Achieving Zero-Knowledge Robustly

J. Kilian*

Abstract

We introduce the notion of *robust transformations* of interactive proof systems. A robust transformation takes an interactive proof system (P, V), and produces a new interactive proof system, (P^*, V^*), such that the power of P^* is within a polynomial factor of that of P. We show that, given an ideal protocol for secure circuit evaluation, there exists a robust transformation that converts interactive proof systems to zero-knowledge interactive proof systems.

1 Introduction.

1.1 Background and Motivation.

There exist many general transformations that take an interactive proof system (P, V) for a language L, and produce a new interactive proof system (P^*, V^*) for L, that has some additional desired property. To give a simple example, one may wish to decrease the error probability of the proof system from $\frac{1}{3}$ to $2^{-|x|}$, on input x. In the new protocol, (P^*, V^*) runs (P, V) many times, and then V^* takes a majority vote. To give some slightly more complicated examples, one can,

1. Transform (P, V) into (P^*, V^*), such that V^* never rejects on a valid input [2],
2. Transform (P, V) into (P^*, V^*), such that V^* only uses public coins [3], and,
3. Given a secure cryptographic committal scheme, transform (P, V) into (P^*, V^*), such that (P^*, V^*) achieves zero-knowledge [5].

This paper is motivated by a consideration of how powerful the new prover, P^*, must be in the transformed proof system. In the error reduction transformation, P^* makes a polynomial number of calls to P, and otherwise runs in polynomial time. However, in

*Harvard University and MIT Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139 USA, joek@theory.lcs.mit.edu. Supported by an NSF Postdoctoral Fellowship. Some of the writing up of these results supported by Bell Communications.

A.J. Menezes and S.A. Vanstone (Eds.): Advances in Cryptology - CRYPTO '90, LNCS 537, pp. 313-325, 1991.
© Springer-Verlag Berlin Heidelberg 1991

all of the other transformations given above, there is no obvious relationship between the power of P and P^*. While much progress has been made in finding simple transformations with the above properties, all known techniques involve a possibly exponential time overhead, even if calls to P are for unit cost. In some sense, then, these transformations are nonconstructive.

To capture a more constructive notion of a protocol transformation, we introduce the notion of a *robust transformation*, as follows.

Definition 1 Let $\phi : (P, V) \rightarrow (P^*, V^*)$ be a transformation from interactive proof systems to interactive proof systems. We say that ϕ is robust if,

1. For all (P, V), $\phi(P, V) = (P^*, V^*)$ accepts the same language as (P, V), and,
2. P^* can be evaluated by a probabilistic polynomial-time Turing machine M with access to a black-box evaluator for P.

By a black-box evaluator for P, we mean a function that takes as input a partial conversation, and outputs a distribution equal to that of P, given the partial conversation so far.

1.2 Our result.

Thus far, no robust transformation is known for converting interactive proof systems into Arthur-Merlin form (i.e., turning a private-coin proof into a public coin proof). However, all of the transformations for converting arbitrary interactive proof systems into zero-knowledge interactive proof systems first transform the protocol into Arthur-Merlin form. This remains true even if ideal, information theoretically secure envelopes are allowed.

The contribution of this paper is to show that a stronger cryptographic primitive can indeed allow for a robust transformation of proofs into zero-knowledge proofs. Our theorem is as follows.

Theorem: Suppose we have ideal secure circuit evaluation as a cryptographic primitive. Then there exists a robust transformation that converts interactive proofs to statistically zero-knowledge interactive proofs.

By secure circuit evaluation, we mean the following: Consider a polynomial-sized circuit, $C(i, j)$. Suppose that P has an input i, and V has an input j. We say that P and V securely evaluate C if they achieve the following state of knowledge (ignorance).

1. V learn the value of $C(i, j)$, but gains no additional information about the value of i.

2. P learns nothing about j.

In the ideal, information theoretic scenario, we can imagine a trusted third party, M, who has private communication lines with P and V. P and V send i and j respectively to M. M computes $C(i,j)$, and sends this value to V. As a simple generalization of this, we can make C a probabilistic circuit; this case may be easily reduced to the case where C is deterministic.

1.3 Reducing our theorem to other assumptions.

Our theorem may be compared to the better known result that IP is in zero-knowledge if one has a bit committal primitive. It seems that the primitive required by our theorem is more powerful than simple bit committal. It is worth considering what other assumptions we should be able to reduce our theorem to. There exist information-theoretic reduction from secure circuit computation to oblivious transfer [6]. Along a more standard vein of research, it should be possible to prove an analog to our theorem that relies on a complexity theoretic assumption. Such a theorem follows, with many details, from the existence of a secure circuit evaluation protocol that is,

1. Information theoretically secure against one of the parties (the prover),
2. Simulatable in a cryptographically secure fashion against the other party, using the auxiliary input model for security,[1] and,
3. Playable by two polynomial-time bounded players.

The circuit evaluation protocols of currently in the literature do not achieve this property. However, it seems that one can implement such a protocol based on either,

1. The Diffie and Hellman assumption for discrete logs,[2] or
2. The existence of trapdoor permutations and families of claw free permutations.

Both of these assumptions seem to be much stronger than that required to make cryptographic analogs of envelopes [4, 7]. Working these techniques out in full, one should be able to get the following theorem.

Future Theorem: There exist a robust transformation from interactive proof systems to cryptographically zero-knowledge interactive proof systems, based on either of the above intractability assumptions.

[1] We refer the reader to an upcoming paper of Rogaway and Micali, which gives a very detailed description of what a good simulation should entail.

[2] This assumption is as follows. Suppose one picks a random k bit prime p, a generator g of Z_p^*, and random $x, y \in Z_p^*$. Then it is hard to compute g^{xy}, knowing only p, g, g^x, g^y

There are no foreseen technical difficulties to proving such a theorem. However, a rigorous proof will most likely prove very lengthy and unwieldy. This paper is part of a two or three pronged attack. In addition to this paper, we require a rigorous write up of the various new implementations of secure circuit evaluation, and some more formalisms on using cryptographically secure protocols in place of their ideal protocols. Much of this theory has been sketched out; essentially, this is an exercise in formalizing and writing up a number of "folk theorems."

Another reasonable approach to constructing a cryptographic protocol is to use the blob techniques of Chaum, Damgard, and van de Graaf [1]. By using cryptographically (but not informationally) secure blobs, many of the problems with chaining together computations in a secure way can be avoided. Again, this approach requires that a large body of simulation machinery be developed or rigorously written up. It is also quite possible that their secure circuit evaluation protocol can be shown to have the necessary properties, outlined above.

1.4 Outline of the paper.

In Section 2, we give an overview of the proof of our theorem. In Section 3 we give the construction of our robust zero-knowledge transformation. In Section 4, we show that the transformed protocol remains a proof system, and achieves zero-knowledge.

2 Overview of the proof.

We wish to take an ordinary proof system, (P, V) for a language L, and robustly transform it into a zero-knowledge proof system, (P^*, V^*). To do this, we must deal with the many ways that a malicious or even an honest verifier may obtain information from the original protocol. One easily dealt with difficulty is that the probability with which V accepts a string $x \in L$ may leak additional information about x. This problem is solved by robustly transforming the protocol so as to achieve an exponentially small error probability. In this case, for all $x \in L$, V will accept with probability indistinguishable from 1, thus yielding no extra information about x.[3]

More serious is the fact that the verifier receives vast amounts of potential information from simply being able to see the transcript of its conversation with the prover. The standard trick, first proposed by Ben-Or, is to encrypt the entire conversation, thus concealing it from the verifier. However, it is then unclear how the prover and the verifier can figure out the verifier's questions to the prover. The verifier can't compute them, since it can't see what is going on, and the prover can't compute them, since they may depend on the verifier's private coin tosses. An elegant way around this impasse, also first proposed by

[3]Note that we cannot use any of the known transformations that make this probability equal to 1, since these transformations are not robust.

Ben-Or, is to transform the original interactive proof system into an Arthur-Merlin game. Then, the verifier's questions may be simulated by a straightforward application of bit comitment and coin-tossing protocols. All the known rigorous proofs that IP is in zero-knowledge (modulo complexity assumptions or the existence of ideal encryption schemes) use this general approach. Unfortunately, we know of no provably robust transformation from IP to AM (or even one that is conjectured to be robust), and so must use a different approach.

Instead, we use standard techniques of secure circuit evaluation to get around this impasse in another way. First, we break up the verifier's private information into shares, one held by the prover, and one held by the verifier. Neither share is enough to give any information about the private data, but when both are input to a secure circuit, the shares may be combined and used to simulate the computations that would have been made by the verifier in the original protocol. The answers that the prover would normally give to the verifier are also fed into the secure circuit, thus hiding this other source of information from the verifier. At the end of the protocol, the prover and the verifier can perform a final circuit computation to see if the verifier should have accepted given the simulated conversation thus far.

There are a few technical issues to be addressed. Most important of these is that we must guarantee that neither party can manipulate the protocol by altering their shares of the verifier's history. To make this guarantee, we set up a system of consistency checks, ensuring that any such cheating will with high probability result in the protocol effectively aborting.

3 Construction of the zero-knowledge proof system.

3.1 Breaking the proof up into a series of circuits.

We can view an interactive proof system as follows. Initially, V computes a private history, h_0, that consists solely of a sequence of $|x|^{c_1}$ uniformly distributed bits, for some constant $c_2 1$. Given h_0, V computes its first question q_1. In Step i, V sends question q_i to P, and P sends back an answer, a_i. V computes a new history and a new question, $h_i, q_{i+1} = H_i(h_{i-1}, a_i)$, where H_i is computed by a polynomial size circuit. At the end of $m = |x|^{c_2}$ steps, for some constant c_2, V computes $\text{ACCEPT}(h_m)$, where ACCEPT is computed by a polynomial size circuit, which outputs either a 0 (for reject) or a 1 (for accept). We assume for convenience that all of the inputs and outputs of H_i can be encoded as $n = |x|^{c_3}$ bit strings, for some constant c_3. Thus, for instance, the string R or h_5 can be padded to make it the proper length if it is not long enough. Since x is known to both parties, we may hardwire it into the circuit, and thus we do not explicitly include it as an input.

For our initial construction, we want to only trust V in a very limited way. We do not trust him to flip coins at random, or to correctly perform any computations on his own. Instead, P and V will perform the initialization step, the computations of q_i, h_i, and the

decision to accept or reject via secure circuit computations. We will, however, trust V to give the output of one circuit as the input to another circuit, or as a message to P.

The construction of an initial history h_0 and the first question q_1, can be performed by a secure circuit evaluation, as follows. Let circuit $H_0(R_1, R_2)$ take an $|x|^{c_1}$- bit string, R_1, from P, and an $|x|^{c_1}$-bit string, R_2, from V. $H_0(R_1, R_2)$ computes h_0, q_1, where h_0 is the pairwise exclusive-or of R_1 and R_2,and q_1 is the first question V would ask on input x, with random bits h_0. H_0 outputs h_0, q_1 with enough additional padding bits to make these outputs of length n. If either P or V is honest, then h_0 will be distributed uniformly. Also, if V is honest, then P will get no information about h_0.

Similarly, in order to compute h_i, q_{i+1}, V first sends q_i to P, who then computes a_i. P and V then securely evaluate H_i, where P inputs a_i, and V inputs h_{i-1}. In order to compute ACCEPT(h_i), P and V securely evaluate ACCEPT, where P inputs nothing, and V inputs the output of circuit H_m.

This sequence of circuit evaluations mirrors the original interactive proof system. V will accept iff she would have accepted in the original protocol, given the provers answers to her queries. However, the protocol also gives V essentially the same knowledge as does the original protocol. The values of a_i are hidden from direct view by the secure circuit evaluation, but will almost certainly be recoverable from h_i.

3.2 Randomizing the values of the intermediate circuits.

To achieve zero-knowledge, it is imperative that we hide the values of $h_0, \ldots, h_m$ and $q_1, \ldots, q_m$ from the verifier. We accomplish this by a simple exclusive-or trick. The resulting protocol is not a proof system in the normal sense, but can be considered a proof system subject to P obeying certain constraints on his behavior. In particular, we will require equality between certain inputs from the prover to the various circuits.

Our technique is quite simple. First, we modify our circuits to allow for n-bit masking vectors. We define $H'_0(R_1, R_2, M)$ as the circuit that outputs $h'_0 = H_0(R_1, R_2) \oplus M$, where $\oplus$ denotes bitwise exclusive-or. Similarly, we define $H'_i(h, a, M_1, M_2, M_3)$ as the circuit that outputs $h'_i = h_i \oplus M_2$, and $q'_i = q_i \oplus M_3$, where $h_i, q_i = H_i(h \oplus M_1, a)$. Finally, we define ACCEPT$'(h, M)$ as the circuit that outputs ACCEPT$(h \oplus M)$. Here, the M_i's are input by P. Our new protocol now proceeds as follows.

Step 0: First, P' uniformly chooses n-bit masks, $M_0^H, \ldots, M_m^H$, $M_1^Q, \ldots, M_m^Q$, and R_1. Then V' randomly selects R_2. The two then securely evaluate $H'_0(R_1, R_2, M_0^H, M_1^Q)$, which outputs $h'_0 = h_0 \oplus M_0^H$ and $q'_1 = q_1 \oplus M_1^Q$ to V'.

Step i: V' sends q'_i to P', who then recovers $q_i = q'_i \oplus M_i^Q$, and computes his answer, a_i. P' and V' securely compute

$$H'_i(h'_{i-1}, a, M_{i-1}^H, M_i^H, M_i^Q),$$

which will output $h'_i = h_i \oplus M_i^H$ and $q'_i = q_i \oplus M_i^Q$ to V'.

Final Step: P' and V' compute $accept'(h'_m, M_m^H)$, and V' accepts iff the output is 1.

The protocol, as stated above, is not a proof, nor is it zero-knowledge against an active adversary. It will be a proof if P' could be guaranteed to be consistent in his input values for M_i^H. That is, if P' was guaranteed to always input the same string in all both circuit evaluations where M_i^H is supposed to be input, then the protocol would essentially mirror the original one. Similarly, the protocol would hide everything but the final accept/reject verdict if V' was guaranteed to always give as input to each circuit the output of the prescribed earlier circuit computation.

3.3 Constraining the prover and the verifier's behavior.

We now describe a simple technique to constrain the prover and the verifier's behavior as required. To explain our technique, we introduce the notion of *triple blobs*. These blobs will be used to enforce consistency for P' and V'. Essentially, we use the exclusive-or tricks that have been used in countless other papers.

3.3.1 Triple Blobs.

We now define triple blobs (blobs, for short), and give some elementary facts about them.

Definition 2 Let $b \in \{0,1\}$, and let

$$B = (b_{1,1}, b_{1,2}, b_{1,3}), \ldots, (b_{k,1}, b_{k,2}, b_{k,3}),$$

where $b_{i,j} \in \{0,1\}$. We say that B is a *triple blob*, with value b, and security parameter k, if for all $1 \leq i \leq k$, $b = b_{i,1} \oplus b_{i,2} \oplus b_{i,3}$. If there is an i and j such that,

$$b_{i,1} \oplus b_{i,2} \oplus b_{i,3} \neq b_{j,1} \oplus b_{j,2} \oplus b_{j,3},$$

we say that B is inconsistent.

For our application, the prover and the verifier need to experiment on a blob in order to verify equality. This is done by evaluating the function E, defined below.

Definition 3 Let $B = \{b_{i,j}\}$ be a triple blob with security parameter k. For $Y = y_1, \ldots, y_k \in \{1,2,3\}$, we define

$$E(B,Y) = b_{1,y_1}, \ldots, b_{k,y_k}.$$

We say that (B,Y,Z) is inconsistent if B is inconsistent or if $E(B,Y) \neq Z$.

Lemmas 1 and 2 follow from a simple probability argument.

Lemma 1 For $b \in \{0,1\}$, let B_b be chosen uniformly from triple blobs with value b. Then for all Y_1 and Y_2, $(E(B_0,Y_1),E(B_0,Y_2))$ and $(E(B_1,Y_1),E(B_1,Y_2))$ will have identical distributions. ∎

Lemma 2 Let B_0 and B_1 be any two triple blobs for 0 and 1 respectively. If Y is chosen at random, then

$$prob(E(B_0,Y) = E(B_1,Y)) \leq \left(\frac{2}{3}\right)^k. \quad ∎$$

Finally, it is useful to define the construction of a random blob that is consistent with a given Y and Z.

Definition 4 For $y \in \{1,2,3\}$ and $b,z \in \{0,1\}$, define distribution $c(b,y,z)$ to be uniform over all triples (b_1,b_2,b_3) such that,

1. $b = b_1 \oplus b_2 \oplus b_3$, and,
2. $b_y = z$.

For $Y = y_1,\ldots,y_k$ and $Z = z_1,\ldots,z_k$, we define distribution $C(b,Y,Z)$ by

$$C(b,Y,Z) = c(b,y_1,z_1),\ldots,c(b,y_k,z_k).$$

The following lemma follows virtually by definition.

Lemma 3 Let B be chosen from $C(b,Y,Z)$. Then B will be distributed uniformly over all triple blobs of value b, subject to $E(B,Y) = Z$. If in addition, Z is chosen uniformly, then B will be distributed uniformly over all triple blobs for b. ∎

Instead of working with single bits, we will typically work with strings of length n. For notational convenience, we will write $\vec{B}$ to denote an n vector of triple blobs. Similarly, we will use $\vec{Y}$ and $\vec{Z}$ to denote n vectors of their usual type. For $w \in \{0,1\}^n$, we will also write $C(w,\vec{Y},\vec{Z})$, and $E(\vec{B},\vec{Y})$, where all operations are done componentwise.

To give some intuition about what is going on, we can think of evaluating $E(\vec{B},\vec{Y})$ as obtaining a shadow of blob $\vec{B}$. If $\vec{B}_{w_0}$ is a blob for a string w_0 and $\vec{B}_{w_1}$ is a blob for a string $w_1 \neq w_0$, and $\vec{Y}$ is chosen at random, then with high probability, $E(\vec{B}_{w_0},\vec{Y}) \neq E(\vec{B}_{w_1},\vec{Y})$ (Lemma 2). This allows one to make sure that someone is using the same string in two different locations. However, if $\vec{B}$ is uniformly generated, with value w, then seeing $E(\vec{B},\vec{Y})$, for two different values of $\vec{Y}$ will still not give any information about w (Lemma 2).

3.3.2 Using triple blobs to enforce consistency.

We now augment our circuits in order to handle blobs. For ease of exposition, we will allow our circuits to flip coins. This can be simulated by a deterministic circuit that takes random bits from both parties and computes pairwise exclusive or's, much as in the design of H_0. We will denote objects which are in "blob" form by putting them in brackets. Thus, $[w]$ denotes a blob representation of w.

We define

$$H_0^*(R_1, R_2, [M_0^H], \vec{Y}_0^M, \vec{Y}_0^h, \vec{Z}_0^h, \vec{Y}_1^q, \vec{Z}_1^q, M_1^Q)$$

to be the circuit that,

1. Outputs "abort" if $[M_0^H]$ is inconsistent,
2. Computes $h_0', q_1' = H_0'(R_1, R_2, M_0^H, M_1^Q)$.
3. Outputs $[h_0'] \leftarrow C(h_0', \vec{Y}_0^h, \vec{Z}_0^h)$, $[q_1'] \leftarrow C(q_1', \vec{Y}_1^q, \vec{Z}_1^q)$, and $\vec{Z}_0^M = E([M_0^H], \vec{Y}_0^M)$ to V.

We define

$$H_i^*([h_{i-1}'], \vec{Y}_{i-1}^h, \vec{Z}_{i-1}^h, a, [M_{i-1}^H], \vec{Y}_{i-1}^M, \vec{Z}_{i-1}^M, [M_i^H], \vec{Y}_i^M, M_{i+1}^Q, \vec{Y}_{i+1}^q, \vec{Z}_{i+1}^q, \vec{Y}_i^h, \vec{Z}_i^h)$$

to be the circuit that,

1. Outputs "abort" if either $([h_{i-1}], \vec{Y}_{i-1}^h, \vec{Z}_{i-1}^h)$, $([M_{i-1}^H], \vec{Y}_{i-1}^M, \vec{Z}_{i-1}^M)$, or $[M_i^H]$ are inconsistent.
2. Computes $h_i', q_{i+1}' = H_i'(h_{i-1}, a, M_{i-1}^H, M_i^H, M_{i+1}^Q)$.
3. Outputs $[h_i'] \leftarrow C(h_i', \vec{Y}_i^h, \vec{Z}_i^h)$, $[q_{i+1}'] = C(q_{i+1}', \vec{Y}_{i+1}^q, \vec{Z}_{i+1}^q)$, and $\vec{Z}_i^M = E([M_i^H], \vec{Y}_i^M)$ to V.

Finally, we define

$$\text{ACCEPT}^*([h_m'], \vec{Y}_m^h, \vec{Z}_m^h, [M_m^H], \vec{Y}_m^M, \vec{Z}_m^M)$$

to be the circuit that,

1. Outputs "abort" if either $(h_m'], \vec{Y}_m^h, \vec{Z}_m^h,)$ or $([M_m^H], \vec{Y}_m^M, \vec{Z}_m^M)$ is inconsistent.
2. Outputs $\text{ACCEPT}'(h_m', M_m^H)$.

3.4 The final protocol.

Using the circuits defined above, we now specify our protocol (P^*, V^*).

Step 0: P^* uniformly selects masks, $M_0^H, \ldots, M_m^H$ and $M_1^Q, \ldots, M_m^Q$, and uniformly chooses blob representations, $[M_0^H], \ldots, [M_m^H]$. P^* uniformly chooses,

$$(\vec{Y}_0^h, \vec{Z}_0^h), \ldots, (\vec{Y}_m^h, \vec{Z}_m^h), (\vec{Y}_1^q, \vec{Z}_1^q), \ldots, (\vec{Y}_m^q, \vec{Z}_m^q),$$

and R_1. V^* uniformly chooses $\vec{Y}_0^M, \ldots, \vec{Y}_m^M$, and R_2.

P^* and V^* securely evaluate,

$$H_0^*(R_1, R_2, [M_0^H], \vec{Y}_0^M, \vec{Y}_0^h, \vec{Z}_0^h).$$

If this circuit, or any other circuit ever outputs "abort," then V^* immediately aborts the protocol and rejects. Otherwise, V^* recovers $[h_0']$, $[q_1']$, and $\vec{Z}_0^M$. Note that $h_0' = h_0 \oplus M_0^H$.

Step i: First, V^* sends $[q_i']$ to P^*, who aborts immediately if $([q_i'], \vec{Y}_i^q, \vec{Z}_i^q)$ is inconsistent. P^* and V^* securely evaluate,

$$H_i^*([h_{i-1}'], \vec{Y}_{i-1}^h, \vec{Z}_{i-1}^h, a, [M_{i-1}^H], \vec{Y}_{i-1}^M, \vec{Z}_{i-1}^M, [M_i^H], \vec{Y}_i^M, M_{i+1}^Q, \vec{Y}_{i+1}^q, \vec{Z}_{i+1}^q, \vec{Y}_i^h, \vec{Z}_i^h),$$

and V^* recovers $[h_i']$, $[q_{i+1}']$, and $\vec{Z}_i^M$. Note that $h_i' = h_i \oplus M_i^H$ and $q_{i+1}' = q_{i+1} \oplus M_{i+1}^Q$.

Final Step: P^* and V^* securely evaluate

$$\text{ACCEPT}^*([h_m'], \vec{Y}_m^h, \vec{Z}_m^h, [M_m^H], \vec{Y}_m^M, \vec{Z}_m^M),$$

and V^* accepts iff the circuit outputs a 1.

4 The protocol is a zero-knowledge proof system.

We now sketch the argument that our final protocol is a zero-knowledge proof system. First, we argue that it will remain a proof system. That is, if (P, V) accepts with high probability on input x, then so will (P^*, V^*), and similarly, if (P, V) rejects x with high probability, then so will (P^*, V^*).

4.1 $(\mathbf{P^*}, \mathbf{V^*})$ is a proof system.

First, we note that if (P, V) accept x with probability at least $p(x)$, then so will (P^*, V^*). If P^* abides by the consistency constraints, then the distribution on $h_0, \ldots, h_m$, $q_1, \ldots, q_m$, and $a_0, \ldots, a_l$ will be identical to that produced by (P, V). By inspection of the protocol, we see that V^* will accept, given a setting of the above variables if V would have as well. The point is that if P^* behaves properly, then (P^*, V^*) will essentially run the original protocol, merely coded up in a strange way.

We now wish to show that for any malicious prover, $\hat{P}$, the probability that $(\hat{P}, V^*)$ accepts x is nonnegligible, then there must be a prover P' such that (P', V) accepts x with nonnegligible probability. P' essentially runs $\hat{P}$ on input x, encodes V's queries into the same form as V^* would use, and then decodes P^*'s answers into a acceptable to V.

Step 0: P' starts running P^*, continuing the run through the point where P^* makes his inputs to circuit H_0^*. In particular, P' learns the values of M_i^Q, $\vec{Y}_i^q$ and $\vec{Z}_i^q$.

Step i: V makes query q_i to P'. P' computes,

$$[q_i'] \leftarrow C(q_i \oplus M_i^Q, \vec{Y}_i^q, \vec{Z}_i^q),$$

and sends $[q_i']$ to P^*. P' continues the run of P^*, through the point where P^* makes his inputs to circuit H_i^*. In particular, P' learns the values of a_i, which he sends to V, and also the values of M_{i+1}^Q, $\vec{Y}_{i+1}^q$ and $\vec{Z}_{i+1}^q$.

Aside from his choice of answers, P^*'s can cheat in two general ways. First, he can violate a consistency constraint, i.e. use a different value for M_i^H in two different places in the protocol. however, if he ever attempts to do this, then by Lemma 2, V^* will immediately reject with probability $1 - \left(\frac{2}{3}\right)^{|x|}$. If V^* doesn't reject though, we can make no guarantees about his chance of finally accepting x. Second, he can give inconsistent values for some experiment, (Y, Z), or give slightly inconsistent blob representations of M_i^H in two different places. This latter form of cheating will only increase the probability that the circuit will abort, without changing its output if it doesn't abort, so we can assume without loss of of generality that this form of cheating doesn't occur.

By a routine probabilistic analysis, we see that, subject to P^* not violating any of his consistency constraints (CC's), having P' supply $[q_i']$ will be essentially indistinguishable from actually running the protocol and having V^*'s send $[q_i']$ to P^*. From this fact, we obtain

$$prob((P', V) \text{ accepts}) \geq prob((\hat{P}, V^*) \text{ accepts } \& \ P^* \text{ didn't violate his CC's}).$$

However, using the security of the blob consistency checks, we have,

$$prob((\hat{P}, V^*) \text{ accepts } \& \ P^* \text{ violates a CC}) \leq \left(\frac{2}{3}\right)^k.$$

Hence, we obtain,

$$prob((\hat{P}, V^*) \text{ accepts}) - prob((P', V) \text{ accepts}) \ \leq \ \left(\frac{2}{3}\right)^k,$$

which implies our claim.

4.2 (P^*, V^*) is a zero-knowledge protocol.

We now produce a simulator S for (P^*, V^*). Given a possibly malicious $\hat{V}$, S must take $\hat{V}$'s queries, $[q'_0], \ldots, [q'_m]$, and $\hat{V}$'s inputs to the various circuits, and simulate the circuits' outputs. Furthermore, S must also correctly simulate situations in which P^* aborts the protocol. S works as follows.

Step 0: S uniformly chooses

- $[M_0^H], \ldots, [M_m^H]$,
- $[h'_0], \ldots [h'_m]$ and $\vec{Y}_0^h, \ldots, \vec{Y}_m^h$,
- $[q'_1], \ldots, [q'_m]$ and $\vec{Y}_0^q, \ldots, \vec{Y}_m^q$,

For $0 \leq i \leq m$, S computes $\vec{Z}_i^h = E([h'_i], \vec{Y}_i^h)$, and for $1 \leq i \leq m$, S computes $\vec{Z}_i^q = E([q'_i], \vec{Y}_i^q)$.

S simulates the secure evaluation of H_0^* as follows. S takes R_2 and $\vec{Y}_0^M$ as input from $\hat{V}$. S gives $\hat{V}$ the values of $[h'_0]$, $[q'_1]$ and $E([M_0^H], \vec{Y}_0^M)$ as output.

Step i: $\hat{V}$ will then send a string $[q_i^{\hat{V}}]$ to S. S simulates P^*, aborting the protocol if $[q_i^{\hat{V}}]$ is not a valid blob, or if $([q_i^{\hat{V}}], \vec{Y}_i^q, \vec{Z}_i^q)$ is inconsistent. S simulates the secure evaluation of H_i^* as follows. S takes $[h'_{i-1}]$, $\vec{Y}_{i-1}^M$, $\vec{Z}_{i-1}^M$, and $\vec{Y}_i^M$ as input from $\hat{V}$. S now executes the following procedure:

1. Output "abort" if $([h'_{i-1}], \vec{Y}_{i-1}^h, \vec{Z}_{i-1}^h)$ or $([M_{i-1}^H], \vec{Y}_{i-1}^M, \vec{Z}_{i-1}^M)$ is inconsistent.
2. Otherwise, output $[h'_i]$, $[q'_i]$, and $E([M_i^H], \vec{Y}_i^M)$.

Final Step: S simulates the running of ACCEPT* as follows. S accepts $[h_m]$, $\vec{Y}_m^M$, and $\vec{Z}_m^M$ from $\hat{V}$ as input. S then follows the following procedure.

1. Output "abort" if $([h'_m], \vec{Y}_m^h, \vec{Z}_m^h)$ or $([M_m^H], \vec{Y}_m^M, \vec{Z}_m^M)$ is inconsistent.
2. Otherwise, output 1 (accept).

Now, we claim that the distribution on $\hat{V}$'s view that is generated by running S with $\hat{V}$ will be statistically indistinguishable from the view generated by running $(P^*, \hat{V})$. First, observe that up to the point where ACCEPT* gives a 0 or a 1 as output, the two distributions are identical. In both cases, the distribution on $[h'_0], \ldots, [h'_m]$ and $[q'_1], \ldots, [q'_m]$ will be identical. This follows from the uniform choice of the mask vectors, and in fact holds regardless of $\hat{V}$'s choice of inputs to the circuits. Furthermore, Lemma 1 implies that for any Y_1 and Y_2, and any blob $[B]$ chosen by the simulator, the distribution on the values of $(E([B], Y_1), E([B], Y_2))$ will be independent of the actual values of B. For this reason, all

of the consistency checks and the $\vec{Z}$ vectors sent to $\hat{V}$ will be distributed identically with the the checks and $\vec{Z}$ vectors output by the actual protocol.

Thus, one can show that until ACCEPT* outputs a 0 or a 1, the simulation and the view from the actual protocol are identical. The only difference in the behavior of the simulation is that the output of ACCEPT* is, when not "abort," always equal to 1. In the actual protocol, ACCEPT* may sometimes output a 0. However, we can show that this event will occur with probability exponentially small in $|x|$.

As with a malicious prover, we can classify a malicious verifier's cheating to be in two forms. We say that an input $[q_i^{\hat{V}}]$ $\{[h_i^{\hat{V}}]\}$ to a circuit or to P^* is *legitimate* if it is a well formed blob and its contents are equal to q_i' $\{h_i'\}$. Otherwise, we say that the input is illegitimate.

Now, if $\hat{V}$ never gives an illegitimate input, then the distribution on $h_i = h_i' \oplus M_i^H$ will be identical to that produced by (P, V), and thus ACCEPT* will be equal to 0 with probability at most $2^{-|x|}$. On the other hand, if $\hat{V}$ ever gives an illegitimate input, then with probability $1 - (2/3)^{|x|}$, (by Lemma 2), the circuit will output "abort," or the prover will abort. From then on, $\hat{V}$'s next such input will be illegitimate with probability $1 - (2/3)^{|x|}$. By a simple probability analysis, ACCEPT* will output "abort" with probability at least $1 - m(2/3)^{|x|}$.

References

[1] D. Chaum, I. Damgard, and J. van de Graaf. Multiparty Computations Ensuring Secrecy of Each Party's Input and Correctness of the Output, Proc. CRYPTO85, Springer-Verlag, 1986, 477–488.

[2] O. Goldreich, Y. Mansour, and M. Sipser. Interactive Proof Systems: Provers that never fail and random selection. Proc. FOCS87.

[3] S. Goldwasser and M. Sipser. On public versus private coins in interactive proof systems. Proc. STOC86.

[4] R. Impagliazzo, L. Levin, and M. Luby. Pseudo-random Generation from One-Way Functions, Proc. STOC89

[5] R. Impagliazzo and M. Yung. Direct Minimum-Knowledge Computation, Proc. of CRYPTO87.

[6] J. Kilian. Founding Cryptography on Oblivious Transfer, Proc. of STOC88. Personal Communication.

[7] M. Naor Pseudorandomnesss and Bit Commitment, Proc. of Crypto89.