermore, we guarentee that if the verifier does not go along with the simulation, he will not get any information.

In order to accomplish these goals, we use a very useful tool called *oblivious circuit computation.* This tool, first developed by Yao [Yao86a], is a protocol by which two parties, A and B, possess secrets i and j respectively, and have agreed upon some circuit f. At the end of the protocol, A learns nothing about j, and B learns $f(i,j)$, but nothing more about i than can be inferred from knowing j and $f(i,j)$. The provers and verifier can compute the next step of an encrypted conversation by obliviously evaluating a circuit. We sketch the reduction from encrypted conversations to oblivious circuit evaluation in appendix A.3.

A large portion of our construction is devoted to implementing oblivious circuit evaluation. Yao's implementation of this protocol relies on complexity theoretic assumptions, and is therefore unsuitable for our purposes. More recently, however, this protocol was implemented using a subprotocol known as *oblivious transfer* in lieu of any cryptographic assumptions[K]. In the standard, two-party scenario, oblivious transfer cannot be implemented without complexity theoretic assumptions. However, we show that oblivious transfer can be implemented in the two-prover scenario without recourse to these assumptions. Our implementation uses a result of Barringtion [Ba] that NC^1 languages can be accepted by bounded width branching programs. We sketch our implementation in appendix A.2.

8 Acknowledgements:

We are grateful to Oded Goldreich for many helpful discussions during the course of this research and otherwise. His remarks have been instrumental in pointing us to possible future directions of this research.

We would also like to thank Yishai Mansour, Nimrod Megiddo and John Rompel for useful remarks.

References

[AL] Angluin, Dana and David Lichtenstein. "Provable Security of Cryptosystems: a Survey," YALEU/DCS/TR-288, 1983.

Proceeding of the 17th STOC, 1985, pp. 421-429.

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

[Bl] Blum, M., Private Communication.

[BC] Brassard, Gilles and Claude Crépeau. "Zero-Knowledge Simulation of Boolean Circuits," *Proceedings of the* 27th *FOCS*, IEEE, 1986, 188–195.

[BHZ] Boppana, Ravi, Johan Hastad, and Stathis Zachos. "Does CoNP Have Short Interactive Proofs?," *IPL*, 25, 1987, 127–132.

[CDvdG] Chaum, David, Ivan Damgard, and Jeroen van de Graaf. "Multiparty Computations Ensuring Secrecy of Each Party's Input and Correctness of the Output," *Proceedings of CRYPTO '87. Proceedings of CRYPTO '85*, Springer-Verlag, 1986, 477–488.

[CG] Chor, B, Goldreich G, "Unbiased Bits from Sources of Weak Randomness and Probabilistic Communication Complexity", *Proceedings of the 26th FOCS*, 1985.

[Ck] Crépeau C., Kilian J., Private Communication.

[CW] Carter L., Wegman M., "Universal Classes of Hash Functions", JCSS 18, 1979, pp. 143-154.

[C] Crépeau Claude, "On the Equivalence of Two Types of Oblivious Transfer", Crypto87.

[FST] Feige, U., Shamir, A., Tennenholtz, M., "The Noisy Oracle Problem", Private Communication of Manuscript

[F] Fortnow, Lance. "The Complexity of Perfect Zero-Knowledge," *Proceedings of the* 19th *STOC*, ACM, 1987, 204–209.

[FRS] Fortnow, Lance., Rompel J., Sipser M., "On the Power of Multi-Prover Interactive Proofs" *In preparation*

.

[FMR] Fischer M., Micali S., Rackoff C., and Wittenberg S., "An Oblivious Transfer Protocol Equivalent to Factoring", In Preparation.

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

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

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

[GMS] Goldreich O.,, Mansour Y.,, and Sipser M.,, "Interactive Proof Systems: Provers that Never Fail and Random Selection", *Proceedings of the 28th FOCS*, 1987, pp. 449-462.

[GMW1] Goldreich, Oded, Silvio Micali, and Avi Wigderson. "Proofs that Yield Nothing but the Validity of the Assertion, and a Methodology of Cryptographic Protocol Design," *Proceedings of the* 27th *FOCS*, IEEE, 1986, 174-187.

[GMW2] Goldreich, Oded, Silvio Micali, and Avi Wigderson. "How to Play ANY Mental Game," *Proceedings of the* 19th *STOC*, ACM, 1987, 218–229.

[GV] Goldreich, O., Vainish, R. "How to Solve any Protocol Problem: An Efficiency Improvement", Crypto 87.

[GMR] Goldwasser, Shafi, Silvio Micali, and Charles

Rackoff. "The Knowledge Complexity of Interactive Proof-Systems," *Proceedings of the* 17th *STOC*, ACM, 1985, 291-304.
[HM] Hastad J., Mansour Y., "Private Communication"
[K] Kilian, J. "On the Power of Oblivious Transfer," *Proceedings of this conference*
[L] Lautemann C., "BPP and the Polynomial Time Hierarchy", IPL, 14, 1983, pp. 215-217.
[R] Rabin, M., "How to exchange secrets by oblivious transfer", Tech. Memo TR-81, Aiken Computation Laboratory, Harvard University, 1981.
[Yao82] Yao, Andrew C. "Protocols for Secure Computations," *Proceedings of the* 23rd *FOCS*, IEEE, 1982, 160-164.
[Yao86a] Yao, Andrew C. "How to Generate and Exchange Secrets," *Proceedings of the* 27th *FOCS*, IEEE, 1986, 162-167.
[Yao86b] Yao, Andrew C. "Theory and Applications of Trapdoor Functions", Proc. of the 23rd FOCS, 1982, IEEE, pp.80-91.

A Structure of the Transformed Protocol.

Given a 2-prover IPS, we transform it into a zero-knowledge 2-prover IPS that has three distinct phases. These stages will be referred to as the *commital* phase, the *oblivious transfer phase*, and the *encrypted conversation* phase. In the commital phase of the protocol, the two provers commit a set of bits to the verifier. In the oblivious transfer phase of the protocol, the provers and verifier create a random sequence O of oblivious transfer bits. Sequence O has the following three properties.

- All of the bits of O are known to the provers.
- Each bit in O is known to the verifier with probability $\frac{1}{2}$.
- Neither prover knows which bits in O the verifer knows.

The third and final stage actually simulates the original 2 prover IPS. In this stage, sequence O is used to perform oblivious circuit computation, which then allows the use of the encrypted conversation technique. We now describe the three phases in greater detail.

A.1 The Commital Phase.

It is necessary for the two provers to be able to commit bits for use in the second, oblivious transfer phase of the protocol. This commital is of the same type as in the proof that any language in NP has a zero-knowledge 2-prover IPS. We use the same commital protocol as is used in Section 5.

The bits committed to in the commital phase may be random. In order to commit a bit b in the oblivious transfer phase, a prover can tell the verifier the value of $b \oplus b_c$, where b_c is a bit committed to in the commital phase. To decommit b, the prover can then simply decommit b_c.

A.2 The Oblivious Transfer Phase.

The oblivious transfer phase of the zero-knowledge IPS consists of several parallel evaluations of the oblivious transfer protocol, described below.

Introduction to Oblivious Transfer

We can view oblivious transfer as a protocol between two parties, A and B. Initially, A knows some random bit b, which is unknown to B. At the end of the protocol, the following two conditions hold.

1. (The Transfer Condition) One of the following two events has occured, each with probability $\frac{1}{2}$. Either B learns the value of b, or B learns nothing. Player B knows which of the two events occurred.
2. (The Obliviousness Condition) Player A receives no information about whether or not B learned the value of b.

Oblivious transfer, first introduced by Rabin[R], is a powerful cryptographic primitive. Its applications include contract signing [EGL] and oblivious circuit evaluation ([Y], [GMW2], [GHY], [AF], [GV], [K]). The first implementation of oblivious transfer by Rabin [R] was based on the difficulty of factoring and only worked for honest parties, Fischer, Micali, and Rackoff[FMR] presented the first implementation based on factoring and robust against computationally bounded adversaries. Even-Goldreich-Lempel[EGL] reduced the intractibility assumption to the existence of trapdoor permutations.

Unfortunately, these reductions are all cryptographic in nature, and thus of no use to us. Our implementation, which is not based on any cryptographic assumptions, exploits the lack of direct communication between the two provers.

A Variant of Oblivious Transfer in the 2-Prover Model.

We implement an analog to oblivious transfer in the two-prover model. At the beginning of the protocol, the provers know(have chosen) some random bit b, which the verifier does not know. The provers and the verifier have also agreed on a security parameter K. At the end of the protocol, the following variants of the usual transfer and obliviousness conditions hold.

1. (The Transfer Condition) One of the following events occurs with probability $\frac{1}{2}$. Either the verifier fully learns the value of b (i.e. can predict b with probability 1), or the verifier gains only partial knowledge of b (i.e. can predict b with probability $\frac{3}{4}$). The verifier knows which of the two events occurred.
2. (The Obliviousness Condition) Let K denote the security parameters. For all $c > 0$, and for K sufficiently large, if the two provers communicate less than K bits of information, they cannot predict, with probability $\frac{1}{2} + 1/K^c$, whether the verifier fully learned b.

Our implementation of this oblivious transfer protocol requires a constant number of rounds. The total number of bits of communication between the provers and the verifier will by polynomial in K and the size of the input.

Both the transfer and the obliviousness conditions are relaxed versions of the standard ones. The transfer condition is relaxed purely for ease of implementation. Using the techniques of Crépeau-Kilian[CK], we can show that achieving this weakened transfer condition is equivalent to achieving the ideal transfer condition. The standard obliviousness condition, however, *cannot* be implemented in this model if the two provers are allowed to freely communicate. To get around this difficulty, we show that for interactive proof systems, a set of bits transferred under the nonideal obliviousness condition may be used in place of a set of bits transferred under the ideal obliviousness condition.

Branching programs.

The main idea behind the oblivious transfer protocol is a simulation of width 5 permutation branching programs(W5PBP), as defined in [B]. Before describing the protocol, we first present a slightly nonstandard way to specify a W5PBP. We then show a way of randomizing this specification. Using this randomized representation, we can then describe our oblivious ransfer protocol.

W5PBP's may be formally thought of as having some polynomial $p(n)$ levels, each with five nodes. On level 1 there is a distinguished start node s; on level $p(n)$ there is a distinguished accept node a. For each level, i, $1 \leq i < p(n)$, there is an input variable, which we denote by v_i, and two $1-1$ mappings, f_0^i and f_1^i, that map the nodes at level i to the nodes at level $i+1$. Intuitively, the mapping f_0^i tells where to go if the input variable v_i is 0, and f_1^i tells where to go if v_i is equal to 1. A branching program may be evaluated by on a set of inputs by computing

$$\text{Branching\_ Program}(x_1, \ldots, x_n) =$$
$$(f_{v_{p(n)-1}}^{p(n)-1} \circ f_{v_{p(n)-2}}^{p(n)-2} \circ \cdots \circ f_{v_1}^1)(s). \qquad (A.2.1)$$

If this value if equal to the accept node a, the branching program accepts, otherwise, it is rejects. An example of a program is in fig. 1.

As described above, our branching programs consist of variables, nodes, and functions from nodes to nodes. For our protocol, we need an alternate representation for branching programs. Given a W5PBP, we first pick a random mapping γ, that maps nodes to $\{1, \ldots, 5\}$, subject to the constraint that no two nodes on the same level are mapped to the same number. We then replace each function f_k^i, $k \in \{0, 1\}$, by a permutation h_k^i, subject to the constraint

$$h_k^i(\gamma(N)) = \gamma(f_k^i(N)), \qquad (A.2.1)$$

for all nodes N on level i. From equations (A.2.1) and (A.2.2) we have

$$\gamma(\text{Branching\_ Program}(x_1, \ldots, x_n)) =$$
$$h_{v_{p(n)-1}}^{p(n)-1} \circ h_{v_{p(n)-2}}^{p(n)-2} \circ \cdots \circ h_{v_1}^1(\gamma(s)). \qquad (A.2.3)$$

This isomorphism between evaluating the permutations h_k^i on $\gamma(s)$ and evaluating the original branching program proves very useful in implementing oblivious transfer, as we will show in the next section. The following simple lemma is useful in analyzing the information transferred by the oblivious transfer protocol we will present.

Lemma A.1: Suppose that for each level, i, of a branching program, exactly one of the functions h_0^i or h_1^i is specified. Suppose also that for some level j, $\gamma(N)$ is specified for all nodes N on level j. Then there is exactly one way of consistently defining γ and the functions h_k^i.

Proof Outline: First, we note that specifying γ specifies all the h's. Thus we need only show that there is exactly one way of consistently defining γ. By equation A.2.2, we have

$$\gamma(N) = {h_k^i}^{-1}(\gamma(f_k^i(N))), \text{ and} \qquad (A.2.4)$$

$$\gamma(N) = h_k^i(\gamma({f_k^i}^{-1}(N))). \qquad (A.2.5)$$

If γ is defined on level i, equation(A.2.4) uniquely extends it to level $i-1$, and equation (A.2.5) uniquely extends it to level $i+1$. Inductively, one can uniquely extend γ from row j to the entire branching program. This extension is easily shown to be consistent. ∎

The oblivious transfer protocol.

We now outline the oblivious transfer protocol between the two provers and the verifier. For the exposition, we assume that the provers follow the protocol. It is not hard to convert this protocol to one that works with adversarial provers.

Stage 1: Let $n = K^2$. Both provers initially start with some canonical W5PBP that, given two vectors

$\vec{x} = [x_1\ x_2\ \dots\ x_n]$ and $\vec{y} = [y_1\ y_2\ \dots\ y_n]$, accepts iff $\vec{x} \cdot \vec{y} = 1$. They then agree on a random mapping γ, and permutations h^i_k. The provers send the verifier the exclusive-or of b and the least significant bit of $\gamma(a)$.

Stage 2: The verifier and Prover 1 pick a random vector x. The verifier and Prover 2 pick a random vector y. As a subprotocol, the prover and verifier flip an unbiased coin in the following manner: Prover i chooses as his bit, r_p, one of the bits committed in the commital phase of the protocol. The verifier chooses a bit r_v at random, and announces it to Prover i. Prover i then decommits r_p. The bit r, defined by $r = r_p \oplus r_v$ will be unbiased if either Prover i or the verifier obeys the protocol.

Stage 3: Prover 1 sends the verifier the permutations $h^i_{v_i}$, for all i such that $v_i = x_j$ for some j. Likewise, Prover 2 sends the verifier the permutations $h^i_{v_i}$, for all i such that $v_i = y_j$ for some j. For example, if $v_i = y_7$, and $y_7 = 0$, then Prover 2 would send the verifier h^i_0, but not send him h^i_1.

We now show how to convert this protocol to one in which the provers may be adversarial. First, we require that the provers commit their γ and their permutations h^i_0 and h^i_1 at Stage 1 of the oblivious transfer protocol, using the commital protocol described in section 1. The verifier must be assured that the following two conditions are met.

1. The permutations it receives correspond to those that have been committed, and
2. The permutations and γ correspond to a legitimate randomized branching program.

The first condition is assured by having the provers decommit their permutations in Stage 3 of the protocol. To assure that the second condition is met, we have the verifier perform a "spot-check" with probability $1/n^c$, where n is the size of the input, and c is some positive constant. To perform a spot-check, the verifier halts the oblivious transfer protocol at the beginning of Stage 2. Instead of using the committed W5PBP to implement oblivious transfer, the verifier requests that γ and all the hash functions are revealed. The verifier can then check whether or not the two provers gave a legitimate randomized W5PBP, and reject if they did not. Note that it is only necessary for the verifier to be able to detect cheating by the provers some polynomial fraction ofthe time. This probability may be amplified by successively running the zero-knowledge proof system sufficiently many times.

Properties of the Oblivious Transfer Protocol

The following theorems state that the above protocol does indeed implement our variant of oblivious transfer.

Theorem: (Transfer) After the above protocol has been executed, one of the following two events may occur, each with probability 1/2.

(1) The verifier knows the value of b.

(2) The verifier can guess the value of b with probability at most 3/4.

Furthermore, the verifier can tell which event occurred.

Proof Outline: Suppose, that $\vec{x} \cdot \vec{y} = 1$. Then the verifier can compute $\gamma(a)$, and thus compute b. This corresponds to event (1). Now suppose that $\vec{x} \cdot \vec{y} \neq 1$. The verifier knows, for each level i, exactly one of the functions h^i_0 or h^i_1. The verifier can also compute $\gamma(a')$, where a' is also on the last level, and $a' \neq a$. Everything else the verifier knows can be computed from this information. Using Lemma 1, we have that any specification of γ on the top level nodes can be consistently extended in exactly one way. Thus, the verifier has no information about $\gamma(a)$ other than the fact that $\gamma(a) \neq \gamma(a')$. The verifier's predictive ability is maximized when $\gamma(a')$ is even, in which case the conditional probability that $\gamma(a)$ is odd is 3/4. In this situation, the verifier can predict b with probability 3/4. ▮

Theorem: (Obliviousness) Let c be a constant, $c > 0$, and K, the security parameter, be sufficiently large (possibly depending on c). If, after the above protocol has been executed, the two provers exchange only K bits of information, they cannot predict, with probability $\frac{1}{2} + 1/K^c$, whether the verifier received the bit.

Proof Outline: We again use the observation that the verifier receives a bit iff the dot product of the two randomly chosen vectors is equal to 1. Determining if the verifier received the bit is equivalent to computing the dot product of two random vectors of size n. We now cite a theorem of Chor and Goldreich [CG] concerning the communication complexity of computing dot products.

Theorem[CG]: Let players A and B each receive random n bit boolean vectors, $\vec{x}$ and $\vec{y}$ respectively. If they exchange $o(n)$ bits, they cannot predict $\vec{x} \cdot \vec{y}$ with probability greater than $\frac{1}{2} + 1/n^c$, for any c.

Our theorem follows directly from this result. ▮

Ideal versus Nonideal Oblivious Transfer Bits.

As we have mentioned above, the oblivious transfer protocol we implement is nonideal in the obliviousness conditions. The nonideal nature of the obliviousness condition is inherent to our model, if the transfer condition is indeed ideal in the information theoretic sense. If the two infinitely powerful provers are allowed to communicate freely, they can each learn the entire transcript of the oblivious transfer protocol, and thus determine everything the verifier could have learned from the protocol. This violates the obliviousness condition of oblivious transfer, yielding the following observation.

Observation: It is impossible to implement an ideal oblivious transfer protocol between two provers and a verifier if the provers are allowed to communicate freely after the protocol.

The nonideal nature of the oblivious condition does not affect whether a protocol is zero-knowledge; the verifier learns exactly as much from a pseudo-oblivious source as from an oblivous one. However, using a pseudo-oblivious source of bit instead of an ideal source could conceivably cause a protocol to no longer be a proof system. We show that, provided the security parameter for our pseudo-oblivious source is sufficiently high, this will not be the case.

Formalizing Proof Systems with Oblivious Transfer Channels.

In order to state our result more precisely, we first augment our definition of two-prover interactive proof systems by adding a fourth party, a transfer source.

Definition: A two-prover interactive protocol with oblivious transfer consists of a four-tuple of parties, $< P_1, P_2, V, T >$. Parties P_1, P_2, V may be formally described as mappings from sequences of Σ^* (informally, the history of that party's conversation so far) to distributions on Σ^*(informally, the next answer/question given/asked by the party).

Player T may be formally described as a mapping from $\{0,1\}^*$ to a distribution on triples (I_{P_1}, I_{P_2}, I_V). The values I_{P_1}, I_{P_2} may be informally thought of as information leaked back to the provers, P_1 and P_2, by a possibly nonideal oblivious transfer protocol. The possible values of I_V on input $O = O_1, \ldots, O_k$ are elements of $\{0,1,\#\}^*$, of the form $O'_1 \ldots O'_k$, where $O'_i = O_i$ or $O'_i = \#$. Informally, I_V consists of the bits that are tranferred to the verifier, V.

For the rest of the discussion, we will anthromorphize our descriptions of the P_1, P_2, V and T, describing their behavior in terms of actions by players instead of as values of functions.

Protocols with oblivious transfer are evaluated in nearly the same way as standard protocols, but for an initial oblivious transfer phase. At the beginning of the protocol, the provers, P_1 and P_2, agree on a sequence of bits O, which they send to the transfer mechanism, T. The transfer mechanism sends some of these bits to the verifier, and sends additional information back to the two provers. At this point, T no longer plays any part in the protocol, and the players P_1, P_2, and V proceed to interact in the same manner as with standard two-prover protocols. Players P_1, P_2, and V treat their views of the oblivious transfer phase as special inputs.

Modeling ideal and nonideal sources in our formalism.

We now give a specification for an oblivious transfer mechanism which models the information received by the provers by the actual oblivious transfer mechanism we have implemented in the two-prover model.

Specification: Oblivious transfer mechanism $T_{n,k}$ is specified by its input from the provers and its output to the provers and the verifier. $T_{n,k}$ takes as input a sequence of bits $O = O_1, \ldots, O_k$. It flips k coins, $b_1, \ldots, b_k$. $T_{n,k}$ randomly selects two sequences of n element boolean vectors, $\vec{x}_1, \ldots, \vec{x}_k$ and $\vec{y}_1, \ldots, \vec{y}_k$, subject to $\vec{x}_i \cdot \vec{y}_i = b_i$. $T_{n,k}$'s output is as follows.

Transfer to V: $T_{n,k}$ sends the verifier sequence $O' = O'_1, \ldots, O'_k$ where $O'_i = O_i$ iff $b_i = 1$. Otherwise, $O'_i = \#$.

Transfer to P_1: $T_{n,k}$ sends P_1 the sequence $\vec{x}_1, \ldots, \vec{x}_k$.

Transfer to P_2: $T_{n,k}$ sends P_2 the sequence $\vec{y}_1, \ldots, \vec{y}_k$.

This model for our transfer channel makes the following simplifications. The verifier does not get any partial glimpses at bits that it hasn't completely received, whereas in the actual protocol, it may guess it with probability 3/4. Also, it does not get any record of its interactions with the provers in the oblivious transfer protocol. For instance, in the actual protocol, the verifier would also know the $\vec{x}_i$'s and $\vec{y}_i$'s, whereas in this model it does not. These simplifications turns out to be irrelevant to our analysis, since the valid verifier completely disregards all of this extra information.

More significantly, the provers do not receive any of the extra information they might obtain in the commital and oblivious transfer phases. One can show that any pair of provers which have any chance of fooling the verifier must abide by rules of the commital and oblivious transfer protocols. The extra information they receive from an honest run of these protocols is of no value to them. They may, in a certain technical sense, simulate all of this extra information, once given their respective vector sequences $\vec{x}_1, \ldots, \vec{x}_k$ and $\vec{y}_1 \ldots, \vec{y}_k$. Thus, the provers cannot cheat any more effectively using our simplified channel than they could using the actual commital and oblivious transfer protocols. The details of this argument are ommitted.

Modeling an ideal oblivious transfer mechanism.

It is fairly straightforward to model an ideal oblivious transfer mechanism in our formalism. We denote this transfer channel T_k^{ideal}, which we specify as follows.

Specification: Oblivious transfer mechanism T_k^{ideal} is specified by its input from the provers and its output to the provers and the verifier. T_k^{ideal} takes as input a sequence of bits $O = O_1, \ldots, O_k$. It flips k coins, $b_1, \ldots, b_k$. It randomly selects two sequences of n element boolean vectors, $\vec{x}_1, \ldots, \vec{x}_k$ and $\vec{y}_1, \ldots, \vec{y}_k$. T_k^{ideal}'s output is as follows.

Transfer to V: T_k^{ideal} sends the verifier sequence $O' = O'_1, \ldots, O'_k$ where $O'_i = O_i$ iff $b_i = 1$. Otherwise, $O'_i = \#$.

Transfer to P_1 and P_2: T_k^{ideal} sends nothing to P_1 or P_2.

A practical equivalence between $T_{n,k}$ and T^{ideal}.

We can now state our theorem concerning the the practical equivalence of our oblivious transfer protocol and the ideal one.

Theorem: Let $< P_1, P_2, V, T_{p(n)}^{ideal} >$ be an interactive proof system with oblivious transfer. Here, $p(n)$ denotes some polynomial in the size of the input. Then there exists some some polynomial $q(n)$ such that $< P_1, P_2, V, T_{q(n),p(n)} >$ is also an interactive proof system with oblivious transfer.

Brief Outline of Proof: The proof of this theorem is somewhat involved. We show that if one could cheat more effectively using a $T_{q(n),p(n)}$ transfer channel, for $q(n)$ arbitrarily large, then one could use this fact to create a protocol for computing the dot product of two random $q(n)$ element boolean vectors. The communication complexity for this protocol will depend on V and n, but not on the function q. From this it is possible to use the Chor-Goldreich lower bound on the communication complexity of boolean dot product to reach a contradiction.

In order to constuct the protocol for computing boolean dot products, we first define a sequence of transfer mechanisms that are intermediate between our nonideal and ideal transfer mechanisms. We show that if the provers can cheat using the nonideal transfer mechanism, then two consecutive transfer mechanisms in our sequence can be distinguished. We then show how to use these transfer mechanisms to generate two very simple and very similar transfer mechanisms whose behavior is distinguishable. Finally, we use the distinguishability of this final pair of transfer mechanisms to create a protocol for boolean dot-product. We proceed to formalize this argument.

Transfer mechanisms that are intermediate between the ideal and nonideal models.

We specify a sequence of oblivious transfer mechanisms as follows.

Specification: Oblivious transfer mechanism $T_{n,k}^i$ is specified by its input from the provers and its output to the provers and the verifier. $T_{n,k}^i$ takes as input a sequence of bits $O = O_1, \ldots, O_k$. It flips k coins, $b_1, \ldots, b_k$. $T_{n,k}^i$ randomly selects two sequences of n element boolean vectors, $\vec{x}_1, \ldots, \vec{x}_k$ and $\vec{y}_1, \ldots, \vec{y}_k$. For $1 \leq j \leq i$, vectors $\vec{x}_j$ and $\vec{y}_j$ are subject to the constraint $\vec{x}_j \cdot \vec{y}_j = b_j$. $T_{n,k}^i$'s output is as follows.

Transfer to V: $T_{n,k}^i$ sends the verifier a sequence $O' = O'_1, \ldots, O'_k$ where $O'_i = O_i$ iff $b_i = 1$. Otherwise, $O'_i = \#$.

Transfer to P_1: $T_{n,k}^i$ sends P_1 the sequence $\vec{x}_1, \ldots, \vec{x}_k$.

Transfer to P_2: $T_{n,k}^i$ sends P_2 the sequence $\vec{y}_1, \ldots, \vec{y}_k$.

The only difference between $T_{n,k}$ and $T_{n,k}^i$ is that the vectors sent to the provers by $T_{n,k}$ all have some correlation with whether the bit was sent to the verifier, whereas only the first i vectors sent to the provers by $T_{n,k}^i$ are so correlated. Note that $T_{n,k}^0$ is equivalent to the ideal channel T_k^{ideal}, and $T_{n,k}^k$ is equivalent to $T_{n,k}$.

Analysis of cheating probabilities for different transfer mechanisms.

The sequence of oblivious transfer mechanisms we defined above is "continuous" in that any two consecutive mechanisms are only incrementally different from each other. Using an argument similar to that of [GM], we show that if the probability of successfully cheating using one transfer mechanism in the sequence is significantly greater than the probability of successfully cheating using a different transfer mechanism in the sequence, then there must be two consecutive mechanisms which differ in the probability of a particular cheating strategy being successful.

Definition: Let L be some language, and $< P_1, P_2, V, T_{p(n)}^{ideal} >$ a two-prover IPS for L, with oblivious transfer. For some $x \notin L, |x| = n$, we define $cheat_{ideal}(x)$ as the probability that V can be tricked into accepting x.

We wish to analyze how frequently the provers can cheat
if they use a nonideal transfer mechanism, $T_{q(n),p(n)}$. Let $P_{1,q(n)}, P_{2,q(n)}$ be optimal cheating provers for the protocol $< P_{1,q(n)}, P_{2,q(n)}, V, T_{q(n),p(n)} >$. For $x \notin L, |x| = n$, we define $cheat_{q(n)}^i(x)$ as the probability that $P_{1,q(n)}, P_{2,q(n)}$ causes V to accept x in protocol $< P_{1,q(n)}, P_{2,q(n)}, V, T_{q(n),p(n)}^i >$.

Clearly, we have $cheat_{q(n)}^0(x) \leq cheat_{ideal}(x)$. We also have, by definition, that $cheat_{q(n)}^{p(n)}(x)$ is the maximum probability that any provers can trick V into accepting x, using transfer mechanism $T_{q(n),p(n)}$.

Using a simple pigeonhole argument, we can show the following.

Lemma A.2: Let $x \notin L$, and $|x| = n$. For all polynomials $q(n)$, there exists some i, $0 \leq i < p(n)$, such that

$$cheat_{q(n)}^{i+1}(x) - cheat_{q(n)}^{i}(x) \geq$$

$$\frac{cheat^{p(n)}_{q(n)}(x) - cheat^{0}_{q(n)}(x)}{p(n)}. \qquad (A.2.6)$$

We now show that if for for all polynomials $q(n)$, there exists a $c > 0$, such that $cheat^{i+1}_{q(n)}(x) - cheat^{i}_{q(n)}(x) > 1/|x|^c$ for infinitely many x, then we can create efficient algorithms for computing dot products of random vectors. To do this, we first must introduce the notion of "hardwired" versions of transfer mechanisms $T^{i}_{q(n),p(n)}$.

Restricted versions of oblivious transfer mechanisms.

Given two easily distinguishable mechanisms $T^{i}_{q(n),p(n)}$ and $T^{i+1}_{q(n),p(n)}$, we would like to create even simpler pairs of mechanisms that are easily distinguishable, yet preserve the essential differences between $T^{i}_{q(n),p(n)}$ and $T^{i+1}_{q(n),p(n)}$. We observe that the only difference between these two mechanisms lies in the distibutions imposed on the vectors $\vec{x}_{i+1}$ and $\vec{y}_{i+1}$ which are sent to $P_{1,q(n)}$ and $P_{2,q(n)}$. We would like to be able to fix all the other aspects ofthese channels. To do this, we make the following definitions.

Definition: A *transfer restriction* $R \in \mathcal{R}^{i}_{n,k}$ is a 3-tuple (R_b, R_x, R_y), where

- R_b is a sequence of bits, $b_1, \ldots, b_k$.
- R_x is a $k-1$ element sequence of n element boolean vectors, $\vec{x}_1, \ldots, \vec{x}_{i-1}, \vec{x}_{i+1}, \ldots, \vec{x}_k$.
- R_y is a $k-1$ element sequence of n element boolean vectors, $\vec{y}_1, \ldots, \vec{y}_{i-1}, \vec{y}_{i+1}, \ldots, \vec{y}_k$.

Furthermore, we require that for $1 \le j < i$, $\vec{x}_j \cdot \vec{y}_j = b_j$

Intuitively, we can think of $R \in \mathcal{R}^{i}_{n,k}$ as a specification for which bits get through to the verifier, and, except for the ith bit, specifications for which vectors are transmitted back to the provers.

Definition: Given a transfer restriction $R \in \mathcal{R}^{j}_{n,k}$ We specify a restricted version of $T^{i}_{n,k}$, which we denote by $T^{i}_{n,k}[R]$, as follows.

Specification: Oblivious transfer mechanism $T^{i}_{n,k}[R]$ takes as input a sequence of bits $O = O_1, \ldots, O_k$. Let $R_b = b_1, \ldots, b_k$, $R_x = \vec{x}_1, \ldots, \vec{x}_{i-1}, \vec{x}_{i+1}, \ldots, \vec{x}_k$, and $R_y = \vec{y}_1, \ldots, \vec{y}_{i-1}, \vec{y}_{i+1}, \ldots, \vec{y}_k$. $T^{i}_{n,k}[R]$ randomly selects two n element boolean vectors, $\vec{x}_j$ and $\vec{y}_j$. If $j \le i$, then $\vec{x}_j$ and $\vec{y}_j$ are chosen s.t $\vec{x}_j \cdot \vec{y}_j = b_j$. $T^{i}_{n,k}[R]$'s output is as follows.

Transfer to V: $T^{i}_{n,k}[R]$ sends the verifier sequence $O'_1, \ldots, O'_k$ where $O'_i = O_i$ iff $b_i = 1$. Otherwise, $O'_i = \#$.

Transfer to P_1: $T^{i}_{n,k}[R]$ sends P_1 the sequence $\vec{x}_1, \ldots, \vec{x}_k$.

Transfer to P_2: $T^{i}_{n,k}[R]$ sends P_2 the sequence $\vec{y}_1, \ldots, \vec{y}_k$.

Analysis of cheating with respect to restricted transfer mechanisms.

Recall that provers $P_{1,q(n)}$ and $P_{2,q(n)}$ cheat optimally, given oblivious transfer mechanism $T_{q(n),p(n)}$. We would like to describe what happens when these provers are run using restricted transfer mechanisms. To this end, we define $cheat^{i}_{q(n)}[R](x)$ as the probability that $P_{1,q(n)}, P_{2,q(n)}$ causes V to accept x in protocol $< P_{1,q(n)}, P_{2,q(n)}, V, T^{i}_{q(n),p(n)}[R] >$.

Using a simple probabilistic argument, we prove the following important lemma.

Lemma A.3: Let $x \notin L$, and $|x| = n$. Let $1 \le i < p(n)$. For all polynomials $q(n)$, there exists a restriction $R \in \mathcal{R}^{i+1}_{q(n),p(n)})$ such that

$$cheat^{i+1}_{q(n)}[R](x) - cheat^{i}_{q(n)}[R](x) \ge$$
$$cheat^{i+1}_{q(n)}(x) - cheat^{i}_{q(n)}(x). \qquad (A.2.7)$$

Using $T^{i}_{q(n),p(n)}[R]$, $T^{i+1}_{q(n),p(n)}[R]$ to compute dot products.

Recall that a restriction $R \in \mathcal{R}^{i+1}_{q(n),p(n)})$ defines the entire input/output properties of a restricted transfer protocol $T^{i}_{q(n),p(n)}[R]$, but for the output vectors $\vec{x}_i, \vec{y}_i$ transmitted back to the provers. If the two provers have a source $M_{q(n)}$, which produces vector pairs $\vec{x}, \vec{y}$, of size $q(n)$ and sends them to *Prover*1 and *Prover*2, respectively, we can use it to simulate $T^{i}_{q(n),p(n)}[R]$.

We also note that, if allowed to communicate directly, two provers can "simulate" the verifier in the following way. They can send to each other the messages they would have sent to the verifier. By knowing the set of transfer bits, which bits were received by the verifier, and a transcript of the conversation so far between the verifier and the provers, the provers can determine exactly what the verifier's next question in the conversaton will be.

We now can explicitly write down a protocol for computing the dot product of random boolean vectors. The assume that the two parties P_1 and P_2 have agreed on some $x(x \notin L.|x| = n), q, i$, and $R = (R_b, R_x, R_y) \in \mathcal{R}^{i+1}_{q(n),p(n)})$. The protocol is specified as follows. Player P_1 receives a random boolean vector $\vec{x}$, and player P_2 receives a random boolean vector $\vec{y}$. At the end of the protocol, player P_1 outputs a 0 or 1, which hopefully corresponds to $\vec{x} \cdot \vec{y}$.

Protocol: Dot-Product($\vec{x}, \vec{y}$) /* P_1 knows $\vec{x}$, P_2 knows $\vec{y}$, and $|\vec{x}| = |\vec{y}| = q(n)$ */

P_1 and P_2 simulate the protocol $< P_{1,q(n)}, P_{2,q(n)}, V, T^{i}_{q(n),p(n)}[R] >$, on input x. They treat vectors $\vec{x}$ and $\vec{y}$ as substitutes for $\vec{x}_{i+1}, \vec{y}_{i+1}$ (which are not defined by R).

If the simulated verifier accepts, then P_1 outputs b_{i+1}, where $R_b = b_1, \ldots, b_{p(n)}$. Otherwise it outputs the complement of b_{i+1}.

We now analyze the communication complexity of this protocol.

Definition: Given a two-prover protocol $\mathcal{P} < P_1, P_2, V, T >$, and some input x, we define the *leakage* $\mathcal{L}(\mathcal{P}, x)$ as the total number of bits transmitted from the provers to the verifier.

The following lemma follows immediately from the definition of Dot-Product.

Lemma A.4: Let $\mathcal{P} =< P_{1,q(n)}, P_{2,q(n)}, V, T^i_{q(n),p(n)}[R] >$. Then protocol Dot-Product requires $\mathcal{L}(\mathcal{P}, x)$ bits of communication. ▮

Finally, we can bound below **Dot − Product's** success rate on random vectors by the following lemma.

Lemma A.5: Given $q(n)$ bit vectors $\vec{x}, \vec{y}$ distributed uniformly, the probability that **Dot-Product**$(\vec{x}, \vec{y}) = \vec{x} \cdot \vec{y}$ is at least

$$\frac{1}{2} + \left(cheat^{i+1}_{q(n)}[R](x) - cheat^{i}_{q(n)}[R](x)\right). \qquad (A.2.8)$$

Proof: our proof is by a straightforward calculation of conditional probabilities, which we outline below. We define the variables *good* and *bad* by

$$good = prob(\text{The simulated verifier accepts}| \vec{x} \cdot \vec{y} = b_i), \text{ and,}$$

$$bad = prob(\text{The simulated verifier accepts}| \vec{x} \cdot \vec{y} \neq b_i).$$

The probability that Dot-Product yields the correct answer is equal to

$$\frac{1}{2} \cdot good + \frac{1}{2} \cdot (1 - bad) \qquad (A.2.9).$$

We now solve for *good* and *bad* in terms of $cheat^{i}_{q(n)}[R](x)$ and $cheat^{i+1}_{q(n)}[R](x)$. Using our definitions for $cheat^{i}_{q(n)}[R](x)$ and $cheat^{i+1}_{q(n)}[R](x)$, we have

$$cheat^{i+1}_{q(n)}[R](x) = good, \text{ and,} \qquad (A.2.A)$$

$$cheat^{i}_{q(n)}[R](x) = \frac{1}{2} \cdot good + \frac{1}{2} \cdot bad. \qquad (A.2.11)$$

Solving for *good* and *bad*, we have

$$good = cheat^{i+1}_{q(n)}[R](x), \text{ and,} \qquad (A.2.12)$$

$$bad = cheat^{i+1}_{q(n)}[R](x) - 2(cheat^{i+1}_{q(n)}[R](x) - cheat^{i}_{q(n)}[R](x)). \qquad (A.2.13)$$

Substituting equations (A.2.12) and (A.2.13) into equation A.2.9), and simplifying, we get equation (A.2.8). ▮

A.3 Implementing zero-knowledge with circuits.

In this section we outline a technique we call *the method of encrypted conversations.* This technique represents a fairly general methodology for converting protocols into zero-knowledge protocols. Its main requirement is the ability of the parties involved to perform oblivious circuit evaluation.

A normal form for two-prover IPS's.

For ease of exposition, we consider a normal form for two-prover interactive proof systems(IPS's). This normal form consists of three stages, as described below.

Notation: Throughout this section, $q_i(x, r, \cdot, \cdot, \cdot)$ will denote the i-th question of the verifier computed on his random coin tosses r, the input x, and the history of the communication so far. (a_i correspond to the provers answers).

Stage 1: On input x, where $|x| = n$, the verifier generates a sequence $r = r_1, \ldots, r_{p(n)}$ of random bits. The verifier computes his first question, $q_1 = q_1(x, r)$.

Stage 2: The verifier sends q_1 to Prover 1. Prover 1 sends its answer, a_1 back to the verifier. The verifier computes his second question, $q_2 = q_2(x, r, a_1)$.

Stage 3: The verifier sends q_2 to Prover 2. Prover 2 sends its answer, a_2, back to the verifier. The verifier computes its decision predicate, $accept(x, r, a_1, a_2)$, and accepts iff $accept(x, r, a_1, a_2)$ evaluates to "true".

We use the following result.

Theorem(normal form for 2 prover IPS's): Given any two prover IPS $\mathcal{P}$ for a language L, there exists an IPS $\mathcal{P}'$, with the following 2 properties.

1. If $x \in L$ then $prob(\mathcal{P}'(x)$ accepts$)= 1$.
2. There exists some $c > 0$ such that if $x \notin L$ then $prob(\mathcal{P}'(x)$ accepts$) \leq 1 - 1/|x|^c$.

Remark: It is currently open whether the $\leq 1 - 1/|x|^c$ failure probability can be reduced. However, if greater reliability is desired, one may run a normal form protocol several times serially to achieve an exponentially low probability of failure.

We now need to show how to convert an IPS in normal form into a zero-knowledge IPS.

Conceptually, we would like to have the use of a black box into which the verifier inputs an encrypted history of the communication, the prover inputs its answer to the question and the output which is given to the verifier is the encrypted answer of the prover and the encrypted next question of the verifier. See fig. 2.

The encryption scheme used to encrypt the questions and answers should be an information theoretically strong encryption scheme with respect to the verifier, while the provers will be given the ability to decrypt.

We describe how this is achieved in the following section A.3.1. The box is achieved by the technique of oblivious circuit evaluation as described in section A.3.2.

A.3.1 Strong encryption using 2-universal hash functions.

We need a cryptographic system (call it E for the sake of discussion) which is both unbreakable, and existentially unforgeable. By unbreakable, we mean that if one is given $E(x)$, an encryption of x, but one does not have the decryption key, then one cannot infer anything about x. By existentially unforgeable, we mean that if one is given $E(x)$, an encryption of x, but one does not have the decryption key, then one cannot produce any string $forge$ such that $forge = E(y)$ for some y. These security requirements are information theoretic, and must apply to someone with arbitrary computational power.

To accomplish this, we use the notion of universal hash functions, first introduced by Carter and Wegman[CW]. In addition, we require the following property of our universal sets.

Definition: A family of 2-universal sets $\mathcal{H}_n$ of functions $h : \{0,1\}^n \rightarrow \{0,1\}^n$ is *almost self-inverse* iff for all c, and for all n sufficiently large (with respect to c), a function h, picked uniformly from $\mathcal{H}$, will have an inverse $h^{-1} \in \mathcal{H}$ with probability $> 1 - n^{-c}$.

One example of an almost self-inverse 2-universal set of hash functions is the set of linear equations over $GF(2^n)$. As there is a trivial correspondence between $\{0,1\}^n$ and $GF(2^n)$, we treat all our elements as being in $\{0,1\}^n$.

For our encryption system, we require that all legal messages m are padded with a number of trailing 0's equal to the length of the original message. We encrypt a message $m \in \{0,1\}^n$ by applying some uniformly selected function $h \in \mathcal{H}_n$ to it. We can decrypt $h(m)$ by by applying its h^{-1} to it. For our purposes, we can safely ignore the possibility that a uniformly chosen h isn't invertible. The following lemma shows that this encryption scheme is unbreakable and unforgeable.

Lemma: Let h be chosen uniformly from $\mathcal{H}_n$. Then

1. (unbreakability) $(\forall x, y \in \{0,1\}^n) prob(h(x) = y) = 2^{-n}$.
2. (unforgeability) $(\forall x, y, z \in \{0,1\}^n)$

$$prob((\exists w \in \{0,1\}^{n/2}0^{n/2}) h(w) = z | h(x) = y)$$
$$= 2^{-n/2}.$$

Proof: Both properties follow immediately from the definition of 2-universal hash functions. ∎

In the protocol the provers will agree on four random hash functions $h_1, h_2, h_3, h_4 \in \mathcal{H}_{p(n)}$. At the end of the protocol, the verifier will possess the values of $h_1(r), h_2(q_1)$, and $h_3(a_1)$, but will *not* possess any extra information about which functions h_1, h_2 and h_3 actually are. However, knowing the value of $h(x)$ gives no information, in the information theoretic sense, about the value of x. This is roughly how the zero-knowledge aspect of our protocol is achieved.

A.3.2 Use of oblivious circuit evaluation .

We use the reduction of Kilian[K] from oblivious transfer to oblivious circuit computation. This reduction maintains the usual security properties desired of oblivious circuit evaluation, without recourse to any intractibility assumptions.[5] Its sole requirement is a sequence $O = O_1, \ldots, O_{p(n)}$ of bits, all of which are known to A, and half of which are known to B(a more detailed description of this condition is given in section A.2). This set of bits(or, more technically, a reasonable approximation to such a set) is provided by the oblivious transfer protocol outlined in section A.2. for the rest of this discussion, we treat oblivious circuit evaluation as a primitive operation.

A.3.3 Outline of the Zero-Knowledge Protocol

We can now describe our zero-knowledge transformed protocol For our expositions, we still treat oblivious circuit computation of as a primitive. (A description of circuits C_0, C_1, C_2 and C_3 is given following the protocol.) Note the similaruty between this description and the description of the normal-form for protocols given above.

On input x, where $|x| = n$.

Step 0: Provers 1 and 2 agree on random invertible hash functions $h_1, h_2, h_3, h_4 \in \mathcal{H}_{2p(n)}$, and random string $r_1 \in \{0,1\}^{p(n)}$. The verifier selects a random string $r_2 \in \{0,1\}^{p(n)}$. The verifier and Prover 1 evaluate $r' = C_0[x](r_1, r_2, h_1)$. ($r$ will the random coin tosses to be used by the verifier).

Step 1: The verifier and Prover 1 then evaluate $q_1' = C_1[x](r', h_1^{-1}, h_2)$, the encrypted version of the verifier's first question.

[5]Goldreich-Vainish describe a simple reduction from oblivious circuit evaluation to oblivious transfer, without any intractibility assumptions. However, this reduction only works for honest parties, and is thus unuseable by us.

Step 2: The verifier sends q_1' to Prover 1. If $h_2^{-1}(q_1')$ does not decrypt to a legitimate message, then Prover 1 halts the conversation. Otherwise, Prover 1 computes his answer, a_1, and sends the verifier $a_1' = h_3(a_1)$. The verifier and Prover 1 evaluate $q_2' = C_2[x](r', a_1', h_1^{-1}, h_3^{-1}, h_4)$, the encrypted version of the verifiers second question.

Step 3: The verifier sends q_2' to Prover 2. If $h_4^{-1}(q_2')$ does not decrypt to a legitimate message, then Prover 2 halts the conversation. Otherwise, Prover 2 computes his answer, a_2. The verifier and Prover 2 evaluate $decision = C_3[x](r', a_1', a_2, h_1^{-1}, h_3^{-1})$.

At the end of this protocol, verifier accepts iff $decision = true$.

We now describe circuits C_i for $i = 0, 1, 2, 3$.

For each circuit, we give the input from the prover, the input from the verifier, and the output given to the verifier. We adopt the convention that $|x| = n$, and assume without loss of generality that all strings being exchanged in the protocol are of length $p(n)$, for some polynomial p. We use the following simple functions to simplify our exposition. Function $pad_n : \{0,1\}^n - \{0,1\}^{2n}$ pads an extra n zeros onto the end of an n-bit string Function $strip_n : \{0,1\}^{2n} \rightarrow \{0,1\}^n$, which removes the last n bits from a $2n$-bit string. Predicate $legal_n : \{0,1\}^{2n} \rightarrow \{true, false\}$ is true iff the last n bits of the input string are equal to 0.

$C_0[x]$:

Input from the prover: A sequence of bits, $b = b_1, \ldots, b_{p(n)}$, and a hash function $h \in \mathcal{H}_{2p(n)}$.

Input from the verifier: A sequence of bits $c = c_1, \ldots, c_{p(n)}$.

Output to the verifier: Output($h(pad_n(b \oplus c))$).

Circuit $C_0[x]$ is the initialization circuit that creates the verifiers random bits in Stage 1 of the protocol described above.

```
C_1[x]:
Input from the prover:
    Hash functions $h_1^{-1}, h_2 \in \mathcal{H}_{2p(n)}$.
Input from the verifier:
    String $r' \in \{0,1\}^{2p(n)}$.
Output to the verifier:
    $r = h_1^{-1}(r')$
    If $legal_{p(n)}(r) = false$
    Then Output($O^{2p(n)}$)
    Else  $r = strip_{p(n)}(r)$
          $q_1 = q_1(x, r)$
          Output($h_2(pad_{p(n)}(q_1))$)
```

Circuit $C_1[x]$ is used to implement Stage 1 of the protocol described above.

```
C_2[x]:
Input from the prover:
    Hash functions $h_1^{-1}, h_3^{-1}, h_4 \in \mathcal{H}_{2p(n)}$
Input from the verifier:
    Strings $r', a_1' \in \{0,1\}^{2p(n)}$.
Output to the verifier:
    $r = h_1^{-1}(r')$
    $a_1 = h_3^{-1}(a_1')$
    If ($legal_{p(n)}(r)$ and $legal_{p(n)}(a_1)$) $= false$
    Then Output($O^{2p(n)}$)
    Else  $r = strip_{p(n)}(r)$
          $a_1 = strip_{p(n)}(a_1)$
          $q_2 = q_2(x, r, a_1)$
          Output($h_4(pad_{p(n)}(q_2))$)
```

Circuit $C_2[x]$ is used to implement Stage 2 of the protocol described above.

```
C_3[x]:
Input from the prover:
    Hash functions $h_1^{-1}, h_3^{-1} \in \mathcal{H}_{2p(n)}$,
String $a_2 \in \{0,1\}^{p(n)}$
Input from the verifier:
Strings $r', a_1' \in \{0,1\}^{2p(n)}$.
Output to the verifier:
$r = h_1^{-1}(r')$
$a_1 = h_3^{-1}(a_1')$
If ($legal_{p(n)}(r)$ and $legal_{p(n)}(a_1)$) $= false$
Then Output($O^{2p(n)}$)
Else  $r = strip_{p(n)}(r)$
      $a_1 = strip_{p(n)}(a_1)$
      Output($accept(x, r, a_1, a_2)$)
```

Circuit $C_3[x]$ is used to implement Stage 3 of the protocol described above.

The two obvious questions we must deal with are, "Is this protocol still a proof system?", and "Is this protocol zero-knowledge?"

Is this protocol a proof system?

If the verifier is honest, and if the provers input the correct hash functions, and their inverses, into the circuits being evaluated, then one can map transcripts of conversations in this protocol into transcripts of the original protocol (with possibly cheating provers). In this case, the provers cannot cheat any more effectively they could in the original protocol, and the new protocol will remain a proof system if the original one was.

If the provers do not input consistent sets of hash functions, then nothing can be guarenteed about whether the protocol remains a proof system. However, using the machinery developed in [K], it is possible for the provers to commit, at the beginning of the protocol, all the hash functions they input to the circuits, along with a zero-knowledge proof that these inputs are consistent with each other.

Is this protocol zero-knowledge?

The proof that this protocol is zero-knowledge is, while not overly complex or difficult, relies too heavily on machinery from [K] to be concisely presented here. We make the following intuitive argument for why the protocol is zero-knowledge.

First, note that the verifier's actions are severely restricted by the use of circuits and the encryption scheme. Except for its random bits, all the inputs it gives to the provers or the circuits are encrypted with an unforgeable system. If the verifier ever attempts to give an *incorrect* string to a prover, the prover will detect the forgery will probability exponentially close to 1. Likewise, if the verifier inputs an incorrect string to a circuit, it will almost certainly output either $0^{2p(n)}$ or *false*. This rules out any active attack on the part of the verifier.

Second, we show that passive attacks by the verifier do not yield ay information. The intermediate outputs of circuits $C_1, \ldots, C_3$ are all uniformly distributed, and thus yield no information.

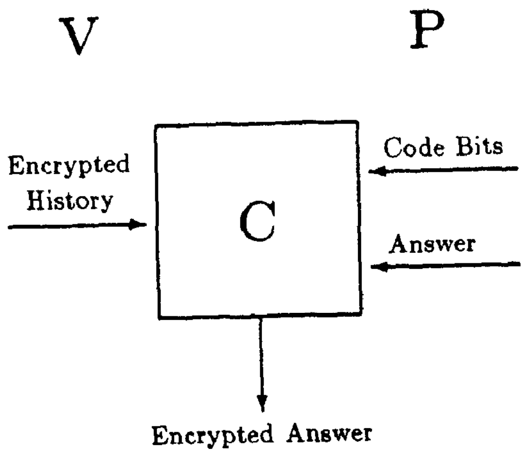

Figure 1: Schematic of encrypted conversation.

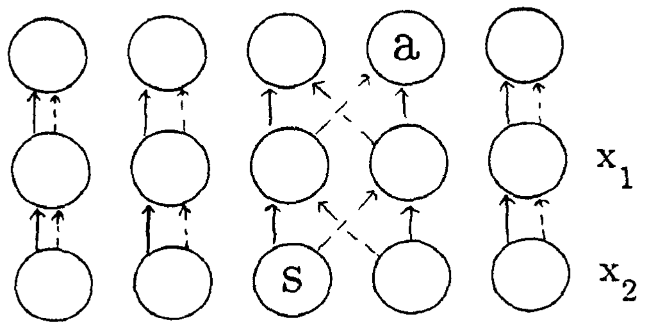

Figure 2: Schematic of a simple W5PBP. Solid lines correspond to functions f_1^i, dashed lines correspond to functions f_0^i. In this program, $v_1 = x_2$ and $v_2 = x_1$. This branching program is equivalent to $x_1 \oplus x_2$.