Founding Cryptography on Oblivious Transfer

Joe Kilian*
MIT

Abstract

Suppose your netmail is being erratically censored by Captain Yossarian. Whenever you send a message, he censors each bit of the message with probability $\frac{1}{2}$, replacing each censored bit by some reserved character. Well versed in such concepts as redundancy, this is no real problem to you. The question is, can it actually be turned around and used to your advantage? We answer this question strongly in the affirmative. We show that this protocol, more commonly known as *oblivious transfer*, can be used to simulate a more sophisticated protocol, known as *oblivious circuit evaluation*([Y]). We also show that with such a communication channel, one can have completely noninteractive zero-knowledge proofs of statements in NP. These results do not use any complexity-theoretic assumptions. We can show that they have applications to a variety of models in which oblivious transfer can be done.

1 Introduction

1.1 Eliminating assumptions: the problem at hand.

Cryptographers seldom sleep well([M]). Their careers are frequently based on very precise complexity-theoretic assumptions, which could be shattered the next morning. A polynomial time algorithm for factoring would certainly prove more crushing than any paltry fluctuation of the Dow Jones. The effect a proof that $\mathcal{P} = \mathcal{NP}$ would have is unspeakable.

*Research supported in part by a Fannie and John Hertz foundation fellowship, and NSF grant 865727-CCR. Some of this research was done while working at BellCore.

More recently, cryptographers have managed to hedge their investments. There have been some exciting new results on proving completeness theorems for multi-party protocols, due to [BGW], and [CCD]. In the area of interactive proof theory, a multi-party generalization of interactive proof systems has also been proposed [BGKW] (also, [FRS]) in which anything provable can be proven in zero-knowledge.

Two-party protocol problems have experienced no such breakthrough. The multi-party protocols do not work if half the players are malicious or trying to get information. Hence, they do not apply to the two-party case. While there is a very mature theory of two-party protocols(notably, [Y] and [GMW1]),the vast majority of the work rests on various cryptographic assumptions. The best that has been done so far is to prove theorems based on more general cryptographic assumptions, such as "trapdoor functions exist," rather than specific assumptions, such as "factoring is hard." Unfortunately, reductions to intractibility assumptions is the best we can do in the standard model.

Why are two-party protocol problems so difficult?

The reason two-party protocol problems are so difficult is due to a simple symmetry condition on what players know about each others data. Both parties possess the entire transcript of the conversation that has taken place between them. Thus, in a protocol between A and B, A can determine exactly what B knows about A's data. Because of this knowledge symmetry condition, there are impossibility proofs for seemingly trivial problems. Cryptographic protocols "cheat" by setting up situations in which A may determine exactly what B can infer about her data, from an information theoretic point of view, but does *not* know what he can easily (i.e. in probabilistic polynomial

time) infer about her data. From an information theoretic point of view, of course, nothing has been accomplished. A message encrypted using a one-way permutation, for example, might as well be sent in the clear in terms of information theory.

1.2 The motivation for our work.

Given the impossibility of solving a host of two-party protocol problems (in the information theoretic sense), some natural questions arise:

- What can we say about the structure of two-party protocol problems? As with conventional complexity classes, we can ask about which problems are reducible to which others.
- Suppose we have some way of breaking the knowledge symmetry condition. What problems become tractable?
- Can knowledge symmetry breaking be implemented and used as a building block in multi-party scenarios?

This abstract represents a first step towards answering these questions. We consider a very simple game between two parties, known as *oblivious transfer*. This game, first introduced by Rabin [R], is one of the simplest games which violates the symmetry condition. We study how this game applies to some much more sophisticated problems: *Oblivious Circuit Evaluation* [Y], and *Zero-Knowledge Proof Systems* [GMR,GMW,FM]. We also note that this work can be applied to multi-prover scenarios.

1.3 Previous Work

Our work touches on the areas of oblivious transfer, oblivious transfer, and oblivious circuit computation. This represents a vast body of research, which we will try to briefly review here.

Oblivious transfer was first implemented in the cryptographic scenario by Rabin[R]. This implementation was based on the difficulty of factoring, and only worked for honest parties. Fischer, Micali, and Rackoff[FMR] presented the first oblivious transfer protcol based on factoring and robust against computationally bounded adversaries. This protocol was also based on the difficulty of factoring. Finally, Even,Goldreich, and Lempel[EGL] reduced the intractibility assumption to the existence of trapdoor permutations. These reductions are all cryptographic in nature.

Oblivious circuit evaluation, first introduced by Yao[Y], is a technique whereby two parties A, who knows some number i, and B, who knows some number j, evaluates a circuit $C(i,j)$. At the end of the protocol, player B learns the value of $C(i,j)$, but learns nothing about i but what he can infer from knowing j and $C(i,j)$. Player A learns nothing. In this paper, we are exclusively interested in the two-party scenario.

Several implementation of oblivious circuit evaluation in the two-party scenario appear in the literature. Yao's first implementation relies on the intractibility of factoring. Later, Goldreich, Micali, and Wigderson[GMW] implemented oblivious circuit evaluation based on the existence of trapdoor permutations. Efficiency improvements appear in Abadi-Feigenbaum[AF], Galil-Haber-Yung[GHY], and Goldreich-Vainish[GV]. Chaum-Damgard-Van de Graaf[CDV] address the issue of protecting an individual's privacy information theoretically.

These implementations rely on cryptographic assumptions other than existence of oblivious transfer. Goldreich-Vainish gives a very simple information-theoretic reduction from oblivious circuit evaluation to oblivious transfer. However, this reduction only works in the case where all the parties involved are honest.

Interactive proof systems, and zero knowledge proof systems were introduced by Goldwasser, Micali, and Rackoff[GMR]. Goldreich, Micali, and Wigderson[GMW1] showed that if one-way functions exist, then every language in NP has a zero knowledge proof system. Brassard and Crepeau proved a similar theorem for the case where the prover is assumed to run in polynomial time.

In our paper, we consider versions of zero knowledge proofs[GMR] which do not contain any interaction. As with oblivious circuit evaluation, completely noninteractive zero-knowledge proofs are impossible to accomplish in the information theoretic sense (unless $\mathcal{BPP} = \mathcal{NP}$). Indeed, it is not at all clear that the notion of a completely non-interactive zero-knowledge proof is even viable: If a verifier is convinced by a proof, he can give it to someone else, who should be equally convinced. This is true even in the cryptographic scenario. However, Blum, Feldman, and Micali [BFM]have

achieved in the cryptographic scenario what is perhaps the closest thing to a completely noninteractive zero knowledge proof system for $\mathcal{NP}$. They only require the two parties to agree on a short random string. Furthermore, this string can be reused essentially for the lifetime of the two parties. Naturally, their protocol is not zero-knowledge in the information theoretic sense. As a corollary to results by [F] and [BHZ], it is impossible to have statistically zero knowledge proofs, even with interaction, for $\mathcal{NP}$ unless the polynomial time hierarchy collapses.

1.4 Main Results.

1.4.1 Results outlined in this abstract.

Theorem 1: Let **OT** be a protocol which implements oblivious transfer. There exists a protocol which, using **OT** as a subprotocol, implements oblivious circuit evaluation.

Theorem 2: Using **OT**, there exists noninteractive zero-knowledge proofs for NP, which rely on no preprocessing or complexity theoretic assumptions.:

(Definitions for oblivious circuit evaluation and oblivious transfer may be found in section 2.1.)

1.4.2 Further applications of our work.

Our theoretical investigation into the power of oblivious transfer have already yielded some spinoffs. We have been able to use our techniques to prove theorems in models which do not assume an **OT** subprotocol. In one case, we have been able to prove an unconditional theorem, in another we have been able to easily weaken an intractability assumption.

Theorem 3: Using **OT**, any single or multiprover interactive proof system can be converted into a zero-knowledge interactive proof system.:

Ben-Or, Goldwasser, Kilian, and Wigderson[BGKW] show how to implement **OT** in the two-prover scenario, and thus, as an application of theorem 3, prove a theorem without any assumptions. The proof of Theorem 3 is sketched in that paper.

Aesthetically, it is nice to be able to compartmentalize the intractibility assumptions one makes in a proof. By reducing a large number of protocols to a simple primitive, new implementations of the primitive automatically give new implementations of the more complicated protocols. This technique can used to simplify proofs of protocol results for the cryptographic model. For example, a recent paper by Chaum, Damgard, and van de Graaf gives the following theorems about two-party protocols(they primarily consider the multiparty scenario):

Theorems[CDG]: Suppose that the quadratic residuosity problem is hard. Then there exists a protocol for oblivious circuit evaluation in which one of the player's inputs is protected information theoretically. Furthermore, IP is in zero-knowledge.:

Since there is a protocol due to Fischer, Micali, and Rackoff [FMR] that implements **OT** based on the assumption that factoring is hard, and which protects one player information theoretically, Theorems 1 and 3 immediately imply a version of this theorem, based on the difficulty of factoring. [1]

Finally, for applications in which the number of rounds of communication is important, the following lemma may be of use.

Lemma 4: Let f be an NC^1 circuit. Using **OT** There exists an oblivious ciruit evaluation protocol which uses only a constant number of rounds.:

1.5 Techniques Used.

In the proof of Theorem 1, we use several standard tricks:

Protocols for Commitment, Consistency: A classic protocol in cryptography is one in which some player, Alice, sends a representation $B(b,r)$ (where r is a random string) of a bit b to another player, Bob. Bob cannot determine b by himself, but with the help of Alice, he can at some later time verify that Alice really meant to send the representation of b, and not $\bar{b}$. This is known as *commitment*. In some cases, Alice may not want to reveal what bits were represented by $B(b_1,r_1)$ or $B(b_2,r_2)$, but may wish to prove that $b_1 = b_2$. This is known as *consistency*. Using oblivious

[1] It should be noted, in fairness to [CDG], that despite its reliance on the quadratic residuosity assumption, much of its technique is very general. There is some overlap between the techniques used in this paper and those in [CDG].

transfer, we develop a protocol which allows Alice to commit a set of bits, $X = \{x_1, \ldots, x_n\}$, along with a zero knowledge proof of some NP predicate on X.

The equivalence of oblivious transfer and 1-2 oblivious string transfer: We use theorems of Crépeau[C2] and Brassard-Crépeau-Robert[BCR] that together imply we can assume the following primitive: Let Alice have two strings S_1 and S_2. Using **OT**, there is a protocol whereby Bob can select exactly one of these strings, without learning anything about the other, and without Alice knowing which string he selected.

A theorem of Barrington: In our proof of Lemma 4, we use Barrington's result that NC^1 languages are recognizable by a special clss of bounded-width branching programs.

1.6 Outline of the Paper

We do not have space to give more than a sketch of our protocols, and informally argue that they have the desired properties. In section 2, we give the definitions necessary to understand the statement of our results. In section 3, we develop a protocol for bit commitment with zero-knowledge proofs, and use it to outline a proof of Theorem 2. In section 4, we give an outline of our proof of Theorem 1. In section In sections 5 and 6, we give acknowledgements and references.

2 Definitions

In this section we define *oblivious transfer*, *1-2 oblivious transfer with strings*, and *oblivious circuit evaluation*. We also give a formal deifinition of width 5 permutation branching programs, á la Barrington[B], and a randomized representation for branching programs.

2.1 Protocol Definitions

Oblivious Transfer: In this protocol, Alice has a secret bit, b. At the end of the protocol, one of the following two events occurs, each with probability $\frac{1}{2}$.

(1) Bob learns the value of b.

(2) Bob gains no further information about the value of b (other than what Bob knew before the protocol).

At the end of the protocol, Bob knows which of these two events actually occurred, and Alice learns nothing.

Less formally, we can view this protocol as one in which Alice sends a letter to Bob, which arrives exactly half the time.

1-2 Oblivious String Transfer: In this protocol, Alice has two strings, S_0 and S_1. Bob has a selection bit, s. At the end of the protocol, the following three conditions hold.

(1) Bob learns the value of S_s.

(2) Bob gains no further information about the value of S_{1-s}.

(3) Alice learns nothing about the value of s.

Less formally, Alice has two secret strings. Bob can select exactly one of them, and Alice doesn't know which secret Bob selected.

Oblivious Circuit Evaluation: In this protocol, Alice has some secret, i, and Bob has some secret, j. Both have agreed on some circuit f. At the end of the protocol, the following three conditions hols.

(1) Bob learns the value of $f(i, j)$.

(2) Bob learns no further information about j (other than that revealed by knowing i, $f(i, j)$).

(3) Alice learns nothing about i or $f(i, j)$.

2.2 Width 5 Permutation Branching Programs and their representation.

We can define width 5 permutation branching programs in the following manner.

Definition: A width 5 permutation branching program M of length k, with inputs $x_1, \ldots, x_n$ is a tuple (P, V), defined by

$$\Pi : \{1, \ldots, k\} \times \{0, 1\} \to \mathbf{S}_5$$
$$V : \{1, \ldots, k\} \to \{1, \ldots, n\}$$

where $\mathbf{S}_5$ is the group of permutations on 5 elements. Informally, Π designates the permutations at each level of the branching program, and V designates which variables are being looked at. We can define the evaluation of this program, $M(x_1, \ldots, x_n)$ by

$$M(x_1, \ldots, x_n) = E(\Pi(1, x_{V(1)}) \circ \cdots \circ \Pi(k, x_{V(k)})).$$

where $E(x \in \mathbf{S}_5)$ is equal to 0 if x is the identity permutation, and 1 otherwise. For convenience, we hereafter refer to width 5 permutation branching programs as W5PBP's. A classic theorem of Barrington[B] states that any boolean function in NC^1 can be computed by a polynomial length W5PBP. In fact, we can require that $(\Pi(1, x_{V(1)}) \circ \Pi(2, x_{V(2)}) \circ \cdots \circ \Pi(k, x_{V(k)})$ always evaluates to the identity permutation, or some fixed final permutation f.

2.3 Randomizing W5PBP's.

We can randomize the permutations Π of a W5PBP in the following manner. Let vector $\vec{R} = [r_1, \ldots, r_{k-1}]$, where $r_i \in \mathbf{S}_5$. Given some W5PBP $M = (\Pi, V)$, we define $M_R = (\Pi_R, V)$, where Π_R is defined by

$$\Pi_R(i,j) = r_{i-1}^{-1} \Pi(i,j) r_i,$$

where $1 \leq i \leq k, j \in \{0,1\}$, and r_0, r_k are defined to be the identity permutation. This randomized representation is useful because it is faithful to the original W5PBP. Evaluating the randomized W5PBP yields the same permutation as evaluating the original.

3 Commital and Zero Knowledge Proofs.

Commital protocols may be informally described as follows. Consider two players, A and B. Player A wishes to bet on a boxing match, and B is a bookie who has connections with the Chicago underworld . Player A is willing to place his bet with B, but does not want B to be able to fix the match against her. Player B, on the other hand, is not willing to accept A's bet if it is placed after the match is over. A commital protocol allows player A to place her bet in such a way that B does not find out what it is until the match is over.

Commital protocols have an initial *commit phase* and a later *decommit phase*. A inputs some value x to the commit phase, which B learns nothing about during the protocol. In the decommit phase, B learns x, and is convinced that x was indeed the input to the protocol.

Commital protocols have proven very useful in the theory of zero knowledge proofs [GMW1, BC] and secure multiparty communication[GMW2, CDG, BGW, CCD]. Unfortunately, all of these implementations of commital protocols are either based on cryptographic assumptions, or only work in the multi-party scenario. Indeed, information theoretic commitment/decommitment cannot be implemented in the two-party scenario. In this section, we show how, given an oblivious transfer channel, one can implement a very strong commital/decommital protocol. As well as committing a set of values $X = \{x_1, \ldots, x_k\}$, player A can also give a zero knowledge proof of any NP statement about X. Individual values, x_i, may be decommitted without compromising the security of any values which have not been decommitted. Both the commital and subsequent decommital phases of the protocol are noninteractive.

3.1 A simple commital protocol using oblivious transfer.

As a building block, we use the following simple protocol, due to Crépeau[C], for commiting a single bit, b. Here, p is the probability that a bit transmitted by the oblivious transfer channel is received, and k is the security parameter.

Simple-Commit(b,k,p)
A picks a random $\vec{b_c} = b_1, \ldots, b_k$, subject to $b = b_1 \oplus b_2 \oplus \cdots \oplus b_k$. A send $\vec{b_c}$ to B through the oblivious transfer channel (with transmission probability p). We denote as $\vec{b'}$ the vector that B actually receives through the oblivious transfer channel. Vector $\vec{b'}$ is of type $\{0,1,\#\}^k$, where the symbol # denotes a bit that was not received.

Simple-Decommit$(\vec{b_c}, \vec{b'}, k)$
A sends $\vec{b_c} = b_1, \ldots, b_k$ to B through a clear channel. If $\vec{b_c}$ and $\vec{b'}$ disagree on any of their defined bits, then B aborts the protocol. Otherwise, B computes $b = b_1 \oplus b_2 \oplus \cdots \oplus b_k$.

We now analyze the performance of the above protocol. Ideally, we would like the probability that player B can recover b after the commit phase, and the probability that player A can cause B to compute a bit other than what she originally committed, to both be equal to 0. These goals are also impossible to acheive, even with an oblivious transfer channel, but we can approximate the ideal case sufficiently for our purposes. Using simple probablity, we can show the following lemma.

Lemma 3.1: At the end of **Simple-Commit**(b, k, p), the probability that B can re-

cover b is p^k. In **Simple-Decommit**$(\vec{b}', k, p)$The probability that A can cause B to recover different bit than what she committed is at most $1-p$.:

The probability that B can "cheat" can be made exponentially small by using a sufficiently large security parameter k. The probability that A can successfully cheat can be made exponentially small by running several parallel commital protocols on the same bit. For this paper, we view exponentially small probabilities as being equivalent to 0 for all practical purposes.

We now show how to enhance this protocol, giving A the ability to prove satements about the bits it is committing. To do this, we first introduce the notion of *permutation randomizers*, *permutation condensors*, and *permutation tableaus*.

3.2 Permutation randomizers, condensors, and tableaus.

Definition 1 Fix m, n. We define a *permutation randomizer* $R \in \mathcal{R}_n^m$ to be an (m(n-1)+1) by n array of permutations in $\mathbf{S}_5$, with the property that for all $i, 0 \leq i < m$ and $j, 0 \leq j < n-1$, the following two constraints hold.

1. If $k \neq j, j+1$, then $R_{i(n-1)+j,k} = R_{i(n-1)+j+1,k}$.
2. Let $l = i(n-1)+j$. Then, $R_{l+1,j}R_{l+1,j+1} = R_{l,j}R_{l,j+1}$

Thus, if row $i(n-1)+j (0 \leq i < m, 0 \leq j < n-1)$ of a permutation randomizer R, is of the form $p_0, p_1, \ldots, p_{n-1}$, then row $i(n-1)+j+1$ will be of the form $p_0, p_1, \ldots, p_{j-1}, p'_j, p'_{j+1}, p_{j+2}, \ldots, p_{n-1}$.

Definition 2 Let $S = s_0, \ldots, s_{n-1}$ be a sequence of elements of $\mathbf{S}_5$. We define the distribution $\mathcal{R}_n^m[S]$ as the uniform distribution on $\{R | R \in \mathcal{R}_n^m, (\forall i, 0 \leq i < n) R_{0,i} = s_i\}$.

The distribution $\mathcal{R}_n^m[S]$ is simply the uniform distribution on permutation randomizers whose first row is equal to S.

Definition 3 Fix $n = 2^k$. We define a *permutation condensor* $C \in \mathcal{C}_n$ to be a $k+1$ by n array of permutations in $\mathbf{S}_5$, subject to $C_{i+1,j} = C_{i,2j}C_{i,2j+1}$.

We may view a permutation condensor as a complete binary tree, where the permutation assigned to each interior node of the tree is equal to the product of the node's two children.

We can combine these permutation randomizers and permutation condensors into a single entitiy, which we call a *permutation tableau*.

Definition 4 Fix m and $n = 2^k$. We define a *permutation tableau* $T \in \mathcal{T}_n^m$ to be an ordered pair (R, C), where $R \in \mathcal{R}_n^m$, $C \in \mathcal{C}_n$, and $C_{0,i} = R_{m(n-1),i}$ for $0 \leq i < n$.

Definition 5 Let $S = s_0, \ldots, s_{n-1}$ be a sequence of elements of $\mathbf{S}_5$. We define the distribution $\mathcal{T}_n^m[S]$ as the uniform distribution on

$$\{T | T = (R, C) \in \mathcal{T}_n^m, R \in \mathcal{R}_n^m[S]\}$$

3.3 Properties of permutation randomizers, condensors, and tableaus.

The following lemmas establish the basic properties we require of our permutation tableaus. Their proofs are all elementary, and are ommitted. First, we have the following invariance properties.

Lemma 3.2: Let $R \in \mathcal{R}_n^m$. Then for $0 \leq i, j, \leq m(n-1)$,

$$\Pi_{l=1}^n R_{i,l} = \Pi_{l=1}^n R_{j,l}. \quad \square$$

:

Lemma 3.3: Let $C \in \mathcal{C}_n$. Then for $0 \leq i, j \leq k$,

$$\Pi_{l=1}^{n/2^i} R_{i,l} = \Pi_{k=1}^{n/2^j} R_{j,l}. \quad \square$$

:

As a consequence of these lemmas, all the rows in a permutation tableau will multiply out to the same value. In particular, if we have $T = (R, C) \in \mathcal{T}_n^m[S]$, where $S = s_0, \ldots, s_{n-1}$, and $n = 2^k$, then

$$C_{k,0} = s_0 \oplus \cdots \oplus s_{n-1}.$$

Next, we have the following randomization property.

Lemma 3.4: Let $i \geq j + (n-1)$, and let R be chosen from $\mathcal{R}_n^m[S]$, where $S = s_0, \ldots, s_{n-1}$. Then rows R_i and R_j will be distributed uniformly, subject to

$$\prod_{l=1}^{n} R_{i,l} = \prod_{l=1}^{n} R_{j,l} = \prod_{l=1}^{n} s_l. \quad \Box$$

:

In other words, we can view R as a randomizing process which, after a sufficiently many stages, completely scrambles a sequence of permutation, leaving only the product of the elements invariant. Another useful property of permutation randomizers is their columnwise independence.

Lemma 3.5: Let $(c_1, \ldots, c_l)$ be a set of column indices. Let $S = s_0, \ldots, s_{n-1}$, and $R \in \mathcal{R}_n^m[S]$. The joint distribution of columns $R_{c_1}, \ldots, R_{c_l}$ will depend only on $s_{c_1}, \ldots, s_{c_l}$. $\Box$:

Thus, if it is impossible to determine anything about s_i from knowing $s_{c_1}, \ldots, s_{c_l}$, it is impossible to determine anything about s_i from knowing $R_{c_1}, \ldots, R_{c_l}$. Finally, we have the following checkability condition for permutation tableaus.

Lemma 3.6: Fix m and $n = 2^k$. Let $T' = (R', C')$, where R' is an $m(n-1)+1$ by n matrix, and C' is a $k+1$ by n matrix. If T' is *not* in $\mathcal{T}_n^m[S]$, then there exist four positions in T' such that no $T \in \mathcal{T}_n^m[S]$ agrees with T on all four positions. $\Box$:

In other words, one can expose a bad tableau by revealing only four of its elements. This is simply because the property of being a valid tableau for a permutation sequence can be expressed as a list of local constraints, each of which refer to only a small number of elements.

3.4 A commital protocol which allows zero knowledge proofs.

In this section we outline our commital/zero knowledge proof protocol. This protocol allows one to commit a set of bits $X = \{x_1, \ldots, x_n\}$, along with a zero knowledge proof of any NP predicate on these bits. For example, one could give a zero knowledge proof that the bits of X represent a prime number, without revealing any extra information about X. First, we give a commital/zero knowledge proof protocol which works for predicates in NC^1. We then use a simple trick to expand our protocol to cover general NP predicates.

Our commital protocol uses the same representation for bits as does **Simple Commit**. That is, each bit x_i is represented as a sequence of bits, $x_i = x_{i,1} \oplus \cdots \oplus x_{i,k}$. We denote by $[X]$ the matrix $x_{i,j}$ so chosen. Let P be any NC^1 predicate on X. We denote by P' the corresponding predicate on $[X]$, . Clearly, P' is also in NC^1. Thus, there is a W5PBP $M_{P'} = (\Pi, V)$ of size $m = 2^l \leq (nk)^c$, for some c, which evaluates P' on $[X]$. Note that we can pad the branching program to force the length of M to be a power of 2. We also require that $m \geq nk$, which is also easily acheived by padding. Since $[X]$ is a two-dimensional matrix, we have function V be of type $\{1, \ldots, m\} \rightarrow \{1, \ldots, n\} \times \{1, \ldots, k\}$. This deviation from the definition given in Section 2 is purely notational. The following notation is useful in the exposition of our protocol.

Definition 6 Let $[X]$ be a boolean n by k matrix, $x_{i,j}$, and let $M = (\Pi, V)$ be a W5PBP on $[X]$, of length $m = 2^l$. We define $S(M, [X])$ as the sequence $s_1, \ldots, s_m$, where $s_i = \Pi(i, x_{V(i)})$.

We now describe our new protocol.

Strong-Commit(X, P, k)
Let $X = \{x_1, \ldots, x_n\}$. Let $P', M_{P'} = (\Pi, V)$ be defined as above. Let m be the length of $M_{P'}$. Player A picks an n by k random matrix $[X]$, subject to $x_i = x_{i,1} \oplus \cdots \oplus x_{i,k}$. Let $S = S(M, [X])$. Player A then picks a random tableau $T \in \mathcal{T}_m^{m_2}[S]$. Player A then sends $([X], T)$ to player B, using an oblivious transfer channel which transmits a bit with probability $1/m^3$. As with **Simple-Commit**, We denote by $([X'], T')$ the message actually received by B.

Strong-Decommit$(i, [X], [X'])$
A sends row $X_i = x_{i,1}, \ldots, x_{i,k}$ to B through a clear channel. If there is any discrepancy between row X_i and $[X']$, then B rejects. Otherwise, B computes $x_i = x_{i,1} \oplus \cdots \oplus x_{i,k}$.

Check-Proof$([X'], T', M_{P'})$
Let $T' = (R', C')$, $M_{P'} = (\Pi, V)$, and $m = 2^l$. Recall that entry $x'_{i,j}$ of $[X']$ may or not be defined. Elements $R'_{i,j}$ and $C'_{i,j}$ may be fully defined, undefined, or partially defined (some but not all of the bits were received). For the checking protocol, partially defined entries are considered to be undefined. Player B makes the following checks, wherever all the elements involved are defined.

- If $k \neq j, j+1$, then $R_{i(n-1)+j,k} = R_{i(n-1)+j+1,k}$.

- For $q = i(m^2-1)+j$, $R_{q+1,j}R_{q+1,j+1} = R_{q,j}R_{q,j+1}$
- $C_{i+1,j} = C_{i,2j}C_{i,2j+1}$
- $C_{0,i} = R_{m^2(m-1),i}$
- $R_{0,i} = \Pi(i, x_{V(i)})$
- $C_{l,0} \neq I$, the identity element of $\mathbf{S}_5$.

If any of these conditions are violated, then B rejects the proof. If no visible violations occur, then B accepts.

3.5 Properties of the strong commital protocol.

We now argue that the commital protocol described above is indeed a commital protocol, and furthermore gives a zero knowledge proof of desired assertion. It is not hard to sow that A cannot cheat effectively.

Lemma 3.7: The probability that A can cause B to recover a bit other than what was originally committed is at most $1 - 1/m^3$.:

Proof: In order to trick B, A must make one of the bits in her decommital transmission inconsistent with the corresponding bit in her commital transmission. This inconsistancy will be detected if B received the original bit in the commital transmission, which will happen with probability $1/m^3$. □

Lemma 3.8: The probability that A can cause B to accept an incorrect proof is at most $1 - 1/m^{84}$.:

Proof: In order to trick B, A must either give him an inconsistent tableau, or a consistent tableau in which $C_{l,0} = I$ (corresponding to a nonaccepting branching program). However, using Lemma 3.6, we have that in either case, there is a set of 4 positions in the tableau, which if revealed would expose A. Each position takes only 7 bits to encode. If all of these 28 bits are received by B, which will occur with probability $1/m^{84}$, then A will be exposed. □

The probability that A can cheat, once bounded away from 1, can be made exponentially small by the trick of running the commital protocol many time in parallel for each bit committed.

We now wish to show that B gets no significant extra information from $[X]'$ and T'. We first show that, with probability close to 1, B will receive less than k bits of $[X]$ and T. Using elementary probability, we can prove the following lemma.

Lemma 3.9: The probability of B receiving more than i bits is at most c/m^i, where c is some absolute constant.:

For the rest of this section, we consider only the case where B receives only $k-$ bits, which, for k large, will be the case almost all the time. We also consider a slightly harder model, where B can designate ahead of time, $Q = q_1, \ldots, q_{k-1}$ positions, either from $[X]$ or T, to be looked at. We allow B to see the entire contents of $[X]$ and T in each of these positions. Clearly, if B receives no significant information in this model, he will not receive any significant information in the model where his views are possibly incomplete, and randomly located. We will show that B's best strategy is equivalent to looking at $k-1$ locations of $[X]$, which tells him nothing. To formalize this notion, we introduce the following definition.

Definition 7 Let $[X]$ be an arbitrary bit-commital matrix chosen by A. Let $Q = q_1, \ldots, q_{k-1}$ be a set of positions (either positions in the bit-commital matrix, or positions in the tableau). We define $View([X], Q)$ to be the distribution of views B sees, obtained by choosing tableau T according to the **Strong-Commit** protocol, and sending B the values of $[X]$ and $T = (R, C)$ at the position specified by P.

We now sketch a proof that, for any choice of Q, one can realize the distribution $View([X], Q)$, given the ability to query $k-1$ bits of $[X]$.

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

Lemma 3.10: For any position set Q of size $k-1$, there exists some i such that rows R_i, R_{i+1}, R_{i+m} of the randomizer R are not looked at by Q. □:

This "gap property" turns out to be very useful in realizing $View([X], Q)$. We can break our view into two sections, as follows.

Definition 8 We denote by $Q_{>i}$ those positions of Q which fall into row R_j, for $j > i$, or fall into C. We denote by $Q_{\leq i}$ those positions of Q which fall into row R_j for $j \leq i$, or fall into $[X]$.

If i has the property that R_i, R_{i+1}, R_{i+m} are not looked at by Q, we get the following useful decomposition of our simulation problem. Using Lemma 3.4, we see that rows $R_{i+m}, R_{i+m+1}, ...$, and the distribution of C are independent from $[X]$ and rows $R_0, \ldots, R_{i-1}$. Thus, $View([X], Q_{>i+m})$ is a fixed distribution, not dependent on $[X]$, and independent of $View([X], Q \leq i$. We can simulate $View([X], Q_{>i+m})$ by picking uniformly a simulated row, R_{i+m}, subject to the constraint $R_{i+m,0} \oplus \cdots \oplus R_{i+m,m-1}$. We can then esily simulate the distribution of rows $R_{i+m+1}, R_{i+m+2}, \ldots$, and C. Now, we can trivially simulate any peeks into this region of the tableau, since we have simulated the entire section of the tableau. Note that this simulation doesn't even require any queries to $[X]$.

We now have to simulate $View([X], Q_{\leq i}$. We first note that R is picked uniformly from $\mathcal{R}_n^m[S]$, where $S = s_1, \ldots, s_m = S(M_{P'}, [X])$. Let $c_1, \ldots, c_l$ be the set of columns mentioned in P. It suffices to simulate the joint distribution of these columns. By Lemma 3.5, we have that the joint distribution of columns $R_{c_1}, \ldots, R_{c_l}$ depends only on $s_{c_1}, \ldots, s_{c_l}$. However, s_i depends only on $x_{V(i)}$, where $M_{P'} = (\Pi, V)$. Thus, we can simulate any l requests from $Q \leq i$ to look at position R, with at most l queries to $[X]$. Finally, each request by $Q \leq i$ to look at $[X]$ can be directly "simulated" by making a simple query to $[X]$.

Of course, making $k-1$ queries to $[X]$ doesn't allow B to guess the value of any of the original x_i's, which are represented as the exclusive or of k bits of $[X]$, any better than by pure chance.

3.6 Zero knowledge proofs of NP statements.

We can trivially extend our commital protocol, which allows NC^1 predicates to be proven in zero-knowledge, to yield a protocol which allows one to prove arbitrary NP predicates. We can allow people to commit bits they have no intention of revealing, but which can be used in the zero knowledge proof. This allows one to prove predicates which are in *nondetermnistic* NC^1, by committing the values of the "nondeterministic" inputs to the circuit, along with the bits one is actually using. Note that nondeterministic NC^1 contains NP.

3.7 Noninteractive zero knowledge proofs of NP.

The machinery built up in this section yields Theorem 2 as a simple corollary. Given some $x \in L$, where L is a language in NP, one can commit a set of bits w, along with a proof that w is a witness for x. Note that protocol **Strong-Commit** is noninteractive.

3.8 Oblivious Decommital

As a last, simple trick of this section, we note that one can combine the strong committal protocol with that of oblivious string transfer to get a protocol for oblivious decommital of strings. In this protocol, A commits a set of strings, along with a zero knowledge proof for some property of these strings. At some later point, A may offer to decommit one of two strings for B. be to B chooses which string he wishes A to decommit. At the end of the protocol, player A decommits the string B selects, but doesn't know which one she has decommitted.

4 Reducing Oblivious Circuit Evaluation to Oblivious Transfer

In this section, we sketch how to perform oblivious circuit evaluation using oblivious transfer as a given subprotocol. This reduction proceeds in two steps. First, we show a constant round reduction from oblivious NC^1 evaluation to oblivious transfer. We then reduce oblivious circuit evaluation to oblivious NC^1 evaluation.

Caveat: As this is an extended abstract, we do not even pretend to give complete constructions, much less formal correctness proofs (this would take tens of thousands of bytes). This section should be viewed as a statement of results, with some hints at the techniques used.

4.1 A protocol for oblivous NC^1 circuit evaluation.

The main technique used for oblivious NC^1 circuit evaluation is the use of randomized branching programs, as discussed in Section 2. There is

a simple protocol which works in the friendly but curious model. This model assumes that both participants follow the protocol, but may later try to derive extra information based on what they see.

Given some NC^1 boolean function $f(x_1, \ldots, x_n)$, of which player A knows some of the arguments, and player B knows the rest. First, players A and B decide on some canonical W5PBP for f, of size k, which we denote by M^f. Player A then picks M^f_R, a randomized representation of M^f, uniformly at random. For all i, $1 \leq i \leq k$, such that A knows $x_{V(i)}$, A sends over $\Pi_R(i, x_{V(i)})$. For all i, $1 \leq i \leq k$, such that B knows $x_{V(i)}$, A and B invoke the 1-2 oblivious string transfer subprotocol. Player A's two secrets are $\Pi_R(i, 0)$ and $\Pi_R(i, 1)$. Player B selects $\Pi_R(i, x_{V(i)})$.

At this point, we wish to make the following three claims.

- Player B can now compute $f(x_1, \ldots, x_n)$.
- Player B does not learn anything more about A arguments to f (other than what he learns by knowing $f(x_1, \ldots, x_n)$),
- Player A learns nothing

Player B can compute $f(x_1, \ldots, x_n)$ by simply multiplying all the permutations, and seeing if they give the identity or the "accept" permutation. Player A learns nothing, since she learns nothing from the 1-2 oblivious transfers with strings. We now wish to informally argue that Player B learns nothing more than $f(x_1, \ldots, x_n))$. Let $View_B(R, x_1, \ldots, x_n)$ be a random variable on what B learns in the protocol, namely $\Pi_R(i, x_{V(i)})$. It is not hard to show that $View_B$ can be generated simply by knowing the values of those variables which B knows, and knowing $f(x_1, \ldots, x_n)$. In the full paper, we give an explicit construction for this distibution.

Finally, if we can obliviously evaluate an NC^1 boolean circuit, we can easily convert this into a protocol for evaluating general NC^1 circuits.

4.2 Dealing with malicious players.

The above protocol does not work against malicious players. For one thing, A is taking B's word that he is the permutations he selects are consistent with a partial variable assignment. Also, B is taking A's word that the permutations he is receiving correspond to a randomized W5PBP. We do not have to worry about A getting any information. However, we do have to worry about B cheating and getting information, and about A cheating so as to manipulate the outcome of the protocol.

It is easy to prevent B from cheating. Instead of allowing B to pick seperately the permutations corresponding to one of his variables, and thus possibly pick an inconsistent set of permutations, we have him pick all the permutations corresponding to that variable. The more serious problem is how to keep A honest. However, using the machinery of section 3, this can be taken care of in an almost automatic fashion. We can require B to commit his branching program, and his choice of permutations, along with a zero knowledge proof that they correspond to the correct branching program, and a legal variable assignment. We then replace all the olivious transfer protocols with oblivious deecommital protocols.

3.2 Reducing Oblivious Circuit Evaluation to Oblivious NC^1 Circuit Evaluation.

Here we give a brief idea of how to solve the general oblivious circuit evaluation given the NC^1 case. First, we note that one can break up a depth $d(n)$ circuit into $s(n) = O(d(n)/\log n)$ "slices" which are NC^1 circuits. Slice 1 takes as input the original inputs to the circuit, and slice i takes as inputs the outputs of slice $i-1$. This suggests the following idea. Have A and B simulate slice 1. Once B knows the output of slice 1, he possesses all the inputs to slice 2, and can continue on from there by himself. This scheme does not work for obvious reasons: knowing the outputs of slice 1 gives one extra information.

The solution to this problem is use an idea from [BGKW], that of *encrypted conversations*. Encrypting conversations is a general technique, by which conversations in which all the results of intermediate computations are encrypted in an information theoretic sense. Thus, B gets no information from them. Furthermore, he is forced to input the results from step i of the computation to the input of step $i+1$, thus simulating the original conversation. There is, however, a consistency checking system built in, which insures that A must faithfully simulate the conversation, or be caught by B.

Using encrypted conversation, one can "hide" the output of stages 1 through s(n)-1, and only see

the output of the last stage. Hence, we can chop the circuit up into NC^1 chunks, and solve them successively.

Wait a minute!

At this point, an embarrassing circularity arises. We invoke this magical method of hiding the results of intermediate computations. This technique, we claim, is the key idea needed to make our theorem go through. However, if one consults [BGKW], one finds that the conversation encryption technique presupposes oblivious circuit evaluation! However, one can implement all required routines for encryption, decryption, and handling capsules in NC^1. Hence, our subprotocol for oblivious NC^1 circuit evaluation is sufficient. [2]

An earlier proof of Theorem 1, and a different proof based on the techniques of [CDG], gives nearly as strong a theorem, but requires asymptotically more rounds of communication. Our techniques randomize on the level of entire NC^1 circuits, rather than on single gates, which gives an $O(\log n)$ "speedup" in the number of rounds we use. Due to its very "fast" implementation, and the power of NC^1, Lemma 4 may be of independent interest in protocol design.

4.3 The nature of our simulators for oblivious circuit evluation.

We do not have space here to include the explicit construction of the simulator for our oblivious circuit evaluation protocol. Instead, we will briefly discuss some of the more interesting properties of the simulator, and how it can be incorporated into other proofs in a hierarchical fashion. This simplifies the construction of simulators for higher level protocols which use our oblivious circuit evaluation protocol as a subroutine.

We actually must construct two simulators: One which interacts with player A, proving that A learns nothing, and one which interacts with player B, proving that B only learns the output of the circuit. The simulator which interacts with player A in the oblivous transfer protocol is relatively straightforward. The more difficult task is to construct a simulator which interacts with player B.

[2] The result in [BGKW] still requires the ability to perform general oblivious circuit evaluation. The reason for this is that a polynomial time verifier is being simulated, which may not be an NC^1 computation. In our application, each intermediate computation is also in NC^1.

We must address how the simulator simulates oblivous transfer. We assume that the simulator has a communication line which obeys the transfer condition, but does not obey the obliviousness condition. Specifically, the simulator knows which bits were actually received by the other party. For our oblivious transfer protocol, the simulator tells player B which bits it received, and what their values are, so this is easily accomplished. In protocols where oblivious transfer is itself implemented as a subprotocol, this "nonobliviousness" condition for the simulator must be established as a seperate part of the construction.

Our simulator for oblivious circuit evaluation behaves in the following manner. Suppose the circuit computes $f(i,j)$, where B knows j. It first interacts with party B, simulating player A, without any specification of what the value of i and $f(i,j)$ are. At some point, it determines what B's input number j is, if B indeed obeyed the protocol sufficiently to define such an input. If B's behavior did not define any input, the simulator can continue the conversaton by itself. Otherwise, the simulator can report the value of j to some controller, who can at that point decide what the value of $f(i,j)$ should be, and give this value to the simulator. The simulator can then complete the conversation. This behavior corresponds to our notion of how a simulator for oblivious circuit evaluation should behave, and has the added benefit of making it easy to incorporate this simulator into simulators for larger programs.

Finally, our simulator has the interesting property that it never "backs up" the verifier. More standard simulator results, even those for perfect zero knowledge protocols, often assume that the simulator can restore the verifier to an earlier state in the protocol. For instance, it might at time t in the protocol, send the other party a 1, find out later that this leads to a situation where it cannot simulate the conversation anymore, back up to time t, and send the other party a 0. In the case where the other party is a polynomial time bounded Turing machine, this is not unreasonable. However, there are classes of players where this is not the case. If, for example, player B has a friend who will compute exactly 17 discrete logarithms for him, then "backing up" simulator constructions no longer show zero knowledge. It may not be possible to "back up" B's friend, who is otherwise unwilling to compute more than 17 discrete logs. Thus, we can use, and prove results for, a much more general definition of zero knowledge.

5. Acknowledgements

We would like to acknowledge Claude Crepeau, Shafi Goldwasser, and Silvio Micali for their comments, ideas, and assistance with the writing of this paper.

6. References

[AF] Abadi, Martín, Joan Feigenbaum, "A Simple and Efficient Protocol for Secure Circuit Computation," to appear.

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

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

[BCR] Brassard, Gilles, Claude Crépeau, and Jean-Marc Robert. "Minimal Disclosure Protocols," *Proceedings of the* 27^{th} *FOCS*, IEEE, 1986, 168–173.

[BGW] Ben-Or, Michael, Shafi Goldwasser, and Avi Wigderson, "Completeness Theorems for Noncryptographic Fault-tolerant Distributed COmputation" *Proceedings of the* 20^{th} *STOC*, ACM, 1988.

[BGKW] Ben-Or, Michael, Shafi Goldwasser, Joe Kilian, and Avi Wigderson, "Multi-Prover Interactive Proof Systems, Removing Intractibility Assumptions," These proceedings.

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

[BFM] Blum, Manuel, Paul Feldman, and Silvio Micali, "Noninteractive Zero-Knowledge Proofs and their Applications," These Proceedings.

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

[CCD] D. Chaum, C. Crépeau and I. Damgard, Multiparty unconditionally secure protocols, These proceedings.

[CDG] 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.

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

[FM] Feldman, Paul, Silvio Micali. "Byzantine Agreement from Scratch," *Proceedings of the* 20^{th} *STOC*, ACM, 1988.

[FRS] Fortnow, Lance, Mike Sipser, John Rompel, "On the Power of Multi-Prover Interactive Proof Systems," to appear.

[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.

[GMR] Goldwasser, Shafi, Silvio Micali, and Charles Rackoff. "The Knowledge Complexity of Interactive Proof-Systems," *Proceedings of the* 17^{th} *STOC*, ACM, 1985, 291–304.

[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* 27^{th} *FOCS*, IEEE, 1986, 174–187.

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

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

[M] Micali, Silvio, Personal Communication.

[R] Rabin, M., "How to exchange secrets by oblivious transfer", Tech. Memo TR-81, Aiken Computation Laboratory, Harvard University, 1981.

[Y] Yao, Andrew C. "How to Generate and Exchange Secrets," *Proceedings of the* 27^{th} *FOCS*, IEEE, 1986, 162–167.